Confidence Intervals for Non-linear Least Squares Models (using VB growth models as an example)

John McKinlay

11 November 2014

## Overview

This document demonstrates obtaining confidence intervals and prediction intervals for fitted nls objects. It is framed in the context of von Bertalanffy growth models, although the concepts are applicable to nls generally.

I'll be using some helper functions and data from the FSA and FSAdata packages ([here](http://fishr.wordpress.com/fsa)). Packages nlstools ([here](http://cran.at.r-project.org/web/packages/nlstools/index.html)) and propagate ([here](http://cran.r-project.org/web/packages/propagate/index.html)) are used to show alternate approaches for obtaining confidence intervals (CI) and prediction intervals (PI) for fitted nls functions. Finally, the Bayesian bootstrap is presented as an alternative to the classical approach.

## Age-Length data

Croaker2 is a dataset contained in package FSAdata. A plot of the length-at-age data for male fish is as follows.

data(Croaker2)  
crm <- Subset(Croaker2,sex=="M")  
dim(crm)

## [1] 114 3

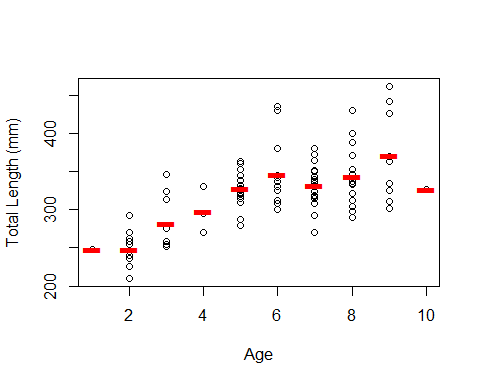
head(crm)

## age tl sex  
## 3 1 248 M  
## 15 2 210 M  
## 16 2 225 M  
## 17 2 236 M  
## 18 2 240 M  
## 19 2 245 M

table(crm$age)

##   
## 1 2 3 4 5 6 7 8 9 10   
## 1 17 9 3 20 14 26 14 9 1

plot(tl~age, data=crm, xlab="Age",ylab="Total Length (mm)")  
points(unique(crm$age), with(crm, tapply(tl, age, mean)), pch="-", cex=4, col=2)



## von Bertalanffy growth models

There are a large variety of these types of models, and the FSA package is an excellent starting point for exploring them. Many models are different parameterisations that serve to decrease the correlation of estimands, or to put more or less focus on particular aspects of growth. The Beverton and Holt (1957) forumation is an oldie but a goodie, at least for pedagogical purposes:

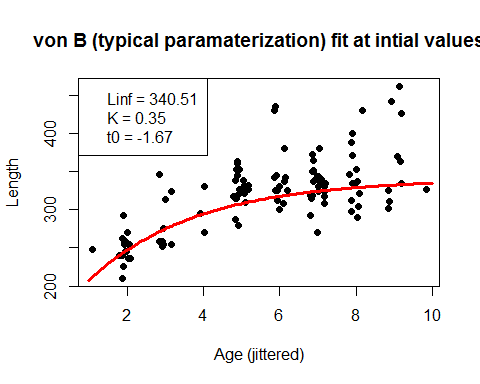
where is the expected average length at time *t*, is the asymptotic average length (assuming mortality is low enough for fish to reach this length), and is the theoretical time when length was zero. [[1]](#footnote-28)

Non-linear least squares (e.g. function nls) is typically used to determine estimates of the parameters. In order to converge, nls models often require reasonable starting values for parameters. In the case of the VB growth equation, function FSA::vbStarts provides methods for obtaining starting values based on a Ford-Walford approach that regresses against .

In the code that follows we:

1. Obtains starting values
2. Fit the VB curve
3. Extract coefficients
4. Plot the fit
5. Summarise the model

svTypical <- vbStarts(tl~age, data=crm, plot=TRUE)



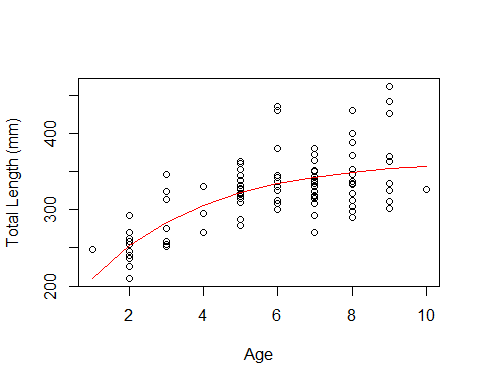
unlist(svTypical)

