# Etude évolutive du transcriptome des cellules pancréatiques matures chez les vertébrés

**Auteur :** Wayet, Jérome
**Promoteur(s) :** Peers, Bernard
**Faculté :** Faculté des Sciences
**Diplôme :** Master en biochimie et biologie moléculaire et cellulaire, à finalité spécialisée en bioinformatique et modélisation
**Année académique :** 2017-2018
**URI/URL :** http://hdl.handle.net/2268.2/5200

# Etude évolutive du transcriptome des cellules pancréatiques matures chez les vertébrés

Mémoire en vue de l'obtention du diplôme de Master en Biochimie, Biologie Moléculaire et Cellulaire - Finalité Bioinformatique

24 août 2018

Jérôme Wayet

Université de Liège (ULiège)

Département des Sciences de la Vie

Faculté des Sciences

Laboratoire : GIGA-Zebrafish Development and Disease Models

Promoteur : Dr. B.Peers

Co-Promoteur : Pr. P.Meyer

# Table des matières

**Remerciements**

Je tiens tout d'abord à remercier le Dr. Bernard Peers pour le temps et l'énergie qu'il a consacré à ma formation dans le cadre de ce mémoire. Je remercie également le Pr. Patrick Meyer, le Dr. Marianne Voz, le Dr. Isabelle Manfroid ainsi que Arnaud Lavergne pour leur encadrement scientifique et tout les conseils qu'ils ont pu me fournir. Enfin, je remercie les membres de l'équipe du laboratoire ≪Zebrafish Development and Disease Models≫ pour leur chaleureux accueille ainsi que pour l'ambiance qui a rendu ce stage aussi agréable que formatif.

**Etude évolutive du transcriptome des cellules pancréatiques matures chez les vertébrés**

Wayet Jérôme - 2018 - GIGA-Zebrafish Development and Disease Models

Promoteur : Dr. B.Peers - Copromoteur : Pr. P.Meyer

Le pancréas est un organe vital chez les vertébrés. Outre son implication dans le mécanisme digestif, il est également chargé de l'homéostasie de la glycémie via une production de plusieurs hormones. Le pancréas est le siège de diverses pathologies comme les adénocarcinomes et le diabète. Afin de mettre en évidence de nouvelles voies thérapeutiques, de nombreuses études tendent à caractériser les spécificités biomoléculaires des différents types cellulaires qui le composent. C'est notamment le cas de l'étude de Tarifeno et al., qui s'est axée sur une mise en évidence des gènes enrichis dans les différentes catégories de cellules et qui sont conservées entre des vertébrés comme l'homme, la souris et le poisson zèbre. Cette étude a été réalisée à partir de données RNAseq de cellules groupées (≪bulk≫ RNAseq). Ce type de technique présente des limites quant à la diversité et la pureté des échantillons générés contrairement aux méthodes récentes de RNAseq en cellules isolées (≪SingleCell≫ ou ≪scRNAseq≫).

Dans le cadre de ce mémoire, les analyses de Tarifeno et al., ont été reproduites sur base des nombreuses études scRNAseq publiées au cours des dernières années. Une première analyse a été menée afin d'évaluer la variabilité entre ces différentes études ≪SingleCell≫ qui décrivent le pancréas. Il s'est avéré que cette variabilité pouvait être conséquente et seuls trois études ont été jugées suffisamment bonnes et complètes pour faire office de référence. Comparées au RNAseq en cellules groupées, un réel avantage quant à l'utilisation de données scRNAseq est perceptible, notamment par rapport à la faible reproductibilité inter-replicas des données ≪bulk≫ RNAseq humaines.

Grâce à l'apport de ces nouvelles données plus riches en types cellulaires et plus précises, il a été possible de réaliser une analyse inter-espèces pour les cellules alphas, bêtas, deltas et ductales. Les résultats montrent que la signature transcriptomique des cellules endocrines alphas, bêtas et deltas est très peu conservée entre les espèces alors que la signature des cellules ductales (et exocrines) est plus conservée. Ces observations confirment donc l'étude de Tarifeno et al. . Par ailleurs, de nouveaux gènes conservés ont pu être mis en évidence, comme SFXN5 et TOX2 pour les cellules delta.

Les raisons de cette faible conservation de gènes entre vertébrés restent encore à élucider et pourront faire l'objet de recherches ultérieures. De plus, de nouvelles données scRNAseq incluant des cellules acinaires murines permettront de comparer le transcriptome de tous les types cellulaires pancréatiques chez la souris, l'homme et le zebrafish.

