Quantitative phase analysis of complex fats during crystallization

Naomi Arita-Merino,[†,‡] Hein van Valenberg,[‡] Elliot Paul Gilbert,[¶] and Elke Scholten[*,†]

[†]*Physics and Physical Chemistry of Foods, Wageningen University, Wageningen, The Netherlands*

[‡]*Dairy Science and Technology, Food Quality and Design, Wageningen University, Wageningen, The Netherlands*

[¶]*Australian Centre for Neutron Scattering, Australian Nuclear Science and Technology Organisation, Kirrawee DC, NSW 2232, Australia*

E-mail: elke.scholten@wur.nl

Phone: +31 (0)317 482288

Supplementary Information

R programming language script used to decompose the time-resolved XRD patterns of AMF and PO.

```
#####Getting started#####
#Clean global environment
rm(list = ls())
#Set working directory
mypath <- 'write here your path' #Adjust
setwd(mypath)

#####Load packages and write functions I will need#####
#Packages
if(!require(dplyr)) install.packages("dplyr")
require(dplyr)
if(!require(tidyverse)) install.packages("tidyverse")
require(tidyverse)
if(!require(plotly)) install.packages("plotly")
library(plotly)
if(!require(viridis)) install.packages("viridis")
library(viridis)
if(!require(magrittr)) install.packages("magrittr")
library(magrittr)
if(!require(ggplot2)) install.packages("ggplot2")
require(ggplot2)
if(!require(expss)) install.packages("expss")
require(expss)
if(!require(tidyr)) install.packages("tidyr")
```

```
library(tidyr)

#Functions
dToAngle <-function(x) asin(lambda/2/x)*360/pi
dToq <- function(x) 10*2*pi/x
AngleTod <- function(x) lambda/2/sin(x*pi/360)
AngleToq <- function(x) 10*4*pi*sin(x*pi/360)/lambda
spacings <- c(5.5, 5, 4.5, 4, 3.5, 3)
plotall <- function (i, fittedModel,WAXDdata, b, el) {
  ggplot(data = WAXDdata[WAXDdata$Index == i,]) +
    geom_point(shape = 4, aes(x = q, y = CPS - b)) +
    geom_line(size = 1, color = "black", aes(x = q, y = fitted(fittedModel)
- b)) +
    geom_point(shape = 20, aes(x = q, y = resid(fittedModel))) +
    geom_line(size = 1, color = "#FDE725FF", aes(x = q,
                                                 y =
coef(fittedModel)['hl']*coef(fittedModel)['fl']^(2*coef(fittedModel)['ml'])
/

(coef(fittedModel)['fl']^2 + (2^(1/coef(fittedModel)['ml'])-1)*(q-
el)^2)^coef(fittedModel)['ml'])) +
    geom_line(size = 1, color = "#94D840FF", aes(x = q,
                                                 y =
coef(fittedModel)['ha']*(coef(fittedModel)['fl']/coef(fittedModel)['fa'])^(
2*coef(fittedModel)['ma']) /

((coef(fittedModel)['fl']/coef(fittedModel)['fa'])^2 +
(2^(1/coef(fittedModel)['ma'])-1)*
                                                   (q-
coef(fittedModel)['ea'])^2)^coef(fittedModel)['ma'])) +
    geom_line(size = 1, color = "#32648EFF", aes(x = q,
                                                 y =
coef(fittedModel)['hbp2']/coef(fittedModel)['hbp1']*(coef(fittedModel)['fl'
]/coef(fittedModel)['fbp1'])^(2*coef(fittedModel)['mbp1']) /

((coef(fittedModel)['fl']/coef(fittedModel)['fbp1'])^2 +
(2^(1/coef(fittedModel)['mbp1'])-1)*
                                                   (q-
coef(fittedModel)['ebp1'])^2)^coef(fittedModel)['mbp1'])) +
    geom_line(size = 1, color = "#32648EFF", aes(x = q,
                                                 y =
coef(fittedModel)['hbp2']*(coef(fittedModel)['fl']/coef(fittedModel)['fbp2'
])^(2*coef(fittedModel)['mbp2']) /

((coef(fittedModel)['fl']/coef(fittedModel)['fbp2'])^2 +
(2^(1/coef(fittedModel)['mbp2'])-1)*
                                                   (q-
coef(fittedModel)['ebp2'])^2)^coef(fittedModel)['mbp2'])) +
    geom_line(size = 1, color = "#32648EFF", aes(x = q,
                                                 y =
coef(fittedModel)['hbp2']/coef(fittedModel)['hbp3']*(coef(fittedModel)['fl'
]/coef(fittedModel)['fbp3'])^(2*coef(fittedModel)['mbp3']) /

((coef(fittedModel)['fl']/coef(fittedModel)['fbp3'])^2 +
(2^(1/coef(fittedModel)['mbp3'])-1)*
                                                   (q-
coef(fittedModel)['ebp3'])^2)^coef(fittedModel)['mbp3'])) +
    geom_line(size = 1, color = "#32648EFF", aes(x = q,
                                                 y =
coef(fittedModel)['hbp2']/coef(fittedModel)['hbp4']*(coef(fittedModel)['fl'
]/coef(fittedModel)['fbp4'])^(2*coef(fittedModel)['mbp4']) /
```

```
((coef(fittedModel)['fl']/coef(fittedModel)['fbp4'])^2 +
(2^(1/coef(fittedModel)['mbp4'])-1)*
                                                    (q-
coef(fittedModel)['ebp4'])^2)^coef(fittedModel)['mbp4'])) +
    geom_line(size = 1, color = "#481568FF", aes(x = q,
                                                 y =
coef(fittedModel)['hb']*(coef(fittedModel)['fl']/coef(fittedModel)['fb'])^(
2*coef(fittedModel)['mb']) /

((coef(fittedModel)['fl']/coef(fittedModel)['fb'])^2 +
(2^(1/coef(fittedModel)['mb'])-1)*
                                                    (q-
coef(fittedModel)['eb'])^2)^coef(fittedModel)['mb'])) +
    labs(x = "q [1/nm]", y = 'Intensity [A. U.]') +
    scale_x_continuous(sec.axis = sec_axis(~. , breaks = dToq(spacings),
                                           labels = rbind(spacings), name =
"d-spacing [nm]"), expand = c(0.001,0.001)) +
    scale_y_continuous(expand = c(0.01,1)) +
    theme(panel.background = element_blank(), panel.grid.major =
element_blank(), panel.border = element_rect(fill = NA),
          panel.grid.minor = element_blank(),
          axis.text = element_text(size = 10), text = element_text(size =
10), legend.text = element_text(size = 10))}
accumulatedPlot <- function(IndexMIN, IndexMAX) {
  ggplot(data = subset(WAXDdata, Index <= IndexMAX & Index >= IndexMIN),
         aes(x = q, y = CPS, group = Index)) +
    scale_x_continuous(sec.axis = sec_axis(~. , breaks =
round(c(unlist(lapply(spacings, dToq))),2), labels = rbind(spacings), name
= "d-spacing [Å]"),
                       expand = c(0,0)) +
    scale_y_continuous(expand = c(0.01,0)) +
    labs(x = "q [1/nm]", y = 'Accumulated intensity [A. U.]', fill =
"Index") +
    geom_area(aes(fill = Index), position = position_stack(reverse = T)) +
    theme(panel.background = element_blank(), panel.grid.major =
element_blank(), panel.border = element_rect(fill = NA), panel.grid.minor =
element_blank(),
          strip.text.x = element_blank(), axis.text =
element_text(size=12), text = element_text(size = 12), legend.text =
element_text(size = 12)) +
    scale_fill_viridis(direction = -1)}

lPeak <- function(q, hl, fl, ml, el) {hl*fl^(2*ml) / (fl^2 + (2^(1/ml)-
1)*(q-el)^2)^ml}
aPeak <- function(q, fl, ha, fa, ma, ea) {ha*(fl/fa)^(2*ma) / ((fl/fa)^2 +
(2^(1/ma)-1)*(q-ea)^2)^ma}
bp2Peak <- function(q, fl, hbp2, fbp2, mbp2, ebp2) {hbp2*(fl/fbp2)^(2*mbp2)
/ ((fl/fbp2)^2 + (2^(1/mbp2)-1)*(q-ebp2)^2)^mbp2}
bp1Peak <- function(q, fl, hbp2, hbp1, fbp1, mbp1, ebp1)
{hbp2/hbp1*(fl/fbp1)^(2*mbp1) / ((fl/fbp1)^2 + (2^(1/mbp1)-1)*(q-
ebp1)^2)^mbp1}
bp3Peak <- function(q, fl, hbp2, hbp3, fbp3, mbp3, ebp3)
{hbp2/hbp3*(fl/fbp3)^(2*mbp3) / ((fl/fbp3)^2 + (2^(1/mbp3)-1)*(q-
ebp3)^2)^mbp3}