## Linf K t0   
## 340.5069 0.3517 -1.6732

vbTypical <- tl~Linf\*(1-exp(-K\*(age-t0))) # make formula  
fitTypical <- nls(vbTypical, data=crm, start=svTypical)  
(p <- coef(fitTypical))

## Linf K t0   
## 366.4141 0.3148 -1.7142

plot(tl~age, data=crm, xlab="Age",ylab="Total Length (mm)")   
# add the fit  
with(crm, lines(age, p["Linf"]\*(1-exp(-p["K"]\*(age-p["t0"]))), col=2))



summary(fitTypical)

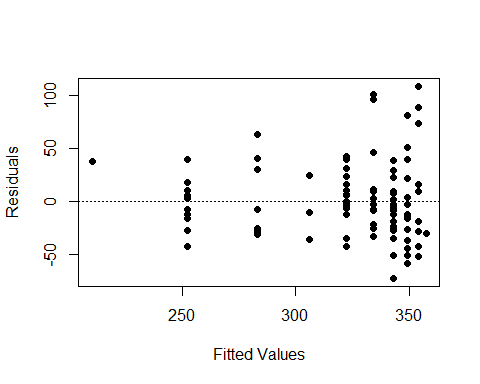
##   
## Formula: tl ~ Linf \* (1 - exp(-K \* (age - t0)))  
##   
## Parameters:  
## Estimate Std. Error t value Pr(>|t|)   
## Linf 366.414 16.754 21.87 <2e-16 \*\*\*  
## K 0.315 0.108 2.92 0.0042 \*\*   
## t0 -1.714 1.049 -1.63 0.1049   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 33.4 on 111 degrees of freedom  
##   
## Number of iterations to convergence: 4   
## Achieved convergence tolerance: 3.84e-06

# nlstools::overview, adds CI's and correlation for params  
overview(fitTypical)

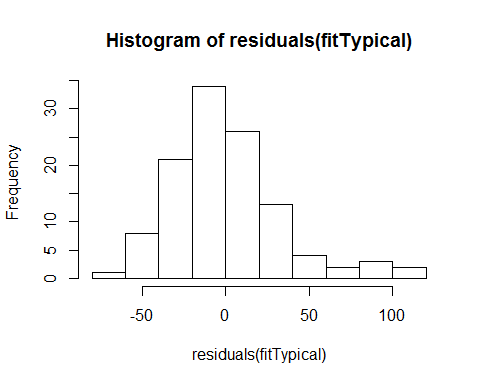
##   
## ------  
## Formula: tl ~ Linf \* (1 - exp(-K \* (age - t0)))  
##   
## Parameters:  
## Estimate Std. Error t value Pr(>|t|)   
## Linf 366.414 16.754 21.87 <2e-16 \*\*\*  
## K 0.315 0.108 2.92 0.0042 \*\*   
## t0 -1.714 1.049 -1.63 0.1049   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 33.4 on 111 degrees of freedom  
##   
## Number of iterations to convergence: 4   
## Achieved convergence tolerance: 3.84e-06  
##   
## ------  
## Residual sum of squares: 124000   
##   
## ------  
## t-based confidence interval:  
## 2.5% 97.5%  
## Linf 333.2146 399.6135  
## K 0.1015 0.5281  
## t0 -3.7921 0.3637  
##   
## ------  
## Correlation matrix:  
## Linf K t0  
## Linf 1.0000 -0.9506 -0.8706  
## K -0.9506 1.0000 0.9718  
## t0 -0.8706 0.9718 1.0000

It is also usual to check the residuals - they should be homoscedastic and approximately normal. In this case they are not, perhaps unsurprising given that we might expect growth to be more variable for large animals than small ones. A multiplicative error structure is likely to be more appropriate, but we'll ignore this and press on.

residPlot(fitTypical)



hist(residuals(fitTypical))



## Confidence intervals on fitted functions

SE's of estimated parameters produced by summary.nls, and thus confidence intervals produced by nlstools::overview, are based on a Normal approximation that is often not supported by the data. Several alternatives are available that are not so restrictive in their assumptions, including:

1. Standard bootstrapping
2. Bayesian bootstrapping
3. First- & second-order Taylor series approximation

### Standard bootstrapping

This approach bootstraps the (mean centred) residuals to generate multiple datasets, from which multiple model fits are obtained. Percentile intervals can be used to represent uncertainty about parameters, and percentiles of predicted values from the many model fits can be used to obtain CI's. PI's can be obtained by inflating predicted values from each bootstrap fit by the residual SE for that fit, then taking appropriate percentile intervals. Function nlsBoot from package nlstools automates this to some extent, and provided associated summary and plot methods.