# 1 Introduction

## 1.1 Généralités

Le pancréas fait partie des organes essentiels à la vie chez les vertébrés. Il comporte une double fonction. Tout d'abord, il est responsable de la synthèse d'enzymes digestives déversées dans le duodénum qui sont essentielles à l'absorption de nutriments. Son autre fonction est d'ordre endocrinien avec la production d'un cocktail d'hormones contrôlant, entre autres, l'homéostasie de la glycémie.

De par sa double fonction, le pancréas se constitue de deux structures tissulaires distinctes. Il est composé d'une part par un réseau de canaux connectés à des centres acineux responsables de la production et de l'évacuation des sucs digestifs, et d'autre part par des îlots de cellules appelés «îlots de Langherans». Ces îlots se composent eux-mêmes de différents types cellulaires endocriniens. Les cellules alphas se caractérisent par la production de glucagon (gcg) qui a un effet hyperglycémiant en favorisant la libération de sucre dans le sang et en limitant son stockage. A l'inverse, les cellules bêtas se caractérisent par la production d'insuline (ins) qui a un effet hypoglycémiant en limitant la libération de sucre dans le sang et en favorisant son stockage. Les cellules deltas produisent la somatostatine (sst), qui a une action inhibitrice sur la sécrétion d'insuline et de glucagon. Les cellules gammas, également appelées PP, produisent du polypeptide pancréatique (ppy) impliqué dans la régulation de la fonction exocrine du pancréas. Enfin, les cellules epsilons produisent la ghréline, hormone stimulant l'appétit. D'autres types cellulaires sont évidemment présents dans le pancréas et assurent des rôles plus communs comme l'innervation (cellule de Schwann), la structuration («stellate cells»), l'irrigation (cellule endothéliale), et la défense immunitaire (macrophage, lymphocyte, ...) (Figure 1).

## 1.2 Anatomie et phylogénie

Les hormones liées à la régulation de la glycémie ne sont pas récentes au regard de l'évolution. En effet, des protéines appartenant aux familles de l'insuline, du glucagon, de la somatostatine ou du PPY sont présentes chez les invertébrés comme le diptère ou le ver à soie [Falkmer S, 1985] . De nombreuses études ont été réalisées sur la drosophile et ont révélé la présence de plusieurs gènes (dilp ou «drosophila inulin-like-peptide») codant pour des protéines similaires à l'insuline. Les cellules exprimant les gènes dilp2, dilp3 et dilp5 sont de type neuroendocrine et sont localisées dans le cerveau [Baker and Thummel, 2007] . La sécrétion de ces hormones est stimulée par les taux de sucre après la prise de nourriture. Inversement, les cellules de la corpora cardiaca, une

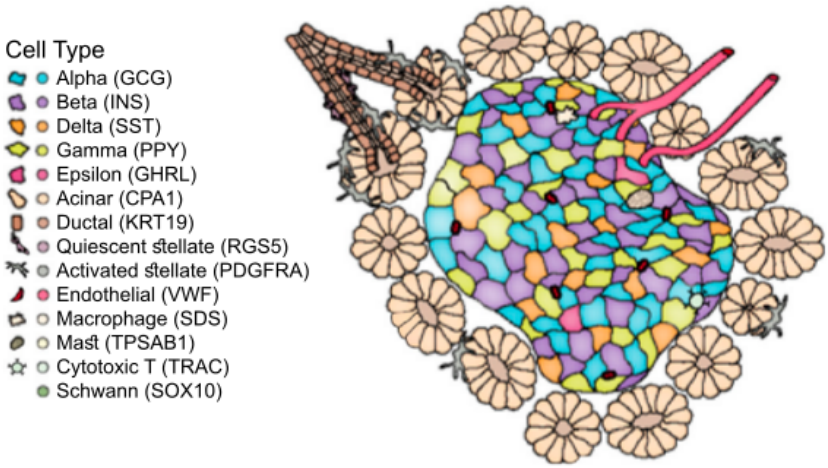


**Figure 1** – *Illustration de la composition d'un îlot de Langherans par [Baron et al., 2016].*

glande endocrine proche du coeur, produisent l'hormone adipokinétique (AKH), qui présente une activité similaire au glucagon. D'autres polypeptides régulateurs sont produits au niveau des cellules entéro-endocrines du tube digestif. Le pancréas n'existe pas chez les invertébrés et il faut attendre l'émergence des vertébrés inférieurs pour voir apparaître cet organe constitué de cellules exocrines et les premiers îlots endocriniens avec au moins quatre types cellulaires distincts, notamment chez les poissons cartilagineux comme le requin et la raie. L'arrivée des vertébrés supérieurs est ensuite marquée par la concentration des cinq types cellulaires décrits précédemment dans les îlots pancréatiques [Heller, 2010] .