bp4Peak <- function(q, fl, hbp2, hbp4, fbp4, mbp4, ebp4)
{hbp2/hbp4*(fl/fbp4)^(2*mbp4) / ((fl/fbp4)^2 + (2^(1/mbp4)-1)*(q-
ebp4)^2)^mbp4}
bPeak <- function(q, fl, hb, fb, mb, eb) {hb*(fl/fb)^(2*mb) / ((fl/fb)^2 +
(2^(1/mb)-1)*(q-eb)^2)^mb}
```

```
reportEstimates <- function(i, fitsList) {
  return(tryCatch(coef(fitsList[[1]][[i]]),
                  error=function(e) rep(NA,
length(coef(fitsList[[1]][[i]])))))}
reportSS <- function(i, fitsList) {
  return(tryCatch(deviance(fitsList[[1]][[i]]),
                  error=function(e) NA))}

colors4plots <- viridis(5)
names(colors4plots) <- c("total", "beta", "betap", "alpha", "liquid")
colors4plots <- rbind(colors4plots, viridis(5, alpha = 0.4))

####Specify values#####
lambda <- 1.54060
#Define the selected peak positions in Å
elInitial <- dToq(4.54)
eaInitial <- dToq(4.15)
ebp1Initial <- dToq(4.35)
ebp2Initial <- dToq(4.20)
ebp3Initial <- dToq(4.03)
ebp4Initial <- dToq(3.80)
ebInitial <- dToq(4.59)

####Load files#####
##From a file containing al the scans
data <- read_excel("data.xlsx") #or change to .csv
###data should have columns named: Index, Angle, CPS, Time, Temperature, q,
d

Tcr <- 17 ##Adapt depending on the crystallization temperature
Time0 <- min(data$Time)
lastScan <- max(data$Index)
firstScan <- min(data$Index)

####Qualitative phase identification####
###Plot accumulated scans to identify polymorphs
###Adjust the range to be plotted to visualize the transitions
accumulatedPlot(firstScan, lastScan) ##Based on this plot, is there ALPHA,
BETA, BETA PRIME?
###Plot to check the first scan
accumulatedPlot(firstScan, firstScan+12) #BASED on this plot: Is there
alpha from the first scan?
###Plot to identify transitions
accumulatedPlot(firstScan+10, firstScan+15)
###PLot to check end of transition
accumulatedPlot(firstScan+15, firstScan+30)
####Plot to check if there is a change at the end
accumulatedPlot(lastScan-25, lastScan)

###Define the scans (Index) at which the transitions are observed
####Type lastScan+1 for min values if a given phase is never present (e.g.
betaMin <- lastScan+1)
alphaMin <- firstScan #First scan in which alpha is detected. Type
(lastScan +1) if it is not there
betaMin <- firstScan+12 #First scan in which beta is detected. Type
(lastScan +1) if it is not there
betapMin <- firstScan+12 #First scan in which beta prime is detected.
alphaMax <- firstScan+20 #Last scan in which alpha is detected.

##Classify scans according to the phases present
###ADD MORE OPTIONS WHEN NEEDED.
```

```
###Define scans range with only liquid signal
if(alphaMin > firstScan & alphaMin > lastScan) lScans <-
c(firstScan:(betapMin-1))
if(alphaMin > firstScan & alphaMin < lastScan) lScans <-
c(firstScan:(alphaMin-1))
###Define scans range with liquid and alpha signal
if(alphaMin <= lastScan & betapMin <= betaMin) aScans =
c(alphaMin:(betapMin-1))
if(alphaMin <= lastScan & betapMin > betaMin) aScans = c(alphaMin:(betaMin-
1))
###Define scans from the transition period
if(alphaMax <= lastScan & betaMin >lastScan) {aANDbpScans =
c(betapMin:alphaMax)} #If beta' and beta appear together after alpha
if(alphaMax > betaMin & betapMin < betaMin) {aANDbpScans =
c(betapMin:(betaMin-1))} #If beta' appears first
if(alphaMin <= lastScan & betapMin >= betaMin) {aANDbpANDbScans <-
c(betapMin:alphaMax)}
if(alphaMin <= lastScan & betapMin < betaMin & betaMin <= alphaMax)
{aANDbpANDbScans = c(betaMin:alphaMax)} #If beta p appears first than beta
if(alphaMin <= lastScan & betapMin > betaMin) {aANDbScans =
c(betaMin:alphaMax)} #If beta appears first
if(alphaMin <= lastScan & betapMin > betaMin & betapMin <= alphaMax)
{aANDbpANDbScans = c(betapMin:alphaMax)} #If beta appears first than beta p
###Define scans with beta prime, beta and liquid
if(alphaMin > lastScan & betapMin == betaMin) {bpandbScans <-
c(betapMin:lastScan)} #If there was crystallization directly into beta and
beta p
if(alphaMin < lastScan & betaMin < lastScan) {bpandbScans <-
c((alphaMax+1):lastScan)} #If there are scans with beta and beta p
if(betaMin > lastScan & alphaMin > lastScan) {bpScans =
c(betapMin:lastScan)} #If there was crystallization directly into beta p
if(betaMin > lastScan & alphaMin < lastScan) {bpScans =
c((alphaMax+1):lastScan)} #If there was an alpha to beta p transition
without beta

####Decompose the first scan####
if (firstScan == alphaMin) {first <- nls(CPS ~ b + lPeak(q, hl, fl, ml, el)
+ aPeak(q, fl, ha, fa, ma, ea),
                                        data = data[data$Index ==
firstScan, ],
                                        start = list(fl =  AngleToq(3.5),
hl = 165, el = elInitial, ml = 0.6, b = 14,
                                                     ha = 0.01, ea =
eaInitial, fa = 20, ma = 0.7),
                                        upper = list(fl = AngleToq(5), hl
= Inf, el = elInitial+0.1, ml = 1, b = Inf,
                                                     ha = Inf, ea =
eaInitial+0.1, fa = 20, ma = 1),
                                        lower = list(fl = AngleToq(3), hl
= 1, el = elInitial-0.3, ml = 0.4, b = 0,
                                                     ha = 0, ea =
eaInitial-0.1, fa = 10, ma = 0.6),
                                        algorithm = "port")} else
                                        {
                                          first <- nls(CPS ~ b + lPeak(q,
hl, fl, ml, el),
                                                       data =
data[data$Index == firstScan, ],
                                                       start = list(fl =
2, hl = 165, el = elInitial, ml = 0.6, b = 14),
```

```
                                                                upper = list(fl =
4, hl = Inf, el = elInitial+0.1, ml = 1, b = Inf),
                                                                lower = list(fl =
2, hl = 1, el = elInitial-0.3, ml = 0.4, b = 0),
                                                                algorithm =
"port")}

plotall(firstScan, first, subset(data, Index == firstScan),
coef(first)['b'], coef(first)['el'])#Check first decomposition
#Fix background and position of the liquid peak
b <- coef(first)['b']
el <- coef(first)['el']
start <- coef(first)

####Decompose liquid scans####
if(alphaMin>firstScan) {
  fitLiquid <- function(i) {return(tryCatch(
    nls(CPS ~ b + lPeak(q, hl, fl, ml, el),
        data = data[data$Index == i, ],
        start = list(fl = start['fl'], hl = start['hl'], ml = start['ml'],
el = el, b = b
        ),
        upper = list(fl = 4, hl = Inf, ml = 1, el = el +0.1, b = b +20),
        lower = list(fl = 2, hl = 1, ml = 0.4, el = el -0.1, b = b -20),
        algorithm = "port"), error=function(e) NA))}
  LiquidFits <- list(lapply(lScans, fitLiquid))
  LiquidEstimates <- as.data.frame(cbind(Index = lScans,
rbind(t(as.data.frame(lapply(c(1:length(lScans)), fitsList = LiquidFits,
reportEstimates))))))
  LiquidEstimates <- cbind(LiquidEstimates, SS =
rbind(t(as.data.frame(lapply(c(1:length(lScans)), fitsList = LiquidFits,
reportSS)))))
  which(is.na(LiquidFits[[1]]))#Scans missing?
