Nonlinear Mixed Effect models

Philip Dixon

7/2/2020

Data and context from Fernando's nlraa package

live fuel moisture content, only for one species: S. bracteolactus

3 plots, data measured over time, 7 times

```
data(lfmc)
sapply(lfmc, class)
```

```
## leaf.type      time      plot      site      lfmc     group
##  "factor" "integer"  "factor"  "factor" "numeric"  "factor"
```

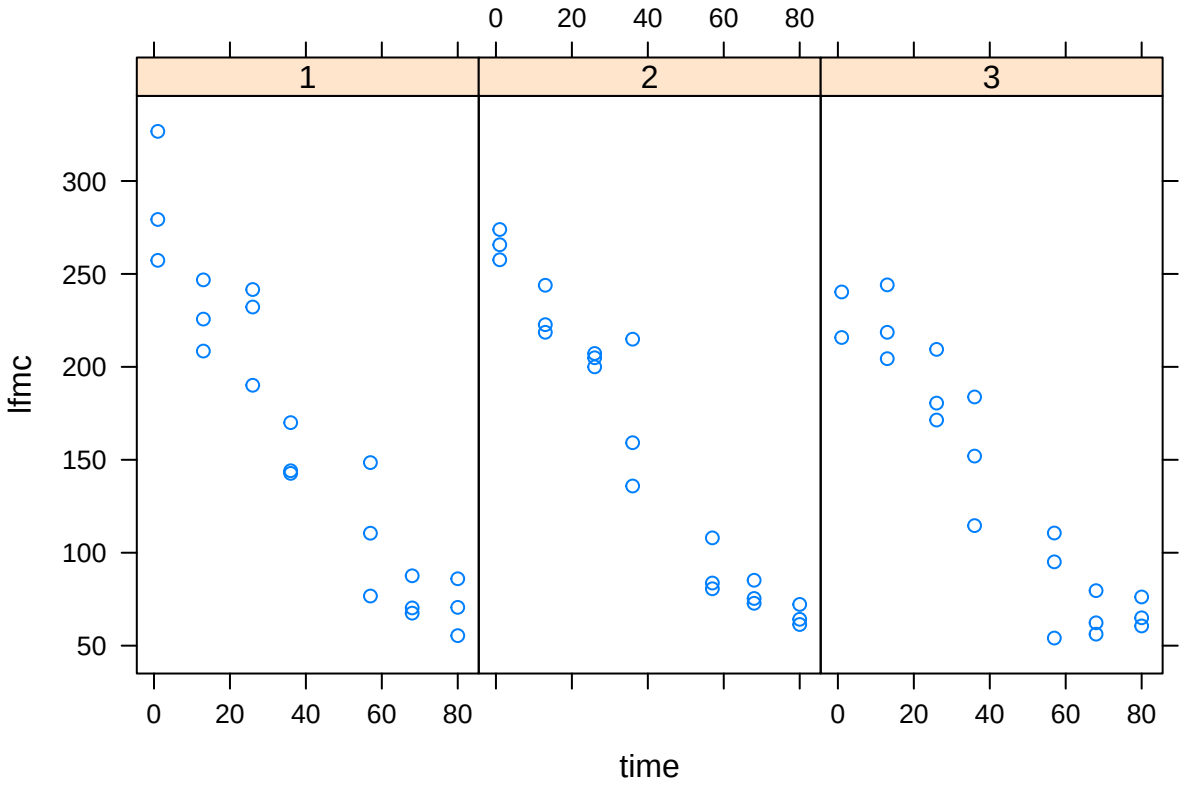
```
with(lfmc, table(site, plot))
```

```
##     plot
## site  4  5  6  1  2  3
##   P   0  0  0 62 62 60
##   SR 21 21 21  0  0  0
```

```
# consider only S. bracteolactus
sb <- lfmc %>% filter(leaf.type=="S. bracteolactus")
with(sb, table(plot, time))
```

```
##     time
## plot 1 13 26 36 57 68 80
##    4 0  0  0  0  0  0  0
##    5 0  0  0  0  0  0  0
##    6 0  0  0  0  0  0  0
##    1 3  3  3  3  3  3  3
##    2 3  3  3  3  3  3  3
##    3 2  3  3  3  3  3  3
```

```
xyplot(lfmc ~ time | plot, data=sb, layout=c(3,1))
```

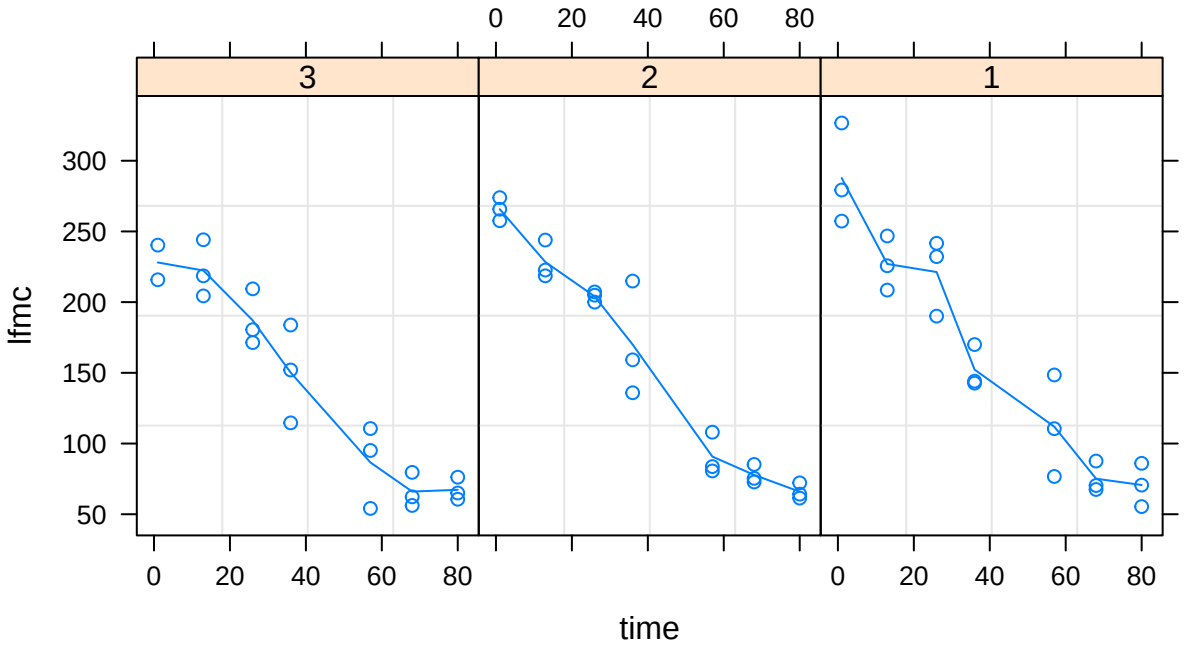


Analyses in the nlme package

fit fixed effects models to each plot

```
# first define the y, x, and grouping variables
#   group must be a factor variable (already set up)
#   can specify nested groups (B within A) by A/B
#     (more later)
#   result depends on whether X is continuous or factor
sbGrp <- groupedData(lfmc ~ time | plot, data=sb)

# and there is a nice lattice graph plot of data in each group
plot(sbGrp)
```



use self-start functions

SSxxxx functions are "self-start" nonlinear models, include common ones in base R and lots of agronomic ones in nlraa. Self-start functions provide information about the model, derivatives wrt parameters, and (usually) reasonable estimators of starting values. Fernando's nlraa vignette, vignette('nlraa') lists the SS functions in nlme and nlraa.

