xCELLanalyzer - Data Analysis

Processing and analysis of the impedance data generated for *xCellAnalyze: A Framework for the Analysis of Cellular Impedance Measurements for Mode of Action Discovery*

Author: Raimo Franke

N.B.: Datasets 1 to 10 were generated on xCelligence machine A and datasets 11 to 16 were generated on xCelligence machine B For each analysis of a dataset (from one 96 E-plate) three files have to be provided with the following naming convention: data_raw.txt: contains the raw data exportet from the RTCA software with the timepoint of the last measurement before compound addition pasted into the first line of the file data_anno.txt: well position - compound replicate pairs (replicate number given with ".x") data_match: compound names used in this assay

The data for this analysis can be accessed: https://github.com/raimofranke/xCELLanalyzer/tree/master/data

Load libraries

```r
library("tidyverse")
library("stringr")
library("knitr")
library("gtools")
library("gplots")
library("dendsort")
library("pheatmap")
library("RColorBrewer")
```

Source the xCELLanalyzer functions:

- **read_xcell:** This function reads the tab-delimeted data exported from the RTCA software version 1.2. If the naming conventions are followed, only the global my_filepath varible and the experiment ID are needed as arguments to the function to import the data. The cryptic column names generated by the export from the RTCA software are fixed to only contain the well identifier (of the E-plate). To match the well labels with the compound IDs an _anno.txt is read containing the annotation of the wells for the appropriate experiment. The column names are then replaced with the corresponding compound IDs.
- **edit_df:** This function takes the dataframe generated by the read_xcell-function and the experiment ID as arguments. The time point of the last measurement before compound addition logged in the RTCA software was pasted into the first row of the raw data file manually. The function takes this value and creates a tibble that only includes the last measurement before and 800 measurements after compound addition.
- **normalize_xcell:** This function performs a global normalization by dividing each cell index recorded after compound addition by the last cell index recorded before compound addition,
- **do_median_polish:** This function applies the median polish algorithm to the technical replicates and returns a dataframe with three colums giving the range of the residuals, the column effect and the sum of the residuals. Additionally, a pdf- file is generated that shows the plots of the normalized cell index over time for each set of replicates.
- **calculate_median_curves:** This function calculates median normalized cell index values for each set of technical replicates. A dataframe with the normalized median TCRPs is returned.
- **normalize_dmso:** This function performs a local normalization of each median TCRP by subtracting the normalized cell index of the DMSO control at each time point. This is done for each independent

experiment to make them more comparable and to address potential batch effects.

- **remove_dmso:** This function removes the DMSO control from the dataset after local normalization.
- **score1.function:** This function takes the dataframe with basis spline coefficients for each compound as an argument and calculates a distance matrix. The distance measure has to be provided as argument to the function. The distance matrix is sorted column-wise for each replicate and a rank sum is calculated for each replicate. A normalized score is calculated by division of the ideal score of a set of replicates by the obtained score. The closer this value is to 1 the better the reproducibility. The function returns a list with the compound names, number of replicates per group, score and normalized score.

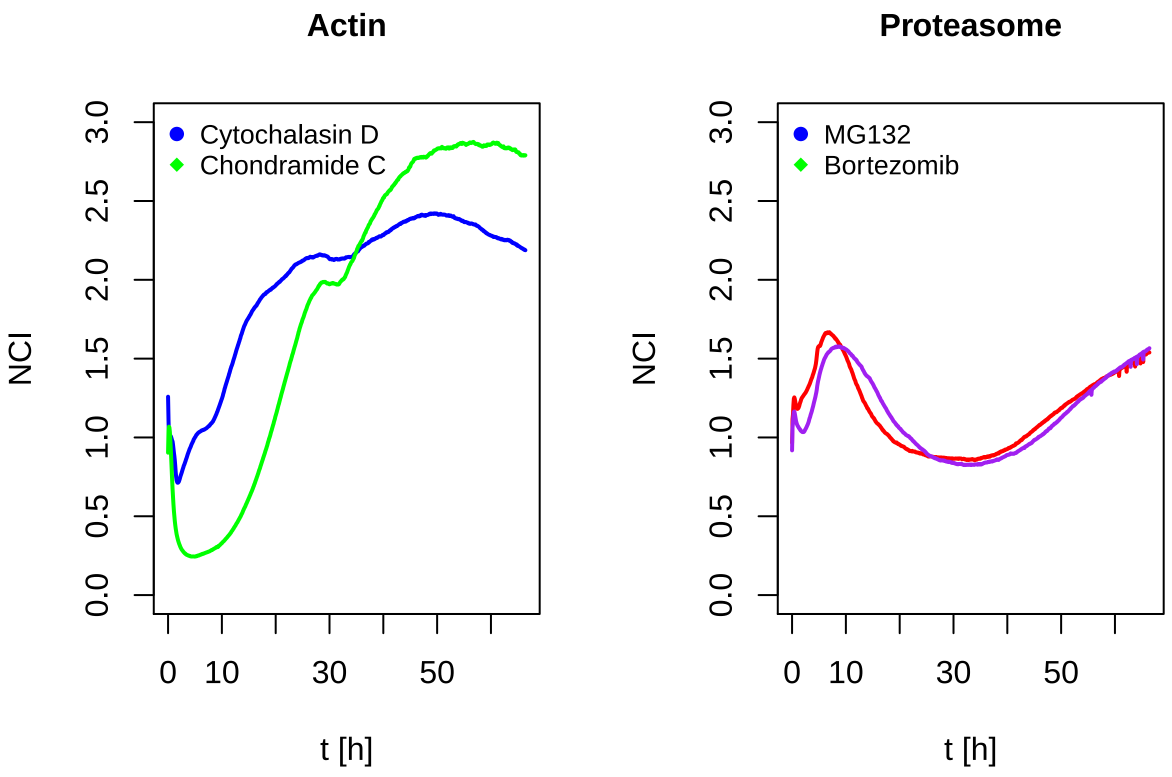
```
source("xCell_functions.R")
```

Process the 12 xCelligence runs with the xcell_process_data.R script.

The xcell_process_data.R script processes the 12 datasets in the following manner: The raw data are read and the dataframe is edited to contain only the last measurement before compound addition and 800 measurements after compound addition. Then global normalization is performed and the do_median_polish function is applied to identify outliers. Outliers with absolute residual sum of greater than 90 are removed. Median TCRPs are calculated and normalized locally by subtracting the DMSO control TCRP. A dataframe without normalization with the DMSO control is also generated for comparisons.

Plot figure 2

```
par(mfrow = c(1,2))

plot(x=rownames(xcell_median_7), y=xcell_median_7[,"7_CytochalasinD"], type="l", col="blue", lwd = 2,
     main = "Actin", cex.main=1, ylim=c(0,3), ylab="NCI", xlab="t [h]")
lines(x=rownames(xcell_median_7), y=xcell_median_7[,"7_ChondramidC"], type="l", col="green", lwd = 2)
legend("topleft",legend=c("Cytochalasin D", "Chondramide C"),
         col= c("blue", "green"),pch=c(16,18),bty="n",ncol=1,cex=0.8,pt.cex=1)

plot(x=rownames(xcell_median_7), y=xcell_median_7[,"7_MG132"], type="l", col="red", lwd = 2,
     main = "Proteasome", cex.main=1, ylim=c(0,3), ylab="NCI", xlab="t [h]")
lines(x=rownames(xcell_median_7), y=xcell_median_7[,"7_Bortezomib"], type="l", col="purple", lwd = 2)
legend("topleft",legend=c("MG132", "Bortezomib"),
       col= c("blue", "green"),pch=c(16,18),bty="n",ncol=1,cex=0.8,pt.cex=1)
```



Combine all matrices containing the normalized median TCRP data from each run in one big matrix and reorder the column names alphabetically

```
median.combined <- cbind(xcell_median_norm_1, xcell_median_norm_2, xcell_median_norm_3,
                         xcell_median_norm_4, xcell_median_norm_5, xcell_median_norm_6,
                         xcell_median_norm_7, xcell_median_norm_8, xcell_median_norm_9,
                         xcell_median_norm_10, xcell_median_norm_11, xcell_median_norm_12,
                         xcell_median_norm_13, xcell_median_norm_14, xcell_median_norm_15,
                         xcell_median_norm_16)


for (i in 1 : length(median.combined[1,])){
  temp <- unlist(strsplit(toString(colnames(median.combined)[i]), "_"))
  colnames(median.combined)[i] <- paste(temp[2], temp[1], sep="_")
}


median.combined.ordered <- median.combined[,mixedorder(colnames(median.combined))]
```

Generate an analogous matrix without the DMSO normalization

```
median.combined_notnorm <- cbind(
                         xcell_median_notnorm_1, xcell_median_notnorm_2, xcell_median_notnorm_3,
                         xcell_median_notnorm_4, xcell_median_notnorm_5, xcell_median_notnorm_6,
                         xcell_median_notnorm_7, xcell_median_notnorm_8, xcell_median_notnorm_9,
                         xcell_median_notnorm_10, xcell_median_notnorm_11, xcell_median_notnorm_12,
                         xcell_median_notnorm_13, xcell_median_notnorm_14, xcell_median_notnorm_15,
                         xcell_median_notnorm_16)
```

```r
for (i in 1 : length(median.combined_notnorm[1,])){
  temp <- unlist(strsplit(toString(colnames(median.combined_notnorm)[i]), "_"))
  colnames(median.combined_notnorm)[i] <- paste(temp[2], temp[1], sep="_")
}

median.combined_notnorm.ordered <-
  median.combined_notnorm[,mixedorder(colnames(median.combined_notnorm))]
```

Calculate smoothing splines

```r
newrownames <- read.delim("newrownames.txt", header=F, stringsAsFactors = FALSE)

#smoothing splines with new rownames
median.sp<-matrix(ncol=22, nrow=219)
row.names(median.sp)<-newrownames$V1
t<-rownames(median.combined.ordered)
t<-as.numeric(t)

i<-0
repeat{
  i<-i+1
  temp<-smooth.spline(x=t, y= median.combined.ordered[,i], nknots=20)
  median.sp[i,]<-temp$fit$coef
  if (i==219) break
}

#analogous for the data without local normalization
median.sp.notnorm<-matrix(ncol=22, nrow=219)
row.names(median.sp.notnorm)<-newrownames$V1
t<-rownames(median.combined_notnorm.ordered)
t<-as.numeric(t)

i<-0
repeat{
  i<-i+1
  temp<-smooth.spline(x=t, y= median.combined_notnorm.ordered[,i], nknots=20)
  median.sp.notnorm[i,]<-temp$fit$coef
  if (i==219) break
}
```

Calculate score for biological replicates

The goal is to evalutate reproducibility for each compound. For this purpose a rank-based score is calculated for each compound, The closer the score to one the better the reproducibility. In addition an overall score is calculated, which is a single number to judge how the experiments and the data analysis overall performed. Again it is a rank-based score, the closer the score to one the better the overall performance.

