# BRCA2 variant classification by a high-throughput functional assay

*Masachika Ikegami*

*May 30, 2019*

## Contents

# 1 Setting up

## 1.1 Loading necessary packages

```
knitr::opts_chunk$set(message=FALSE)
knitr::opts_chunk$set(tidy=TRUE)
knitr::opts_chunk$set(warning=FALSE)
knitr::opts_chunk$set(error=FALSE)
knitr::opts_chunk$set(fig.width=6, fig.height=6,
    fig.align='center', fig.pos='H', out.extra='',
    tidy.opts = list(width.cutoff = 45))
knitr::opts_knit$set(root.dir="~/MANO")
setwd("~/MANO")


options(scipen=100)

library(knitr)
library(ggplot2)
library(ggmcmc)
library(rstan)
library(mclust)
library(tidyverse)
library(reshape2)
library(ggsci)
library(pROC)
library(car)
library(shinystan)
```

## 1.2 Checking the package versions

```
paste("ggplot2 version:", packageVersion("ggplot2"))
```

```
## [1] "ggplot2 version: 3.1.1"
```

```
paste("ggmcmc version:", packageVersion("ggmcmc"))
```

```
## [1] "ggmcmc version: 1.2"
```

```
paste("rstan version:", packageVersion("rstan"))
```

```
## [1] "rstan version: 2.18.2"
```

```
paste("mclust version:", packageVersion("mclust"))
```

```
## [1] "mclust version: 5.4.3"
```

```
paste("tidyverse version:", packageVersion("tidyverse"))
```

```
## [1] "tidyverse version: 1.2.1"
```

```
paste("reshape2 version:", packageVersion("reshape2"))
```

```
## [1] "reshape2 version: 1.4.3"
```

```
paste("ggsci version:", packageVersion("ggsci"))
```

```
## [1] "ggsci version: 2.9"
```

```
paste("pROC version:", packageVersion("pROC"))
```

```
## [1] "pROC version: 1.14.0"
```

```
paste("car version:", packageVersion("car"))
```

```
## [1] "car version: 3.0.3"
```

```
paste("shinystan version:", packageVersion("shinystan"))
```

```
## [1] "shinystan version: 2.5.0"
```

### 1.3 Multi-core setting: optional

Only when using a multi-core processor.

```
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
```

# 2 Processing barcode read numbers

Every functional assay was performed with 2 standards composed of wild-type and D2723H variants of BRCA2 as positive and negative controls. The ratio of cell number with each BRCA2 variant was calculated with barcode read numbers obtained from the next-generation sequencer. The ratio of every experiments was normalized by dividing by that of drugless culture. The normalized viability values was transformed into logarithmic scale with a base of 10, and then rescaled into log10(1.0) for wild-type and log10(0.003) for D2723H. The average and standard error across all replicated experiments in each batch were calculated.

## 2.1 Logarithmic transformation of barcode read numbers

Load the barcode-read-summary file obtained from a next-generation sequencer; change the file name as appropriate before running.

```
r <- read.csv(file = "read_summary.csv")
```

Add 1 to each barcode read number for logarithmic transformation.

```
r[3:ncol(r)] <- r[3:ncol(r)] + 1
```

Sum-up of barcode read numbers from paired-end reads.

```
r_paired <- r[1:(ncol(r)/2+1)]
for(i in 3:ncol(r_paired)){
  r_paired[i] <- r[2*i-3] + r[2*i-2]
}
```

Obtain the logarithmic read ratio of each variant in each experiment.

```
r_sum <- matrix(0, 1, (ncol(r_paired)))
r_sum[3:ncol(r_sum)] <- apply(r_paired[3:ncol(r_paired)], 2, sum)
r_prop <- r_paired
for(i in 3:ncol(r_prop)){
  r_prop[[i]] <- r_prop[[i]] / r_sum[[i]]
  r_prop[i] <- log(r_prop[i], base=10)
}
```

Logarithmic viability values relative to drugless culture in each experiments.

```
r_FC <- r_prop
r_drugless <- apply(r_prop[3:5],1,mean)
for(i in 3:ncol(r_prop)){
  r_FC[i] <- r_prop[i] - r_drugless
}
```

Obtain the average and standard error of logarithmic viability values in each batch.

```
r_ave <- r_FC[1:((ncol(r_FC)+4)/3)]
r_se <- r_FC[1:((ncol(r_FC)+4)/3)]
for(i in 1:((ncol(r_FC)-2)/3)){
  r_ave[i+2] <- apply(r_FC[(3*i):(3*i+2)],1,mean)
  r_se[i+2] <- apply(r_FC[(3*i):(3*i+2)],1,sd)/sqrt(3)
}
```

Normalize and rescale the viability values to wild-type (log10(1)) & D2723H (log10(0.003)).

```
r_WT <- apply(subset(r_ave, Name == 'Wild-type')[3:ncol(r_ave)],2,mean)
r_D2723H <- apply(subset(r_ave, Name == 'D2723H')[3:ncol(r_ave)],2,mean)
```

```
r_norm <- r_FC[1:(ncol(r_FC)-3)]
for(i in 2:((ncol(r_norm)-2)/3)){
  for(j in 1:nrow(r_norm)){
    r_norm[[(3*i-3)]][j] <- (r_FC[[(3*i)]][j] - r_WT[i]) * (- log(0.003,10)) / (r_WT[i] - r_D2723H[i])
    r_norm[[(3*i-2)]][j] <- (r_FC[[(3*i+1)]][j] - r_WT[i]) * (- log(0.003,10)) / (r_WT[i] - r_D2723H[i])
    r_norm[[(3*i-1)]][j] <- (r_FC[[(3*i+2)]][j] - r_WT[i]) * (- log(0.003,10)) / (r_WT[i] - r_D2723H[i])
  }
}
```

Obtain the average and standard error of normalized logarithmic relative viability in each batch.

```
r_norm_ave <- r_norm[1:((ncol(r_FC)+1)/3)]
r_norm_se <- r_norm[1:((ncol(r_FC)+1)/3)]
for(i in 1:((ncol(r_norm)-2)/3)){
  r_norm_ave[i+2] <- apply(r_norm[(3*i):(3*i+2)],1,mean)
  r_norm_se[i+2] <- apply(r_norm[(3*i):(3*i+2)],1,sd)/sqrt(3)
}
```

Save the normalized data.

```
write.csv(r_norm, "data.csv",row.names=FALSE)
write.csv(r_ave, "data_ave.csv",row.names=FALSE)
write.csv(r_se, "data_se.csv",row.names=FALSE)
write.csv(r_norm_ave, "data_norm_ave.csv",row.names=FALSE)
write.csv(r_norm_se, "data_norm_se.csv",row.names=FALSE)
```

## 2.2 Beta distribution parameter estimation

Parameters were determined by identifying the Beta distribution whose 2.5th and 97.5th percentiles of pathogenic/intermediate component fit the upper and lower limits of Align-GVGD prediction in each class.

```
# C65: P(pathogenic) 0.81 (95% CI: 0.61-0.95); Beta(15.00,3.48)
qbeta(0.025,15.00,3.48)
```

```
## [1] 0.6104754
```

```
qbeta(0.975,15.00,3.48)
```

```
## [1] 0.9497518
```

```
# C35-55: P(pathogenic) 0.66 (95% CI: 0.34-0.93); Beta(5.38,2.57)
qbeta(0.025,5.38,2.57)
```

```
## [1] 0.3402095
```

```
qbeta(0.975,5.38,2.57)
```

```
## [1] 0.9302839
```

```
# C15-25: P(pathogenic) 0.29 (95% CI: 0.09-0.56); Beta(3.76,9.00)
qbeta(0.025,3.76,9.00)
```

```
## [1] 0.0897597
```

```
qbeta(0.975,3.76,9.00)
```

```
## [1] 0.5601259
```

```r
# C0: P(pathogenic) 0.01 (95% CI: 0.00-0.06); Beta(1.43,73.10)
qbeta(0.025,1.43,73.10)
```

```
## [1] 0.001266502
```

```r
qbeta(0.975,1.43,73.10)
```

```
## [1] 0.06008288
```

```r
# Outside key domains: P(pathogenic) 0.01 (95% CI: 0.00-0.04); Beta(1.64,120.44)
qbeta(0.025,1.64,120.44)
```

```
## [1] 0.001165912
```

```r
qbeta(0.975,1.64,120.44)
```

```
## [1] 0.04002138
```

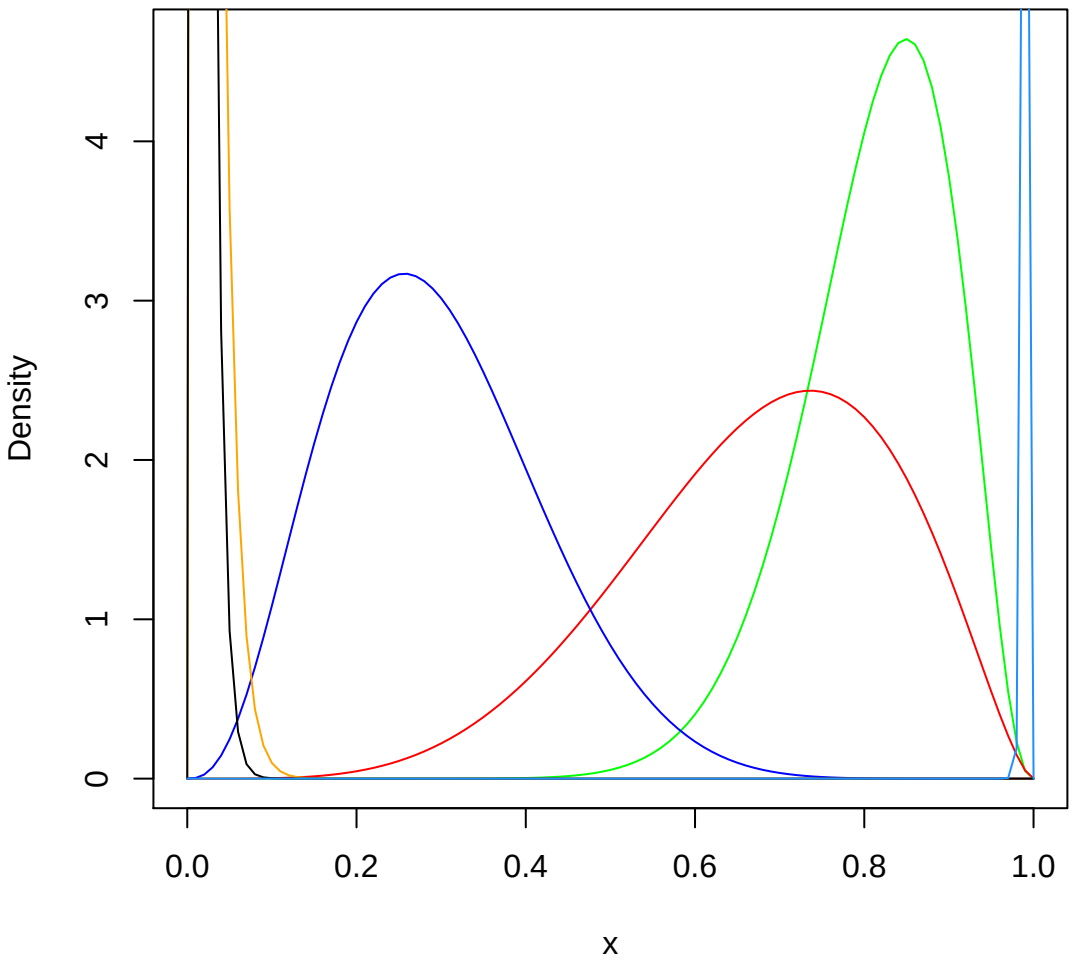
```r
# Nonsense variant: P(pathogenic) 0.995 (95% CI: 0.99-1.00); Beta(387,1.07)
qbeta(0.025,387,1.07)
```

```
## [1] 0.9901415
```

```r
qbeta(0.975,387,1.07)
```

```
## [1] 0.999914
```

```r
curve(dbeta(x,15.00,3.48),0,1,col = "green",ylab="Density") # C65
curve(dbeta(x,5.38,2.57),0,1,add=T,col = "red") # C35-55
curve(dbeta(x,3.76,9.00),0,1,add=T,col = "blue") # C15-25
curve(dbeta(x,1.43,73.10),0,1,add=T,col = "orange") # C0
curve(dbeta(x,1.64,120.44),0,1,add=T,col = "black")# Outside key domains
curve(dbeta(x,387,1.07),0,1,add=T,col = "dodgerblue") # Nonsense variant
```




### 2.3 Manual data processing

Combine the results of all experiments and add information on variants, experiments, batches and drugs to the normalized data for Hamiltonian Monte Carlo simulation. Save the data as “data_HMC.csv”.

# 3 Model fitting

## 3.1 Data loading

Load the processed data composed of normalized logarithmic fold-change from all experiments.

```
d <- read.csv(file = "data_HMC.csv", stringsAsFactors=FALSE)
Drug <- c("olaparib", "niraparib", "rucaparib", "CBDCA")
Drug_No <- length(Drug)
d_ <- NULL
v_l <- NULL
v_l2 <- NULL
v_list <- NULL
for(i in 1:Drug_No){
  d__ <- list(subset(d, drug == Drug[i]))
  d_ <- c(d_,list(subset(d__[[1]], batch <2)))
  v_l <- c(v_l,list(unique(d_[[i]]$var_name)))
  v_l2 <- c(v_l2,list(v_l[[i]][-which(v_l[[i]] %in% c("Wild-type","D2723H"))]))
  v_list <-c(v_list,list(c(c("Wild-type","D2723H"),v_l2[[i]])))
  var_num <- rep(0,length(d_[[i]]$var_name))
  var_cnt <- rep(0,length(d_[[i]]$var_name))
  score_ave <- rep(0,length(d_[[i]]$var_name))
  for(k in 1:(length(d_[[i]]$var_name))){
    for(l in 1:(length(d_[[i]]$var_name))){
      if(d_[[i]]$var_name[[k]] == d_[[i]]$var_name[[l]]){
        score_ave[k] = score_ave[k] + d_[[i]]$score[[l]]
        var_cnt[k]  = var_cnt[k] + 1
      }
    }
    score_ave[k] = score_ave[k] / var_cnt[k]
  }
  d_[[i]] <-cbind(d_[[i]],var_num,score_ave)
}
d_107 <- d_
v_list_107 <- v_list
d_ <- NULL
d___<-NULL
v_l <- NULL
v_l2 <- NULL
v_list <- NULL
IARC <- NULL
ClinVar <- NULL
aGVGD <- NULL
position <- NULL
Iclass <- NULL
for(i in 1:Drug_No){
  d__ <- list(subset(d, drug == Drug[i]))
  d___ <- c(d___,list(subset(d__[[1]], batch < 4)))
  d_ <- c(d_,list(merge(subset(d___[[i]], standard=="Class 1/2"), subset(d___[[i]], standard=="Class 4/5
  v_l <- c(v_l,list(unique(d_[[i]]$var_name)))
  v_l2 <- c(v_l2,list(v_l[[i]][-which(v_l[[i]] %in% c("Wild-type","D2723H"))]))
  v_list <-c(v_list,list(c(c("Wild-type","D2723H"),v_l2[[i]])))
  var_num <- rep(0,length(d_[[i]]$var_name))
  var_cnt <- rep(0,length(d_[[i]]$var_name))
```

```
  score_ave <- rep(0,length(d_[[i]]$var_name))
  IARC <- c(IARC, list(rep(0,length(v_l[[i]]))))
  position <- c(position, list(rep(0,length(v_l[[i]]))))
  Iclass <- c(Iclass, list(rep(0,length(v_l[[i]]))))
  aGVGD <- c(aGVGD, list(rep(0,length(v_l[[i]]))))
  for(k in 1:(length(d_[[i]]$var_name))){
    for(l in 1:(length(d_[[i]]$var_name))){
      if(d_[[i]]$var_name[[k]] == d_[[i]]$var_name[[l]]){
        score_ave[k] = score_ave[k] + d_[[i]]$score[[l]]
        var_cnt[k]  = var_cnt[k] + 1
      }
    }
    score_ave[k] = score_ave[k] / var_cnt[k]
    for(j in 1:(length(v_list[[i]]))){
      if(d_[[i]]$var_name[[k]] == v_list[[i]][j]){
        var_num[k] <- j
        IARC[[i]][j] <- d_[[i]]$standard[[k]]
        aGVGD[[i]][j] <- d_[[i]]$aGVGD_class[[k]]
        Iclass[[i]][j] <- d_[[i]]$IARC_class[[k]]
        position[[i]][j] <- d_[[i]]$position[[k]]
      }
    }
  }
  d_[[i]] <-cbind(d_[[i]],var_num,score_ave)
}
d_t <- d_
v_list_t <- v_list
IARC_t <- IARC
position_t <- position
Iclass_t <- Iclass
aGVGD_t <- aGVGD

d_ <- NULL
d___<-NULL
v_l <- NULL
v_l2 <- NULL
v_list <- NULL
IARC <- NULL
ClinVar <- NULL
aGVGD <- NULL
position <- NULL
Iclass <- NULL
for(i in 1:Drug_No){
  d__ <- list(subset(d, drug == Drug[i]))
  d_ <- c(d_,list(subset(d__[[1]], batch < 4)))
  v_l <- c(v_l,list(unique(d_[[i]]$var_name)))
  v_l2 <- c(v_l2,list(v_l[[i]][-which(v_l[[i]] %in% c("Wild-type","D2723H"))]))
  v_list <-c(v_list,list(c(c("Wild-type","D2723H"),v_l2[[i]])))
  var_num <- rep(0,length(d_[[i]]$var_name))
  var_cnt <- rep(0,length(d_[[i]]$var_name))
  score_ave <- rep(0,length(d_[[i]]$var_name))
  IARC <- c(IARC, list(rep(0,length(v_l[[i]]))))
  position <- c(position, list(rep(0,length(v_l[[i]]))))
```

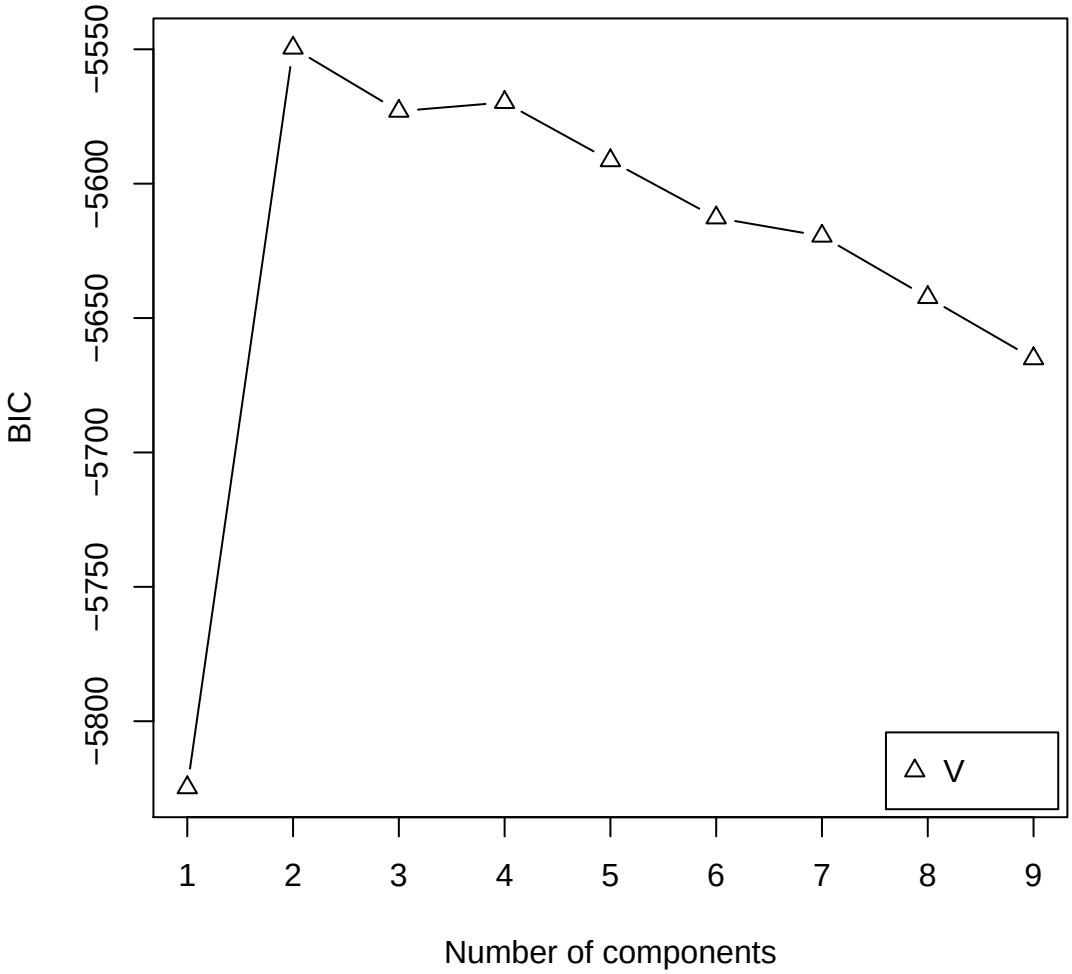
```
  Iclass <- c(Iclass, list(rep(0,length(v_l[[i]]))))
  aGVGD <- c(aGVGD, list(rep(0,length(v_l[[i]]))))
  ClinVar <- c(ClinVar, list(rep(0,length(v_l[[i]]))))
  for(k in 1:(length(d_[[i]]$var_name))){
    for(l in 1:(length(d_[[i]]$var_name))){
      if(d_[[i]]$var_name[[k]] == d_[[i]]$var_name[[l]]){
        score_ave[k] = score_ave[k] + d_[[i]]$score[[l]]
        var_cnt[k]  = var_cnt[k] + 1
      }
    }
    score_ave[k] = score_ave[k] / var_cnt[k]
    for(j in 1:(length(v_list[[i]]))){
      if(d_[[i]]$var_name[[k]] == v_list[[i]][j]){
        var_num[k] <- j
        IARC[[i]][j] <- d_[[i]]$standard[[k]]
        ClinVar[[i]][j] <- d_[[i]]$ClinVar[[k]]
        aGVGD[[i]][j] <- d_[[i]]$aGVGD_class[[k]]
        Iclass[[i]][j] <- d_[[i]]$IARC_class[[k]]
        position[[i]][j] <- d_[[i]]$position[[k]]
      }
    }
  }
  d_[[i]] <-cbind(d_[[i]],var_num,score_ave)
}
d_f <- d_
v_list_f <- v_list
IARC_f <- IARC
position_f <- position
Iclass_f <- Iclass
aGVGD_f <- aGVGD

d_ <- NULL
d___<-NULL
v_l <- NULL
v_l2 <- NULL
v_list <- NULL
IARC <- NULL
ClinVar <- NULL
aGVGD <- NULL
position <- NULL
Iclass <- NULL
for(i in 1:Drug_No){
  d__ <- list(subset(d, drug == Drug[i]))
  d_ <- c(d_,list(d__[[1]]))
  v_l <- c(v_l,list(unique(d_[[i]]$var_name)))
  v_l2 <- c(v_l2,list(v_l[[i]][-which(v_l[[i]] %in% c("Wild-type","D2723H"))]))
  v_list <-c(v_list,list(c(c("Wild-type","D2723H"),v_l2[[i]])))
  var_num <- rep(0,length(d_[[i]]$var_name))
  var_cnt <- rep(0,length(d_[[i]]$var_name))
  score_ave <- rep(0,length(d_[[i]]$var_name))
  IARC <- c(IARC, list(rep(0,length(v_l[[i]]))))
  position <- c(position, list(rep(0,length(v_l[[i]]))))
  Iclass <- c(Iclass, list(rep(0,length(v_l[[i]]))))
```