  View(LiquidEstimates)
  plotLiquid <- function(i) plotall(i, LiquidFits[[1]][[i-min(lScans)+1]],
subset(data, Index ==i), b, el)
  lapply(lScans, plotLiquid)
  b <- mean(LiquidEstimates$b)
  el <- mean(LiquidEstimates$el)
}
####Decompose alpha scans####
firstAlpha <- nls(CPS ~ b + lPeak(q, hl, fl, ml, el) + aPeak(q, fl, ha, fa,
ma, ea),
                  data = data[data$Index == alphaMin, ],
                  start = list(fl =  AngleToq(3.5), hl = 165, ml = 0.6,
                               ha = 0.01, ea = eaInitial, fa = 20, ma =
0.7),
                  upper = list(fl = AngleToq(5), hl = Inf, ml = 1,
                               ha = Inf, ea = eaInitial+0.1, fa = 20, ma =
1),
                  lower = list(fl = AngleToq(3), hl = 1, ml = 0.4,
                               ha = 0, ea = eaInitial-0.1, fa = 10, ma =
0.6),
                  algorithm = "port")
CoefFirstA <- coef(firstAlpha)
if(firstScan< alphaMin) {start <- c(coef(first), CoefFirstA['ha'],
CoefFirstA['ea'], CoefFirstA['fa'], CoefFirstA['ma'])}

fitAlpha <- function(i) {return(tryCatch(
  nls(CPS ~ b + lPeak(q, hl, fl, ml, el) + aPeak(q, fl, ha, fa, ma, ea),
      data = data[data$Index == i, ],
```

```
      start = list(fl = start['fl'], hl = start['hl'], ml = start['ml'],
                   ha = start['ha'], ea = start['ea'], fa = start['fa'], ma
= start['ma']),
      upper = list(fl = 4, hl = Inf, ml = 1,
                   ha = Inf, ea = coef(firstAlpha)['ea']+0.03, fa = 20, ma
= 1),
      lower = list(fl = 2, hl = 1, ml = 0.4,
                   ha = 0, ea = coef(firstAlpha)['ea']-0.03, fa = 10, ma =
0.6),
      algorithm = "port"), error=function(e) NA))}
AlphaFits <- list(lapply(aScans, fitAlpha))
AlphaEstimates <- as.data.frame(cbind(Index = aScans,
rbind(t(as.data.frame(lapply(c(1:length(aScans)), fitsList = AlphaFits,
reportEstimates))))))
AlphaEstimates <- cbind(AlphaEstimates, SS =
rbind(t(as.data.frame(lapply(c(1:length(aScans)), fitsList = AlphaFits,
reportSS)))))

#Check for unsolved
which(is.na(AlphaFits[[1]]))#Scans missing? If so, run the chunck below and
adapt when needed
missingScanList <- which(is.na(AlphaFits[[1]]))

j <- 1 #Change to start with values of scans around the one not decomposed
start <- coef(AlphaFits[[1]][[missingScanList[1]+j]])
decomposemissing <- function(missingScan) {fitAlpha(aScans[missingScan])}
i <- 1
repeat {
  AlphaFits[[1]][[missingScanList[i]]] <-
decomposemissing(missingScanList[i])
  i <- i +1
  if (i == length(missingScanList)+1) {break}
}
which(is.na(AlphaFits[[1]]))#Scans missing? If so, run the chunck below and
adapt when needed

AlphaEstimates <- as.data.frame(cbind(Index = aScans,
rbind(t(as.data.frame(lapply(c(1:length(aScans)), fitsList = AlphaFits,
reportEstimates))))))
AlphaEstimates <- cbind(AlphaEstimates, SS =
rbind(t(as.data.frame(lapply(c(1:length(aScans)), fitsList = AlphaFits,
reportSS)))))

###Check plots and estimates
View(AlphaEstimates)
plotAlpha <- function(i) plotall(i, AlphaFits[[1]][[i-min(aScans)+1]],
subset(data, Index ==i), b, el)
#lapply(AlphaScans, plotAlpha)
plotAlpha(min(aScans))
plotAlpha(min(aScans)+5)
plotAlpha(max(aScans)) #Check if there is beta or beta p signs in this one
plotAlpha(max(aScans)-1)
plotAlpha(max(aScans)-2)
####Decompose the last scan####
##Adjust depending on the peaks present
if (lastScan > betaMin & alphaMin > firstScan) {last = nls(CPS ~ b +
lPeak(q, hl, fl, ml, el) +

bp2Peak(q, fl, hbp2, fbp2, mbp2, ebp2) +

bp1Peak(q, fl, hbp2, hbp1, fbp1, mbp1, ebp1) +
```

```
bp3Peak(q, fl, hbp2, hbp3, fbp3, mbp3, ebp3) +

bp4Peak(q, fl, hbp2, hbp4, fbp4, mbp4, ebp4) +

bPeak(q, fl, hb, fb, mb, eb)

, data = data[data$Index == lastScan, ],

start = list(fl = mean(LiquidEstimates$fl), hl = mean(LiquidEstimates$hl),

ml = mean(LiquidEstimates$ml)

, hbp2 = 43, ebp2 = ebp2Initial, fbp2 = 8, mbp2 = 100

, hbp1 = 1.3, ebp1 = ebp1Initial, fbp1 = 8, mbp1 = 100

, hbp3 = 1.7, ebp3 = ebp3Initial, fbp3 = 8, mbp3 = 100

, hbp4 = 0.7, ebp4 = ebp4Initial, fbp4 = 15, mbp4 = 5

, hb = 1, eb = ebInitial, fb = 20, mb = 1

), upper = list(fl = 4, hl = Inf, ml = 1

, hbp2 = Inf, ebp2 = ebp2Initial+0.1, fbp2 = 40, mbp2 = 100

, hbp1 = 1.9, ebp1 = ebp1Initial+0.1, fbp1 = 40, mbp1 = 100

, hbp3 = 1.9, ebp3 = ebp3Initial+0.1, fbp3 = 40, mbp3 = 100

, hbp4 = 0.9, ebp4 = ebp4Initial+0.1, fbp4 = 40, mbp4 = 100

, hb = Inf, eb = ebInitial+0.1, fb = 30, mb = 1

), lower = list(fl = 2, hl = 1, ml = 0.4

, hbp2 = 0.1, ebp2 = ebp2Initial-0.2, fbp2 = 8, mbp2 = 90

, hbp1 = 1.2, ebp1 = ebp1Initial-0.2, fbp1 = 8, mbp1 = 90

, hbp3 = 1.2, ebp3 = ebp3Initial-0.2, fbp3 = 8, mbp3 = 90

, hbp4 = 0.5, ebp4 = ebp4Initial-0.2, fbp4 = 15, mbp4 = 1

, hb = 0.1, eb = ebInitial-0.2, fb = 8, mb = 0.7

),

algorithm = "port")}
if (lastScan > betaMin & alphaMin == firstScan) {last = nls(CPS ~ b +
lPeak(q, hl, fl, ml, el) +

bp2Peak(q, fl, hbp2, fbp2, mbp2, ebp2) +

bp1Peak(q, fl, hbp2, hbp1, fbp1, mbp1, ebp1) +

