Perform Variance Component Analyses using R-Package **VCA**

Siegfried Erhardt, Andre Schuetzenmeister

2017-07-10

Contents

1 Introduction

The **VCA** package's main objective is to perform variance component analyses (VCA). VCA are a way to assess how the variability of a dependent variable is structured taking into account its association with one or multiple random-effects variables. Proportions of the total variability found to be attributed to these random effects variables are called *variance components* (VC). Thus, VCA is the procedure of estimating the amount of the VCs' contribution to the total variability in the dependent variable. Moroever, there are methods provided for estimating confidence intervals (CI) of VCs along with different graphical tools to better "understand" the data and for detecting outliers. Also included, but usually of less importance in the field of VCA: Estimation of fixed effects and least square means (LS means) as well as testing linear hypotheses of fixed effects/LS means of linear mixed models (LMMs).

VCs can be predicted in random models (*random effects* or *variance component models*) and LMMs (*linear mixed -effects- models*) by application of either ANOVA-type estimation or *Restricted Maximum Likelihood* (REML). Experiments of this type frequently occur in performance evaluation analyses of diagnostic tests or analyzers (devices) quantifying various types of measurement (im)precision (see e.g. *CLSI EP05-A3* guideline). In this setting it is important to point out that *precision* and *imprecision* both refer to the variability of measurements and differ only by their respective point of view, i.e. the larger the precision of a measuring method the smaller is its imprecision and vice versa.

2 Examples

In the course of the discussion of R-package **VCA**, several examples will be given to allow the user to better understand the application of the most important functions. For all of the examples, a simulated data set called *VCAdata1* will be used. **VCAdata1**, among other data sets, is included in the **VCA**-package and comprises 2520 observations. There are 6 variables for 3 devices (device), 3 lots (*lot*), 10 samples (*sample*), 21 days (*day*) and 2 runs within day (*run*) with 2 replicates per run. This means, for each run two measurements (y) were performed under conditions which are as constant as they can get. One commonly speaks of *repeatability* measurement conditions.

```
## 'data.frame':    2520 obs. of  6 variables:
##  $ lot   : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...
##  $ sample: Factor w/ 10 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ day   : Factor w/ 21 levels "1","2","3","4",..: 1 1 1 1 2 2 2 2 3 3 ...
##  $ run   : Factor w/ 2 levels "1","2": 1 1 2 2 1 1 2 2 1 1 ...
##  $ y     : num  2.11 2.09 2.73 2.72 2.87 ...
##  $ device: Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...
```

3 Visualization of Variability via Function *varPlot*

Visualization can often help in understanding new or unknown data. When performing a VCA it is highly recommended to initially take a look at a variability chart to better understand the major sources of variability and to get a rough idea of the general total variability to expect. Moreover, the variability chart can help spotting abnormalities such as outliers in the data, i.e. extreme values that lie an abnormous distance from the other remaining values and therefore are likely to have a negative effect on the analysis and possible lead to invalid results (see *4 Outlier Detection* and *5 Checking Normality and Extreme Values Using R-Package STB*).

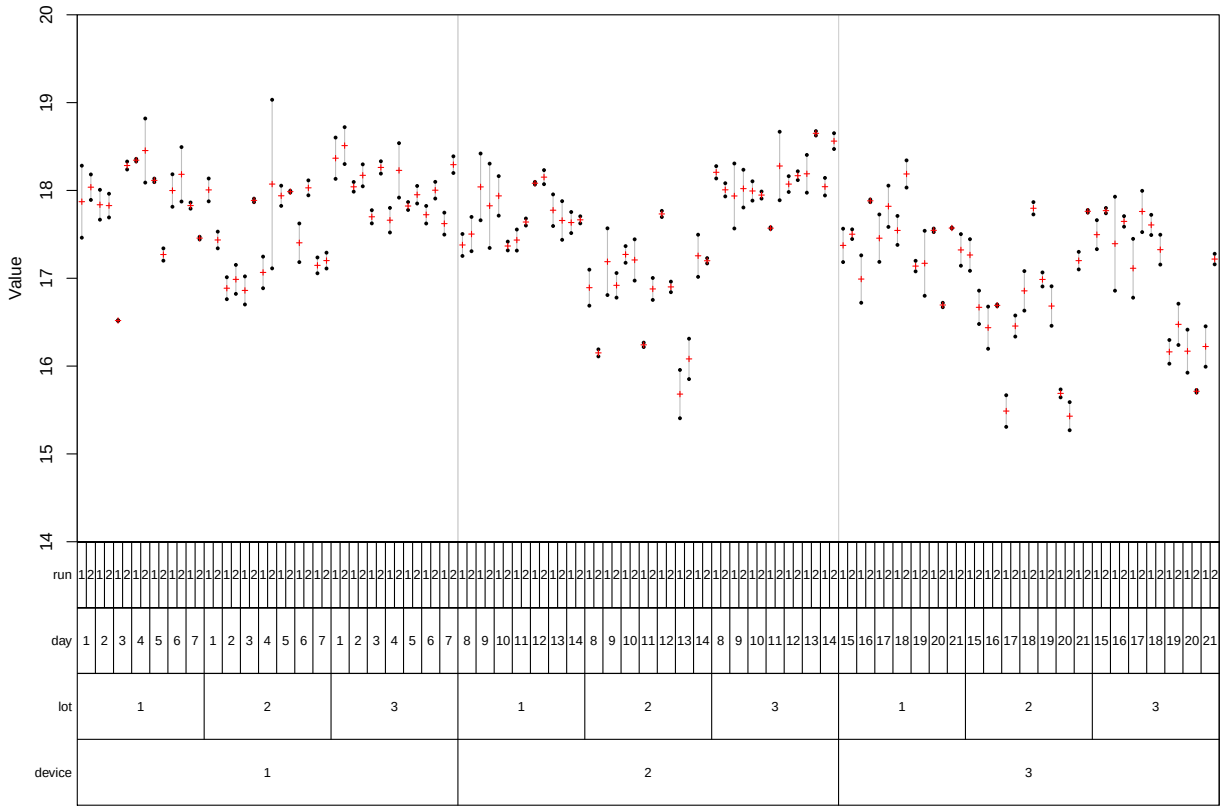
3.1 Default-Settings

VCA provides a function called **varPlot** for generating such a variability chart. To create a variability chart by *varPlot()*, it is necessary to state the model formula as well as the data set as function parameters. In this first example, a variability chart for a random model of the form *y~(device+lot)/day/run* will be plotted. All effects in a VCA are modelled as random since the interest lies in their distribution, i.e. their contribution to the total variability (variance). *Run* is nested within *day*, and *day* is, according to the model formula, nested within combinations of *lot* and *device*. **Please note** that in *varPlot()* the real model is not relevant but rather the order of the variables which determines the layout of the table depicted at the bottom of the variability chart. The implementation of the variability chart cannot distinguish between nested and crossed terms. The data that is used for drawing the variability chart is **datS5**, a subset of *VCAdata1* consisting only of 252 observations from sample number 5 from a total of 10 samples:

```
library(VCA)
data(VCAdata1)
datS5 <- subset(VCAdata1, sample==5)
```

Given the parameters **form** and **Data** only, *varPlot()* creates the variability chart in a plain design:

```
varPlot(form=y~(device+lot)/day/run, Data=datS5)
```



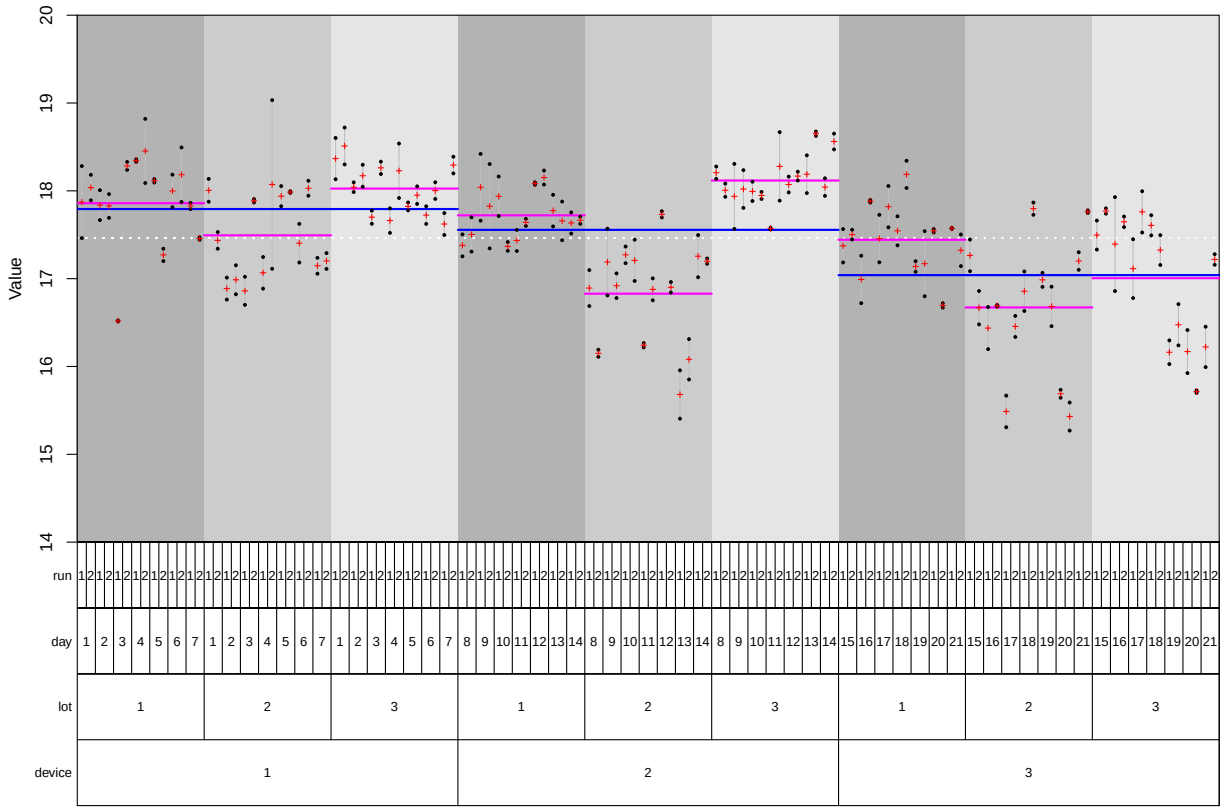
The variability chart shows all 252 observations in *datS5* according to the nesting structure of the model formula. Again, this equals 3 devices (bottom line) with 3 lots each (second from bottom), 7 measuring days (third from bottom) per lot and 2 runs per day. Since each run consists of two replicates, these two values are drawn and connected by a vertical line highlighting the complete range of replicated measurements. Whenever measurements are performed under identical conditions (replicated), i.e. no other sources of variability have an influence, the variance of residual error will be inferred from these differences. In the setting of in-vitro diagnostics one speaks of *repeatability* variance meaning the pure assay (im)precision.