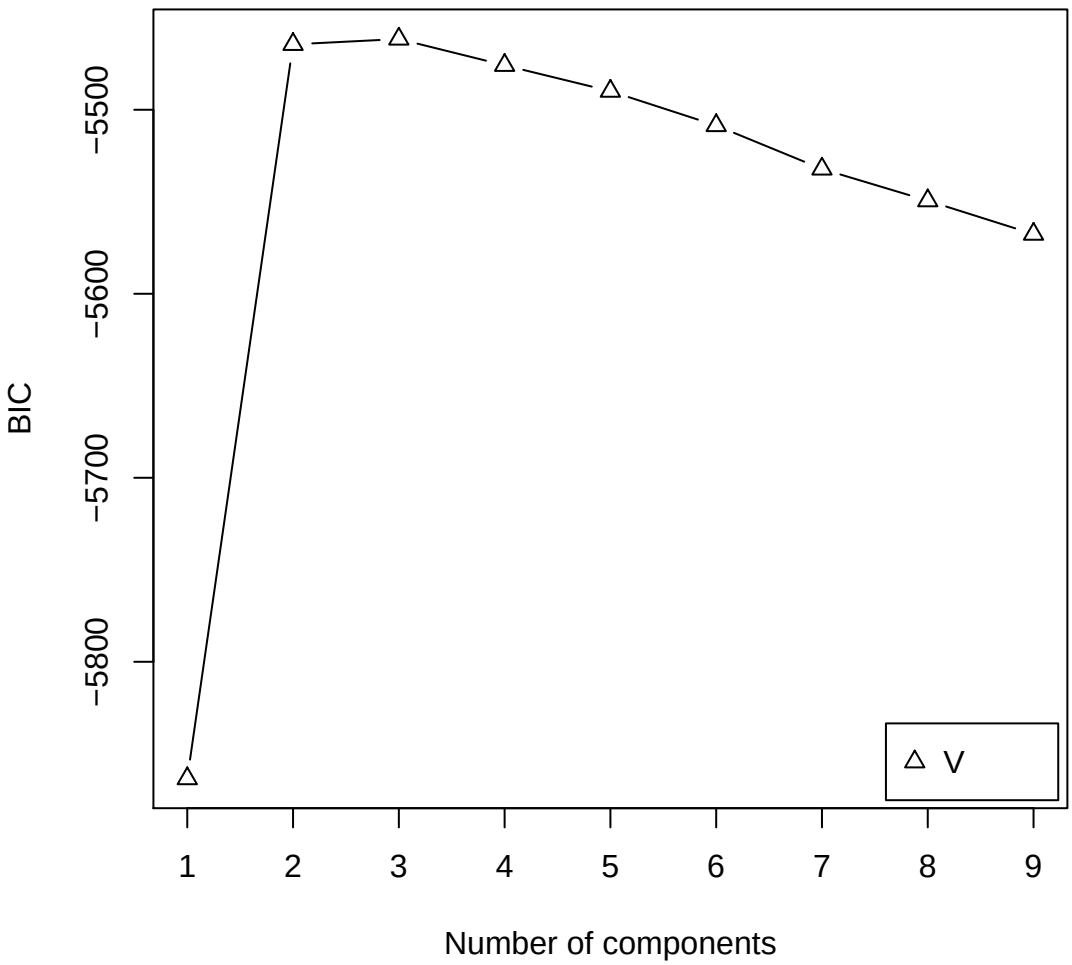
```
  aGVGD <- c(aGVGD, list(rep(0,length(v_l[[i]]))))
  ClinVar <- c(ClinVar, list(rep(0,length(v_l[[i]]))))
  for(k in 1:(length(d_[[i]]$var_name))){
    for(l in 1:(length(d_[[i]]$var_name))){
      if(d_[[i]]$var_name[[k]] == d_[[i]]$var_name[[l]]){
        score_ave[k] = score_ave[k] + d_[[i]]$score[[l]]
        var_cnt[k]  = var_cnt[k] + 1
      }
    }
    score_ave[k] = score_ave[k] / var_cnt[k]
    for(j in 1:(length(v_list[[i]]))){
      if(d_[[i]]$var_name[[k]] == v_list[[i]][j]){
        var_num[k] <- j
        IARC[[i]][j] <- d_[[i]]$standard[[k]]
        ClinVar[[i]][j] <- d_[[i]]$ClinVar[[k]]
        aGVGD[[i]][j] <- d_[[i]]$aGVGD_class[[k]]
        Iclass[[i]][j] <- d_[[i]]$IARC_class[[k]]
        position[[i]][j] <- d_[[i]]$position[[k]]
      }
    }
  }
  d_[[i]] <-cbind(d_[[i]],var_num,score_ave)
}
d_A <- d_
v_list_A <- v_list
IARC_A <- IARC
position_A <- position
Iclass_A <- Iclass
aGVGD_A <- aGVGD
```

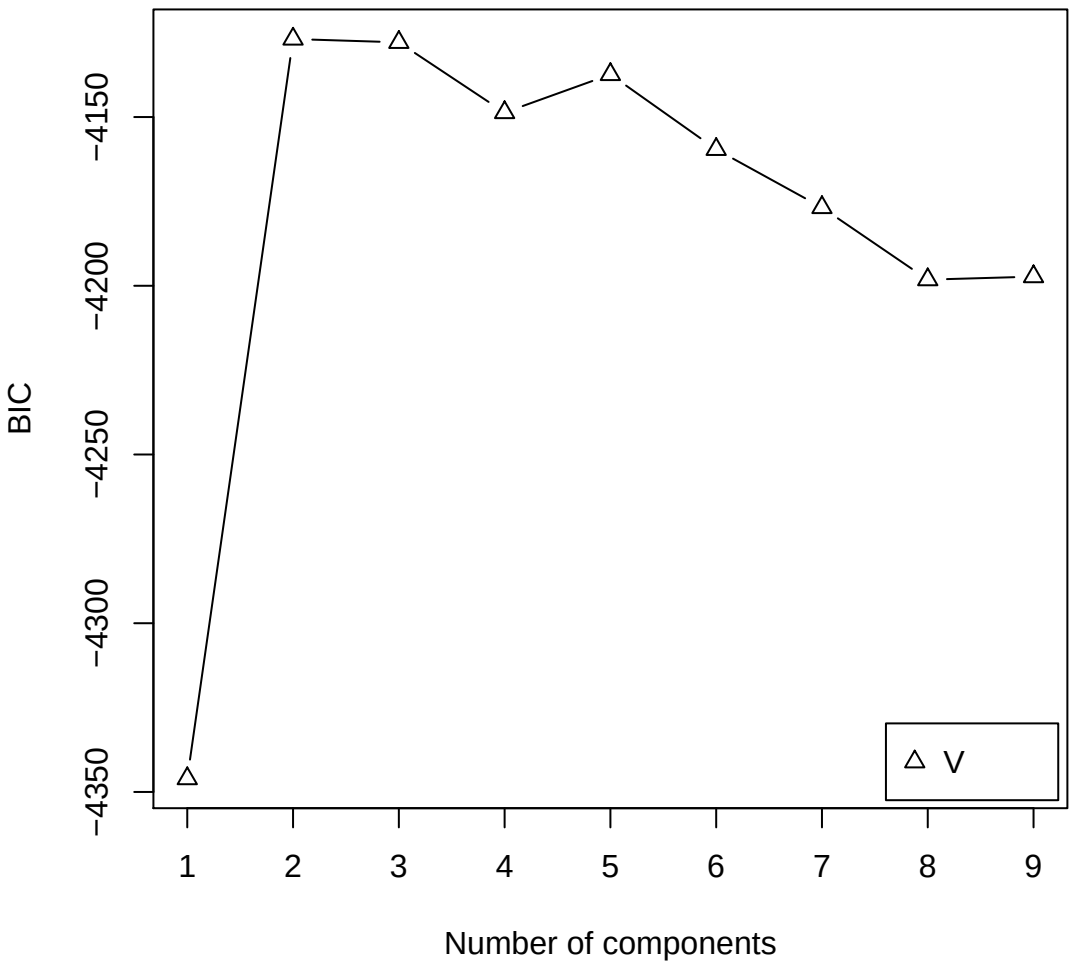
## 3.2 Gaussian finite mixture model fitted by expectation–maximization algorithm

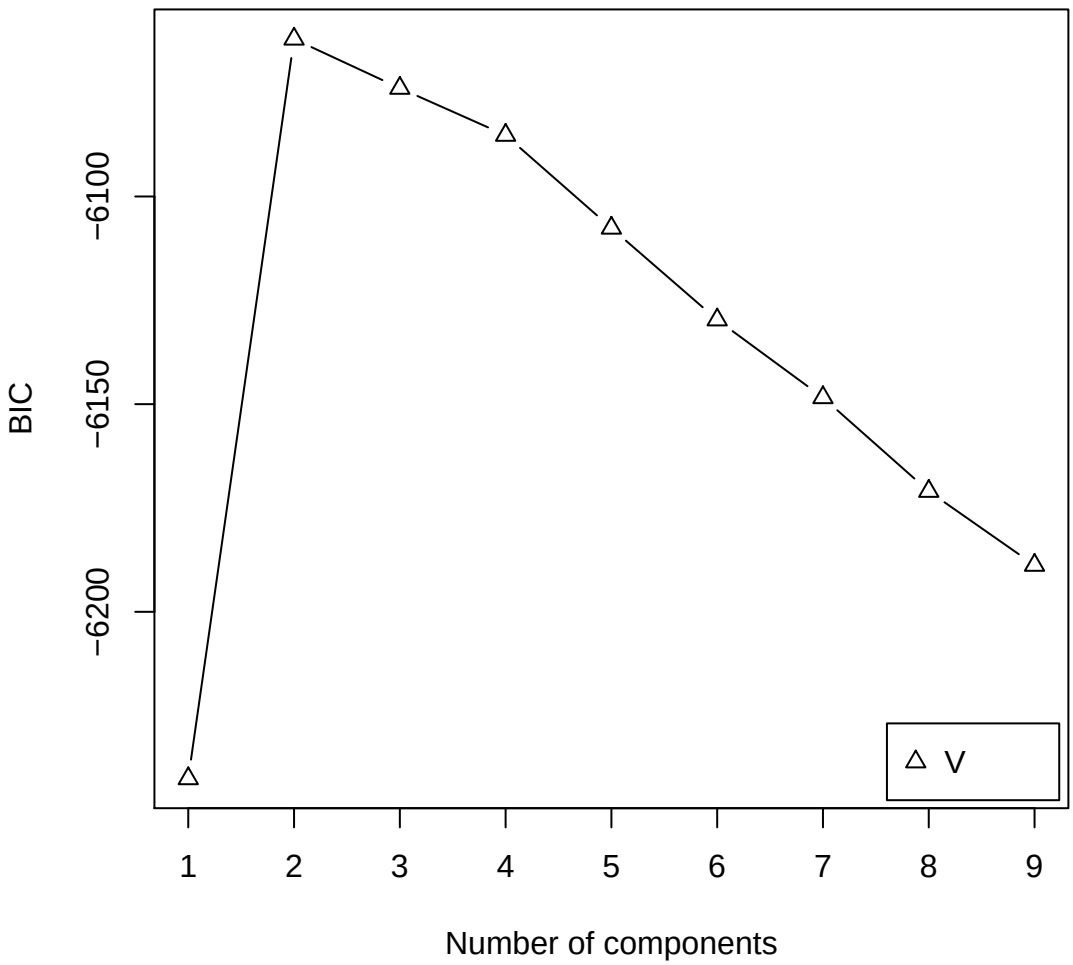
Use expectation–maximization algorithm to determine the appropriate Gaussian mixture model for the data. BIC: Bayesian information criterion.

```
for(i in 1:Drug_No){
  plot(mclustBIC(d_f[[i]]$score,modelNames="V"),
       main=paste("EM estimation for",Drug[[i]]))
}
```


BIC
−5550
−5600
−5650
−5700
−5750
−5800
V
1
2
3
4
5
6
7
8
9
Number of components


BIC
−5500
−5600
−5700
−5800
1
2
3
4
5
6
7
8
9
Number of components
V


BIC
−4150
−4200
−4250
−4300
−4350
1
2
3
4
5
6
7
8
9
Number of components
V

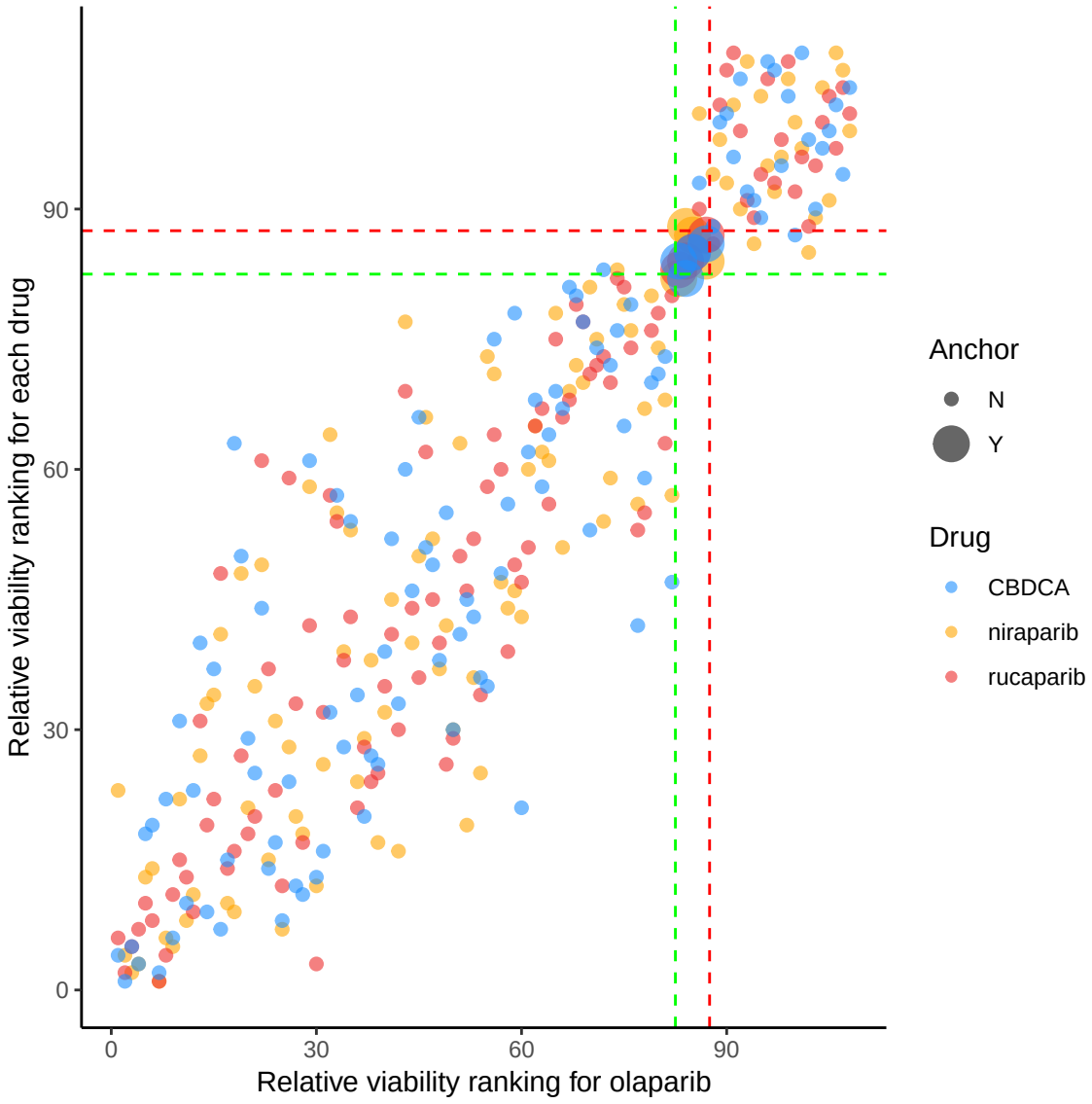



### 3.3 Scatter plot of relative viability ranking data

Relative viability ranking data of 107 variants between olaparib and other drugs.

```
d_rank <- NULL
ranking <- NULL
rank_ <- NULL
for(i in 1:Drug_No){
  d_rank <- c(d_rank,list(d_107[[i]][!duplicated(d_107[[i]]$var_name),]))
  len <- length(d_rank[[i]]$var_name)
  rank_ <- c(rank_,list(len+1-rank(d_rank[[i]]$score_ave)))
}
anchor <- rep("N",3*len)
rank_o <- rep(rank_[[1]],3)
rank_nrc <- c(rank_[[2]],rank_[[3]],rank_[[4]])
rank_d <- c(rep(Drug[2],len),rep(Drug[3],len),rep(Drug[4],len))
```

```
for(i in 1:(3*len)){
  if(rank_o[i] == 83 | rank_o[i] == 84
     | rank_o[i] == 85 | rank_o[i] == 87)
    anchor[i] <- "Y"
}
ranking <- data.frame(rank_o, rank_nrc, rank_d, anchor)
colnames(ranking) <- c("Olap_rank","Ranking","Drug","Anchor")

write.csv(ranking, "Ranking.csv",row.names=FALSE)

Colors <- c("dodgerblue", "orange", "firebrick2")

g <- ggplot(ranking, aes(x=Olap_rank, y=Ranking, col= Drug, size = Anchor))
g <- g + geom_point(alpha= 0.6)
g <- g + xlab("Relative viability ranking for olaparib")
g <- g + ylab("Relative viability ranking for each drug")
g <- g + geom_hline(yintercept=87.5,linetype="dashed",colour="red")
g <- g + geom_hline(yintercept=82.5,linetype="dashed",colour="green")
g <- g + geom_vline(xintercept=87.5,linetype="dashed",colour="red")
g <- g + geom_vline(xintercept=82.5,linetype="dashed",colour="green")
g <- g + theme_classic()
g <- g + scale_color_manual(values = Colors)
g <- g + scale_fill_manual(values = Colors)
g
```



