**Supplementary materials**

**Modification of the code of the fitplc package to add ‘seedling ID’ as random effect in the Weibull model.**

library(fitplc)

library(nlme)

library(MASS)

library(ggplot2)

#1. Creation of the nlme model

#data: df

varnames = c(PLC = "PLC\_brut", WP = "Pot")

dfr = df

x = 50

weights = NULL

nboot = 999

#function:

#from fitplc

plc\_to\_relk <- function(plc)

 (100 - plc) / 100

ab\_to\_px <- function(a, b, x)

 (log(1 / (1 - x / 100) - 1) / a) + b

do\_sigmoid\_fit <- function(data,

 W = NULL,

 boot = FALSE,

 nboot) {

 data <- data[data$PLC < 100 & data$PLC > 0, ]

 # Transformation as per P&vW

 data$logPLC <- log(100 / data$PLC - 1)

 if (!is.null(W)) {

 lmfit <- lm(logPLC ~ minP, data = data, weights = W)

 br <-

 if (boot)

 suppressWarnings(bootfit(

 lmfit,

 n = nboot,

 Data = data,

 startList = NULL,

 weights = W

 ))

 else

 NA

 } else {

 lmfit <- lm(logPLC ~ minP, data = data)

 br <-

 if (boot)

 suppressWarnings(bootfit(

 lmfit,

 n = nboot,

 Data = data,

 startList = NULL

 ))

 else

 NA

 }

 return(list(fit = lmfit, boot = br))

}

# Calculate Sx, Px, given log-linear fit of sigmoidal model

sigfit\_coefs <- function(c1, c2, x) {

 a <- c2

 b <- c1 / c2

 Px <- ab\_to\_px(a, b, x)

 # Derivative of sigmoid

 sig2d <-

 function(Px, a, b)

 - (exp(a \* (Px - b)) \* a / (1 + exp(a \* (Px - b))) ^ 2)

 Sx <- -100 \* sig2d(Px, a, b)

 list(Px = unname(Px), Sx = unname(Sx))

}

if (!is.null(substitute(Identifiant\_Ind))) {

 G <- eval(substitute(Identifiant\_Ind), dfr)

 fitran <- TRUE

 bootci <- FALSE

}

# Extract data

plc <- dfr[[varnames["PLC"]]]

P <- dfr[[varnames["WP"]]]

if (any(is.na(c(plc, P))))

 stop("Missing values found in PLC or WP - remove first!")

relK <- plc\_to\_relk(plc)

# Need absolute values of water potential

if (mean(P) < 0)

 P <- -P

# Dataset tidied

Data <- data.frame(P = P,

 PLC = plc,

 relK = relK,

 G = G)

Data$minP <- -Data$P # negative valued water potential

# guess starting values from sigmoidal

f <- do\_sigmoid\_fit(Data, boot = FALSE)

p <- coef(f$fit)

sp <- sigfit\_coefs(p[1], p[2], x = x)

# fit

Data$X <- x

#2. Creation of confidence intervals around the curve

rm(x)

Data$G <- as.factor(Data$G)

# Weighted NLS

fm1\_ <- nlme(

 relK ~ fweibull(P, SX, PX, X),

 fixed = list(SX ~ 1, PX ~ 1),

 random = SX + PX ~ 1 | G,

 start = list(fixed = c(SX = sp$Sx,

 PX = sp$Px)),

 #weights=W,

 control = nlmeControl(msMaxIter = 1000, eval.max = 1e06),

 data = Data

)

Data.new <- # create a new copy of the groupedData object

 groupedData(relK ~ P | G,

 data = as.data.frame(Data))

xvals\_ <- with(Data.new, seq(min(P), max(P), length.out = 100))

## CI calculated by Normal (not presented in the article)

nresamp <- 1000

## pick new parameter values by sampling from multivariate normal distribution based on fit

pars.picked\_ <-

 mvrnorm(nresamp, mu = fixef(fm1\_), Sigma = vcov(fm1\_))

## predicted values: useful below

pframe\_ <- with(Data.new, data.frame(P = xvals\_))

pframe\_$X <- 50

pframe\_$relK <- predict(fm1\_, newdata = pframe\_, level = 0)

## utility function

get\_CI <- function(y, pref = "") {

 r1 <- t(apply(y, 1, quantile, c(0.025, 0.975)))

 setNames(as.data.frame(r1), paste0(pref, c("lwr", "upr")))

}

set.seed(101)

yvals\_ <- apply(pars.picked\_, 1,

 function(x) {

 fweibull(xvals\_, x[1], x[2])