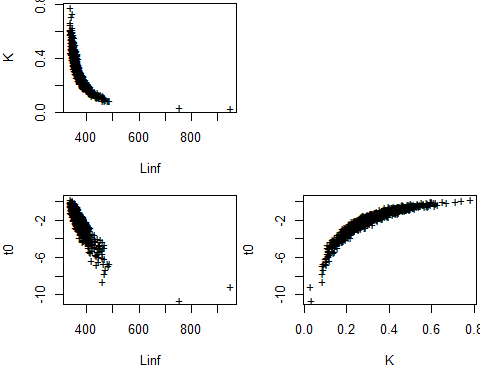
bootTypical <- nlsBoot(fitTypical, niter=1000)

## Warning: The fit did not converge 1 times during bootstrapping

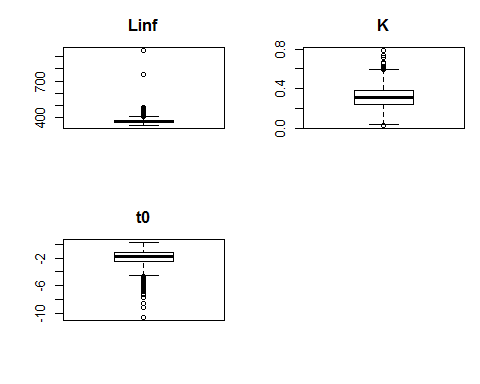
str(bootTypical)

## List of 4  
## $ coefboot: num [1:999, 1:3] 362 360 363 388 353 ...  
## ..- attr(\*, "dimnames")=List of 2  
## .. ..$ : NULL  
## .. ..$ : chr [1:3] "Linf" "K" "t0"  
## $ rse : num [1:999] 30 31.3 39.1 38.1 36.6 ...  
## $ bootCI : num [1:3, 1:3] 368.064 0.307 -1.764 344.907 0.119 ...  
## ..- attr(\*, "dimnames")=List of 2  
## .. ..$ : chr [1:3] "Linf" "K" "t0"  
## .. ..$ : chr [1:3] "Median" "2.5%" "97.5%"  
## $ estiboot: num [1:3, 1:2] 374.042 0.317 -1.99 31.808 0.108 ...  
## ..- attr(\*, "dimnames")=List of 2  
## .. ..$ : chr [1:3] "Linf" "K" "t0"  
## .. ..$ : chr [1:2] "Estimate" "Std. error"  
## - attr(\*, "class")= chr "nlsBoot"

plot(bootTypical) # plot method from nlstools



plot(bootTypical, type = "boxplot", ask = FALSE)

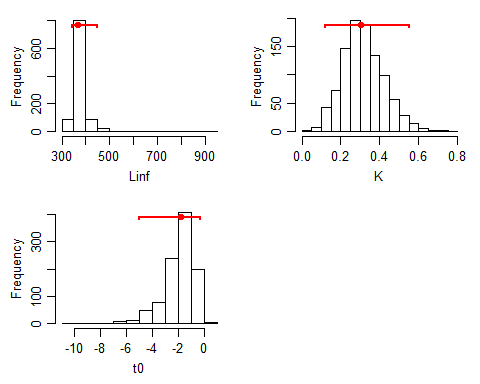


confint(bootTypical, plot=TRUE) # confint method from FSA

## 95% LCI 95% UCI  
## Linf 344.9066 447.8241  
## K 0.1192 0.5499  
## t0 -5.0373 -0.3541

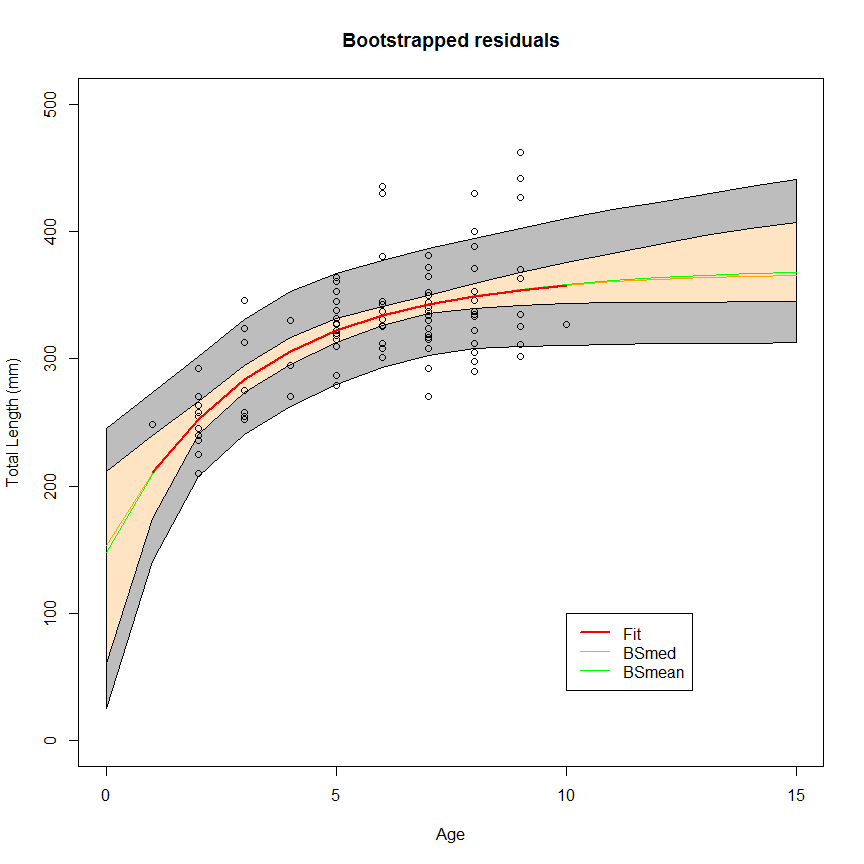
summary(bootTypical)