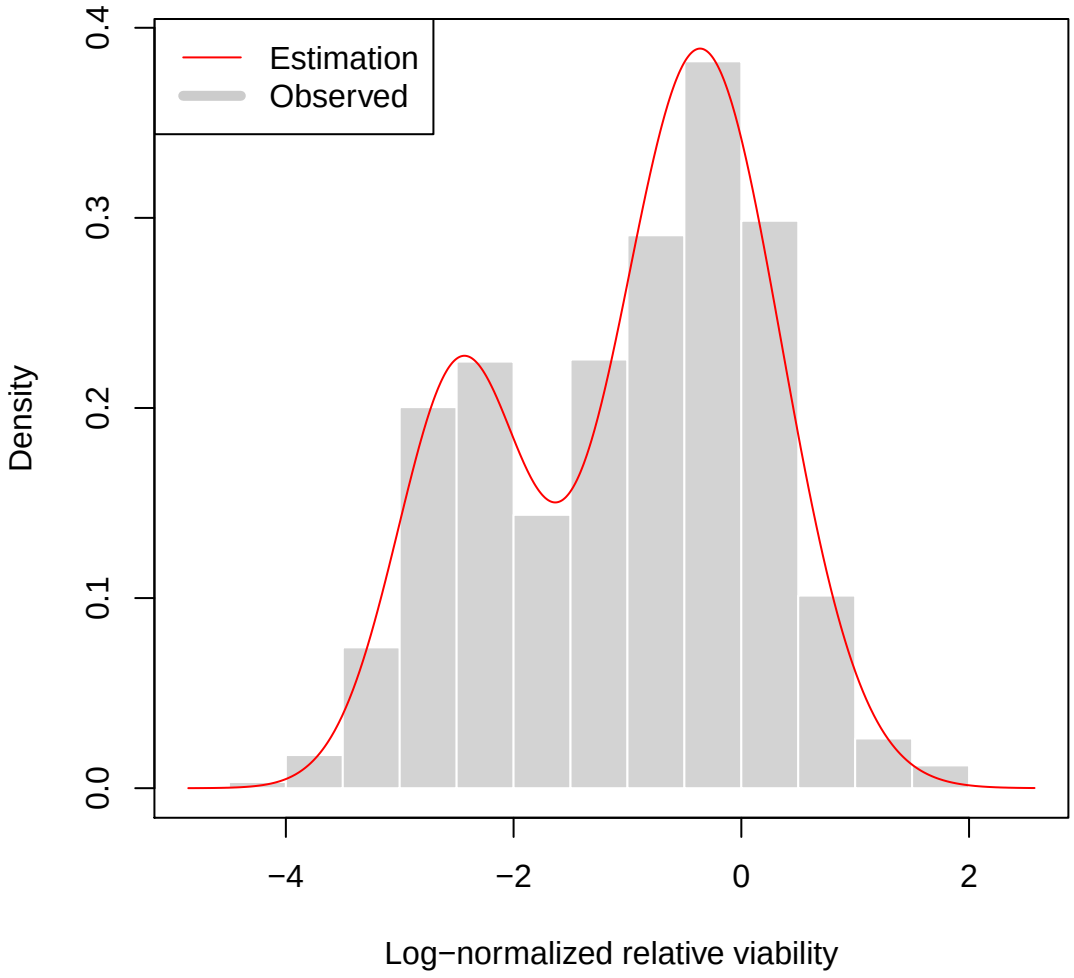

### 3.4 Density estimation

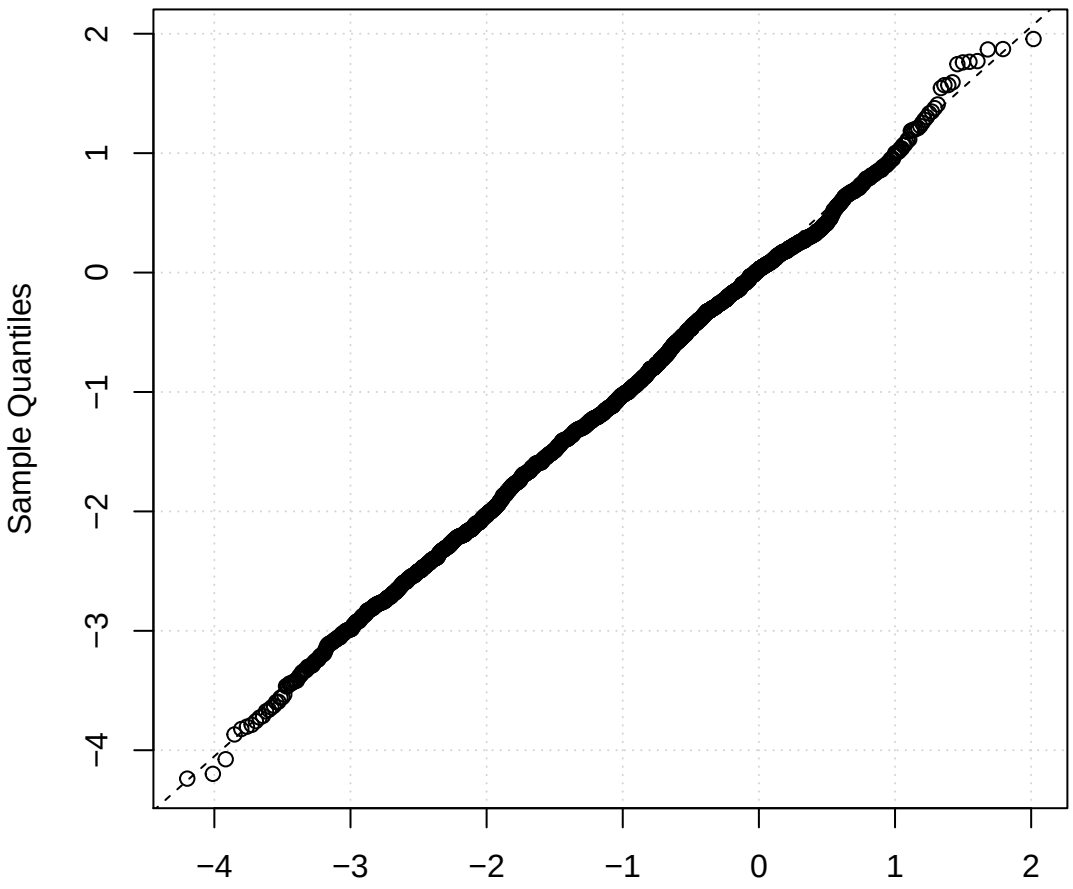
Estimated density curve of 244 variants on the basis of 2-component mixed Gaussian distribution. QQ-plots of standardized residuals are also shown.

```
Mc_B = NULL
for(i in 1:Drug_No){
  Mc_B <- c(Mc_B, list(densityMclust(d_f[[i]]$score, modelNames="V", G=2)))
}
```

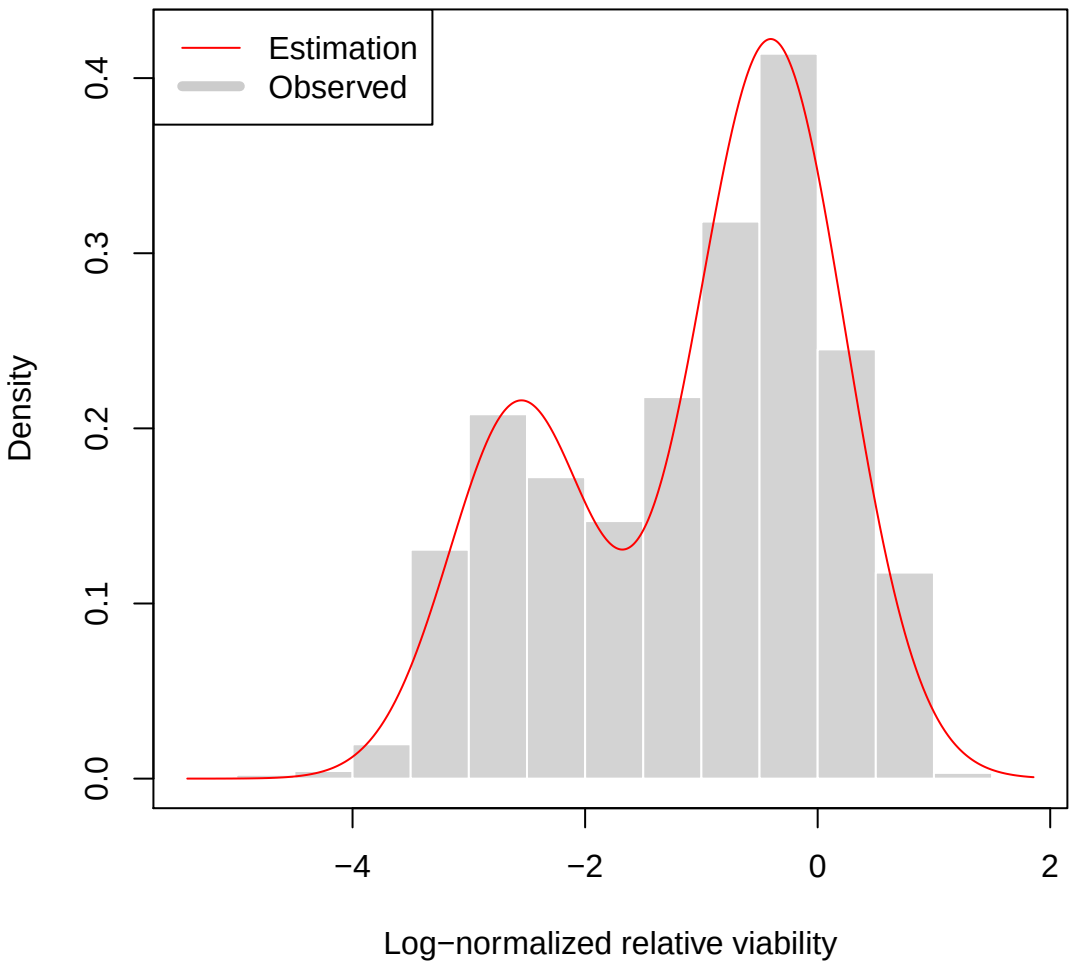
```
Mc_par <- NULL
for(i in 1:Drug_No){
  plotDensityMclust1(Mc_B[[i]], data=d_f[[i]]$score, xlab="Log-normalized relative viability", col = "r
       main=paste("Density estimation  with ", Drug[[i]]))
  legend("topleft", legend = c("Estimation","Observed"),
       col = c("red","grey80"), lty = c(1,1), lwd = c(1,5))
  plot(Mc_B[[i]], what = "diagnostic", type="qq", sub=c(paste(
```

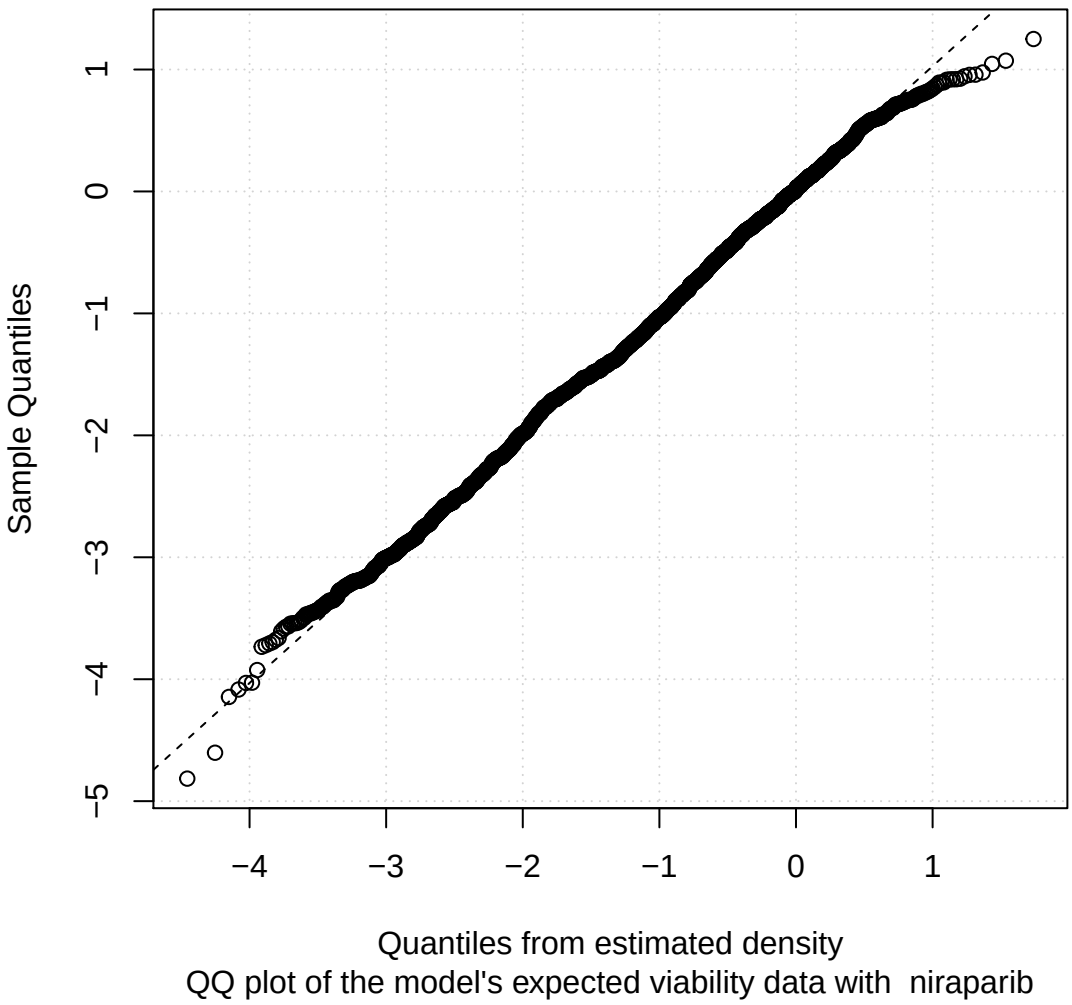
```
    "QQ plot of the model's expected viability data with ", Drug[[i]]))
  Mc_par <- c(Mc_par, list(summary(Mc_B[[i]], parameters=T))) # check the mean values of benign compone
}
```

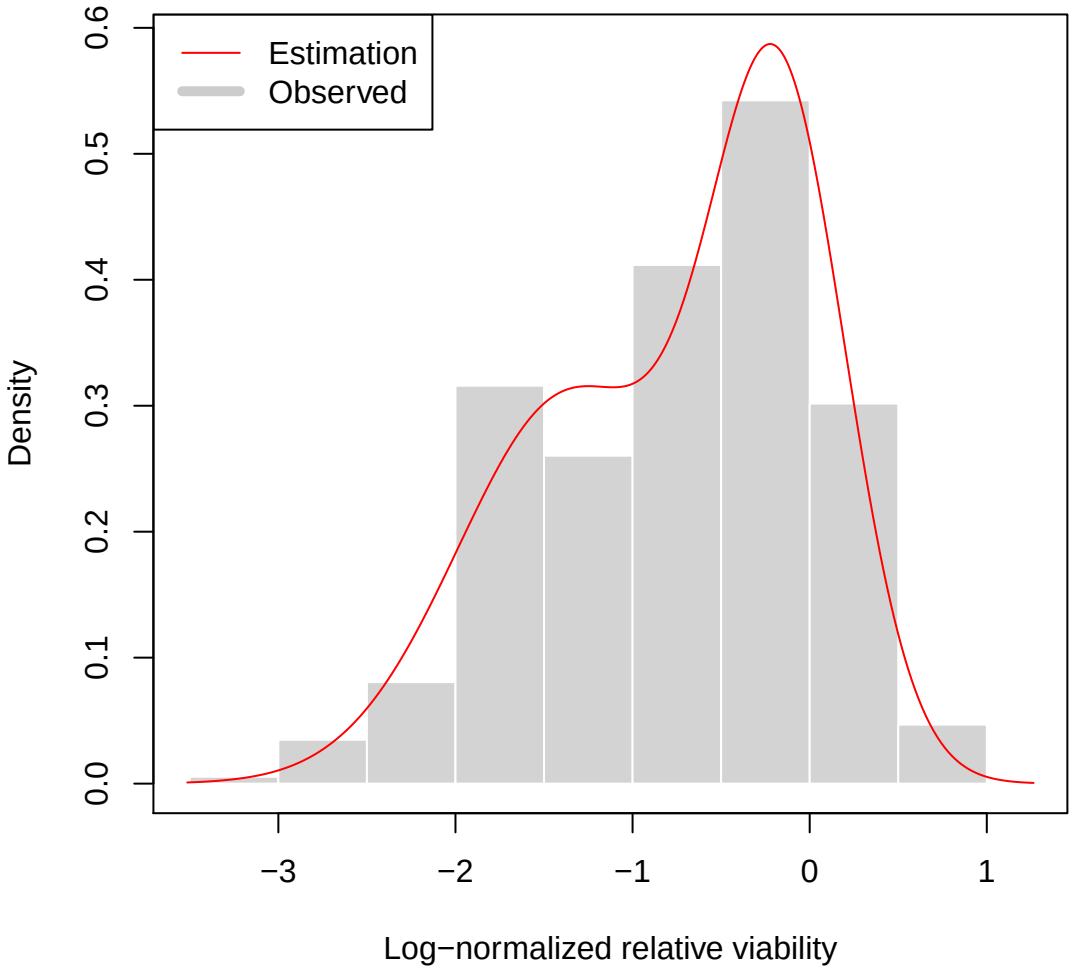


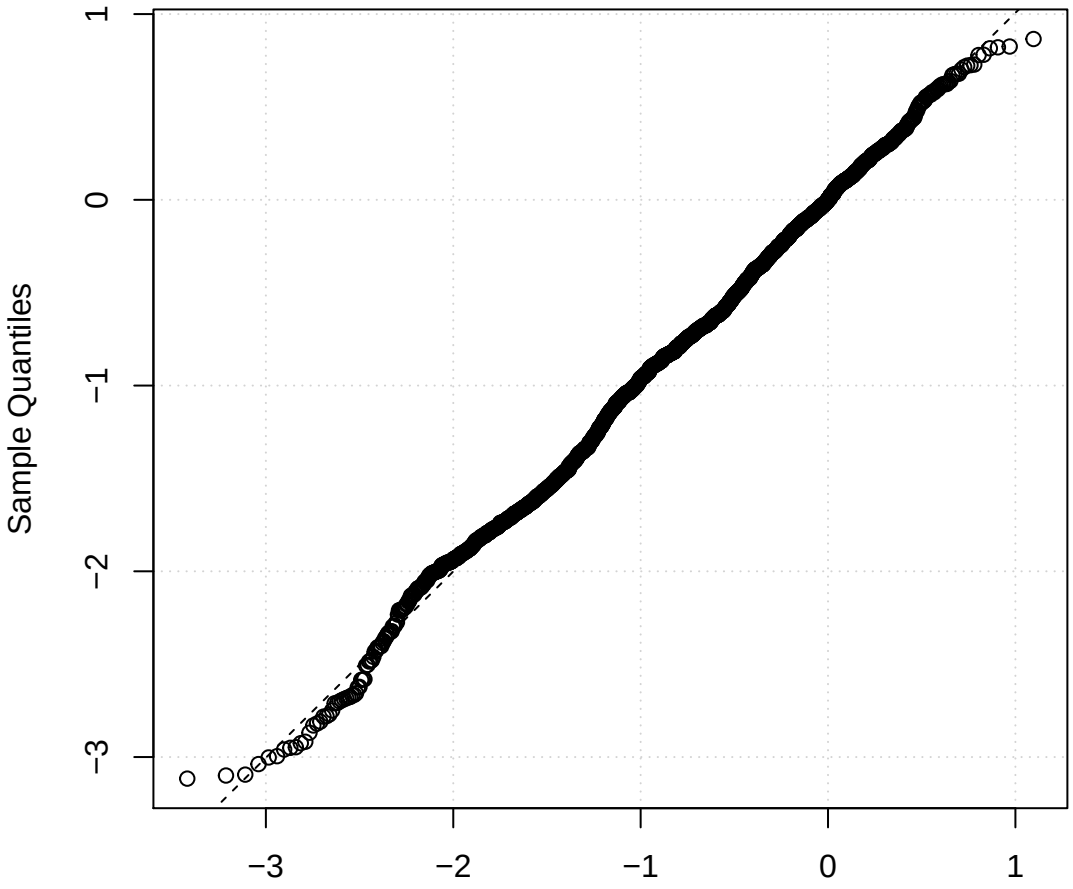



QQ plot of the model's expected viability data with olaparib

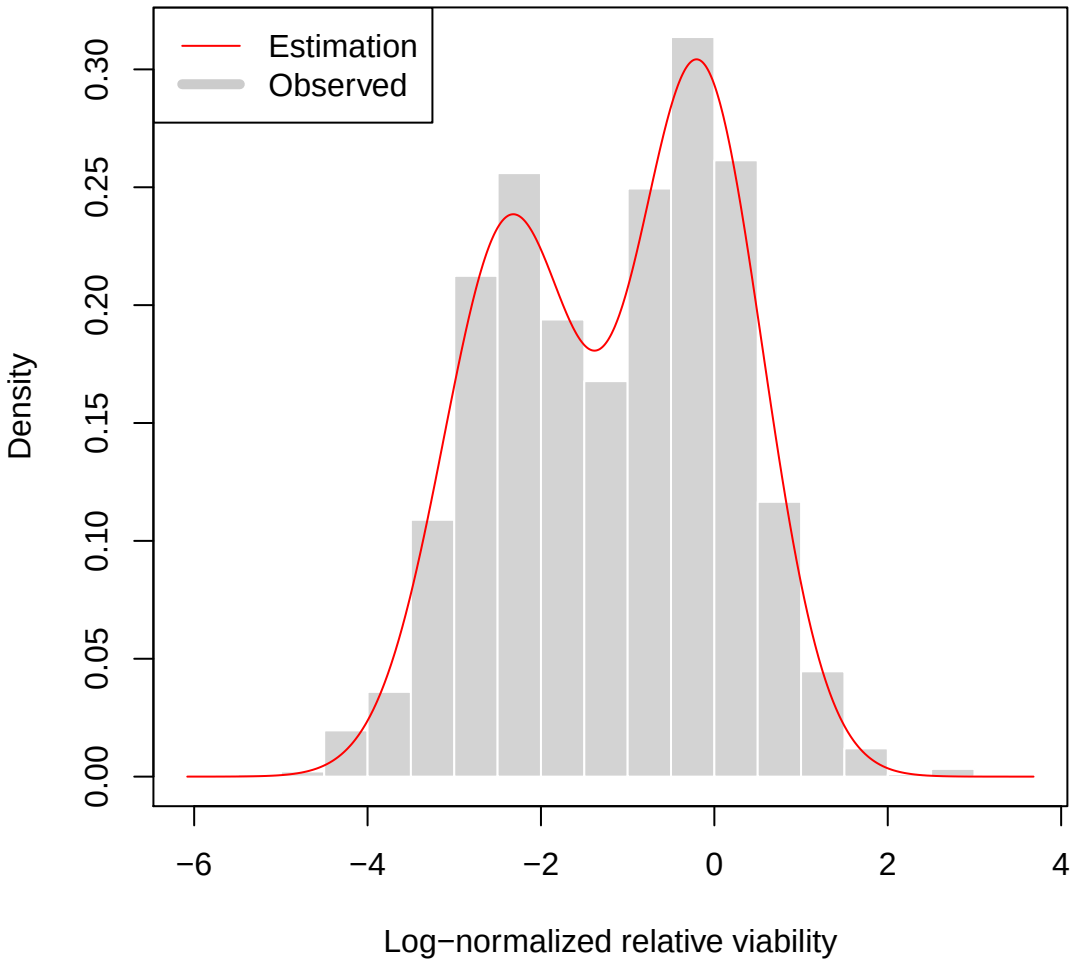

Estimation
Observed
Density
0.0
0.1
0.2
0.3
0.4
−4
−2
0
2
Log−normalized relative viability

