Toulouse School of Economics
Université Toulouse 1 Capitole

Unité Mathématiques et Informatique Appliquées Toulouse
INRA, Castanet-Tolosan

Normalisation et analyse de données RNASeq

Sayma Besbes

Stage Master 1 Economie et Statistique - TSE
Juin-Août 2014

Tutrice de stage : Nathalie Villa-Vialaneix
Tutrice pédagogique : Christine Thomas-Agnan

Remerciements

En préambule à ce rapport, je souhaite adresser mes remerciements les plus sincères aux personnes ayant contribué au bon déroulement de mon stage.

Je remercie en particulier ma tutrice, Mme Nathalie Villa-Vialaneix, qui m'a donné la chance d'intégrer l'unité MIA-T et qui a su se montrer pédagogue et très disponible malgré ses nombreux déplacements. Je remercie également Mesdames Christine Gaspin, Hélène Chiapello et Annick Moisan.

Je tiens à remercier vivement Mme Céline Noirot pour l'aide précieuse qu'elle m'a apportée, ainsi que Mme Fabienne Ayrignac et Monsieur Alain Perault, membres de l'équipe administrative, qui ont contribué à rendre agréable cette expérience.

Enfin je remercie Monsieur Regis Sabbadin, directeur de l'unité MIA-T, pour m'avoir accueillie au sein de son unité.

Sommaire

Première partie .

Description de l'entreprise

1. L'INRA

L'Institut National de la Recherche Agronomique (INRA) est un organisme de recherche placé sous le statut d'établissement public à caractère scientifique et technologique, et sous la double tutelle du ministère chargé de la Recherche et du ministère chargé de l'Agriculture. Il s'agit du premier institut de recherche agronomique en Europe et deuxième dans le monde en nombre de publications en sciences agricoles et en sciences de la plante et de l'animal.

Dans un contexte climatique, démographique et énergétique complexe, la recherche agronomique doit répondre à des problématiques telles que la sécurité alimentaire mondiale en 2050, la limitation du gaz à effet de serre d'origine agricole ou encore l'adaptation de l'agriculture et des forêts au changement climatique. Ces problématiques impliquent, entre autres, de connaître les comportements des individus à l'échelle des territoires ou des marchés, d'étudier les liens entre la santé des plantes, des animaux et des hommes, de rechercher de nouvelles voies pour la production d'énergie et de matériaux issus de l'agriculture et d'en limiter en général l'impact environnemental.

Pour cela, l'INRA produit des connaissances scientifiques et accompagne l'innovation économique et sociale dans les domaines de l'alimentation, de l'agriculture et de l'environnement. L'INRA mène une politique de partenariat active avec :

- les acteurs socio-économiques (entreprises, organisations collectives agricoles) ;
- les collectivités territoriales ;
- les pouvoirs publics qui sollicitent l'expertise de ses chercheurs aux plans national, européen et international.

L'INRA est constitué de 13 grands départements scientifiques dont le département Mathématiques et Informatique Appliquées (MIA).
Chaque département est constitué d'unités de recherche réparties sur 19 centres régionaux. Avec plus de 850 chercheurs, ingénieurs et techniciens, le centre de Toulouse Midi-Pyrénées représente environ 10% des publications et près de 12% des brevets de l'INRA.

Les équipes du centre INRA Toulouse Midi-Pyrénées privilégient des activités de recherche et d'innovation en réponse à trois grands enjeux :

- des systèmes de production agricoles (végétaux, animaux) et forestiers plus durables et adaptés au changement climatique,
- une alimentation attentive aux questions de santé,
- de nouvelles filières de transformation des agro-ressources en faveur d'une valorisation du carbone renouvelable.

2. L'unité MIA-T

Le laboratoire MIA-T est l'une des 15 unités de recherche du centre INRA Toulouse et est rattachée au département Mathématiques et Informatique Appliquées (MIA).

Afin de favoriser et d'accélérer les recherches exploratoires in silico [1] des objets allant du gène à l'écosystème, l'informatique, la statistique et, plus généralement, les mathématiques sont indispensables. En effet ces disciplines sont nécessaires pour organiser les données sur ces objets et en permettre l'accès, concevoir des outils formels et calculatoires d'analyse de données, de prédiction, de simulation et d'optimisation. La mission de l'unité est de contribuer à apporter une réponse à ces besoins et ses recherches s'accompagnent d'une activité de production de logiciels pour leur valorisation et d'une activité de formation pour leur diffusion.

Les travaux conduits au sein de l'unité MIA-T s'inscrivent dans deux équipes de recherche :

- l'équipe MAD (Modélisation des Agro-écosystèmes et Décision) qui travaille sur la conception de modes de gestion durable des agro-écosystèmes et sur la mobilisation et le développement de méthodes mathématiques et informatiques.
- l'équipe SaAB (Statistiques et Algorithmique pour la Biologie) développe des méthodes mathématiques et informatiques originales pour la biologie qu'elle met à disposition (logiciels) permettant ainsi de contribuer à la compréhension du vivant. Mon travail au sein de l'unité s'est inscrit dans le cadre de cette équipe.

Ces équipes s'appuient sur l'activité de deux plateformes :

- Bioinformatique du GIS GENOTOUL (Génopole Toulouse Midi-Pyrénées)
- RECORD : plateforme de modélisation et de simulation des agro-écosystèmes.

L'organigramme de l'unité est donné dans la figure 2.1.

1. Recherches effectuées au moyen de calculs complexes informatisés ou de modèles informatiques.

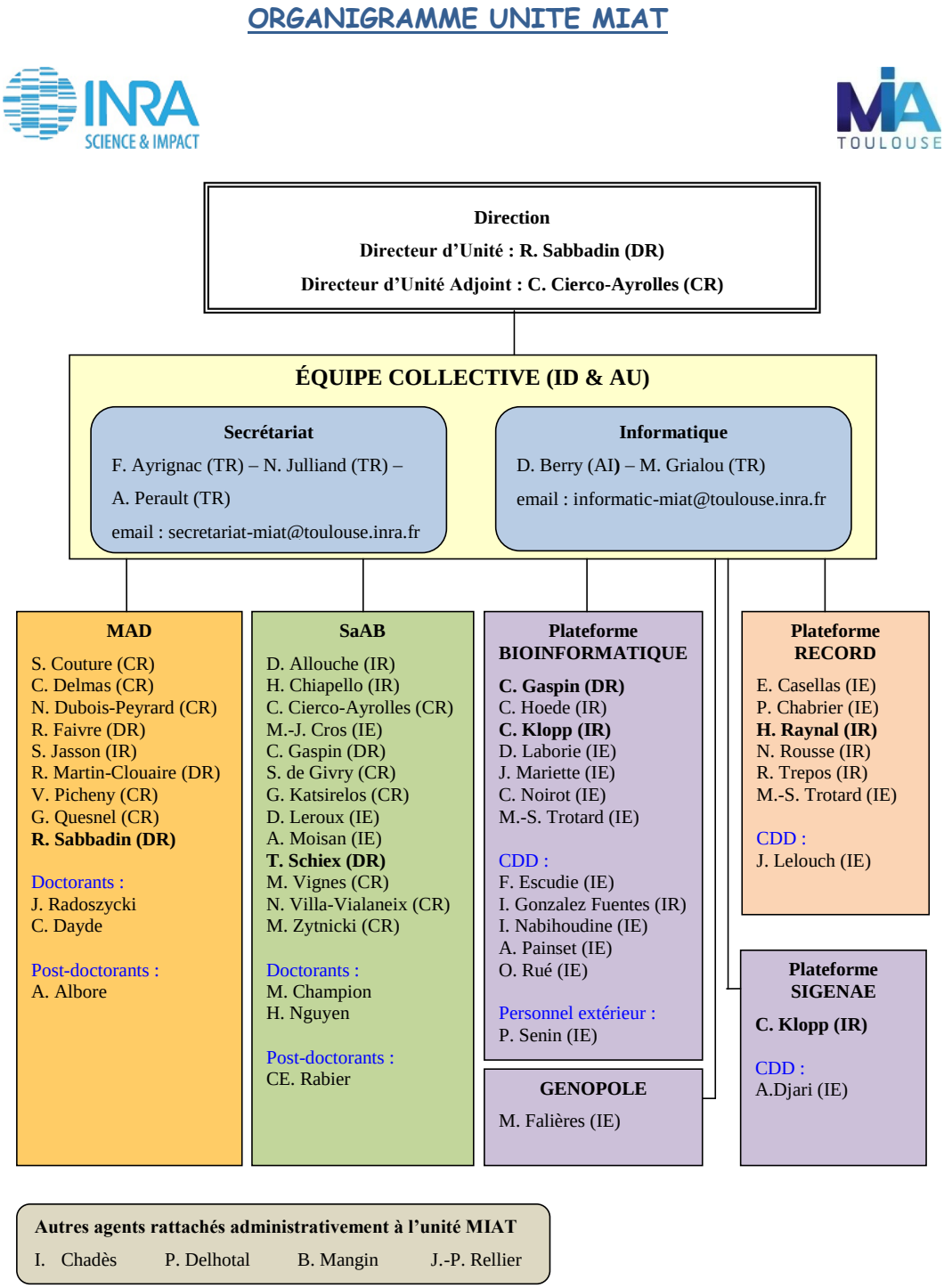


FIGURE 2.1. – Organigramme de l'unité MIA-T

Deuxième partie .

Problématique du stage

3. Notions biologiques de base

Afin de comprendre la problématique de mon stage, nous devons d'abord présenter quelques notions biologiques de base sur l'expression des gènes.

3.1. L'ADN

L'acide désoxyribonucléique (ADN) est une molécule présente dans toutes les cellules vivantes et qui renferme l'ensemble des informations nécessaires au développement et au fonctionnement d'un organisme. Il porte l'information génétique (génotype) et constitue le génome des êtres vivants.

La structure standard de l'ADN est une double-hélice droite, composée de deux brins complémentaires se faisant face. Chaque brin d'ADN est constitué d'un enchaînement de nucléotides. On trouve quatre nucléotides différents dans l'ADN, notés A, G, C et T, du nom des bases azotées qui les composent. Les nucléotides trouvés dans un brin possèdent des nucléotides complémentaires dans l'autre brin avec lesquels ils peuvent interagir (A complémentaire avec T, G avec C). Le génotype est inscrit dans l'ordre dans lequel s'enchaînent les quatre nucléotides. La figure 3.1 donne une représentation de l'ADN.

3.2. L'ARN

L'ARN est une copie d'une région de l'un des brins de l'ADN. Les ARN produits peuvent avoir trois grands types de fonctions : ils peuvent être supports de l'information génétique d'un ou plusieurs gènes codant des protéines (on parle alors d'ARN messagers), ils peuvent adopter une structure secondaire et tertiaire stable et accomplir des fonctions catalytiques et ils peuvent enfin servir de guide ou de matrice pour des fonctions catalytiques accomplies par des facteurs protéiques.

Un ARN non codant (ou ARNnm pour ARN non-messager) est un ARN issu de la transcription de l'ADN qui ne sera pas traduit en protéine. Il en existe différents types dont les ARNs régulateurs de l'expression des gènes.

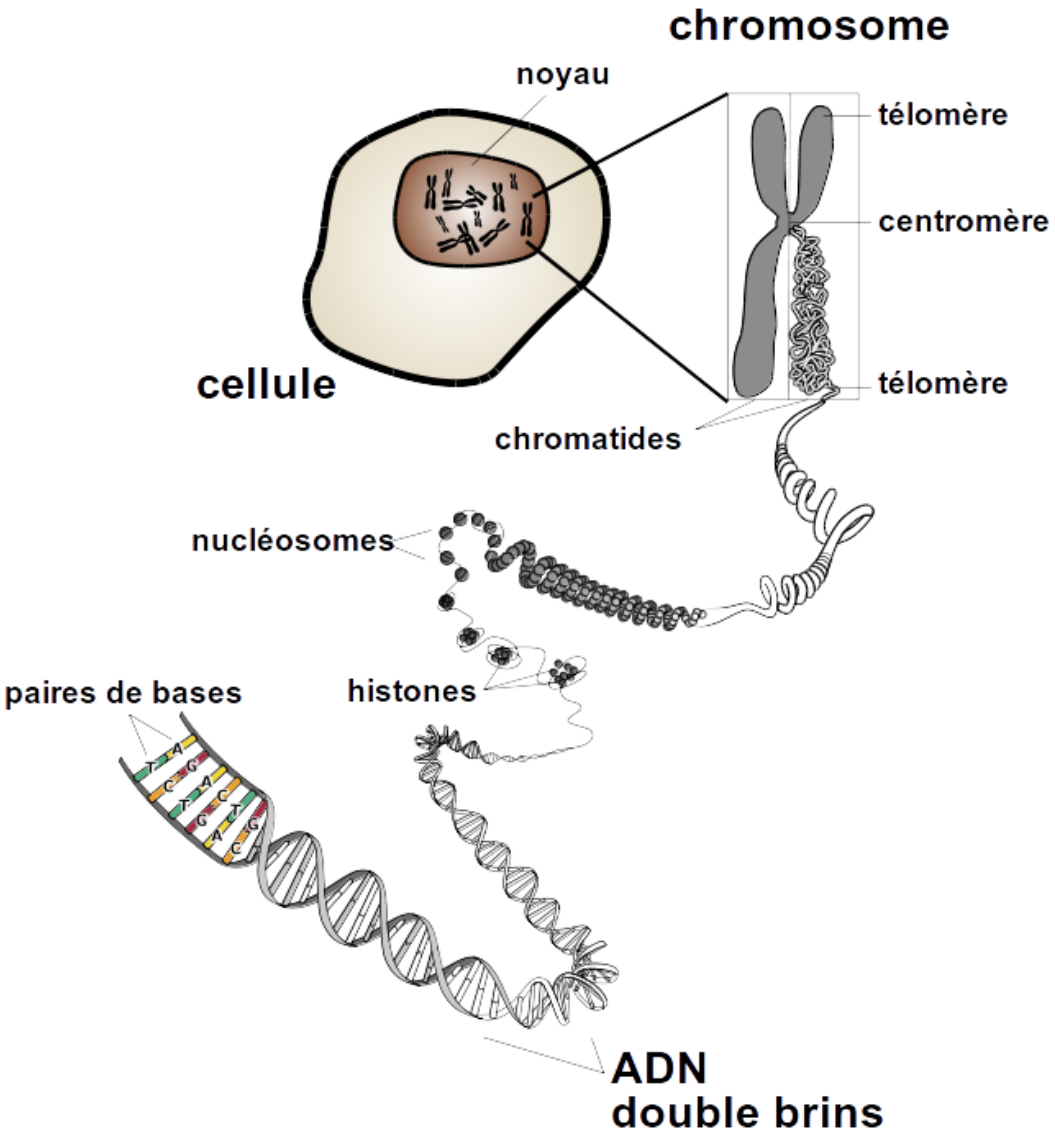


FIGURE 3.1. – De la cellule à l'ADN

3.3. Expression de l'information portée par l'ADN

L'information génétique portée par l'ADN constitue le génotype d'un organisme qui s'exprime pour donner naissance à un phénotype, c'est-à-dire l'ensemble des caractères de cet organisme. Cette expression du génome se fait en plusieurs étapes, comme on peut le voir illustré dans la figure 3.2 :

- La transcription, qui consiste à copier des régions dites codantes de l'ADN en molécules d'ARN.
- La traduction, qui est un transfert d'information depuis l'ARN vers les protéines.
- L'activité des protéines, qui détermine l'activité des cellules, qui vont ensuite déterminer le fonctionnement des organes et de l'organisme.