##   
## ------  
## Bootstrap statistics  
## Estimate Std. error  
## Linf 374.042 31.808  
## K 0.317 0.108  
## t0 -1.990 1.265  
##   
## ------  
## Median of bootstrap estimates and percentile confidence intervals  
## Median 2.5% 97.5%  
## Linf 368.0640 344.9066 447.8241  
## K 0.3072 0.1192 0.5499  
## t0 -1.7635 -5.0373 -0.3541



Let's now calculate confidence and prediction intervals, and plot them.

# prediction intervals  
ests <- bootTypical$coefboot  
ages2plot <- 0:15  
LCI <- UCI <- LPI <- UPI <- MED <- MEAN <- numeric(length(ages2plot))  
for (i in 1:length(ages2plot)) {  
 pv <- ests[,"Linf"]\*(1-exp(-ests[,"K"]\*(ages2plot[i]-ests[,"t0"])))  
 LCI[i] <- quantile(pv,0.025)  
 UCI[i] <- quantile(pv,0.975)  
 MED[i] <- quantile(pv, 0.5)  
 MEAN[i] <- mean(pv)  
 LPI[i] <- quantile(pv-bootTypical$rse,0.025)  
 UPI[i] <- quantile(pv+bootTypical$rse,0.975)  
}  
  
plot(tl~age, data=crm, type='n', xlim=c(0,15), ylim=c(0,500), xlab="Age",ylab="Total Length (mm)", main="Bootstrapped residuals")  
polygon(c(0:15, 15:0), c(UPI, rev(LPI)), col="grey74")  
polygon(c(0:15, 15:0), c(UCI, rev(LCI)), col="bisque")  
lines(0:15, MED, type="l", col="orange", lwd=1, lty=1)   
lines(0:15, MEAN, type="l", col="green", lwd=1, lty=1)   
with(crm, points(age, tl))  
with(crm, lines(age, p["Linf"]\*(1-exp(-p["K"]\*(age-p["t0"]))), col=2, lwd=2))  
legend(10,100, c("Fit", "BSmed", "BSmean"), col=c("red", "orange", "green"), lwd=c(2,1,1))



### Bayesian bootstrapping

There are several problems with standard bootstrapping. In the words of Bill Venables:

The trouble with bootstrapping residuals is that often non-linear regressions have natural bounds on the fitted value, such as asymptotes, where the usual optimistic assumption of equal variance fails. The observations themselves also obey the bounds. You still have to include points close to the bound to get decent information on it, but if you start slapping on the fitted value a residual from somewhere else where the bound was not biting, you can easily find yourself with nonsense bootstrap observations. The fact that the variance approaches zero near a bound is just something you have to put up with and ignore - it usually doesn't matter a whole lot anyway. But if you slap on a large bootstrap residual from somewhere else, things can go crazy.

