iotc
ctoi
Indian Ocean Tuna Commission
Commission des Thons de l'Océan Indien

IOTC-2006-WPTT-INF07

IOTC Technical Report 06/02

Converting length into age

IOTC Secretariat

IOTC – Seychelles, June 2006

Objective

The objective of this report is to document the statistical procedure applied to convert SF data reported in weight into length measurements in 2003 and later. This procedure has been slightly modified in 2006.

The Procedure

The method can be thought of a parametric equivalent of an age-length key. In the traditional view of age-length keys, direct age readings from a particular period are used to produce a mapping of lengths to ages. The goal is to capture, in contrast with the method of cohort slicing, the individual variability in ages that is present in fish of a given age. The disadvantage of the traditional age-length key is that is requires large numbers of individual age-length readings in order to have sensible sample sizes and robust estimation of the correct transition probabilities.

Setting up a program of direct age reading with large target sample sizes is not usually feasible as a routine practice in large pelagics like tunas. Therefore, a frequently used alternative has been to apply a cohort slicing, basically a one-to-one mapping that is based on a growth-curve and some assumption about the progression of the variance-at-age. However, the slicing has the undesirable feature of assigning all the fish of a given size to a given age, clearly not a realistic assumption. The objective of this report was to propose a method that would represent one step up from the cohort slicing, by combining the information contained in a set of direct age readings with certain assumptions about the way the mean and variance of ages progresses with length.

In other words, the basic idea is that the conditional distribution function can be summarised in a parametric form by assuming that the distribution of ages by length can be reasonably well approximated by a distribution indexed by a mean and a variance function (which could be length dependent).

For example,

If $P(a \mid l) \sim N(\mu, \sigma^2)$ we could use plugin estimates:

$$\hat{\mu} = t_0 - \frac{1}{k}\left(\log\left(\frac{l}{L_\infty} - 1\right)\right) \qquad (1)$$

$$\hat{\sigma}^2 = g(l) \qquad (2)$$

In other words, the mean ages-at-length are assumed to follow an inverse von Bertalanffy curve, while the variance of age-at-length has been assumed to follow some function of length.

The final step is to integrate the conditional distribution of ages over a particular length interval, centered around the target age, so that the probability that a fish of length l_a is of age $a_k \in [a_a, a_b]$, is

$$P(a_k \in [a_a, a_b] \mid l = l_a) = \Phi_{l_a}(a_b) - \Phi_{l_a}(a_a)$$

Where Φ_{l_a} is the appropriate cumulative distribution function that corresponds to the length l_a.

The implementation for bigeye tuna

The parameters of the IVB are fitted using the raw data of age and lengths that B. Stequert used to estimate the growth curve of bigeye in his paper. The fitting procedure was a simple least-squares method, as implemented in function `nls`.

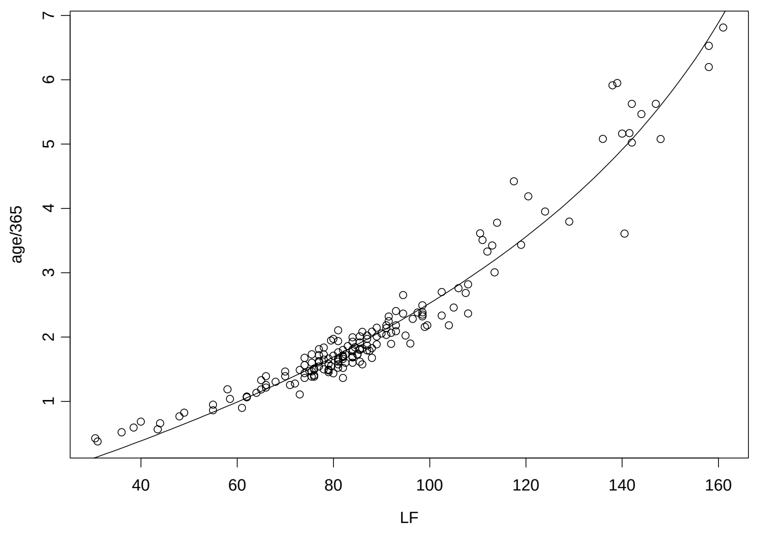
This is the original data set used by Stequert, consisting of readings of age and lengths. The original age readings in days were converted in years.

```
> bet.vb2
Parameters:
$Linf:
[1] 160

$k:
[1] 0.0007

$t0:
[1] -100


Variables:
      LF        age
 1  77.0 1.8109589
 2  86.0 2.0785388
 3  79.5 1.9442922
 4  78.0 1.8365297
 5  86.0 1.8228311
 6  79.5 1.5479452
 7  88.0 2.0831050
 8  77.0 1.7178082
 9  80.0 1.9689498
. . .
```