Here we also want to optimize certain parameters for the data analysis, namely the distance meassure and data scaling / centering.

Distance measures used for the dist function of R base: euclidean, maximum and manhattan. Scaling and centering of the matrices were done using the scale function of R base.

```r
#criterion how the overall procedure scored
res <- score1.function(median.sp, "euclidean")
euclidean <- sum(res$normscore)/i

median.sp.scaled <- scale(median.sp, center = FALSE, scale = TRUE)
res <- score1.function(median.sp.scaled, "euclidean")
euclidean.scaled <- sum(res$normscore)/i

median.sp.centered.scaled <- scale(median.sp, center = TRUE, scale = TRUE)
res <- score1.function(median.sp.centered.scaled, "euclidean")
euclidean.centered.scaled <- sum(res$normscore)/i
###

res <- score1.function(median.sp, "maximum")
maximum <- sum(res$normscore)/i

res <- score1.function(median.sp.scaled, "maximum")
maximum.scaled <- sum(res$normscore)/i

res <- score1.function(median.sp.centered.scaled, "maximum")
maximum.centered.scaled <- sum(res$normscore)/i
###

res <- score1.function(median.sp, "manhattan")
manhattan <- sum(res$normscore)/i

res <- score1.function(median.sp.scaled, "manhattan")
manhattan.scaled <- sum(res$normscore)/i

res <- score1.function(median.sp.centered.scaled, "manhattan")
manhattan.centered.scaled <- sum(res$normscore)/i
###

not_scaled <- c(euclidean, maximum, manhattan)
scaled <- c(euclidean.scaled, maximum.scaled, manhattan.scaled)
cent_scaled <- c(euclidean.centered.scaled, maximum.centered.scaled, manhattan.centered.scaled)

tab1 <- rbind(not_scaled, scaled, cent_scaled)
colnames(tab1) <- c("euclidean", "maximum", "manhattan")
tab1 <- as.data.frame(tab1)

#grid.table(round(tab1, 3))
kable(round(tab1, 4), caption = "scores for 5 distance measures +/- scaling and centering")
```

Table 1: scores for 5 distance measures +/- scaling and centering

	euclidean	maximum	manhattan
not_scaled	0.3780	0.4038	0.3424
scaled	0.4793	0.4729	0.4119
cent_scaled	0.4788	0.4725	0.4102

Result: Euclidean distance with scaled (not centred) data performed best.

Now we want to investigate if the local normalization with the TCRP from DMSO treated cells for each run lead to an improvement of reproducibility.

```
median.sp.notnorm.scaled <- scale(median.sp.notnorm, center = FALSE, scale = TRUE)
res <- score1.function(median.sp.notnorm.scaled, "euclidean")
euclidean.scaled.notnorm <- sum(res$normscore)/i

median.sp.scaled <- scale(median.sp, center = FALSE, scale = TRUE)
res <- score1.function(median.sp.scaled, "euclidean")
euclidean.scaled <- sum(res$normscore)/i
cat(paste0("without local normalization: ", round(euclidean.scaled.notnorm, 3),
           "\nwith local normalization: ", round(euclidean.scaled,3)))
```

```
## without local normalization: 0.274
## with local normalization: 0.479
```

Clearly the local normalization has lead to a strong improvement: 0.479 >> 0.274.

Evaluate reproducibility of biological replicates group-wise, calculate a score for each group of replicates

```
res <- score1.function(median.sp, "euclidean")
groupmatch <- read.delim("groupmatch.txt", header=F)$V1
group.score <- c()

for(i in 1:length(groupmatch)){

  ma <- grep(groupmatch[i], res$rep)
  gscore <- sum(res$normscore[ma])/length(ma)
  group.score <- c(group.score, gscore)

}
group.result <- data.frame(groupmatch,group.score)
group.result <- group.result[order(-group.result$group.score),]
kable(group.result)
```

	groupmatch	group.score
15	Chelerythrine	1.0000000
28	H89	1.0000000
48	SaframycinMx1	1.0000000
54	Staurosporine	1.0000000
60	Wortmannin	1.0000000
19	Cycloheximide	0.9250000
14	Cerulenin	0.9166667
7	Apicidin	0.8333333
57	TubulysinB	0.7857143
2	ActinomycinD	0.7555556
5	Anisomycin	0.6807359
32	Mevastatin	0.6750000
45	Rapamycin	0.6427947
47	Rhizopodin	0.5454259
21	Cytochalasin	0.5416667
23	Emetine	0.5000000
41	PD169316	0.4908789
20	CyclosporinA	0.4810458
16	ChondramidC	0.4530303
33	MG132	0.4431241

	groupmatch	group.score
44	PurvalanolA	0.4395161
4	Amanitin	0.4377358
42	Podophyllotoxin	0.4328454
11	Bortezomib	0.4243003
58	Vinblastin	0.4202786
43	Puromycin	0.4164809
29	Indirubin3monoxime	0.3968689
17	Colchicine	0.3932894
3	Alsterpaullone	0.3887535
1	A23187	0.3631470
8	Apicularen	0.3248723
34	Myriaporone	0.2915516
35	MyxothiazolA	0.2776854
50	SB203580	0.2615083
51	Scriptaid	0.2605042
56	Trichostatin	0.2388983
37	Nocodazol	0.2386498
59	Vioprolide	0.2304310
25	Etoposide	0.1995340
27	Griseofulvin	0.1931039
13	CCCP	0.1835840
49	SB202190	0.1775103
53	Soraphen	0.1771044
10	ArgyrinA	0.1745614
6	Aphidicolin	0.1530034
52	Simvastatin	0.1428155
24	EpothiloneB	0.1351025
26	GephyronicAcidA	0.1312164
38	OkadaicAcid	0.1064823
31	Methotrexate	0.1038177
12	Camptothecin	0.1002997
46	RatjadonC	0.0859606
55	Taxol	0.0708859
30	LY294002	0.0632620
22	Doxorubicin	0.0558336
18	CruentarenA	0.0545505
39	Oligomycin	0.0538105
40	Oxamflatin	0.0514741
36	Neopeltolide	0.0459717
9	ArchazolidB	0.0348333

```r
write.csv2(group.result, file = "group_results.csv")
```

What we can do now is to use the score calculated for each biological replicate and define a threshold to remove replicates which are outliers. And then calculate the groupwise scores and the overall score again to check the improvement.

```r
#Filter reference set
#normailzed score < 0.1 is defined as outlier
res <- score1.function(median.sp, "euclidean")

my_hitlist <- res$rep[res$normscore < 0.1]
```

```r
my_outliers <- c()

for (i in 1: length(my_hitlist)) {
temp <- unlist(strsplit(toString(my_hitlist[i]), "_"))
new_name <- paste0(temp[2], "_", temp[3])
my_outliers <- c(my_outliers, new_name)
}

print(my_outliers)
```

```
##  [1] "Apicularen_1"         "ArchazolidB_3"        "ArchazolidB_6"
##  [4] "ArchazolidB_13"       "ArgyrinA_4"           "Bortezomib_9"
##  [7] "Camptothecin_4"       "Camptothecin_12"      "Colchicine_16"
## [10] "CruentarenA_4"        "CruentarenA_7"        "CruentarenA_14"
## [13] "Doxorubicin_2"        "Doxorubicin_9"        "Doxorubicin_10"
## [16] "Doxorubicin_11"       "Doxorubicin_12"       "EpothiloneB_13"
## [19] "Etoposide_3"          "Etoposide_15"         "GephyronicAcidA_9"
## [22] "GephyronicAcidA_10"   "GephyronicAcidA_13"   "Indirubin3monoxime_3"
## [25] "LY294002_2"           "LY294002_5"           "LY294002_11"
## [28] "LY294002_13"          "Methotrexate_1"       "Myriaporone_16"
## [31] "Neopeltolide_2"       "Neopeltolide_6"       "Neopeltolide_16"
## [34] "Nocodazole_15"        "OkadaicAcid_2"        "Oligomycin_1"
## [37] "Oligomycin_7"         "Oligomycin_15"        "Oligomycin_16"
## [40] "Oxamflatin_4"         "Oxamflatin_8"         "Oxamflatin_16"
## [43] "PD169316_9"           "RatjadonC_2"          "RatjadonC_9"
## [46] "RatjadonC_11"         "RatjadonC_12"         "Simvastatin_9"
## [49] "Taxol_1"              "Taxol_6"              "Taxol_7"
## [52] "Taxol_13"             "Taxol_14"             "Vioprolide_10"
```