If you frequently use a function not on either list, it is worth writing your own self-start function. If a one-off, can just specify the model and provide starting values.

nlsList() fits the model each group of data

Have the usual R helper functions. Plus I show one way to graph data and predictions for each group.

```
sb.plot <- nlsList(lfmc ~ SSdlf(time, upper, lower, mid, scale), data=sbGrp)
sb.plot
```

```
## Call:
##   Model: lfmc ~ SSdlf(time, upper, lower, mid, scale) | plot
##    Data: sbGrp
##
## Coefficients:
##      upper    lower      mid     scale
## 3 236.8476 62.36245 36.17923 -10.58574
## 2 281.3863 52.34283 35.47981 -15.21243
## 1 393.2827 31.65032 22.06641 -25.82920
##
## Degrees of freedom: 62 total; 50 residual
```

```
## Residual standard error: 21.68724
```

```
summary(sb.plot)
```

```
## Call:
##   Model: lfmc ~ SSdlf(time, upper, lower, mid, scale) | plot
##    Data: sbGrp
##
## Coefficients:
##    upper
##   Estimate Std. Error   t value     Pr(>|t|)
## 3 236.8476   19.75637 11.988416 7.805525e-10
## 2 281.3863   32.89500  8.554076 5.512009e-09
## 1 393.2827  188.84630  2.082555 1.053693e-01
##    lower
##   Estimate Std. Error   t value     Pr(>|t|)
## 3 62.36245   14.11960 4.4167288 0.0002288407
## 2 52.34283   24.51988 2.1347103 0.0159537837
## 1 31.65032   61.91918 0.5111553 0.6798732077
##    mid
##   Estimate Std. Error  t value     Pr(>|t|)
## 3 36.17923   3.774491 9.585195 1.935037e-08
## 2 35.47981   4.852805 7.311197 5.469788e-08
## 1 22.06641  21.534612 1.024695 4.116933e-01
##    scale
##    Estimate Std. Error   t value    Pr(>|t|)
## 3 -10.58574   4.337559 -2.440484 0.018920648
## 2 -15.21243   6.271545 -2.425627 0.007384783
## 1 -25.82920  19.433768 -1.329088 0.290235558
##
## Residual standard error: 21.68724 on 50 degrees of freedom
```

```
# wald confidence intervals
intervals(sb.plot)
```

```
## , , upper
##
##       lower     est.    upper
## 3 197.16580 236.8476 276.5295
## 2 215.31478 281.3863 347.4579
## 1  13.97378 393.2827 772.5917
##
## , , lower
##
##        lower     est.    upper
## 3  34.002394 62.36245  90.7225
## 2   3.093211 52.34283 101.5925
## 1 -92.718025 31.65032 156.0187
##
## , , mid
##
##       lower     est.    upper
## 3  28.59794 36.17923 43.76052
## 2  25.73267 35.47981 45.22696
## 1 -21.18713 22.06641 65.31995
```

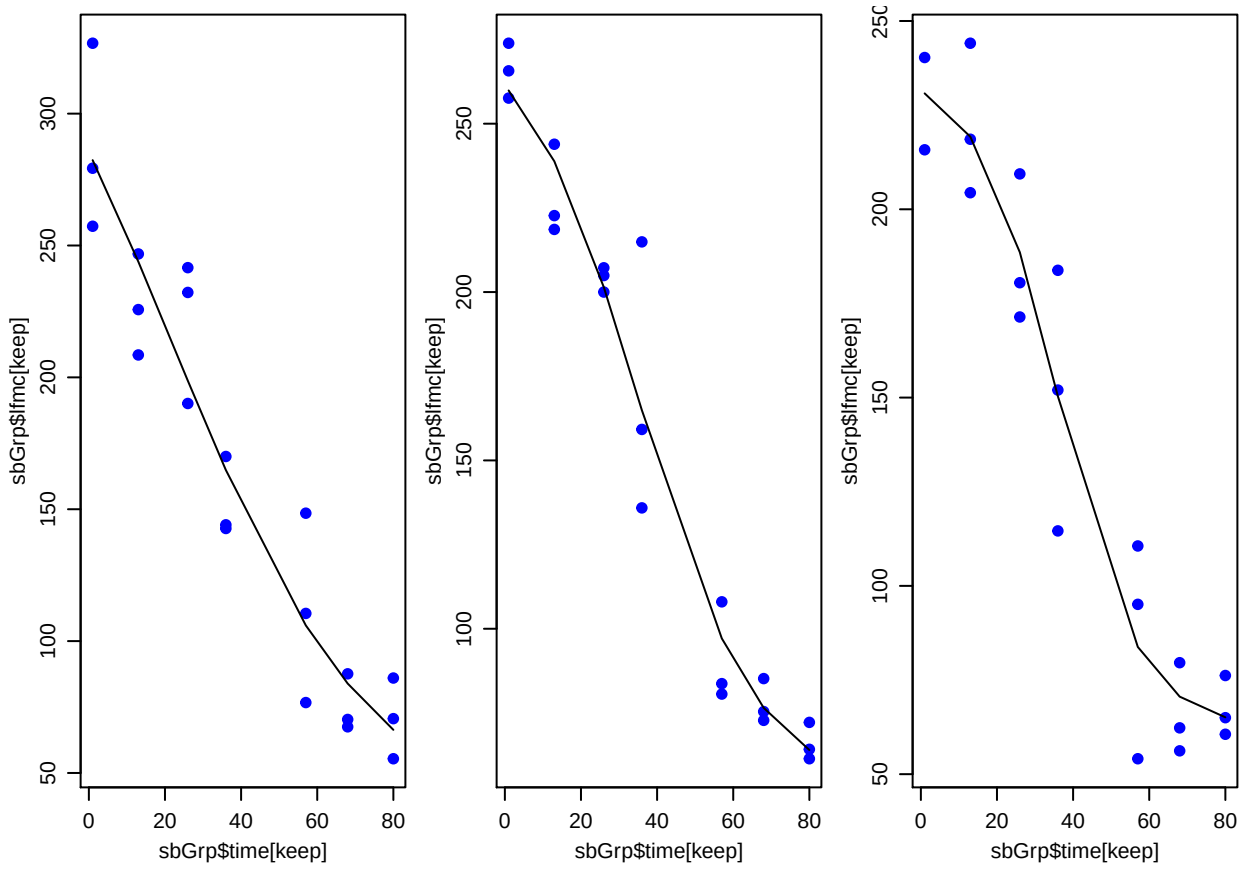
```
##
## , , scale
##
##       lower      est.     upper
## 3 -19.29799 -10.58574 -1.873501
## 2 -27.80920 -15.21243 -2.615657
## 1 -64.86307 -25.82920 13.204675
```

```
# profile likelihood intervals, but doesn't converge here
confint(sb.plot)
```

```
## Waiting for profiling to be done...
## Waiting for profiling to be done...
## Waiting for profiling to be done...
```

```
## $`3`
## [1] NA NA
## attr(,"errMsg")
## [1] "number of iterations exceeded maximum of 50"
##
## $`2`
## [1] NA NA
## attr(,"errMsg")
## [1] "number of iterations exceeded maximum of 50"
##
## $`1`
## [1] NA NA
## attr(,"errMsg")
## [1] "step factor 0.000488281 reduced below 'minFactor' of 0.000976562"
```

```
par(mfrow=c(1,3), mar=c(3,3,0,0)+0.3, mgp=c(2,0.8,0))
sb.pred1 <- predict(sb.plot)
for (i in unique(sbGrp$plot)) {
  keep <- sbGrp$plot==i
  plot(sbGrp$time[keep], sbGrp$lfmc[keep], pch=19, col=4)
  lines(sbGrp$time[keep], sb.pred1[keep])
}
```