On peut également observer des différences entre les îlots des mammifères. Ces différences sont essentiellement liées aux proportions des types cellulaires en présence ainsi qu'à l'architecture de leur disposition (figure 2). Chez le rat et la souris, on trouve respectivement jusqu'à 75 % de cellules bêtas, 15 à 20 % d'alphas, 5 à 10 % de deltas, 1 à 5 % de gammas et moins de 1 % d'epsilons. Dans le cas de l'homme, on retrouve respectivement 30 à 45 % pour les alphas, 50 à 60 % pour les bêtas, moins de 10 % pour les deltas et gammas et moins de 1 % pour les epsilons [Cabrera et al., 2006] . Les cellules bêtas sont concentrées au centre des îlots chez les rongeurs alors que leur répartition est plus dispersée chez les primates.

## 1.3 Pathologies

Le pancréas possède un rôle crucial pour l'homéostasie des vertébrés et il est sujet à diverses pathologies comme des adénocarcinomes ou le diabète. Le cancer du pancréas fait partie des plus agressifs qui soient et offre peu de chances de rémission. Bien que médicalement gérable, le diabète débouche fréquemment sur des complications menant à l'amputation de certains membres

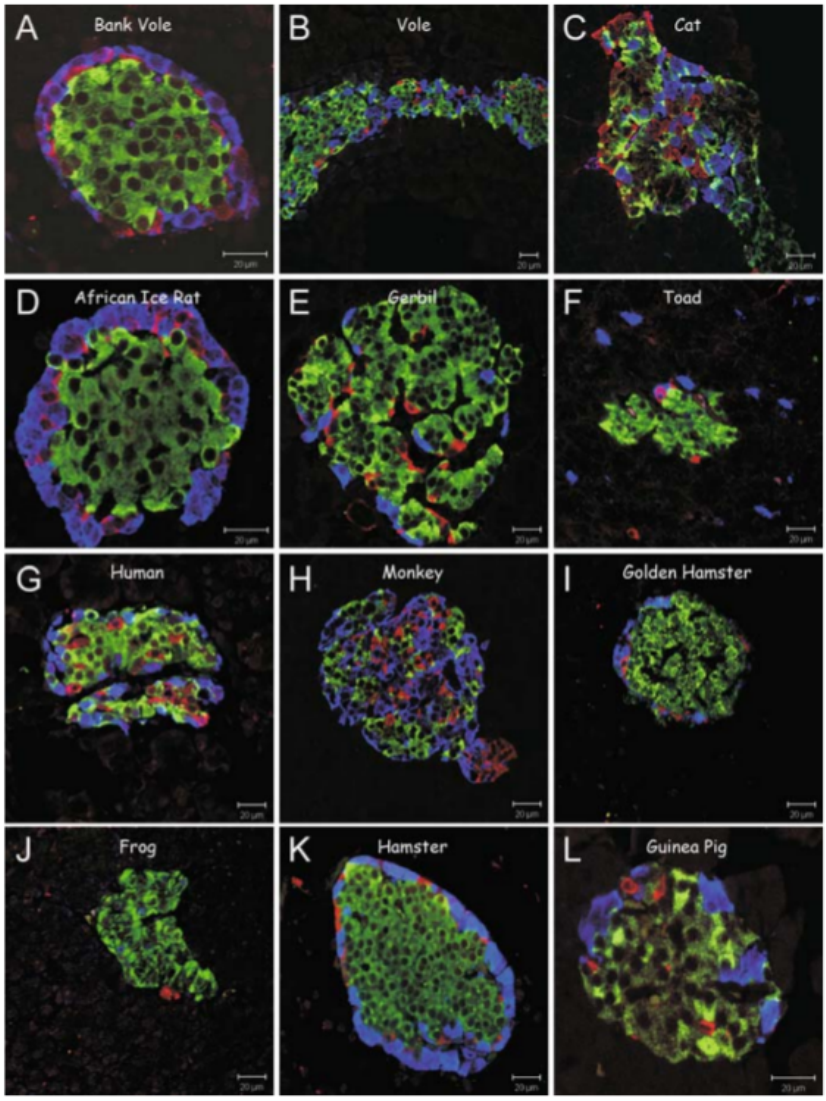


**Figure 2** – *Analyse comparative de l'anatomie d'îlots de Langerhans pour 12 espèces différentes par immunofluorescence. Les sections sont colorées selon l'expression d'insuline (vert), de somatostatine (rouge), et de glucagon (bleu) [Heller, 2010].*