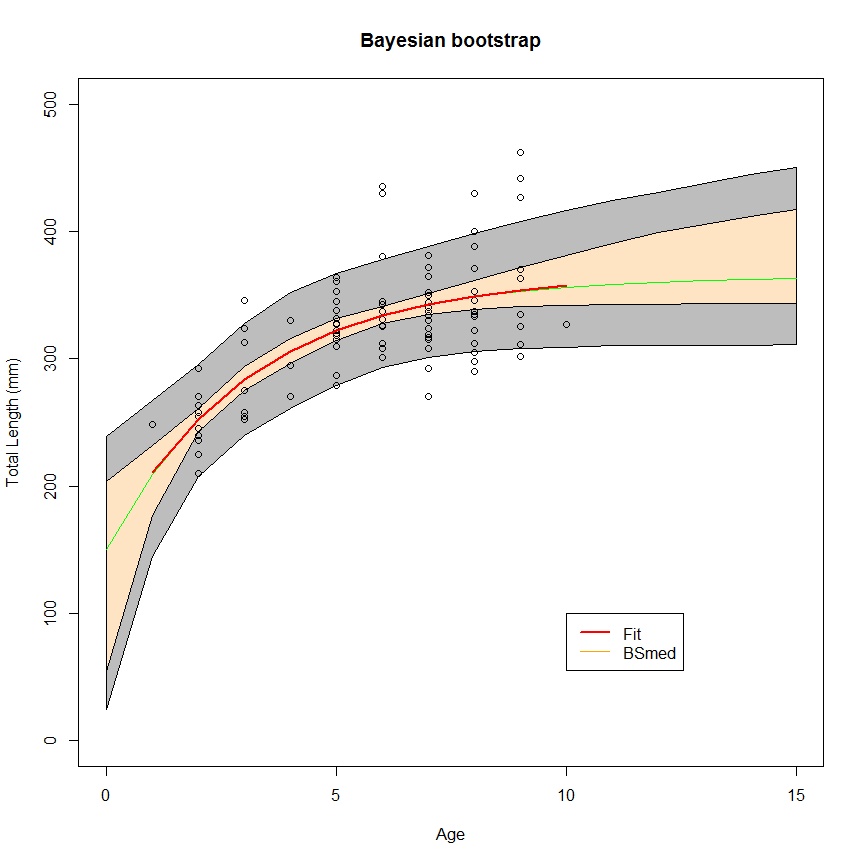
In standard bootstrapping, observations are sampled with replacement to empirically simulate the sampling distribution of a statistic estimating a parameter. This is the equivalent of conducting a model fit using observation weights that follow a multinomial distribution.

In Bayesian bootstrapping (Rubin 1981), the multinomial distribution is replaced by the Dirichlet distribution. By considering parameter estimates derived from many model fits using many samples of weights, we effectively simulate the Bayesian posterior distribution for our parameters. Praestgaard and Wellner (1993) generalised these ideas by introducing exchangeably weighted bootstraps, such that the multinomial weights of standard bootstrapping are replaced by (a variety of) exchangeable sequences that sum to the number of observations. By this method, if we select exponentially distributed weights with mean 1 we recover the Bayesian bootstrap. This approach is adopted for supplying weights generated using rexp to nls.

nr <- nrow(crm)  
NonlinM <- nls(vbTypical, data=crm, start=svTypical, control=nls.control(maxiter=1000, minFactor=1e-06))  
(a12 <- coef(NonlinM))

## Linf K t0   
## 366.4141 0.3148 -1.7142

nd <- data.frame(age=0:15)  
OK <- function(object) !inherits(object, "try-error")  
NonlinM <- update(NonlinM, trace = FALSE) ## turn off tracing  
Z <- replicate(500, {  
 repeat {  
 tmpM <- try(update(NonlinM, weights = rexp(nr)))  
 if(OK(tmpM)) break  
 }  
 list(pv=predict(tmpM, newdata = nd), sigma=summary(tmpM)$sigma)  
}, simplify=FALSE)  
   
pvs <- sapply(Z, '[[', 1)  
sigmas <- sapply(Z, '[[', 2)  
lims1 <- apply(pvs, 1, quantile, probs = c(1, 20, 39)/40) # CI: lo, med, hi  
lims2 <- apply(sweep(pvs, 2, sigmas, "+"), 1, quantile, probs = 39/40) # PI hi  
lims3 <- apply(sweep(pvs, 2, sigmas, "-"), 1, quantile, probs = 1/40) # PI lo  
  
plot(tl~age, data=crm, type='n', xlim=c(0,15), ylim=c(0,500), xlab="Age",ylab="Total Length (mm)", main="Bayesian bootstrap")  
polygon(c(0:15, 15:0), c(lims2, rev(lims3)), col="grey74")  
polygon(c(0:15, 15:0), c(lims1[3,], rev(lims1[1,])), col="bisque")  
lines(0:15, lims1[2,], type="l", col="green", lwd=1, lty=1)   
with(crm, points(age, tl))  
with(crm, lines(age, p["Linf"]\*(1-exp(-p["K"]\*(age-p["t0"]))), col=2, lwd=2))  
legend(10,100, c("Fit", "BSmed"), col=c("red", "orange"), lwd=c(2,1,1))