3.2 Advanced-Settings

It is possible to specify the display of further graphical elements in the variability chart using various style parameters to improve the overview and facilitate information extraction. Almost all parameters correspond to list-type objects. This allows very detailed specification of the graphical appearance, e.g. in the parameter **MeanLine** variables can be selected whose mean-values are to be highlighted as horizontal lines, their colors (**col**), line-width (**lwd**), line-type etc.. In fact, all function arguments accepted by the R-function **lines** can be specified, since the list *MeanLine* is passed forward to *lines()* with some modifications taking place beforehand. In the next example, horizontal lines for mean-values (*MeanLine*) are drawn for the intercept as well as the factors device and lot. Via list-element *col* the appropriate colors are set to white, blue and magenta. Using list-element *lwd* allows to change the line width of the mean value horizontal lines. The allocation of style parameter settings to the respective graphical elements/variables is done in accordance with the order of element denomination. Furthermore, the three levels of variable lot will be highlighted in different shades of grey using function argument **BG**, which also needs to be specified as list:

```
varPlot(y~(device+lot)/day/run, datS5,
        MeanLine=list(var=c("int", "device", "lot"),
                      col=c("white", "blue", "magenta"),
                      lwd=c(2,2,2)),
```

```
        BG=list(var="lot", col=paste0("gray", c(70,80,90))))
```



4 Outlier Detection

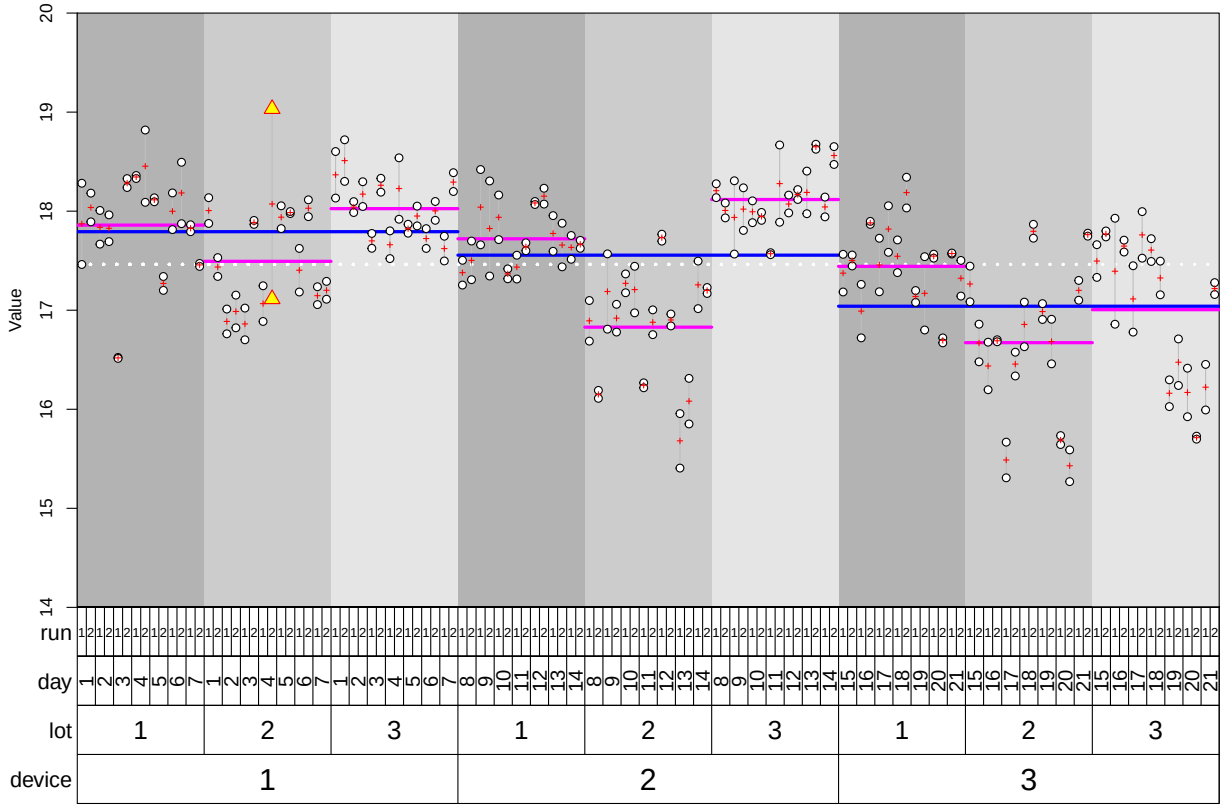
4.1 Outlier Detection by Visual Inspection

From looking at the variability chart it is noticeable that there is one pair of measurements that deviates remarkably far from its within-run mean, i.e. the small red cross on the line connecting the values vertically. These two potential outliers are the replicates of run 2 on day 4, measured in lot 2 with device 1. Extreme values might negatively influence (violate) the normality assumption applied for all random variates of a linear mixed effects model (i.e. random effects, residuals). Thereby, outliers can influence the validity of the model and its results. Following commented R-code is used to generate the variability chart with the potential outliers highlighted:

```
# indicate outliers by new variable
datS5$out <- 0
datS5$out[which(datS5$device==1 & datS5$lot==2 & datS5$day==4 &datS5$run==2)] <- 1

varPlot(y~(device+lot)/day/run, datS5,
        # plots horizontal lines for sub-sets
        MeanLine=list(var=c("int", "device", "lot"),
                      col=c("white", "blue", "magenta"),
                      lwd=c(3,3,3)),
        # colors the background according to a variable's levels
```

```
        BG=list(var="lot", col=paste0("gray", c(70,80,90))),
        # tailoring the appearance of measurements (points)
        Points=list(pch=list(var="out", pch=c(21, 24)),
                    col=list(var="out", col=c("black", "red")),
                    bg=list(var="out", bg=c("white", "yellow")),
                    cex=list(var="out", cex=c(1, 1.5))),
        # specification of the text within the table below
        VarLab=list(list(cex=1.75), list(cex=1.5),
                    list(cex=1.15, srt=90), list()),
        # variable names left of the table
        VCnam=list(cex=1.25),
        # use 33% of the height of the upper part for the table
        htab=.33)
```



The function argument **Points** can also be used as list of arguments passed forward to function *points()*. But it can be used in a more detailed manner letting the user specify list-elements *pch*, *col*, *bg*, and *cex* as lists themselves. By doing so one can incorporate additional information indicated by different plotting symbols, color, and size of the symbols. *bg* is only useful in case of setting *pch* equal to 21-25. Then, *col* is interpreted as the color of the border and *bg* as the background color of these symbols (see also the last example of the *varPlot* help).

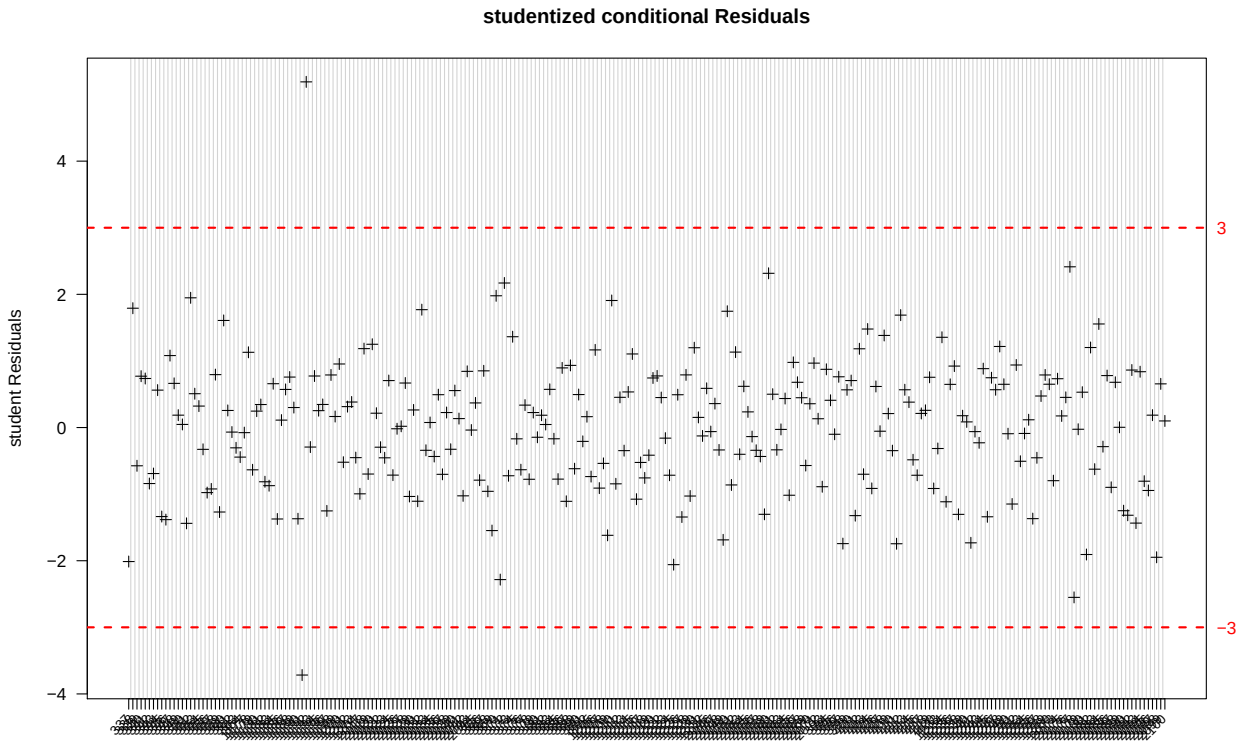
Studentization of residuals and/or random effects transforms these random variates to have mean zero and standard deviation (SD) equal to 1 using their element-wise variance estimates for standardization. Outliers amongst studentized conditional residuals correspond to outliers on the replicate level, i.e. where all measuring conditions are kept as constant as possible (here: *device*, *lot*, *day* and *run*). Studentized conditional residuals can be assessed and plotted by the **VCA**-function **plotRandVar**. *plotRandVar()* requires a fitted model as input parameter. In addition, the parameter **term** specifies the type of residuals whereas the parameter