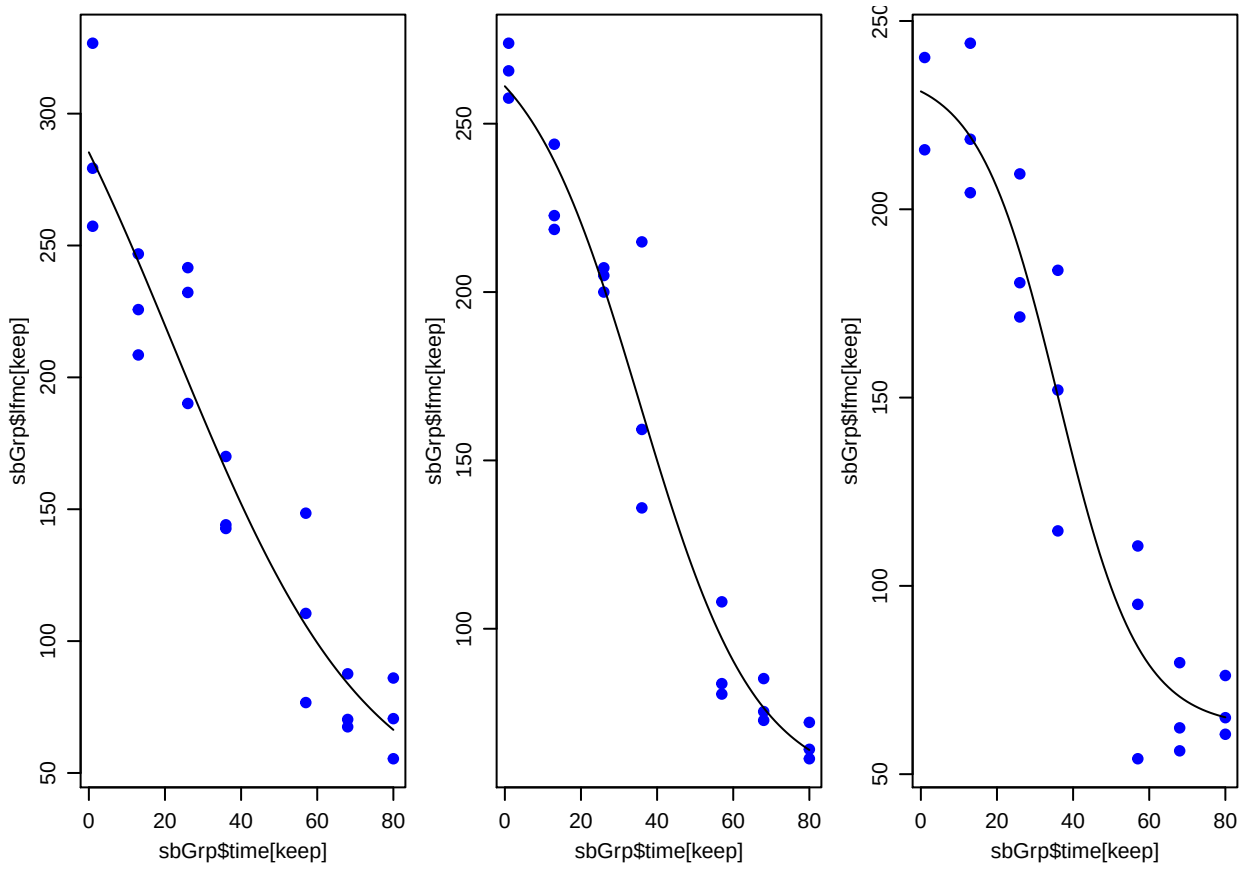


Predicting every day to get smoother plot requires a bit more setup.

```
days <- expand.grid(time=0:80, plot=1:3)
# include plot so can extract for each plot
days$pred <- predict(sb.plot, newdata=days)
# and store prediction in that data frame
# one quirk - plot variable ignored; group info used instead
#   so days needs to have all times for plot 1, then for plot 2
#   order of variables in the expand.grid() matters

# if have
#  days <- expand.grid(plot=1:3, time=0:80)
# you clearly get the wrong fitted curve

par(mfrow=c(1,3), mar=c(3,3,0,0)+0.3, mgp=c(2,0.8,0))
for (i in unique(sbGrp$plot)) {
  keep <- sbGrp$plot==i
  plot(sbGrp$time[keep], sbGrp$lfmc[keep], pch=19, col=4)
  keeppred <- days$plot==i
  lines(days$time[keeppred], days$pred[keeppred])
}
```



fit nl mixed effect model

Easy if start with the plot-specific fits (the nlsList object)

```
# get a warning message - don't ignore it
sb.nlme <- nlme(sb.plot)
```

```
## Warning in (function (model, data = sys.frame(sys.parent()), fixed, random, :
## Iteration 1, LME step: nlminb() did not converge (code = 1). Do increase
## 'msMaxIter'!
```

```
# blindly follow the advice and increase # iterations

sb.nlme <- nlme(sb.plot,
  control=nlmeControl(MaxIter = 200) )
```

```
## Warning in (function (model, data = sys.frame(sys.parent()), fixed, random, :
## Iteration 1, LME step: nlminb() did not converge (code = 1). Do increase
## 'msMaxIter'!
```

```
sb.nlme <- nlme(sb.plot,
  control=nlmeControl(MaxIter = 200, msMaxIter=200) )
```

```
## Warning in (function (model, data = sys.frame(sys.parent()), fixed, random, :
## Iteration 1, LME step: nlminb() did not converge (code = 1). PORT message:
## function evaluation limit reached without convergence (9)
```

instead of blindly trying harder, let's look at the output:

```
sb.nlme
```

```
## Nonlinear mixed-effects model fit by maximum likelihood
##   Model: lfmc ~ SSdlf(time, upper, lower, mid, scale)
##   Data: sbGrp
##   Log-likelihood: -276.37
##   Fixed: list(upper ~ 1, lower ~ 1, mid ~ 1, scale ~ 1)
##     upper     lower       mid     scale
## 280.54362  54.75035  34.09346 -14.98835
##
## Random effects:
##  Formula: list(upper ~ 1, lower ~ 1, mid ~ 1, scale ~ 1)
##  Level: plot
##  Structure: General positive-definite, Log-Cholesky parametrization
##          StdDev    Corr
## upper    33.872832 upper lower mid
## lower     5.555145 -1
## mid       2.395827 -1     1
## scale     3.283522 -1     1     1
## Residual 20.103476
##
## Number of Observations: 62
## Number of Groups: 3
```

we see that nlme is trying to fit 4 random effects (one per parameter) with arbitrary correlation matrix.

General positive-definite or look at estimated RE structure: includes correlations between parameters. We're trying to estimate 4 variances and 6 correlations from 3 units (plots). General pdMat (i.e., with correlations) is the default.

Simplify the random effects structure - no correlations. That is a pdDiag() variance covariance matrix. This runs without error.

```
sb.nlme2 <- nlme(sb.plot, random = pdDiag(upper + lower + mid + scale ~ 1) )

# could also use update(sp.nlme, random = pdDiag(upper + lower + mid + scale ~ 1) )
#   to change the random effect specification without specifying everything else again
```

?pdClasses tells you the various possibilities. default is pdLogChol, which is a better way to parameterize pdSymm

More things you can do with a fitted nlme object

```
# look at the output
summary(sb.nlme2)
```

```
## Nonlinear mixed-effects model fit by maximum likelihood
##   Model: lfmc ~ SSdlf(time, upper, lower, mid, scale)
##  Data: sbGrp
##        AIC      BIC    logLik
##   572.4362 591.5804 -277.2181
##
## Random effects:
##  Formula: list(upper ~ 1, lower ~ 1, mid ~ 1, scale ~ 1)
##  Level: plot
```

```
##  Structure: Diagonal
##            upper        lower          mid        scale Residual
## StdDev: 14.04349 0.0002898518 0.0003791913 5.946943e-05 20.49017
##
## Fixed effects: list(upper ~ 1, lower ~ 1, mid ~ 1, scale ~ 1)
##           Value Std.Error DF   t-value p-value
## upper 281.78221 22.739335 56 12.391840   0e+00
## lower  54.02396 13.525748 56  3.994157   2e-04
## mid    33.94148  2.964235 56 11.450332   0e+00
## scale -15.43268  3.754977 56 -4.109927   1e-04
##  Correlation:
##       upper  lower  mid
## lower -0.638
## mid   -0.601 -0.038
## scale -0.821  0.880  0.313
##
## Standardized Within-Group Residuals:
##        Min         Q1        Med         Q3        Max
## -1.8883324 -0.6808989 -0.0617014  0.4585169  2.8185419
##
## Number of Observations: 62
## Number of Groups: 3
```

```
# confidence intervals for the fixed effects
intervals(sb.nlme2, which='fixed')
```

```
## Approximate 95% confidence intervals
##
##  Fixed effects:
##           lower      est.      upper
## upper 237.72378 281.78221 325.840644
## lower  27.81725  54.02396  80.230676
## mid    28.19814  33.94148  39.684809
## scale -22.70811 -15.43268  -8.157253
## attr(,"label")
## [1] "Fixed effects:"
```

```
# look at the plot-specific coefficients: two ways

# estimated fixed effects and predicted random effects
fixef(sb.nlme2)
```

```
##     upper     lower       mid     scale
## 281.78221  54.02396  33.94148 -15.43268
```

```
ranef(sb.nlme2)
```