…for the call to the non-linear fitting procedure…

```
> fit.inv.vb.bet <- nls(age ~ t0 + (log(1. - LF/Linf)/ - k), data=bet.vb2,
start=list(Linf=Linf, t0=t0, k=k), trace=T)
145462000 : 169.068 -123.752 0.000877866
44865200 : 169.1 -0.506139 0.00175287
11143100 : 169.163 -0.506966 0.00349432
2749440 : 169.289 -0.508598 0.00694317
669444 : 169.542 -0.511802 0.0137063
158656 : 170.051 -0.518015 0.026707
35560.9 : 171.074 -0.52969 0.0507051
7080.57 : 173.126 -0.550293 0.091445
1086.32 : 177.136 -0.582203 0.149355
97.8645 : 184.12 -0.619158 0.204929
14.0026 : 192.378 -0.637822 0.225853
```

Figure 1. Age as a function of length fitted using equation (1)



```
12.5016 : 195.432 -0.624683 0.227097
12.4992 : 195.032 -0.615188 0.228782
12.4992 : 195.125 -0.616521 0.228513
12.4992 : 195.104 -0.616212 0.228578
> summary(fit.inv.vb.bet)

Formula: age ~ t0 + (log(1. - LF/Linf)/ - k)

Parameters:
                Value Std. Error   t value
Linf.Linf  195.104000  6.2412100  31.26060
    t0.t0   -0.616212  0.1045720  -5.89272
      k.k    0.228578  0.0190575  11.99410

Residual standard error: 0.27863 on 161 degrees of freedom

Correlation of Parameter Estimates:
      Linf.Linf  t0.t0
t0.t0 -0.856
  k.k -0.977     0.938
>
```

This gives us a model that predicts the mean age-at-length. In order to complete the specification of the age-length key, we need some expression for the variance of ages at length that describes how the variance of ages is assumed to change as a function of length.

Usually, we can incorporate two sources of uncertainty: uncertainty about the mean predicted age, and the residual variation around the predicted value. That is

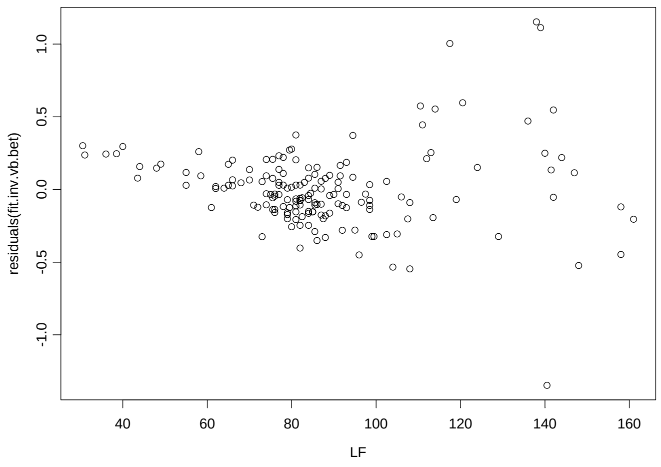
$$\mathrm{var}(\hat{a}) = \mathrm{var}(E(a\,|\,l)) + E(\mathrm{var}(a\,|\,l))$$

