Nonlinear Models and Bootstrapping in R

Workshop 2 (2020)

Dr. Sparkle L. Malone

The primary objectives of the workshop:

1. Fit monthly light response curves for Harvard forest to understand annual patterns ecosystem photosynthetic potential and respiration rates in temperate mixed forests.
2. Estimate monthly variance using bootstrapping.
3. Assignment: Fit monthly temperature response curves.
4. Compare the Harvard forest tower to one other AmeriFlux Tower.

Data:

The data provided in NLM_Workshop includes 3 data frames. The harv dataframe includes flux data downloaded from Ameriflux (https://ameriflux.lbl.gov) for the Harvard Forest tower (Ha1). The only behind the scenes processing includes formating the timestamp and including year, and month. The harv dataset was then divided into two files: day (PAR $>$ 0) and night (PAR $==$ 0).

```
load("~/OneDrive - Florida International University/Teaching/Workshops/Workshops/NLM_Workshop.RData")
```

Libraries:

```
library(nlstools)
```

Visualizing Data:

```
par(mai=c(1,1,0.1,0.1))
plot(harv$TIMESTAMP, harv$NEE,
     ylab=expression(paste("NEE (",mu,"mol m"^{-2} ~ s^{-1} ~ ")" )), xlab="")
```

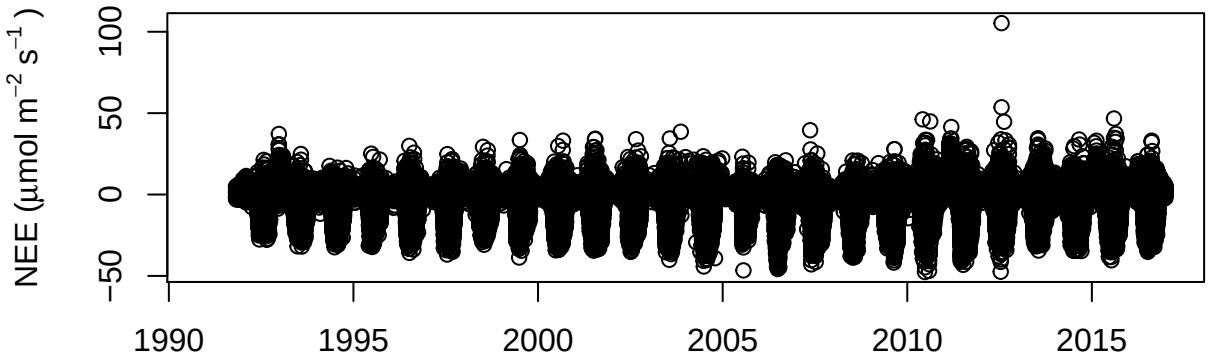


Figure 1. Net Ecosystem Exchange Rates at Harvard Forest from October 1990 to 2016.

Fitting Light Response Curves With *nls*():

To measure the relationship between photosynthetically active radiation (PAR; 400 - 700 nm) and net ecosystem exchange (NEE), we can fit a light response curve. Both PAR and NEE are in μmol m^{-2} s^{-1}. The first step in fitting a nonlinear model is to take a look at the data.

```
plot( NEE ~ PAR, data= day)
```

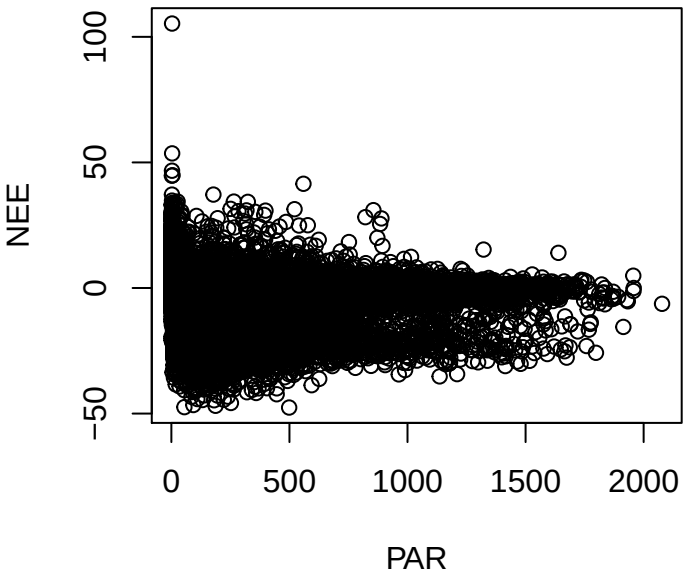


Figure 2.Net Ecosystem Exchange Rates relative to photosynthetically active radiation at Harvard forest from 1990 to 2016.