et réduit l'espérance de vie [Leal et al., 2009] . Deux types de diabètes se distinguent : le diabète de type I qui correspond à une attaque des cellules bêtas par le système immunitaire (pathologie auto-immune) et le diabète de type II dû à un dysfonctionnement des cellules bêtas combiné à une diminution de réponse à l'insuline des tissus cibles. L'hyperglycémie chronique devient toxique pour les cellules bêtas. On observe concrètement une diminution de la proportion de cellules bêtas de l'ordre de 60 % ([Butler et al., 2013], [Rahier et al., 2008]) et une sécrétion insuffisante d'insuline accentuant les périodes hyperglycémiques. Une augmentation de la prévalence de cette maladie est prédite durant le 21e siècle notamment due à une augmentation de l'obésité, [Klonoff, 2009] . Dès lors, l'étude du pancréas et de son fonctionnement représente un enjeu

crucial pour le développement de nouvelles thérapies. Cette étude passe notamment par une analyse des caractéristiques biomoléculaires des différents types cellulaires.

## 1.4 Etude du transcriptome des cellules pancréatiques

### 1.4.1 Les analyses RNAseq sur population de cellules (≪bulk RNAseq≫)

Parmi les nombreuses techniques d'analyse biomoléculaire, le séquençage d'ARN (RNAseq) permet d'apprécier le transcriptome. Il peut se résumer comme une prise de vue instantanée de la composition en ARN d'une cellule ou d'un tissu à un moment donné.

Plusieurs méthodes permettent de déterminer le transcriptome : les premières études ont été réalisées à l'aide de micro-puces, mais une véritable émergence de ce type d'étude est apparue avec l'arrivée du séquençage à haut débit (notamment avec les systèmes ILLUMINA) et le RNAseq, technique plus sensible et précise par rapport aux micro-puces.

L'approche classique du RNAseq se base sur un séquençage de cDNA obtenu après la rétro-transcription d'ARNm extrait à partir de biopsies complètes. Néanmoins, cette opération ne permet généralement pas l'étude des spécificités de chaque type cellulaire présent dans la biopsie. De nombreuses études transcriptomiques ont ainsi été réalisées sur des îlots de Langerhans, mais ne permettent pas de distinguer le transcriptome des différents types cellulaires endocrines [Eizirik et al., 2012]. Il peut donc être préférable de désolidariser les prélèvements pour mettre les cellules en suspension et les trier par cytométrie en flux (FACS) via des marqueurs de surface spécifiques ou par fluorescence du produit d'un transgène (la protéine GFP, par exemple)([Blodgett et al., 2015], [DiGruccio et al., 2016]). Il est dès lors possible de sélectionner des cellules du même type en quantité suffisante pour poursuivre le protocole classique de RNAseq. L'analyse de ce type de données RNAseq pour une population de cellules déterminées (analyse en masse ou ≪bulk analysis≫) permet de mettre en évidence de façon précise et efficace les gènes ayant une expression spécifique d'un type cellulaire (appelés marqueurs). La signature transcriptomique d'un type cellulaire correspond alors à l'ensemble des marqueurs spécifiques pour ce type cellulaire. Cependant, cette analyse en masse ne permet pas d'identifier les gènes ayant une expression différente au sein de ce type cellulaire ; ceci ne peut être réalisé que par le RNAseq sur cellules isolées.

### 1.4.2 Les analyses RNAseq sur cellules isolées («single-cell RNAseq»)

Ces dernières années ont vu l'émergence d'une nouvelle méthode de RNAseq, le «Single-Cell RNAseq» (scRNAseq). Cette technique commence par un isolement des cellules dissociées où chacune est soit déposée par FACS dans un puits d'une plaque, soit incorporée dans une micro-goutte contenant les réactifs de lyse et de préparation de librairies (méthode «InDrop» ou «Drop-seq»). Chaque cellule génère sa propre librairie caractérisée par un code-barres (courte séquence unique) inséré sur les fragments amplifiés d'ADNc (Figure 3a et 3b).