### Second-order Taylor Series approximation

Error propagation using Taylor series approximation uses the var-cov matrix of the fitted parameters to estimate uncertainty in fitted values. First-order Taylor series (also known as the Delta method) assumes a linear function around f(x) which may introduce biases in highly non-linear functions. The second-order Taylor expansion helps correct for this bias by introducing a second-order polynomial around f(x). [Here](http://rmazing.wordpress.com/2013/08/26/predictnls-part-2-taylor-approximation-confidence-intervals-for-nls-models/) is a good summary of the issue by the author of the propagate package, which provides the function predictNLS for this purpose.

new2 <- data.frame(age=0:15)  
p2.ci <- suppressMessages(predictNLS(fitTypical, new2, interval = "confidence"))

## Propagating predictor value #1 ...  
## Propagating predictor value #2 ...  
## Propagating predictor value #3 ...  
## Propagating predictor value #4 ...  
## Propagating predictor value #5 ...  
## Propagating predictor value #6 ...  
## Propagating predictor value #7 ...  
## Propagating predictor value #8 ...  
## Propagating predictor value #9 ...  
## Propagating predictor value #10 ...  
## Propagating predictor value #11 ...  
## Propagating predictor value #12 ...  
## Propagating predictor value #13 ...  
## Propagating predictor value #14 ...  
## Propagating predictor value #15 ...  
## Propagating predictor value #16 ...

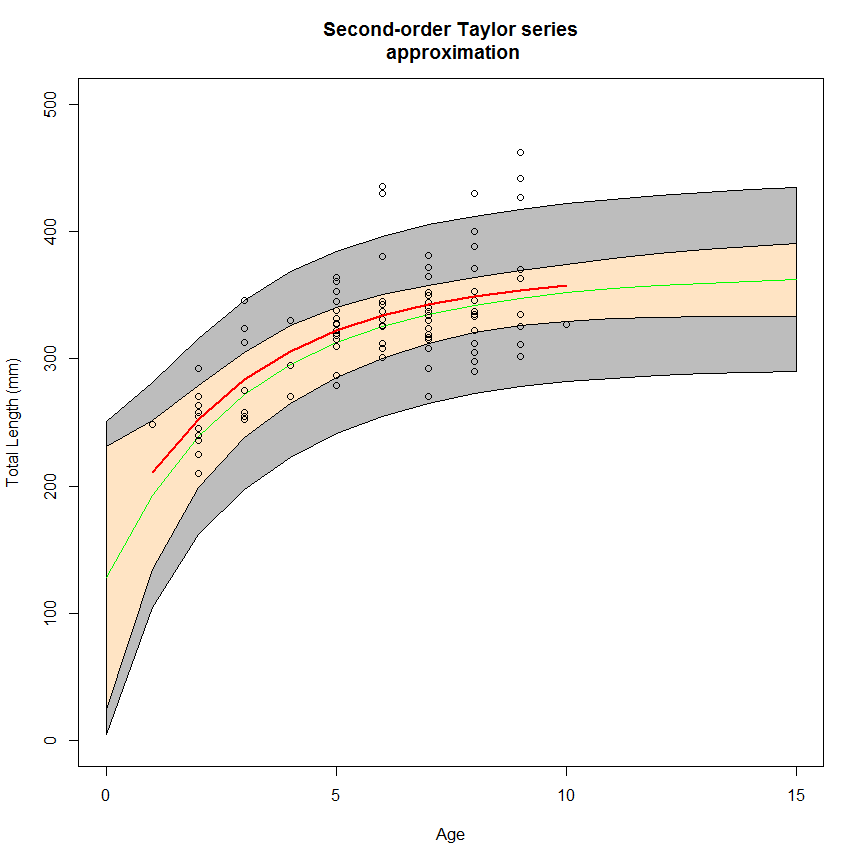
p2.pi <- suppressMessages(predictNLS(fitTypical, new2, interval = "prediction"))

## Propagating predictor value #1 ...  
## Propagating predictor value #2 ...  
## Propagating predictor value #3 ...  
## Propagating predictor value #4 ...  
## Propagating predictor value #5 ...  
## Propagating predictor value #6 ...  
## Propagating predictor value #7 ...  
## Propagating predictor value #8 ...  
## Propagating predictor value #9 ...  
## Propagating predictor value #10 ...  
## Propagating predictor value #11 ...  
## Propagating predictor value #12 ...  
## Propagating predictor value #13 ...  
## Propagating predictor value #14 ...  
## Propagating predictor value #15 ...  
## Propagating predictor value #16 ...

head(p2.ci$summary,2)