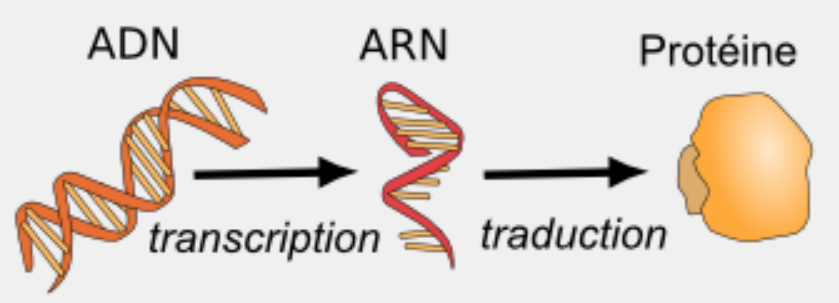


FIGURE 3.2. – Transcription, traduction

Le transcriptome est donc l'ensemble de toutes les molécules d'ARN produites dans une population de cellules. Il peut varier avec des conditions environnementales externes et reflète les gènes qui ont été exprimés de façon active à un temps donné. **Ainsi l'expression d'un gène est mesurée par la mesure quantitative de ses transcrits.**

3.4. Mesure du transcriptome, technologie RNASeq

Pour mesurer le transcriptome, on dispose d'une technologie récente : le RNASeq. Il s'agit d'une technique de séquençage à haut débit qui mesure l'abondance de séquences d'ARN dans différentes cellules pour des milliers de gènes simultanément. Rappelons que le séquençage de l'ARN consiste à déterminer l'ordre d'enchaînement des nucléotides pour un fragment d'ARN donné. Le nombre de séquences lues (appelées aussi « reads ») et alignées sur une région d'intérêt (le gène par exemple) est considéré comme proportionnel au niveau d'expression de cette région d'intérêt. En effet, plus un gène s'exprime, plus grand est le nombre de ses transcrits.

4. Problématique du stage

4.1. Les données

Les données fournies sont des données brutes issues du séquençage haut débit. Il s'agit donc, pour chaque échantillon, d'une liste de séquences de nucléotides lues par le séquenceur. Rappelons la méthode expérimentale : au sein de chaque culture de l'organisme étudié, on récolte des ARN que l'on découpe en fragments qui seront amplifiés et lus par le séquenceur. Cette machine va ainsi nous retourner de plus petits fragments de séquences génomique, les « reads » qu'elle aura lu, sous forme de texte correspondant à l'enchaînement des nucléotides dans l'ADN (A, T, G, C).

Les données se présentent dans un fichier .fastq dont on donne un exemple d'extrait dans la figure 4.1.

```
@SRR794833.2 HWI-ST1155:82:D0NNFACXX:4:1101:1489:2232 length=100
TAATCCCCTGCGTAGACGGTGACAGAACGCTAAGATTATTCTTTTATATTCTGGCTTGTTTCTGCTCACCGTAATTAAGA
CGCTCTCTCCGTTTGGAGGA
+SRR794833.2 HWI-ST1155:82:D0NNFACXX:4:1101:1489:2232 length=100
@@CDFFFFHGHGHJJBGHIEGIIGEGHJJIJIIGCE@GG@@DHG@D@GGGIB4)=CDA=@AAE3=?EECDFDDDD@CC3>
59=@:ADDDCDA@CB@CC?<
@SRR794833.3 HWI-ST1155:82:D0NNFACXX:4:1101:1542:2107 length=100
ACACTGGGATCGCTGAATTAGAGATCGGCGTCCTTTCANNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNTTN
+SRR794833.3 HWI-ST1155:82:D0NNFACXX:4:1101:1542:2107 length=100
@@@DD?DDFAFFFBD?GII>GD@9AE?FFFDD?DFFCD##########################################
####################
```

FIGURE 4.1. – Extrait d'un fichier .fastq

Chaque ligne commençant par un `@` correspond à un read de l'échantillon. Ici par exemple, ce sont les reads nommés SRR794833.2 et SRR794833.3 de l'échantillon SRR794833. On trouve, dans l'ordre et pour chaque read, son nom, sa longueur (« length ») la séquence de nucléotides, et enfin un code (composé de lettres et de symboles) qui correspond à la fiabilité de lecture de chaque nucléotides du read. Cette qualité de séquençage sera vérifiée grâce au programme FastQC qui nous indiquera s'il faut réduire la longueur de séquençage pour améliorer la fiabilité de la lecture.

Ces données brutes ne nous indiquent pas de quelles parties du génome proviennent chacun des reads. Cette information sera obtenue à l'issue du processus d'alignement.

4.2. L'alignement

Le but est de transformer les données brutes en tableau de comptage, pour pouvoir comparer les niveaux d'expression des gènes selon les conditions biologiques. Rappelons que l'on considère que le nombre de reads est proportionnel à l'abondance des ARN correspondants dans la cellule, cette abondance étant elle même considérée comme une mesure du niveau d'expression des gènes.

Prenons par exemple une condition A « organisme non traité » (c'est-à-dire à l'état « sauvage ») et une condition B « organisme traité ». Dans chaque condition on prélève des échantillons (réplicats biologiques) de cellules pour pouvoir en séquencer l'ADN et obtenir ainsi les reads dans un fichier .fastq (un fichier par réplicat) comme décrit précédemment. Une fois que l'on aura vérifié la qualité du séquençage, on pourra aligner ces reads sur le génome de référence de l'organisme étudié afin de pouvoir les faire correspondre à des régions d'intérêt (par exemple le gène). Autrement dit, on va rechercher la position d'origine de chacun des reads sur le génome de référence. On comptabilise ainsi le nombre de reads pour chaque régions d'intérêt.

À l'issue de l'alignement, on obtient un tableau de comptage dont voici un exemple d'extrait dans la figure 4.2 :

	treated1fb	treated2fb	treated3fb	untreated1fb	untreated2fb	untreated3fb	untreated4fb
FBgn0000003	0	0	1	0	0	0	0
FBgn0000008	78	46	43	47	89	53	27
FBgn0000014	2	0	0	0	0	1	0
FBgn0000015	1	0	1	0	1	1	2
FBgn0000017	3187	1672	1859	2445	4615	2063	1711
FBgn0000018	369	150	176	288	383	135	174

FIGURE 4.2. – Extrait d'un tableau de comptage

On trouve en colonnes les échantillons (trois réplicats pour la condition « treated », quatre pour « untreated »), en ligne les régions d'intérêt du génome, ici des gènes. Ainsi, l'élément (i, j) du tableau correspond au nombre de reads issus de l'échantillon j et alignés sur le gène i.

4.3. La normalisation

Avant de pouvoir analyser notre tableau de comptage, il faut normaliser les données. Autrement dit, avant de pouvoir comparer ces données, il faut les rendre comparables entre elles. En effet, il existe plusieurs biais, notamment le biais de profondeur de séquençage (nombre de reads alignés dans l'échantillon) qui nous empêchent de passer directement à l'analyse statistique pour la recherche de gènes différentiellement exprimés.

Imaginons qu'il y ait deux fois plus de reads dans un échantillon (A) que dans un autre (B). Alors, pour un gène qui s'exprime de la même manière dans les deux échantillons, le nombre de reads dans (A) sera approximativement égal à deux fois le nombre de reads dans (B) et les comptages ne seront pas directement comparables. Une des solutions que l'ont pourrait imaginer serait de diviser chaque comptage par le nombre total de reads dans l'échantillon, mais cette méthode est insuffisante car le nombre total de reads d'un échantillon est fortement influencé par quelques gènes dont le comptage est très élevé (la distribution du nombre de reads par gène est en général très asymétrique).

La méthode qui sera utilisée sera celle développée par Anders et Hubers dans [1] et nommée « Relative Log Expression ». Il s'agit d'utiliser, pour chaque échantillon j, un facteur d'échelle S_j qui sera utilisé en coefficient multiplicateur pour corriger le nombre de comptage dans l'échantillon. Ce facteur d'échelle est obtenu comme suit :

- on calcule la moyenne géométrique M_i des comptages $K_{i,j}$ de chaque gène i au travers de tous les échantillons (notons N le nombre d'échantillons) :

$$M_i = \left(\prod_{j=1}^{N} K_{i,j} \right)^{\frac{1}{N}}$$

  On obtient ainsi un pseudo échantillon de référence j' composé de ces comptages moyens ;

- on calcule pour chaque comptage le ratio $\frac{\text{comptage}_{(i,j)}}{\text{comptage}_{(i,j')}}$: c'est un ratio entre l'échantillon j et le nouvel échantillon de référence j' ;

- on calcule enfin, pour chaque échantillon j, la médiane S_j du ratio d'échantillons. Cette médiane sera notre facteur d'échelle par lequel tout les comptages seront multipliés pour rendre les différents échantillons comparables et corriger le biais de profondeur de séquençage.

Remarque : On utilise la moyenne géométrique qui est plus robuste que la moyenne arithmétique car moins sensible aux valeurs extrêmes. Elle donne donc une meilleure estimation de la tendance centrale des données.

4.4. L'analyse dite « différentielle »

Après avoir normalisé les données, on arrive à l'étape finale : la recherche de gènes différentiellement exprimés. Autrement dit on recherche les gènes qui ont un niveau d'expression significativement différent d'une condition à une autre. On teste donc, pour chaque gène, si son niveau moyen d'expression diffère d'une condition à une autre.

Le test de Fisher n'est adapté qu'en l'absence de réplicats (un échantillon par condition). En effet, dans le cas de réplicats biologiques [1], on peut estimer la dispersion de l'expression d'un gène dans une condition. Or, le test de Fisher est basé sur la seule table de contingence des comptages et ne tient donc pas compte de la dispersion spécifique de l'expression du gène : il est donc sous optimal par rapport à un test classique de comparaison de moyennes.

Il est commun de supposer que la variable « comptage du gène g », notée K_g, suit, dans le cas de réplicats techniques, une distribution binomiale notée $\text{Bin}(n, p_g)$ avec :

- n : le nombre total de reads alignés ;
- p_g : la probabilité qu'un read pris au hasard parmi l'ensemble des reads corresponde au gène g.

Lorsque n est grand et p_g très petit, on peut approximer la loi binomiale par une loi de Poisson de paramètre λ, notée $\text{Pois}(\lambda)$, avec $E(K_g) = \text{Var}(K_g) = \lambda$.

Cependant, cette approximation n'est plus valable dès lors que nous nous trouvons dans le cas de réplicats biologiques. En effet, le fait de générer un nouvel échantillon d'ADN implique une probabilité p_g qui varie d'un échantillon à un autre : la variance des comptages pour le gène g sera plus grande que la moyenne, et on parle alors de « surdispersion » des comptages par rapport à une distribution de Poisson. Pour modéliser cette surdispersion des comptages, on modifie l'hypothèse de modélisation comme suit :

$$\mathcal{L}(K_g|\epsilon) \sim \text{Pois}(\theta) \text{ avec } \theta = \lambda\epsilon$$

et

$$\epsilon \sim \text{Gamma}(\alpha, \beta), \text{ avec } \alpha = \beta = r, \text{ le paramètre de disperson.}$$

ϵ est un paramètre introduit pour modéliser l'hétérogénéité biologique des comptages [4].

Cette modification, avec introduction du nouveau paramètre aléatoire ϵ, nous permet d'écrire :

$$K_g \sim \text{BN}(\alpha, p) \text{ avec } \alpha = r, p = \frac{\lambda}{\lambda + r}$$

où BN désigne la loi binomiale négative. On a alors

$$E(K_g) = \lambda \text{ et } Var(K_g) = \alpha + \phi\lambda^2 = \alpha + \frac{\lambda^2}{r}.$$

En effet, la distribution Binomiale Négative est une alternative à la distribution de Poisson quand il s'agit de modéliser des données pour lesquelles la variance empirique est supérieure à la moyenne empirique. La distribution d'une variable K_g de loi binomiale

1. échantillons biologiques différents, à la différence de réplicats techniques qui sont extraits d'un même échantillon

négative de paramètres n et p est donnée par :

$$\forall\, k \in \mathbb{N}, \qquad P(K_g = k) = \begin{pmatrix} k+n-1 \\ k \end{pmatrix} p^n q^k$$

et elle est fréquemment utilisée pour modéliser le nombre d'échecs avant le premier succès pour une suite de variables indépendantes suivant une loi de Bernouilli.

Les paramètres de ce modèle seront estimés grâce à une estimation de la relation dispersion-moyenne des comptages et les tests de différence d'expression seront réalisés à partir de ce modèle.

Il est nécessaire de noter que nous réalisons des tests indépendants sur plusieurs milliers de gènes simultanément. La p-value classique n'est donc pas adaptée. En effet, choisissons par exemple un taux d'erreur du premier type (déclarer un gène differentiellement exprimé alors qu'il ne l'est pas) de 5% par exemple. Imaginons que nous testions 10000 gènes et que tous soient non différentiellement exprimés. Alors pour un test multiple avec un seuil de p-value à 5%, nous déclarons $10000 * 0.05 = 500$ gènes DE alors qu'ils ne le sont pas : c'est beaucoup trop. La solution est d'utiliser une p-value ajustée qui est adaptée aux tests multiples. Il existe différentes méthodes d'ajustement de la p-value, notamment :

- la procédure de Bonferroni qui consiste à contrôler le Family Wise Error Test (FWER), c'est à dire la probabilité d'avoir au moins un « faux positif » [2]. Pour garantir FWER $\leqslant \alpha$, on réalise chacun des tests à un taux de $\frac{\alpha}{G}$ avec G le nombre de gènes testés.
- la procédure de Benjamini-Hochberg (BH) [2], qui consiste à contrôler le False Discovery Rate (FDR), c'est à dire la proportion de faux positifs dans les gènes déclarés différentiellement exprimés. Soit p_g la p-value associée au gène g, les gènes étant classés par p-value croissante. Les gènes déclarés DE au taux α sont tous ceux indicés par $g \leq g_0$ où g_0 est le plus grand indice $\tilde{g}$ tel que :

$$p_{\tilde{g}} \leqslant \frac{\alpha}{G}\tilde{g}.$$

Remarque : Il est moins conservatif de contrôler une proportion que de contrôler un risque. Ainsi, le nombre de gènes déclarés DE sera plus faible en utilisant la procédure de Bonferroni qu'en utilisant celle de Benjamini-Hochberg, mais le risque d'erreur sera aussi plus faible.