Plot the biological replicates of each compound (generated by the medians of the technical replicates on each E-plate) with and without local normalization (store as pdf).

```r
pdf(file = "median_TCRP.pdf", paper = "a4")

par(mfrow = c(2,2))
i<-0
repeat{
  i<-i+1


  ma<-grep(groupmatch[i], colnames(median.combined_notnorm.ordered))
  z<-median.combined_notnorm.ordered[,ma]
  z<-as.data.frame(z)
  z<-z[,mixedorder(names(z))]


  plot(x=rownames(z), y=z[,1], type="l", col="blue", main = "not normalized",
       cex.main=0.8, ylim=c(0,3), ylab="NCI", xlab="t [h]")



  myC <- length(ma)
  myColor <- c("green", "red", "black", "orange")

  for (n in 1:(myC-1))
  {
    lines(x=rownames(z), y=z[,n+1], type="l", col=myColor[n])
  }
  legend("topleft",legend=colnames(z),
         col= c("blue", myColor),pch=c(16,18),bty="n",ncol=1,cex=0.6,pt.cex=0.7)

  z<-median.combined.ordered[,ma]
  z<-as.data.frame(z)
  z<-z[,mixedorder(names(z))]


  plot(x=rownames(z), y=z[,1], type="l", col="blue", main = "normalized",
       cex.main=0.8, ylim=c(-1.5,1.5), ylab="NCI", xlab="t [h]")


  for (n in 1:(myC-1))
  {
    lines(x=rownames(z), y=z[,n+1], type="l", col=myColor[n])
  }
  legend("topleft",legend=colnames(z),
         col= c("blue", myColor),pch=c(16,18),bty="n",ncol=1,cex=0.6,pt.cex=0.7)





  if (i==60) break
}


dev.off()
```

```
## pdf
##   2
```

In the case where more than one biological replicate or one compound was found to be below the threshold, the one that deviates the most is removed from the data.

```r
my_outliers_selected <- c("Apicularen_1", "ArchazolidB_13", "ArgyrinA_4", "Camptothecin_4", "Colchicine

ma <- match(my_outliers_selected, colnames(median.combined.ordered))
median.combined.edited <- median.combined.ordered[, -ma]


#calculate cubic smoothing splines

median.sp.edited<-matrix(ncol=22, nrow=192)
row.names(median.sp.edited)<-newrownames$V1[-ma]
t<-rownames(median.combined.edited)
t<-as.numeric(t)

i<-0
repeat{
  i<-i+1
  temp<-smooth.spline(x=t, y= median.combined.edited[,i], nknots=20)
  median.sp.edited[i,]<-temp$fit$coef
  if (i==192) break
}



median.sp.edited.scaled <- scale(median.sp.edited, center = FALSE, scale = TRUE)
res <- score1.function(median.sp.edited.scaled, "euclidean")
euclidean.scaled <- sum(res$normscore)/i
cat(paste0("Euclidean.scaled after removal of outliers: ", round(euclidean.scaled, 3)))
```

```
## Euclidean.scaled after removal of outliers: 0.647
```

```r
group.score <- c()

for(i in 1:length(groupmatch)){

  ma <- grep(groupmatch[i], res$rep)
  gscore <- sum(res$normscore[ma])/length(ma)
  group.score <- c(group.score, gscore)

}
group.result <- data.frame(groupmatch,group.score)
group.result <- group.result[order(-group.result$group.score),]
kable(group.result)
```

	groupmatch	group.score
8	Apicularen	1.0000000
10	ArgyrinA	1.0000000
14	Cerulenin	1.0000000
15	Chelerythrine	1.0000000
17	Colchicine	1.0000000
28	H89	1.0000000
30	LY294002	1.0000000
31	Methotrexate	1.0000000
36	Neopeltolide	1.0000000
40	Oxamflatin	1.0000000
48	SaframycinMx1	1.0000000
54	Staurosporine	1.0000000
56	Trichostatin	1.0000000
57	TubulysinB	1.0000000
60	Wortmannin	1.0000000
1	A23187	0.9523810
24	EpothiloneB	0.9523810
19	Cycloheximide	0.9304348
29	Indirubin3monoxime	0.8484848
41	PD169316	0.8484848
2	ActinomycinD	0.8333333
59	Vioprolide	0.8333333
5	Anisomycin	0.8320261
7	Apicidin	0.8055556
20	CyclosporinA	0.7777778
21	Cytochalasin	0.7619048
4	Amanitin	0.7023810
45	Rapamycin	0.6944444

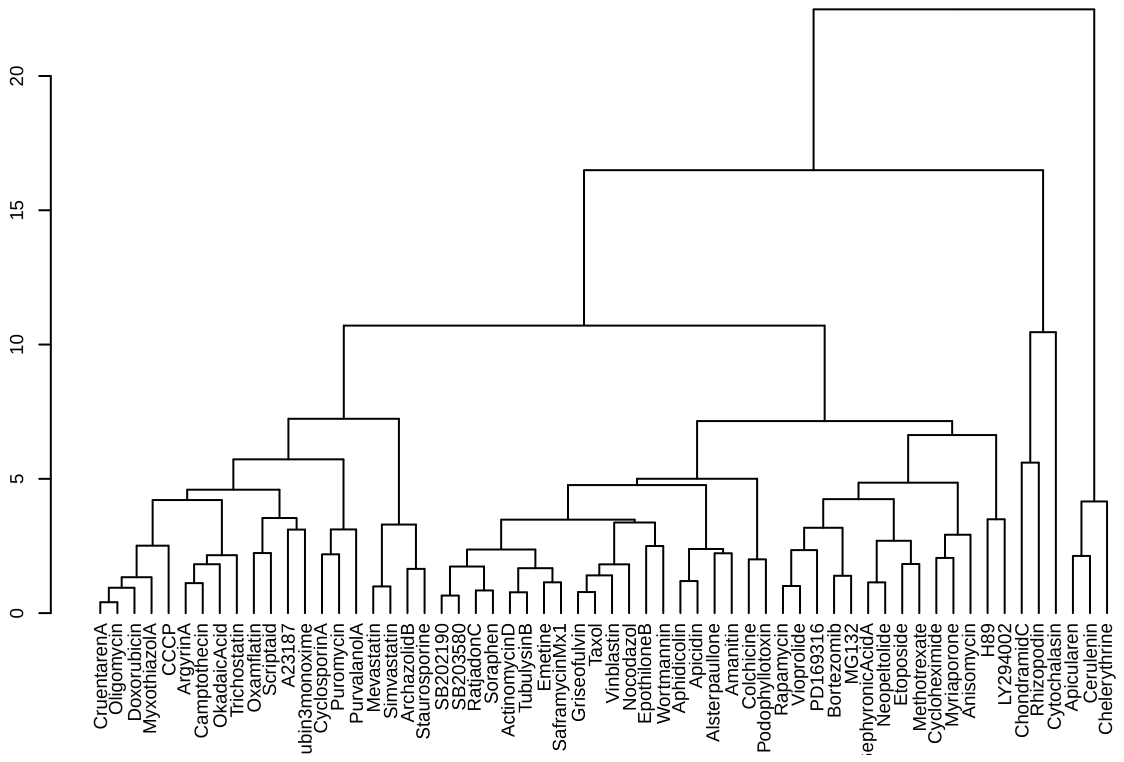
	groupmatch	group.score
3	Alsterpaullone	0.6835017
42	Podophyllotoxin	0.6631485
51	Scriptaid	0.6395604
50	SB203580	0.6392857
18	CruentarenA	0.6250000
43	Puromycin	0.6110276
16	ChondramidC	0.5865801
52	Simvastatin	0.5578866
11	Bortezomib	0.5544890
32	Mevastatin	0.5500000
37	Nocodazol	0.5454981
58	Vinblastin	0.5440476
44	PurvalanolA	0.5438596
38	OkadaicAcid	0.5435235
23	Emetine	0.5158730
12	Camptothecin	0.5142857
47	Rhizopodin	0.4989508
25	Etoposide	0.4779202
33	MG132	0.4687747
49	SB202190	0.3631785
6	Aphidicolin	0.3492424
27	Griseofulvin	0.3425232
53	Soraphen	0.3399933
34	Myriaporone	0.3336412
55	Taxol	0.3334500
35	MyxothiazolA	0.2868851
13	CCCP	0.2359447
22	Doxorubicin	0.1784604
9	ArchazolidB	0.1758621
26	GephyronicAcidA	0.1675484
46	RatjadonC	0.1181932
39	Oligomycin	0.0831636

```r
write.csv2(group.result, "group_result_filter.csv")
```

With the improved data the medians of the biological replicates is calculated.

```r
#Calculate medians of medians
median.combined.median<-matrix(ncol=60, nrow=800)
colnames(median.combined.median)<-groupmatch
rownames(median.combined.median)<-row.names(median.combined.edited)
i<-0
repeat{
  i<-i+1

  name<-groupmatch[i]
  ma<-grep(groupmatch[i], colnames(median.combined.edited))
  z<-median.combined.edited[,ma]
  median.combined.median[,i]<-apply(z, 1, median)

  if (i==60) break
}
```

```r
#smoothing splines for median.combined
xmedian.combined<-matrix(ncol=22, nrow=60)
row.names(xmedian.combined)<-colnames(median.combined.median)
t<-rownames(median.combined.median)
t<-as.numeric(t)


i<-0
repeat{
  i<-i+1
  temp<-smooth.spline(x=t, y= median.combined.median[,i], nknots=20)
  xmedian.combined[i,]<-temp$fit$coef
  if (i==60) break
}


#Scaling
xmedian.combined.scaled <- scale(xmedian.combined, scale = TRUE, center = FALSE)
colnames(xmedian.combined.scaled) <- c("c1", "c2", "c3", "c4", "c5", "c6", "c7", "c8", "c9",
                                       "c10", "c11", "c12", "c13", "c14", "c15", "c16", "c17",
                                       "c18", "c19", "c20", "c21", "c22")

##distmat and hierarchical clustering
xmedian.combined.distmat <- dist(xmedian.combined.scaled, method = "euclidean")
xmedian.hclust.sorted <- dendsort(hclust(xmedian.combined.distmat, method = "complete"))
par(cex = 0.6)
plot(as.dendrogram(xmedian.hclust.sorted))
```



```r
#heatmap
my_color = colorRampPalette(rev(brewer.pal(n = 10, name = "RdYlBu")))(100)
#pheatmap(xmedian.combined.scaled, cluster_rows = xmedian.hclust.sorted, cluster_cols = TRUE,
#         color = my_color, fontsize = 5.0)
```