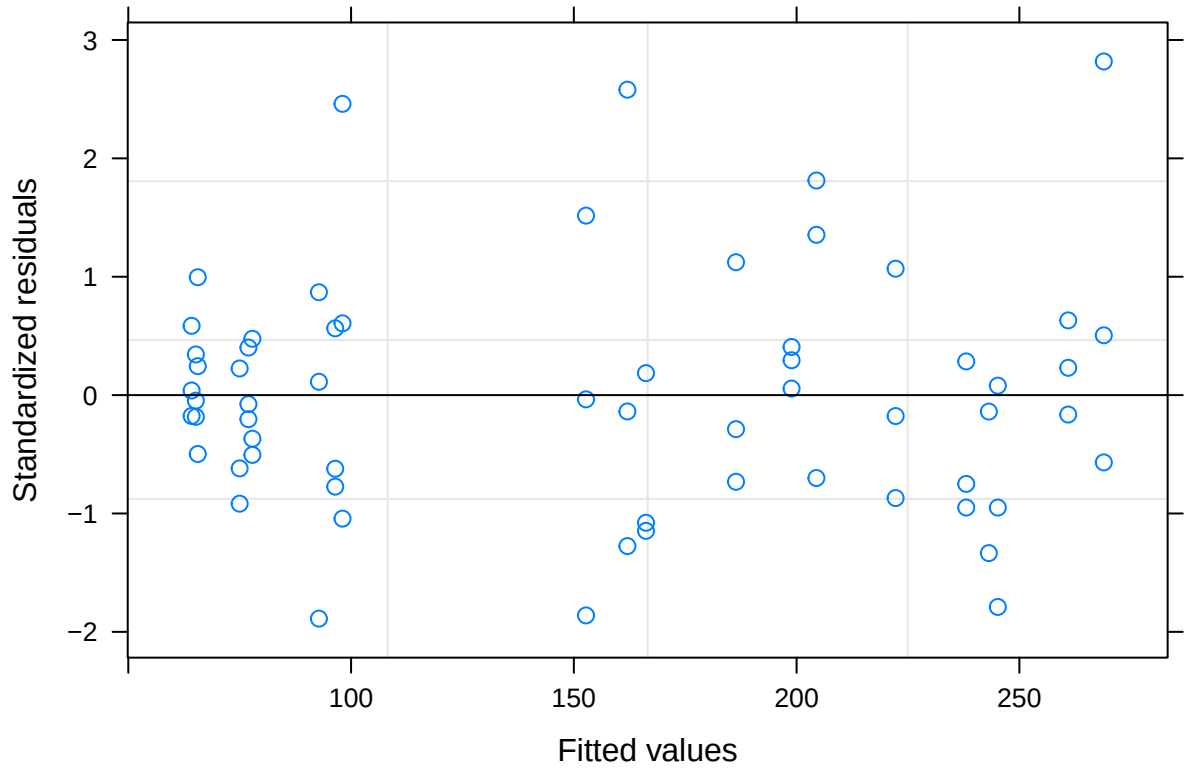
```
##        upper         lower           mid         scale
## 3 -16.258434 -7.425377e-09 -4.222835e-08  3.120019e-10
## 2   3.667469  9.332730e-10  2.046099e-08  6.964220e-11
## 1  12.590966  6.492104e-09  2.176736e-08 -3.816441e-10
```

```
# matrix of coefficients, rows = plots
coef(sb.nlme2)
```

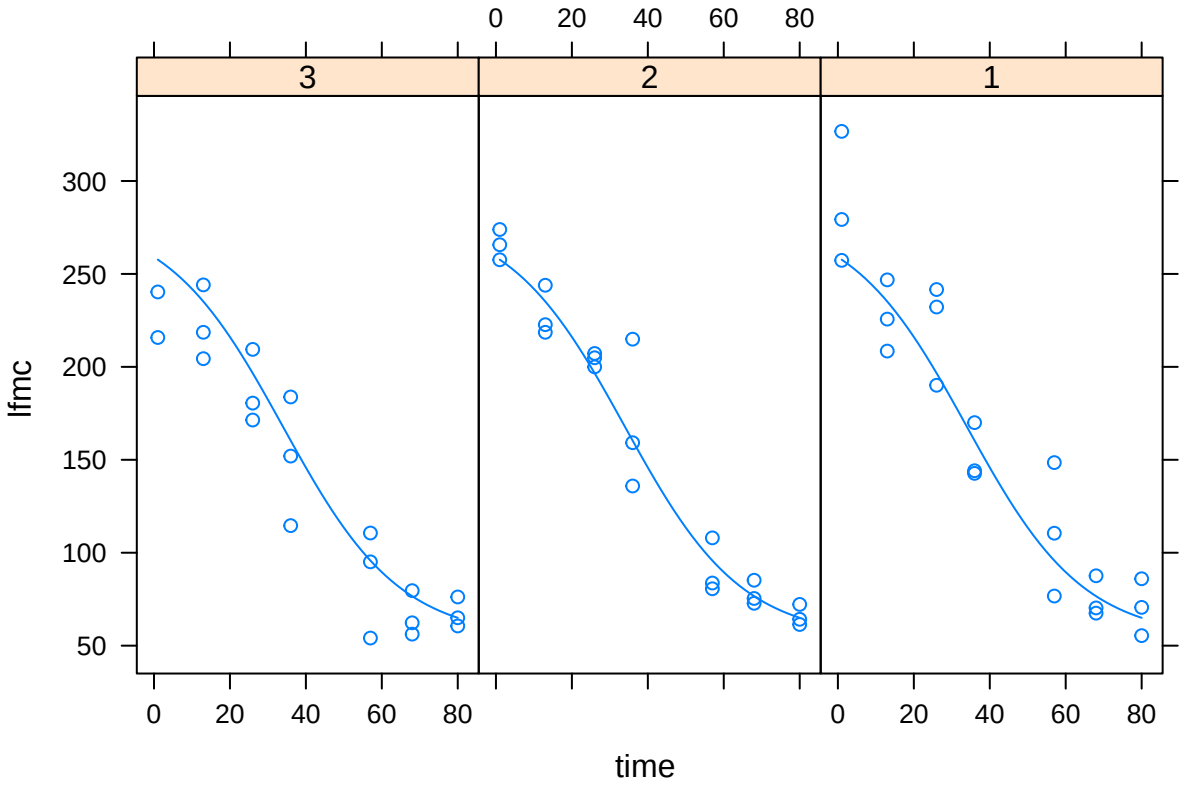
```
##      upper    lower      mid     scale
## 3 265.5238 54.02396 33.94148 -15.43268
```

```
## 2 285.4497 54.02396 33.94148 -15.43268
## 1 294.3732 54.02396 33.94148 -15.43268
```