2. un gène non différentiellement exprimé (DE) déclaré DE

Troisième partie .

Travail réalisé

Lors de ce stage, deux jeux de données ont été traités :

- le premier jeu était un jeu de données brutes, public et provenant de la bactérie E.Coli. J'ai transformé ces données brutes en tableau de comptage (partie 5). C'est avec ce tableau que j'ai écrit le premier script R pour l'analyse différentielle (partie 6.2, page 24). Ce script est joint en annexe de ce rapport, page 35 ;
- le deuxième jeu de données, envoyé par des biologistes belges, se présentait déjà sous la forme d'un tableau de comptage. L'organisme étudié était un papillon. Le travail effectué avec ces données est développé dans la partie 6.3, page 28.

5. Première problématique : l'alignement

Le but de cette partie était d'aligner les données brutes de séquençage afin d'obtenir un tableau de comptage, mais aussi d'écrire un script pour l'alignement destiné à être réutilisé avec des données RNASeq.

5.1. Qualité des données

Les données utilisées en premier lieu sont des données brutes de séquençage de la bactérie E.Coli. Il s'agit, pour chacun des six échantillons récupérés, d'un fichier .fastq avec la liste des reads obtenus.

Avant toute chose, nous devons vérifier la qualité des séquences avec le logiciel FastQC. Ce dernier nous retourne à partir de chaque fichier .fastq un ensemble d'informations et notamment des graphiques qui permettent d'évaluer rapidement la qualité de séquençage de l'échantillon en question. Le graphique dont on voit un exemple dans la figure 5.1 évalue, pour un échantillon, la fiabilité de la lecture des nucléotides en fonction de leur position sur le read grâce à la construction de boîtes à moustaches. La qualité est évaluée sur une échelle de 1 à 40 divisée en trois zones (« mauvaise », « moyenne », « bonne »).

En plus des boîtes à moustaches, une courbe représentant la qualité moyenne en fonction de la position sur le read est tracée. Si cette courbe reste globalement dans la zone verte sans atteindre la zone rouge, nous pouvons conclure que la qualité de séquençage de l'échantillon est suffisamment bonne pour passer à l'alignement.

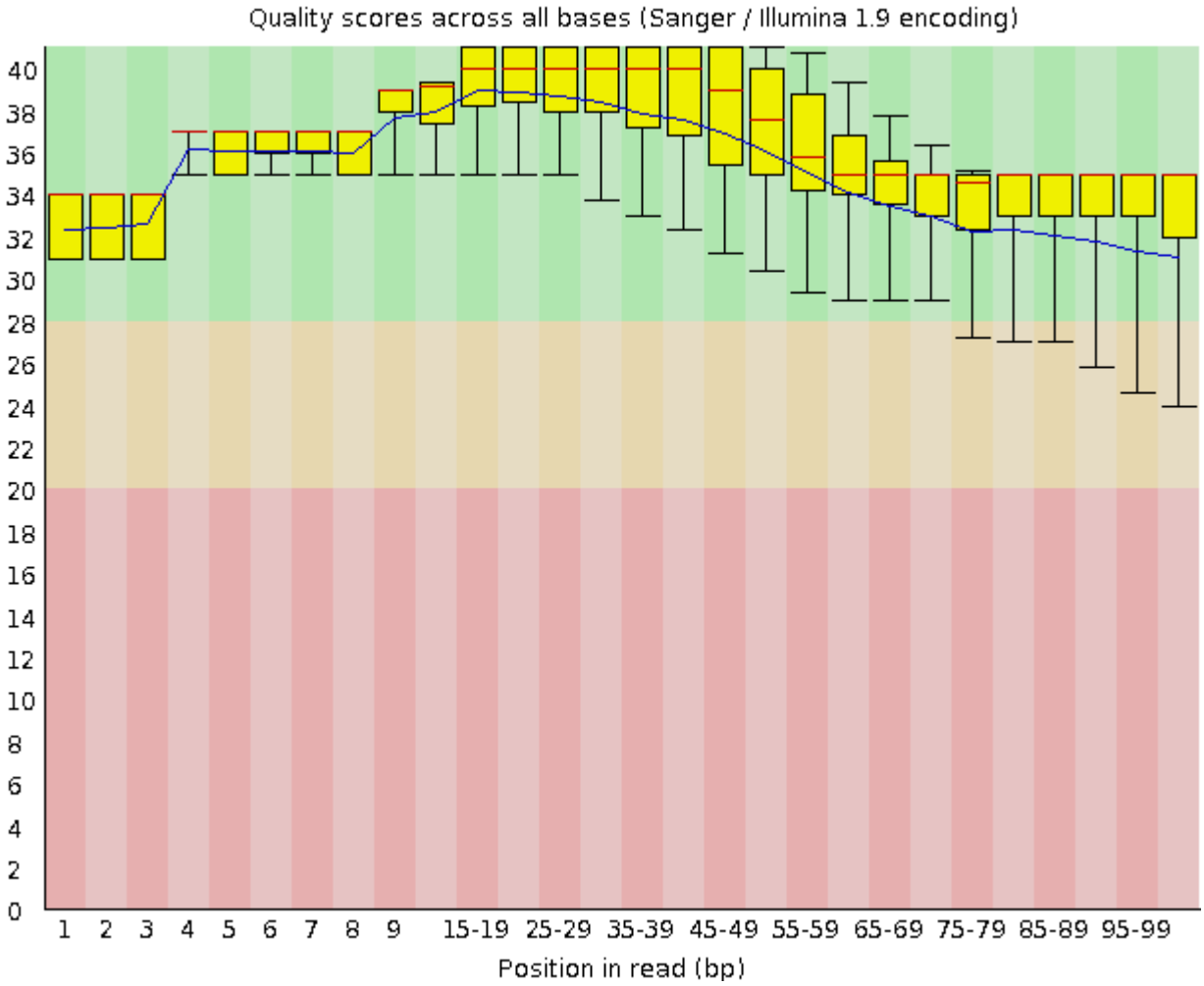


FIGURE 5.1. – FastQC : qualité des données. Fiabilité de la lecture des nucléotides en fonction de leur position sur le read (ici de 1 à 100)

5.2. Indexation

Une fois la qualité des reads vérifiée, nous devons les aligner sur le génome de référence de la bactérie en question. Ce génome est téléchargé depuis une banque de données publiques sous le format « fasta » (.fa). Avant de passer à la partie alignement à proprement parler, nous devons indexer le génome de référence. En effet, le génome de la bactérie ayant une longueur de plusieurs millions de paires de bases, et le nombre de reads à aligner sur ce génome étant de plusieurs millions, il faut rendre le processus d'alignement le plus rapide possible. Pour cela, on « divise » le génome en plusieurs parties afin d'optimiser le processus d'alignement. L'indexation est effectuée par la fonction `bowtie2-build` du logiciel « Bowtie2 ». Cette fonction est exécutée en ligne de commande depuis le terminal (comme toutes les fonctions qui suivent dans la partie alignement) et la syntaxe est la suivante :

```
bowtie2-build      Ecoli_k12_mg1655.fa      Ecoli_k12_mg1655
```

- le premier argument est le nom du fichier du génome de référence en format .fa ;
- le deuxième argument est le nom que l'on veut donner à notre index, c'est à dire le nom de base de tous les fichiers de sortie.

Cette fonction construit ainsi un index constitué de 6 fichiers avec pour suffixes `.1.ebwt`, `.2.ebwt`, `.3.ebwt`, `.4.ebwt`, `.rev.1.ebwt` et `.rev.2.ebwt` (et qui auront tous le même préfixe/ nom de base, ici `Ecoli_k12_mg1655`). C'est à partir de cet index, et non pas du génome entier au format .fa, que sera effectué l'alignement.

5.3. Alignement

Une fois l'index créé, nous pouvons aligner les reads grâce à la fonction `bowtie2` exécutée comme suit pour chacun des échantillons :

```
bowtie2 [options] -x chemin/Ecoli_k12_mg1655 -U chemin/sample1_reads.fastq
      -S aligne1.sam
```

avec :

`-x chemin/base_name` : c'est le chemin vers l'index que l'on désigne grâce à son nom de base ;

`-U chemin/sample1_reads.fastq` : c'est le chemin vers l'échantillon à aligner ;

`-S aligne1.sam` : c'est le nom que l'on veut donner au fichier de sortie (format « .sam »).

Remarque : les options sont spécifiques aux données et à la manière dont le biologiste veut les traiter, elles ne sont donc pas détaillées ici.

Pour chaque échantillon, nous obtiendrons un fichier de sortie qui donnera, pour chacun des reads, sa position sur la référence et la qualité de l'alignement. Par défaut, les fichiers de sortie sont au format « .sam » (texte) que l'on va convertir en format « .bam » (binaire) pour rendre le reste des calculs plus efficace. La conversion se fait grâce à la fonction ci-après (du logiciel `samtools`) qui sera répétée pour chaque fichier « .sam » :

```
samtools view -bS aligne1.sam > aligne1.bam
```

Avant de pouvoir commencer le comptage, on doit ordonner les résultats de l'alignement par leur coordonnées, et ce grâce à la fonction « samtools sort » exécutée pour chaque fichier « .bam » :

```
samtools sort [options] aligne1.bam aligne1_sort
```

Les arguments sont le fichier « .bam » à ordonner et le nom que l'on veut donner au fichier de sortie.

5.4. Comptage

Le comptage est effectué grâce au logiciel « HTSeq » et sa fonction `htseq-count`, mais il faut d'abord télécharger le fichier GFF (ou GTF) qui contient les annotations du génome de l'organisme étudié. Ce fichier, téléchargé depuis ftp://ftp.ensemblgenomes.org, permet de faire correspondre chaque read à une région d'intérêt (ici le gène) grâce à sa position sur le génome obtenue avec l'alignement, et ainsi compter combien de reads ont été alignés sur chaque gène. La commande qui doit être exécutée pour chaque échantillon est la suivante :

```
htseq-count [options] alignment1_sort.bam fichier.gff > count1
```

Le dernier argument (`> count1`) permet de rediriger le résultat dans un fichier de sortie. Les options sont spécifiques aux données et ne sont donc pas détaillées ici.

On obtient donc en sortie et pour chaque échantillon un fichier contenant nos comptages, c'est à dire le nombre de reads alignés sur chaque gène. On doit ensuite réunir nos comptages au sein d'une même table grâce à la fonction suivante :

```
/usr/local/bioinfo/Scripts/bin/merge_cols.py -f count.1,count.2,count.3
        -n ech1,ech2,ech3 -o table_comptage.txt
```

avec :

- `-f` les comptages séparés par « , » ;
- `-n` les noms qu'on attribue aux échantillons ;
- `-o` le nom de le notre fichier de sortie avec extension.

6. Deuxième problématique : normalisation et analyse différentielle des données

6.1. But et outils

Il faut rappeller que le but de ce stage était de produire un script reproductible pour le traitement de données RNASeq pour la recherche de gènes différentiellement exprimés. Le logiciel que j'ai utilisé est le logiciel statistique libre R [3] avec son interface utilisateur graphique « Rstudio » [1].

J'ai en particulier utilisé le package **DESeq2** [1], que l'on trouve, non pas sur le CRAN, mais sur un dépôt de packages R spécifiques à la bioinformatique, Bioconductor [2]. C'est grâce à ce package et sur la base d'un tutoriel intitulé « Analyse statistique des données RNASeq, DESeq2 » [3] que j'ai écrit ce script.

6.2. Premier jeu de données

C'est en traitant le tableau de comptage obtenu précedemment grâce à l'alignement des données brutes de séquençage de la bactérie E.Coli que j'ai écrit le premier script. Ce script, dont une copie est jointe en annexe de ce rapport à la page 35, se devait d'être structuré et commenté afin d'être réutilisé par notamment ma tutrice, Mme Villa-Vialaneix, ou tout autre membre de l'équipe. Le script est donc séparé en 10 « codes », titrés et commentés dont voici la liste et la description :

1. **Installation des packages**. Quatre packages sont chargés ici :
   - **DESeq2** : normalisation et recherche de gènes différentiellement exprimés en se basant sur un modèle de distribution binomiale négative ;
   - **Biobase** : contient des fonctions nécessaires à l'utilisation de nombreux packages utilisés en biologie (comme DESeq2).
   - **gplots** : outils pour la représentation graphique des données.
   - **RColorBrewer** : fournit des palettes de couleurs pour améliorer les graphiques.
2. **Chargement de la table de comptage**.
3. **Construction de l'objet DESeqDataSet (dds)**. Un tel objet contient la table de comptage mais également un tableau décrivant le plan d'expérience (nommé `coldata`). Dans ce plan d'expérience, il y a au moins une variable de type facteur :

1. http://www.rstudio.com
2. http://www.bioconductor.org/
3. Tutoriel réalisé en 2013 dans le cadre de la formation continue INRA par J. Aubert, A. de la Foye et C. Hennequet-Antier.

la condition. Il s'agit de la condition biologique qui peut prendre par exemple deux niveaux : « organisme traité », « organisme non traité ». Il y a également un facteur qui décrit le format de séquençage qui peut prendre deux niveaux : `single-read` quand on séquence l'ARN d'un seul coté, ou `paired-end`,quand on séquence des deux cotés.

4. **Exploration des données**. Il s'agit d'explorer nos échantillons et d'observer les différences qui existent entre eux. On observe ainsi la proportion de comptages nuls par échantillons, la profondeur de séquençage (c'est à dire le nombre de reads) par échantillons, la distribution des comptages des gènes par échantillons, le tout pour pouvoir faire une première comparaison des échantillons entre eux. Pour mieux visualiser la distribution des comptages, on fait un histogramme des valeurs de comptages en utilisant leur log-fréquence (on aura ainsi en ordonnées les fréquences des « log(comptage + 1) » qui se trouvent en abscisses).
5. **Filtrage**. Il s'agit de supprimer les gènes qui n'apportent aucun intérêt à l'analyse statistique : typiquement, les gènes aux comptages toujours nuls quelle que soit la condition. On peut également appliquer un filtrage arbitraire : ici, on décide de supprimer les gènes dont le comptage moyen est inférieur à 5. Ce type de filtrage est laissé à l'appréciation du biologiste.
6. **Normalisation et analyse différentielle**. La fonction `DESeq()` réalise la normalisation et l'analyse différentielle en une seule étape. Dans ce script, on a utilisé les valeurs par défaut des options `test` [4] et `fitType` [5] de la fonction [6]. Le traitement se décompose en deux étapes :
   - la normalisation est effectuée par la méthode décrite dans la section 4.3 à la page 15. Les facteurs d'échelle résultant de la normalisation sont automatiquement inscrits dans la table `coldata` dans une colonne `sizeFactor` ;
   - l'analyse différentielle est basée sur une modélisation de la distribution des gènes par une loi Binomiale Négative. L'estimation des paramètres de ce modèle se fait grâce à une estimation de la relation entre la variance et la moyenne des comptages. C'est à partir de ce modèle que les tests seront réalisés.