mode specifies the transformation applied to the residuals. To assess random variates of a fitted model via *plotRandVar()*, the model has to be fitted to the data first. There are two options available for fitting a variance component model. One can use either ANOVA-type estimation via function **anovaVCA** or REML-estimation via function **remlVCA**. For balanced data and in situations where ANOVA-estimation does not produce negative variance-estimates, both methods generate identical results. Otherwise, both approaches to the estimation of random effects, and therefore VCs, are likely to differ. Although this difference is usually small. We are using ANOVA-type estimation here but all statements are also valid for REML-estimation. **Note** that there is function **fitVCA** wrapping function calls to **anovaVCA** and **remlVCA**.

Fitting the model to the data returns the object *fitS5* of class **VCA**:

```
fitS5 <- anovaVCA(y~(device+lot)/day/run, datS5)
# if varPlot was called before, better shut down the old graphics device
# graphical parameters were not reset to allow the user to add further
# information to the variability chart, which would not be possible otherwise
dev.off()
plotRandVar(fitS5, term="cond", mode="student")
```

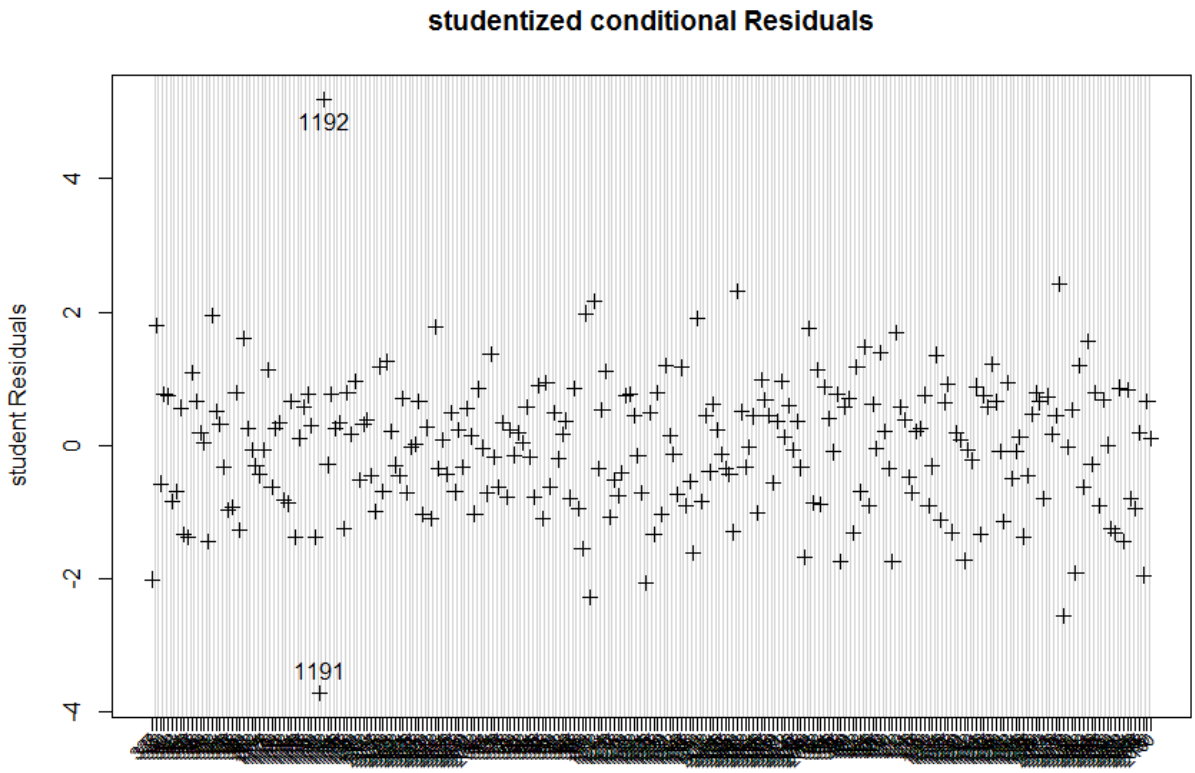
Residuals that deviate further than 3 times the SD (here SD~1) can be considered as *extreme*, since the expected value of the mean after studentization is equal to 0. Thus, only 0.3% of all observations are expected to be outside of the +/- 3xSD interval. There are two values that come into question as can be seen in the plot drawn by *plotRandVar()*:



However, since there are too many values, the labelling of the x-axis has become unreadable. Setting the function's parameter **pick** to TRUE allows selecting specific residuals by clicking on them within the graphics device window:

```
plotRandVar(fitS5, term="cond", mode="student", pick=TRUE)
```

For every value selected in the plot the respective row name within the data set will be displayed next to the value:



Please note: After having selected the desired data points, the graphics procedure has to be stopped manually by right-clicking on the graphics device output and selecting "Stop" in order to be able to execute further arguments within the R-console.

To counter-check whether the manually picked data points really are the replicate measurements initially verified from sighting the variability chart, simply select the observations from the subset datS5 using their row names:

```
datS5[c("1191","1192"),]
```

	lot	sample	day	run	y	device	out
1191	2	5	4	2	17.1125	1	1
1192	2	5	4	2	19.0325	1	1

In fact, these observations are the ones initially suspected. The values should be considered for exclusion from the data set to prevent invalid results. See *5 Checking Normality and Extreme Values Using R-Package STB* for further methods that help determine whether these observations really have to be excluded in view of the normality assumption.

4.2 An Outlier Detection Algorithm

Extreme values on the replicate-level are supposed to be detected by means of the Grubbs-test (CLSI EP05-A3). A modified version of Grubbs-test may be defined by means of the median and the MD68-statistic. The median can be seen as the robust version of the mean which is known to be sensitive to outliers, thus itself biased in case real outlying observations exist. The MD68 can be seen as the robust version of the SD, defined as follows:

```
# Function computing the MD68 using SAS PCTLDEF5 quantile definition.
# x         (numeric) values from which the MD68 shall be computed
# na.rm     (logical) TRUE = missing values will be excluded automatically
md68 <- function(x, na.rm=FALSE)
{
    stopifnot(is.numeric(x))
    Med     <- median(x, na.rm=na.rm)
    Diff    <- abs(x-Med)
    MD68    <- quantile(Diff, probs=.68, type=2)
    MD68
}
```

The critical value for this robust version of the MD68-based outlier detection algorithm was found to be reasonably well working when set equal to 2.75 x MD68. Specifically, the deviation on the replicate-level from the median of the respective replicate group may not exceed a 2.75 x MD68. The replicate groups in our running example are always *run* as the smallest grouping-variable where constant measuring conditions can be assumed. The application of this outlier-detection algorithm should be done in groups where so called intermediate precision measuring conditions exist. In the example this refers to a single lot on a single device, i.e. nine such groups exist here (3 devices x 3 lots). For each of these nine groups the specific MD68-estimate should be computed and the observation-specific deviation from the median of the replicate group should be compared to the critical value of 2.75 x MD68.

```
# identify groups of intermediate precision (IP) measuring conditions
datS5$IPgroup <- paste(datS5$device, datS5$lot, sep="_")
# uniquely identify replicate groups within IP-groups
datS5$RepGroup <- paste(datS5$day, datS5$run, sep="_")

# Define a function performing the MD68-based outlier-algorithm.
# The data.frame will be returned with two additional variables,
# "diff" (absolute differences from the replicate-group median) and
# "outlier" (TRUE = outlier, FALSE = no outlier).
#
# obj       (data.frame) with at least two variables
# resp      (character) name of the numeric response variable
# RepGroup  (character) name of the replicate-grouping variable
md68OutlierDetection <- function(obj=NULL, resp=NULL, RepGroup=NULL)
{
    stopifnot(is.data.frame(obj))
    cn <- colnames(obj)
    stopifnot(is.character(resp) && resp %in% cn && is.numeric(obj[,resp]))
    stopifnot(is.character(RepGroup) && RepGroup %in% cn)
    MD68 <- md68(obj[, resp])
    Crit <- MD68*2.75
    # tapply returns a list with as many elements as there are unique
    # elements in obj[,RepGroup], which must be converted to a vector
    obj$diff <- unlist(
                    tapply(obj[,resp], obj[,RepGroup],
```

```
                        function(x)
                        {
                            m <- median(x)
                            d <- abs(x - m)
                            d
                        }))
    obj$threshold <- Crit
    obj$outlier   <- obj$diff > Crit
    obj
}
```

To apply this MD68-based outlier detection algorithm to the data, IP-group by IP-group, one can use a simple for-loop. The result is data.frame *out* which has three additional variables (*threshold*, *diff*, and *outlier*).

```
IPgroups <- unique(datS5$IPgroup)
for(i in 1:length(IPgroups))
{
    tmpData <- subset(datS5, IPgroup == IPgroups[i])
    if(i == 1)
        out <- md68OutlierDetection(tmpData, resp="y", RepGroup="RepGroup")
    else
        out <- rbind(out, md68OutlierDetection(tmpData, resp="y", RepGroup="RepGroup"))
}

head(out)
```

	lot	sample	day	run	y	device	out	IPgroup	RepGroup	diff	threshold	outlier
337	1	5	1	1	17.46195	1	0	1_1	1_1	0.410	1.251536	FALSE
338	1	5	1	1	18.28195	1	0	1_1	1_1	0.410	1.251536	FALSE
339	1	5	1	2	17.89205	1	0	1_1	1_2	0.145	1.251536	FALSE
340	1	5	1	2	18.18205	1	0	1_1	1_2	0.145	1.251536	FALSE
341	1	5	2	1	18.00719	1	0	1_1	2_1	0.170	1.251536	FALSE
342	1	5	2	1	17.66719	1	0	1_1	2_1	0.170	1.251536	FALSE

```
# Were there any outliers detected?
any(out$outlier)
```

```
## [1] FALSE
```

The expression shown above states that no element of variable 'outlier' takes the value *TRUE*, i.e. no outlier was identified. As one can see by thoroughly inspecting the **R** source-code, for these data the outlier detection algorithm seems to suffer from relatively large between-day and between-run variability. Both contribute to the total variance within IP-group *1_2* additionally to the measurement imprecision, which is reflected in a rather large threshold value 2.75 x MD68 = 1.624. In consequence, this results in not detecting the previously identified two observations *1191* and *1192* which deviate substantially from each other under repeatability conditions. The above outlined extreme value detection is thus rather conservative, i.e. replicates have to differ a lot in order to be termed extreme.