bp3Peak(q, fl, hbp2, hbp3, fbp3, mbp3, ebp3) +

bp4Peak(q, fl, hbp2, hbp4, fbp4, mbp4, ebp4) +
```

```
bPeak(q, fl, hb, fb, mb, eb)

, data = data[data$Index == lastScan, ],

start = list(fl = mean(AlphaEstimates$fl), hl = mean(AlphaEstimates$hl),

ml = mean(AlphaEstimates$ml)

, hbp2 = 43, ebp2 = ebp2Initial, fbp2 = 8, mbp2 = 100

, hbp1 = 1.3, ebp1 = ebp1Initial, fbp1 = 8, mbp1 = 100

, hbp3 = 1.7, ebp3 = ebp3Initial, fbp3 = 8, mbp3 = 100

, hbp4 = 0.7, ebp4 = ebp4Initial, fbp4 = 15, mbp4 = 5

, hb = 1, eb = ebInitial, fb = 20, mb = 1

), upper = list(fl = 4, hl = Inf, ml = 1

, hbp2 = Inf, ebp2 = ebp2Initial+0.1, fbp2 = 40, mbp2 = 100

, hbp1 = 1.9, ebp1 = ebp1Initial+0.1, fbp1 = 40, mbp1 = 100

, hbp3 = 1.9, ebp3 = ebp3Initial+0.1, fbp3 = 40, mbp3 = 100

, hbp4 = 0.9, ebp4 = ebp4Initial+0.1, fbp4 = 40, mbp4 = 100

, hb = Inf, eb = ebInitial+0.1, fb = 30, mb = 1

), lower = list(fl = 2, hl = 1, ml = 0.4

, hbp2 = 0.1, ebp2 = ebp2Initial-0.2, fbp2 = 8, mbp2 = 90

, hbp1 = 1.2, ebp1 = ebp1Initial-0.2, fbp1 = 8, mbp1 = 90

, hbp3 = 1.2, ebp3 = ebp3Initial-0.2, fbp3 = 8, mbp3 = 90

, hbp4 = 0.5, ebp4 = ebp4Initial-0.2, fbp4 = 15, mbp4 = 1

, hb = 0.1, eb = ebInitial-0.2, fb = 8, mb = 0.7

),

algorithm = "port")}
plotall(lastScan, last, subset(data, Index == lastScan), b, el)

####Decompose bp scans (no beta)####
start <- coef(last)

fitbp <- function(i) {return(tryCatch(
  nls(CPS ~ b + lPeak(q, hl, fl, ml, el) +
        bp2Peak(q, fl, hbp2, fbp2, mbp2, ebp2) +
        bp1Peak(q, fl, hbp2, hbp1, fbp1, mbp1, ebp1) +
        bp3Peak(q, fl, hbp2, hbp3, fbp3, mbp3, ebp3) +
        bp4Peak(q, fl, hbp2, hbp4, fbp4, mbp4, ebp4)
      , data = data[data$Index == i, ],
      start = list(fl = start["fl"], hl = start["hl"], ml = start["ml"]
                   , hbp2 = start["hbp2"], ebp2 = start["ebp2"], fbp2 =
start["fbp2"], mbp2 = start["mbp2"]
```

```
                    , hbp1 = start["hbp1"], ebp1 = start["ebp1"], fbp1 =
start["fbp1"], mbp1 = start["mbp1"]
                    , hbp3 = start["hbp3"], ebp3 = start["ebp3"], fbp3 =
start["fbp3"], mbp3 = start["mbp3"]
                    , hbp4 = start["hbp4"], ebp4 = start["ebp4"], fbp4 =
start["fbp4"], mbp4 = start["mbp4"]
      ), upper = list(fl = 4, hl = Inf, ml = 1
                      , hbp2 = Inf, ebp2 = coef(last)["ebp2"]+0.03, fbp2 =
40, mbp2 = 100
                      , hbp1 = 1.9, ebp1 = coef(last)["ebp1"]+0.03, fbp1 =
40, mbp1 = 100
                      , hbp3 = 1.9, ebp3 = coef(last)["ebp3"]+0.03, fbp3 =
40, mbp3 = 100
                      , hbp4 = 0.9, ebp4 = coef(last)["ebp4"]+0.03, fbp4 =
40, mbp4 = 100
      ), lower = list(fl = 2, hl = 1, ml = 0.4
                      , hbp2 = 0.1, ebp2 = coef(last)["ebp2"]-0.04, fbp2 =
8, mbp2 = 90
                      , hbp1 = 1.2, ebp1 = coef(last)["ebp1"]-0.03, fbp1 =
8, mbp1 = 90
                      , hbp3 = 1.2, ebp3 = coef(last)["ebp3"]-0.03, fbp3 =
8, mbp3 = 90
                      , hbp4 = 0.5, ebp4 = coef(last)["ebp4"]-0.03, fbp4 =
15, mbp4 = 1
      ),
      algorithm = "port"), error=function(e) NA))}

bpFits <- list(lapply(bpScans, fitbp))
bpEstimates <- as.data.frame(cbind(Index = bpScans,
rbind(t(as.data.frame(lapply(c(1:length(bpScans)), fitsList = bpFits,
reportEstimates))))))
bpEstimates <- cbind(bpEstimates, SS =
rbind(t(as.data.frame(lapply(c(1:length(bpScans)), fitsList = bpFits,
reportSS)))))

#Check for unsolved
which(is.na(bpFits[[1]]))#Scans missing? If so, run the chunck below and
adapt when needed
missingScanList <- which(is.na(bpFits[[1]]))
decomposemissing <- function(missingScan) {fitbp(bpScans[missingScan])}

j <- 1 #Change to start with values of scans around the one not decomposed
start <- coef(bpFits[[1]][[missingScanList[1]+j]])
i <- 1
repeat {
  bpFits[[1]][[missingScanList[i]]] <- decomposemissing(missingScanList[i])
  i <- i +1
  if (i == length(missingScanList)+1) {break}
}

which(is.na(bpFits[[1]]))#Scans missing? Repeat changing j if needed

bpEstimates <- as.data.frame(cbind(Index = bpScans,
rbind(t(as.data.frame(lapply(c(1:length(bpScans)), fitsList = bpFits,
reportEstimates))))))
bpEstimates <- cbind(bpEstimates, SS =
rbind(t(as.data.frame(lapply(c(1:length(bpScans)), fitsList = bpFits,
reportSS)))))
###Check plots and estimates
View(bpEstimates)
```

```
plotbp <- function(i) plotall(i, bpFits[[1]][[i-min(bpScans)+1]],
subset(data, Index ==i), b, el)
#lapply(bpScans, plotbp)
plotbp(min(bpScans))
plotbp(min(bpScans)+5)
plotbp(max(bpScans))

####Decompose beta prime  and beta scans####
start <- coef(last)
fitbpandb <- function(i) {return(tryCatch(
  nls(CPS ~ b + lPeak(q, hl, fl, ml, el) +
        bp2Peak(q, fl, hbp2, fbp2, mbp2, ebp2) +
        bp1Peak(q, fl, hbp2, hbp1, fbp1, mbp1, ebp1) +
        bp3Peak(q, fl, hbp2, hbp3, fbp3, mbp3, ebp3) +
        bp4Peak(q, fl, hbp2, hbp4, fbp4, mbp4, ebp4) +
        bPeak(q, fl, hb, fb, mb, eb)
      , data = data[data$Index == i, ],
