Normalisation de données transcriptomiques bi-couleur par le package `limma` (Linear Models for Microarray Data)

Mots-clés : Importation de données Genepix, Filtrage de spots, Normalisation, Loess.

David Causeur (david.causeur@agrocampus-ouest.fr)

Table des matières

1 Préambule

Le package `limma` est spécialement conçu pour analyser les données de biopuces bi-couleurs. Il permet l'importation de fichiers de sortie des logiciels d'analyse d'image les plus courants dans le domaine des biopuces, la normalisation des données et l'analyse différentielle (pour une présentation détaillée, voir la page web `http://bioinf.wehi.edu.au/limma/`). Ce document se veut une introduction à l'utilisation de `limma` : il ne présente donc pas de manière exhaustive les fonctionnalités du package et se concentre en particulier sur l'importation et la normalisation de données bi-couleurs à partir de fichiers `Genepix`.

Le package fait partie du projet `bioconductor`. Il est donc automatiquement installé via la commande suivante :

```
> source("http://www.bioconductor.org/biocLite.R")
# Chargement du fichier de commandes biocLite.R
# disponible sur la page web http://www.bioconductor.org.
> biocLite()
# La fonction biocLite est définie dans biocLite.R. Elle commande le
# téléchargement automatique des packages (version allégée de Bioconductor).
```

Le chargement du package est ensuite commandé par :

```
> library(limma)    # Chargement du package limma
```

Ceux qui veulent abandonner ici la lecture de ce document et s'aventurer seuls dans l'apprentissage de `limma` peuvent le faire à partir de la commande suivante, qui provoque l'ouverture d'un document d'aide :

```
> limmaUsersGuide()    # Ouvre un document PDF d'aide pour limma
```

Dans les sections suivantes sont présentées les procédures d'importation et de normalisation de données `Genepix` contenues dans 46 fichiers d'images, présents dans le répertoire `C:\chezmoi\`. Afin d'éviter d'avoir à indiquer de manière répétée le répertoire de travail au cours de la procédure d'importation des données, on peut le définir comme répertoire de travail pour l'ensemble de la session :

```
> setwd("C:/chezmoi/")    # définit le répertoire C:\chezmoi\
                          # comme répertoire de travail
```

2 Importation des données par `limma`

Le format des données de biopuces nécessaire à toute analyse statistique est un tableau dont les lignes sont associées à des puces et les colonnes à des gènes. Or, le plus souvent le point de départ de l'analyse repose sur autant de fichiers de sortie d'un logiciel d'analyse

d'images qu'il y a de puces. Le package `limma` prévoit donc une procédure d'importation de ces fichiers conduisant à la création d'un tableau de données.

La conversion de plusieurs fichiers en un seul tableau `puces` $\times$ `gènes` nécessite quelques indications permettant de piloter l'ordonnancement des valeurs d'expressions géniques dans le jeu de données. Ces informations sont contenues dans un fichier supplémentaire nommé dans la suite `Targets.txt`.

2.1 Le fichier `Targets.txt`

Le fichier `Targets.txt` doit contenir les colonnes suivantes :

- `FileName` donnant le nom des fichiers d'images,
- `Cy3` donnant la modalité du facteur marquée en vert,
- `Cy5` donnant la modalité du facteur marquée en rouge.

Les autres colonnes sont optionnelles. Dans l'extrait du fichier `Targets.txt` donné dans le tableau 1, une colonne `ArrayName` a par exemple été rajoutée afin d'identifier de manière lisible les puces et par conséquent les lignes du jeu de données à construire.

Factor	ArrayName	...	Cy3	Cy5	FileName
F	F1	...	1089	ref	refCy3-1089Cy5.gpr.txt
F	F10	...	3789	ref	refCy3-3789Cy5.gpr.txt
F	F11	...	3743	ref	refCy3-3743Cy5.gpr.txt
F	F12	...	499	ref	refCy3-499Cy5.gpr.txt
F	F13	...	516B	ref	refCy3-516BCy5.gpr.txt
F	F14	...	3793	ref	refCy3-3793Cy5.gpr.txt
⋮	⋮	⋮		⋮	⋮

TAB. 1: *Extrait d'un fichier* ***`Targets.txt`***

La commande suivante permet de lire le fichier `Targets.txt` et de créer un objet `R`, nommé ici `targets`, héritant de l'information contenue dans le fichier :

```
> targets=readTargets("targets.txt",sep="\t")    # Lecture de Targets.txt
```

L'argument `sep` définit le type de séparateur utilisé pour délimiter les colonnes dans le fichier `Targets.txt`. Ici, on précise que le séparateur est la tabulation (`\t` – en fait, c'est la valeur par défaut de l'argument `sep` : dans le cas présent, il n'est donc pas utile de le définir).