   Nous pouvons visualiser les résultats de la normalisation en représentant à nouveau la profondeur de séquençage et la distribution des comptages des données normalisées [7]. Les résultats de l'estimation des paramètres de dispersion, les détails sur les tests statistiques et p-values sont extraits grâce à des fonctions spécifiques au package (se reporter au script).
   Nous pouvons également visualiser l'estimation de la relation dispersion moyenne, comme dans la figure 6.1
7. **Résultats de l'analyse**. Les résultats de l'analyse sont accessibles via la fonction `results(dds)` : pour chaque gène nous avons, entre autres, la statistique de test, la p-value, la p-value ajustée (par la méthode de Benjamini-Hochberg). Nous pouvons donc extraire une liste des gènes déclarés différentiellement exprimés à 1%, 2% ou 5% en les séléctionnant par rapport à leur p-value ajustée.

4. `"Wald"`, par défaut ou `"LRT"` (Likelihood Ratio Test)
5. `"parametric"`, par défaut ou `"local"` ou `"mean"`
6. Nous utiliserons des options différentes dans le deuxième jeu de données.
7. On spécifie que l'on veut les données normalisées grâce à l'option `"normalized=TRUE"` dans la fonction `counts(dds)`.

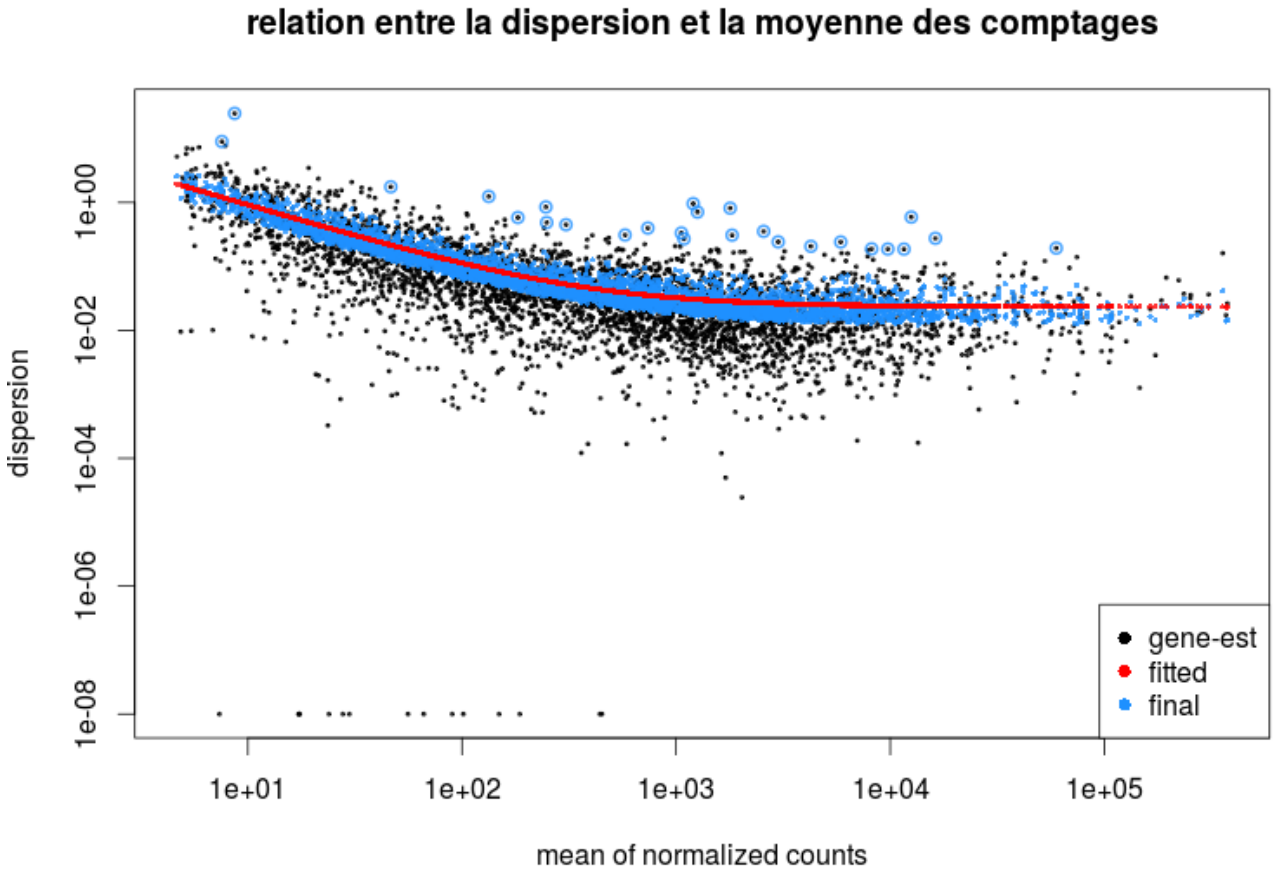


FIGURE 6.1. – Estimation de la relation entre dispersion et moyenne

8. **Comparaison des p-values brutes, ajustées BH et ajustées Bonferroni** Il est intéressant de comparer les résultats selon ces trois types de p-values. En générant un graphique représentant les p-values (ordonnées par ordre croissant) selon la méthode utilisée, on peut constater que la méthode de Bonferroni est plus conservatrice que la méthode de Benjamini-Hochberg, elle même plus conservative que la p-value brute.
9. **Classification Ascendante Hiérarchique (CAH)**. Il s'agit de regrouper les échantillons les plus proches afin de former des classes qui elles mêmes seront regroupées selon leur distance pour former de nouvelles classes, et ainsi de suite jusqu'à obtention d'un dendogramme comme présenté dans la figure 6.2. On regroupe les classes (échantillons à la première étape) selon une distance que l'on définit dans les paramètres de la fonction. Ici, il s'agit d'une distance en termes de corrélation. Ainsi, plus les classes sont « éloignées », moins elles sont corrélées. Le but est de vérifier que les échantillons que l'on suppose a priori être « proches » le sont effectivement.

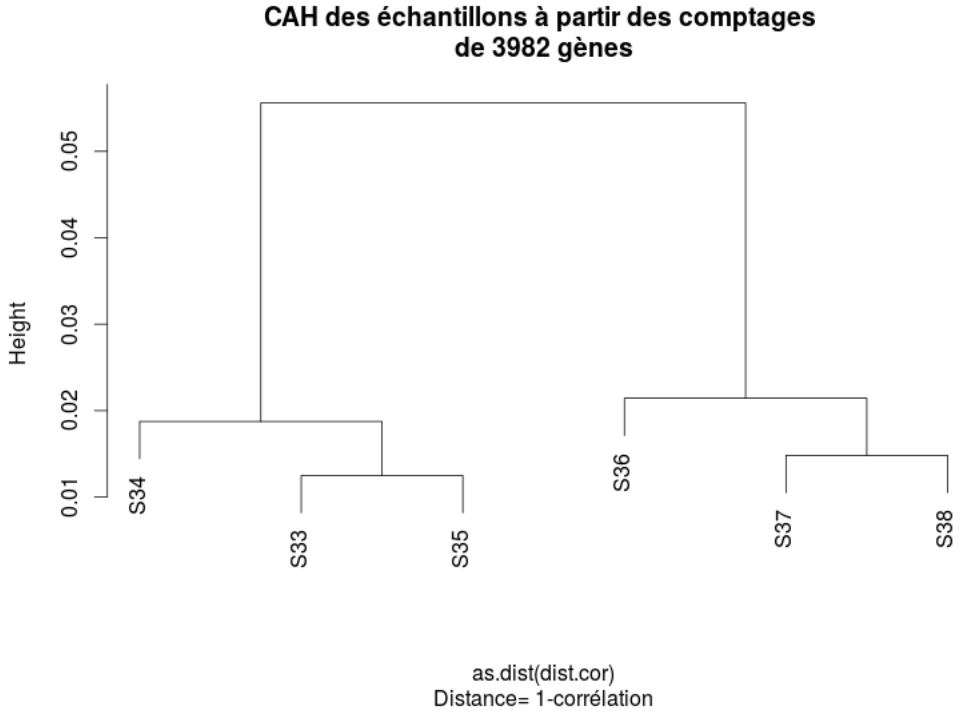


FIGURE 6.2. – Exemple de Classification Hiérarchique Ascendante

On remarque dans cet exemple de dendogramme et en se reportant au plan d'expérience de cet exemple particulier, qu'à la plus haute partition hiérarchique, les échantillons sont regroupés en classes définies par un facteur « condition » à deux niveaux (condition 1 pour les échantillons S33, S34, S35, condition 2 pour les échantillons S36, S37, S38). Lorsqu'il existe plus d'un facteur dans le plan d'expérience, il est intéressant de noter que la structure interne des classes peut dépendre de ces autres facteurs.

10. **Heatmap**. Pour mieux visualiser cette classification, notamment lorqu'il y a un plus grand nombre d'échantillons avec plusieurs facteurs de conditions, il est intéressant de générer une « heatmap » qui permet aussi de visualiser les ressemblances entre les échantillons. Cette méthode exploratoire étant sensible à la présence de variances différentes entre les échantillons, il est nécessaire de stabiliser la variance grâce à la fonction `rlogTransformation()`. On obtient ainsi une « heatmap » dont on voit un exemple dans la figure 6.3. Plus la zone est foncée, plus les échantillons correspondants sont dits « proches ».

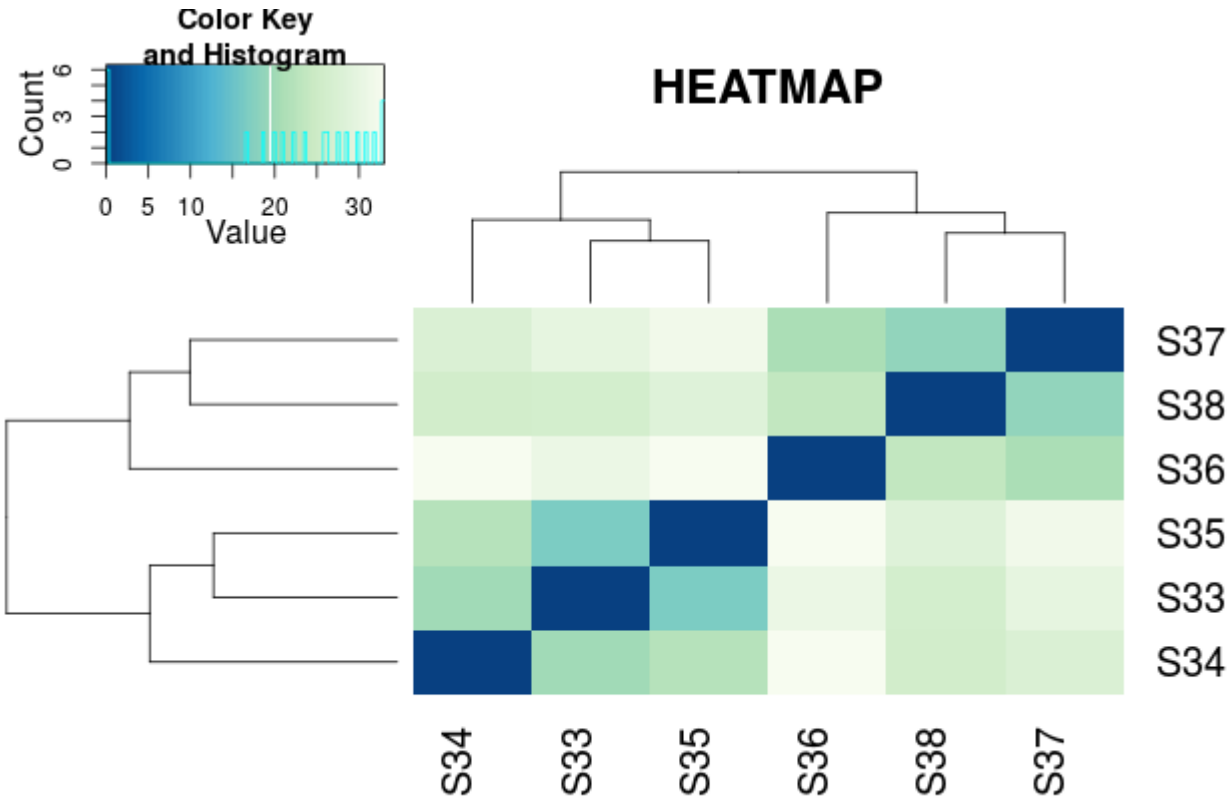


FIGURE 6.3. – Exemple de « heatmap » avec stabilisation de la variance

6.3. Deuxième jeu de données

Après avoir écrit ce premier script, nous avons reçu de la part de biologistes un jeu de données RNASeq. J'ai été chargée de la normalisation et de l'analyse de ce jeu de données afin de les traiter à l'aide du script que je venais d'écrire et en utilisant donc le package **DESeq**. Afin de rendre plus lisibles la méthode et les résultats, j'ai écris ce script sous le format « R Markdown » qui permet de visualiser à la fois le code R et le résultat correspondant de manière dynamique. Ce type de script s'édite depuis RStudio et permet d'entrelacer des commentaires, des scripts R et le résultat des scripts : à la compilation du fichier, le script R est exécuté et les résultats automatiquement inclus pour produire un unique fichier de sortie au format HTML.

Ce document est joint en annexe de ce rapport à la page 39. Il a été écrit sur la base du premier script tout en apportant les modifications nécessaires au traitement d'un jeu

de données différent. Notons que dans ce deuxième jeu de données il ne s'agit pas de gènes mais d'une autre région d'intérêt du génome, des séquences génomiques appelées « contigs ».

6.3.1. Les données

L'organisme étudié est un papillon. Les échantillons et leurs réplicats appartiennent à huit librairies différentes selon qu'ils proviennent d'une chrysalide [8] ou d'un papillon adulte, et selon le tissu dont ils ont été extraits : les ailes, les antennes ou le cerveau de l'insecte. Le biologiste était intéressé par une comparaison de la librairie L8 (antenne, adulte) contre l'ensemble des librairies L1 (ailes, chrysalide), L2 (idem), L6 (cerveau, adulte) et L7 (idem). On a donc retiré de la table de comptages les librairies auxquelles on ne s'interessait pas et on a crée un facteur « condition » à 2 niveaux :

- condition 1 pour L1, L2, L6, L7,
- condition 2 pour L8.