The light response curve model includes three parametrs: *a1* is the apparent quantum efficiency, *ax* is the maximum ecosystem CO_2 uptake rate, and *r* is the ecosystem respiration . We can use the *nls*() by specifying the model, data set, and starting values for the parameters.

Usage: nls(formula, data, start, . . .)

```
y = nls( NEE ~ (a1 * PAR * ax)/(a1 * PAR + ax) + r, data=day[which(day$MONTH == 07),],
         start=list(a1= -1 , ax= -1, r= 1),
         na.action=na.exclude, trace=F, control=nls.control(warnOnly=T))

summary(y)
```

```
##
## Formula: NEE ~ (a1 * PAR * ax)/(a1 * PAR + ax) + r
##
## Parameters:
##    Estimate Std. Error t value Pr(>|t|)
## a1  19428.2   152692.4   0.127    0.899
## ax    199.4      763.6   0.261    0.794
## r    -208.7      763.7  -0.273    0.785
##
## Residual standard error: 12.54 on 6656 degrees of freedom
##
## Number of iterations till stop: 13
## Achieved convergence tolerance: 0.8081
## Reason stopped: step factor 0.000488281 reduced below 'minFactor' of 0.000976562
##   (1760 observations deleted due to missingness)
```

You can see here that the data doesn't support the model very well. Termination before convergence happens

upon completion of maximum iterations, in the case of a singular gradient, and in the case that the step-size factor is reduced below a minimum factor. The starting values are the issue here.

Starting Values for Nonlinear Models:

To reduce the bias introduced by the selection of starting values we can use selfStart to construct self-starting nonlinear models.

Usage: selfStart(model, initial)

```
# 1. Create a function of the model:

lrcModel <- function(PAR, a1, ax, r) {
  NEE <- (a1 * PAR * ax)/(a1 * PAR + ax) + r
  return(NEE)
}

# 2. Initial: create a function that calculates the initial values from the data.

lrc.int <- function (mCall, LHS, data){
  x <- data$PAR
  y <- data$NEE

  r <- max(na.omit(y), na.rm=T) # Maximum NEE
  ax <- min(na.omit(y), na.rm=T) # Minimum NEE
  a1 <- (r + ax)/2 # Midway between r and a1

  # Create limits for the parameters:
  a1[a1 > 0]<- -0.1
  r[r > 50] <- ax*-1
  r[r < 0] <- 1

  value = list(a1, ax, r) # Must include this for the selfStart function
  names(value) <- mCall[c("a1", "ax", "r")] # Must include this for the selfStart function
  return(value)
}
```

Use the selfStart function to calculate initial values:

```
# Selfstart function
SS.lrc <- selfStart(model=lrcModel,initial= lrc.int)

# 3. Find initial values:
iv <- getInitial(NEE ~ SS.lrc('PAR', "a1", "ax", "r"),
                 data = day[which(day$MONTH == 07),])
iv
```

```
## $a1
## [1] -0.1
##
## $ax
## [1] -47.44
##
## $r
## [1] 47.44
```

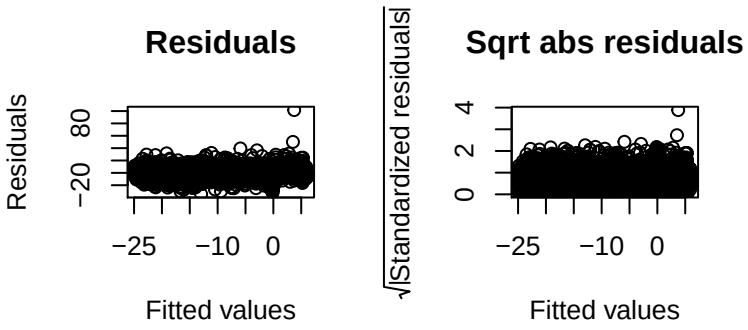
Use initial values in the model:

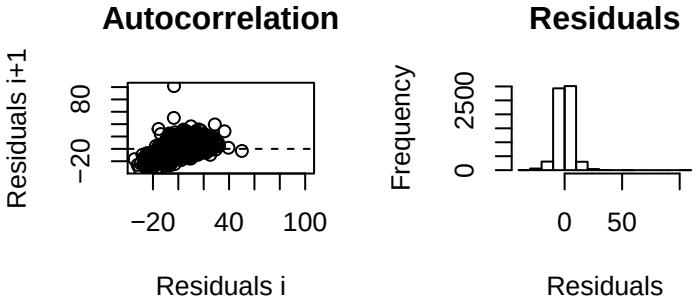
```
y = nls( NEE ~ (a1 * PAR * ax)/(a1 * PAR + ax) + r, day[which(day$MONTH == 07),],
         start=list(a1= iv$a1 , ax= iv$ax, r= iv$r),
         na.action=na.exclude, trace=F, control=nls.control(warnOnly=T))

summary(y)
```

```
##
## Formula: NEE ~ (a1 * PAR * ax)/(a1 * PAR + ax) + r
##
## Parameters:
##    Estimate Std. Error t value Pr(>|t|)
## a1  -0.8732     0.0289  -30.21   <2e-16 ***
## ax -31.7395     0.2828 -112.25   <2e-16 ***
## r    6.1558     0.1878   32.79   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.728 on 6656 degrees of freedom
##
## Number of iterations to convergence: 6
## Achieved convergence tolerance: 4.108e-06
##   (1760 observations deleted due to missingness)
```

Here, the model converged. Now, let check assumptions

```
res.lrc <- nlsResiduals(y)
par(mfrow=c(2,2))
plot(res.lrc, which=1)# Residulas vs fitted values (Constant Variance)
plot(res.lrc, which=3) # Standardized residuals
plot(res.lrc, which=4) # Autocorrelation
plot(res.lrc, which=5) # Histogram (Normality)
```





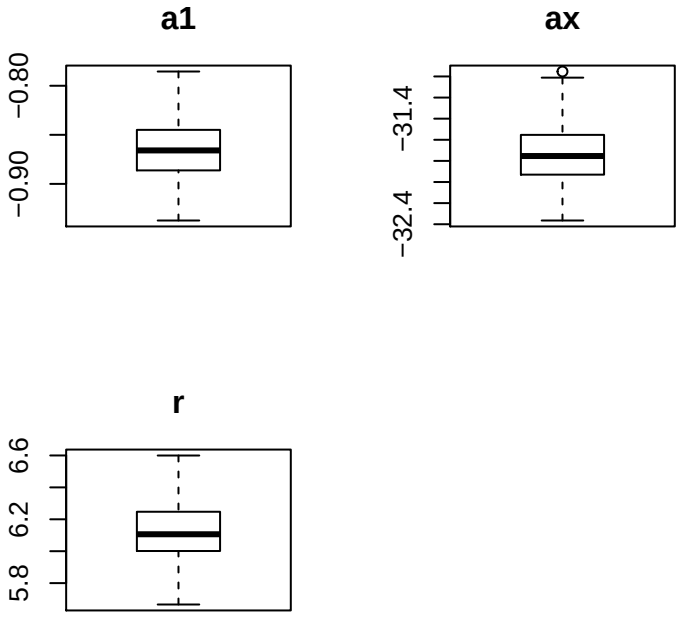
In addition to the visual assessment of the model assumptions, the normality of residuals maybe evaluated using the Shapiro-Wilk test and autocorrelation in residuals may be assessed with the runs test. These tests are supplements that are occasionally useful next to the routine visual assessment of the model assumptions. Both tests are available through the function *test.nlsResiduals()*.

We can bootstrap to estimate errors for the parameters by resampling the data. The function *nlsBoot()* uses non-parametric bootstrap of mean centered residuals to obtain a number (niter) of bootstrap estimates. Bootstrap estimates and standard errors together with the median and percentiles confidence intervals are displayed by the *summary()*. The *nlsBoot()* provides confidence intervals even if the optimization algorithm fails to converge for some of the bootstrapped samples.

```
results <- nlsBoot(y, niter=100 )
summary(results)
```

```
##
## ------
## Bootstrap statistics
##       Estimate Std. error
## a1  -0.8667021 0.03116918
## ax -31.7379704 0.32420987
## r    6.1130026 0.19694716
##
## ------
## Median of bootstrap estimates and percentile confidence intervals
##         Median        2.5%       97.5%
## a1  -0.8658962  -0.9285697  -0.8042262
## ax -31.7537476 -32.2734110 -31.0507561
## r    6.1062014   5.6961805   6.4672755
```

```
plot(results, type = "boxplot")
```



You know:

1. How to create a function for your model of interest.
2. How to use *selfStart*() to find starting values based on your data set.

3. How to use *nls*() to fit nonlinear models.
4. How to use *nlsBoot*() to estimate the error around the parameter values by resampling your data.

Exercise: How variable are NEE rates over an annual cycle in Harvard Forest?

Harvard Forest is a mixed temperate forest. We can see seasonal patterns in NEE in *Figure 1.* We want to quantify just how variable rates of NEE are annually. To do this we will fit light and temperature response curves monthly and compare parameter values.