4.3 Extreme Values on All Levels

Outliers may occur on all levels in an LMM, other levels than that of replicates should be assessed using studentized random effects.
This is possible by specifying the desired random effects term in the *term* parameter of *plotRandVar()*. The

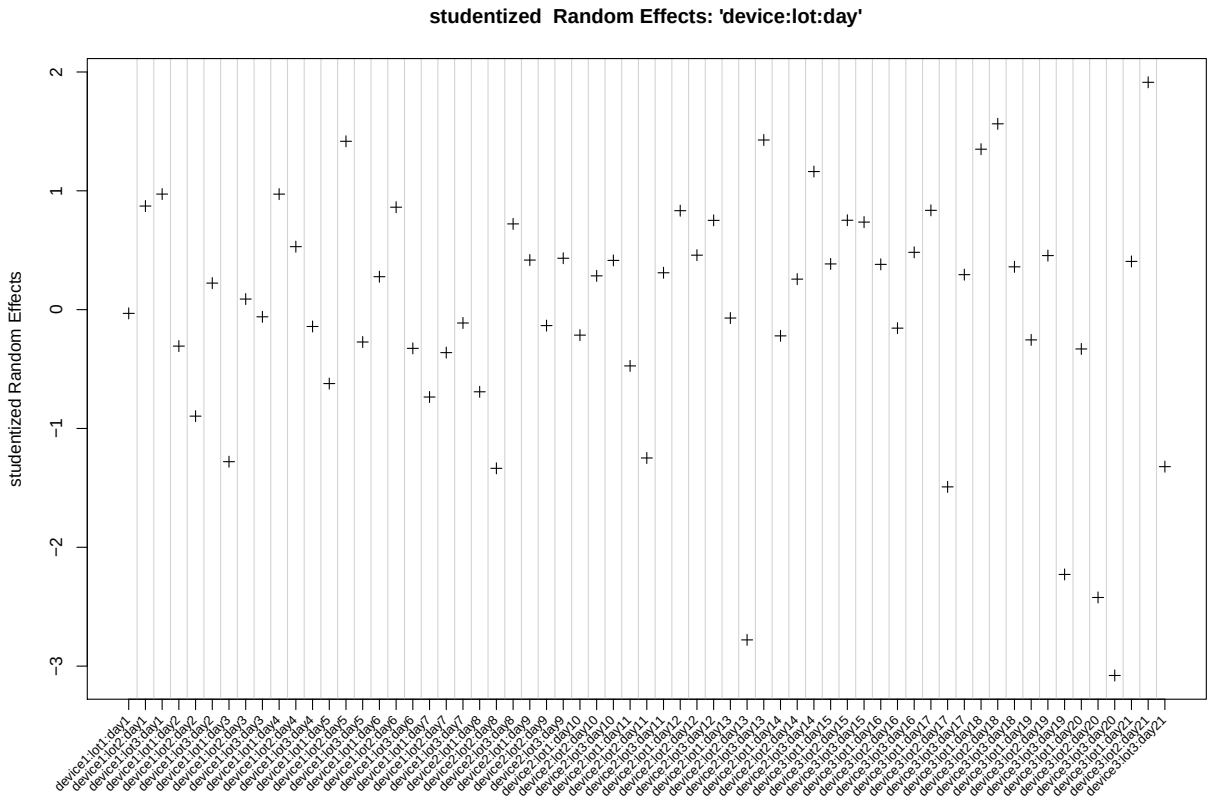
exact term denomination can be obtained from the ANOVA-table visible when printing the *VCA*-object. It is also part of the *VCA*-object *fitS5*:

```
fitS5
```

	DF	SS	MS	VC	%Total	SD	CV[%]
total	15.17855	NA	NA	0.6145008	100.00000	0.7839010	4.488937
device	2.00000	24.86953	12.4347641	0.1379096	22.44254	0.3713618	2.126569
lot	2.00000	27.28425	13.6421262	0.1522830	24.78157	0.3902345	2.234642
device:lot:day	58.00000	49.32077	0.8503581	0.1402789	22.82811	0.3745383	2.144759
device:lot:day:run	63.00000	18.22227	0.2892424	0.1052131	17.12171	0.3243656	1.857450
error	126.00000	9.93085	0.0788163	0.0788163	12.82606	0.2807424	1.607645

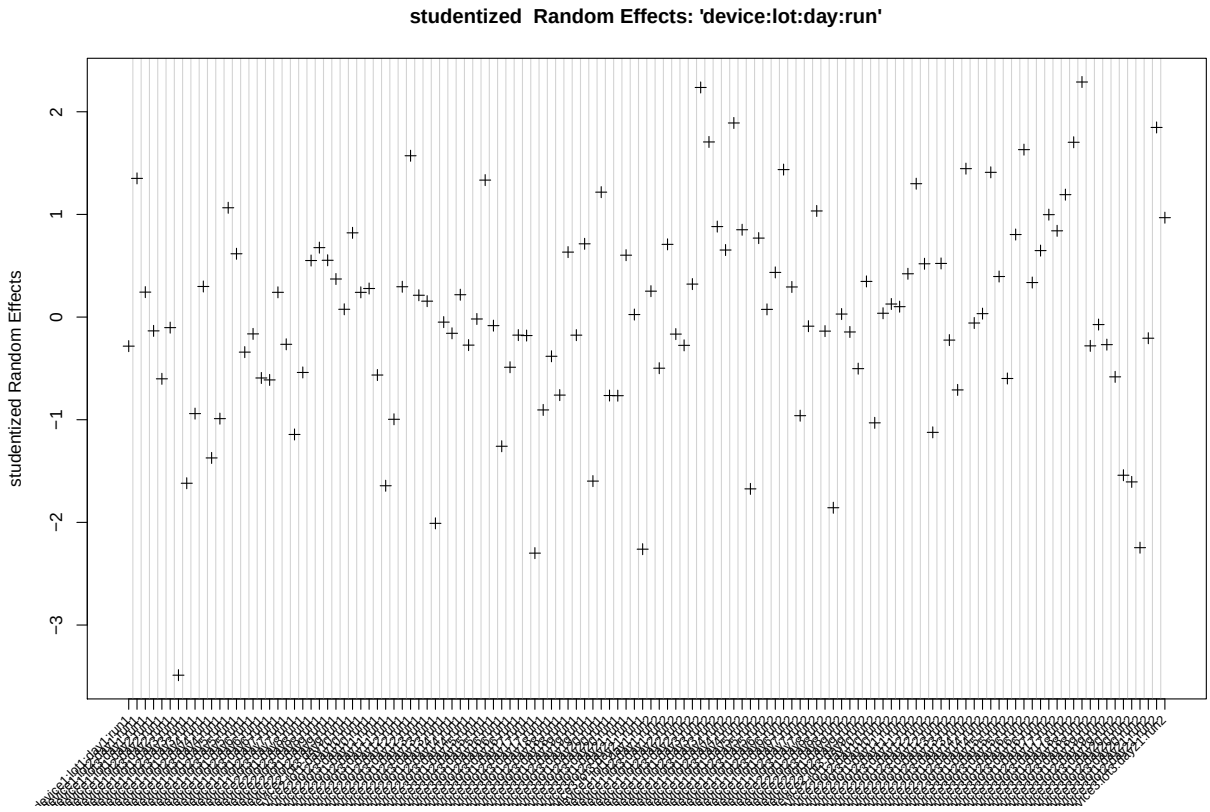
As an example, the studentized random effects of "device:lot:day". This reveals the between-day variability:

```
plotRandVar(fitS5, term="device:lot:day", mode="student")
```



As well as the studentized random effects of "device:lot:day:run", which show the within-run variability:

```
plotRandVar(fitS5, term="device:lot:day:run", mode="student")
```



Please note: If there are too few factor levels (here e.g. for *device* and *lot* with 3 levels each), the assessment of studentized random effects is only of limited use.