2.2 Filtrage de spots

Lors de la lecture des fichiers de sortie de logiciels d'analyse d'image, certaines fonctionnalités de `limma` permettent de contrôler la qualité des données d'expressions en chaque spot. En effet, ces fichiers contiennent eux-mêmes différentes indications sur la qualité des spots. Par exemple, les fichiers `Genepix` sont structurés comme des tableaux de données dont les lignes correspondent aux spots et les colonnes à des caractéristiques des données en chacun de ces spots. On trouve parmi ces caractéristiques :

- `Flags` : type de Flag
- `SNR 635` : rapport signal-bruit de fond pour le signal rouge
- `SNR 532` : rapport signal-bruit de fond pour le signal vert
- `F635 Median` : intensité médian pour le signal rouge
- `F532 Median` : intensité médiane pour le signal vert
- `F635 Mean` : intensité moyenne pour le signal rouge
- `F532 Mean` : intensité moyenne pour le signal vert

Il est donc possible de créer sa propre fonction de filtrage de spots : cette fonction, dont l'argument principale est le tableau `spots` $\times$ `caractéristiques` d'un fichier `Genepix`, prendra la valeur 1 si un spot est de bonne qualité et 0 sinon.

Par exemple, si on souhaite supprimer les spots dont les flags de `Genepix` sont inférieurs ou égaux à -49, il suffit de créer la fonction de filtrage suivante :

```
MonFiltrage = function(X) {      # X désigne le tableau Genepix
okFLAG = X$Flags > -49           # okFLAG=TRUE si Flags>-49, FALSE sinon
return(as.numeric(okFLAG)) }     # la fonction as.numeric convertit une valeur logique
# en valeur numérique selon la règle suivante : TRUE ↔ 1, FALSE ↔ 0
```

On peut aussi affiner la procédure de filtrage en s'assurant que le rapport signal-bruit de fond dépasse un certain seuil, nommé ci-après `seuilSNR`. Les commandes suivantes, qui seront insérées dans la fonction `MonFiltrage`, permettent ce test, simultanément pour le signal rouge et le signal vert :

```
okSNRred = X["SNR 635"] > seuilSNR
# okSNRred=TRUE si le rapport signal rouge - bruit de fond > seuilSNR
okSNRgreen = X["SNR 532"] > seuilSNR
# okSNRred=TRUE si le rapport signal vert - bruit de fond > seuilSNR
okSNR = okSNRred & okSNRgreen
# okSNR=TRUE si okSNRred=TRUE et okSNRgreen=TRUE
```

De la même manière, il est parfois souhaitable de filtrer les spots présentant une forte hétérogénéité du signal observé. Cette hétérogénéité se traduit par une grande différence entre le signal moyen et le signal médian. Un test permettant d'identifier simplement les cas d'hétérogénéité consiste donc à s'assurer que le rapport H suivant n'est pas trop grand:

$$H = \frac{|\text{signal moyen} - \text{signal médian}|}{\frac{1}{2}[\text{signal moyen} + \text{signal médian}]}$$

Si `seuilH` désigne la valeur limite au-delà de laquelle le signal est considéré comme hétérogène, les commandes suivantes permettent de tester l'homogénéité des signaux rouge et vert :

```
NumRed = abs(X[,"F635 Median"]-X[,"F635 Mean"])
# NumRed = | signal rouge moyen - signal rouge médian |
DenomRed = 0.5*(X[,"F635 Median"]+X[,"F635 Mean"])
# DenomRed = 1/2 (signal rouge moyen + signal rouge médian)
okHRed = (NumRed/DenomRed) < seuilH
# okHRed=TRUE si le rapport H rouge < seuilH
NumGreen = abs(X[,"F532 Median"]-X[,"F532 Mean"])
# NumGreen = | signal vert moyen - signal vert médian |
DenomGreen = 0.5*(X[,"F532 Median"]+X[,"F532 Mean"])
# DenomGreen = 1/2 (signal vert moyen + signal vert médian)
okHGreen = (NumGreen/DenomGreen) < seuilH
# okHGreen=TRUE si le rapport H vert < seuilH
okH = okHRed & okHGreen
# okH=TRUE si okHred=TRUE et okHgreen=TRUE
```