      start = list(fl = start["fl"], hl = start["hl"], ml = start["ml"]
                   , hbp2 = start["hbp2"], ebp2 = start["ebp2"], fbp2 =
start["fbp2"], mbp2 = start["mbp2"]
                   , hbp1 = start["hbp1"], ebp1 = start["ebp1"], fbp1 =
start["fbp1"], mbp1 = start["mbp1"]
                   , hbp3 = start["hbp3"], ebp3 = start["ebp3"], fbp3 =
start["fbp3"], mbp3 = start["mbp3"]
                   , hbp4 = start["hbp4"], ebp4 = start["ebp4"], fbp4 =
start["fbp4"], mbp4 = start["mbp4"]
                   , hb = start["hb"], eb = start["eb"], fb = start["fb"],
mb = start["mb"]
      ), upper = list(fl = 4, hl = Inf, ml = 1
                      , hbp2 = Inf, ebp2 = coef(last)["ebp2"]+0.03, fbp2 =
40, mbp2 = 100
                      , hbp1 = 1.9, ebp1 = coef(last)["ebp1"]+0.03, fbp1 =
40, mbp1 = 100
                      , hbp3 = 1.9, ebp3 = coef(last)["ebp3"]+0.03, fbp3 =
40, mbp3 = 100
                      , hbp4 = 0.9, ebp4 = coef(last)["ebp4"]+0.03, fbp4 =
40, mbp4 = 100
                      , hb = Inf, eb = coef(last)["eb"]+0.03, fb = 40, mb =
1
      ), lower = list(fl = 2, hl = 1, ml = 0.4
                      , hbp2 = 0.1, ebp2 = coef(last)["ebp2"]-0.04, fbp2 =
8, mbp2 = 90
                      , hbp1 = 1.2, ebp1 = coef(last)["ebp1"]-0.03, fbp1 =
8, mbp1 = 90
                      , hbp3 = 1.2, ebp3 = coef(last)["ebp3"]-0.03, fbp3 =
8, mbp3 = 90
                      , hbp4 = 0.5, ebp4 = coef(last)["ebp4"]-0.03, fbp4 =
15, mbp4 = 1
                      , hb = 0.1, eb = coef(last)["eb"]-0.03, fb = 8, mb =
0.7
      ),
      algorithm = "port"), error=function(e) NA))}

bpandbFits <- list(lapply(bpandbScans, fitbpandb))
bpandbEstimates <- as.data.frame(cbind(Index = bpandbScans,
rbind(t(as.data.frame(lapply(c(1:length(bpandbScans)), fitsList =
bpandbFits, reportEstimates))))))
bpandbEstimates <- cbind(bpandbEstimates, SS =
rbind(t(as.data.frame(lapply(c(1:length(bpandbScans)), fitsList =
bpandbFits, reportSS)))))
```

```
#Check for unsolved
which(is.na(bpandbFits[[1]]))#Scans missing? If so, run the chunck below
and adapt when needed
missingScanList <- which(is.na(bpandbFits[[1]]))
decomposemissing <- function(missingScan)
{fitbpandb(bpandbScans[missingScan])}

j <- 3 #Change to start with values of scans around the one not decomposed
start <- coef(bpandbFits[[1]][[missingScanList[1]+j]])
i <- 1
repeat {
  bpandbFits[[1]][[missingScanList[i]]] <-
decomposemissing(missingScanList[i])
  i <- i +1
  if (i == length(missingScanList)+1) {break}
}

which(is.na(bpandbFits[[1]]))#Scans missing? Repeat changing j if needed
missingScanList <- which(is.na(bpandbFits[[1]]))

j <- -2 #Change to start with values of scans around the one not decomposed
start <- coef(bpandbFits[[1]][[missingScanList[1]+j]])
i <- 1
repeat {
  bpandbFits[[1]][[missingScanList[i]]] <-
decomposemissing(missingScanList[i])
  i <- i +1
  if (i == length(missingScanList)+1) {break}
}
which(is.na(bpandbFits[[1]]))#Scans missing? Repeat changing j if needed

bpandbEstimates <- as.data.frame(cbind(Index = bpandbScans,
rbind(t(as.data.frame(lapply(c(1:length(bpandbScans)), fitsList =
bpandbFits, reportEstimates))))))
bpandbEstimates <- cbind(bpandbEstimates, SS =
rbind(t(as.data.frame(lapply(c(1:length(bpandbScans)), fitsList =
bpandbFits, reportSS)))))
###Check plots and estimates
View(bpandbEstimates)
plotbpandb <- function(i) plotall(i, bpandbFits[[1]][[i-
min(bpandbScans)+1]], subset(data, Index ==i), b, el)
#lapply(bpandbScans, plotbpandb)
plotbpandb(min(bpandbScans))
plotbpandb(min(bpandbScans)+1)
plotbpandb(min(bpandbScans)+2)
plotbpandb(min(bpandbScans)+3)
plotbpandb(min(bpandbScans)+4)
plotbpandb(min(bpandbScans)+5)
plotbpandb(max(bpandbScans))
plotbpandb(max(bpandbScans)-5)

####Prepare to decompose scans from the transition period using parameter-
reduced models####
##Calculate fixed parameters for alpha, beta and beta' peaks
ma <- mean(AlphaEstimates$ma, na.rm = TRUE)
ea <- mean(AlphaEstimates$ea, na.rm = TRUE)

mbp2 <- mean(bpandbEstimates$mbp2, na.rm = TRUE)
mbp1 <- mean(bpandbEstimates$mbp1, na.rm = TRUE)
mbp3 <- mean(bpandbEstimates$mbp3, na.rm = TRUE)
mbp4 <- mean(bpandbEstimates$mbp4, na.rm = TRUE)
```

```
ebp2 <- mean(bpandbEstimates$ebp2, na.rm = TRUE)
ebp1 <- mean(bpandbEstimates$ebp1, na.rm = TRUE)
ebp3 <- mean(bpandbEstimates$ebp3, na.rm = TRUE)
ebp4 <- mean(bpandbEstimates$ebp4, na.rm = TRUE)
hbp1 <- mean(bpandbEstimates$hbp1, na.rm = TRUE)
hbp3 <- mean(bpandbEstimates$hbp3, na.rm = TRUE)
hbp4 <- mean(bpandbEstimates$hbp4, na.rm = TRUE)
mb <- mean(bpandbEstimates$mb, na.rm = TRUE)
eb <- mean(bpandbEstimates$eb, na.rm = TRUE)
mbp2 <- mean(bpEstimates$mbp2, na.rm = TRUE)
mbp1 <- mean(bpEstimates$mbp1, na.rm = TRUE)
mbp3 <- mean(bpEstimates$mbp3, na.rm = TRUE)
mbp4 <- mean(bpEstimates$mbp4, na.rm = TRUE)
ebp2 <- mean(bpEstimates$ebp2, na.rm = TRUE)
ebp1 <- mean(bpEstimates$ebp1, na.rm = TRUE)