## Prop.Mean.1 Prop.Mean.2 Prop.sd.1 Prop.sd.2 Prop.2.5% Prop.97.5%  
## 1 152.8 127.8 38.82 52.26 24.28 231.4  
## 2 210.5 192.9 16.99 29.70 134.08 251.8  
## Sim.Mean Sim.sd Sim.Median Sim.MAD Sim.2.5% Sim.97.5%  
## 1 122.5 73.18 145.5 30.77 -70.31 180.3  
## 2 189.4 40.32 200.9 18.90 84.15 225.0

plot(tl~age, data=crm, type='n', xlim=c(0,15), ylim=c(0,500), xlab="Age",ylab="Total Length (mm)", main="Second-order Taylor series\n approximation")  
polygon(c(0:15, 15:0), c(p2.pi$summary$`Prop.2.5%`, rev(p2.pi$summary$`Prop.97.5%`)), col="grey74")  
polygon(c(0:15, 15:0), c(p2.ci$summary$`Prop.2.5%`, rev(p2.ci$summary$`Prop.97.5%`)), col="bisque")  
lines(0:15, p2.pi$summary$Prop.Mean.2, type="l", col="green", lwd=1, lty=1)   
with(crm, points(age, tl))  
with(crm, lines(age, p["Linf"]\*(1-exp(-p["K"]\*(age-p["t0"]))), col=2, lwd=2))



### Errors in X

An interesting feature of predictNLS is that it allows incorporation of error due to the predictors as well as the estimated parameters. These are passed into the function as part of the newdata data.frame, such that if there are *p* predictors in columns 1:*p*, then associated errors should occur in the following *p* columns (i.e. predictors [,1:3], errors [,4:6]). So, for example, if we have known error in our fish ages in the VB example (which we somtimes do) they can be incorporated thus:

head(new3 <- data.frame(age=0:15, error=seq(0.2, 3, length.out=16)))

## age error  
## 1 0 0.2000  
## 2 1 0.3867  
## 3 2 0.5733  
## 4 3 0.7600  
## 5 4 0.9467  
## 6 5 1.1333

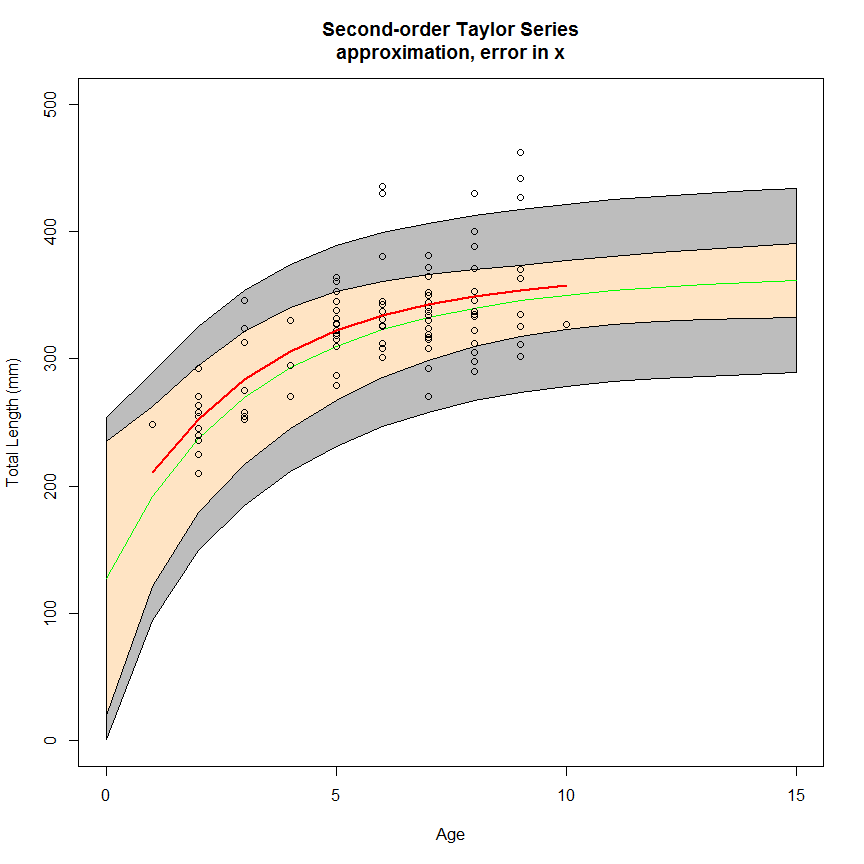
p2.ci.e <- suppressMessages(predictNLS(fitTypical, new3, interval = "confidence"))

## Propagating predictor value #1 ...  
## Propagating predictor value #2 ...  
## Propagating predictor value #3 ...  
## Propagating predictor value #4 ...  
## Propagating predictor value #5 ...  
## Propagating predictor value #6 ...  
## Propagating predictor value #7 ...  
## Propagating predictor value #8 ...  
## Propagating predictor value #9 ...  
## Propagating predictor value #10 ...  
## Propagating predictor value #11 ...  
## Propagating predictor value #12 ...  
## Propagating predictor value #13 ...  
## Propagating predictor value #14 ...  
## Propagating predictor value #15 ...  
## Propagating predictor value #16 ...