Il s'agit donc de rechercher les contigs qui sont différentiellement exprimés d'une condition à une autre.

6.3.2. Le filtrage

Un filtrage bien particulier des données a été demandé et exécuté :

- filtrage des contigs inexprimés,
- filtrage des contigs qui ne s'expriment que dans la condition 2 (L8),
- filtrage des contigs qui n'ont que des comptages inférieurs à 5.

6.3.3. Options de la fonction DESeq()

Rappelons que la fonction `DESeq()` effectue la normalisation et l'analyse différentielle des données en une seule étape. Mais il existe deux options pour cette fonction :

- l'option « test » : Wald par défaut ou LRT (Likelihood Ratio Test).
- l'option « fitType » : il s'agit du type de regression utilisé pour estimer la relation entre la dispersion et la moyenne des comptages. Cette option peut prendre trois valeurs : « parametric », « local » ou « mean ».

Pour déterminer quelle combinaison d'options était la meilleure, nous avons testé les 6 différentes combinaisons possibles et observé les résultats suivants :

- combien de contigs étaient déclarés différentiellement exprimés, non différentiellement exprimés et surtout combien de valeurs manquantes (`NA`) étaient produites (il s'agit de contigs pour lesquels la p-value n'a pas pu être déterminée par le test) ;
- la courbe de la relation estimée entre variance et moyenne des comptages.

Nous avons décidé de ne pas utiliser l'option `"mean"` car celle-ci donne une mauvaise estimation de la courbe de dispersion comme on peut le voir dans la figure 6.4.
En comparant les résultats des combinaisons restantes, on décide d'opter pour la combinaison `test="LRT", fitType="local"` car c'est celle-ci qui donne le moins de valeurs manquantes pour les p-values.

8. stade de développement intermédiaire entre la larve et l'adulte

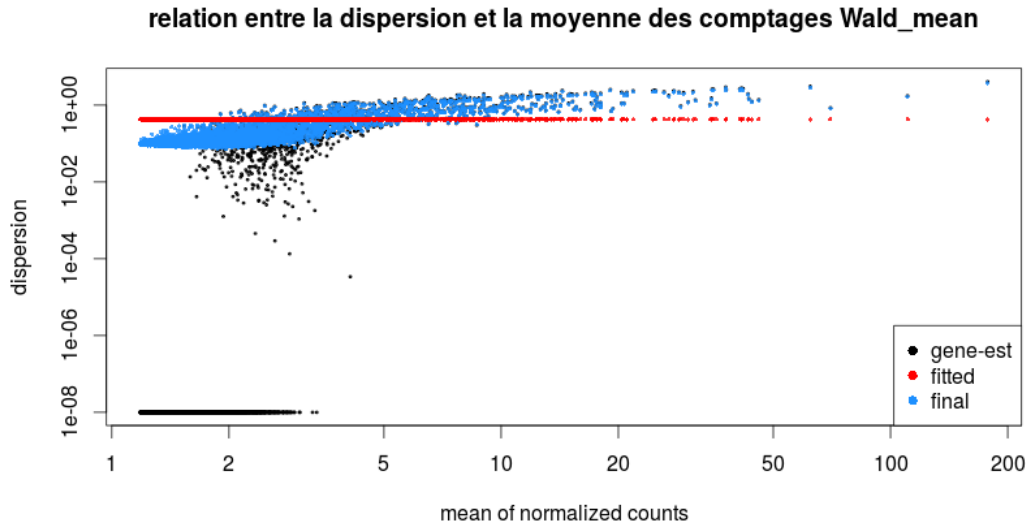


FIGURE 6.4. – Relation estimée dispersion-moyenne avec l'option `fitType="mean"`.

6.3.4. Les résultats

Les résultats ont été intégrés dans une table à part, ordonnés du gène le plus significatif au moins significatif. Nous voulions également séparer les contigs surexprimés des sous-exprimés grâce à leur statistique de test, mais dans ce cas particulier, les contigs étaient tous surexprimés. Sur les 5101 contigs traités après filtrage, nous avons obtenus 984 contigs déclarés différentiellement exprimés à 5% et 694 valeurs manquantes.

Quatrième partie .

Conclusion

Afin de mener à bien ce stage, j'ai dû commencer par m'approprier des notions de base en biologie, discipline à laquelle j'étais complètement étrangère. Mais ce défi m'a permis de comprendre qu'un statisticien doit avoir une grande capacité d'adaptation face au large éventail de disciplines dans lesquelles il peut être ammené à travailler.

La première problématique de ce stage relevant de la bioinformatique, je n'ai pas tout de suite vu l'intérêt de cette partie. Mais je me suis vite rendu compte que cette problématique m'apportais beaucoup : j'ai pu découvrir et appréhender le système Unix/Linux, très utilisé dans le domaine scientifique, et j'ai appris à travailler sur des logiciels en ligne de commande. J'ai ainsi gagné en confiance quant à mes capacités en informatique. La transformation des données brutes en tableau de comptage m'a appris qu'un statisticien doit pouvoir manipuler les données brutes qu'il doit traiter, quelle que soit leur nature.

Enfin, la partie « normalisation et analyse des données » m'a permis de compléter mes connaissances en statistique et de progresser dans l'utilisation du logiciel RStudio. En travaillant avec des biologistes, j'ai pu comprendre qu'un script se devait d'être clair, reproductible et adapté au niveau et aux exigences de l'utilisateur.

Cette expérience professionelle m'a confortée dans mon choix de poursuivre mes études en statistique. Cette discipline débouche sur un grand choix de carrières professionelles, permet de travailler dans une multitude de champs d'applications, avec des personnes aux parcours professionels différents et c'est en cela, je pense, que réside la richesse du métier de statisticien.

Bibliographie

[1] Simon Anders and Wolfgang Huber (2010) Differential expression analysis for sequence count data. *Genome Biology*, **11** :R106.

[2] Yosef Hochberg and Yoav Benjamini (1990) More powerful procedures for multiple significance testing. *Statistics in Medicine*, **9**(7) : 811–818.

[3] R Core Team (2014) *R : A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org

[4] Rainer Winkelmann (2008) *Econometric analysis of count data*. Lecture Notes in Economics and Mathematical Systems, Springer.

Cinquième partie .

Annexes

A. Script R, premier jeu de données

```
#############################################################
### code 1: InstalLibrairies
#############################################################
#source("http://www.bioconductor.org/biocLite.R")
#biocLite(pkgs=c("DESeq","DESeq2","Biobase") )
library(DESeq2)
library(Biobase)
library(gplots)
require(RColorBrewer)
sessionInfo()

#############################################################
### code 2: Chargement de la table de comptage
#############################################################
count <- read.table("table.txt", header=TRUE, row.names=1)
head(count)
## suppression des dernières lignes : count[4141:4145,]
count[4141:4145,]
last <- c(4141:4145)
count <- count[-last,]

#############################################################
### Code 3: Construction objet DESeqDataSet : dds
#############################################################
## création de la table colData qui décrit le plan d'expérience :
## "condition", "type"
condition <- factor(c("condition1","condition1","condition1","condition2",
                      "condition2","condition2"))
type <- factor(rep("single",6))
colData <- data.frame(condition,type,row.names=colnames(count))
## création de l'objet dds
dds <- DESeqDataSetFromMatrix(count, colData, design=~condition)
colData(dds)$condition <- relevel(colData(dds)$condition,ref="condition2")
# objet dds
class(dds)
colData(dds)
design(dds)
# manipulation des données de comptage
dim(counts(dds))
head(counts(dds))
summary(counts(dds))

#############################################################
### code 4: Exploration des données
#############################################################
## combien il y a-t-il de comptages nuls par échantillon ?
apply(counts(dds), 2, FUN = function(x) sum(x ==0))
## quelles sont les profondeurs de séquençage de chaque échantillon ?
colSums(counts(dds))
```

```
barplot(colSums(counts(dds)))
## distribution des gènes suivant leur log-fréquence pour chaque échantillon
par(mfrow = c(2,3))
for (i in 1:6) {
  hist(log(counts(dds)[,i] +1),
       breaks = seq(0,14,1), col="grey", cex.lab=0.8,
       main = colnames(counts(dds))[i],
       xlab = "Valeur du comptage (nbr de reads par gènes) en log(count+1)",
       ylab = "Fréquence du comptage")
}

############################################################
### code 5: Filtrage
############################################################
## filtrage arbitraire: on ne garde que les gènes avec un comptage moyen
## supérieur à 5.
count_mean5 <- count[rowMeans(count)>=5,]
dds <- DESeqDataSetFromMatrix(count_mean5, colData, design=~condition)
dim(counts(dds))
par(mfrow = c(2,3))
for (i in 1:6) {
  hist(log(counts(dds)[,i] +1),
       breaks = seq(0,14,1), col="grey", cex.lab=0.8,
       main = colnames(counts(dds))[i],
       xlab = "Valeur du comptage (nbr de reads par gènes) en log(count+1)",
       ylab = "Fréquence du comptage")
}

############################################################
### code 6: Normalisation et Analyse différentielle
############################################################
## la fonction DESeq() réalise la normalisation et l'analyse diff en une
## seule étape
## la normalisation des données de comptage se fait par la méthode
## "effective library size".
dds <- DESeq(dds)
colData(dds)
# les facteurs d'échelle ont été obtenus grâce au calcul suivant:
# MoyGeomLignes <- apply(counts(dds), 1, function(x) (prod(x))^(1/length(x)))
# apply(counts(dds)[MoyGeomLignes != 0, ] / MoyGeomLignes[MoyGeomLignes != 0], 2,median)
### code 6.1: Visualisation de la normalisation ###
#profondeur de séquençage après normalisation
par(mfrow = c(1,1))
colSums(counts(dds, normalized = TRUE))
barplot(colSums(counts(dds, normalized = TRUE)),
        main="profondeur de séquençage après normalisation")
#données de comptage normalisées
head(counts(dds, normalized = TRUE), n=3)
summary(counts(dds, normalized = TRUE))
# distribution des gènes suivant leur log-fréquence après normalisation
par(mfrow = c(2,3))
for (i in 1:6) {
  hist(log(counts(dds)[,i] +1),
       breaks = seq(0,14,1), col="grey", main = colnames(counts(dds))[i],
       xlab = "Valeur du comptage (nbr de reads par gènes) en log(count+1)",
       ylab = "Fréquence du comptage",cex.lab=0.8)
}
### code 6.2: Dispersion des gènes ###
```

```
# La fonction DESeq() a effectué une estimation de la dispersion des gènes
# Valeurs de dispersion:
disp <- as.data.frame(dispersions(dds))
head(disp,n=3)
print(summary(disp))
# Boxplot des valeurs de dispersion
par(mfrow=c(1,1))
boxplot(sqrt(disp),horizontal=T,las=1,cex=0.5)
title("racine carrée de la dispersion calculée par DESeq2")
### code 6.3: relation entre la dispersion et la moyenne des comptages ###
par(mfrow = c(1,1))
DESeq2::plotDispEsts(dds)
title("relation entre la dispersion et la moyenne des comptages")
### code 6.4: details sur les calculs des valeurs de dispersion, tests
## statistiques et pvalues ###
details<-mcols(dds,use.names=T)
head(details,n=3)
mcols(mcols(dds,use.names=T))
### code 6.5: Graphique MA-plot : relation entre le comptage moyen d'un gène
## et son log2-ratio entre les 2 conditions ###
par(mfrow = c(1,1))
DESeq2::plotMA(dds)
# les gènes différentiellement exprimés au seuil de 10% après ajustement
# des tests multiples parprocédure de Benjamini-Hochberg (BH) sont indiqués
# en rouge

#############################################################
### code 7: Résultats de l'analyse différentielle
#############################################################
#res contient les résultats de l'analyse différentielle pour la dernière variable
res <- results(dds)
head(res)
resultsNames(dds)
mcols(res,use.names=T)
# distribution des p-valeurs brutes
par(mfrow = c(1,1))
hist(res$pvalue, breaks=100, border="slateblue",
     main="Histogramme des p-values brutes")
#Combien de gènes sont déclarés DE à 5% ?
table(res$padj < 0.05,useNA="always")
# Selection des gènes déclarés DE à 5% dans une table ordonnée par p-value
# croissante
resSig <- na.omit(res)
resSig <- resSig[resSig$padj < 0.05, ]
resSig <- resSig[ order(resSig$pval), ]
nrow(resSig)
head(resSig)

#############################################################################
### code 8: Comparaison des p-values brutes, ajustées Bonferroni et BH
#############################################################################
# calcul des p-values ajustées selon la méthode de bonferroni
res$padjBonf <- p.adjust(res$pval,meth="bonferroni")
head(res)
#histogramme des distributions des p-values
par(mfrow=c(1,1))
hist(res$padjBonf, border = "blue",xlab = "pvalue",
     main = "distributions des p-values")
```

```
hist(res$padj, add = TRUE, border = "red")
hist(res$pval, add = TRUE)
legend(x = "topleft", legend = c("pVal", "padjBonf", "padjBH"),
       fill = c("black", "blue", "red"))
# cumul des p-values
resord <- res[order(res$pval),]
plot(resord$pval,ylab="Pvalues",cex=0.5,xlab="rang du gène")
points(resord$padjBonf,col="blue",cex=0.5)
points(resord$padj,col="red",cex=0.5)
abline(h=0.05,lty=2)
legend(x = "bottomright", legend = c("pVal", "padjBonf", "padjBH"),
       col = c("black", "blue", "red"),pch=1)

#######################################################################
### code 9: Classification Ascendante Hiérarchique (CAH)
#######################################################################
#CAH
cdslog <- log(counts(dds)+1)
dist.cor <- 1-cor(cdslog,use="pairwise.complete.obs", method = "spearman")
hc.cor <- hclust(as.dist(dist.cor),method="ward")
#graph
plot(hc.cor, main=paste("CAH des échantillons à partir des comptages \n de ",
                        nrow(cdslog),"gènes ",sep=""),
     sub="Distance= 1-corrélation")
colData(dds)

#######################################################################
### code 10: HEATMAP
#######################################################################
# Stabilization variance is usefull for visualization methods like
# clustering, PCA
rld <- rlogTransformation(dds,blind=T)
dists <- dist( t( assay(rld) ) )
mat <- as.matrix( dists )
hmcol = colorRampPalette(brewer.pal(9, "GnBu"))(100)
heatmap.2(mat, trace="none", col = rev(hmcol), margin=c(13, 13),
          main="heatmap with stabilization variance regularized log Transformation")
```