Finalement, la fonction de filtrage intégrant tous les contrôles de qualité des spots définis ci-dessus prend la forme suivante :

```
MonFiltrage = function(X, seuilSNR=2, seuilH=0.2) {
# Par défaut, seuilSNR=2 et seuilH=0.2
okFLAG = X$Flags > -49
# Début du test sur le rapport signal - bruit
okSNRred = X["SNR 635"] > seuilSNR
okSNRgreen = X["SNR 532"] > seuilSNR
okSNR = okSNRred & okSNRgreen
# Début du test d'hétérogénéité
NumRed = abs(X["F635 Median"]-X["F635 Mean"])
DenomRed = 0.5*(X["F635 Median"]+X["F635 Mean"])
okHRed = (NumRed/DenomRed) < seuilH
NumGreen = abs(X["F532 Median"]-X["F532 Mean"])
DenomGreen = 0.5*(X["F532 Median"]+X["F532 Mean"])
okHGreen = (NumGreen/DenomGreen) < seuilH
okH = okHRed & okHGreen
ok = okFLAG & okSNR & okH
# ok=TRUE si okFLAG=TRUE et okSNR=TRUE et okH=TRUE
return(as.numeric(ok)) }
```

2.3 Lecture des fichiers `Genepix`

La fonction `read.maimages` permet de lire les fichiers `Genepix` :

```
> RG=read.maimages(files=targets$FileName,source="genepix",wt.fun=MonFiltrage)
# wt.fun [pour weight function] sert à spécifier un mode de filtrage
```

Tel que défini ci-dessus, l'objet `RG` peut-être vu comme une *armoire* contenant toute l'information contenue dans les fichiers `Genepix`. Les noms des différents *tiroirs* de cette armoire sont les suivants :

```
> names(RG)
[1] "R" "G" "Rb" "Gb" "weights" "targets" "genes" "source" "printer"
```

Plus précisément,

- RG$R est un tableau gènes $\times$ puces contenant les valeurs du signal rouge,
- RG$G est un tableau gènes $\times$ puces contenant les valeurs du signal vert,
- RG$Rb est un tableau gènes $\times$ puces contenant les valeurs du bruit de fond rouge,

- RG$Gb est un tableau gènes × puces contenant les valeurs du bruit de fond vert,
- RG$weights est un tableau gènes × puces contenant les valeurs 1 pour les spots conservés lors du filtrage et 0 sinon,
- RG$targets contient les information du fichier `Targets.txt`,
- RG$genes est lui-même une *armoire* dont les différents *tiroirs* contiennent des informations sur chaque gène,
- RG$source rappelle le logiciel d'analyse d'images d'où viennent les données,
- RG$printer décrit la structure d'une puce (taille d'un bloc d'aiguilles, ...).

2.3.1 Taux de filtrage

On peut utiliser `RG$weights` pour mesurer l'impact de la procédure de filtrage. Par exemple, la commande suivante calcule, pour chaque puce, le pourcentage de gènes non-filtrés :

```
> NonFiltres = apply(RG$weights,MARGIN=2,FUN=mean)
# NonFiltres reçoit les taux de spots non-filtrés pour chaque puce
> round(NonFiltres[1:10],2)
# Affichage des 10 premiers éléments de NonFiltres arrondis à 2 décimales
 refCy3-1089Cy5.gpr  refCy3-3789Cy5.gpr
               0.36                0.85
 refCy3-3743Cy5.gpr   refCy3-499Cy5.gpr
               0.80                0.82
refCy3-516BCy5.gpr  refCy3-3793Cy5.gpr
               0.88                0.89
 refCy3-1072Cy5.gpr  refCy3-1026Cy5.gpr
               0.57                0.61
 refCy3-1042Cy5.gpr  refCy3-641BCy5.gpr
               0.82                0.87
```

2.3.2 Nom et identifiant d'un gène

Les noms des différents *tiroirs* de l'*armoire* nommée `RG$genes` sont accessibles par la commande suivante :

```
> names(RG$genes)
[1] "Block" "Row" "Column" "ID" "Name"
```

Pour obtenir les identifiants des 10 premiers gènes, on peut utiliser le *tiroir* nommé `RG$genes$ID` :

```
> RG$genes$ID[1:10]
[1] "RIGG14714" "RIGG14714" "RIGG00105" "RIGG15897" "RIGG04616" "RIGG01680"
[7] "RIGG12912" "RIGG04987" "RIGG12912" "RIGG04987" "RIGG03113" "RIGG06115"
```

Il est possible d'en savoir plus sur chacun de ces gènes à partir de leur nom :

```
> RG$genes$Name[1:10]
[1] "GAPDH cntrl"
[2] "GAPDH cntrl"
[3] "Gallus gallus mRNA for hypothetical protein, clone 6a18"
[4] "ENSGALT00000017516.1"
[5] "Weakly similar to Q9NX88 (Q9NX88) Hypothetical protein FLJ20374"
[6] "Genome Hit Contig103.14"
[7] "ENSGALT00000008978.1"
[8] "Genome Hit Contig1.750"
[9] "Contig Hit 037550.1"
[10] "Genome Hit Contig228.14"
```