This would implemented as :

```
>     prediction <- predict(fit.inv.vb.bet, list(LF=seq(10,200)), se.fit = T)
>     var.mean <- (prediction$se.fit)^2
>     var.sampl <- (prediction$residual.scale)^2
>     sd <- sqrt(var.mean + var.sampl)
```

The problem is that the residual variance seems to be length-dependent, as we can see in Figure 2

Figure 2. Residuals from the fit of age vs length, as a function of length

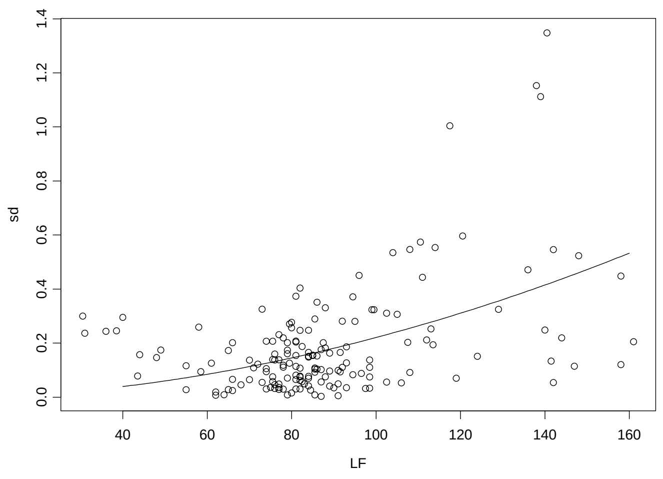


Therefore, we need some further modelling. After some failed attempts at modeling this effect in glm[1], we followed a simpler approach to accommodate that dependency. To produce a smoother estimate by length, we fitted a power function of the absolute residuals as a function of length (see Figure 3).

```
> sd_sqrt(residuals(fit.inv.vb.bet)^2)
> nls(sd~a*LF^b, start=list(a=0.0003, b=3.))
Residual sum of squares : 5.291746
parameters:
             a        b
 0.00003948042 1.873905
formula: sd ~ a * LF^b
164 observations
> summary(fit.sd)

Formula: sd ~ a * LF^b

Parameters:
         Value    Std. Error   t value
a 0.0000394804 0.0000477523 0.826775
b 1.8739100000 0.2563310000 7.310500

Residual standard error: 0.180735 on 162 degrees of freedom

Correlation of Parameter Estimates:
       a
b -0.998
> plot(LF,sd)
> lines(40:160, predict(fit.sd, list(LF=40:160)))
```

[1] The prediction were very unstable using a Gamma or poisson family assumption, probably because of the small sample sizes. There is room for further work there, including modelling directly the size dependency using the variance function options in the generalized least squares function `gnls`.

Figure 3. Absolute value of the residuals as a function of length, with line showing the fitted model.



The next step is to build a data frame that would allow us to make predictions for a large range of sizes, using the results of the models fitted so far.

```
> prediction <- predict(fit.inv.vb.bet, list(LF=seq(10,195)), se.fit = T)
> pred.sd_predict(fit.sd, list=LF=c(10:195))
> pred.caa.df_data.frame(LF=c(10:195), pred.age=prediction$fit,
pred.sd=sqrt(prediction$se.fit^2 + pred.sd^2))
```

Note that we cannot exceed the value of L_∞ (about 195cm in the bigeye example) in the range of sizes to predict, as we start getting logs of negatives in equation 1. As expected, the quality prediction gets poorer as we get away from the range of sizes analyzed in the direct age readings sample and closer to the critical value of L_∞, as can be seen in the standard error of the prediction shown in the Figure 4.