B. Script R Markdown, deuxième jeu de données

Analyse DESeq Données “Papillon” RNA-Seq

1. Packages

```
library(DESeq2)
library(Biobase)
library(gplots)
library(ggplot2)
require(RColorBrewer)
```

2. Tables de comptage et de conditions

On charge la table des données brutes de comptage, ainsi que la table des conditions à laquelle ont été ajoutées au préalable une colonne “librairie” (de L1 à L8), et une colonne “sampleID” (pour identifier chaque échantillon). Ainsi on peut renommer les échantillons dans la table de comptage pour rendre plus lisibles les résultats ainsi que les graphiques.

```
setwd("/home/sbesbes/Bureau/doc/papillon")
data<-read.table("matrice.txt", header=TRUE, row.names=1)
head(data)
cond_table<-read.csv("~/Bureau/doc/papillon/Conditions(1).csv", sep=";")
colnames(data)<-cond_table$sample
```

```
head(data[,1:11],n=3)
```

```
##                    L1-1 L1-2 L1-3 L1-4 L1-5 L2-1 L2-2 L2-3 L2-4 L2-5 L3-1
## BA_-W-2.1.1           0    0    0    0    0    0    0    0    0    0    0
## BA_128UP.1.1          0    0    0    0    0    0    0    2    0    2    0
## BA_14-3-3ZETA.1.10    0    0    0    1    0    0    0    0    0    0    0
```

3. Contigs “ONLY L8”

On recherche des contigs qui s'expriment seulement dans la librairie 8 (“adult_antenna”). Quels sont donc les contigs aux comptage nuls dans les 36 premiers échantillons et avec au moins un comptage non nul dans les 6 derniers échantillons (librairie 8)?

```
sum36<-apply(data[,1:36], 1, sum)
sum6<-apply(data[,37:42], 1, sum)
pos<-which(sum36==0 & sum6!=0)        #positions des contigs qui satisfont
cette condition
onlyL8<-data[pos,37:42]               # table des contigs qui s'expriment que
dans L8
onlyL8$total<-rowSums(onlyL8)         # ajout d'une colonne "total"
onlyL8<-onlyL8[order(onlyL8$total),]  # on ordonne par le total des copies
onlyL8_more5<-onlyL8[onlyL8$total>=5,] # on séléctionne dans une table les
contigs seulement exprimés en L8 avec plus de 5 copies
head(onlyL8)
```

```
##                L8-1 L8-2 L8-3 L8-4 L8-5 L8-6 total
## BA_-W-2.1.1       0    0    1    0    0    0     1
## BA_18W.3.7        0    0    0    0    1    0     1
## BA_26-29-P.1.2    0    0    0    0    1    0     1
## BA_26-29-P.2.2    0    0    0    0    0    1     1
## BA_2A5G.1.2       0    0    0    1    0    0     1
## BA_3HCD.1.2       0    0    1    0    0    0     1
```

Des fichiers .csv contenant ces tables sont crées:

```
write.table(onlyL8, file="onlyL8_tab.csv", sep="\t", eol="\n", dec=".")
write.table(onlyL8_more5, file="onlyL8_more5_tab.csv", sep="\t", eol="\n",
dec=".")
```

Pour lire ces fichiers, utiliser les codes :

```
onlyL8<-read.csv("~/Bureau/doc/papillon/onlyL8_tab.csv", sep="\t")
onlyL8_more5<-read.csv("~/Bureau/doc/papillon/onlyL8_more5_tab.csv", sep="\t")
```

4. Filtrage

On va effectué 3 types de filtrage de contigs :

- Filtrage des contigs qui ne s'expriment que dans L8;
- Filtrage des contigs aux comptages toujours nuls;
- Filtrage"5" : on ne garde que les contigs tels que au moins un des échantillons a une mesure supérieure à 5.

A l'issue de ce filtrage on obtient la table **“final_data”**

```
# Filtrage des "only L8"
if (length(pos)!=0) {my_data<-data[-pos,]} else {my_data<-data}
dim(my_data)
```

```
## [1] 90578    42
```

```
# Filtrage des contigs inexprimés
my_data<-my_data[rowSums(my_data)!=0,]
dim(my_data)
```

```
## [1] 83209    42
```

```
# Filtrage"5"
pos5<-which(apply(my_data,1,max)>=5) # Position des gènes dont le comptage
maximum est supérieur à 5
final_data<-my_data[pos5,]
dim(final_data)
```

```
## [1] 5101   42
```

5. Construction objet DESeqDataSet : dds

Plan d'action : comparer la librairie 8 avec l'ensemble des librairies {1,2,6,7}.

On va donc créer 2 conditions:

- condition 1 pour L8
- condition 2 pour L1, L2, L6 et L7

On va également retirer de l'analyse suivante les librairies 3, 4, et 5 auxquels on ne s'interesse pas ici.On obtiendra ici une table de comptage **“new_data”** sans les librairies 3,4 et 5.

```
## On crée la table colData qui décrit le plan d'expérience: librairies et
conditions pour l'analyse statistique
# création du vecteur des librairies :"lib"
lib<-factor(cond_table$Library)
# création d'un premier colData_full (qui contient toutes les librairies)
colData_full<-data.frame(lib,row.names=colnames(data))
head(colData_full, n=12)
```

```
# création du colData sans les librairies 3, 4 et 5 et ajout d'un facteur
"condition"
out<-which(lib=="L3" | lib=="L4" | lib=="L5")
colData<-data.frame(lib=lib[-out],row.names=colnames(data[,-out]))
# condition 1 pour L1, L2, L6 et L7, condition 2 pour L8
colData$condition<-factor(append(rep("cond1",length(colData$lib)-6),rep("cond2",6)))
colData
```

```
##      lib condition
## L1-1  L1     cond1
## L1-2  L1     cond1
## L1-3  L1     cond1
## L1-4  L1     cond1
## L1-5  L1     cond1
## L2-1  L2     cond1
## L2-2  L2     cond1
## L2-3  L2     cond1
## L2-4  L2     cond1
## L2-5  L2     cond1
## L6-1  L6     cond1
## L6-2  L6     cond1
## L6-3  L6     cond1
## L6-4  L6     cond1
## L6-5  L6     cond1
## L7-1  L7     cond1
## L7-2  L7     cond1
## L7-3  L7     cond1
## L7-4  L7     cond1
## L7-5  L7     cond1
## L7-6  L7     cond1
## L8-1  L8     cond2
## L8-2  L8     cond2
## L8-3  L8     cond2
## L8-4  L8     cond2
## L8-5  L8     cond2
## L8-6  L8     cond2
```

```
# création d'une table de comptage sans les librairies 3, 4 et 5 => new_data
new_data<-final_data[,-out]
head(new_data)
```

```
##                     L1-1 L1-2 L1-3 L1-4 L1-5 L2-1 L2-2 L2-3 L2-4 L2-5 L6-1
## BA_14-3-3ZETA.10.10    0    0    3   12    4    0    0    9   19    9    1
## BA_14-3-3ZETA.7.10     0    0    0    2    2    0    0    0    2    2    0
## BA_1433E.1.1           1    2   22   15   41    0    3   35   50   46    0
## BA_1433Z.12.14         0    0    3    2    4    0    1    5    6    4    0
## BA_1433Z.9.14          0    0    2    2    2    0    0    9    3    6    0
## BA_3HCD.2.2            0    0    1    3    0    0    0    2    1    0    0
##                     L6-2 L6-3 L6-4 L6-5 L7-1 L7-2 L7-3 L7-4 L7-5 L7-6 L8-1
## BA_14-3-3ZETA.10.10    0    0    1    0    0    5   13   18   17   11    0
## BA_14-3-3ZETA.7.10     0    0    0    0    0    0    1    0    2    1    0
## BA_1433E.1.1           0    0    0    0    0    0    9    0    0    0    0
## BA_1433Z.12.14         0    0    0    0    0    0    2    4    1    1    0
## BA_1433Z.9.14          0    0    0    0    0    0    1    1    3    1    0
## BA_3HCD.2.2            0    0    0    0    0    0    2    2    5    4    0
##                     L8-2 L8-3 L8-4 L8-5 L8-6
## BA_14-3-3ZETA.10.10    0   11   21   37   30
## BA_14-3-3ZETA.7.10     0    1    0    5    6
## BA_1433E.1.1           0   11    0    2    1
## BA_1433Z.12.14         0    2    8    6    5
## BA_1433Z.9.14          0    4    9    9   13
## BA_3HCD.2.2            0    5   11   15   14
```

On construit maintenant l'objet **dds** et on indique que le niveau de référence du facteur “condition” est la condition 1.

```
## création de l'objet dds et niveau de référence
dds<-DESeqDataSetFromMatrix(new_data, colData, design=~condition)
colData(dds)$condition<-relevel(colData(dds)$condition,ref="cond1")
# objet dds
class(dds)
```

```
## [1] "DESeqDataSet"
## attr(,"package")
## [1] "DESeq2"
```

```
colData(dds)
```

```
## DataFrame with 27 rows and 2 columns
##           lib condition
##      <factor>  <factor>
## L1-1       L1     cond1
## L1-2       L1     cond1
## L1-3       L1     cond1
## L1-4       L1     cond1
## L1-5       L1     cond1
## ...       ...       ...
## L8-2       L8     cond2
## L8-3       L8     cond2
## L8-4       L8     cond2
## L8-5       L8     cond2
## L8-6       L8     cond2
```

```
design(dds)
```

```
## ~condition
```

```
# manipulation des données de comptage
dim(counts(dds))
head(counts(dds))
print(summary(counts(dds)))
```

6. Exploration des données APRÈS filtrage

```
## combien il y a-t-il de comptages nuls par échantillon ?
apply(counts(dds), 2, FUN = function(x) sum(x ==0))
```

```
## L1-1 L1-2 L1-3 L1-4 L1-5 L2-1 L2-2 L2-3 L2-4 L2-5 L6-1 L6-2 L6-3 L6-4 L6-5
## 4671 4285 2815 2256 2319 4681 4272 2391 1846 1817 4646 4243 5010 4972 4970
## L7-1 L7-2 L7-3 L7-4 L7-5 L7-6 L8-1 L8-2 L8-3 L8-4 L8-5 L8-6
## 4559 3994 1270 2925 2423 2480 4886 4620 1408 1554 1125 1229
```

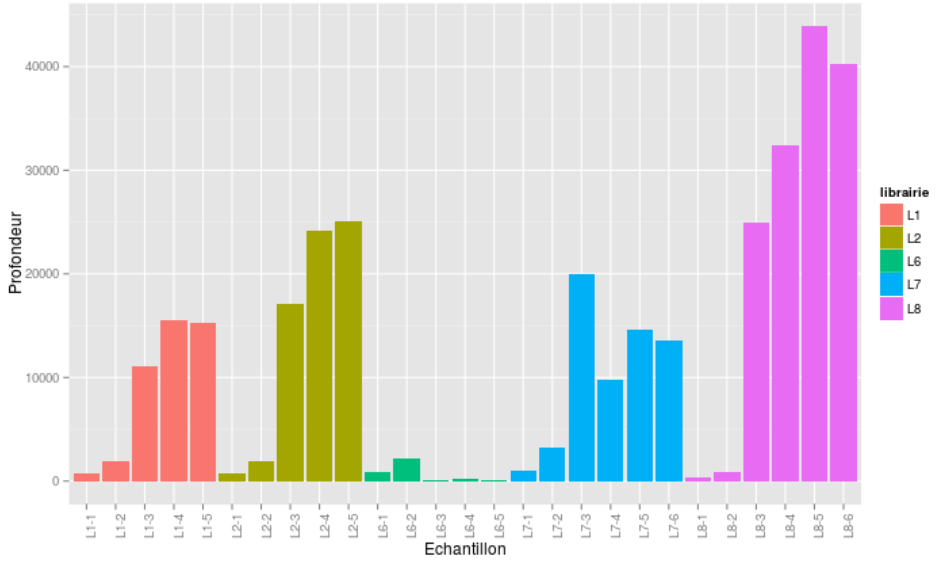
```
## combien de comptages nuls en % ?
apply(counts(dds), 2, FUN = function(x) sum(x ==0))/nrow(counts(dds))
```

```
##   L1-1   L1-2   L1-3   L1-4   L1-5   L2-1   L2-2   L2-3   L2-4   L2-5
## 0.9157 0.8400 0.5519 0.4423 0.4546 0.9177 0.8375 0.4687 0.3619 0.3562
##   L6-1   L6-2   L6-3   L6-4   L6-5   L7-1   L7-2   L7-3   L7-4   L7-5
## 0.9108 0.8318 0.9822 0.9747 0.9743 0.8937 0.7830 0.2490 0.5734 0.4750
##   L7-6   L8-1   L8-2   L8-3   L8-4   L8-5   L8-6
## 0.4862 0.9579 0.9057 0.2760 0.3046 0.2205 0.2409
```

```
## quelles sont les profondeurs de séquençage de chaque échantillon?
colSums(counts(dds))
```

```
##  L1-1  L1-2  L1-3  L1-4  L1-5  L2-1  L2-2  L2-3  L2-4  L2-5  L6-1  L6-2
##   713  1915 11040 15502 15270   764  1989 17099 24114 25135   875  2249
##  L6-3  L6-4  L6-5  L7-1  L7-2  L7-3  L7-4  L7-5  L7-6  L8-1  L8-2  L8-3
##   108   173   156  1073  3274 20038  9807 14588 13509   331   944 24957
##  L8-4  L8-5  L8-6
## 32413 43901 40207
```

```
df=data.frame(librairie=colData$lib, Individu=rownames(colData),
prof=colSums(counts(dds)))
ggplot(df, aes(x=factor(Individu), y=prof, fill =librairie))+geom_bar(stat =
"identity") + xlab("Echantillon") + ylab("Profondeur")+  theme(axis.text.x =
element_text(angle = 90))
```



7. Normalisation et Analyse différentielle

La fonction DESeq() réalise la normalisation et l'analyse diff en une seule étape. Après avoir comparé les résultats des 6 combinaisons possibles des options “test” et “fitType”,on a choisi le Likelihood Ratio Test et une estimation local et non paramétrique.

```
dds<-DESeqDataSetFromMatrix(new_data+1, colData, design=~condition)
dds<-DESeq(dds, test="LRT", fitType="local", reduced=~1)
```

```
## estimating size factors
## estimating dispersions
## gene-wise dispersion estimates
## mean-dispersion relationship
## final dispersion estimates
## fitting model and testing
```

```
colData(dds)
```

```
## DataFrame with 27 rows and 3 columns
##           lib condition sizeFactor
##      <factor>  <factor>  <numeric>
## L1-1       L1     cond1     0.6605
## L1-2       L1     cond1     0.6850
## L1-3       L1     cond1     0.8298
## L1-4       L1     cond1     1.1143
## L1-5       L1     cond1     1.0655
## ...       ...       ...        ...
## L8-2       L8     cond2     0.6625
## L8-3       L8     cond2     1.6638
## L8-4       L8     cond2     1.8642
## L8-5       L8     cond2     3.1299
## L8-6       L8     cond2     2.8792
```

Remarque: On modifie ici l'objet dds, et plus précisément la table de comptage, avec “new_data**+1**”. En effet tous les contigs ont au moins un comptage nul ce qui empêche le calcul du facteur de normalisation (“size factor”) qui est calculé entre autre au moyen de la moyenne géométrique qui doit être différente de zéro.