5 Checking for Normality and Extreme Values Using R-Package STB

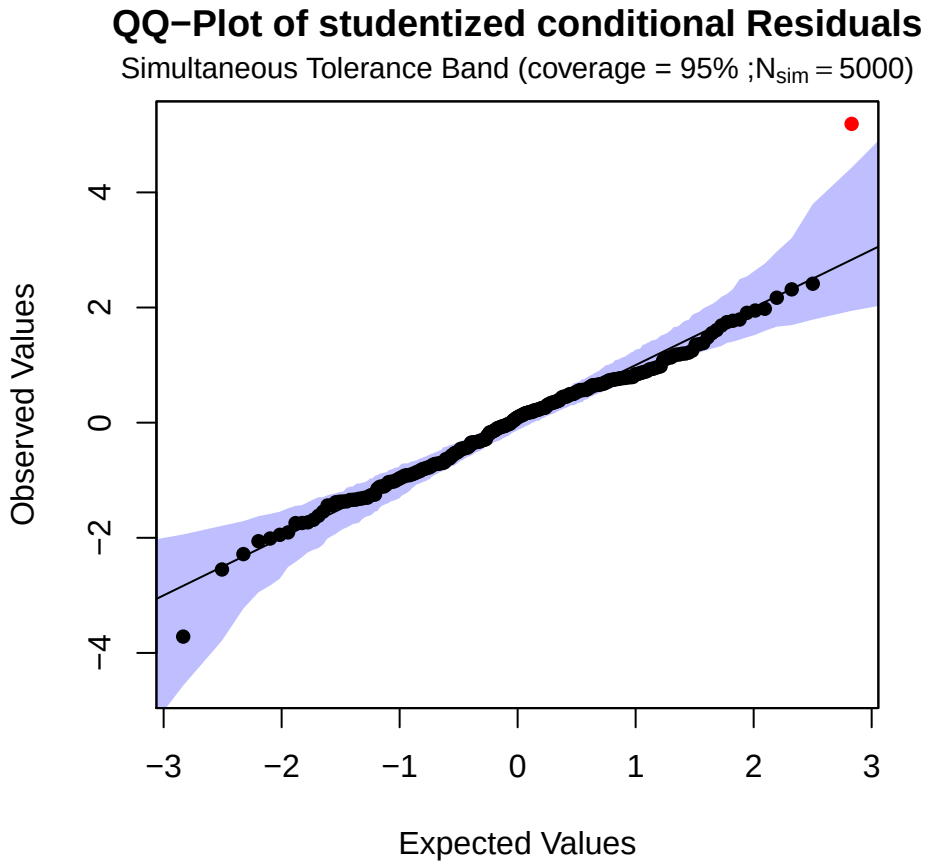
As indicated before in *2.1 Outlier Detection*, it is assumed that random factors are stochastically independent and random effects are normally distributed. However, extreme values, i.e. outliers, can violate this normality assumption and therefore distort estimations. The rule of thumb is to exclude as few observations as possible and as many as necessary, i.e. generally a maximum of two outliers that violate the normality assumptions the most.

A reasonable means to visually check for observations or extreme values that violate this assumption is to draw a **Q-Q plot** (quantile-quantile plot). In this context, the Q-Q plot computes the theoretically expected random variates (i.e. random effects or residuals) given a normal distribution. This results in a straight diagonal line. Then the observed random variate values are drawn against the expected values. If the observed values evidently deviate from the straight diagonal line, the data most likely are not normally distributed. An additional aid in visually determining if observations violate the normal assumption is to add tolerance bands to the Q-Q plot.

The R-package **STB** provides the function **stb.VCA** which simulates N-times data sets based on an LMM fit object of class *VCA* and constructs a 100(1-alpha)% simultaneous tolerance band (STB) for the simulated data. The tolerance band, then again, is based on random variates exctracted from the simulated data sets. The construction of the tolerance band requires a reasonably large number of simulations N to calculate sufficiently exact tolerance bands. In the following example, simultaneous tolerance bands from studentized (**mode**) conditional (**term**) residuals extracted from 5000 simulated data sets (**N**) are created using methode

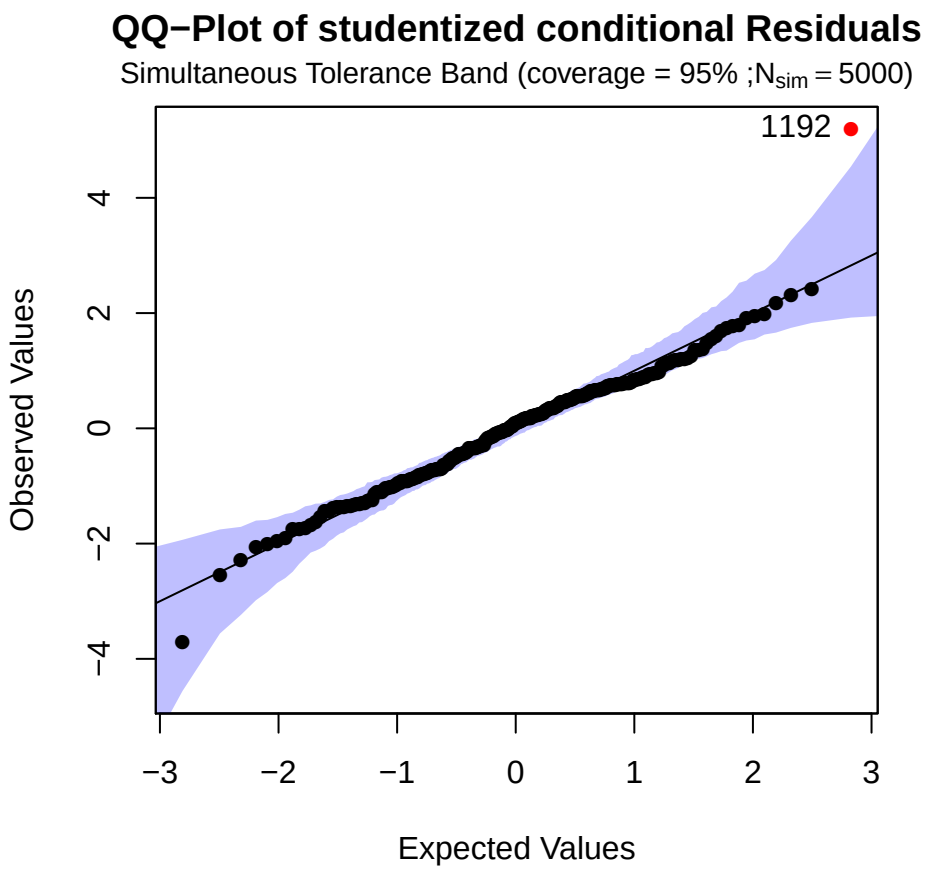
stb for objects of class *VCA*. To do so, an LMM fit for *datS5* is required. By way of example, all model terms are assumed to be random:

```
fitS5.LMM <- anovaMM(y~((device)+(lot))/(day)/(run), datS5)
STB.res <- stb(fitS5.LMM, term="cond", mode="student", N=5000)
```



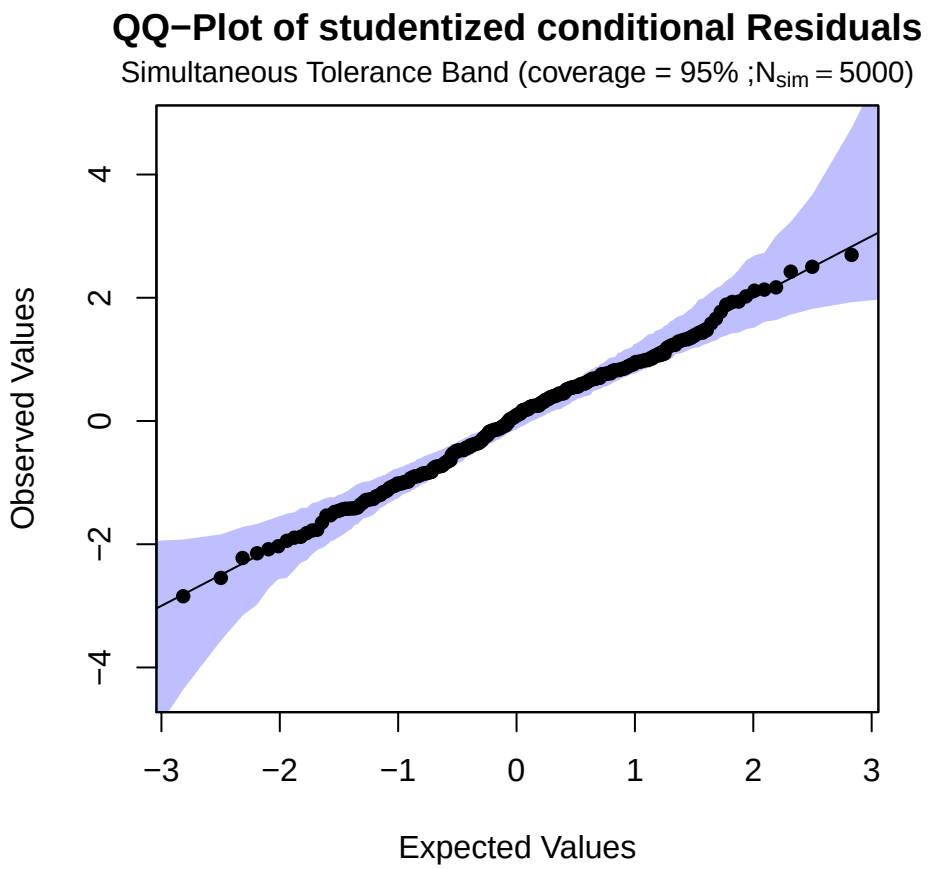
One can clearly see that there is one observation in the top right corner lying outside of the tolerance band. Applying the generic R-function **plot** to the object *STB.res* created above and setting its parameter **pick** to TRUE will allow for manual selection of observations by clicking on them within the graphics device window - just as shown in the example above when discussing the variability chart:

```
plot(STB.res, pick=TRUE)
```