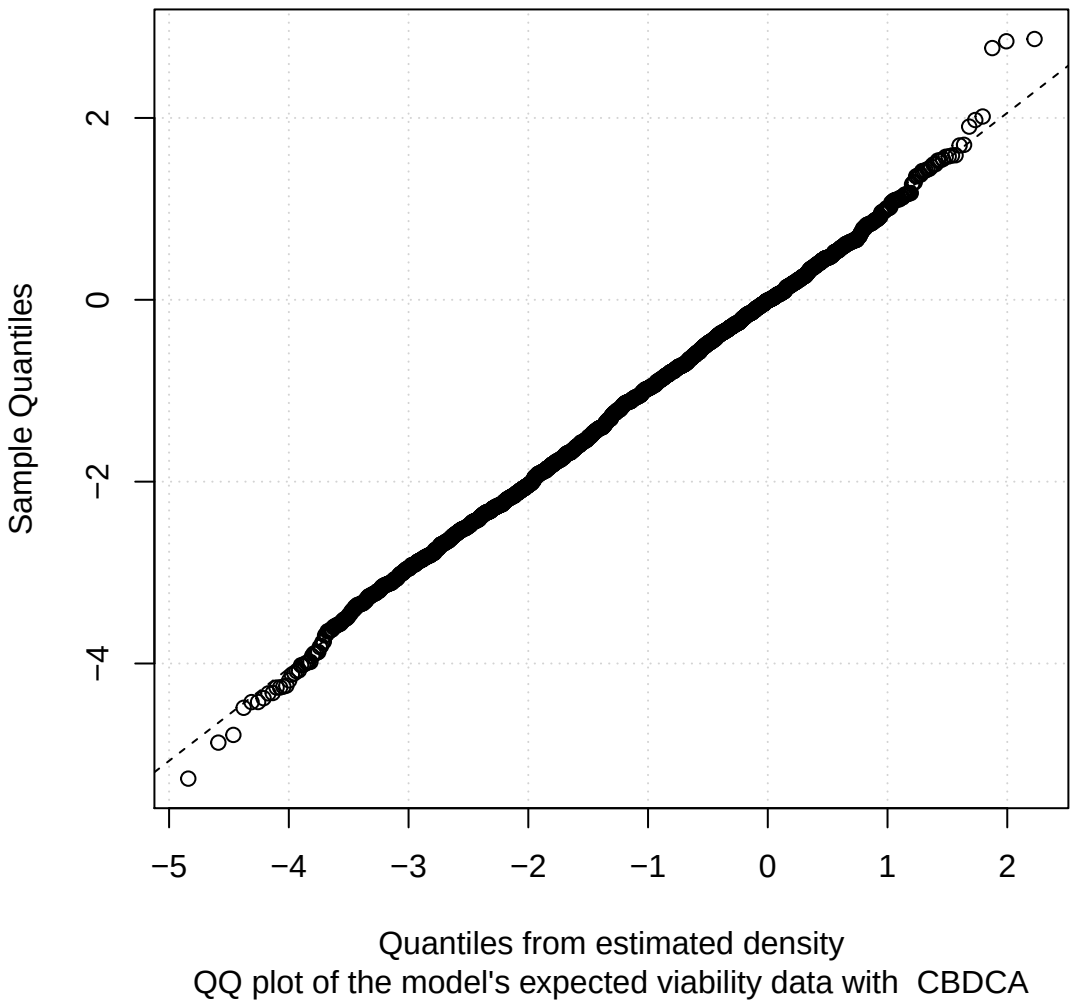

Sample Quantiles
1
0
−1
−2
−3
−4
−5
−4
−3
−2
−1
0
1
Quantiles from estimated density
QQ plot of the model's expected viability data with niraparib


Estimation
Observed
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
−3
−2
−1
0
1
Log−normalized relative viability




QQ plot of the model's expected viability data with rucaparib


Estimation
Observed
Density
0.00
0.05
0.10
0.15
0.20
0.25
0.30
−6
−4
−2
0
2
4
Log−normalized relative viability

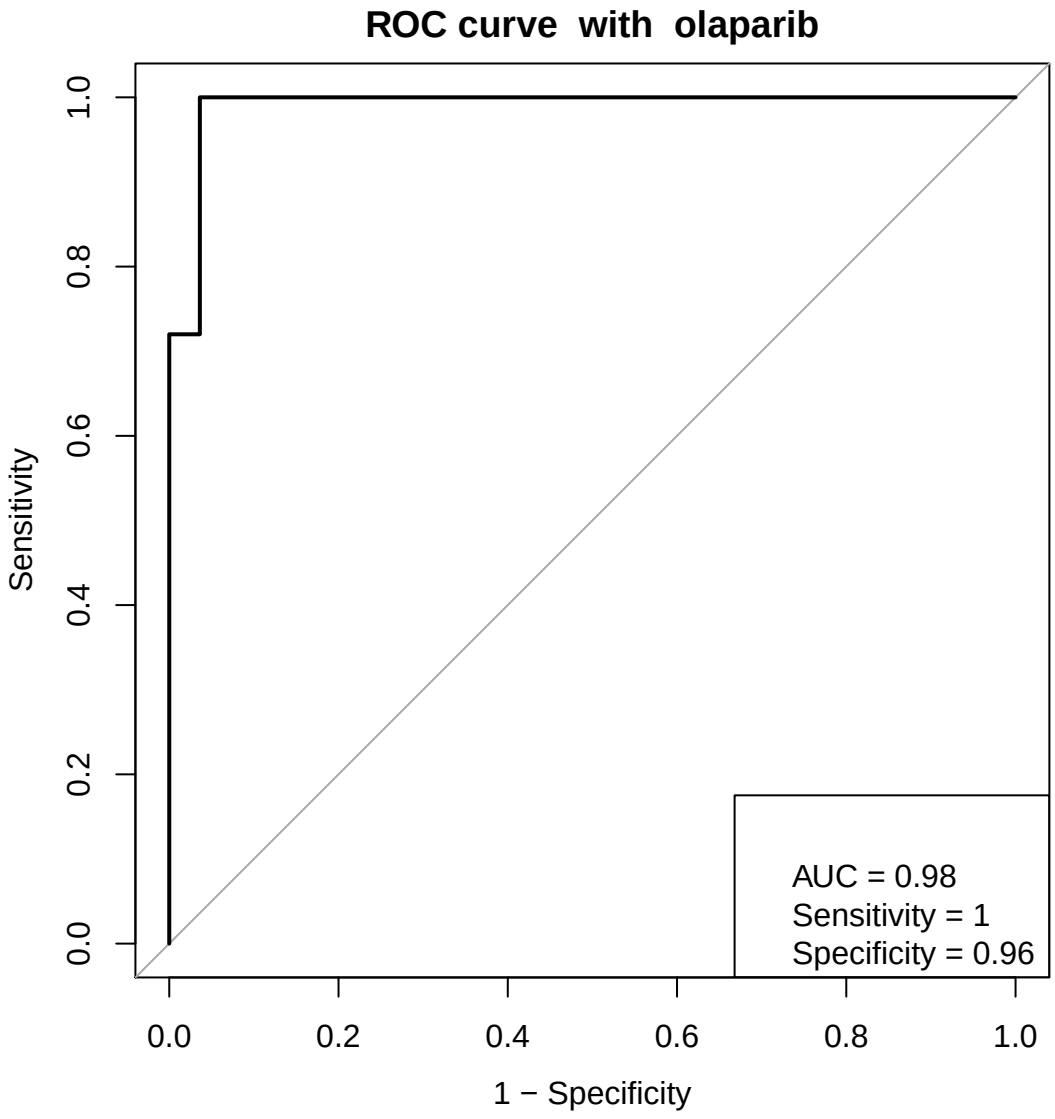


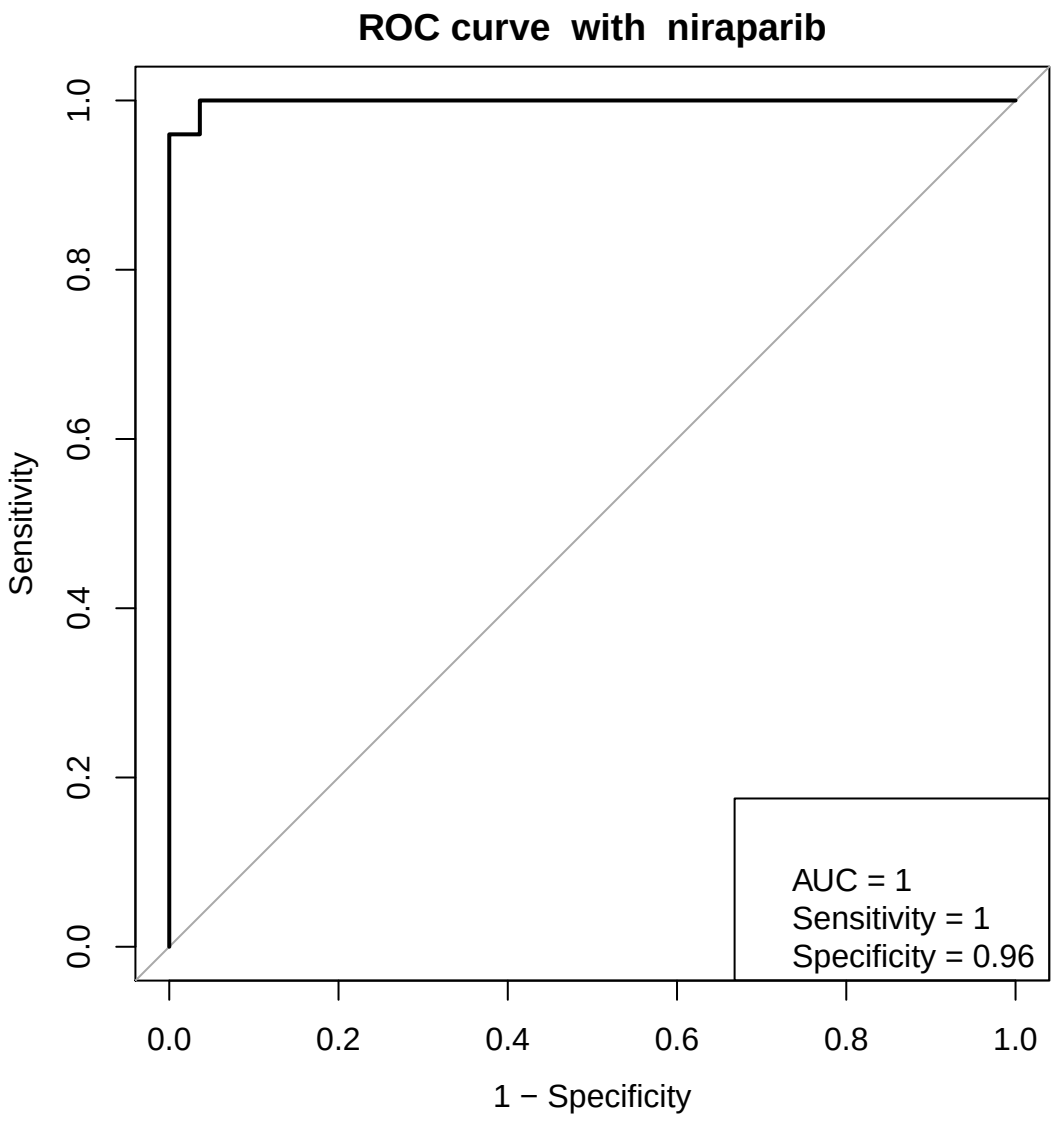

### 3.5 Receiver operating characteristic curve

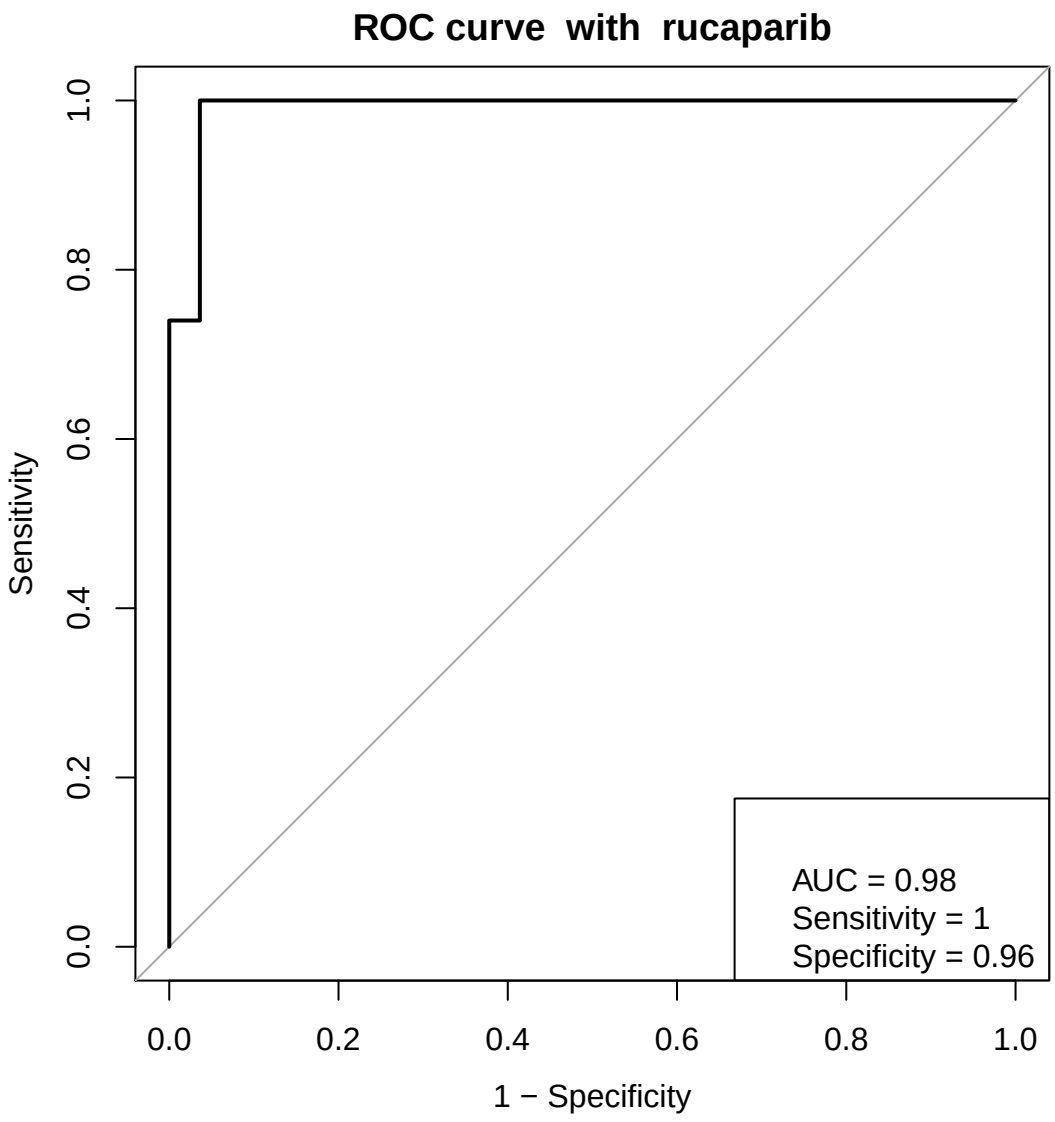
A operating characteristics analysis of the data obtained from the established pathogenic and benign variants.

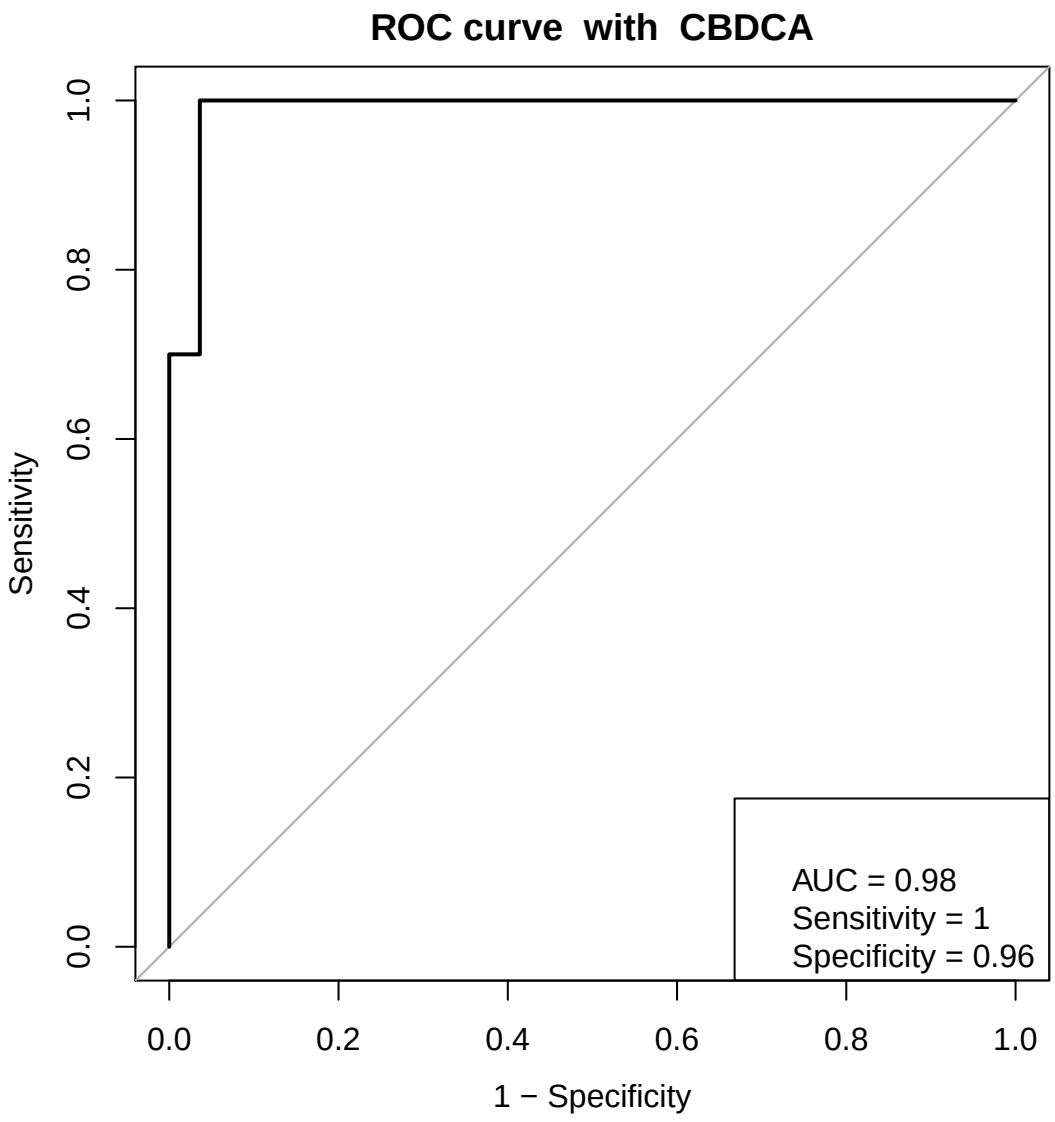
```
roc.model <- NULL
roc.auc <- NULL
roc.thr <- NULL
roc.cut <- NULL
for(i in 1:Drug_No){
  roc.model <- c(roc.model, list(roc(standard ~ score_ave,
          data=d_t[[i]])))
  roc.auc <- c(roc.auc,list(ci(roc.model[[i]], of = "auc",
                               method = "delong")))
  roc.cut <- c(roc.cut,list(coords(roc.model[[i]], x="best",
      ret=c("threshold", "sensitivity", "specificity"),
      best.method="youden")))
```

```
  plot(roc.model[[i]], legacy.axes = TRUE,
       main=paste("ROC curve ",
      "with ", Drug[[i]]))
  legend("bottomright", legend = paste(
        "AUC = ", round(roc.auc[[i]][1], digits=2),
        "\nSensitivity = ", round(roc.cut[[i]][2], digits=2),
        "\nSpecificity = ", round(roc.cut[[i]][3], digits=2),
        "\n", sep=""))
}
```




ROC curve with niraparib
Sensitivity
1 − Specificity
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 1
Sensitivity = 1
Specificity = 0.96


ROC curve with rucaparib
Sensitivity
1 − Specificity
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.98
Sensitivity = 1
Specificity = 0.96


ROC curve with CBDCA
Sensitivity
1 − Specificity
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.98
Sensitivity = 1
Specificity = 0.96

# 4 Hamiltonian Monte Carlo simulation

Data formatting for Hamiltonian Monte Carlo simulation to fit a 2-component Gaussian mixture model with unequal variances. The wild-type is labeled as benign and D2723H is labeled as pathogenic. All the remaining variants are unlabeled in the following analysis.