ebp3 <- mean(bpEstimates$ebp3, na.rm = TRUE)
ebp4 <- mean(bpEstimates$ebp4, na.rm = TRUE)
hbp1 <- mean(bpEstimates$hbp1, na.rm = TRUE)
hbp3 <- mean(bpEstimates$hbp3, na.rm = TRUE)
hbp4 <- mean(bpEstimates$hbp4, na.rm = TRUE)


####Decompose alpha and bp scans####
if(betaMin > lastScan){start <-
c(coef(AlphaFits[[1]][[length(AlphaFits[[1]])]])["ha"],

coef(AlphaFits[[1]][[length(AlphaFits[[1]])]])["fa"],
                                    coef(bpFits[[1]][[1]]))}
if(betaMin < lastScan) {start <-
c(coef(AlphaFits[[1]][[length(AlphaFits[[1]])]])["ha"],

coef(AlphaFits[[1]][[length(AlphaFits[[1]])]])["fa"],
                                     coef(bpandbFits[[1]][[1]]))}


fitaANDbp <- function(i) {return(tryCatch(
  nls(CPS ~ b + lPeak(q, hl, fl, ml, el) +
        aPeak(q, fl, ha, fa, ma, ea) +
        bp2Peak(q, fl, hbp2, fbp2, mbp2, ebp2) +
        bp1Peak(q, fl, hbp2, hbp1, fbp1, mbp1, ebp1) +
        bp3Peak(q, fl, hbp2, hbp3, fbp3, mbp3, ebp3) +
        bp4Peak(q, fl, hbp2, hbp4, fbp4, mbp4, ebp4)
      , data = data[data$Index == i,],
      start = list(fl = start['fl'],
                   hl = start['hl'],
                   ml = start['ml'],
                   ha = start['ha'],
                   fa = start['fa'],
                   hbp2 = start['hbp2'], fbp2 = start['fbp2'],
                   fbp1 = start['fbp1'],
                   fbp3 = start['fbp3'],
                   fbp4 = start['fbp4']),
      upper = list(fl = 4, hl = Inf, ml = 1
                   , ha = Inf, fa = 20
                   , hbp2 = Inf, fbp2 = 40
                   , fbp1 = 40, fbp3 = 40, fbp4 = 40),
      lower = list(fl = 2, hl = 1, ml = 0.4
                   , ha = 0, fa = 10
                   , hbp2 = 0.1, fbp2 = 8
                   , fbp1 = 8, fbp3 = 8, fbp4 = 15),
      algorithm = "port")
```

```
  , error=function(e) NA))}

aANDbpFits <- list(lapply(aANDbpScans, fitaANDbp))
aANDbpEstimates <- as.data.frame(cbind(Index = aANDbpScans,
rbind(t(as.data.frame(lapply(c(1:length(aANDbpScans)), fitsList =
aANDbpFits, reportEstimates))))))
aANDbpEstimates <- cbind(aANDbpEstimates, SS =
rbind(t(as.data.frame(lapply(c(1:length(aANDbpScans)), fitsList =
aANDbpFits, reportSS)))))
which(is.na(aANDbpFits[[1]]))#To check if there are unsolved scans

missingScanList <- which(is.na(aANDbpFits[[1]]))
decomposemissing <- function(missingScan)
{fitaANDbp(aANDbpScans[missingScan])}

j <- 1 #Change to start with values of scans around the one not decomposed
start <- coef(aANDbpFits[[1]][[missingScanList[1]+j]])
i <- 1
repeat {
  aANDbpFits[[1]][[missingScanList[i]]] <-
decomposemissing(missingScanList[i])
  i <- i +1
  if (i == length(missingScanList)+1) {break}
}
which(is.na(aANDbpFits[[1]]))#To check if there are unsolved scans

missingScanList <- which(is.na(aANDbpFits[[1]]))
j <- 4 #Change to start with values of scans around the one not decomposed
start <- coef(aANDbpFits[[1]][[missingScanList[1]+j]])
i <- 1
repeat {
  aANDbpFits[[1]][[missingScanList[i]]] <-
decomposemissing(missingScanList[i])
  i <- i +1
  if (i == length(missingScanList)+1) {break}
}
which(is.na(aANDbpFits[[1]]))#To check if there are unsolved scans

aANDbpEstimates <- as.data.frame(cbind(Index = aANDbpScans,
rbind(t(as.data.frame(lapply(c(1:length(aANDbpScans)), fitsList =
aANDbpFits, reportEstimates))))))
aANDbpEstimates <- cbind(aANDbpEstimates, SS =
rbind(t(as.data.frame(lapply(c(1:length(aANDbpScans)), fitsList =
aANDbpFits, reportSS)))))
#Add fixed parameters to data frame
FixedParameters <- cbind(ea = ea, ma = ma, mbp2 = mbp2, ebp2 = ebp2,
                         hbp1 = hbp1, mbp1 = mbp1, ebp1 = ebp1,
                         hbp3 = hbp3, mbp3 = mbp3, ebp3 = ebp3,
                         hbp4 = hbp4, mbp4 = mbp4, ebp4 = ebp4)
aANDbpEstimates <- cbind(aANDbpEstimates, FixedParameters)

###Check plots and estimates
View(aANDbpEstimates)
plotaANDbp <- function(i) plotall(i, aANDbpFits[[1]][[i-
min(aANDbpScans)+1]], subset(data, Index ==i), b, el)
plotaANDbp(min(aANDbpScans)+5)
plotaANDbp(max(aANDbpScans))

####Decompose alpha, beta prime and beta scans####

start <- c(coef(AlphaFits[[1]][[length(AlphaFits[[1]])]])["ha"],
```

```
          coef(AlphaFits[[1]][[length(AlphaFits[[1]])]])["fa"],
          coef(bpandbFits[[1]][[1]]))
fitaANDbpANDb <- function(i) {  return(tryCatch(
  nls(CPS ~ b + lPeak(q, hl, fl, ml, el) +
        aPeak(q, fl, ha, fa, ma, ea) +
        bp2Peak(q, fl, hbp2, fbp2, mbp2, ebp2) +
        bp1Peak(q, fl, hbp2, hbp1, fbp1, mbp1, ebp1) +
        bp3Peak(q, fl, hbp2, hbp3, fbp3, mbp3, ebp3) +
        bp4Peak(q, fl, hbp2, hbp4, fbp4, mbp4, ebp4) +
        bPeak(q, fl, hb, fb, mb, eb)
      , data = data[data$Index == i,],
      start = list(fl = start['fl'],
                   hl = start['hl'],
                   ml = start['ml'],
                   ha = start['ha'],
                   fa = start['fa'],
                   hbp2 = start['hbp2'], fbp2 = start['fbp2'],