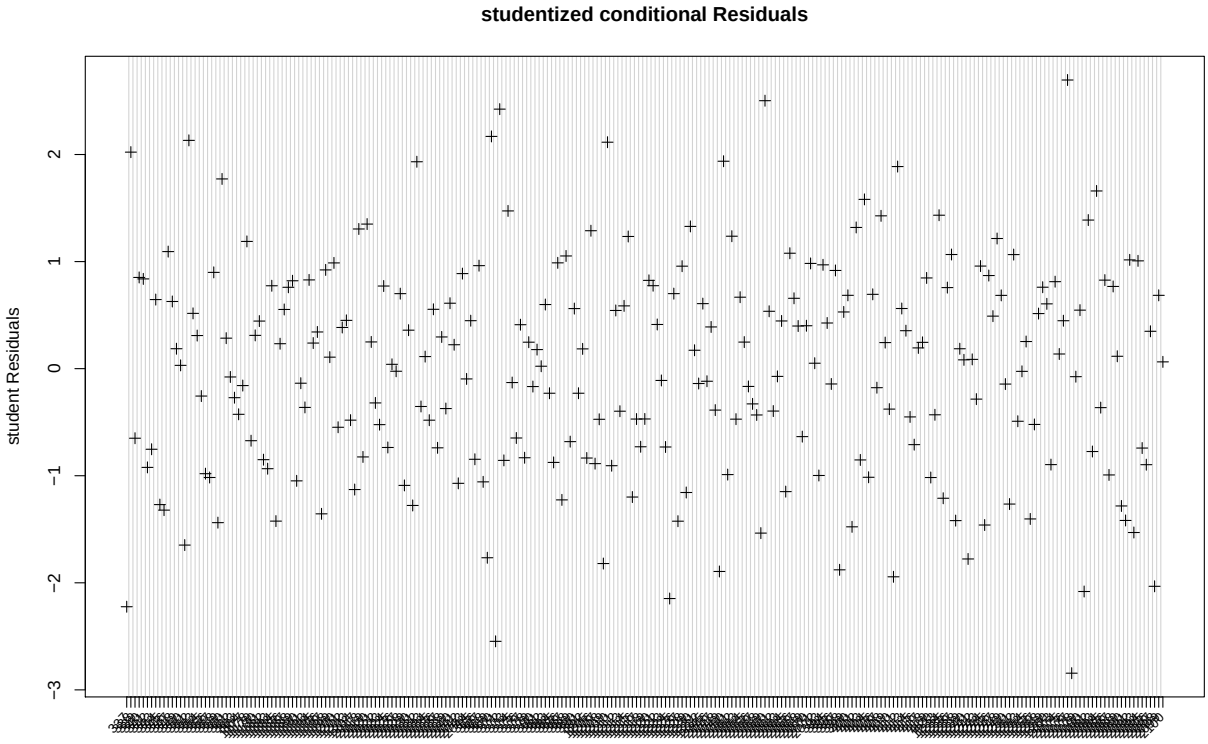
Since observation 1192 is violating the normality assumption, it is removed from the original data set and the tolerance bands are calculated again:

```
datS5.reduced <- datS5[!rownames(datS5) == "1192",]
fitS5.LMM2 <- anovaMM(y~((device)+(lot))/(day)/(run), datS5.reduced)
STB.res2 <- stb(fitS5.LMM2, term="cond", mode="student", N=5000)
```



As it turns out, removing only the observation 1192 already results in a QQ-plot that obviously does not show any indications for the violation of the normality assumption anymore. To make absolutely sure, another variability chart is drawn. Another model fit with the reduced data has to be performed in advance:

```
fitS5.reduced <- anovaVCA(y~(device+lot)/day/run, datS5.reduced)
plotRandVar(fitS5.reduced, term="cond", mode="student")
```



In fact, observation 1191 of the initially (see *4 Outlier Detection*) suspected value pair does not have to be excluded from the data set anymore.

Excluding too many observations potentially leads to inaccurate estimates since information is lost. On the other hand, not excluding observations that should be removed from the data because of the violation of the normality assumption leads to invalid estimates. Once again, the following ANOVA table shows the estimates that result from fitting the original full subset *datS5* still containing observation 1191 and 1192. Pay special attention to the error's contribution to the total amount of variance (see *%Total* column):

	DF	SS	MS	VC	%Total	SD	CV[%]
total	15.17855	NA	NA	0.6145008	100.00000	0.7839010	4.488937
device	2.00000	24.86953	12.4347641	0.1379096	22.44254	0.3713618	2.126569
lot	2.00000	27.28425	13.6421262	0.1522830	24.78157	0.3902345	2.234642
device:lot:day	58.00000	49.32077	0.8503581	0.1402789	22.82811	0.3745383	2.144759
device:lot:day:run	63.00000	18.22227	0.2892424	0.1052131	17.12171	0.3243656	1.857450
error	126.00000	9.93085	0.0788163	0.0788163	12.82606	0.2807424	1.607645

Removing only observation 1192 since it causes the violation of the normality asusmption results in the following estimates. Note how the *error %Total* reduced from 12.83% before to 10.64% now:

	DF	SS	MS	VC	%Total	SD	CV[%]
total	14.40503	NA	NA	0.6080822	100.00000	0.7797963	4.467031
device	2.00000	23.95170	11.9758513	0.1329849	21.86955	0.3646709	2.089002
lot	2.00000	28.69275	14.3463766	0.1613291	26.53080	0.4016579	2.300881
device:lot:day	58.00000	49.21067	0.8484598	0.1442632	23.72429	0.3798199	2.175783
device:lot:day:run	63.00000	17.21160	0.2732000	0.1048039	17.23515	0.3237343	1.854499

	DF	SS	MS	VC	%Total	SD	CV[%]
error	125.00000	8.08765	0.0647012	0.0647012	10.64021	0.2543643	1.457116

6 Commonly Used VCA-Models

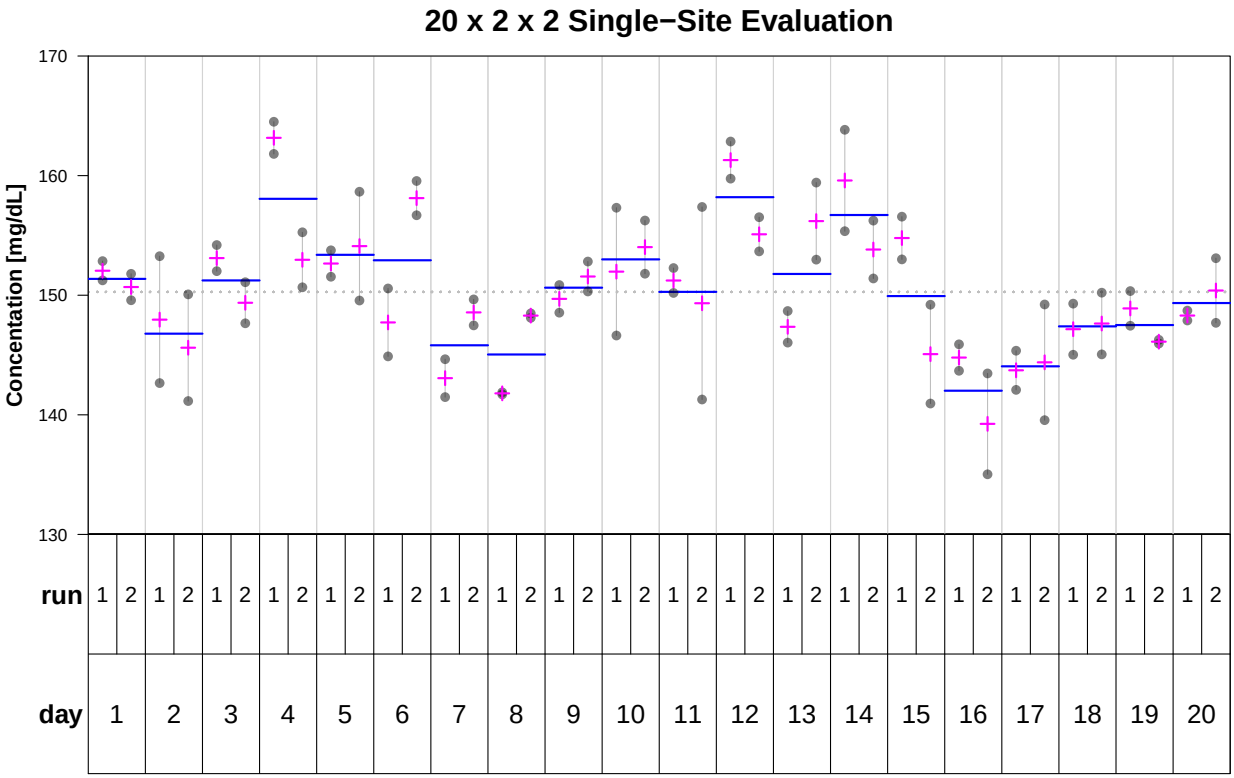
In this section we want to give an overview on the most commonly used VCA-models. The *CLSI EP05-A3* guideline describes multiple experimental settings, e.g. the single-site evaluation study and the multi-site evaluation study. Here, we show how these experiments can be analyzed using R-package **VCA**.

6.1 Single Site Evaluation

The single site experiment recommended in the *CLSI EP05-A3* consists of 20 days, 2 runs per day with 2 replicates per run (20 x 2 x 2).

```
# Function converts a color-string into RGB-code
# col       (character) string specifying an R-color
# alpha     (numeric) degree of transparency in [0, 1], 0=fully transparency, 1=opaque
asRGB <- function(col, alpha) return( rgb(t(col2rgb(col))/255, alpha=alpha) )

data(dataEP05A2_3)
varPlot(y~day/run, dataEP05A2_3,
        # controls horizontal mean lines
        MeanLine=list(var=c("int", "day"), col=c("gray75", "blue"), lwd=c(2,2)),
        # controls how points (concentrations) are plotted, here using semi-transparency
        # to see overlayed points
        Points=list(pch=16, col=asRGB("black", .5), cex=1.25),
        # controls how replicate-means are plotted
        Mean=list(col="magenta", cex=1.25, lwd=2),
        # controls how the title is shown
        Title=list(main="20 x 2 x 2 Single-Site Evaluation", cex.main=1.75),
        # controls plotting of levels per VC, if as many lists as there are VCs are
        # specified, each VC can be specified individually
        VarLab=list(list(cex=1.5), list(cex=1.25)),
        # controls how names of VCs are plotted
        VCnam=list(font=2, cex=1.5),
        # controls appearance of the Y-axis label
        YLabel=list(text="Concentation [mg/dL]", las=0, line=3, font=2, cex=1.25),
        # Y-axis labels rotated
        las=1)
```