2.3.3 Statut d'un spot

Dans l'optique d'une meilleure lisibilité des résultats de l'analyse, il peut être intéressant de tenir compte d'une typologie des spots à partir de leur statut : `gène`, `spike-in`, `blanc`, `contrôle`, ... On utilise pour cela un fichier texte (les séparateurs de colonnes sont des tabulations) appelé `SpotTypes.txt`. Pour les données dont on dispose, ce fichier prend la forme du tableau 2.

SpotType	ID	Name	Color
gene	*	*	black
blancBlanK	blank*	blank*	orange
blancEmpty	Empty*	Empty*	pink
blancOperon	opHsV04NC*	operon QC control*	red
blancBuffer	Spotting	buffer Spotting buffer	brown
blancVide			yellow

TAB. 2: *Exemple de fichier* ***`SpotType.txt`***

Chaque ligne du fichier est associé à un type de spot. Le fichier décrit comment classer un spot selon cette typologie. Par exemple,

- les spots dont l'identifiant contient la chaîne de caractère `blank` dans l'identifiant `ID`

et la chaîne de caractères `blank` dans le nom `Name` seront classés `blancBlanK`,

- les spots `gene` ne se définissent pas par une chaîne de caractères particulière ni dans leur identifiant, ni dans leur nom : `gene` est donc un type de spots par défaut.

La dernière colonne du fichier, nommée `Color`, permet d'associer une couleur à chaque type de spot, afin de mieux les distinguer dans les analyses à venir.

Les commandes suivantes lisent le fichier `SpotTypes.txt` et associent à chaque spot son statut dans `RG$genes` :

```
> spottypes = readSpotTypes("SpotTypes.txt")
# spottypes reçoit le contenu du fichier SpotTypes.txt
> RG$genes$Status = controlStatus(spottypes,RG)
# RG$genes$Status est un nouveau tiroir de RG$genes contenant
# les statuts de chaque spot
Matching patterns for: ID Name
Found 22176 gene
Found 218 blancBlanK
Found 272 blancEmpty
Found 218 blancOperon
Found 48 blancBuffer
Found 768 blancVide
Setting attributes: values Color
```

En outre, la commande `controlStatus` affiche les nombres de spots de chaque type.

3 Normalisation des données

La normalisation des données est une étape importante dans la création du tableau de données sur lequel reposera l'analyse statistique. L'objectif de cette normalisation est d'identifier des sources parasites de variations des expressions. Parmi ces sources, on cible plus précisément :

- l'hétérogénéité du bruit de fond.

  Si le bruit de fond présente des variations très différentes d'une puce à une autre, ou très structurées spatialement sur une puce, alors on peut être amené à corriger le signal par soustraction du bruit de fond.

- l'hétérogénéité du signal.

  De la même manière, le principe d'invariance d'une très grande majorité des expressions géniques d'une puce à une autre doit se traduire par une répartition comparable des

valeurs des signaux entre les différentes puces. Si des différences marquées existent, il est judicieux de ramener les signaux moyens de chaque puce à la même valeur. D'autre part, il arrive que le fluorochrome soit lui-même source de variations du signal. Cet effet du fluorochrome dépend même parfois de la valeur d'expression et de la structure de la puce (partitionnement en blocs d'aiguilles).