                   fbp1 = start['fbp1'],
                   fbp3 = start['fbp3'],
                   fbp4 = start['fbp4'],
                   hb = start['hb'], fb = start['fb']
      ),
      upper = list(fl = 4, hl = Inf, ml = 1
                   , ha = Inf, fa = 20
                   , hbp2 = Inf, fbp2 = 40
                   , fbp1 = 40, fbp3 = 40, fbp4 = 40
                   , hb = Inf, fb = 40
      ), lower = list(fl = 2, hl = 1, ml = 0.4
                      , ha = 0, fa = 10
                      , hbp2 = 0.1, fbp2 = 8
                      , fbp1 = 8, fbp3 = 8, fbp4 = 15
                      , hb = 0.1, fb = 10
      ), algorithm = "port")
  , error=function(e) NA))}

aANDbpANDbFits <- list(lapply(aANDbpANDbScans, fitaANDbpANDb))
aANDbpANDbEstimates <- as.data.frame(cbind(Index = aANDbpANDbScans,
rbind(t(as.data.frame(lapply(c(1:length(aANDbpANDbScans)), fitsList =
aANDbpANDbFits, reportEstimates))))))
aANDbpANDbEstimates <- cbind(aANDbpANDbEstimates, SS =
rbind(t(as.data.frame(lapply(c(1:length(aANDbpANDbScans)), fitsList =
aANDbpANDbFits, reportSS)))))
which(is.na(aANDbpANDbFits[[1]]))#To check if there are unsolved scans

missingScanList <- which(is.na(aANDbpANDbFits[[1]]))
decomposemissing <- function(missingScan)
{fitaANDbpANDb(aANDbpANDbScans[missingScan])}

j <- 1 #Change to start with values of scans around the one not decomposed
start <- coef(aANDbpANDbFits[[1]][[missingScanList[1]+j]])
i <- 1
repeat {
  aANDbpANDbFits[[1]][[missingScanList[i]]] <-
decomposemissing(missingScanList[i])
  i <- i +1
  if (i == length(missingScanList)+1) {break}
}
which(is.na(aANDbpANDbFits[[1]]))#To check if there are unsolved scans

missingScanList <- which(is.na(aANDbpANDbFits[[1]]))
```

```
j <- -1 #Change to start with values of scans around the one not decomposed
start <- coef(aANDbpANDbFits[[1]][[missingScanList[1]+j]])
i <- 1
repeat {
  aANDbpANDbFits[[1]][[missingScanList[i]]] <-
decomposemissing(missingScanList[i])
  i <- i +1
  if (i == length(missingScanList)+1) {break}
}

which(is.na(aANDbpANDbFits[[1]]))#To check if there are unsolved scans

aANDbpANDbEstimates <- as.data.frame(cbind(Index = aANDbpANDbScans,
rbind(t(as.data.frame(lapply(c(1:length(aANDbpANDbScans)), fitsList =
aANDbpANDbFits, reportEstimates))))))
aANDbpANDbEstimates <- cbind(aANDbpANDbEstimates, SS =
rbind(t(as.data.frame(lapply(c(1:length(aANDbpANDbScans)), fitsList =
aANDbpANDbFits, reportSS)))))
#Add fixed aprameters to data frame
FixedParameters <- cbind(ea = ea, ma = ma, mbp2 = mbp2, ebp2 = ebp2,
                         hbp1 = hbp1, mbp1 = mbp1, ebp1 = ebp1,
                         hbp3 = hbp3, mbp3 = mbp3, ebp3 = ebp3,
                         hbp4 = hbp4, mbp4 = mbp4, ebp4 = ebp4,
                         eb = eb, mb = mb)
aANDbpANDbEstimates <- cbind(aANDbpANDbEstimates, FixedParameters)

###Check plots and estimates
View(aANDbpANDbEstimates)
plotaANDbpANDb <- function(i) plotall(i, aANDbpANDbFits[[1]][[i-
min(aANDbpANDbScans)+1]], subset(data, Index ==i), b, el)
plotaANDbpANDb(min(aANDbpANDbScans)+5)
plotaANDbpANDb(max(aANDbpANDbScans))

####Merge all estimates and calculate areas####
##Add more options when needed
if(alphaMin > lastScan & betaMin < lastScan) {allFits <- c(LiquidFits[[1]],
bpandbFits[[1]])
Estimates <- bind_rows(LiquidEstimates, bpandbEstimates)
}#If there is a liquid to betap and beta transition
if(betaMin > lastScan){allFits <- c(AlphaFits[[1]], aANDbpFits[[1]],
bpFits[[1]])
Estimates <- bind_rows(AlphaEstimates, aANDbpEstimates, bpEstimates)
} #If there is an alpha to betap transition
if(betaMin < lastScan & alphaMin > firstScan & alphaMin < lastScan){
  allFits <- c(LiquidFits[[1]], AlphaFits[[1]],aANDbpANDbFits[[1]],
bpandbFits[[1]])
  Estimates <- bind_rows(LiquidEstimates, AlphaEstimates,
aANDbpANDbEstimates, bpandbEstimates)
} #if there are liquid to alpha and alpha to beta' and beta transitions
if(betaMin < lastScan & alphaMin == firstScan & alphaMin < lastScan){
  allFits <- c(AlphaFits[[1]],aANDbpANDbFits[[1]], bpandbFits[[1]])
  Estimates <- bind_rows(AlphaEstimates, aANDbpANDbEstimates,
bpandbEstimates)
}#if there is an alpha to beta' and beta transition
if(betaMin < alphaMax & betaMin > betapMin & alphaMin == firstScan){
  allFits <- c(LiquidFits[[1]], AlphaFits[[1]], aANDbpFits[[1]],
aANDbpANDbFits[[1]], bpandbFits[[1]])
  Estimates <- bind_rows(LiquidEstimates, AlphaEstimates, aANDbpEstimates,
aANDbpANDbEstimates, bpandbEstimates)
} #If there are liquid to alpha and then alpha to betap transition and
alpha to betap and beta transition.
```

```
if(betaMin < alphaMax & betaMin > betapMin & alphaMin == firstScan){
  allFits <- c(AlphaFits[[1]], aANDbpFits[[1]], aANDbpANDbFits[[1]],
bpandbFits[[1]])
  Estimates <- bind_rows(AlphaEstimates, aANDbpEstimates,
aANDbpANDbEstimates, bpandbEstimates)
} #If there is alpha to betap transition and alpha to betap and beta
transition.