Workflow:

1. Create a dataframe to store month parameter values (parms.Month).
2. Write a function to the fit model and extract paramters (nee.day).
3. Write a loop to fit monthly curves and add parameters to a dataframe (parms.Month).
4. Bootstrapping for error estimation.

1. Create a dataframe to store month parameter values (parms.Month):

```
# Dataframe to store parms and se

parms.Month <- data.frame(
  MONTH=numeric(),
  a1=numeric(),
  ax=numeric(),
  r=numeric(),
  a1.pvalue=numeric(),
  ax.pvalue=numeric(),
  r.pvalue=numeric(), stringsAsFactors=FALSE, row.names=NULL)

parms.Month[1:12, 1] <- seq(1,12,1) # Adds months to the file
```

2. Write a function to fit the model and extract paramters (nee.day).

```
nee.day <- function(dataframe){ y = nls( NEE ~ (a1 * PAR * ax)/(a1 * PAR + ax) + r, dataframe,
                                         start=list(a1= iv$a1 , ax= iv$ax, r= iv$r),
                                         na.action=na.exclude, trace=F,
                                         control=nls.control(warnOnly=T))


y.df <- as.data.frame(cbind(t(coef(summary(y)) [1:3, 1]), t(coef(summary(y)) [1:3, 4])))
names(y.df) <-c("a1","ax", "r", "a1.pvalue", "ax.pvalue", "r.pvalue")
return (y.df )}
```

3. Write a loop to fit monthly curves and add paramters to a dataframe (parms.Month).

```
try(for(j in unique(day$MONTH)){

  # Determines starting values:
    iv <- getInitial(NEE ~ SS.lrc('PAR', "a1", "ax", "r"), data = day[which(day$MONTH == j),])

  # Fits light response curve:
    y3 <- try(nee.day(day[which(day$MONTH == j),]), silent=T)

  # Extracts data and saves it in the dataframe
    try(parms.Month[c(parms.Month$MONTH == j ), 2:7 ] <- cbind(y3), silent=T)

    rm(y3)
  }, silent=T)

parms.Month
```

4. Bootstrapping

```
# Create file to store parms and se
boot.NEE <- data.frame(parms.Month[, c("MONTH")]); names (boot.NEE) <- "MONTH"
boot.NEE$a1.est <- 0
boot.NEE$ax.est<- 0
boot.NEE$r.est<- 0
boot.NEE$a1.se<- 0
boot.NEE$ax.se<- 0
boot.NEE$r.se<- 0

 for ( j in unique(boot.NEE$Month)){

 y1 <-day[which(day$MONTH == j),] # Subsets data

  # Determines the starting values:
  iv <- getInitial(NEE ~ SS.lrc('PAR', "a1", "ax", "r"), data = y1)

  # Fit curve:
  day.fit <- nls( NEE ~ (a1 * PAR * ax)/(a1 * PAR + ax) + r, data=y1,
                      start=list(a1= iv$a1 , ax= iv$ax, r= iv$r),
                      na.action=na.exclude, trace=F, control=nls.control(warnOnly=T))

   # Bootstrap and extract values:
  try(results <- nlsBoot(day.fit, niter=100 ), silent=T)
  try(a <- t(results$estiboot)[1, 1:3], silent=T)
  try(names(a) <- c('a1.est', 'ax.est', 'r.est'), silent=T)
  try( b <- t(results$estiboot)[2, 1:3], silent=T)
  try(names(b) <- c('a1.se', 'ax.se', 'r.se'), silent=T)
  try(c <- t(data.frame(c(a,b))), silent=T)
```

```
  # Add bootstrap data to dataframe:
  try(boot.NEE[c(boot.NEE$MONTH == j), 2:7] <- c[1, 1:6], silent=T)
  try(rm(day.fit, a, b, c, results, y1), silent=T)

}

  lrc <- merge( parms.Month, boot.NEE, by.x="MONTH", by.y="MONTH") # Merge dataframes
  lrc
```

- Notice I used *try(code , silent=T)* to ensure my loop will continue to move through the months even if a model does not converge. *try* is a wrapper to run an expression that might fail and allow the user's code to handle error-recovery.

Assignment:

Fit monthly temperature response curves using a similar approach with the night data from harv (night).

$$\boldsymbol{NEE} \sim \boldsymbol{a} \quad \mathbf{exp}^{b*TA}$$

a is the base respiration rate when air temperature is 0 řC and b is an empirical coefficient.

Workflow:

1. Create a dataframe to store month parameter values (parms.Month).
2. Write a function to the fit model and extract parameters (nee.night).
3. Write a loop to fit monthly curves and add parameters to a dataframe (parms.Month).
4. Bootstrapping for error estimation.