La normalisation des données de comptage se fait par la méthode “effective library size”: Les facteurs d'échelle ont été obtenus grâce au calcul suivant:

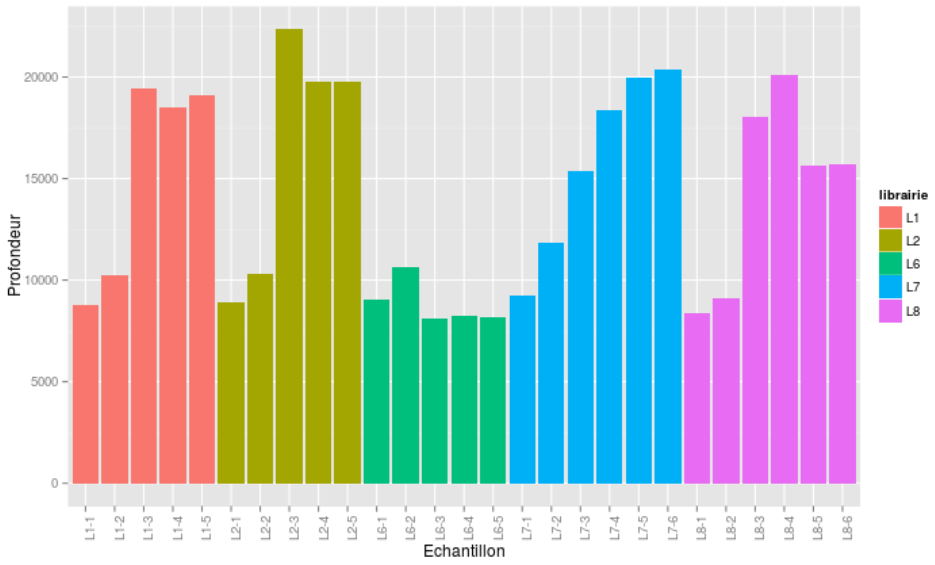
- MoyGeomLignes <- apply(counts(dds), 1, function(x) (prod(x))$^{1/\text{length}(x}$))
- apply(counts(dds)[MoyGeomLignes != 0,] / MoyGeomLignes[MoyGeomLignes != 0], 2, median)

Visualisation de la normalisation

```
#profondeur de séquençage après normalisation
colSums(counts(dds, normalized = TRUE))
```

```
##  L1-1  L1-2  L1-3  L1-4  L1-5  L2-1  L2-2  L2-3  L2-4  L2-5  L6-1  L6-2
##  8802 10243 19451 18489 19118  8883 10330 22358 19753 19791  9020 10663
##  L6-3  L6-4  L6-5  L7-1  L7-2  L7-3  L7-4  L7-5  L7-6  L8-1  L8-2  L8-3
##  8130  8223  8196  9252 11817 15398 18343 19979 20404  8388  9124 18066
##  L8-4  L8-5  L8-6
## 20123 15656 15736
```

```
df=data.frame(librairie=colData$lib, Individu=rownames(colData),
prof=colSums(counts(dds,normalized = TRUE)))
ggplot(df, aes(x=factor(Individu), y=prof, fill =librairie))+geom_bar(stat =
"identity") + xlab("Echantillon") + ylab("Profondeur")+  theme(axis.text.x =
element_text(angle = 90))
```



```
#données de comptage normalisées
head(counts(dds, normalized = TRUE),n=3)
summary(counts(dds, normalized = TRUE))
```

Dispersion des contigs La fonction DESeq() a effectué une estimation de la dispersion des contigs

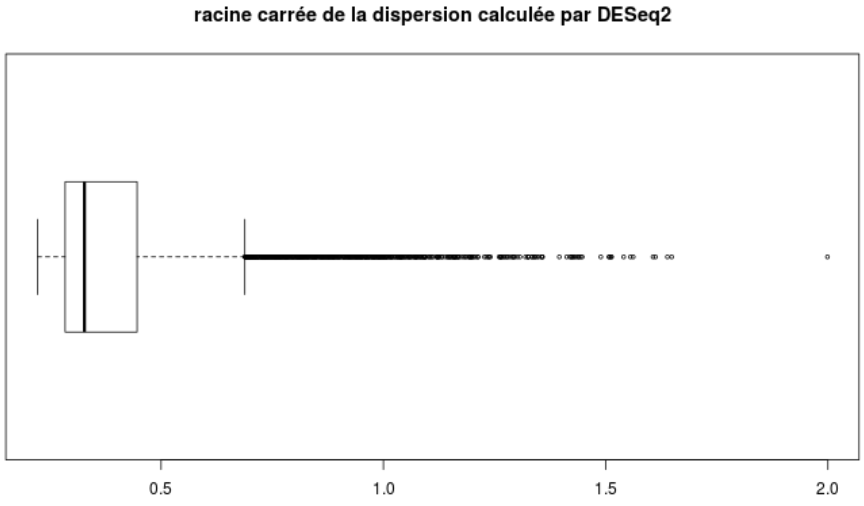
```
# Valeurs de dispersion:
disp<-as.data.frame(dispersions(dds))
head(disp,n=3)
```

```
##   dispersions(dds)
## 1          0.61538
## 2          0.07652
## 3          1.30943
```

```
print(summary(disp))
```

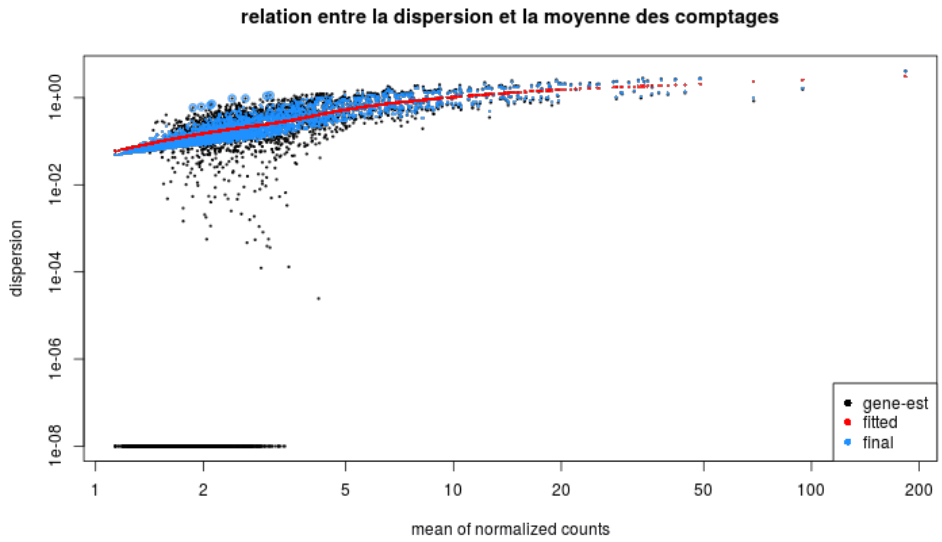
```
##  dispersions(dds)
##  Min.   :0.049
##  1st Qu.:0.081
##  Median :0.107
##  Mean   :0.220
##  3rd Qu.:0.199
##  Max.   :3.996
```

```
# Boxplot des valeurs de dispersion
par(mfrow=c(1,1))
boxplot(sqrt(disp),horizontal=T,las=1,cex=0.5)
title("racine carrée de la dispersion calculée par DESeq2")
```



Elle a ensuite estimé la relation dispersion-moyenne des comptages (estimation local et non paramétrique, voir option DESeq())

```
par(mfrow = c(1,1))
DESeq2::plotDispEsts(dds)
title("relation entre la dispersion et la moyenne des comptages")
```

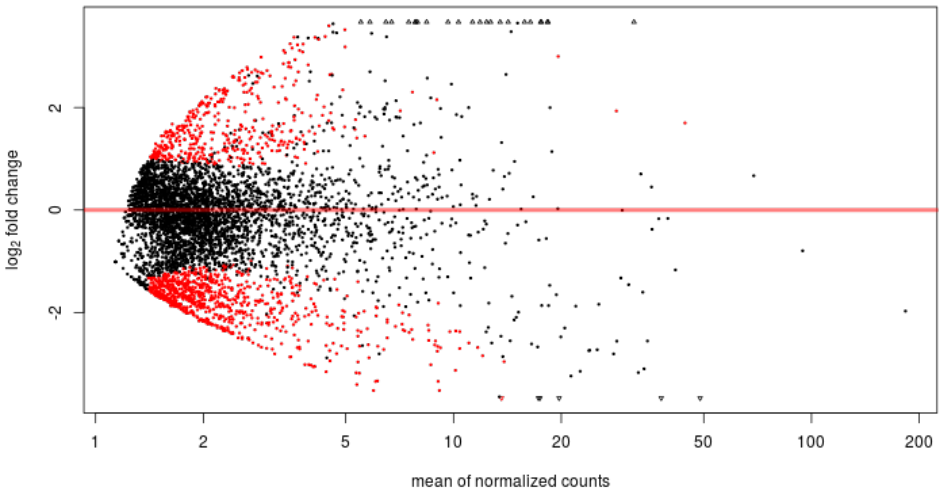


Pour obtenir plus details sur les calculs des valeurs de dispersion, tests statistiques et pvalues:

```
details<-mcols(dds,use.names=T)
head(details,n=3)
mcols(mcols(dds,use.names=T))
```

Graphique MA-plot : relation entre le comptage moyen d'un contig et son log2-ratio entre les 2 conditions

```
par(mfrow = c(1,1))
DESeq2::plotMA(dds)
```



Les contigs différentiellement exprimés au seuil de 10% après ajustement des tests multiples par la procédure de Benjamini-Hochberg (BH) sont indiqués en rouge

8. Résultats de l'analyse différentielle

res contient les résultats de l'analyse différentielle pour le dernier facteur (ici l'unique facteur : “condition”)

```
res<-results(dds)
head(res)
```

```
## DataFrame with 6 rows and 6 columns
##                      baseMean log2FoldChange     lfcSE      stat    pvalue
##                     <numeric>      <numeric> <numeric> <numeric> <numeric>
## BA_14-3-3ZETA.10.10     6.899         0.1841    0.5705   0.10974   0.74044
## BA_14-3-3ZETA.7.10      1.668        -0.1134    0.4718   0.05874   0.80850
## BA_1433E.1.1            8.527        -2.3478    0.8481   6.08158   0.01366
## BA_1433Z.12.14          2.608        -0.1970    0.4288   0.21356   0.64399
## BA_1433Z.9.14           2.681         0.4298    0.4172   1.09562   0.29523
## BA_3HCD.2.2             2.555         1.0046    0.4027   6.51921   0.01067
##                          padj
##                     <numeric>
## BA_14-3-3ZETA.10.10   0.86818
## BA_14-3-3ZETA.7.10    0.90158
## BA_1433E.1.1          0.05706
## BA_1433Z.12.14        0.81041
## BA_1433Z.9.14         0.51003
## BA_3HCD.2.2           0.04891
```

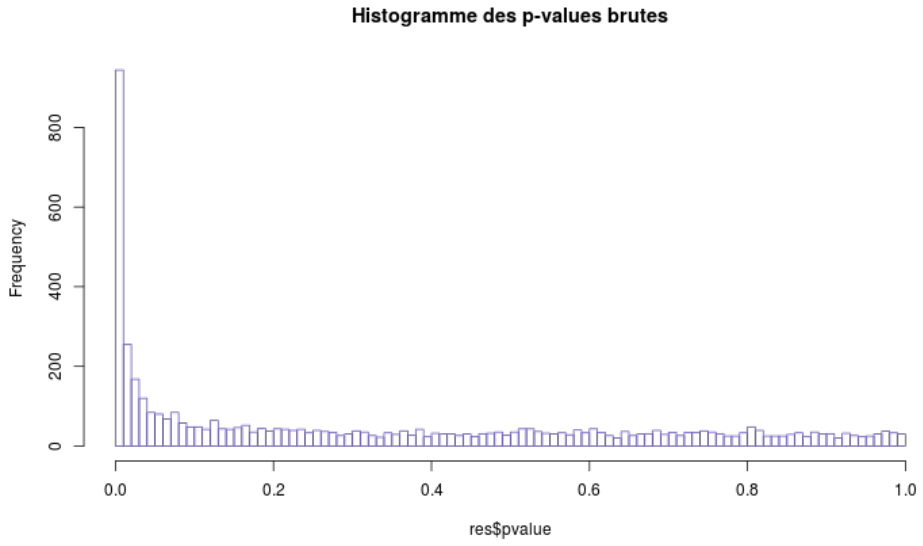
```
resultsNames(dds)
```

```
## [1] "Intercept"               "condition_cond2_vs_cond1"
```

```
mcols(res,use.names=T)
```

```
## DataFrame with 6 rows and 2 columns
##                        type                                 description
##                 <character>                                 <character>
## baseMean       intermediate                the base mean over all rows
## log2FoldChange      results log2 fold change: condition cond2 vs cond1
## lfcSE               results   standard error: condition cond2 vs cond1
## stat                results      LRT statistic: '~ condition' vs '~ 1'
## pvalue              results        LRT p-value: '~ condition' vs '~ 1'
## padj                results                          BH adjusted p-values
```

```
# distribution des p-valeurs brutes
par(mfrow = c(1,1))
hist(res$pvalue,
     breaks=100,
     border="slateblue",
     main="Histogramme des p-values brutes")
```



Combien de gènes sont déclarés DE à 5% ?

```
table(res$padj < 0.05,useNA="always")
```

```
##
## FALSE  TRUE  <NA>
##  3423   984   694
```