Estimates <- arrange(Estimates, Index)
Estimates$b <- b
Estimates$el <- el
View(Estimates)

#####Calculate areas
lPeak <- function(q) subset(Estimates, Index == i)$hl*subset(Estimates,
Index == i)$fl^(2*subset(Estimates, Index == i)$ml) /
  (subset(Estimates, Index == i)$fl^2 + (2^(1/subset(Estimates, Index ==
i)$ml)-1)*(q - subset(Estimates, Index == i)$el)^2)^subset(Estimates, Index
== i)$ml
aPeak <- function(q) subset(Estimates, Index == i)$ha*(subset(Estimates,
Index == i)$fl/subset(Estimates, Index == i)$fa)^(2*subset(Estimates, Index
== i)$ma) /
  ((subset(Estimates, Index == i)$fl/subset(Estimates, Index == i)$fa)^2 +
(2^(1/subset(Estimates, Index == i)$ma)-1)*(q - subset(Estimates, Index ==
i)$ea)^2)^subset(Estimates, Index == i)$ma
BetaP2peak <- function(q) subset(Estimates, Index ==
i)$hbp2*(subset(Estimates, Index == i)$fl/subset(Estimates, Index ==
i)$fbp2)^(2*subset(Estimates, Index == i)$mbp2) /
  ((subset(Estimates, Index == i)$fl/subset(Estimates, Index == i)$fbp2)^2
+ (2^(1/subset(Estimates, Index == i)$mbp2)-1)*(q - subset(Estimates, Index
== i)$ebp2)^2)^subset(Estimates, Index == i)$mbp2
BetaP1peak <- function(q) (subset(Estimates, Index ==
i)$hbp2/subset(Estimates, Index == i)$hbp1)*(subset(Estimates, Index ==
i)$fl/subset(Estimates, Index == i)$fbp1)^(2*subset(Estimates, Index ==
i)$mbp1) /
  ((subset(Estimates, Index == i)$fl/subset(Estimates, Index == i)$fbp1)^2
+ (2^(1/subset(Estimates, Index == i)$mbp1)-1)*(q - subset(Estimates, Index
== i)$ebp1)^2)^subset(Estimates, Index == i)$mbp1
BetaP3peak <- function(q) (subset(Estimates, Index ==
i)$hbp2/subset(Estimates, Index == i)$hbp3)*(subset(Estimates, Index ==
i)$fl/subset(Estimates, Index == i)$fbp3)^(2*subset(Estimates, Index ==
i)$mbp3) /
  ((subset(Estimates, Index == i)$fl/subset(Estimates, Index == i)$fbp3)^2
+ (2^(1/subset(Estimates, Index == i)$mbp3)-1)*(q - subset(Estimates, Index
== i)$ebp3)^2)^subset(Estimates, Index == i)$mbp3
BetaP4peak <- function(q) (subset(Estimates, Index ==
i)$hbp2/subset(Estimates, Index == i)$hbp4)*(subset(Estimates, Index ==
i)$fl/subset(Estimates, Index == i)$fbp4)^(2*subset(Estimates, Index ==
i)$mbp4) /
  ((subset(Estimates, Index == i)$fl/subset(Estimates, Index == i)$fbp4)^2
+ (2^(1/subset(Estimates, Index == i)$mbp4)-1)*(q - subset(Estimates, Index
== i)$ebp4)^2)^subset(Estimates, Index == i)$mbp4
BetaPeak <- function(q) subset(Estimates, Index == i)$hb*(subset(Estimates,
Index == i)$fl/subset(Estimates, Index == i)$fb)^(2*subset(Estimates, Index
== i)$mb) /
  ((subset(Estimates, Index == i)$fl/subset(Estimates, Index == i)$fb)^2 +
(2^(1/subset(Estimates, Index == i)$mb)-1)*(q - subset(Estimates, Index ==
i)$eb)^2)^subset(Estimates, Index == i)$mb

qMin <- min(data$q)
```

```
qMax <- max(data$q)
Estimates[is.na(Estimates)] <- 0 #replace NA in areas with 0
View(Estimates)
###Try to fix this ugly chunck
#Create all possible columns
Estimates$Liquid <- 0
Estimates$Alpha <- 0
Estimates$BetaP1 <- 0
Estimates$BetaP2 <- 0
Estimates$BetaP3 <- 0
Estimates$BetaP4 <- 0
Estimates$Beta <- 0

##Calculate liquid areas
i <- firstScan
repeat {
  Estimates[Estimates$Index == i, 'Liquid'] <- integrate(lPeak, qMin, qMax,
stop.on.error = FALSE)$value
  i = i+1
  if (i == lastScan+1){
    break
  }
}

##Calculate alpha areas
i <- alphaMin
repeat {
  Estimates[Estimates$Index == i, 'Alpha'] <- integrate(aPeak, qMin,
qMax)$value
  i = i+1
  if (i == alphaMax+1){
    break
  }
}

##Calculate beta' areas
i <- lastScan
repeat {
  Estimates[Estimates$Index == i, 'BetaP4'] <- integrate(BetaP4peak, qMin,
qMax, stop.on.error = FALSE)$value
  Estimates[Estimates$Index == i, 'BetaP3'] <- integrate(BetaP3peak, qMin,
qMax, stop.on.error = FALSE)$value
  Estimates[Estimates$Index == i, 'BetaP2'] <- integrate(BetaP2peak, qMin,
qMax, stop.on.error = FALSE)$value
  Estimates[Estimates$Index == i, 'BetaP1'] <- integrate(BetaP1peak, qMin,
qMax, stop.on.error = FALSE)$value
  i = i-1
  if (i == betapMin-1){
    break
  }
}

####Calculate beta areas
i <- lastScan
repeat {
  Estimates[Estimates$Index == i, 'Beta'] <- integrate(BetaPeak, qMin,
qMax, stop.on.error = FALSE)$value
  i = i-1
  if (i == betaMin -1){
    break
  }
```

```
}

Estimates[is.na(Estimates)] <- 0 #replace NA in areas with 0
View(Estimates)


####Plot to check areas####
###Calculate area fractions and DoC###
Estimates$BetaP <- Estimates$BetaP1 + Estimates$BetaP2 + Estimates$BetaP3 +
Estimates$BetaP4
Estimates$TotalArea <- Estimates$BetaP + Estimates$Alpha + Estimates$Liquid
+ Estimates$Beta
Estimates$CrystallineArea <- Estimates$TotalArea - Estimates$Liquid
Estimates$AlphaFraction <- Estimates$Alpha/Estimates$TotalArea
Estimates$BetaPFraction <- Estimates$BetaP/Estimates$TotalArea
Estimates$BetaFraction <- Estimates$Beta/Estimates$TotalArea
Estimates$LiquidFraction <- Estimates$Liquid/Estimates$TotalArea
Estimates$DoC <- Estimates$CrystallineArea/Estimates$TotalArea
##Adjust the apperance and disappearance of the different polymorphs
according to this plot (e.g. alpha disappeared 1 scan earlier)

ggplot(data = Estimates) +
  geom_point(aes(x = Index, y = 100*AlphaFraction, color = 'Alpha')) +
  geom_point(aes(x = Index, y = 100*BetaPFraction, color = 'Beta prime')) +
  geom_point(aes(x = Index, y = 100*BetaFraction, color = 'Beta')) +
  geom_point(aes(x = Index, y = 100*LiquidFraction, color = 'Liquid')) +
  geom_point(aes(x = Index, y = 100*DoC, color = 'DoC')) +
  labs(x = "Index", y = 'Integrated intensity [%]') +
  theme_bw()
##Add Rwp values
getRwp <- function(i) {sqrt(sum(resid(allFits[[i-
firstScan+1]])^2/data[data$Index == i, ]$CPS)/sum(1/data[data$Index == i,
]$CPS))}
Estimates$Rwp <- unlist(lapply(c(firstScan:lastScan), getRwp))
```