Analyse de données d'expression, TD 1

Christophe Ambroise

October 5, 2018

Table 1: Experience de 10 puces à deux canaux

R	190	266	144	268	254	208	194	206	130	94
G	20	90	36	96	85	67	67	70	23	40

Normalisation par Normal Score

Des données d'expression de 10 gènes d'une puce à deux canaux sont observées

1. Quelles sont les moyennes des expressions dans les deux canaux
2. Tracer les histogrammes par canaux et le $\log_2(\frac{R}{G})$
3. Tracer le MA plot
4. Faite la normalisation par normal score
5. Tracer les histogrammes par canaux et le $\log_2(\frac{R_{ns}}{G_{ns}})$
6. Retracer le MA plot
7. Commenter

Solution

```
> p<-length(R)
> M<-log2(R/G)
> A<-1/2*log2(R*G)
> par(mfrow=c(1,2))
> plot(A,M,main="MA plot sur données brutes")
> Mns<-qnorm(rank(R)/(p+1))  - qnorm(rank(G)/(p+1))
> Ans<-qnorm(rank(R)/(p+1))  + qnorm(rank(G)/(p+1))
> plot(Ans,Mns,main="MA plot sur données normalisées")
```

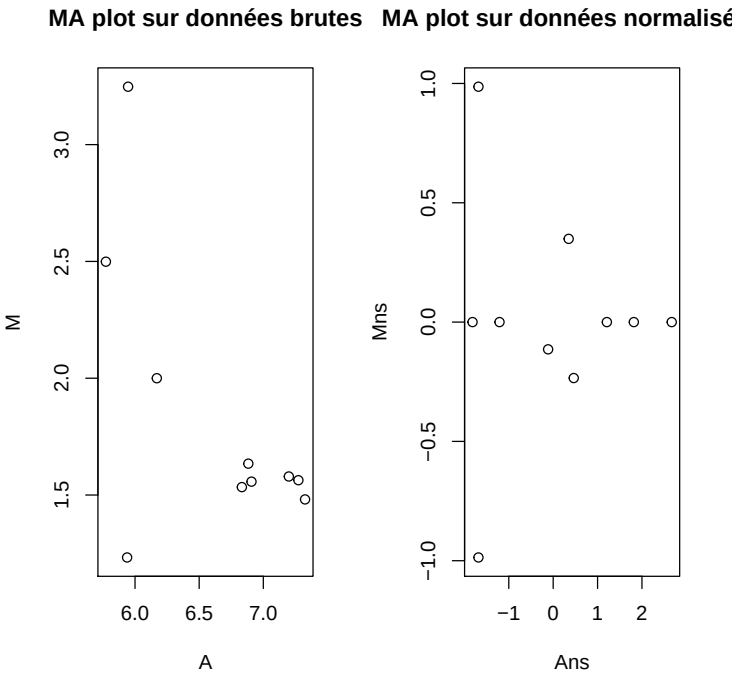

Table 2: Experience de 10 puces à deux canaux

R	190	266	144	268	254	208	194	206	130	94
G	20	90	36	96	85	67	67	70	23	40

Exercice: normalisation par quantile

Des données d'expression de 10 gènes d'une puce à deux canaux sont observées

1. Quelles sont les moyennes des expressions dans les deux canaux
2. Tracer les histogrammes par canaux et le $\log_2(\frac{R}{G})$
3. Tracer le MA plot
4. Faite la normalisation par quantile sur papier.
5. Ecrire une fonction R `quantile.normalization`.
6. Tracer les histogrammes par canaux après normalisation et le $\log_2(\frac{R_{nq}}{G_{nq}})$
7. Retracer le MA plot
8. Commenter