The **VCA** package comes with three example data sets of this type (*dataEP05A2_1*, *dataEP05A2_3*, *dataEP05A2_3*). Here, *run* is nested within *day*, since all 20 days are independent from each other, thus, run No. 1 on a given day does not have anything in common with run No. 1 on another day. This can be expressed as shown below using the nesting-operator '/'. After the model was fitted to the data some additional inferential statistics are usually of interest, such as confidence intervals for VCs and/or performing Chi-Squared tests for claims of repeatability or total imprecision. This can be addressed using function *VCAinference*.

```
# fit 20 x 2 x 2 model to data
fit.SS3 <- anovaVCA(y~day/run, dataEP05A2_3)
fit.SS3
```

```
##
##
## Result Variance Component Analysis:
## ----------------------------------
##
##   Name    DF        SS          MS        VC        %Total    SD
## 1 total   50.129098                       35.546313 100       5.962073
## 2 day     19        1506.145222 79.270801 12.231099 34.40891  3.497299
## 3 day:run 20        606.928138  30.346407 7.031193  19.780372 2.65164
## 4 error   40        651.360839  16.284021 16.284021 45.810718 4.035346
##   CV[%]
## 1 3.967511
## 2 2.327307
## 3 1.764556
## 4 2.685355
```

```
##
## Mean: 150.2724 (N = 80)
##
## Experimental Design: balanced  |  Method: ANOVA
```

```
# estimate 95% confidence intervals, request CI for
# all variance components via 'VarVC=TRUE'
inf.SS3 <- VCAinference(fit.SS3, VarVC=TRUE)
inf.SS3
```

```
##
##
##
## Inference from (V)ariance (C)omponent (A)nalysis
## ------------------------------------------------
##
## > VCA Result:
## -------------
##
##   Name    DF      SS        MS      VC      %Total  SD     CV[%]  Var(VC)
## 1 total   50.1291                   35.5463 100     5.9621 3.9675
## 2 day     19      1506.1452 79.2708 12.2311 34.4089 3.4973 2.3273 47.0968
## 3 day:run 20      606.9281  30.3464 7.0312  19.7804 2.6516 1.7646 26.3372
## 4 error   40      651.3608  16.284  16.284  45.8107 4.0353 2.6854 13.2585
##
## Mean: 150.2724 (N = 80)
##
## Experimental Design: balanced  |  Method: ANOVA
##
##
## > VC:
## -----
##         Estimate CI LCL  CI UCL  One-Sided LCL One-Sided UCL
## total   35.5463  24.8957 54.8935 26.338        51.0984
## day     12.2311  0*      25.6818 0.9429        23.5193
## day:run 7.0312   0*      17.0897 0*            15.4726
## error   16.284   10.9764 26.659  11.6818       24.571
##
## > SD:
## -----
##         Estimate CI LCL CI UCL One-Sided LCL One-Sided UCL
## total   5.9621   4.9896 7.409  5.1321        7.1483
## day     3.4973   0*     5.0677 0.9711        4.8497
## day:run 2.6516   0*     4.134  0*            3.9335
## error   4.0353   3.3131 5.1632 3.4179        4.9569
##
## > CV[%]:
## --------
##         Estimate CI LCL CI UCL One-Sided LCL One-Sided UCL
## total   3.9675   3.3203 4.9304 3.4152        4.7569
## day     2.3273   0*     3.3724 0.6462        3.2273
## day:run 1.7646   0*     2.751  0*            2.6176
## error   2.6854   2.2047 3.4359 2.2744        3.2986
##
##
```

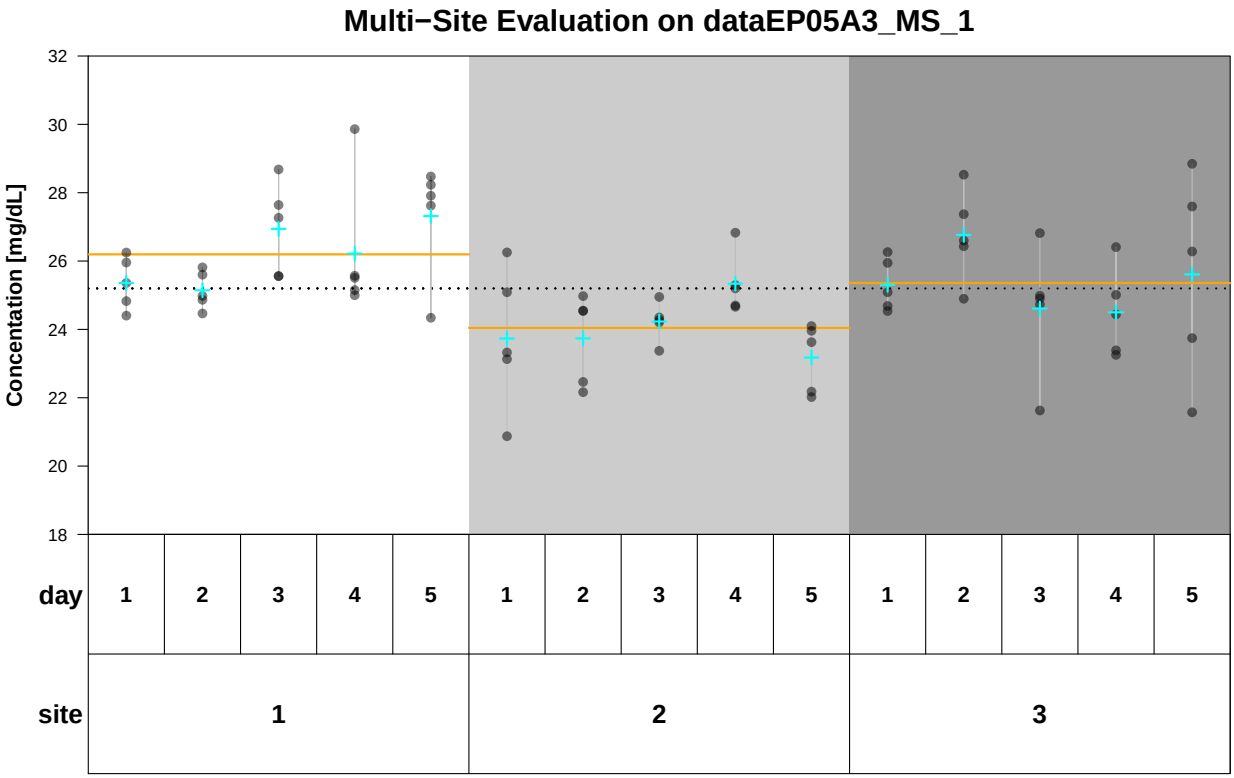
```
## 95% Confidence Level  | * CI-limits constrained to be >= 0
## SAS PROC MIXED method used for computing CIs
```

The default-setting of function 'anovaVCA' sets all negative variance estimates equal to zero, since negative values for variances are not defined. This is a known problem of ANOVA-estimation for variance components. There are several reasons which might cause negative variance estimates, e.g. the model specified does not match the structure of the data (wrong model) or there might be outlying observations negatively influencing model assumptions (normality) or the variability might just be too large. These explanation are given in *SAS Documentation of PROC VARCOMP*, in section *Negative Variance Component Estimates*.

6.2 3 x 5 x 1 x 5 Multi-Site Evaluation

The next model we would like to look at is used in the 3 x 5 x 1 x 5 multi-site evaluation study. Here, 3 devices (site, laboratories, ...) are used and each sample is measured on 3 days in a single run in 5 replicates resulting in 75 observations overall. R-package **VCA** comes with 3 such data sets (*dataEP05A3_MS_1*, *dataEP05A3_MS_3*, *dataEP05A3_MS_3*). This model assumes a single reagent-lot to be used on all three sites (devices, labs, ...). A nice-looking variability-chart for such data can be generated as follows:

```
data(dataEP05A3_MS_1)
varPlot(y~site/day, dataEP05A3_MS_1,
        BG=list(var="site", col=paste0("gray", c(100, 80, 60))),
        Points=list(pch=16, col=asRGB("black", .5), cex=1.25),
        MeanLine=list(var=c("int", "site"), col=c("black", "orange"), lwd=c(2,2)),
        Mean=list(col="cyan", cex=1.25, lwd=2), las=1,
        YLabel=list(text="Concentation [mg/dL]", las=0, line=3, font=2, cex=1.25),
        Title=list(main="Multi-Site Evaluation on dataEP05A3_MS_1", cex.main=1.75),
        VCnam=list(font=2, cex=1.5),
        VarLab=list(list(cex=1.5, font=2), list(cex=1.25, font=2)))
```



The model itself can be fitted to the data using following code (now using REML).

```
# fit 3 x 5 x 1 x 5 model to data
fit.MS1 <- fitVCA(y~site/day, dataEP05A3_MS_1, method="REML")
fit.MS1
```

```
##
##
## Result Variance Component Analysis:
## ------------------------------------
##
##   Name      DF         VC       %Total     SD       CV[%]    Var(VC)
## 1 total     16.889879 3.635232 100        1.906629 7.566293 1.564832
## 2 site      1.488641  1.017401 27.987245  1.008663 4.002794 1.390671
## 3 site:day  2.105837  0.345882 9.514705   0.588117 2.333892 0.113621
## 4 error     60        2.271949 62.49805   1.507299 5.981587 0.172058
##
## Mean: 25.19898 (N = 75)
##
## Experimental Design: balanced  |  Method: REML
```