## 4.1 The classification model

The 2-component mixed normal distribution model is as follows:

$$\prod_{v} \Pr\left(f_v | D_v, X_v, \theta\right) =$$
$$\prod_{\{v:D_v=D\}} \Pr\left(f_v | D_v = D, X_v, \theta\right) \prod_{\{v:D_v=N\}} \Pr\left(f_v | D_v = N, X_v, \theta\right)$$
$$\prod_{\{v:D_v \text{ is Unknown }\}} \left[\pi_{a(v,D)} \Pr\left(f_v | D_v = D, X_v, \theta\right) + \pi_{a(v,N)} \Pr\left(f_v | D_v = N, X_v, \theta\right)\right]$$

$\Pr\left(\text{Data} \,|\, \text{Parameters}\right)$: likelihood of Data observation with Parameters

$v$: variant index

$f_v$: measurements of functional experiments for $v$

$D_v$: $\begin{cases} D_v = D \ (v = \text{deleterious}) \\ D_v = N \ (v = \text{neutral}) \end{cases}$

$X_v$: batch and experimental indices for each measurement

$\theta$: model parameters

$\pi_{a(v,D/N)}$: prior probability that $v$ is deleterious/neutral

## 4.2 Incorporating the functional assay data

The true distribution of data and parameters are estimated by the formula below:

$$\Pr\left(f_{v^*} | D_{v^*}, X_{v^*}, \theta\right) = \prod_{\{(v,b,e):v=v^*\}} \Pr\left(f_v | D_v, b, e, \theta\right)$$

$b$: batch index

$e$: experimental index

$\beta_b$: batch-specific random intercept effect

$\tau_b$: batch-specific random slope effect

$\kappa_1$: center of the distribution $\beta_b$

$\kappa_2$: center of the distribution $\tau_b$

$\lambda_1$: standard deviation of distribution $\beta_b$

$\lambda_2$: standard deviation of distribution $\tau_b$

$\eta_v$: variant-specific random effect

$\eta_{\text{del}}$: center of the deleterious variants' $\eta_v$ distribution

$\eta_{\text{neu}}$: center of the neutral variants' $\eta_v$ distribution

$\sigma_1$: standard deviation of the deleterious variants' $\eta_v$ distribution

$\sigma_2$: standard deviation of the neutral variants' $\eta_v$ distribution

$\psi$: residual error

Key domains: $\begin{cases} \text{PALB2 interaction domain (amino acid residues 10-40)} \\ \text{DNA-binding domain (2481-3186)} \\ \text{TR2 RAD51-binding domain (3269-3305)} \end{cases}$

The right-hand side term in the equation above is specified by the following hierarchical model. Prior probability is selectable from noninformative distribution or Align-GVGD based distribution.

iid: independent and identically distributed

$$
\begin{aligned}
f_v &\sim \text{Normal}\left(\beta_b + \tau_b\eta_v, \tau_b\psi\right) \\
\beta_b &\sim \text{Normal}\left(\kappa_1, \lambda_1\right) \\
\tau_b &\sim \text{Normal}\left(\kappa_2, \lambda_2\right), \tau_b > 0 \\
\eta_v \left| D_v = D \right. &\sim \text{Normal}\left(\eta_{\text{del}}, \sigma_1\right) \quad \{v : D_v = D \wedge v \neq \text{D2723H}\} \\
\eta_v \left| D_v = N \right. &\sim \text{Normal}\left(\eta_{\text{neu}}, \sigma_2\right) \quad \{v : D_v = N \wedge v \neq \text{WT}\} \\
\eta_{\text{WT}} &= \log_{10}(1.0) \\
\eta_{\text{D2723H}} &= \log_{10}(0.003) \\
\eta_{\text{nue}} &= \text{estimated value by mclust package with the training data set} \\
\eta_{\text{del}} &\sim \text{Normal}(\eta_{\text{D2723H}}, 5) \qquad \text{(training data set)} \\
\eta_{\text{del}} &= \text{as } \beta_b + \tau_b\eta_{\text{del}} \text{ value is the same in the training data set} \quad \text{(full analysis)} \\
\kappa_1 &\sim \text{Normal}(0, 5) \\
\kappa_2 &\sim \text{Normal}(1, 5) \\
\psi, \sigma_1, \sigma_2, \lambda_1, \lambda_2 &\overset{iid}{\sim} \text{HalfNormal}(0, 5) \\
D_v &\sim \text{Bernoulli}\left(\pi_{a(v)}\right) \quad \{v : D_v \text{ is unknown}\} \\
\pi_{a(v)} &\sim \text{Beta}(1, 1) \quad \text{(noninformative prior probability)}
\end{aligned}
$$

*or*

Align-GVGD-based prior probability

$$
\begin{cases}
\pi_{a(v)} \sim \text{Beta}(15.00, 3.48) & \{v \in C65, \text{inside key domains}\} \\
\pi_{a(v)} \sim \text{Beta}(5.38, 2.57) & \{v \in C35, C45, C55, \text{inside key domains}\} \\
\pi_{a(v)} \sim \text{Beta}(3.76, 9.00) & \{v \in C15, C25, \text{inside key domains}\} \\
\pi_{a(v)} \sim \text{Beta}(1.43, 73.1) & \{v \in C0, \text{inside key domains}\} \\
\pi_{a(v)} \sim \text{Beta}(1.64, 120.44) & \{v \in \text{outside key domains}\} \\
\pi_{a(v)} \sim \text{Beta}(387, 1.07) & \{v \in \text{nonsense variants}\}
\end{cases}
$$

### 4.3 Stan simulation program

Hamiltonian Monte Carlo simulation code for 2-component model written in Stan language is as follows. Choose prior distribution as you think.

For the training data set

```
stan_code <-
"data{
  int N;
  int B[N];
  int BN;
  int V;
  vector[N] Y;
  int variant[N];
  real mu2;
}

transformed data{
  vector[2] fix;
  fix[1] = 0;
  fix[2] = -2.522879;
```

```
}

parameters{
  simplex[2] pi_[V];
  real mu1;
  real kappa_[2];
  real<lower=0> lambda_[2];
  real<lower=0> psi_;
  vector[V] eta;
  vector[BN] bet_;
  vector<lower=0>[BN] tau_;
  vector<lower=0>[2] sigma_;
}

transformed parameters{
  vector[V] eta_;
  matrix[V,2] lnp;
  vector[2] mu_;
  eta_[1] = fix[1];
  eta_[2] = fix[2];
  mu_[1] = mu1;
  mu_[2] = mu2;
  for(v in 3:V){
    eta_[v] = eta[v];
  }
  for(i in 1:2){
    for(v in 1:V){
      lnp[v,i] = log(pi_[v,i]) +
          normal_lpdf(eta_[v] | mu_[i], sigma_[i]);
    }
  }
}

model{
  for(v in 1:V){
    target += log_sum_exp(lnp[v]);
  }
  for(i in 1:BN){
    bet_[i] ~ normal(kappa_[1], lambda_[1]);
    tau_[i] ~ normal(kappa_[2], lambda_[2]);
  }
  for(n in 1:N){
      Y[n] ~ normal(bet_[B[n]]
          +tau_[B[n]]*eta_[variant[n]], tau_[B[n]]*psi_);
  }

  sigma_ ~ normal(0, 5);
  psi_ ~ normal(0, 5);
  lambda_ ~ normal(0, 5);
  kappa_[1] ~ normal(0, 5);
  kappa_[2] ~ normal(1, 5);
  mu1 ~ normal(fix[2], 5);
  for(n in 1:N)
```

```
    pi_[variant[n]][1] ~ beta(1,1);
}

generated quantities{
  matrix[V,2] p_y;
  real tmp[N];
  real mu1_;
  for(v in 1:V){
    p_y[v] = softmax(lnp[v]')';
  }
  for(n in 1:N){
    tmp[n] = bet_[B[n]]+tau_[B[n]]*mu1;
  }
  mu1_ = sum(tmp)/N;
}
"
stanmodel2 <- stan_model(model_code = stan_code)
```

For the full analysis with noninformative prior probability or with Align-GVGD prior probability

```
stan_code <-
"data{
  int N;
  int B[N];
  int BN;
  int V;
  vector[N] Y;
  vector[N] la;
  vector[N] lb;
  int variant[N];
  real mu1_;
  real mu2;
}

transformed data{
  vector[2] fix;
  fix[1] = 0;
  fix[2] = -2.522879;
}

parameters{
  simplex[2] pi_[V];
  real kappa_[2];
  real<lower=0> lambda_[2];
  real<lower=0> psi_;
  vector[V] eta;
  vector[BN] bet_;
  vector<lower=0>[BN] tau_;
  vector<lower=0>[2] sigma_;
}

transformed parameters{
  vector[V] eta_;
  matrix[V,2] lnp;
```

```
  vector[2] mu_;
  real mu1;
  real tmp1[N];
  real tmp2[N];
  eta_[1] = fix[1];
  eta_[2] = fix[2];
  for(n in 1:N){
    tmp1[n] = mu1_ - bet_[B[n]];
    tmp2[n] = tau_[B[n]];
  }
  mu1 = sum(tmp1)/sum(tmp2);
  mu_[1] = mu1;
  mu_[2] = mu2;
  for(v in 3:V){
    eta_[v] = eta[v];
  }
  for(i in 1:2){
    for(v in 1:V){
      lnp[v,i] = log(pi_[v,i]) +
          normal_lpdf(eta_[v] | mu_[i], sigma_[i]);
    }
  }
}

model{
  for(v in 1:V){
    target += log_sum_exp(lnp[v]);
  }
  for(i in 1:BN){
    bet_[i] ~ normal(kappa_[1], lambda_[1]);
    tau_[i] ~ normal(kappa_[2], lambda_[2]);
  }
  for(n in 1:N){
      Y[n] ~ normal(bet_[B[n]]
          +tau_[B[n]]*eta_[variant[n]], tau_[B[n]]*psi_);
  }

  sigma_ ~ normal(0, 5);
  psi_ ~ normal(0, 5);
  lambda_ ~ normal(0, 5);
  kappa_[1] ~ normal(0, 5);
  kappa_[2] ~ normal(1, 5);
  for(n in 1:N)
//  pi_[variant[n]][1] ~ beta(la[n],lb[n]); // Align-GVGD prior
  pi_[variant[n]][1] ~ beta(1,1); // noninformative prior
}

generated quantities{
  matrix[V,2] p_y;
  for(v in 1:V){
    p_y[v] = softmax(lnp[v]')';
  }
}
```

```
"
stanmodel3 <- stan_model(model_code = stan_code)
```

## 4.4 Execution of Hamiltonian Monte Carlo simulation

Change “chains”, “iter”, “warmup” referring to R-hat values and posterior expectations.

For the training data set analysis

```
mu2 <- c(-0.45, -0.47, -0.31, -0.35)
data_t <- NULL
for(i in 1:Drug_No){
  data_t <- c(data_t, list(list(N = length(d_t[[i]]$var_num), V = max(d_t[[i]]$var_num),
                  Y = d_t[[i]]$score, variant = d_t[[i]]$var_num,
                  B = d_t[[i]]$batch, BN = max(d_t[[i]]$batch)
                  ,mu2 = mu2[[i]]
                  )))
}
fit_t <- NULL
for(i in 1:Drug_No){
  fit_t <- c(fit_t, list(stan(file="Stan_code_2.Stan", data=data_t[[i]],
            control=list(adapt_delta=0.99, max_treedepth = 20),
            seed=1234, chains=4, iter=5000, warmup=1500, thin=1)))
}
save(fit_t ,file="fit_2.txt")
```

For the full data set analysis

```
mu1_ <- c(-2.60, -2.71, -1.92, -2.59)
mu2 <- c(-0.45, -0.47, -0.31, -0.35)
data_f <- NULL
for(i in 1:Drug_No){
  data_f <- c(data_f, list(list(N = length(d_f[[i]]$var_num), V = max(d_f[[i]]$var_num),
                  Y = d_f[[i]]$score, variant = d_f[[i]]$var_num,
                  la = d_f[[i]]$a, lb = d_f[[i]]$b,
                  B = d_f[[i]]$batch, BN = max(d_f[[i]]$batch)
                  ,mu1_ = mu1_[[i]]
                  ,mu2 = mu2[[i]]
                  )))
}
fit_f <- NULL
for(i in 1:Drug_No){
  fit_f <- c(fit_f, list(stan(file="Stan_code_3.Stan", data=data_f[[i]],
            control=list(adapt_delta=0.99, max_treedepth = 20),
            seed=1234, chains=4, iter=5000, warmup=1500, thin=1)))
}
save(fit_f, file="fit_3.txt")
```

For the accurate BRCA2 companion diagnostic test with niraparib

```
mu1_ <- c(-2.60, -2.71, -1.92, -2.59)
mu2 <- c(-0.45, -0.47, -0.31, -0.35)
data_A <- NULL
data_A <- list(list(N = length(d_A[[2]]$var_num), V = max(d_A[[2]]$var_num),
                Y = d_A[[2]]$score, variant = d_A[[2]]$var_num,
```

```
                la = d_A[[2]]$a, lb = d_A[[2]]$b,
                B = d_A[[2]]$batch, BN = max(d_A[[2]]$batch)
                ,mu1_ = mu1_[[2]]
                ,mu2 = mu2[[2]]))
fit_A <- list(stan(file="Stan_code_3.Stan", data=data_A[[1]],
            control=list(adapt_delta=0.99, max_treedepth = 20),
            seed=1234, chains=4, iter=5000, warmup=1500, thin=1))
save(fit_A,file="fit_4.txt")
```

# 5 Quality check

Judge the convergence of simulation if R-hat values of all parameters are $< 1.1$ or not.

```
for(i in 1:Drug_No){
  launch_shinystan(fit_t[[i]])
  launch_shinystan(fit_f[[i]])
  launch_shinystan(fit_A[[i]])
}
```

Select the data to analyse.

```
#Dataset = "test"
 Dataset = "full"
#Dataset = "ABCD"

if (Dataset == "test"){
  Drug <- c("olaparib", "niraparib", "rucaparib", "CBDCA")
  Drug_No <- length(Drug)
  d_ <- d_t
  v_list <- v_list_t
  IARC <- IARC_t
  position <- position_t
  Iclass <- Iclass_t
  aGVGD <- aGVGD_t
  fit_ <- fit_t
}
if (Dataset == "full"){
  Drug <- c("olaparib", "niraparib", "rucaparib", "CBDCA")
  Drug_No <- length(Drug)
  d_ <- d_f
  v_list <- v_list_f
  IARC <- IARC_f
  position <- position_f
  Iclass <- Iclass_f
  aGVGD <- aGVGD_f
  fit_ <- fit_f
}
if (Dataset == "ABCD"){
  Drug <- "niraparib"
  Drug_No <- length(Drug)
  d_ <- d_A[[2]]
  v_list <- list(v_list_A[[2]])
  IARC <- list(IARC_A[[2]])
  position <- list(position_A[[2]])
  Iclass <- list(Iclass_A[[2]])
  aGVGD <- list(aGVGD_A[[2]])
  fit_ <- fit_A
}
```

## 5.1 Simulation summary

Extract the posterior information of the parameters.

```
df_ <- NULL
df_eta <- NULL
df_mu <- NULL
df_mu1 <- NULL
df_bet <- NULL
df_tau <- NULL
df_kappa <- NULL
df_lambda <- NULL
df_psi <- NULL
df_sigma <- NULL
df_p <- NULL
df_pp <- NULL
for(i in 1:Drug_No){
  df_eta <- c(df_eta,list(subset(summary(fit_[[i]])$summary, grepl("eta_",
              rownames(summary(fit_[[i]])$summary)))))
  df_mu <- c(df_mu,list(subset(summary(fit_[[i]])$summary, grepl("mu_",
              rownames(summary(fit_[[i]])$summary)))))
  df_mu1 <- c(df_mu1,list(subset(summary(fit_[[i]])$summary, grepl("mu1_",
              rownames(summary(fit_[[i]])$summary)))))
  df_bet <- c(df_bet,list(subset(summary(fit_[[i]])$summary, grepl("bet",
              rownames(summary(fit_[[i]])$summary)))))
  df_tau <- c(df_tau,list(subset(summary(fit_[[i]])$summary, grepl("tau",
              rownames(summary(fit_[[i]])$summary)))))
  df_kappa <- c(df_kappa,list(subset(summary(fit_[[i]])$summary, grepl("kappa",
              rownames(summary(fit_[[i]])$summary)))))
  df_lambda <- c(df_lambda,list(subset(summary(fit_[[i]])$summary, grepl("lambda",
              rownames(summary(fit_[[i]])$summary)))))
  df_psi <- c(df_psi,list(subset(summary(fit_[[i]])$summary, grepl("psi",
              rownames(summary(fit_[[i]])$summary)))))
  df_sigma <- c(df_sigma,list(subset(summary(fit_[[i]])$summary, grepl("sigma",
              rownames(summary(fit_[[i]])$summary)))))
  df_p <- c(df_p,list(subset(summary(fit_[[i]])$summary, grepl("p_y",
              rownames(summary(fit_[[i]])$summary)))))
  df_ <- (rbind(df_mu[[i]],df_bet[[i]],df_kappa[[i]],df_lambda[[i]],
                df_psi[[i]],df_sigma[[i]],df_eta[[i]],df_p[[i]]))
  filename <- paste(Drug[[i]], '.txt', sep='')
  write.table(data.frame(df_), file=filename, sep='\t',
            quote=FALSE, col.names=NA)
}
```

Graphical summary of HMC simulation

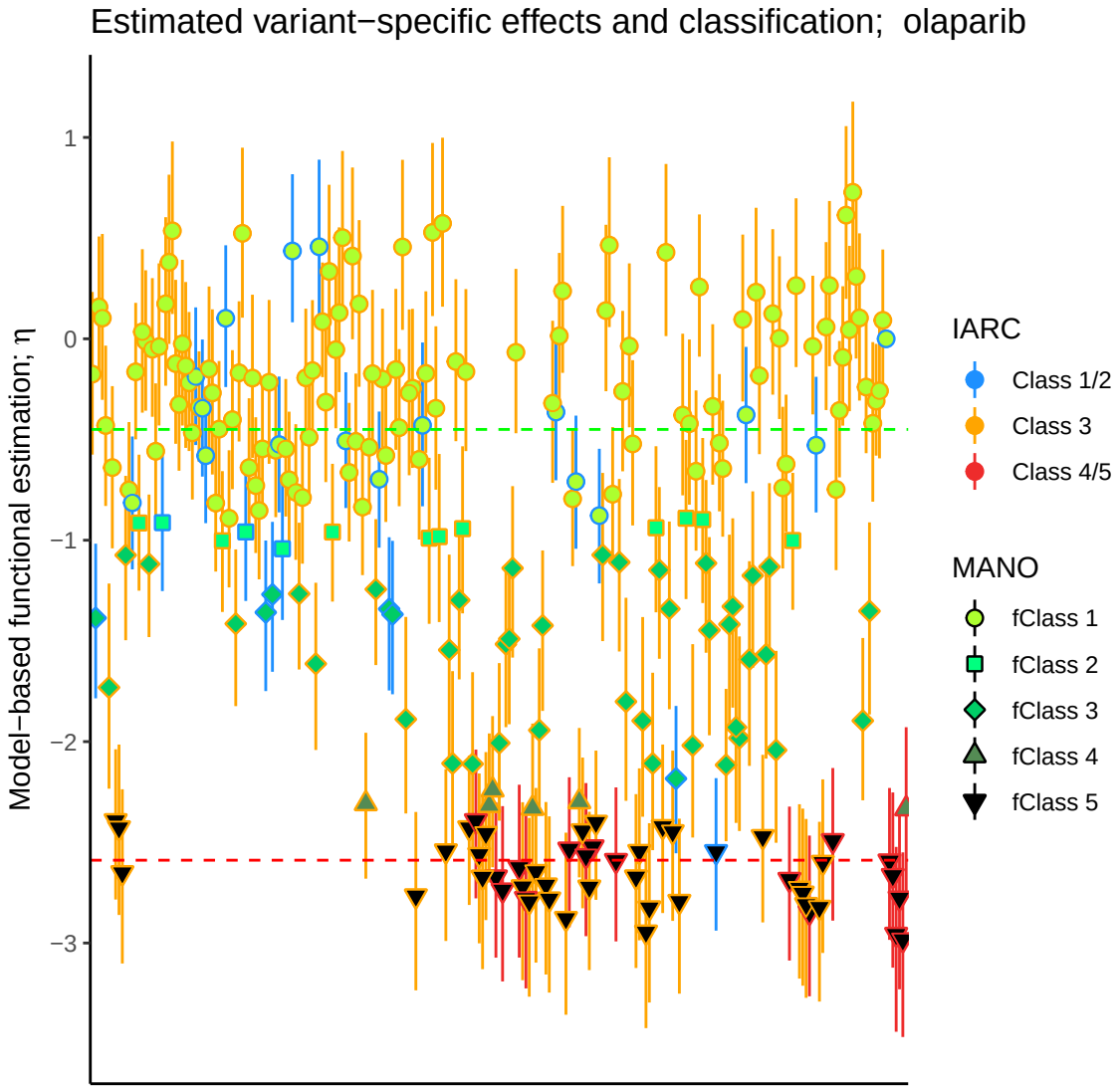
```
ggs_ <- NULL
for(i in 1:Drug_No){
  ggs_ <- c(ggs_, list(ggs(fit_[[i]])))
  filename <- paste(Drug[[i]], '-ggmcmc_density.pdf', sep='')
  ggmcmc(ggs_[[i]], file=filename, plot=c('density'))
  filename <- paste(Drug[[i]], '-ggmcmc_trace.pdf', sep='')
  ggmcmc(ggs_[[i]], file=filename, plot=c('traceplot'))
  filename <- paste(Drug[[i]], '-ggmcmc_running.pdf', sep='')
  ggmcmc(ggs_[[i]], file=filename, plot=c('running'))
  filename <- paste(Drug[[i]], '-ggmcmc_caterpillar.pdf', sep='')
  ggmcmc(ggs_[[i]], file=filename, plot=c('caterpillar'))
}
```