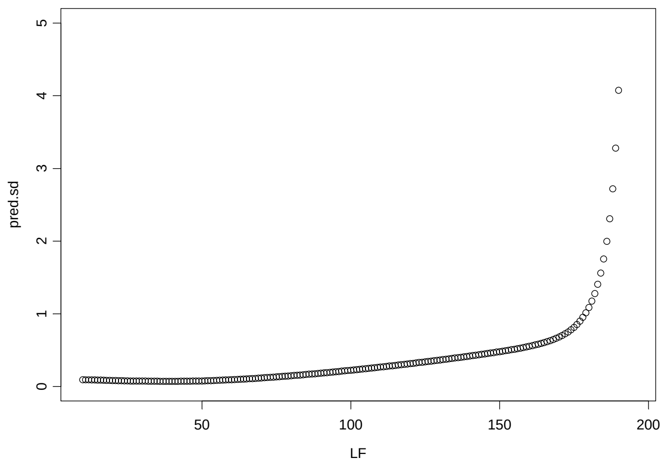
The data frame `pred.caa.df` contains the information, for each size class, that will be assumed to determine the conditional distribution of ages. The next step is to build the age-length key, by iterating over a large range of sizes and integrating, in each class, the probability over the different age classes. The function `get.alk` (listed at the end of this report) implements those calculations. Given that the mean ages are estimated, a Student's t distribution was assumed as a distributional form instead of a normal distribution. The number of degrees of freedom was assumed to be $n-p$, in this case, 161.

```
> alk.bet <- get.alk(pred.caa.df, degfr=161)
```

We start the calculations by allowing ages to be negative as some negative ages could be predicted by the symmetry of the distribution of predicted ages-at-length. At the end of the calculations we simply place all the negative ages in the first age category. The actual calculation for the separation into age classes takes place in an auxiliary function (`predict.ages)` also listed below.

```
> alk.bet[50:60, 1:10]
           0.5       1.5           2.5 3.5 4.5 5.5 6.5 7.5 8.5 9.5
59 0.689686320 0.3103137 0.000000e+000   0   0   0   0   0   0   0
60 0.539827413 0.4601726 0.000000e+000   0   0   0   0   0   0   0
61 0.392054834 0.6079452 0.000000e+000   0   0   0   0   0   0   0
62 0.265034297 0.7349657 0.000000e+000   0   0   0   0   0   0   0
63 0.167825680 0.8321743 0.000000e+000   0   0   0   0   0   0   0
64 0.100335851 0.8996641 1.110223e-016   0   0   0   0   0   0   0
65 0.057118559 0.9428814 4.218847e-015   0   0   0   0   0   0   0
66 0.031221606 0.9687784 1.252332e-013   0   0   0   0   0   0   0
67 0.016516117 0.9834839 2.881473e-012   0   0   0   0   0   0   0
```

Figure 4. Standard errors of the prediction as a function of length for the age vs length model.



```
68 0.008516314 0.9914837 5.224710e-011   0   0   0   0   0   0   0
69 0.004307910 0.9956921 7.527312e-010   0   0   0   0   0   0   0
```

The structure of `alk.bet` is a matrix with named dimensions for the size classes and the mid-points of the age classes. This matrix can now be used to get conversion from a matrix of size frequencies into a matrix of age frequencies, using the function `get.caa`, which takes care of other details concerning the plus group as well. Also a small percentage of fish might be identified as being larger than the maximum size that we can safely predict ages from. Those fish would be arbitrarily be assigned to the last age category available.

```
> caa.ll_get.caa(llsz,alk.bet)
 A total of  7759.4  fish are bigger than the max class : 195 =  0.0136270684938639
percent.
> sum(caa.ll)
[1] 56941035
> sum(llsz)
[1] 56941080
```