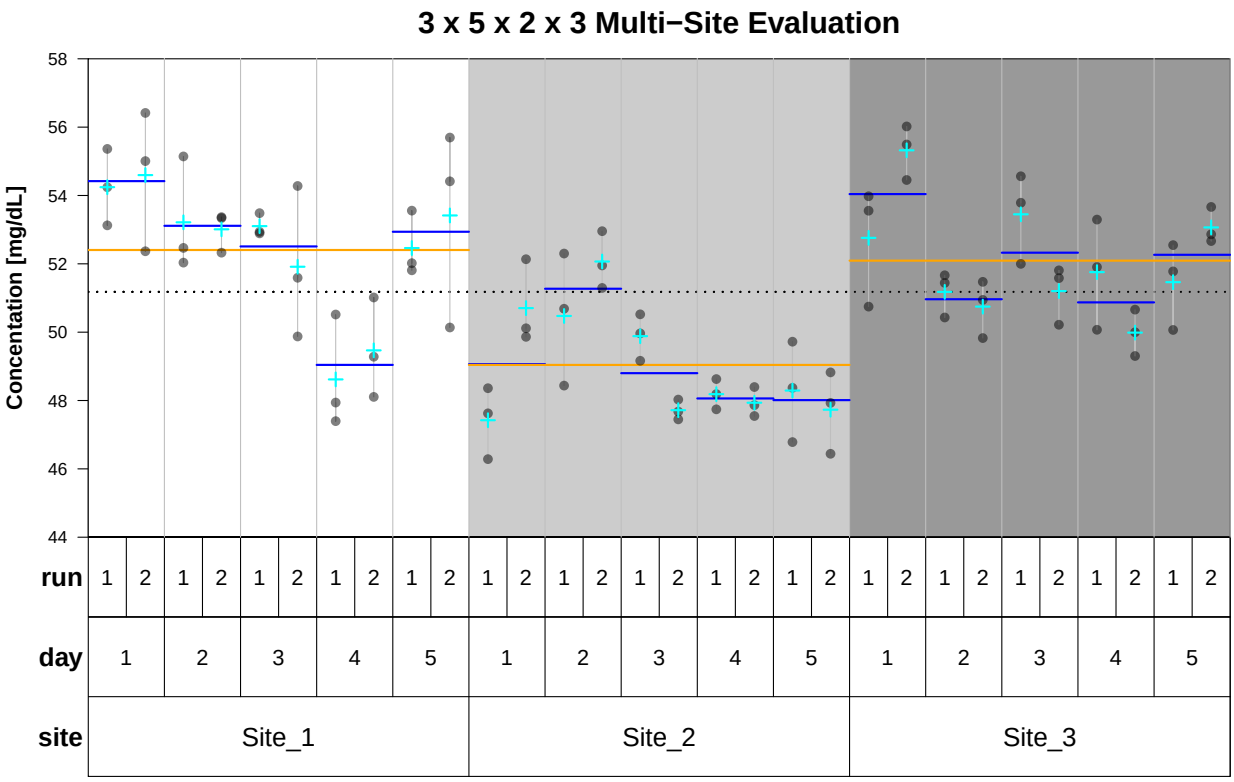
6.3 3 x 5 x 2 x 3 Multi-Site Evaluation

CLSI EP05-A3 describes a second type of multi-site evaluation which takes into account variability between-runs additionally. This model requires 15 additional observations compared to the 3 x 5 x 1 x 5 model and is the 3 x 5 x 2 x 3 model. Thus, there are 2 runs per day with 3 replicates each resulting in 90 observations overall. Since there is no such data set contained in R-package **VCA** we will simulate such data, plot it and fit the respective model.

```
# simulate fit 3 x 5 x 2 x 3 model to data
set.seed(23)
dat.MS2 <- data.frame(  y=50 +
                        # 3 random effects for sites
                        rep(rnorm(3,0,2.5), rep(30, 3)) +
                        # 15 random effects for days
                        rep(rnorm(15,0, 2), rep(6, 15)) +
                        # 30 random effects for runs
                        rep(rnorm(30,0, 1), rep(3, 30)) +
                        # residual error (repeatability)
                        rnorm(90,0,1.5),
                        site = gl(3, 30, labels=paste0("Site_", 1:3)),
                        day = gl(5, 6, 90),
                        run =gl(2, 3, 90)
                        )
```

Now visualize this data set.

```
varPlot(y~site/day/run, dat.MS2,
        BG=list(var="site", col=paste0("gray", c(100, 80, 60))),
        Points=list(pch=16, col=asRGB("black", .5), cex=1.25),
        MeanLine=list(  var=c("int", "site", "day"),
                        col=c("black", "orange", "blue"),
                        lwd=c(2,2,2)),
        Mean=list(col="cyan", cex=1.25, lwd=2), las=1,
        YLabel=list(text="Concentation [mg/dL]", las=0, line=3, font=2, cex=1.25),
        Title=list(main="3 x 5 x 2 x 3 Multi-Site Evaluation", cex.main=1.75),
        VCnam=list(font=2, cex=1.5),
        # controls for which variable vertical lines are added between levels
        # and how these are plotted
        VLine=list(var="day", col="gray75"),
        VarLab=list(list(cex=1.5), list(cex=1.25), list(cex=1.25)))
```



And finally fit the respective VCA-model using ANOVA-type estimation.

```
# fit 3 x 5 x 2 x 3 model to data (ANOVA is default)
fit.MS2 <- fitVCA(y~site/day/run, dat.MS2)
fit.MS2
```

```
##
##
## Result Variance Component Analysis:
## -----------------------------------
##
##   Name          DF       SS          MS          VC       %Total    SD
## 1 total         8.285081                         7.250308 100       2.69264
## 2 site          2        207.222526  103.611263  2.956047 40.771331 1.719316
## 3 site:day      12       179.158257  14.929855   1.837032 25.337294 1.355371
## 4 site:day:run  15       58.61496    3.907664    0.725217 10.002575 0.851597
## 5 error         60       103.92069   1.732012    1.732012 23.8888   1.316059
##   CV[%]
## 1 5.261143
## 2 3.359368
## 3 2.648257
## 4 1.663934
## 5 2.571445
##
## Mean: 51.17975 (N = 90)
##
## Experimental Design: balanced  |  Method: ANOVA
```

```
# extract original (unconstrained) VC-estimates
fit.MS2$VCoriginal
```

```
##              [,1]
## [1,] 2.956047e-04
## [2,] 1.837032e-04
## [3,] 7.252175e-05
## [4,] 1.732012e-04
```

6.4 Multi-Site Mulit-Lot Evaluation

The last model we want to address is the multi-lot reproducibility model. This refers to an experimental setup, where one of the two multi-site model is used with the additional variable reagent-lot. In in-vitro diagnostics reagent-lot is a factor where only a limited number of levels is available. Therefore, the typical multi-lot reproducibility design consists of 3 different reagent-lots used on 3 different sites. Different allocation schemes exist for assigning the 3 lots to the 3 sites. The best, but also the least parsimonious, design has all lots tested on all sites. These, among others, are valid lab-lot assignment schemes allowing to estimates VC reagent-lot:

```
##              Site_1 Site_2 Site_3
## ReagentLot_1 X      X      X
## ReagentLot_2 X      X      X
## ReagentLot_3 X      X      X
```

```
##              Site_1 Site_2 Site_3
## ReagentLot_1 X      X
## ReagentLot_2        X      X
## ReagentLot_3 X             X
```

```
##              Site_1 Site_2 Site_3
## ReagentLot_1 X      X      X
## ReagentLot_2        X      X
## ReagentLot_3               X
```

The original example data set VCAdata1 is very similar to the experimental designs described here. The sub-set *datS5* refers to an experimental design with 3 site (device), 3 reagent-lots tested on each site, 7 days per site-lot combination, and 2 runs per day with 2 replicates per run.

In previous examples in this section we could always use a fully-nested model, because the testing-days in different laboratories are independent from each other, despite the fact that they might be encoded by integers 1 to 5. The same holds for runs, 2 runs performed on 2 different days are independent from each other despite the fact that both might be encoded using the same factor-level. This independence does not hold for reagent-lots. The same lots are used in all laboratories, each lot showing lot-specific performance, i.e. each lot will have a lot-specific bias compared to the unkown true concentration of a sample. This lot-specific bias directly translates to random effects for variable lot and their variation is modelled as following a normal distribution. At least this is assumed when estimation takes place. To reflect this in the model formula site and lot have to modelled as main-effects, and all other terms are nested within combinations of all main-effects. This can be done using following expressions.

```
fit.MSML <- fitVCA(y~(device+lot)/day/run, datS5)
fit.MSML
```

```
##
##
## Result Variance Component Analysis:
## -----------------------------------
```

```
##
##   Name                DF        SS        MS        VC       %Total
## 1 total               15.178552                     0.614501 100
## 2 device              2         24.869528 12.434764 0.13791  22.442541
## 3 lot                 2         27.284252 13.642126 0.152283 24.781571
## 4 device:lot:day      58        49.32077  0.850358  0.140279 22.828111
## 5 device:lot:day:run  63        18.222272 0.289242  0.105213 17.121713
## 6 error               126       9.93085   0.078816  0.078816 12.826064
##   SD       CV[%]
## 1 0.783901 4.488937
## 2 0.371362 2.126569
## 3 0.390234 2.234642
## 4 0.374538 2.144759
## 5 0.324366 1.85745
## 6 0.280742 1.607645
##
## Mean: 17.46296 (N = 252)
##
## Experimental Design: balanced  |  Method: ANOVA
```

```
# using a wrong-model formulation
fit.MSML2 <- fitVCA(y~device/lot/day/run, datS5)
fit.MSML2
```

```
##
##
## Result Variance Component Analysis:
## -----------------------------------
##
##   Name                DF        SS        MS        VC       %Total
## 1 total               21.063478                     0.56374  100
## 2 device              2         24.869528 12.434764 0.074791 13.266926
## 3 device:lot          6         36.913949 6.152325  0.193475 34.319941
## 4 device:lot:day      54        39.691074 0.73502   0.111444 19.768759
## 5 device:lot:day:run  63        18.222272 0.289242  0.105213 18.663409
## 6 error               126       9.93085   0.078816  0.078816 13.980965
##   SD       CV[%]
## 1 0.750826 4.299536
## 2 0.273479 1.566054
## 3 0.439858 2.518807
## 4 0.333833 1.911663
## 5 0.324366 1.85745
## 6 0.280742 1.607645
##
## Mean: 17.46296 (N = 252)
##
## Experimental Design: balanced  |  Method: ANOVA
```

Above, the correct model (fit.MSML) specification results in 2 degrees of freedom (DF) for factor-variables "device" and "lot". This is intuitively right, since there are 3 levels each for both variables. Using the wrong model (fit.MSML2) leads to 6 DF for "lot" (3 x 3 - 3 = 6), which should raise concerns since the number of lots has increased all of a sudden. Otherwise, there would be no justification for having more DF than levels for a variable.

Whenever a non-standard precision experiment has to be evaluated, one should ask which factor-levels can be assumed independent from each other and which not. This is sometimes not as easy as it seems at the first

glance. If you are not 100% confident that you have specified the correct model, ask someone who can help you answering these questions.