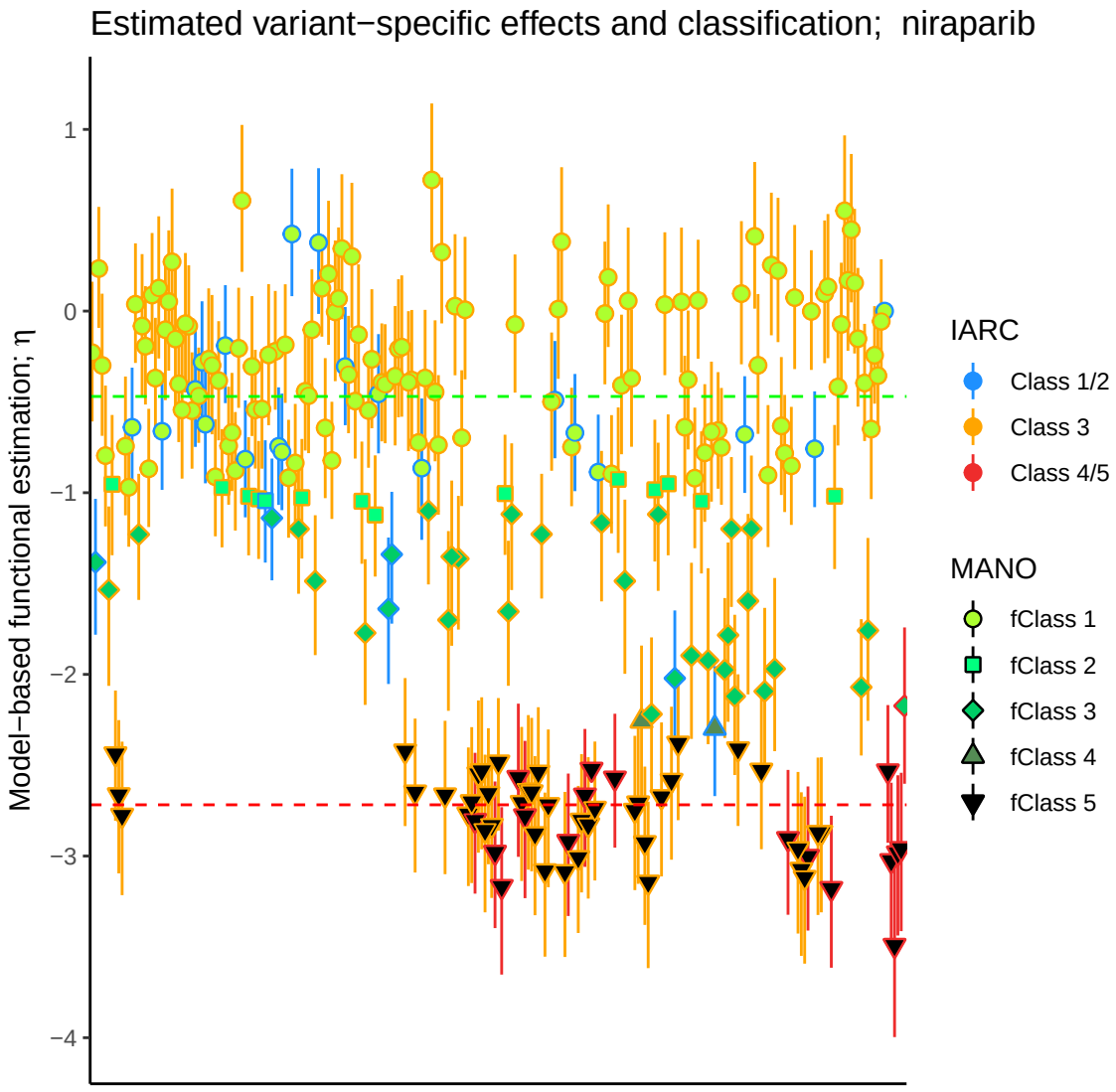
## 5.2 Diagnostic figures

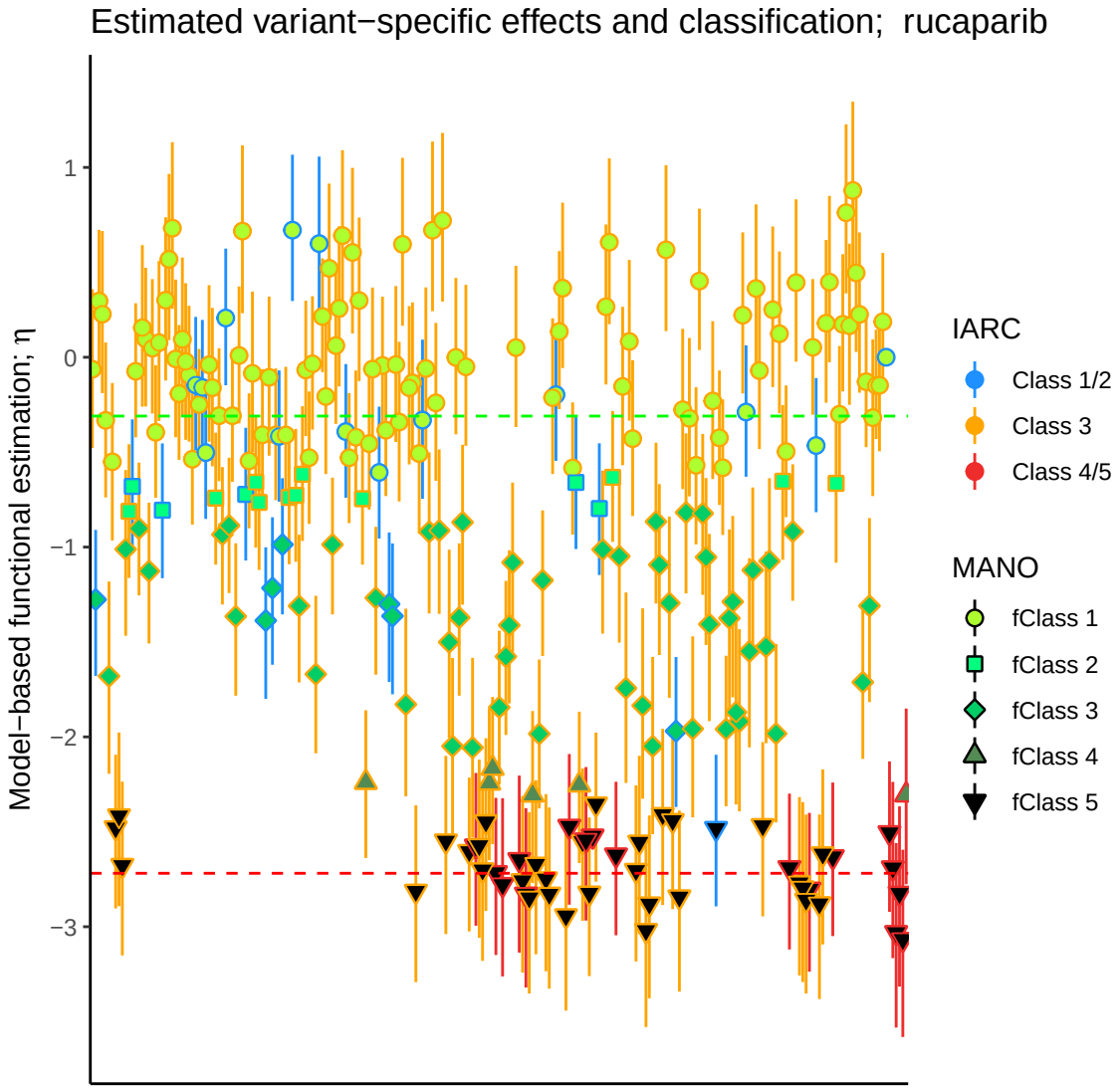
Reorder by position or functional score as you want.

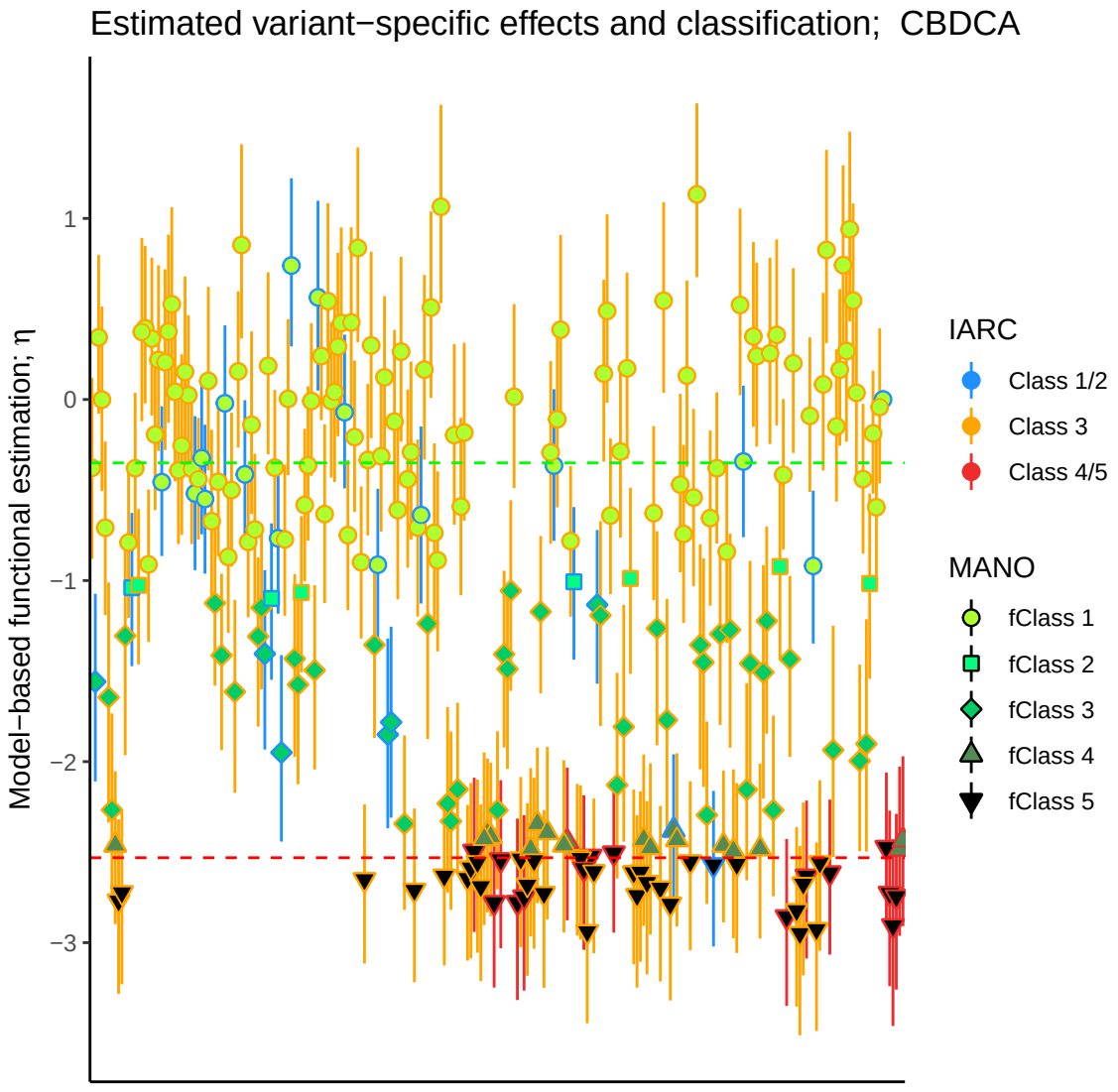
```
est_ <- NULL
for(i in 1:Drug_No){
  len <- length(v_list[[i]])
  eta <- NULL
  eta$IARC <- as.factor(IARC[i][[1]])
  eta$aGVGD <- as.factor(aGVGD[i][[1]])
  eta$Iclass <- as.factor(Iclass[i][[1]])
  eta$ave <- df_eta[[i]][1:len]
  eta$pi <- df_pp[[i]][seq(1,(len*2-1),2)]
  eta$le95 <- df_eta[[i]][(len*3+1):(len*4)]
  eta$ue95 <- df_eta[[i]][(len*7+1):(len*8)]
  eta$patho <- df_p[[i]][seq(1,(len*2-1),2)]
  eta$ben <- df_p[[i]][seq(2,(len*2),2)]
  eta$BF <- eta$patho/eta$ben
  eta$name <- v_list[[i]]
  eta$position <- position[[i]]

  for(j in 1:len){
    if(eta$position[j] == 485)
      eta$position[j] <- 3502
    if(eta$position[j] == 997)
      eta$position[j] <- 3503
    if(eta$position[j] == 1502)
      eta$position[j] <- 3504
    if(eta$position[j] == 1984)
      eta$position[j] <- 3505
    if(eta$position[j] == 2535)
      eta$position[j] <- 3506
  }
  eta$MANO_diagnosis[which(eta$BF<31.2)] <- "fClass 3"
  eta$MANO_diagnosis[which(eta$BF<=0.032)] <- "fClass 2"
  eta$MANO_diagnosis[which(eta$BF<=0.01)] <- "fClass 1"
  eta$MANO_diagnosis[which(eta$BF>=31.2)] <- "fClass 4"
  eta$MANO_diagnosis[which(eta$BF>=100)] <- "fClass 5"

  est_ <- c(est_, list(eta))
  fact_IARC <- factor(eta$IARC,
              levels = c("Class 1/2","Class 3","Class 4/5"))
  fact_MANO <- factor(eta$MANO_diagnosis,
              levels = c("fClass 1","fClass 2","fClass 3",
                         "fClass 4","fClass 5"))

  Colors <- c("dodgerblue", "orange", "firebrick2")
  Colors2 <- c("greenyellow", "springgreen", "springgreen3", "palegreen4","black")
  Shape <- c(21, 22, 23, 24, 25)

  g_eta <- data.frame(eta, IARC=fact_IARC, MANO=fact_MANO)

  g <- ggplot(g_eta, aes(x=reorder(name,position),  #sort by position
# g <- ggplot(g_eta, aes(x=reorder(name,-ave),      #sort by eta value
```

```
        y=ave, ymin=le95,  ymax=ue95, order=position,
        colour=IARC, fill=MANO, shape=MANO))

  g <- g + theme_classic()
  g <- g + geom_pointrange()
  g <- g + geom_point(size = 2.5)
  g <- g + scale_shape_manual(values = Shape)
  g <- g + theme(axis.text.x = element_blank(), panel.background = element_rect(fill="white"))
  g <- g + xlab("")
  g <- g + ylab(expression("Model-based functional estimation; "*eta))
  g <- g + geom_hline(yintercept=(df_mu[[i]][1]),linetype="dashed",colour="red")
  g <- g + geom_hline(yintercept=(df_mu[[i]][2]),linetype="dashed",colour="green")
  g <- g + theme(axis.text.x = element_text(angle = 90, hjust = 1))
  g <- g + ggtitle(paste("Estimated variant-specific effects",
          "and classification; ",Drug[[i]]))
  g <- g + theme(axis.ticks.x = element_blank(), axis.text.x = element_blank())
  g <- g + scale_color_manual(values = Colors)
  g <- g + scale_fill_manual(values = Colors2)
  plot(g)
  filename <- paste('BF-', Drug[[i]], '.txt', sep='')
  write.table(g_eta, file=filename, sep='\t',
            quote=FALSE, col.names=NA)
}
```


Estimated variant−specific effects and classification;  olaparib
Model−based functional estimation; η
1
0
−1
−2
−3
IARC
Class 1/2
Class 3
Class 4/5
MANO
fClass 1
fClass 2
fClass 3
fClass 4
fClass 5


Estimated variant−specific effects and classification;  niraparib
Model−based functional estimation; η
1
0
−1
−2
−3
−4
IARC
Class 1/2
Class 3
Class 4/5
MANO
fClass 1
fClass 2
fClass 3
fClass 4
fClass 5


Estimated variant−specific effects and classification;  rucaparib
Model−based functional estimation; η
1
0
−1
−2
−3
IARC
Class 1/2
Class 3
Class 4/5
MANO
fClass 1
fClass 2
fClass 3
fClass 4
fClass 5




# 6 Posterior Predictive Checks

## 6.1 Sampling from posterior distributions

Draw 512 parameter samples from the trace randomly, then generate 612 random values per variant from a normal distribution, taking batch-to-batch ratio into account.

```
repN <- 512
predY <- NULL
for(i in 1:Drug_No){
  BN <- max(d_[[i]]$batch)
  BN_ <- NULL
  for(j in 1:BN){
    BN_ <- c(BN_,length(subset(d_[[i]], batch == j)$batch)/3)
  }
```

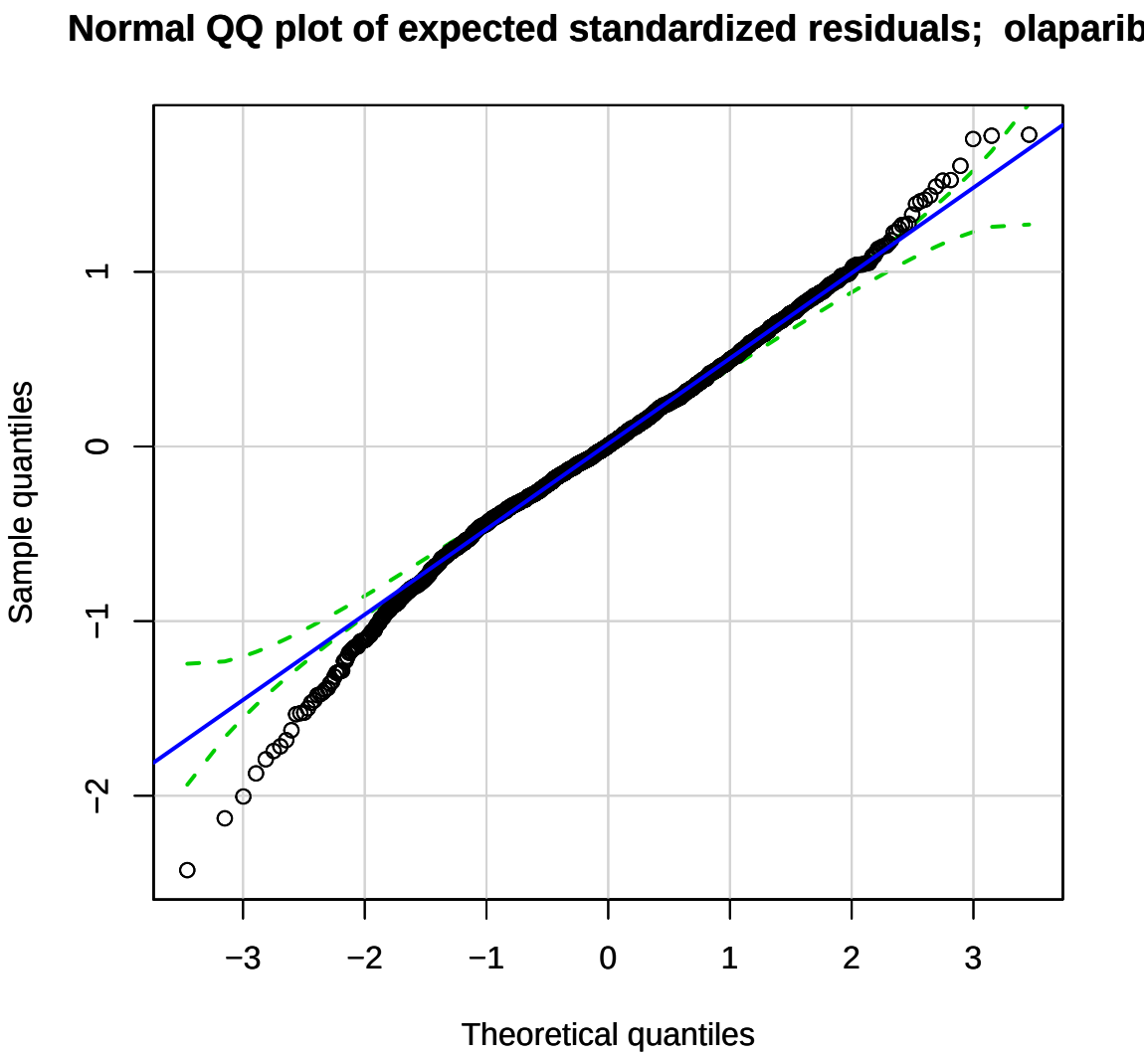
```
  NY <- length(v_list[[i]])
  p_Y <- array(NA,c(repN, sum(BN_),NY))
  d.ext<-rstan::extract(fit_[[i]])
  for(k in 1:repN){
    for(j in 1: NY){
      tmp <- 0
      for(l in 1: BN){
        tmp = tmp + BN_[l]
        p_Y[k,(tmp-BN_[l]+1):tmp,j] <-
        rnorm(BN_[l], d.ext$bet_[k,l] + d.ext$tau_[k,l]*d.ext$eta_[k,j],
                            d.ext$tau_[k,l]*d.ext$psi_[k])
      }
    }
  }
  predY <- c(predY, list(p_Y))
}
```