Rank-based MoA prediction

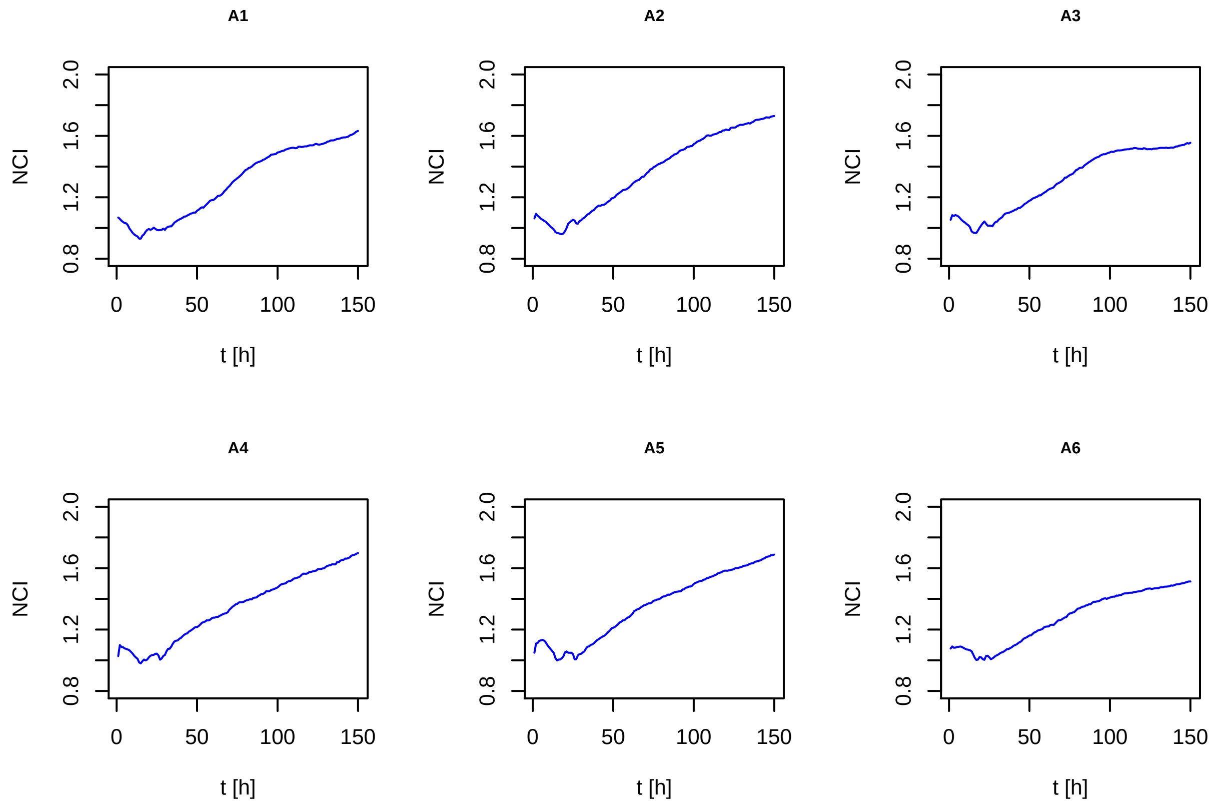
```r
xmedian.combined.scaled <- scale(xmedian.combined)
mydistmat <- dist(xmedian.combined.scaled, method = "euclidean")
mydistmat <- as.matrix(mydistmat)
rank.predict <- matrix(ncol=60, nrow=60)
colnames(rank.predict) <- colnames(mydistmat)
rownames(rank.predict) <- c(1:60)

for (i in 1:60){
mydistmat.ordered <- mydistmat[order(mydistmat[,i]),]
rank.predict[,i] <- rownames(mydistmat.ordered)
}

rank <- 1:59
rank.predict <- rank.predict[-1,]
rank.predict <- as.data.frame(cbind(rank, rank.predict))


write.csv2(rank.predict, "rankpredict.csv")
#rank.predict
```

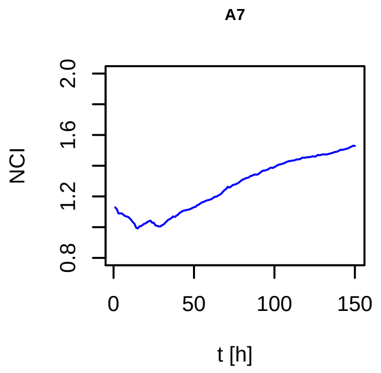
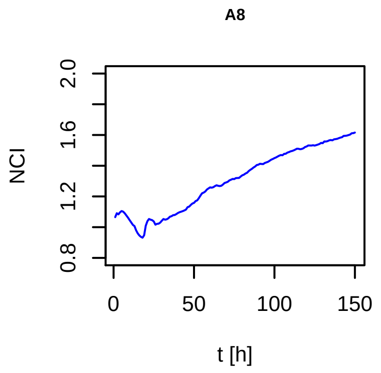
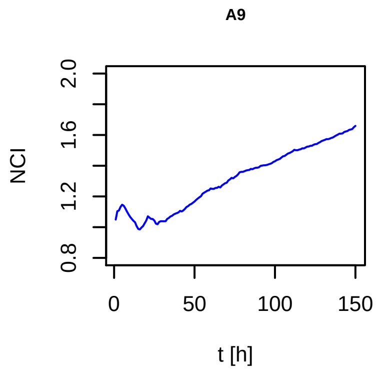
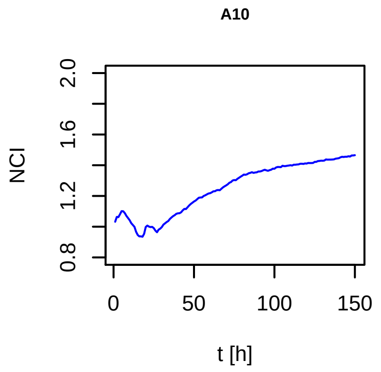
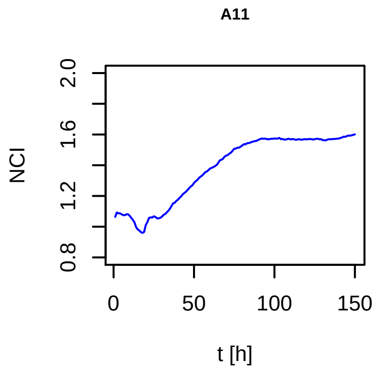
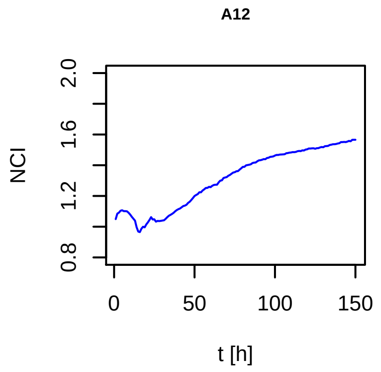
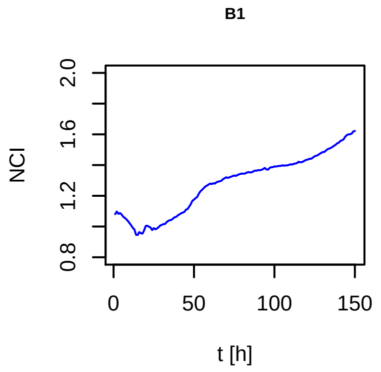
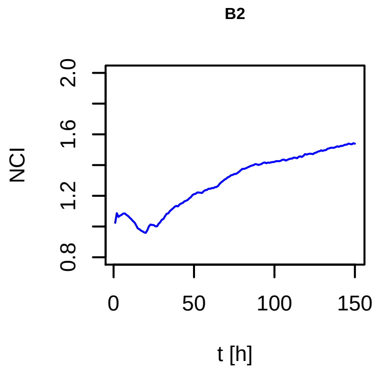
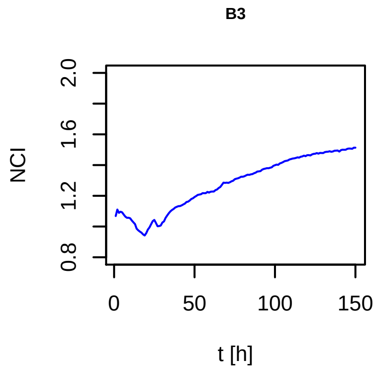
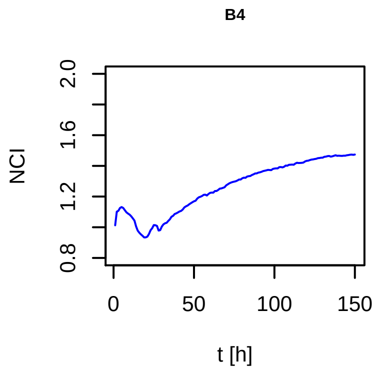
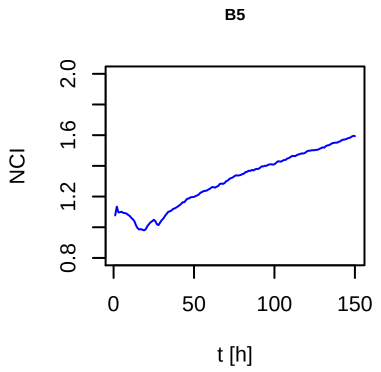
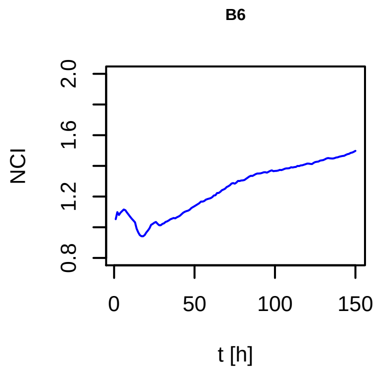
DMSO pilot test for figure 1