Rounding errors determine that some fish are missing (45 fish in almost 57 million!). The output is a matrix where only the columns are named, so that they can be merged easily with the sample header information that presumably lies in a separate matrix. The last size category is labelled `“plus”` which could be inconvenient at the time of making some plot or calculations. An example of the final output:

```
> caa.ll [1:10,]
               0.5         1.5       2.5       3.5       4.5       5.5       6.5        7.5        8.5
 [1,] 3.569142e-006   67.84201  1762.720  2044.909  1673.590 1047.1586  465.8502   70.96448   2.549203
 [2,] 1.348787e-003   73.45819  1104.047  1516.133  1124.588  555.1124  187.3065   29.23466   3.624474
 [3,] 1.749130e-002  377.95328  1428.126  2809.829  2681.818 1009.4576  233.9762   41.28364   4.397278
 [4,] 2.799149e-005  114.30005  1098.799  1892.703  1684.981  736.5933  179.4780   24.26419   2.594834
 [5,] 4.249902e+000  246.58912  5010.892  5583.432  5994.589 3712.6588 1815.1876  316.12485  13.193904
 [6,] 2.774350e-001 1215.70932  4716.441  6938.059  5549.136 2867.6720 1029.2320  465.61863 149.133127
 [7,] 2.194442e-001 1960.09618 10717.276 14406.448 11120.716 6219.2457 2992.2378 1278.90186 453.957445
 [8,] 1.866416e-001 1644.34465 12424.262 16947.542 12805.044 5715.9821 1509.6410  285.67044  50.848127
 [9,] 1.318627e+002 1128.68033 12058.899 14397.616 13623.106 8156.8834 3680.4698  729.05307  71.808407
[10,] 5.230456e-001 1428.36564 10591.849 18589.139 16588.071 8114.6291 2774.1137 1048.69030 314.668293

              9.5          10.5          11.5          12.5          13.5          14.5         15.5        16.5
 [1,]   0.01738723 2.131967e-005 0.00000000000 0.00000000000 0.000000e+000 0.00000000000 0.0000000000 0.000000000
 [2,]   0.19199412 1.946429e-003 0.00000404521 0.00000000000 0.000000e+000 0.00000000000 0.0000000000 0.000000000
 [3,]   0.51030248 1.138214e-001 0.01735249822 0.00056004514 3.137733e-006 0.00000000000 0.0000000000 0.000000000
 [4,]   0.27503942 1.216562e-002 0.00010432328 0.00000000000 0.000000e+000 0.00000000000 0.0000000000 0.000000000
 [5,]   0.45490486 2.705547e-002 0.00054021939 0.00000186706 0.000000e+000 0.00000000000 0.0000000000 0.000000000
 [6,]  25.17378320 4.873967e+000 1.43713409880 0.40699374519 1.105783e-001 0.01841844016 0.0009431613 0.000010963
 [7,] 106.65535743 1.736983e+001 2.18251865269 0.18599268416 7.009544e-003 0.00006800165 0.0000000000 0.000000000
 [8,]   8.35082120 2.179017e+000 0.40527658612 0.04165379735 1.983432e-003 0.00002266722 0.0000000000 0.000000000
 [9,]  11.52818140 3.374333e+000 0.81638398098 0.09770484616 4.315533e-003 0.00004533443 0.0000000000 0.000000000
[10,]  78.08837156 2.583791e+001 8.47194985067 2.29265653535 6.090393e-001 0.19345072058 0.0529802479 0.004643592

              17.5 plus
 [1,] 0.00000000000    0
 [2,] 0.00000000000    0
 [3,] 0.00000000000    0
 [4,] 0.00000000000    0
 [5,] 0.00000000000    0
 [6,] 0.00000000000    0
 [7,] 0.00000000000    0
 [8,] 0.00000000000    0
 [9,] 0.00000000000    0
[10,] 0.00009153754    0
>
```

```
predict.ages <-
function(age, sd, df, freq = rep(1., length(age)), int = 1., nint = 20.)
{
#---------------------------------------------------------------------
# This function takes age values and expand them into a distribution of predicted values
#  according to the size of the interval for a category (int) and the number of
#  bins to predict (nint).
# The number of observations in each category is given by freq.
#
# This is the low limit of the predicted lt class.
#
#
# Two cases : either val is a scalar or a vector.
#
       age.low <- int * trunc(age/int)