3.1 Hétérogénéité du bruit de fond

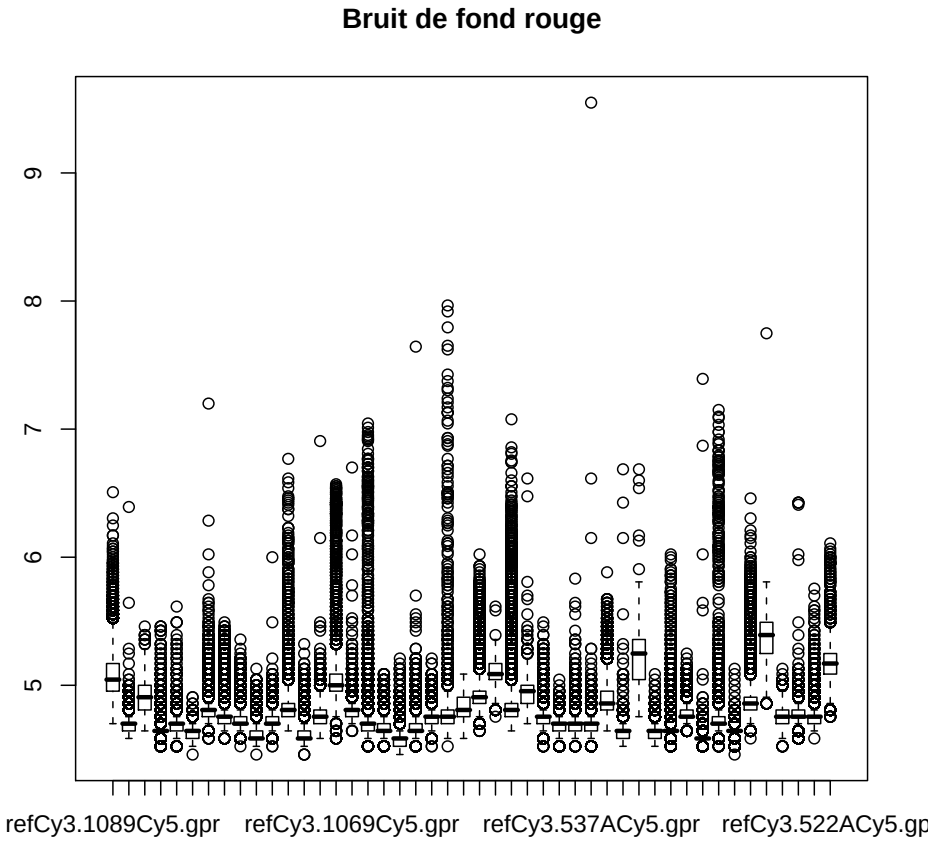
De manière générale, il est important de s'assurer que les variations du bruit de fond des images analysées par `Genepix` ne sont pas structurées :

- d'une part, par une comparaison des variations du bruit entre les différentes puces.

  La commande suivante construit une boîte de dispersion du logarithme du bruit de fond rouge :

```
> boxplot(data.frame(log2(RG$Rb)),main="Bruit de fond rouge")
# Boîtes de dispersion des colonnes de log2(RG$Rb).
# L'argument main ajoute un titre au graphique.
```

Le graphique produit est le suivant :

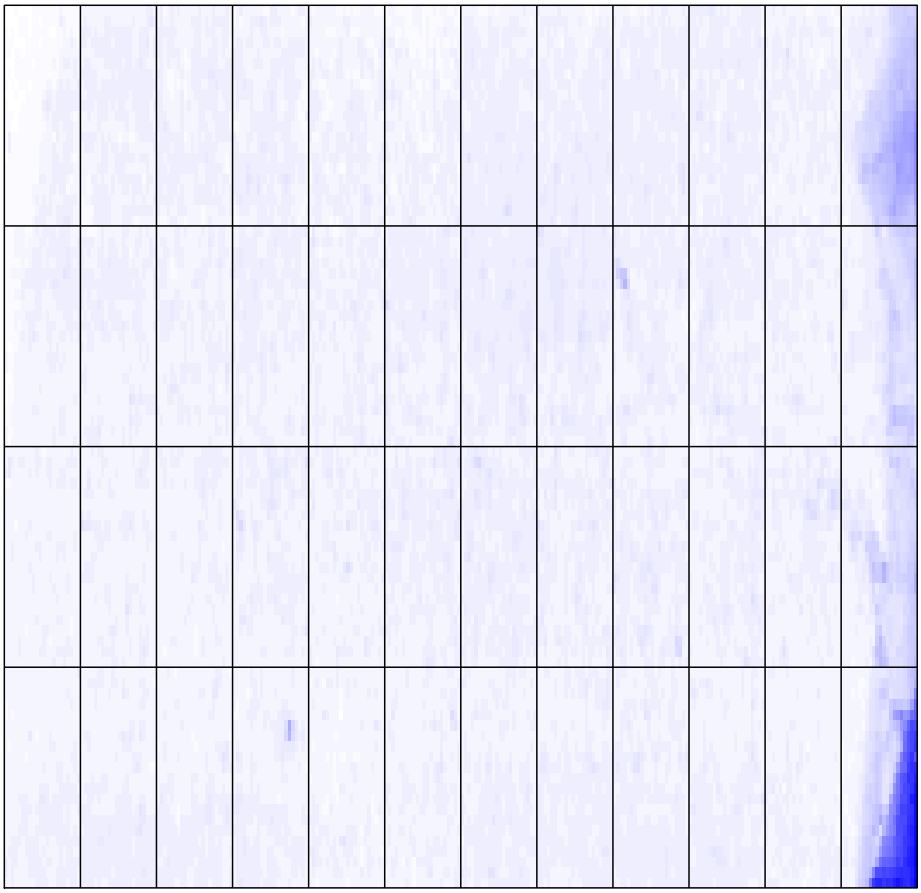


- d'autre part, par un examen de la répartition spatiale du bruit de fond de chaque puce.