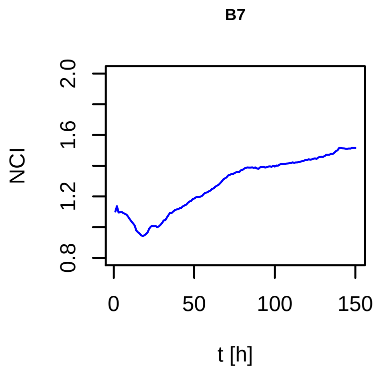
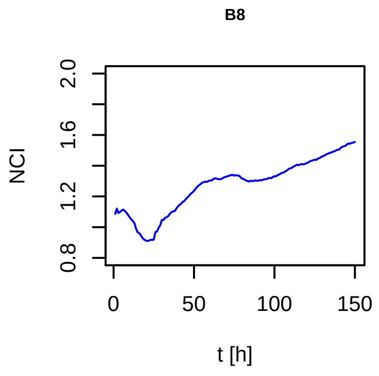
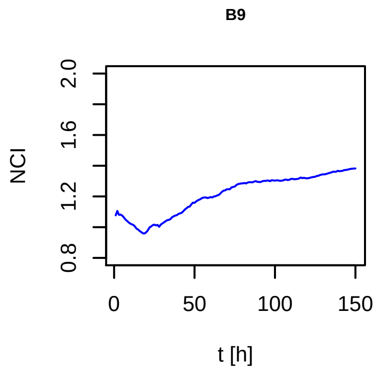
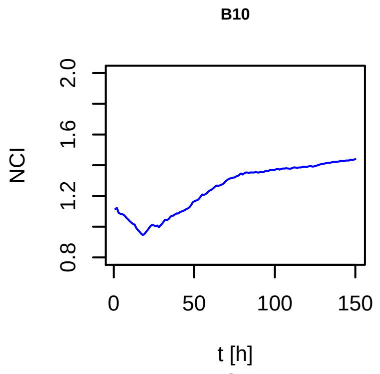
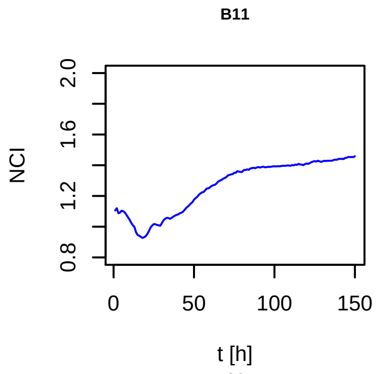
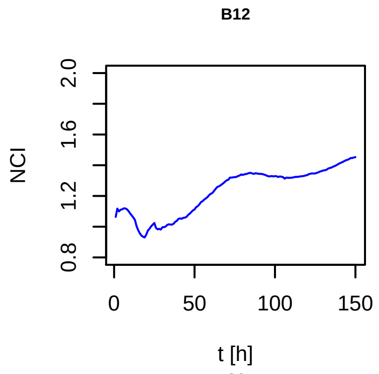
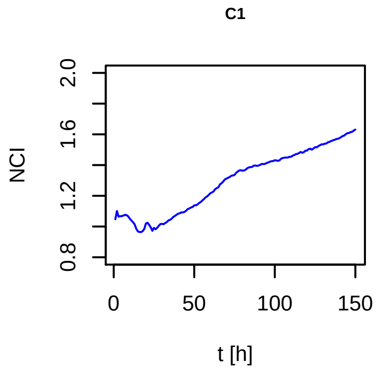
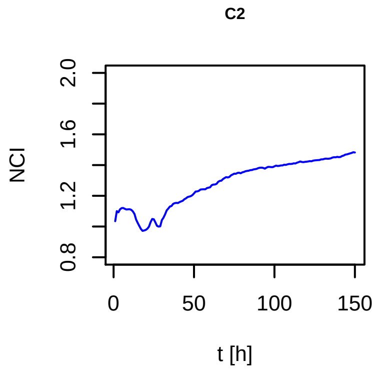
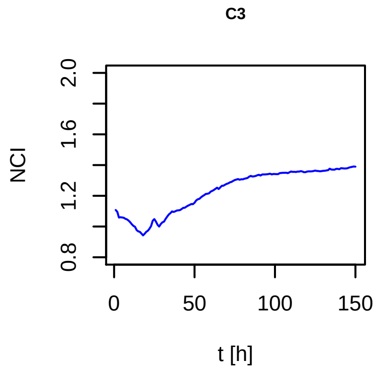
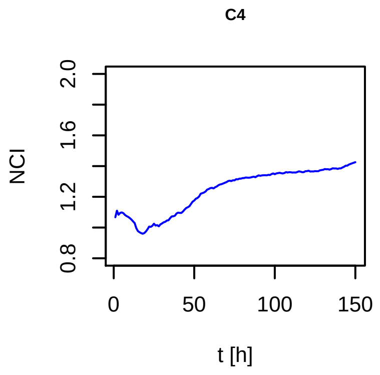
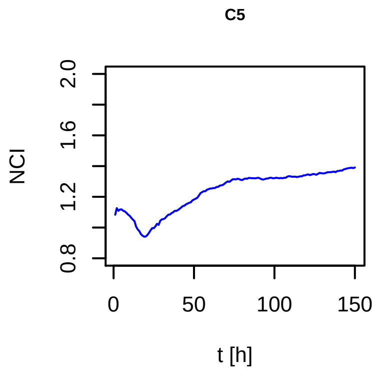
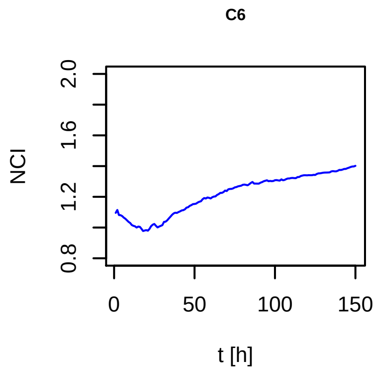
```r
#working directory
setwd("./DMSO_test/")

#compounds
DMSO_test.raw<-read.csv2(file="DMSO_test_raw.csv", header=T)


###############################################
#Normalization
x<-as.matrix(DMSO_test.raw[,2:97])
norm<-x[1,]
norm<-as.vector(norm)
DMSO_test.norm<-x/rep(norm, each = nrow(x))
DMSO_test.norm<-DMSO_test.norm[2:163,] #remove first row (last measurement before compound addition)



#set measurement
DMSO_test.norm<-DMSO_test.norm[1:150,]
DMSO_test.norm <- as.data.frame(DMSO_test.norm)

#boxplot
postscript("figure_1.eps", width = 860, height = 600)
par(mar=c(5,3,2,2)+0.1)
boxplot(DMSO_test.norm,  ylab = "NCI", xlab = "well position", cex.axis=0.4, las=2, col = "lightgray")
dev.off()
```