       n <- length(age.low)
       if(n == 1.) {
              ltt <- seq((age.low - int * (nint/2.)), by = int, length = nint)
              pp <- diff(pt((ltt - age)/sd, df))
              pp <- pp * freq
              names(pp) <- ltt[-1.] - int/2.
       }
       else if(n > 1.) {
              prdmtx <- clssmtx <- matrix(0., nrow = n, ncol = nint - 1.)
              for(i in 1.:n) {
                     tgt <- seq((age.low[i] - int * (nint/2.)), by = int, length = nint)
                     prdmtx[i,  ] <- freq[i] * diff(pnorm(tgt, age[i], sd))
                     ss <- sum(prdmtx[i,  ])/freq[i]
                     if(ss < 0.95)
                            warning(" Coverage < 0.95. Increase number of intervals.")
                     clssmtx[i,  ] <- tgt[-1.] - int/2.
              }
              pp <- tapply(prdmtx, clssmtx, sum)
       }
       pp
}
```

```
> get.alk
function(df, degfr, lowest.age = -5.5, nages = 40, nint = 10)
{
      # Produces the age-length key, a matrix of projections from lengths to ages.
      # df is a data frame with at least two columns named pred.age, and pred.sd,
      # that will be used in predict.ages to estimate a distribution by ages.
      # degfr is the number of degrees of freedom implicit in the estimation of
pred.sd.
      # Not very sensitive for values > 50.
      #
      attach(df)
      matx <- matrix(0., nrow = nrow(df), ncol = nages)
      a <- seq(lowest.age, length = nages, by = 1.)
      for(i in 1.:nrow(df)) {
            l <- predict.ages(pred.age[i], pred.sd[i], degfr, nint = nint)
            #
            # Tricky code alert: id has the 'coordinates' of the matrix to fill with
the value
s of l.
            #
            id <- cbind(rep(i, length(l)), match(names(l), a))
            matx[id] <- l
            cat(i, "lf: ", LF[i], "  ::", sum(l), " a: ", pred.age[i], " sd: ",
pred.sd[
                  i], "\n")
      }
      dimnames(matx) <- list(LF, as.character(a))
      detach("df")
      #
      # For simplicity, the ages predicted as negative are accumulated in the age-0
class.
      #
      st <- (dimnames(matx)[[2]] < 0)
      sm.szs <- rowSums(matx[, st])
      matx <- matx[, !st]
      matx[, 1] <- matx[, 1] + sm.szs
      invisible(matx)
}
```

```
> get.caa
function(sz, alk, plus.group = 18)
{
	#
	# Gets a SF distribution and distributes in ages according to the age-length key
in alk.
	#
	sizes <- as.matrix(as.numeric(dimnames(alk)[[1]]))
	szs <- as.numeric(dimnames(sz)[[2.]])
	mx.sz <- max(sizes)
	st <- (szs > mx.sz)
	cat(" A total of ", sum(sz[, st]), " fish are bigger than the max class :",
mx.sz)
	cat(" = ", (100 * sum(sz[, st]))/sum(sz), " percent. \n")
	#
	# Accumulate the biggest fish to put later in the plus group.
	#
	big.f <- rowSums(sz[, st])
	#
	# Exclude the large classes from the rest of the analysis.
	#
	sz <- sz[, !st]
	c <- matrix(0., nrow = nrow(sz), ncol = ncol(alk))
	#
	#
	#
	for(i in 1.:ncol(sz)) {
		sz.class <- szs[i]
		id <- match(sz.class, sizes)
		if(is.na(id))
			cat("NA in ", sz.class, "\n")
		else {
			a <- as.matrix(alk[id,  ])
			c <- c + outer(sz[, i], as.numeric(a))
		}
	}
	#
	# Put the big fish in the last class and accumulate the plus group.
	#
	c[, ncol(c)] <- c[, ncol(c)] + big.f
	dimnames(c) <- list(NULL, dimnames(alk)[[2]])
	st <- (as.numeric(dimnames(alk)[[2]]) >= plus.group)
	plus <- rowSums(c[, st])
	c <- cbind(c[, !st], plus)
	c
}
```