  La commande suivante crée une image associant à chaque spot de la 17ème puce un pixel dont le niveau de couleur est proportionnel à la valeur du logarithme du bruit de fond :

  ```
  > imageplot(log2(RG$Rb[,17]),layout=RG$printer)    # Image de log2(RG$Rb)
  # pour la 17ème puce. L'argument layout permet de tenir compte
  # des blocs d'aiguilles dans la puce.
  ```

On obtient le graphique suivant, dont le quadrillage matérialise les blocs d'aiguille :



z−range 4.5 to 7 (saturation 4.5, 7)

La correction du signal par soustraction du bruit de fond peut être obtenue par la fonction `backgroundCorrect`. Dans la suite, on choisit de ne pas appliquer cette correction.

3.2 Hétérogénéité du signal

L'hétérogénéité des signaux mesurés sur une puce est souvent attribuée à des variations uniquement liées aux fluorochromes. Cette différence entre les signaux est décelable sur le nuage dont les points, représentant des spots, ont pour abscisses les moyennes entre les logarithmes des signaux verts et rouges (dites valeurs `A`) et pour ordonnées les différences

entre logarithmes des signaux verts et rouges (dites valeurs `M`). Ce graphique, dit `graphe MA` (voir figure 1), est obtenu par la fonction `plotMA` :

```
> plotMA(RG,status=RG$genes$Status)
# L'argument status permet d'identifier les différents types de gènes.
```

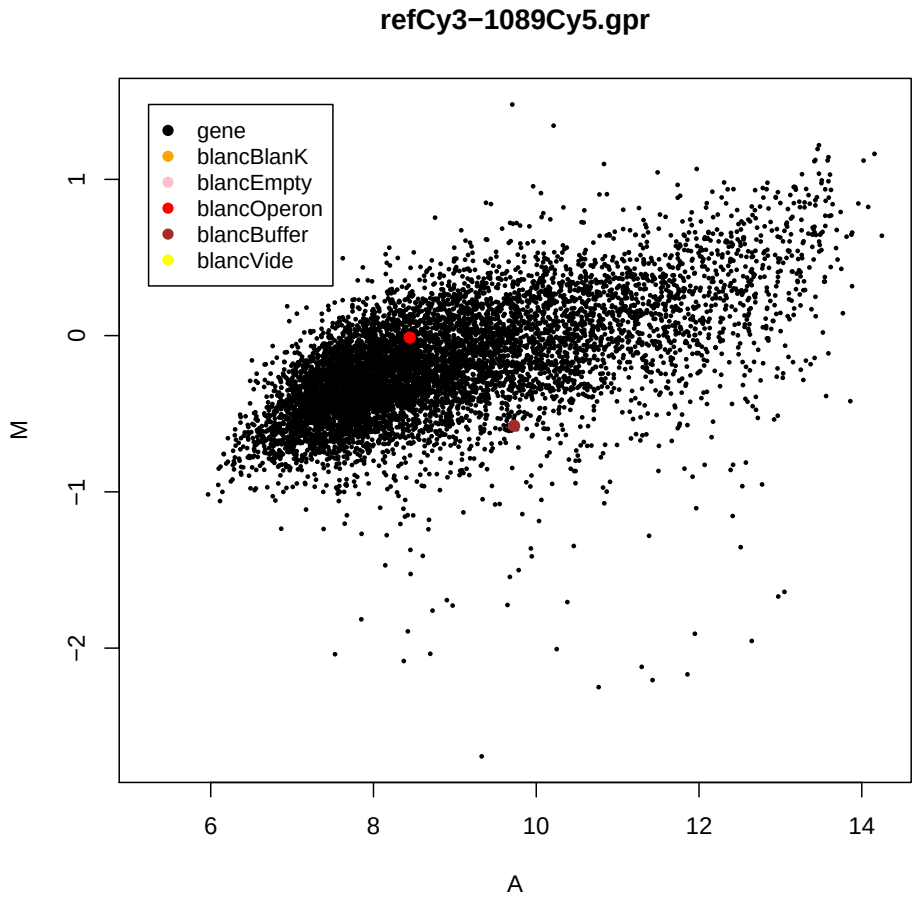


FIG. 1: *Graphe MA des signaux bruts (1ère puce)*

La fonction `normalizeWithinArrays` corrige cette hétérogénéité en ajustant un modèle des variations des valeurs `M` en fonction des valeurs `A` par une méthode de régression locale

(dite méthode `loess`) :

```
> MA = normalizeWithinArrays(RG,method="loess",bc.method="none")
# MA contient désormais des données corrigées.
```

Le résultat de la commande précédente, l'objet `MA`, ressemble en tous points à l'objet `RG` : seuls les *tiroirs* nommés `R` et `G` et contenant les signaux rouges et verts sont remplacés par d'autres, nommées `M` et `A` et contenant les valeurs du même nom.