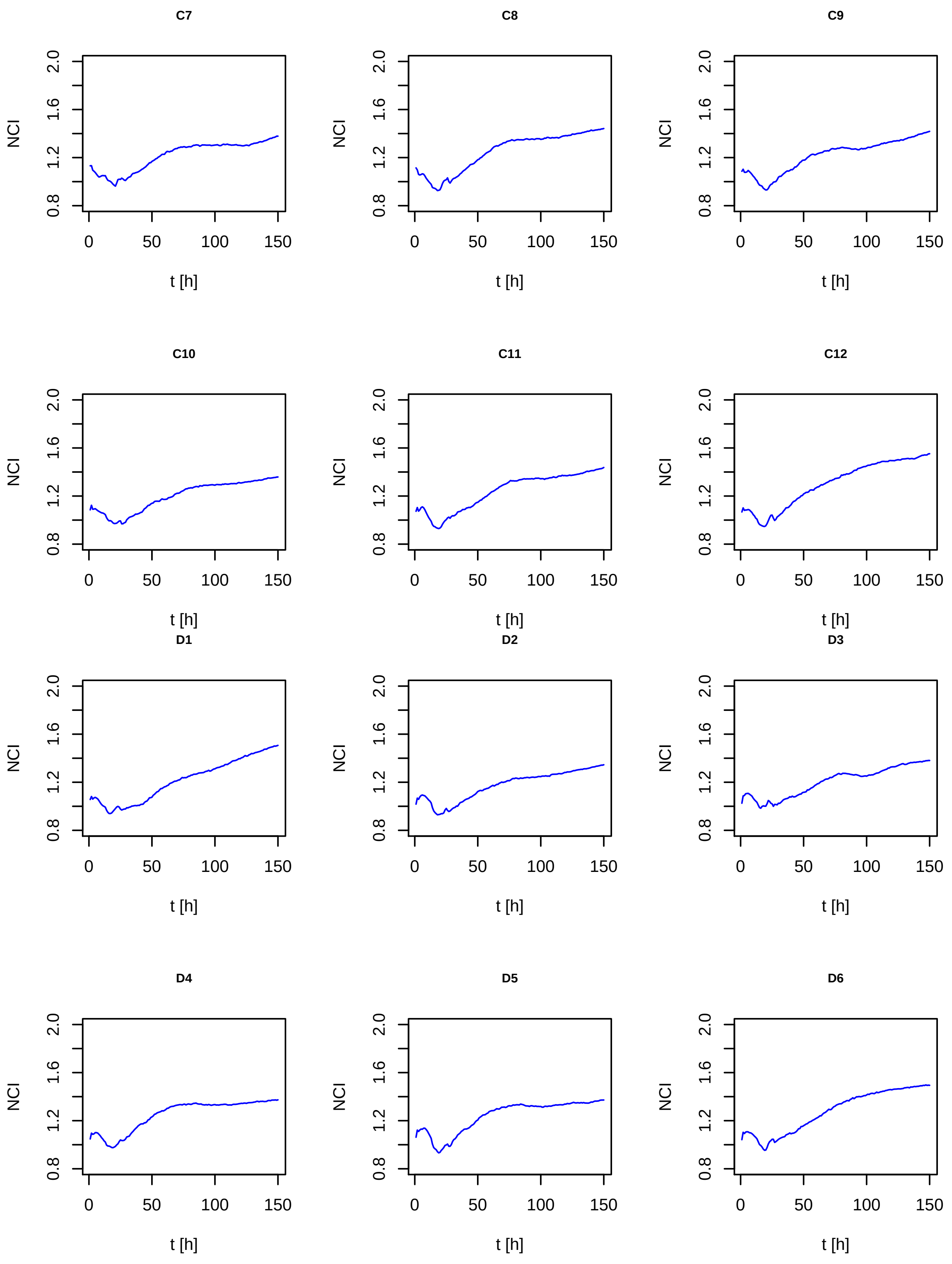
```
## pdf
##   2
```

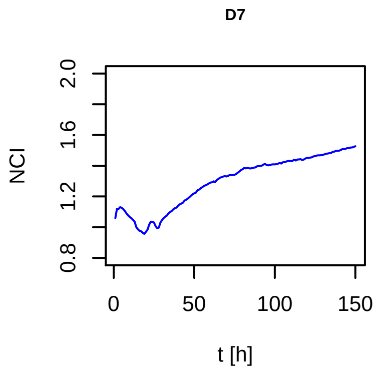
```
# plot the TCRPs
my_timepoints <- DMSO_test.raw[2:151,]$t - DMSO_test.raw[2,]$t
rownames(DMSO_test.norm) <- my_timepoints

#pdf(file = "DMSO_controls.pdf", paper = "a4r")
par(mfrow = c(2,3))
for (i in 1: ncol(DMSO_test.norm)){
plot(DMSO_test.norm[,i], type="l", col="blue",
     main=colnames(DMSO_test.norm)[i],cex.main=0.8, ylim=c(0.8,2), ylab="NCI", xlab="t [h]")
}
```

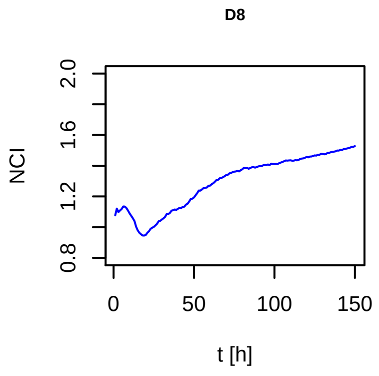


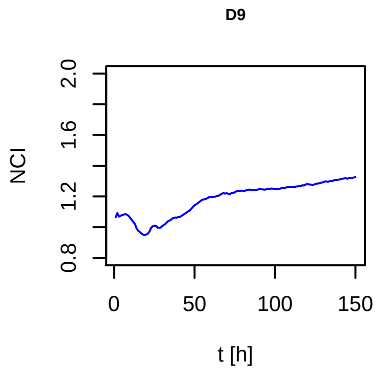

A7
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

A8
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

A9
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

A10
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

A11
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

A12
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

B1
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

B2
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

B3
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

B4
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

B5
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

B6
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

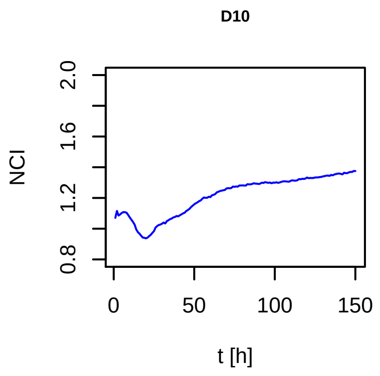

B7
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

B8
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

B9
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

B10
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

B11
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

B12
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

C1
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

C2
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

C3
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

C4
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

C5
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

C6
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

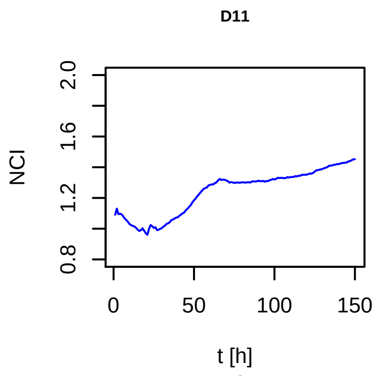

C7
C8
C9
C10
C11
C12
D1
D2
D3
D4
D5
D6
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

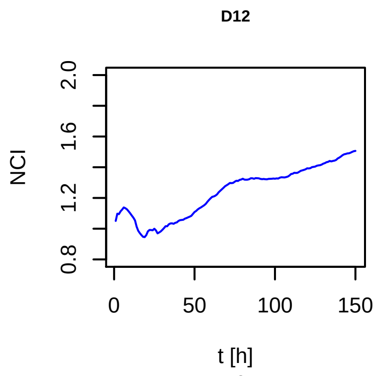

D7
NCI
t [h]

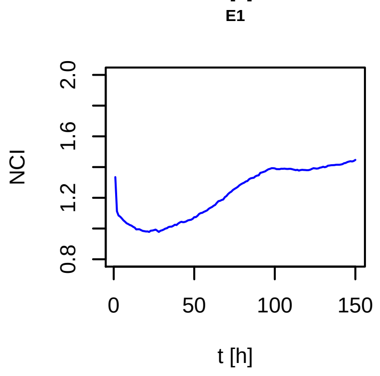

D8
NCI
t [h]

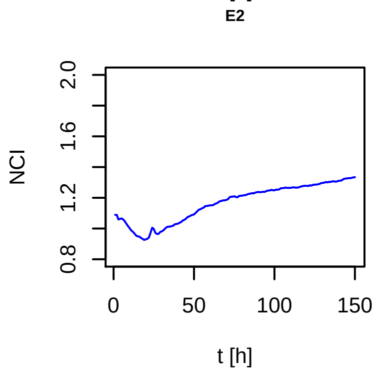

D9
NCI
t [h]

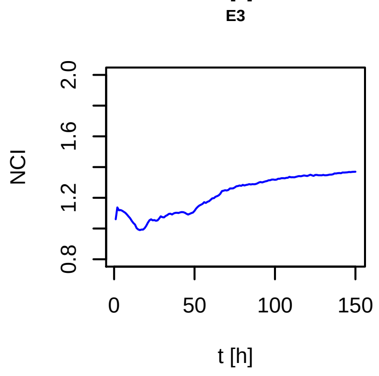

D10
NCI
t [h]

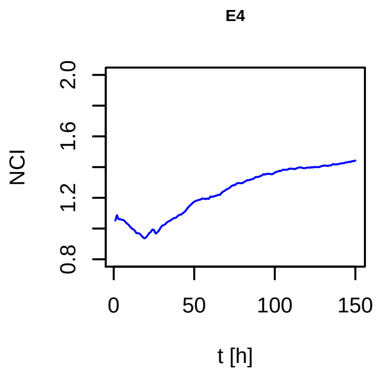

D11
NCI
t [h]

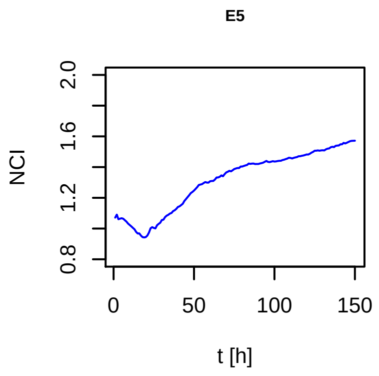

D12
NCI
t [h]

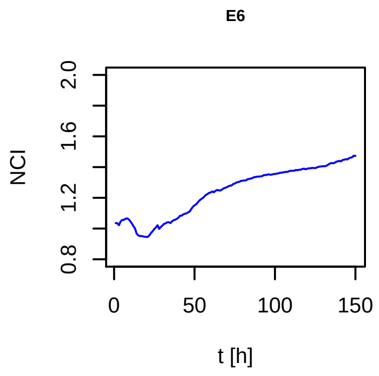

E1
NCI
t [h]