 })

c1\_ <- get\_CI(yvals\_) # CI calculé par Normale

## bootstrapping

sampfun\_ <- function(fitted, data, idvar = "G") {

 pp <- predict(fitted, levels = 1)

 rr <- residuals(fitted)

 dd <- data.frame(data, pred = pp, res = rr)

 ## sample groups with replacement

 iv <- levels(data[[idvar]])

 bsamp1 <- sample(iv, size = length(iv), replace = TRUE)

 bsamp2 <- lapply(bsamp1,

 function(x) {

 ## within groups, sample \*residuals\* with replacement

 ddb <- dd[dd[[idvar]] == x, ]

 ## bootstrapped response = pred + bootstrapped residual

 ddb$relk <- ddb$pred +

 sample(ddb$res, size = nrow(ddb), replace = TRUE)

 return(ddb)

 })

 res <- do.call(rbind, bsamp2) ## collect results

 if (is(data, "groupedData"))

 res <- groupedData(res, formula = formula(data))

 return(res)

}

pfun\_ <- function(fm) {

 tryCatch({

 predict(fm, newdata = pframe\_, level = 0)

 }, error = function(e) {

 })

}

set.seed(101)

yvals2\_ <- replicate(nresamp,

 pfun\_(update(fm1\_, data = sampfun\_(fm1\_, Data.new, "G"))))

get\_CI2 <- function(y, pref = "") {

 y = y[-(which(sapply(y, is.null), arr.ind = TRUE))] # ne pas faire tourner cette ligne pour EPN

 r1 <- t(apply(as.data.frame(y), 1, quantile, c(0.025, 0.975)))

 setNames(as.data.frame(r1), paste0(pref, c("lwr", "upr")))

}

c2\_ <- get\_CI2(yvals2\_, "boot\_") # CI calculé par bootstrap

pframe\_ <- data.frame(pframe\_, c1\_, c2\_)

pframe\_$minP <- -pframe\_$P

## plot

theme\_set(theme\_bw())

ggplot(Data, aes(minP, relK)) +

 geom\_line(alpha = 0.2, aes(group = G)) +

 geom\_line(data = pframe\_, col = "red") +

 geom\_ribbon(

 data = pframe\_,

 aes(ymin = boot\_lwr, ymax = boot\_upr),

 colour = NA,

 alpha = 0.3,

 fill = "red"

 )

ggplot(Data, aes(P)) +

 geom\_hline(yintercept = 0, lty = 2) +

 geom\_ribbon(

 data = pframe\_,

 aes(ymin = lwr - relK, ymax = upr - relK),

 colour = "blue",

 fill = NA

 ) +

 geom\_ribbon(

 data = pframe\_,

 aes(ymin = boot\_lwr - relK, ymax = boot\_upr - relK),

 colour = "red",

 fill = NA

 )

ggplot(Data, aes(-minP, (1 - relK) \* 100)) +

 geom\_line(alpha = 0.2, aes(group = G)) +

 geom\_line(data = pframe\_, col = "red") +

 geom\_ribbon(

 data = pframe\_,

 aes(

 ymin = (1 - boot\_lwr) \* 100,

 ymax = (1 - boot\_upr) \* 100

 ),

 colour = NA,

 alpha = 0.3,

 fill = "red"

 )

ggplot(Data, aes(-minP, (1 - relK) \* 100)) +

 geom\_line(alpha = 0.2, aes(group = G)) +

 geom\_line(data = pframe\_, col = "red") +

 geom\_ribbon(

 data = pframe\_,

 aes(ymin = (1 - lwr) \* 100, ymax = (1 - upr) \* 100),

 colour = NA,

 alpha = 0.3,

 fill = "red"

 )

#3. Extraction of P12, P50 and P88 values

#from getPx

fm1\_

x = c(12, 50, 88)

resc\_cons <- 1

X <- 1 - x / 100

px <- fm1\_$coefficients$fixed[2]

sx <- fm1\_$coefficients$fixed[1]

v <- (50 - 100) \* log(1 - 50 / 100)

p <- px \* (log(1 - x / 100) / log(1 - 50 / 100)) ^ (v / (px \* sx))

#Sometimes the upper CI cannot be calculated and will be reported as NA.

#This indicates that the upper confidence bound is outside the range of the data,

#and can therefore not be reliably reported.

lwrci <-

 approx(x = pframe\_$boot\_lwr \* resc\_cons,

 y = pframe\_$P,

 xout = X)$y

uprci <-

 approx(x = pframe\_$boot\_upr \* resc\_cons,

 y = pframe\_$P,

 xout = X)$y

vec <- c(p, lwrci, uprci)