L'effet de la normalisation intra-puces est visible sur le graphique de la figure 2, obtenu par la commande suivante :

```
> plotMA(MA,status=RG$genes$Status)
```

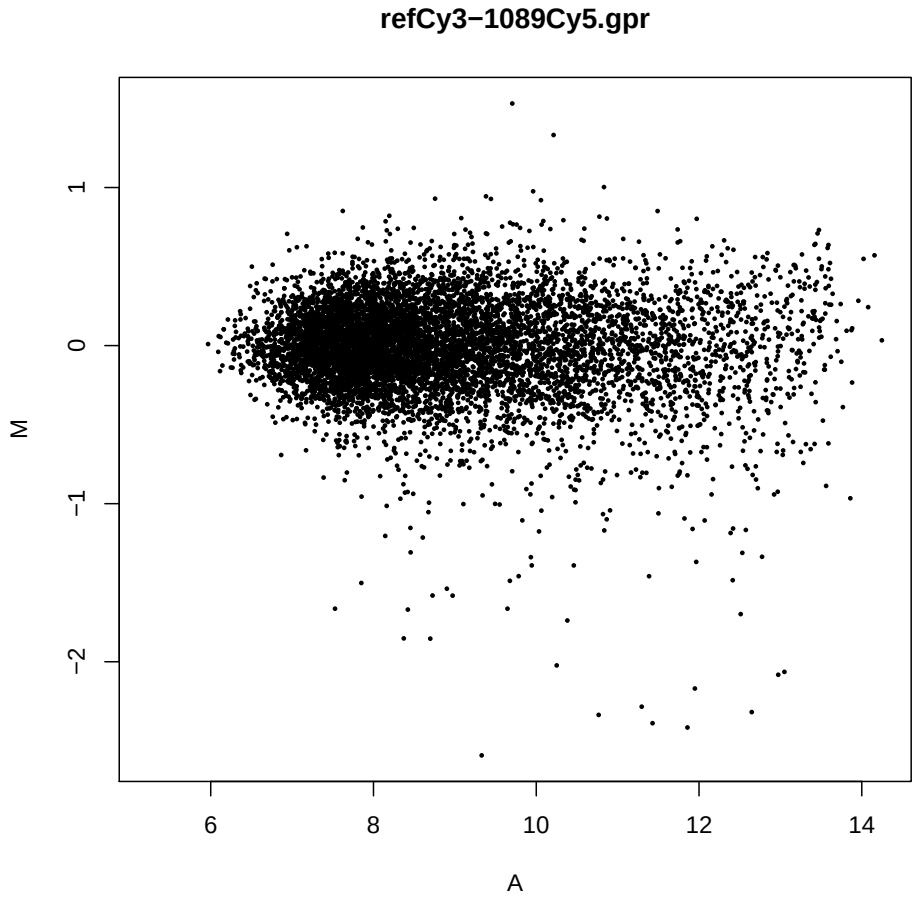


FIG. 2: *Graphe MA après normalisation intra-puce (1ère puce)*

Le graphique 2 montre bien qu'un autre effet de la normalisation décrite ci-dessus est le centrage par puces des valeurs `M`. Il est possible de comparer les répartitions des valeurs dans les 46 puces à l'aide de boîtes de dispersion. Dans un premier temps, les commandes suivantes visent à construire une série de couleurs, nommée `cols`, associant une couleur à chaque modalité du facteur étudié.

```
> condition = factor(targets$Factor)
# condition est désormais un facteur dont les valeurs sont celles
# de la chaîne de caractères targets$Factor.
> cols = condition
# cols est dans un premier temps identique à targets$Factor
> levels(cols) = palette(rainbow(8))
# levels associe finalement à chaque modalité de cols une couleur choisie
# dans la collection rainbow
```