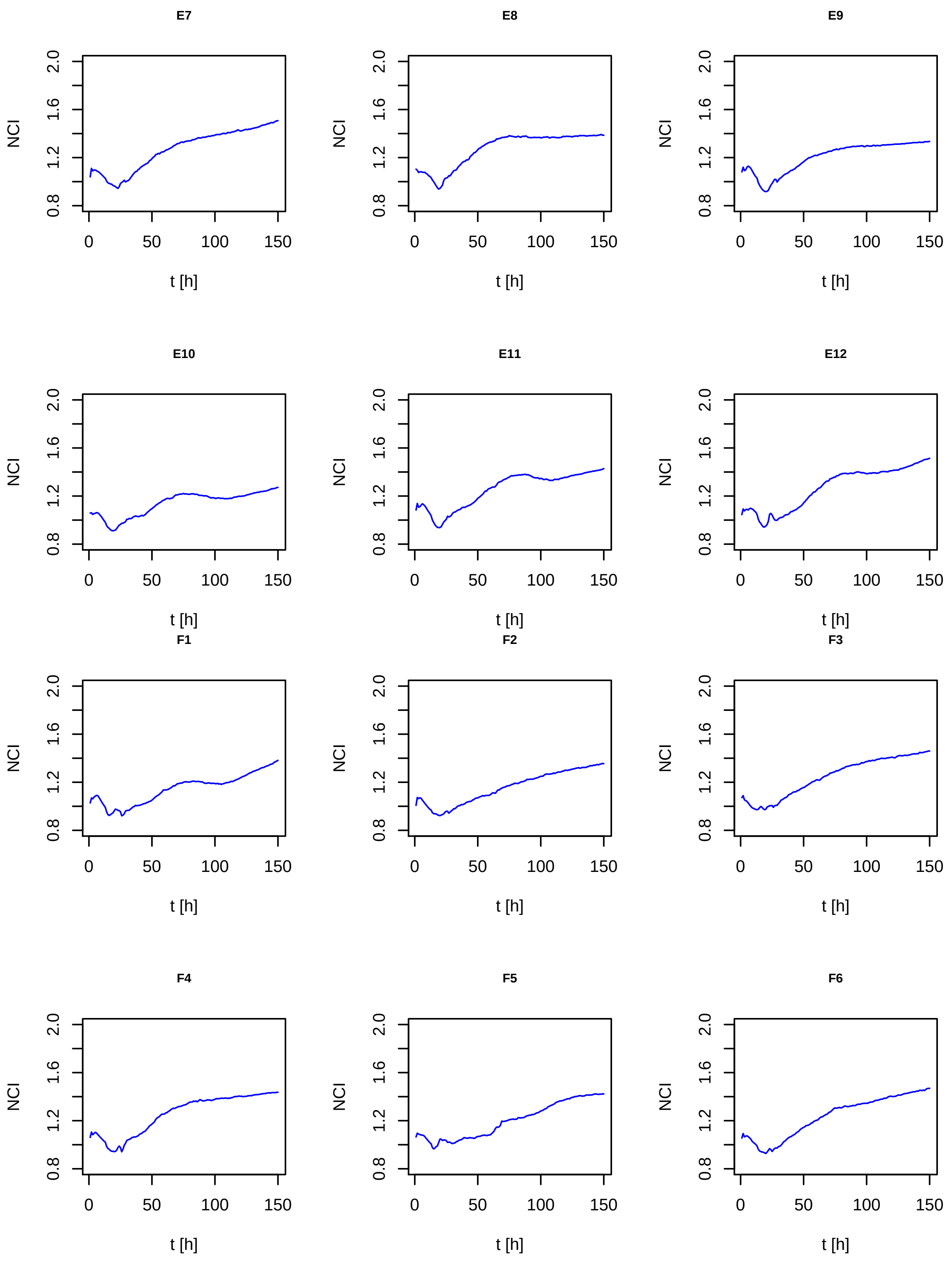

E2
NCI
t [h]

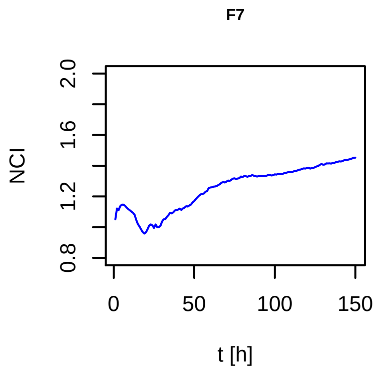

E3
NCI
t [h]

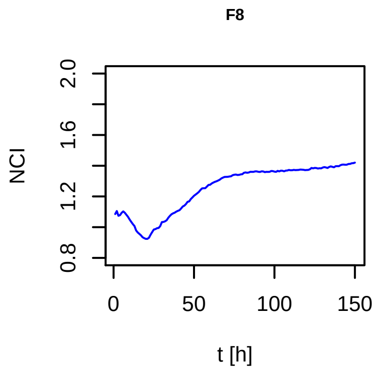

E4
NCI
t [h]

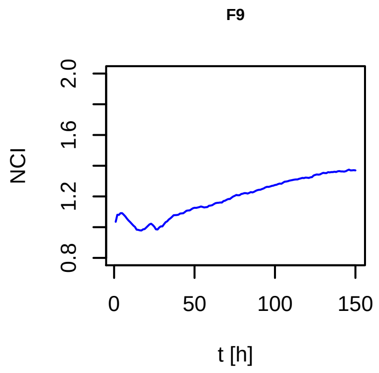

E5
NCI
t [h]

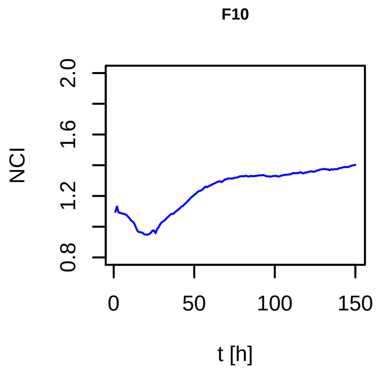

E6
NCI
t [h]

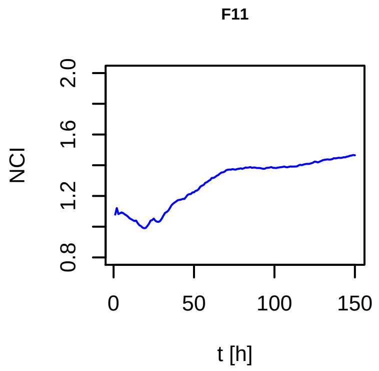

E7
E8
E9
E10
E11
E12
F1
F2
F3
F4
F5
F6
NCI
0.8
1.2
1.6
2.0
t [h]
0
50
100
150

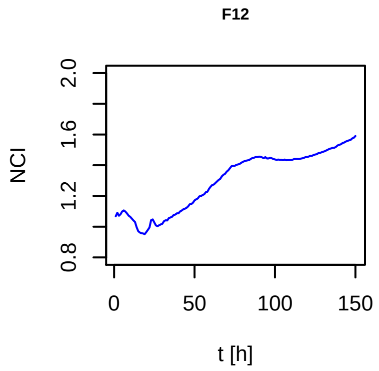

F7
NCI
t [h]

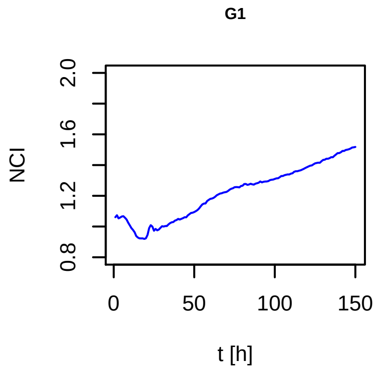

F8
NCI
t [h]

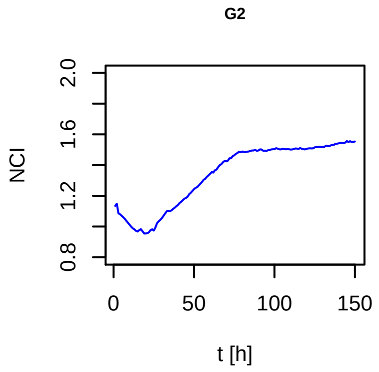

F9
NCI
t [h]

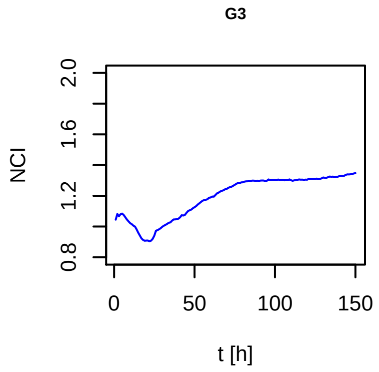

F10
NCI
t [h]

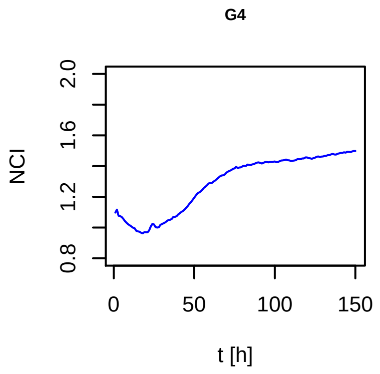

F11
NCI
t [h]

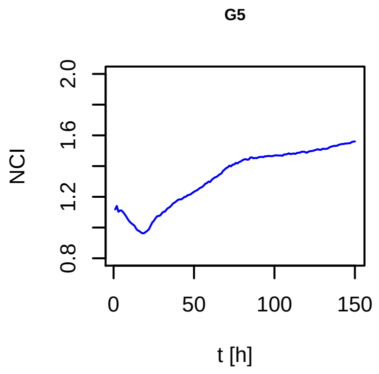

F12
NCI
t [h]

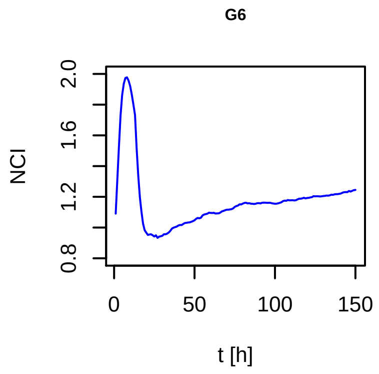

G1
NCI
t [h]

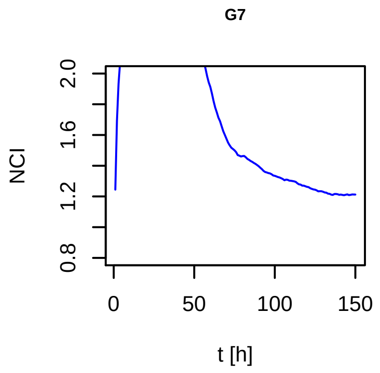

G2
NCI
t [h]

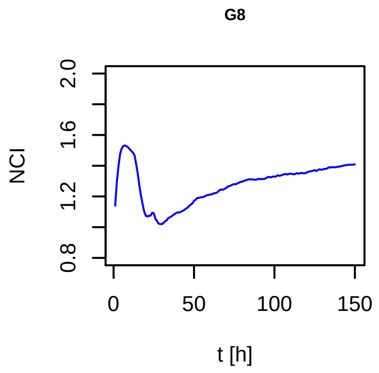

G3
NCI
t [h]