Solution

```
> quantile.normalization <- function(df){
+   df_rank <- apply(df,2,rank,ties.method="min")
+   df_sorted <- data.frame(apply(df, 2, sort))
+   df_mean <- apply(df_sorted, 1, mean)
+
+   index_to_mean <- function(my_index, my_mean){
+     return(my_mean[my_index])
+   }
+
+   df_final <- apply(df_rank, 2, index_to_mean, my_mean=df_mean)
+   rownames(df_final) <- rownames(df)
+   return(data.frame(df_final))
+ }
> p<-length(R)
> M<-log2(R/G)
> A<-1/2*log2(R*G)
> Xqs<-quantile.normalization(X)
> Mqs<-log2(Xqs$R/Xqs$G)
> Aqs<-1/2*log2(Xqs$R*Xqs$G)
> par(mfrow=c(1,2))
> plot(A,M,main="MA plot sur données brutes")
> plot(Aqs,Mqs,main="MA plot sur données normalisées")
```

Table 3: Microarray experiment

	array 1	array 2	array 3
gene 1	14	15	14
gene 2	7	4	7
gene 3	8	2	10
gene 4	15	9	10
gene 5	0	2	0

Exercice: Median polish

1. Sur le tableau résumé des expériences ci-dessous, calculer sur le papier le tableau des résidus de l'algorithme "Median polish" de Tuckey.

2. Écrire la formule de reconstitution du tableau en fonction des résidus, des effets lignes et colonnes.

3. Écrire votre propre fonction `median.polish` et comparer vos résultats

Solution

```
> med.d <- medpolish(data.microarrays,trace.iter = TRUE)

1: 19
Final: 19

> print(med.d)

Median Polish Results (Dataset: "data.microarrays")

Overall: 8

Row Effects:
gene  1 gene  2 gene  3 gene  4 gene  5
      6      -1       0       2      -8

Column Effects:
array  1 array  2 array  3
       0       -1        0

Residuals:
        array  1 array  2 array  3
gene  1        0        2        0
gene  2        0       -2        0
gene  3        0       -5        2
gene  4        5        0        0
gene  5        0        3        0

> ## Check decomposition:
> all(data.microarrays ==
+     med.d$overall + outer(med.d$row,med.d$col, "+") + med.d$residuals)

[1] TRUE

> my.median.polish<-function (x, eps = 0.01, maxiter = 10L, trace.iter = TRUE, na.rm = FALSE
+ {
+     z <- as.matrix(x)
+     nr <- nrow(z)
+     nc <- ncol(z)
+     t <- 0
+     r <- numeric(nr)
+     c <- numeric(nc)
+     oldsum <- 0
+     for (iter in 1L:maxiter) {
+         # Compute the median for each row
+         rdelta <- apply(z, 1L, median, na.rm = na.rm)
+         # Substract the row median  to each row
```

```
+         z <- z - matrix(rdelta, nrow = nr, ncol = nc)
+         # Update the row effect
+         r <- r + rdelta
+         # Compute the median of the column effect
+         delta <- median(c, na.rm = na.rm)
+         # Move this median from the column to the total
+         c <- c - delta
+         t <- t + delta
+         # Compute the median of each column
+         cdelta <- apply(z, 2L, median, na.rm = na.rm)
+         # Substract the column median to each column
+         z <- z - matrix(cdelta, nrow = nr, ncol = nc, byrow = TRUE)
+         # Update the column effect
+         c <- c + cdelta
+         # Compute the median of the row effect
+         delta <- median(r, na.rm = na.rm)
+         # Move this median from the column to the total
+         r <- r - delta
+         t <- t + delta
+         # Are we stable
+         newsum <- sum(abs(z), na.rm = na.rm)
+         converged <- newsum == 0 || abs(newsum - oldsum) < eps *
+             newsum
+         if (converged)
+             break
+         oldsum <- newsum
+         if (trace.iter)
+             cat(iter, ": ", newsum, "\n", sep = "")
+     }
+     if (converged) {
+         if (trace.iter)
+             cat("Final: ", newsum, "\n", sep = "")
+     }
+     else warning(sprintf(ngettext(maxiter, "medpolish() did not converge in %d iteration",
+         "medpolish() did not converge in %d iterations"), maxiter),
+         domain = NA)
+     names(r) <- rownames(z)
+     names(c) <- colnames(z)
+     ans <- list(overall = t, row = r, col = c, residuals = z,
+         name = deparse(substitute(x)))
+     class(ans) <- "medpolish"
+     ans
+ }
```