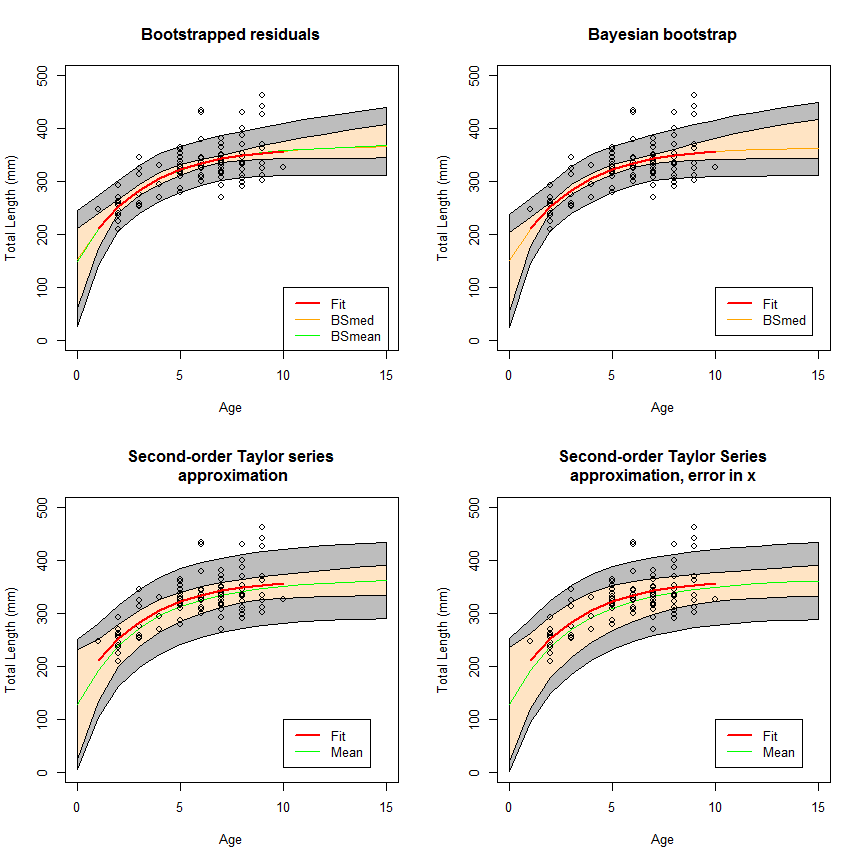
p2.pi.e <- suppressMessages(predictNLS(fitTypical, new3, interval = "prediction"))

## Propagating predictor value #1 ...  
## Propagating predictor value #2 ...  
## Propagating predictor value #3 ...  
## Propagating predictor value #4 ...  
## Propagating predictor value #5 ...  
## Propagating predictor value #6 ...  
## Propagating predictor value #7 ...  
## Propagating predictor value #8 ...  
## Propagating predictor value #9 ...  
## Propagating predictor value #10 ...  
## Propagating predictor value #11 ...  
## Propagating predictor value #12 ...  
## Propagating predictor value #13 ...  
## Propagating predictor value #14 ...  
## Propagating predictor value #15 ...  
## Propagating predictor value #16 ...

plot(tl~age, data=crm, type='n', xlim=c(0,15), ylim=c(0,500), xlab="Age",ylab="Total Length (mm)", main="Second-order Taylor Series\napproximation, error in x")  
polygon(c(0:15, 15:0), c(p2.pi.e$summary$`Prop.2.5%`, rev(p2.pi.e$summary$`Prop.97.5%`)), col="grey74")  
polygon(c(0:15, 15:0), c(p2.ci.e$summary$`Prop.2.5%`, rev(p2.ci.e$summary$`Prop.97.5%`)), col="bisque")  
lines(0:15, p2.pi.e$summary$Prop.Mean.2, type="l", col="green", lwd=1, lty=1)   
with(crm, points(age, tl))  
with(crm, lines(age, p["Linf"]\*(1-exp(-p["K"]\*(age-p["t0"]))), col=2, lwd=2))



### Put them all together



1. *theoretical* since we typically model data from post-recruited fish, and you might well imagine that larval growth is quite different from typical adult growth [↑](#footnote-ref-28)