Le graphique souhaité (voir figure 3) est maintenant produit par les commandes suivantes :

```
> boxplot(data.frame(MA$M,main="log-ratios",col=as.character(cols),ylim=c(-5,5))
# l'argument col permet de définir une couleur pour chaque boîte de dispersion
# ylim permet de préciser les limites inférieures et supérieures de l'axe vertical
# sous la forme c(valeur minimale , valeur maximale).
> legend(x="topleft",fill=levels(cols),legend=levels(condition),bg="white")
# la fonction legend ajoute une légende au graphique.
# l'argument x localise la légende sur le graphique
# fill trace des boîtes de couleur dans la légende
# legend donne le texte associé à chaque couleur
# bg donne la couleur de fond dans le cadre de la légende
```

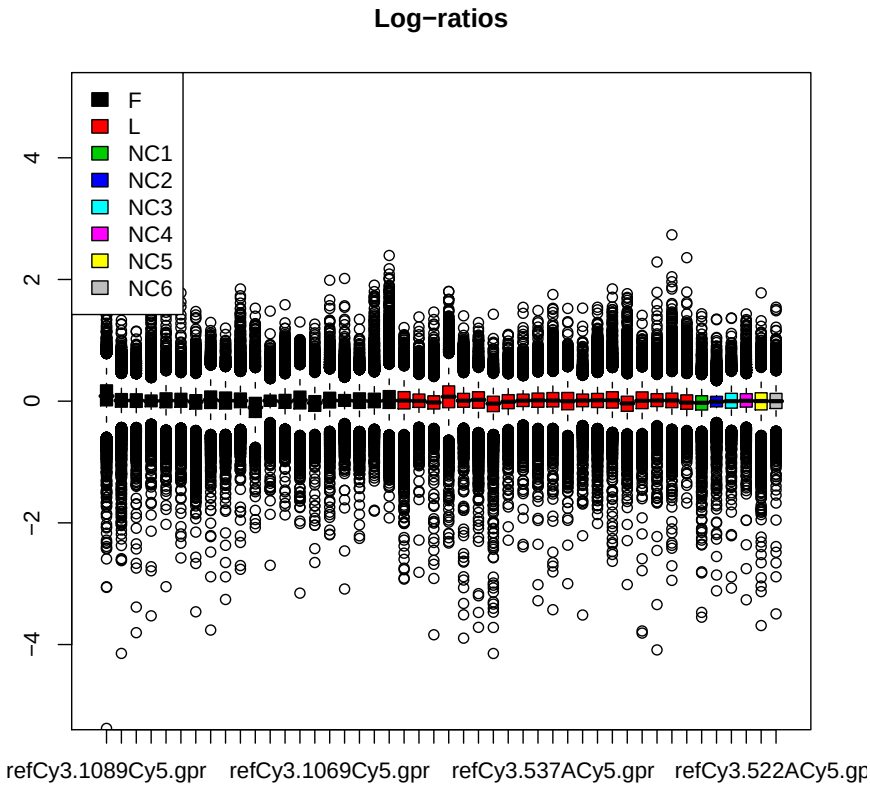


FIG. 3: *Boîtes de dispersion des valeurs* `M` *par puce.*

4 Création du tableau puces × gènes

La normalisation des données a conduit à la création d'une matrice gènes × puces de valeurs M. Le tableau de données qui servira finalement de support à l'analyse est obtenu à partir d'une sélection des colonnes de cette matrice dont le statut est *gene*.

```
> Gene = (1:22176)[MA$genes$Status=="gene"]
# le vecteur Gene contient les indices des colonnes ayant le statut gene.
```

Les commandes suivantes visent à exporter dans un fichier au format texte le tableau transposé puces × gènes.

```
> poulets.data = t(MA$M[Gene,])
# poulets.data contient maintenant le tableau puces × gènes
# Seuls les spots dont le statut est gene ont été conservés
> colnames(poulets.data) = MA$genes$ID[Gene]
# La fonction colnames affecte des noms aux colonnes d'un tableau
# Les noms des colonnes sont les identifiants des spots.
> GM = (1:46)[(targets$Factor=="F")|(targets$Factor=="L")]
# le vecteur GM contient les indices des lignes associées à des puces F ou L.
> poulets.data = data.frame(poulets.data[GM,],Genotype=targets$Factor[GM])
# la fonction data.frame est utilisée ici pour construire un tableau de données
# contenant les données d'expression et une dernière colonne nommée Genotype
# donnant le groupe d'appartenance de chaque puce.
> write.table(poulets.data,file="pouletsdata.txt",sep="\t")
# write.table exporte le tableau de données dans un fichier texte
# sep définit le séparateur de colonnes dans le fichier (ici la tabulation).
```

De la même manière, il est utile de créer un fichier contenant les annotations des gènes du tableau :

```
> write.table(data.frame(ID=MA$genes$ID[Gene],Name=MA$genes$Name[Gene]),
+ file="annotations.txt",sep=";")
```