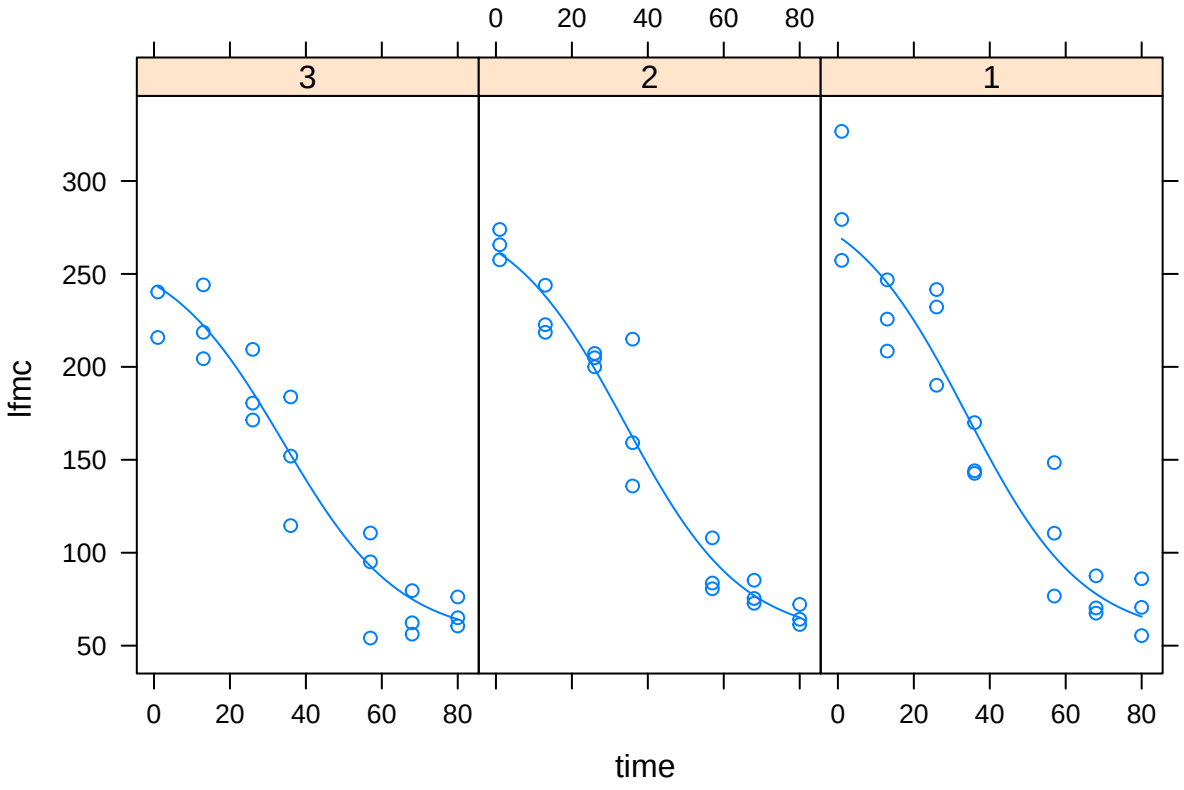
```
# look at residual vs predicted value plot
# these are standardized (variance = 1) conditional residuals
#   i.e. given BLUPs of the random effects
# useful to validate assumptions about the error distribution
plot(sb.nlme2)
```



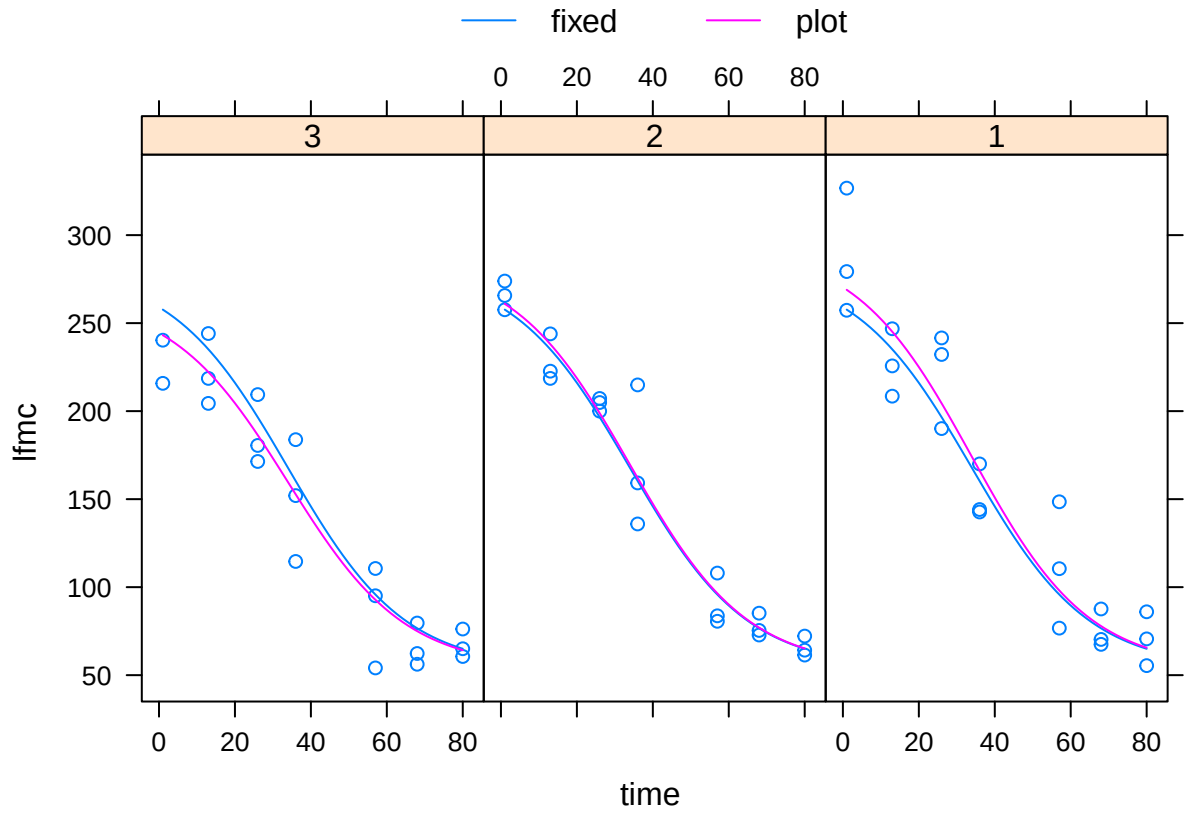

```
# plot the results: two possibilities:
#  predict using the fixed effects (same curve for all three plots)
plot(augPred(sb.nlme2, level=0))
```




```
# or using the plot-specific coefficients
plot(augPred(sb.nlme2, level=1))
```




```
# or combine the two
plot(augPred(sb.nlme2, level=0:1))
```



Results suggest no variability in lower, mid and scale, just in upper. Include random effect only in upper.

Also illustrate starting with a data frame, not the nlsList. Possible with update because model specified in sb.nlme2. I've never had success starting a new with a data frame. Haven't figured out how to specify everything nlme needs to know. My recommendation: start with grouped data.

```
sb.nlme3 <- update(sb.nlme2, random = pdDiag(upper ~ 1), data=sbGrp)

# if start with Grouped data, need to include the model
sb.nlme3 <- nlme(lfmc ~ SSdlf(time, upper, lower, mid, scale),
  random = pdDiag(upper ~ 1), data=sbGrp)

# but you need to provide more information if try to start with data.  This fails, so it isn't run here

# temp <- nlme(lfmc ~ SSdlf(time, upper, lower, mid, scale),
#  random = pdDiag(upper ~ 1), data=sb)

# can fit a model with no random effects using nls().  log-Likelihoods and AIC values can be compared b

sb.nlme4 <- nls(lfmc ~ SSdlf(time, upper, lower, mid, scale),
   data=sbGrp)

# which random effect model is more appropriate?
c(corr = AIC(sb.nlme), var4 = AIC(sb.nlme2),
  var1=AIC(sb.nlme3), fixed=AIC(sb.nlme4) )
```

```
##     corr     var4     var1    fixed
```

```
## 582.7400 572.4362 566.4362 568.0611
```

```
 c(corr = BIC(sb.nlme), var4 = BIC(sb.nlme2),
  var1=BIC(sb.nlme3), fixed=BIC(sb.nlme4) )
```

```
##     corr     var4     var1    fixed
## 614.6471 591.5804 579.1990 578.6967
```

I suspect there is a subtle issue with BIC for a mixed model. BIC depends on # nobs. To me, n for evaluating a random effect should be the number of groups (levels of the random effect). It seems that BIC() is the usual stats BIC function, which uses the total # observations. So all models other than the nls one get more heavily penalized than "they should".

This highlights a tension that runs throughout complicated models and especially NL mixed models. Do you use available functions and hope they do the correct things with your model? Or, do you write your own functions so you control exactly what they do?