Selection des gènes déclarés DE à 5% dans une table ordonnée par p-value croissante => **resSig**

```
resSig<-na.omit(res)
resSig <- resSig[resSig$padj < 0.05, ]
resSig<-resSig[ order(resSig$pval), ]
head(resSig)
```

```
## DataFrame with 6 rows and 6 columns
##                   baseMean log2FoldChange     lfcSE      stat    pvalue
##                  <numeric>      <numeric> <numeric> <numeric> <numeric>
## BA_contig_104489     4.199          3.331    0.4455     69.18 8.966e-17
## BA_CCCP.5.7          4.486          3.598    0.5107     62.25 3.030e-15
## BA_PBP6.3.4          4.971          3.519    0.5198     57.69 3.071e-14
## BA_contig_104409     2.901          2.985    0.4371     57.36 3.627e-14
## BA_contig_109734     4.979          3.184    0.4823     53.81 2.206e-13
## BA_RPS15.11.12       4.103          3.371    0.5147     53.55 2.517e-13
##                       padj
##                  <numeric>
## BA_contig_104489 3.952e-13
## BA_CCCP.5.7      6.677e-12
## BA_PBP6.3.4      3.996e-11
## BA_contig_104409 3.996e-11
## BA_contig_109734 1.849e-10
## BA_RPS15.11.12   1.849e-10
```

Une table **resSig.csv** qui contient les contigs déclarés DE à 5% ET ordonnée par p-value croissante est crée:

```
write.table(resSig, file="results_Sig.csv", sep="\t", eol="\n", dec=".")
```

Rappelons que le niveau de référence est la “cond1” qui correspond à l'ensemble {L1,L2,L6,L7}.

```
# contigs surexprimés dans L8 ordonnés par pval croissante
over<-resSig[resSig$stat>=0,]
dim(over)
```

```
## [1] 984   6
```

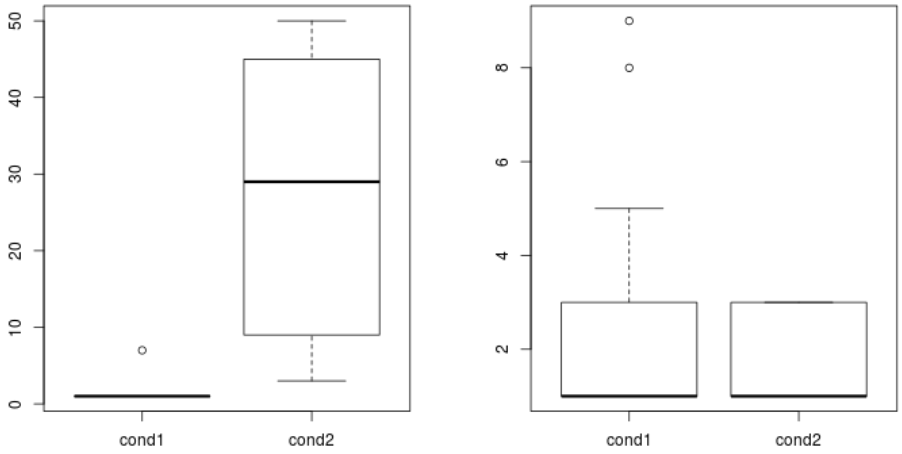
```
# contigs sous-exprimés dans L8 ordonnés par pval croissante
under<-resSig[resSig$stat<0,]
dim(under)                    # pas de sous-expression
```

```
## [1] 0 6
```

Boxplot du niveau d'expression selon la condition pour le gène le plus significatif et pour le gène le moins significatif

```
dim(resSig)
resSig[c(1,984),]
which(rownames(new_data)=="BA_contig_104489")    #le plus significatif
which(rownames(new_data)=="BA_DVIR_GJ17302.3.3") #le moins significatif
```

```
par(mfrow=c(1,2))
boxplot(counts(dds)[4563,]~colData(dds)$condition)  # +
boxplot(counts(dds)[1037,]~colData(dds)$condition)  # -
```

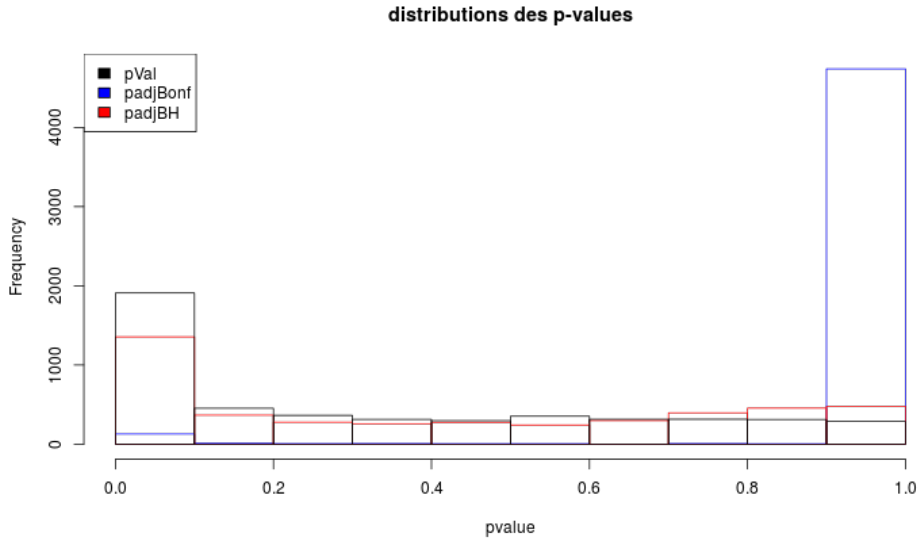


9. Comparaison des p-values brutes, ajustées Bonferroni et BH

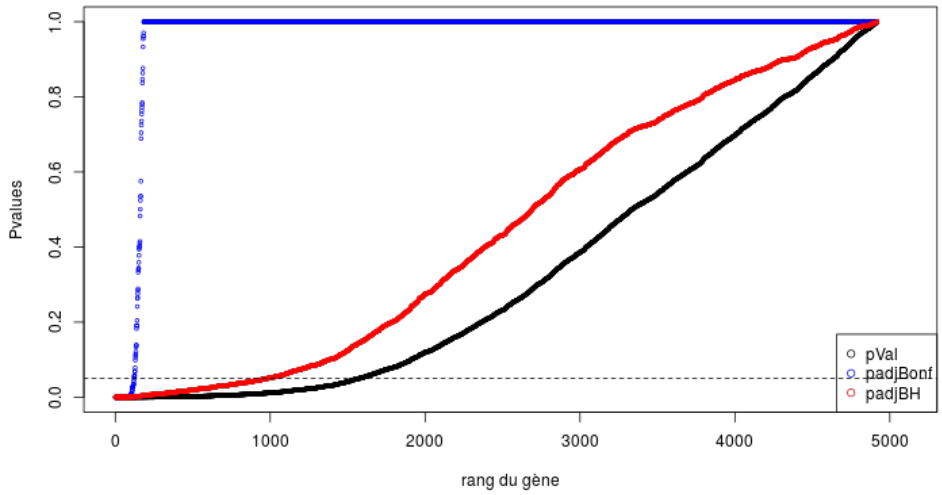
```
# calcul des p-values ajustées selon la méthode de bonferroni
res$padjBonf <- p.adjust(res$pval,meth="bonferroni")
head(res)
```

```
## DataFrame with 6 rows and 7 columns
##                      baseMean log2FoldChange     lfcSE      stat    pvalue
##                     <numeric>      <numeric> <numeric> <numeric> <numeric>
## BA_14-3-3ZETA.10.10     6.899         0.1841    0.5705   0.10974   0.74044
## BA_14-3-3ZETA.7.10      1.668        -0.1134    0.4718   0.05874   0.80850
## BA_1433E.1.1            8.527        -2.3478    0.8481   6.08158   0.01366
## BA_1433Z.12.14          2.608        -0.1970    0.4288   0.21356   0.64399
## BA_1433Z.9.14           2.681         0.4298    0.4172   1.09562   0.29523
## BA_3HCD.2.2             2.555         1.0046    0.4027   6.51921   0.01067
##                          padj  padjBonf
##                     <numeric> <numeric>
## BA_14-3-3ZETA.10.10   0.86818         1
## BA_14-3-3ZETA.7.10    0.90158         1
## BA_1433E.1.1          0.05706         1
## BA_1433Z.12.14        0.81041         1
## BA_1433Z.9.14         0.51003         1
## BA_3HCD.2.2           0.04891         1
```

```
#histogramme des distributions des p-values
par(mfrow=c(1,1))
hist(res$padjBonf, border = "blue",xlab="pvalue",main= "distributions des
p-values")
hist(res$padj, add = TRUE, border = "red")
hist(res$pval, add = TRUE)
legend(x = "topleft",
       legend = c("pVal", "padjBonf", "padjBH"),
       fill = c("black", "blue", "red"))
```




```
# cumul des p-values
resord <- res[order(res$pval),]
plot(resord$pval,ylab="Pvalues",cex=0.5,xlab="rang du gène")
points(resord$padjBonf,col="blue",cex=0.5)
points(resord$padj,col="red",cex=0.5)
abline(h=0.05,lty=2)
legend(x = "bottomright",
       legend = c("pVal", "padjBonf", "padjBH"),
       col = c("black", "blue", "red"),pch=1)
```

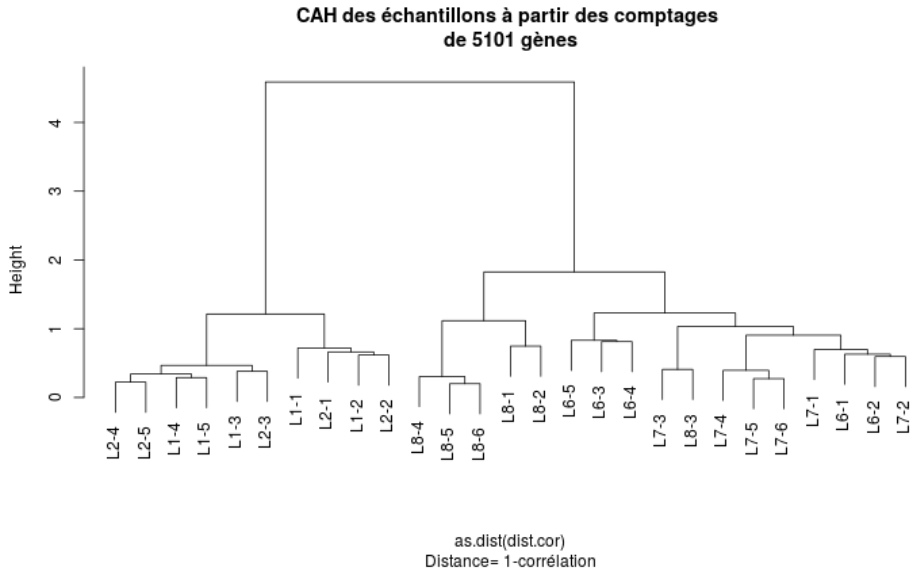


10. Classification Ascendante Hiérarchique (CAH)

```
#CAH
cdslog<-log(counts(dds))
dist.cor<-1-cor(cdslog,use="pairwise.complete.obs", method = "spearman")
hc.cor<-hclust(as.dist(dist.cor),method="ward")
```

```
## The "ward" method has been renamed to "ward.D"; note new "ward.D2"
```

```
#graph
plot(hc.cor,
     main=paste("CAH des échantillons à partir des comptages \n de ",
                nrow(cdslog)," gènes ",sep=""),
     sub="Distance= 1-corrélation")
```



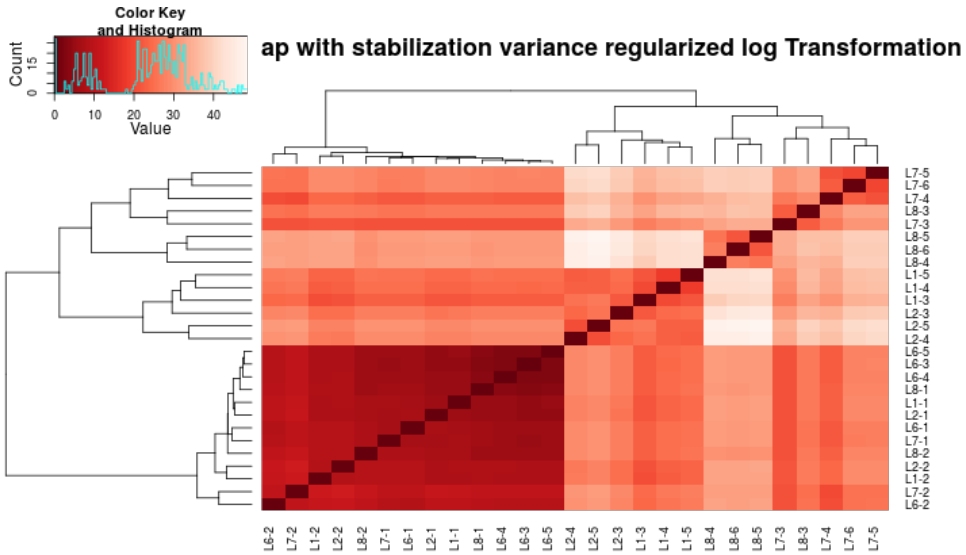
11. HEATMAP

```
#Stabilization variance is usefull for visualization methods like clustering,
PCA
rld<-rlogTransformation(dds,blind=T)
```

```
## you had estimated gene-wise dispersions, removing these
## you had estimated fitted dispersions, removing these
```

```
## Warning: production de NaN
## Warning: step size truncated due to divergence
## Warning: production de NaN
## Warning: step size truncated due to divergence
## Warning: glm.fit: algorithm did not converge
## Warning: the parametric fit of dispersion estimates over the mean of counts
## failed, which occurs when the trend is not well captured by the
## function y = a/x + b. A local regression fit is automatically performed,
## and the analysis can continue. You can specify fitType='local' or 'mean'
## to avoid this message if re-running the same data.
## When using local regression fit, the user should examine plotDispEsts(dds)
## to make sure the fitted line is not sharply curving up or down based on
## the position of individual points.
```

```
dists <- dist( t( assay(rld) ) )
mat <- as.matrix( dists )
hmcol = colorRampPalette(brewer.pal(9, "Reds"))(100)
heatmap.2(mat, trace="none", col = rev(hmcol), margin=c(5, 5),main="heatmap
with stabilization variance regularized log Transformation")
```


Color Key
and Histogram
Count
0
15
0
10
20
30
40
Value
ap with stabilization variance regularized log Transformation
L7-5
L7-6
L7-4
L8-3
L7-3
L8-5
L8-6
L8-4
L1-5
L1-4
L1-3
L2-3
L2-5
L2-4
L6-5
L6-3
L6-4
L8-1
L1-1
L2-1
L6-1
L7-1
L8-2
L2-2
L1-2
L7-2
L6-2
L6-2
L7-2
L1-2
L2-2
L8-2
L7-1
L6-1
L2-1
L1-1
L8-1
L6-4
L6-3
L6-5
L2-4
L2-5
L2-3
L1-3
L1-4
L1-5
L8-4
L8-6
L8-5
L7-3
L8-3
L7-4
L7-6
L7-5