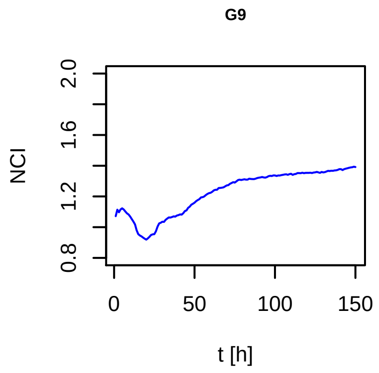

G4
NCI
t [h]

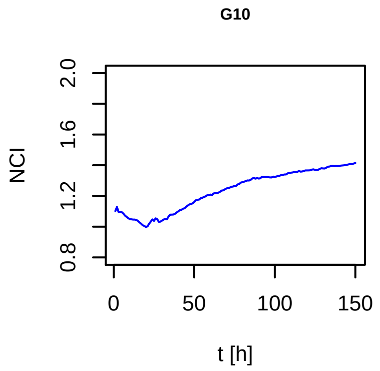

G5
NCI
t [h]

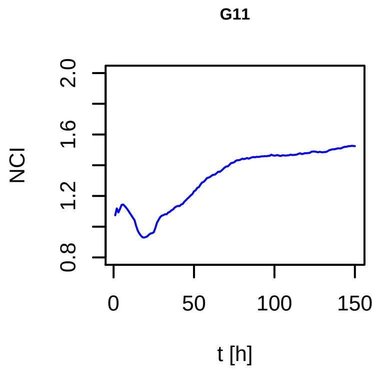

G6
NCI
t [h]

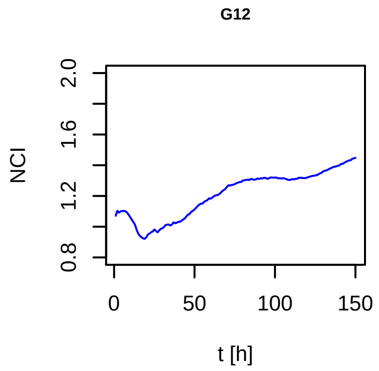

G7
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

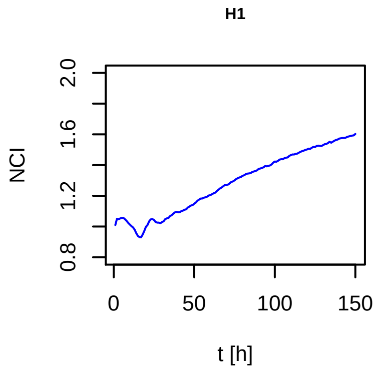

G8
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

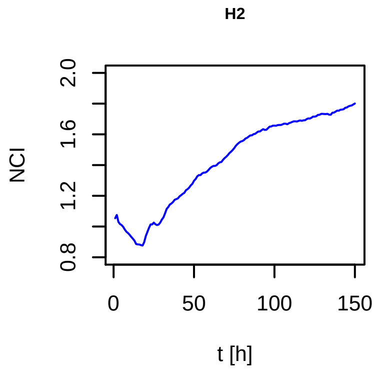

G9
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

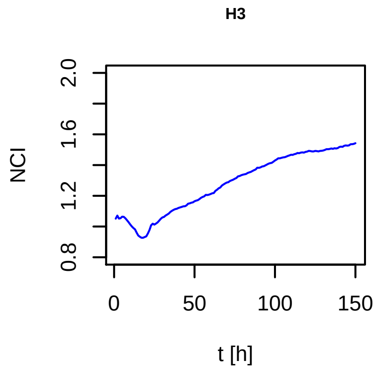

G10
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

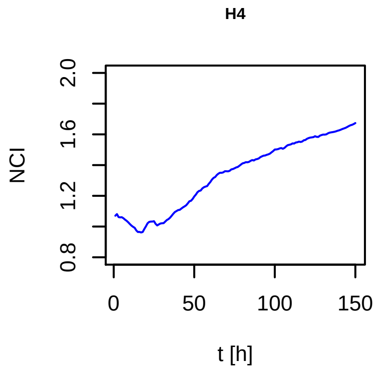

G11
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

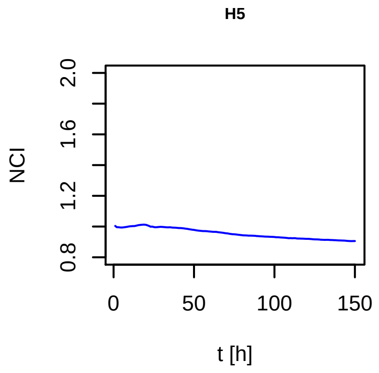

G12
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

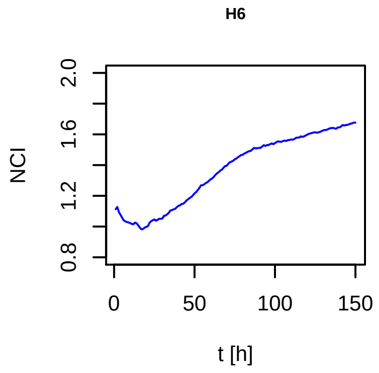

H1
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]

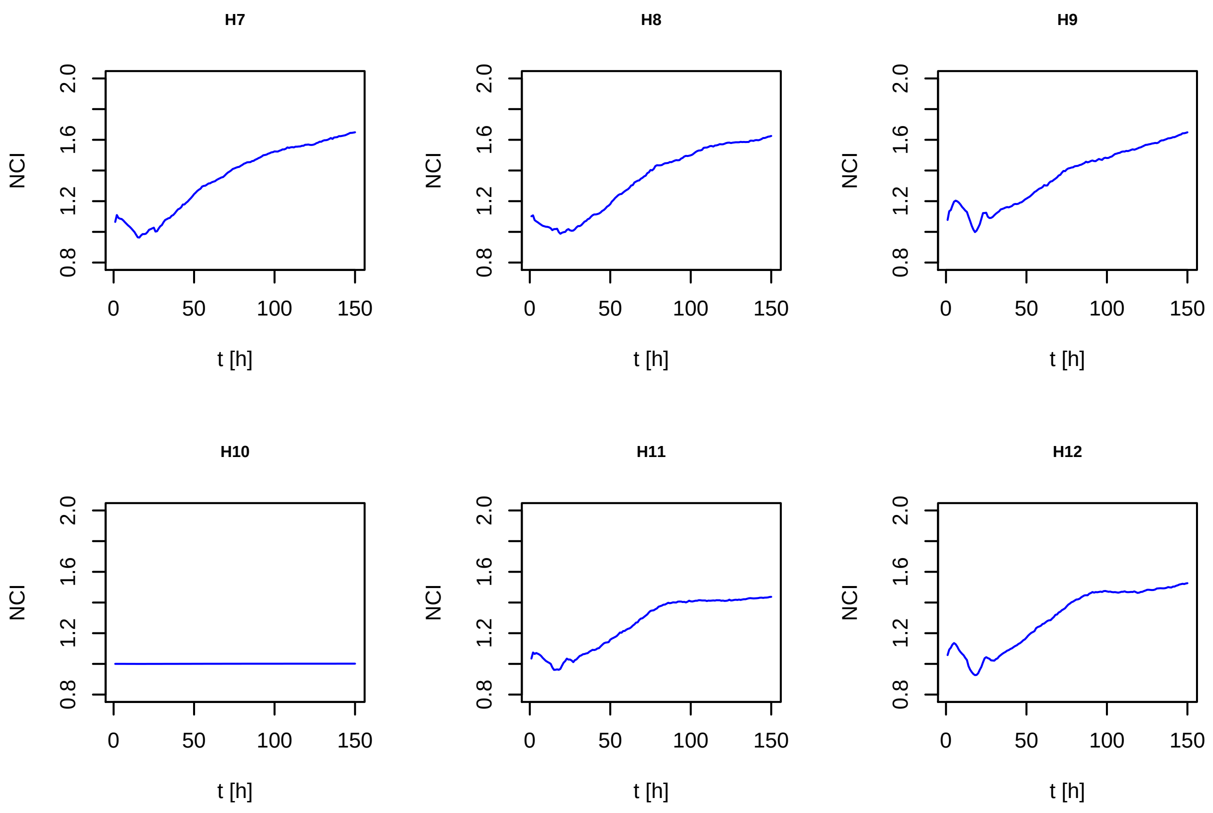

H2
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]


H3
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]


H4
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]


H5
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]


H6
NCI
0.8
1.2
1.6
2.0
0
50
100
150
t [h]



```
#dev.off()

#calculate medians
my_matrix <- as.matrix(DMSO_test.norm)
col_medians <- apply(my_matrix, 2, median)

#wilcox test
col_medians["G7"]<- NA
col_medians["H2"] <- NA
col_medians["H5"] <- NA
col_medians["H10"] <- NA

row_A <- col_medians[1:12]
row_B <- col_medians[13:24]
row_C <- col_medians[25:36]
row_D <- col_medians[37:48]
row_E <- col_medians[49:60]
row_F <- col_medians[61:72]
row_G <- col_medians[73:84]
row_G["G6"] <- NA
row_G["G7"] <- NA
row_H <- col_medians[85:96]
row_H["H5"] <- NA
row_H["H10"] <- NA

outer_rows <- c(row_A, row_H)
inner_rows <- c(row_B, row_C, row_D, row_E, row_F, row_G)
median(outer_rows, na.rm = T)
```

```
## [1] 1.357712
```

```r
median(inner_rows, na.rm = T)
```

```
## [1] 1.310951
```

```r
wilcox.test(outer_rows, inner_rows, na.rm = T, alternative = "two.sided")
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  outer_rows and inner_rows
## W = 1122, p-value = 0.0002719
## alternative hypothesis: true location shift is not equal to 0
```