What if multiple levels of nesting?

lfmc data set has 3 observations per plot, except for plot 3, time 1. Imagine the same three plants are repeatedly sampled over time. Need to add an ID variable. I've done that and saved it as lfmcID.csv. Nested random effects are specified as big / small, where small is nested in big.

```
sb2 <- read.csv('lfmcID.csv', as.is=T)
sb2$plot <- factor(sb2$plot)
sb2$ID <- factor(sb2$ID)

sb2.Grp <-  groupedData(lfmc ~ time | plot/ID, data=sb2)

# models fit separately to each plot and ID
sb2.plotID <- nlsList(lfmc ~ SSdlf(time, upper, lower, mid, scale),
  data=sb2.Grp)
sb2.plotID
```

```
## Call:
##   Model: lfmc ~ SSdlf(time, upper, lower, mid, scale) | plot/ID
##    Data: sb2.Grp
##
## Coefficients:
##         upper     lower         mid      scale
## 1/1 1926.2275 10.680271 -64.6737735 -39.558509
## 1/2  403.2904 -6.883219  21.4203661 -33.605513
## 1/3  323.1643 40.607884  39.1521829 -23.035825
## 2/1  232.2169 70.283967  44.9885378  -5.021440
## 2/2  280.2708 64.495765  32.5560780 -12.144227
## 2/3  548.5064  4.629031  -0.3218393 -37.167951
## 3/1  291.9436 49.158663  28.7078714 -20.783643
## 3/3  211.9093 54.143099  47.8045683 -10.746849
## 3/2  246.2809 58.243782  32.2393330  -4.395192
##
## Degrees of freedom: 62 total; 26 residual
## Residual standard error: 19.13152
```

Syntax for fitting nlme with nested groups is not well documented. What to do is not immediately obvious and the errors aren't helpful. Another thing I haven't yet figured out.

It would be great to get parametric bootstrap for confidence intervals or standard errors. nlme includes a simulate.lme() function, but that fails when provided a non-linear fit. Although nlme fits inherit the lme

class, simulate.lme() seems to require an lme object. Not run here

```
# sb.sim <- simulate(sb.nlme3, method='ML')
```

Using nlmer models

nlmer and nlme have the same relationship as lmer and lme for linear models. nlmer can do almost everything that nlme does. Differences are here and summarized in my notes.

Differences:

- nlme can include correlated errors; nlmer only as random effects
- specify random effects in a 3 part formula, no need to group data first
- nlmer uses a different default optimization algorithm; for smooth problems, want to specify a faster optimizer
- nlmer doesn't use the self-start part of a SS function
- need to specify starting values, partly because start= also names the parameters to estimate

Fitting the upper only RE model

```
# using the default optimizer (Nelder-Mead)
sb.nlmer3 <- nlmer(
  lfmc ~ SSdlf(time, upper, lower, mid, scale) ~ upper | plot,
  start=c(upper=286, lower=53, mid=33, scale=-16),
  data=sb)
```

```
## Warning in (function (fn, par, lower = rep.int(-Inf, n), upper = rep.int(Inf, :
## failure to converge in 10000 evaluations
```

```
# complains about not converged
# Nelder-Mead is often slow and troublesome
# if needed more interations, add , optCtrl=list(maxfun=20000)
#   to the nlmerControl argument

# better solution is to change optimizer
# bobyqa is an enhanced version of BGFS from optim
#   uses gradient information
sb.nlmer3 <- nlmer(
  lfmc ~ SSdlf(time, upper, lower, mid, scale) ~ upper | plot,
  start=c(upper=286, lower=53, mid=33, scale=-16),
  control=nlmerControl(optimizer='bobyqa'),
  data=sb)
summary(sb.nlmer3)
```

```
## Warning in vcov.merMod(object, use.hessian = use.hessian): variance-covariance matrix computed from
## not positive definite or contains NA values: falling back to var-cov estimated from RX
```

```
## Warning in vcov.merMod(object, correlation = correlation, sigm = sig): variance-covariance matrix co
## not positive definite or contains NA values: falling back to var-cov estimated from RX
```

```
## Nonlinear mixed model fit by maximum likelihood  ['nlmerMod']
## Formula: lfmc ~ SSdlf(time, upper, lower, mid, scale) ~ upper | plot
##    Data: sb
## Control: nlmerControl(optimizer = "bobyqa")
##
##      AIC      BIC   logLik deviance df.resid
##    566.4    579.2   -277.2    554.4       56
##
## Scaled residuals:
```

```
##      Min       1Q   Median       3Q      Max
## -1.89847 -0.66641 -0.06132  0.45853  2.79823
##
## Random effects:
##  Groups   Name  Variance Std.Dev.
##  plot     upper 200.8    14.17
##  Residual       419.7    20.49
## Number of obs: 62, groups:  plot, 3
##
## Fixed effects:
##       Estimate Std. Error t value
## upper  283.569     23.087  12.283
## lower   53.213     13.544   3.929
## mid     33.680      2.976  11.317
## scale  -15.771      3.777  -4.175
##
## Correlation of Fixed Effects:
##       upper  lower  mid
## lower -0.651
## mid   -0.622 -0.004
## scale -0.831  0.885  0.342
```

```
# could also use the SSfpl(): four param logistic function from base stats.  Param order changes
sb.nlmer3b <- nlmer(
  lfmc ~ SSfpl(time, lower, upper, mid, scale) ~ upper | plot,
  start=c(upper=286, lower=53, mid=33, scale=-16),
  control=nlmerControl(optimizer='bobyqa'),
  data=sb)
sb.nlmer3b
```

```
## Nonlinear mixed model fit by maximum likelihood  ['nlmerMod']
## Formula: lfmc ~ SSfpl(time, lower, upper, mid, scale) ~ upper | plot
##    Data: sb
##       AIC       BIC    logLik  deviance  df.resid
##  570.0543  582.8171 -279.0272  558.0543        56
## Random effects:
##  Groups   Name  Std.Dev.
##  plot     upper  1.971
##  Residual       21.755
## Number of obs: 62, groups:  plot, 3
## Fixed Effects:
##  upper   lower     mid   scale
##  53.37  287.13   33.12  -15.91
```

More possible models with nlmer()

Can add additional random effects, either correlated or not. Syntax identical to lme()

```
#   Correlated (fails because corr=1)

sb.nlmer4a <- nlmer(
  lfmc ~ SSdlf(time, upper, lower, mid, scale) ~ (upper + lower | plot),
  start=c(upper=286, lower=53, mid=33, scale=-16),
  control=nlmerControl(optimizer='bobyqa'),
  data=sb)
```

```
# independent random effects
sb.nlmer4b <- nlmer(
  lfmc ~ SSdlf(time, upper, lower, mid, scale) ~
    (upper | plot)  + (lower | plot),
  start=c(upper=286, lower=53, mid=33, scale=-16),
  control=nlmerControl(optimizer='bobyqa'),
  data=sb)
```

lme4 includes a bootMer() function for bootstrap confidence intervals, but I haven't figured out how to use it correctly (yet).