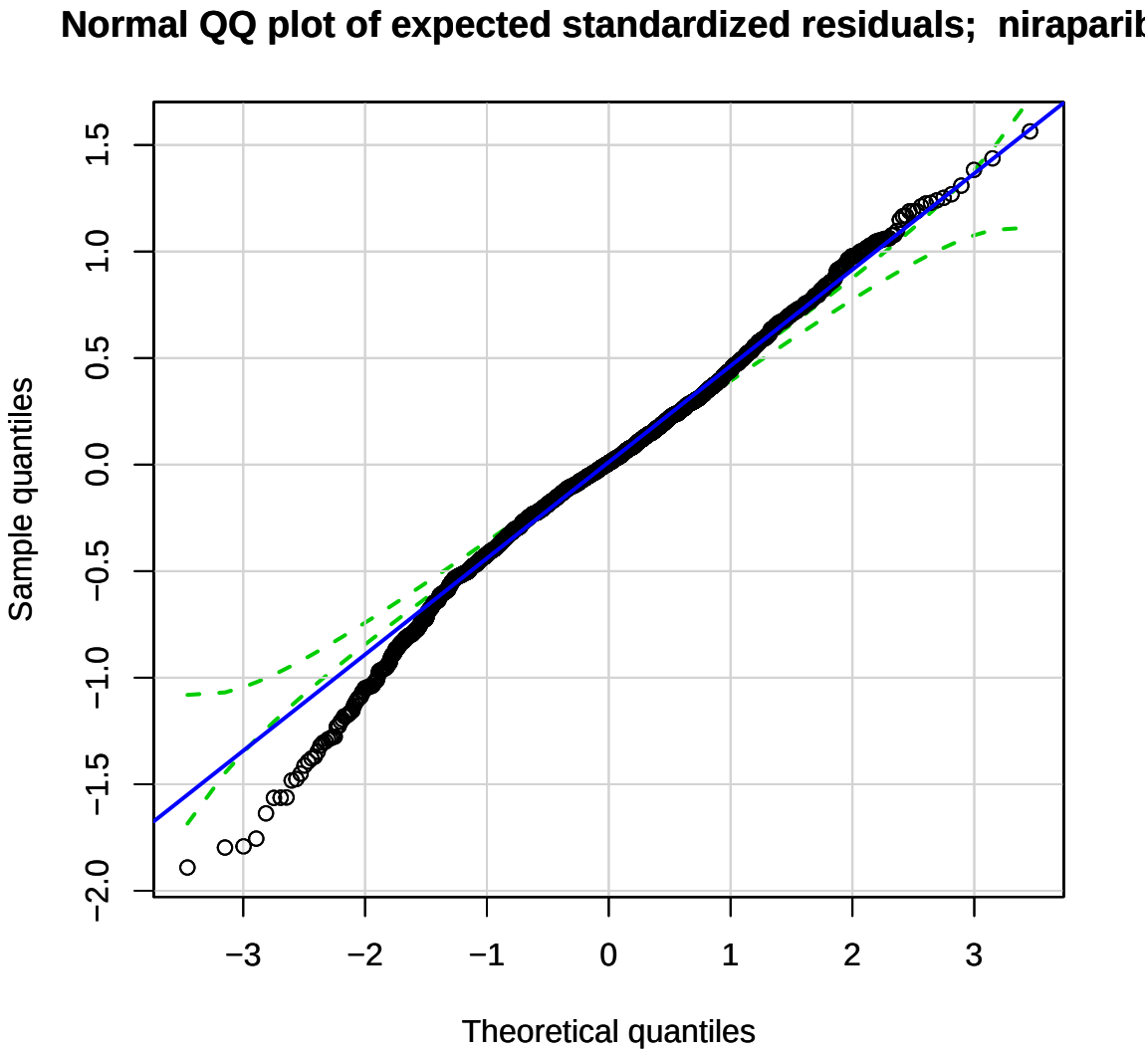
## 6.2 Normal QQ plots of standardized residuals

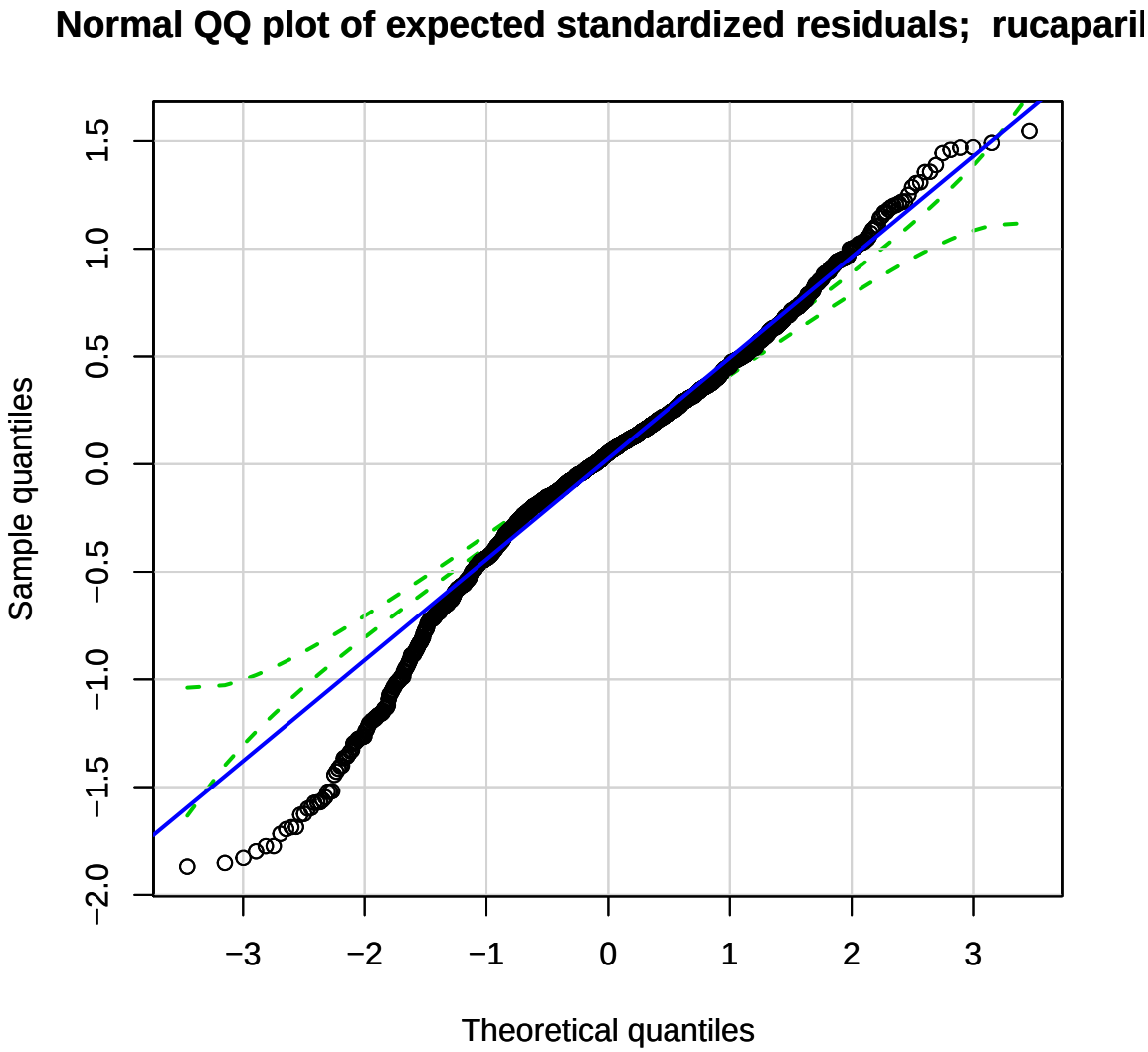
Quantile-quantile plots (QQ plots) for standardized residuals from our fit of the all observed data. When the residuals are assumed to be normally distributed, a normal QQ plot is used to check that assumption. Standardized residuals were averaged over posterior uncertainty in the model's parameters. The green lines indicate a simultaneous 95% confidence intervals. If the model fit perfectly to the observed data, all plots are on the 45-degree reference line.

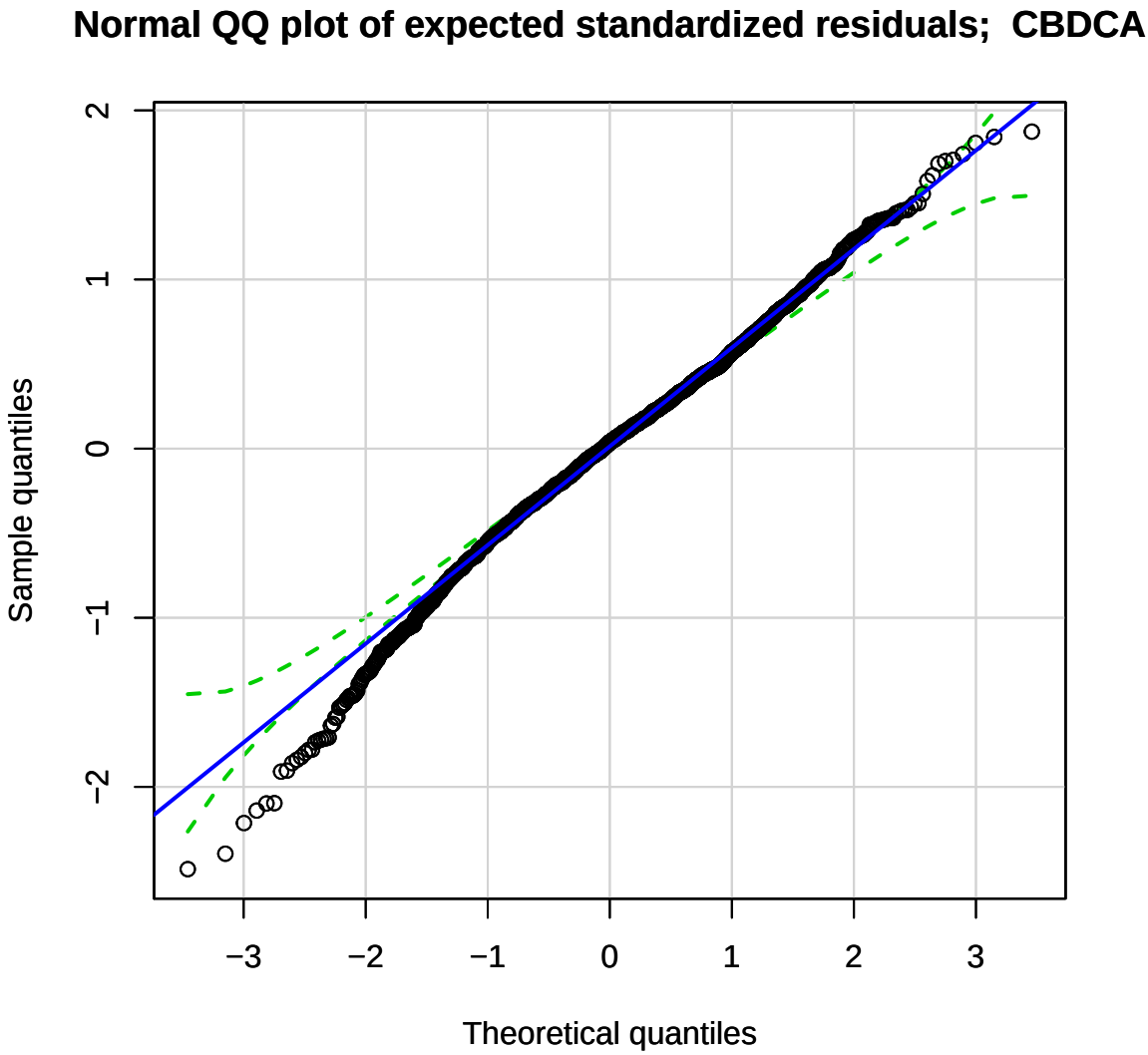
```
for(i in 1:Drug_No){
  y_obs <- d_[[i]]$score
  len <- length(v_list[[i]])
  y_ave <- rep(0,len)
  y_sd <- rep(0,len)
  y_res <- rep(0,len)
  y_std_res <- rep(0,len)
  for(j in 1:(length(d_[[i]]$var_name))){
    y_ave[j] <- mean(predY[[i]][,,d_[[i]]$var_num[j]])
  }
  y_sd <- sd(y_ave[1:1836])
  y_res <- y_obs - y_ave
  y_std_res <- rep(0,len)
  for(j in 1:(length(d_[[i]]$var_name))){
      y_std_res[j] <- y_res[j] / y_sd
  }
  car::qqPlot(y_std_res, distribution="norm",
              ylab=paste("Sample quantiles"),
              xlab=paste("Theoretical quantiles"),
              main=paste("Normal QQ plot of expected standardized",
                         "residuals; ", Drug[[i]]),
              col.lines=carPalette()[7],
              envelope=.95, line="none",id=FALSE)
  par(new=T)
  car::qqPlot(y_std_res,distribution="norm",
              ylab=paste("Sample quantiles"),
              xlab=paste("Theoretical quantiles"),
              main=paste("Normal QQ plot of expected standardized",
```

```
                        "residuals; ", Drug[[i]]),
                col.lines=carPalette()[2],
                envelope=FALSE, line="robust", id=FALSE)
}
```




Normal QQ plot of expected standardized residuals; niraparil
Sample quantiles
Theoretical quantiles
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
−3
−2
−1
0
1
2
3


Normal QQ plot of expected standardized residuals;  rucapari
Sample quantiles
Theoretical quantiles
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
−3
−2
−1
0
1
2
3

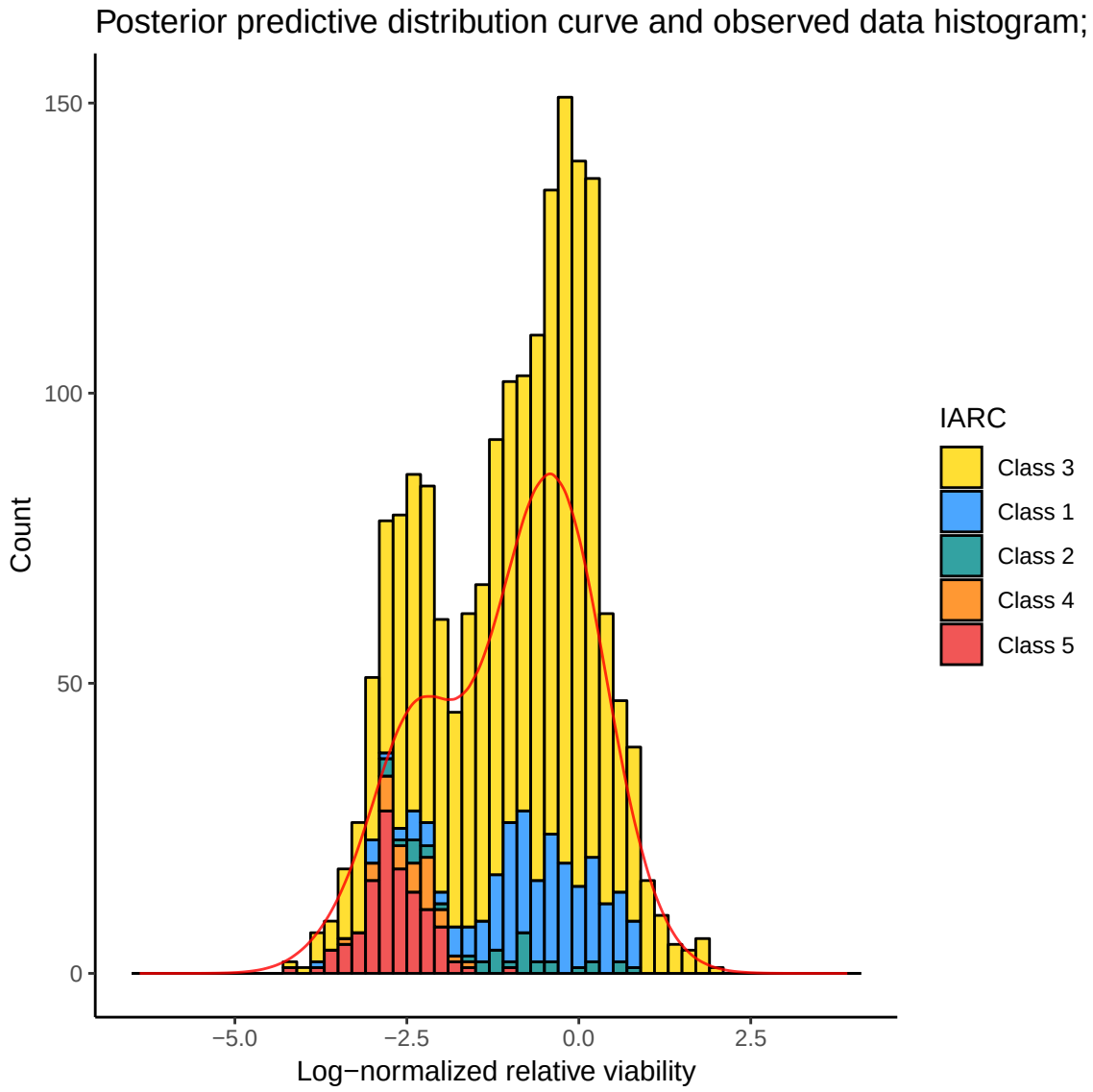


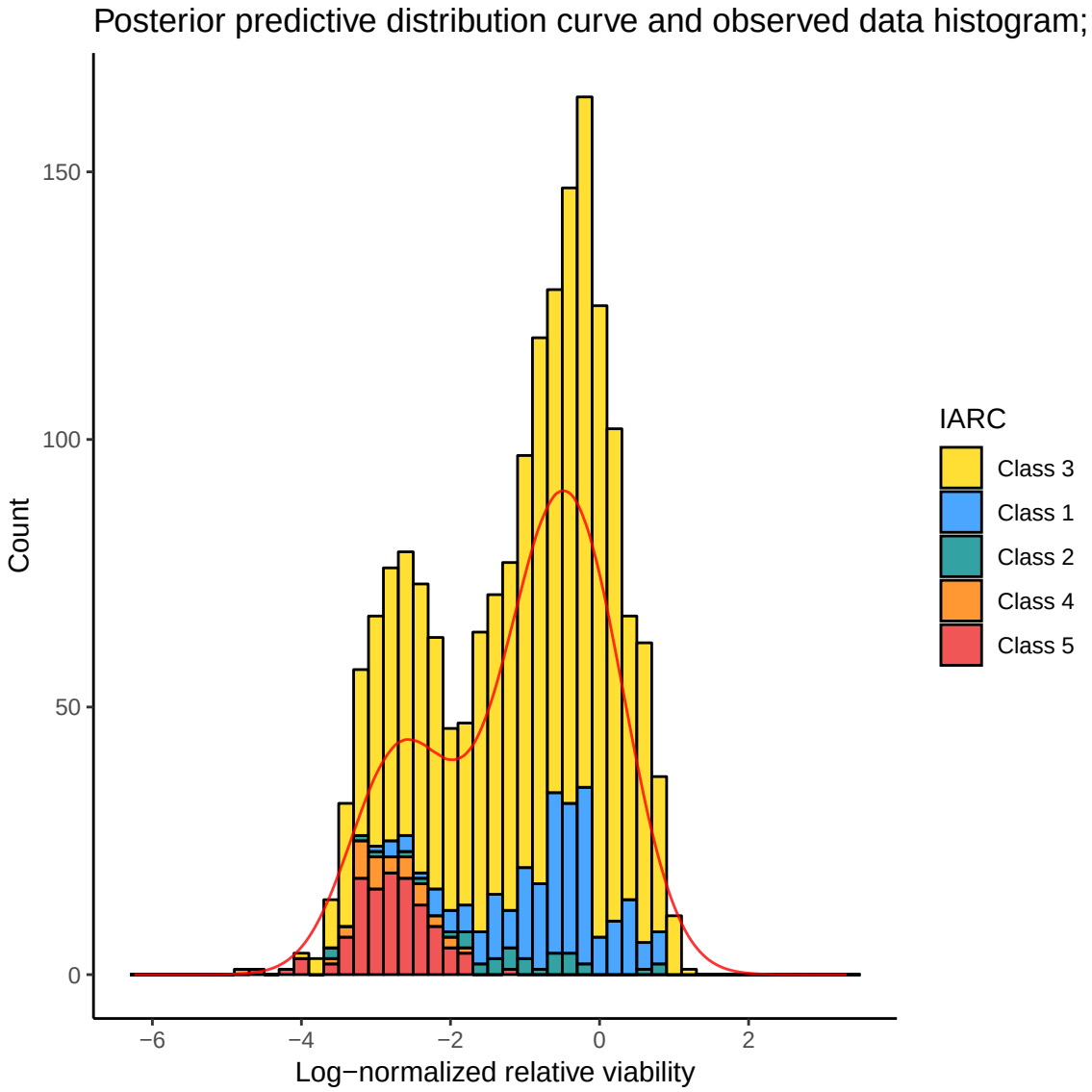

### 6.3 Estimated density curves and observed data histograms

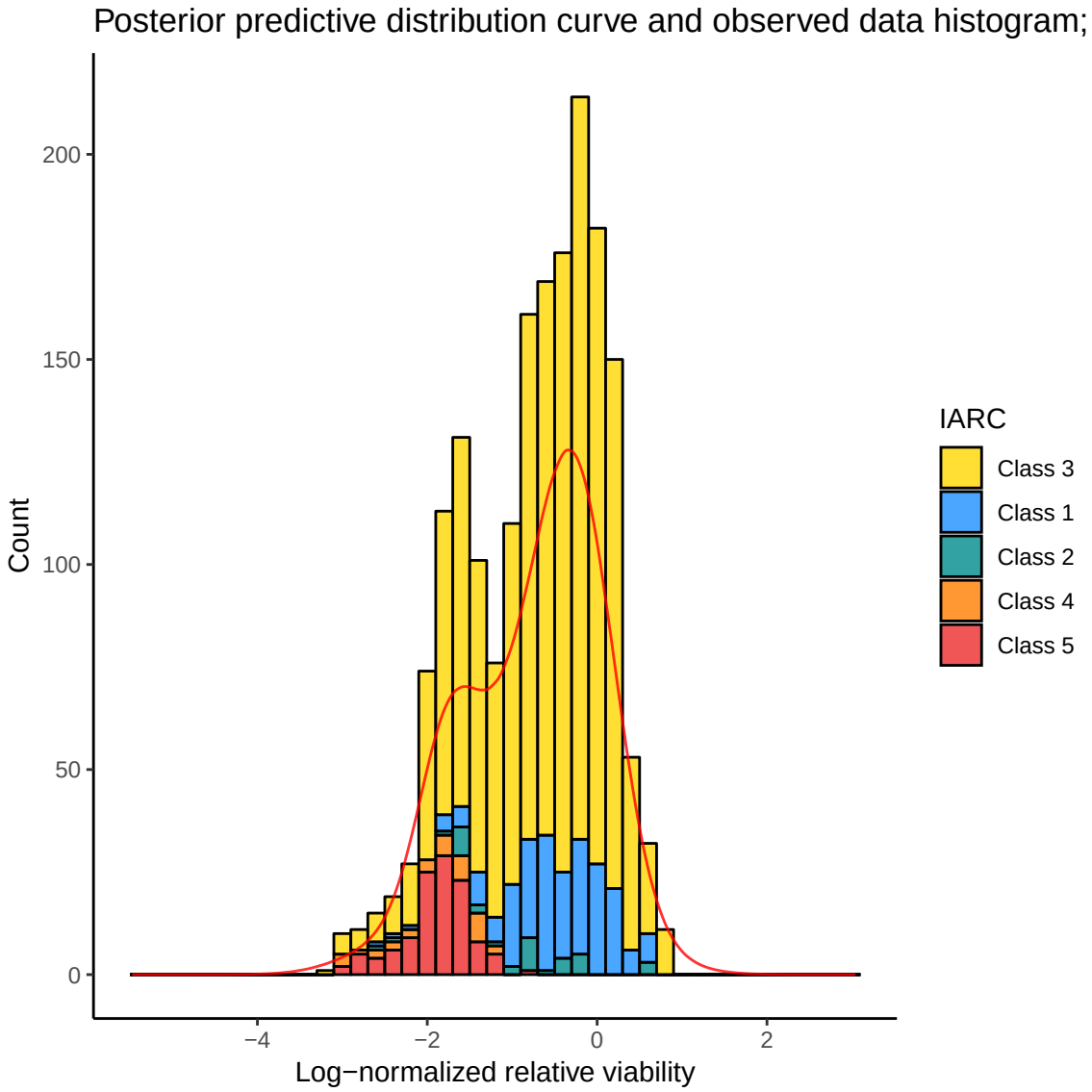
Visual check the fit of predictive values to observed data using histogram and probability distribution line. A posterior predictive density curve of log-normalized relative viability data was computed from 612 generated data sets containing 512 samples each.