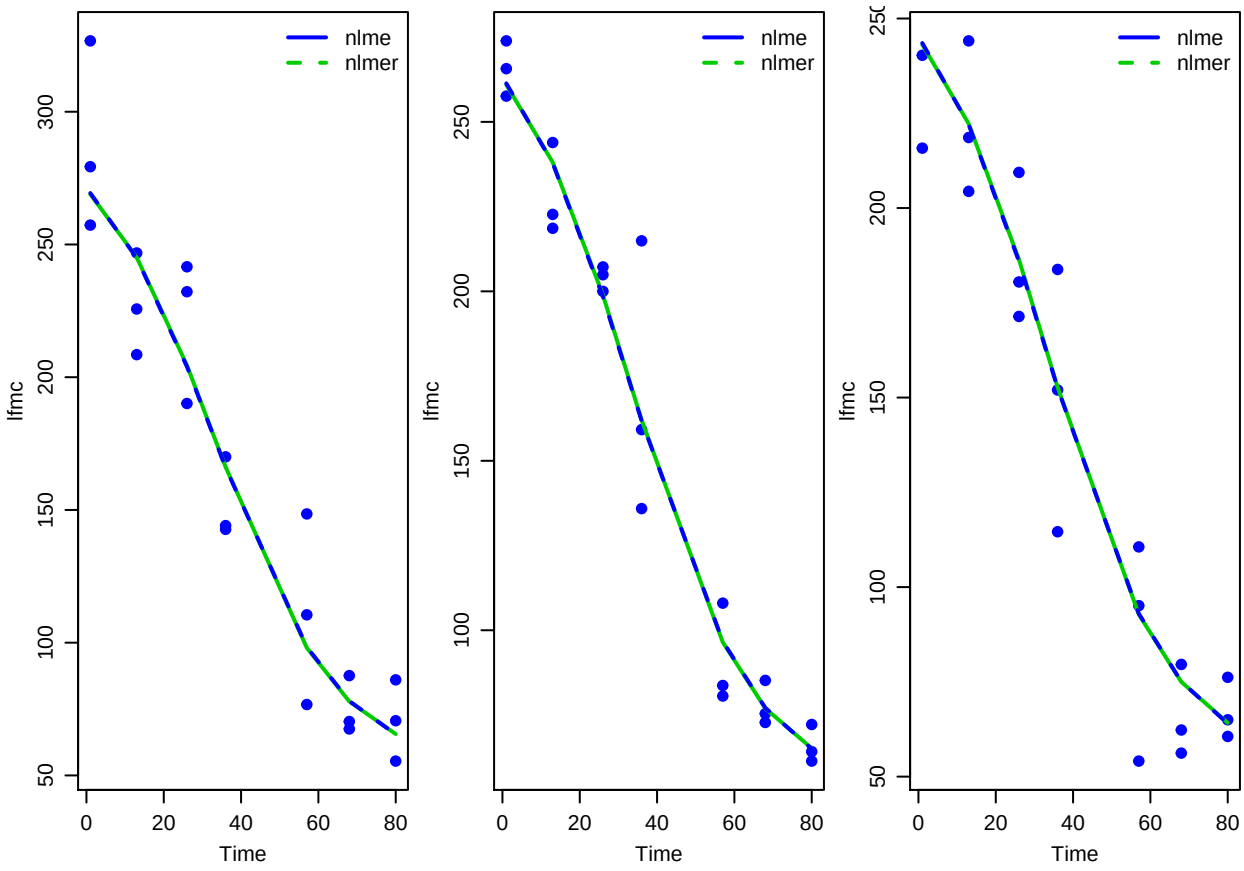
comparisons of nlme and nlmer predictions

```
# for plots in the data set
# predictions - for plots in the data set
#   use blups of the random effects for each plot

sb.nlme.pred <- predict(sb.nlme3)
# default is finest grouping level

sb.nlmer.pred <- predict(sb.nlmer3)

# compare them
par(mar=c(3,3,0,0)+0.2, mgp=c(2,0.8,0))
par(mfrow=c(1,3))
for (i in unique(sb$plot)) {
  bit <- subset(sb, plot==i)
  plot(bit$time, bit$lfmc, pch=19, col=4,
    xlab='Time', ylab='lfmc')
  lines(bit$time, sb.nlme.pred[sb$plot==i], lty=1, col=3, lwd=2)
  lines(bit$time, sb.nlmer.pred[sb$plot==i], lty=2, col=4, lwd=2)
  legend('topright',bty='n', lty=1:2, col=c(4,3),
    legend=c('nlme', 'nlmer'), lwd=2)
  }
```




```
# predictions for a new plot
#   based on fixed effect model

newdata <- data.frame(time=1:72)

sb.nlme.pred0 <- predict(sb.nlme3, newdata=newdata, level=0)
# level 0 is the population estimates of fixed effects
# when you have a new plot, don't have any knowledge of its random effects
# so predictions are based on the population (only fixed effects) parameters

# Should be possible for predictions from nlmer objects
#   but I haven't figured it out (yet)
```

meta analysis:

Starts with plot-specific estimates and standard errors. Requires that can fit model to each subject (e.g. plot).

Easiest way (that I know) is to extract from the coefficients part of the summary of an nlsList object. Then use the metafor::rma() function to do a random effects meta analysis. Have to specify the response (yi) and its standard error (sei). Or, could specify a variance instead of an se.

```
temp <- summary(sb.plot)$coef
temp
```

```
## , , upper
##
##   Estimate Std. Error   t value     Pr(>|t|)
## 3 236.8476   18.46751 12.825101 7.805525e-10
```

```
## 2 281.3863   26.24113 10.723103 5.512009e-09
## 1 393.2827  229.93011  1.710445 1.053693e-01
##
## , , lower
##
##   Estimate Std. Error   t value     Pr(>|t|)
## 3 62.36245   13.19847 4.7249769 0.0002288407
## 2 52.34283   19.56009 2.6760013 0.0159537837
## 1 31.65032   75.38980 0.4198223 0.6798732077
##
## , , mid
##
##   Estimate Std. Error    t value     Pr(>|t|)
## 3 36.17923   3.528251 10.2541555 1.935037e-08
## 2 35.47981   3.871199  9.1650717 5.469788e-08
## 1 22.06641  26.219500  0.8416029 4.116933e-01
##
## , , scale
##
##    Estimate Std. Error   t value    Pr(>|t|)
## 3 -10.58574   4.054585 -2.610808 0.018920648
## 2 -15.21243   5.002962 -3.040684 0.007384783
## 1 -25.82920  23.661615 -1.091608 0.290235558
```

```
# first column is the estimate, second is the se
upper <- temp[,1:2, 'upper']
upper
```

```
##   Estimate Std. Error
## 3 236.8476   18.46751
## 2 281.3863   26.24113
## 1 393.2827  229.93011
```

```
# random effects meta analysis
rma(yi=upper[,1], sei=upper[,2], method='REML')
```

```
##
## Random-Effects Model (k = 3; tau^2 estimator: REML)
##
## tau^2 (estimated amount of total heterogeneity): 478.9198 (SE = 1404.4187)
## tau (square root of estimated tau^2 value):      21.8842
## I^2 (total heterogeneity / total variability):   32.07%
## H^2 (total variability / sampling variability):  1.47
##
## Test for Heterogeneity:
## Q(df = 2) = 2.3046, p-val = 0.3159
##
## Model Results:
##
## estimate       se     zval    pval     ci.lb     ci.ub
## 256.4582  21.8487  11.7379  <.0001  213.6355  299.2809  ***
##
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# fixed effects meta analysis
rma(yi=upper[,1], sei=upper[,2], method='FE')
```

```
##
## Fixed-Effects Model (k = 3)
##
## I^2 (total heterogeneity / total variability):   13.22%
## H^2 (total variability / sampling variability):  1.15
##
## Test for Heterogeneity:
## Q(df = 2) = 2.3046, p-val = 0.3159
##
## Model Results:
##
## estimate       se     zval    pval     ci.lb     ci.ub
## 252.2088  15.0700  16.7359  <.0001  222.6722  281.7454  ***
##
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Differences from nlme:

1. MA estimates based on plot-specific fits
   - all parameters differ among plots
   - nlme can fit models with some parameters differing, others same for all plots (omitted from random
2. MA uses only moments (estimates, se's)
   - nlme uses full distribution of random effects
   - experience => not very sensitive to non-normal re's
3. self-starting part of SSxxxx functions not (currently) used
   - need to specify starting values as a named vector
   - fixed effect estimates ignoring plot is often a good start

Bayesian inference

Implemented in various packages, e.g. rjags, rstan, rstanarm and brms. rstanarm and brms use a more intuitive model-based syntax, not a programming-based model specification.

Using rstanarm

I'll demonstrate rstanarm. It takes advantage of pre-programmed features of models. Hence, can ONLY use self starting functions in base stats: SSasymp, SSasympOff, SSasympOrig, SSbiexp, SSfol, SSfpl, SSgompertz, SSlogis, SSmicmen, and SSweibull.

rstanarm functions are named stan_XX() where XX is the corresponding non-stan function.

```r
sb.bayes <- stan_nlmer(
  lfmc ~ SSfpl(time, upper, lower, mid, scale) ~ upper | plot,
  chains=3,
  iter=5000,
  data=sb)
```

```
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
```

```
## Chain 1:
## Chain 1: Gradient evaluation took 0.001 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 10 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1: Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1: Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1: Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 1: Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 1: Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 1: Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 1: Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 1: Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 1: Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 1: Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 1: Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 123.042 seconds (Warm-up)
## Chain 1:                4.789 seconds (Sampling)
## Chain 1:                127.831 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 0 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 2: Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 2: Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 2: Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 2: Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 2: Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 2: Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 2: Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 2: Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 2: Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 2: Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 2: Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 128.36 seconds (Warm-up)
## Chain 2:                4.899 seconds (Sampling)
## Chain 2:                133.259 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 0 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.
## Chain 3: Adjust your expectations accordingly!
```

```
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 3: Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 3: Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 3: Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 3: Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 3: Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 3: Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 3: Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 3: Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 3: Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 3: Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 3: Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 309.161 seconds (Warm-up)
## Chain 3:                842.196 seconds (Sampling)
## Chain 3:                1151.36 seconds (Total)
## Chain 3:
```

Lots of things you can do with the collection of posterior samples. Should ALWAYS check for convergence before going any further.

```
# check convergence using Rhat
summary(sb.bayes)[,'Rhat']
```

```
##                   upper                   lower                     mid
##               1.0032706               1.0053055               1.0001610
##                   scale         b[upper plot:1]         b[upper plot:2]
##               1.0042968               1.0013710               1.0013715
##         b[upper plot:3]                   sigma Sigma[plot:upper,upper]
##               1.0008491               1.0006162               1.0010116
##                mean_PPD          log-posterior
##               0.9999439               1.0051863
```

```
# visual exploration of model results using shiny via launch_shinystan  (commented out in the Rmd file)
#  if a large data set, probably want to turn off
# posterior predictive checks (   ,ppd = F)
#
#  launch_shinystan(sb.bayes)

# lots of information about estimates and diagnostics
summary(sb.bayes)
```

```
##
## Model Info:
##  function:     stan_nlmer
##  family:       gaussian [inv_SSfpl]
##  formula:      lfmc ~ SSfpl(time, upper, lower, mid, scale) ~ upper | plot
##  algorithm:    sampling
##  sample:       7500 (posterior sample size)
##  priors:       see help('prior_summary')
##  observations: 62
##  groups:       plot (3)
##
## Estimates:
```

```
##                            mean   sd     10%    50%    90%
## upper                     312.4   54.1  261.7  299.0  382.7
## lower                      35.8   31.3    0.9   43.6   63.3
## mid                        31.5    6.2   24.3   32.6   37.3
## scale                      20.7    7.9   13.1   18.7   30.7
## b[upper plot:1]            15.8   17.3   -1.4   13.4   36.8
## b[upper plot:2]             6.2   17.0  -11.3    4.3   26.3
## b[upper plot:3]           -15.2   17.2  -36.0  -14.0    2.2
## sigma                      21.6    2.1   19.1   21.5   24.4
## Sigma[plot:upper,upper]  697.3 1050.2   62.9  367.6 1629.9
##
## Fit Diagnostics:
##            mean   sd    10%   50%   90%
## mean_PPD 153.7    3.9 148.8 153.7 158.8
##
## The mean_ppd is the sample average posterior predictive distribution of the outcome variable (for de
##
## MCMC diagnostics
##                          mcse Rhat n_eff
## upper                     1.5  1.0 1278
## lower                     0.9  1.0 1126
## mid                       0.1  1.0 2476
## scale                     0.2  1.0 1177
## b[upper plot:1]           0.3  1.0 3391
## b[upper plot:2]           0.3  1.0 3416
## b[upper plot:3]           0.3  1.0 3509
## sigma                     0.0  1.0 4823
## Sigma[plot:upper,upper] 16.7  1.0 3946
## mean_PPD                  0.0  1.0 7498
## log-posterior             0.1  1.0 1503
##
## For each parameter, mcse is Monte Carlo standard error, n_eff is a crude measure of effective sample
```

```
# extract plot-specific coefficients
coefficients(sb.bayes)
```

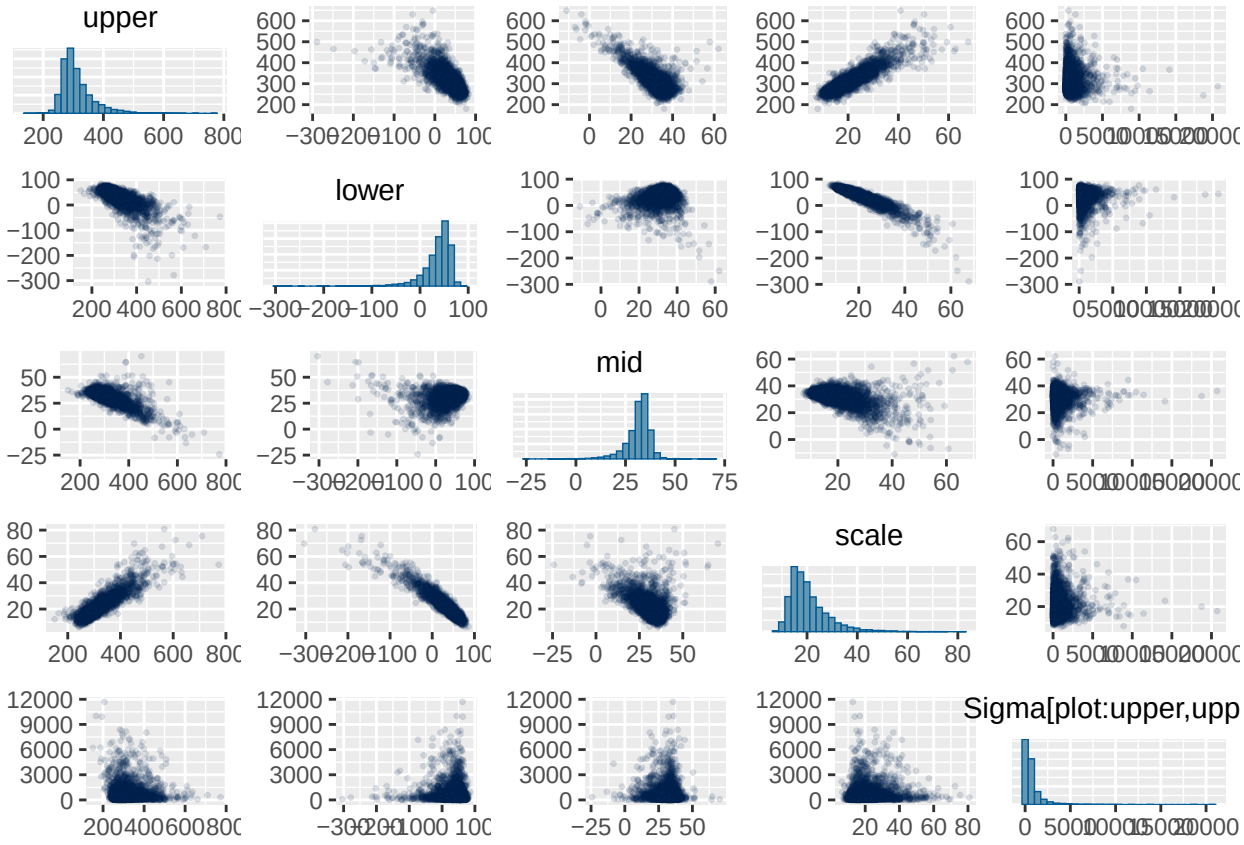
```
## $plot
##      upper    lower      mid    scale
## 1 312.4149 43.55896 32.62696 18.72636
## 2 303.2212 43.55896 32.62696 18.72636
## 3 284.9464 43.55896 32.62696 18.72636
##
## attr(,"class")
## [1] "coef.mer"
```

```
posterior_interval(sb.bayes)
```

```
##                                5%         95%
## upper                  252.420293  419.005471
## lower                  -19.767678   67.008299
## mid                     20.095631   38.699857
## scale                   11.874518   35.843876
## b[upper plot:1]         -6.280661   45.949582
## b[upper plot:2]        -17.392254   35.011668
## b[upper plot:3]        -43.726006    8.864516
```

```
## sigma                      18.480817   25.348382
## Sigma[plot:upper,upper]  29.233082 2461.583085
```

```
pairs(sb.bayes,
  pars=c('upper','lower','mid','scale',
         'Sigma[plot:upper,upper]'),
  off_diag_args=list(size=0.5, alpha = 0.15)
  )
```



I note that the posterior distributions are far from normal. Hence, Wald inference (i.e. default se's, tests, and confidence intervals from nlme or nlmer) is suspect. Alternatives are profile likelihood intervals (not yet available in nlme/nlmer) or a parametric bootstrap (nlmer:bootMer).