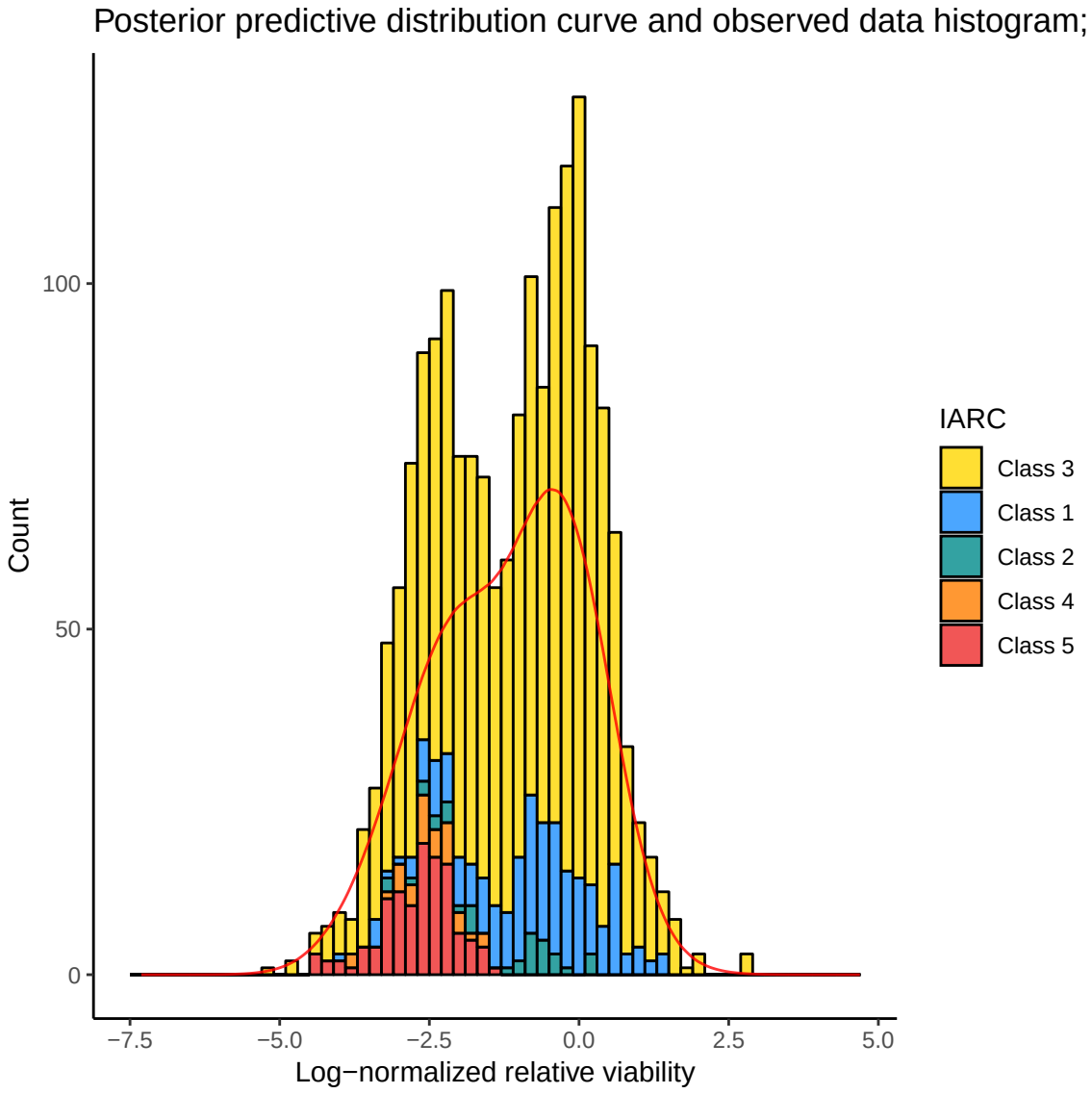
```
for(i in 1:Drug_No){
  m.pDat <- melt(predY[[i]],idvar=c())
  fact_IARC <- factor(d_[[i]]$IARC_class,
              levels = c("Class 3", "Class 1","Class 2",
                         "Class 4","Class 5"))
  m.Dat <- data.frame(d_[[i]]$score, fact_IARC)
  colnames(m.Dat) <- c("value","IARC")
  m.Dat <- melt(m.Dat)
  Colors <- c("gold1", "dodgerblue", "cyan4", "darkorange1","firebrick2")
  g <- ggplot(m.Dat,aes(x=value))
```

```
  g <- g + theme_classic()
  g <- g + ggtitle(paste("Posterior predictive distribution curve",
                         "and observed data histogram; ", Drug[[i]]))
  g <- g + xlab("Log-normalized relative viability") + ylab("Count")
  g <- g + geom_histogram(data=m.Dat,aes(y=..count..,fill=IARC),
        alpha=0.8, position = "stack",colour="black", binwidth = 0.2)
  g <- g + geom_line(data=m.pDat,aes(x=value,y=..density..*NY),
        alpha=0.8,stat="density",col ="red")
  g <- g + scale_color_manual(values = Colors)
  g <- g + scale_fill_manual(values = Colors)
  plot(g)
}
```




Posterior predictive distribution curve and observed data histogram;
Count
150
100
50
0
−6
−4
−2
0
2
Log−normalized relative viability
IARC
Class 3
Class 1
Class 2
Class 4
Class 5


Posterior predictive distribution curve and observed data histogram;
Count
200
150
100
50
0
−4
−2
0
2
Log−normalized relative viability
IARC
Class 3
Class 1
Class 2
Class 4
Class 5

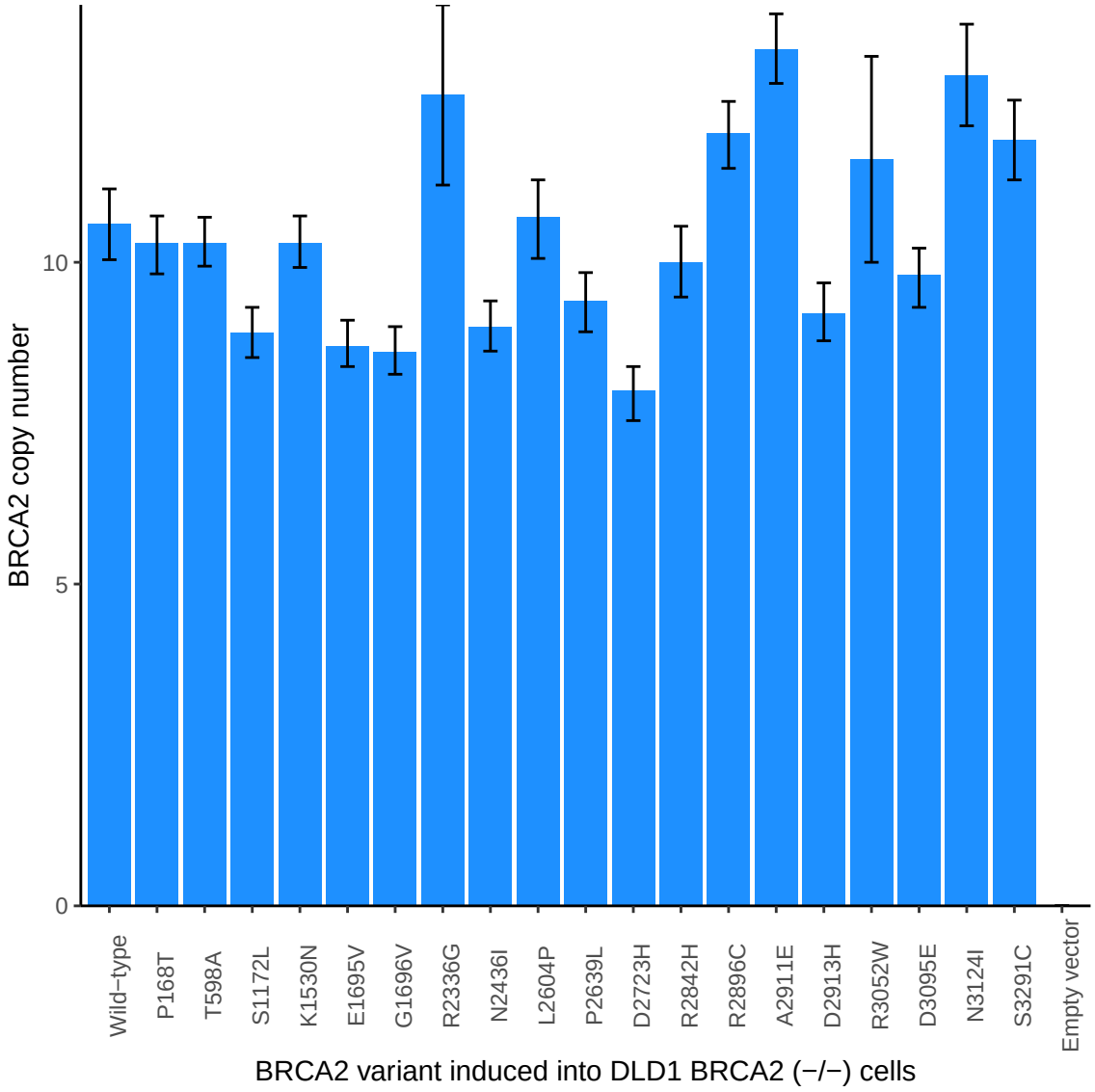



# 7 Digital droplet PCR

Histograms of the copy numbers of integrated BRCA2 cDNA.

```
RT <- read.csv(file = "RT-PCR.csv")
Colors <- c("dodgerblue")
g <- ggplot(RT, aes(x = reorder(x = variant, X = No), y = copy, fill = "grey40"))
g <- g + geom_bar(stat = "identity")
g <- g + geom_errorbar(aes(ymin = copy_95max, ymax = copy_95min, width = 0.3))
g <- g + theme_classic()
g <- g + theme(legend.position = 'none')
g <- g + theme(axis.text.x=element_text(angle = 90))
g <- g + xlab("BRCA2 variant induced into DLD1 BRCA2 (-/-) cells")
g <- g + ylab("BRCA2 copy number")
g <- g + scale_y_continuous(expand=c(0,0))
```

```
g <- g + scale_color_manual(values = Colors)
g <- g + scale_fill_manual(values = Colors)
plot(g)
```

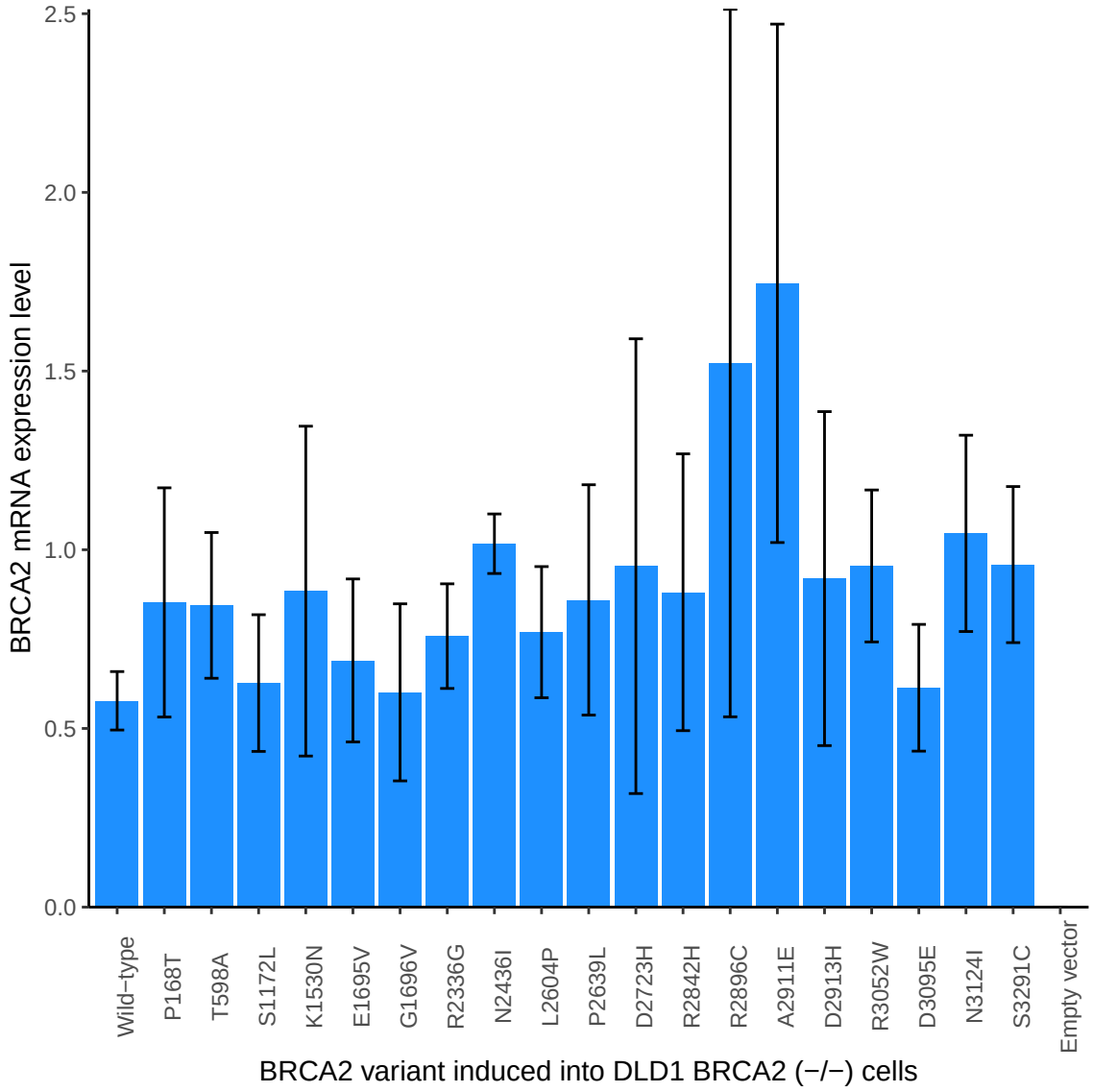



## 8 Real time quantitative RT-PCR

The Kruskal–Wallis one-way analysis of the variance of mRNA expression level. Error bars: standard deviation.

```
g <- ggplot(RT, aes(x = reorder(x = variant, X = No), y = average, fill = "grey40"))
g <- g + geom_bar(stat = "identity")
g <- g + geom_errorbar(aes(ymin = average - std, ymax = average + std, width = 0.3))
g <- g + theme_classic()
g <- g + theme(legend.position = 'none')
g <- g + theme(axis.text.x=element_text(angle = 90))
```

```
g <- g + xlab("BRCA2 variant induced into DLD1 BRCA2 (-/-) cells")
g <- g + ylab("BRCA2 mRNA expression level")
g <- g + scale_y_continuous(expand=c(0,0))
g <- g + scale_color_manual(values = Colors)
g <- g + scale_fill_manual(values = Colors)
plot(g)
```




```
Anov_ <- melt(data.frame(head(RT[2:5],20)), variable_name = "variant")
shapiro.test(x=Anov_$value)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Anov_$value
## W = 0.85458, p-value = 0.000004169
```

```
val_=Anov_$value
var_=Anov_$variant
res <-  kruskal.test(val_~var_)
res
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  val_ by var_
## Kruskal-Wallis chi-squared = 20.397, df = 19, p-value = 0.3711
```

# 9 Session information

```
sessionInfo()
```

```
## R version 3.5.3 (2019-03-11)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.2 LTS
##
## Matrix products: default
## BLAS: /usr/lib/x86_64-linux-gnu/openblas/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/libopenblasp-r0.2.20.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] shinystan_2.5.0    shiny_1.3.2        car_3.0-3
##  [4] carData_3.0-2      pROC_1.14.0        ggsci_2.9
##  [7] reshape2_1.4.3     forcats_0.4.0      stringr_1.4.0
## [10] purrr_0.3.2        readr_1.3.1        tibble_2.1.2
## [13] tidyverse_1.2.1    mclust_5.4.3       rstan_2.18.2
## [16] StanHeaders_2.18.1 ggmcmc_1.2         tidyr_0.8.3
## [19] dplyr_0.8.1        ggplot2_3.1.1      knitr_1.23
##
## loaded via a namespace (and not attached):
##  [1] nlme_3.1-140       matrixStats_0.54.0 xts_0.11-2
##  [4] lubridate_1.7.4    threejs_0.3.1      RColorBrewer_1.1-2
##  [7] httr_1.4.0         tools_3.5.3        backports_1.1.4
## [10] DT_0.6             R6_2.4.0           lazyeval_0.2.2
## [13] colorspace_1.4-1   withr_2.1.2        tidyselect_0.2.5
## [16] gridExtra_2.3      prettyunits_1.0.2  GGally_1.4.0
## [19] processx_3.3.1     curl_3.3           compiler_3.5.3
## [22] cli_1.1.0          rvest_0.3.4        shinyjs_1.0
```

```
## [25] xml2_1.2.0         labeling_0.3       colourpicker_1.0
## [28] dygraphs_1.1.1.6   scales_1.0.0       ggridges_0.5.1
## [31] callr_3.2.0        digest_0.6.19      foreign_0.8-71
## [34] rmarkdown_1.13     rio_0.5.16         base64enc_0.1-3
## [37] pkgconfig_2.0.2    htmltools_0.3.6    htmlwidgets_1.3
## [40] rlang_0.3.4        readxl_1.3.1       rstudioapi_0.10
## [43] generics_0.0.2     zoo_1.8-6          jsonlite_1.6
## [46] gtools_3.8.1       crosstalk_1.0.0    zip_2.0.2
## [49] inline_0.3.15      magrittr_1.5       loo_2.1.0
## [52] bayesplot_1.7.0    Rcpp_1.0.1         munsell_0.5.0
## [55] abind_1.4-5        stringi_1.4.3      yaml_2.2.0
## [58] MASS_7.3-51.4      pkgbuild_1.0.3     plyr_1.8.4
## [61] grid_3.5.3         parallel_3.5.3     promises_1.0.1
## [64] crayon_1.3.4       miniUI_0.1.1.1     lattice_0.20-38
## [67] haven_2.1.0        hms_0.4.2          ps_1.3.0
## [70] pillar_1.4.1       igraph_1.2.4.1     markdown_0.9
## [73] stats4_3.5.3       glue_1.3.1         evaluate_0.13
## [76] data.table_1.12.2  modelr_0.1.4       httpuv_1.5.1
## [79] cellranger_1.1.0   gtable_0.3.0       reshape_0.8.8
## [82] assertthat_0.2.1   xfun_0.7           openxlsx_4.1.0.1
## [85] mime_0.6           xtable_1.8-4       broom_0.5.2
## [88] later_0.8.0        rsconnect_0.8.13   shinythemes_1.1.2
```