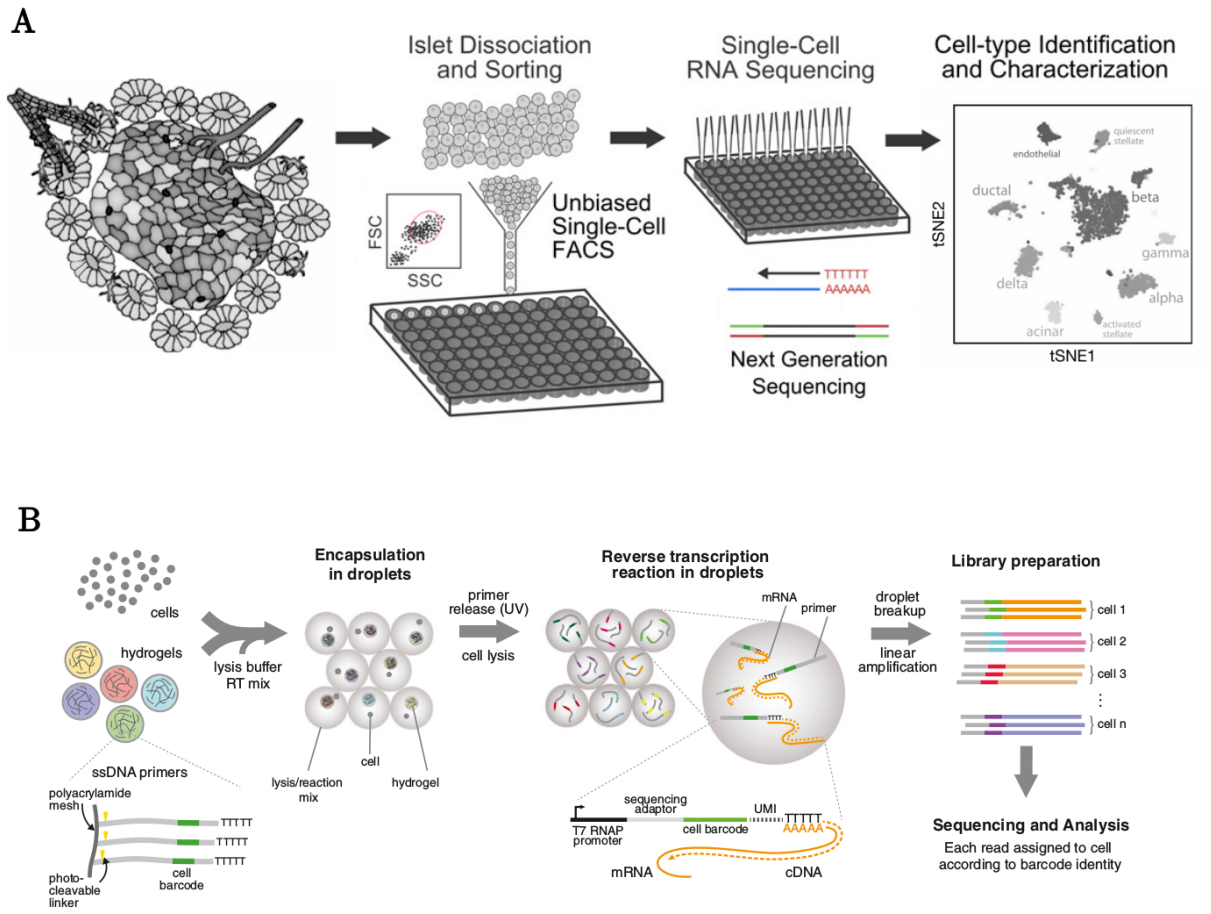


**Figure 3** – *A : Illustration du procédé standard d'analyse Single-Cell en plaques ) micro-puits. B : Illustration du procédé «InDrop» ou «Drop-seq» [Klein et al., 2015].*

La méthode de «Single-Cell» peut donc se résumer en une séparation de cellules en vue d'une préparation de librairies individualisées par l'ajout d'une courte séquence unique (un «code-barres» par cellule) en bordure des adaptateurs de séquençage du premier «read» (séquence partielle du cDNA). Les librairies ainsi marquées peuvent être mélangées pour le séquençage et il est dès lors possible d'utiliser l'information de ces code-barres pour lier l'appartenance des «reads» à chaque cellule initiale. Malgré le mélange des librairies, il est donc possible de

récupérer un fichier de séquences par cellule. Chacun de ces fichiers de données devra être aligné contre un génome de référence comme pour la méthode «Bulk». Au-delà de ce concept général, il est possible de distinguer différentes techniques de «Single-Cell» dont les variations se basent essentiellement sur la méthode de production des librairies.

Dans ce mémoire, notre étude s'est basée sur des données scRNAseq obtenues par deux méthodes différentes : le CelSeq et le SmartSeq (Figure 4).

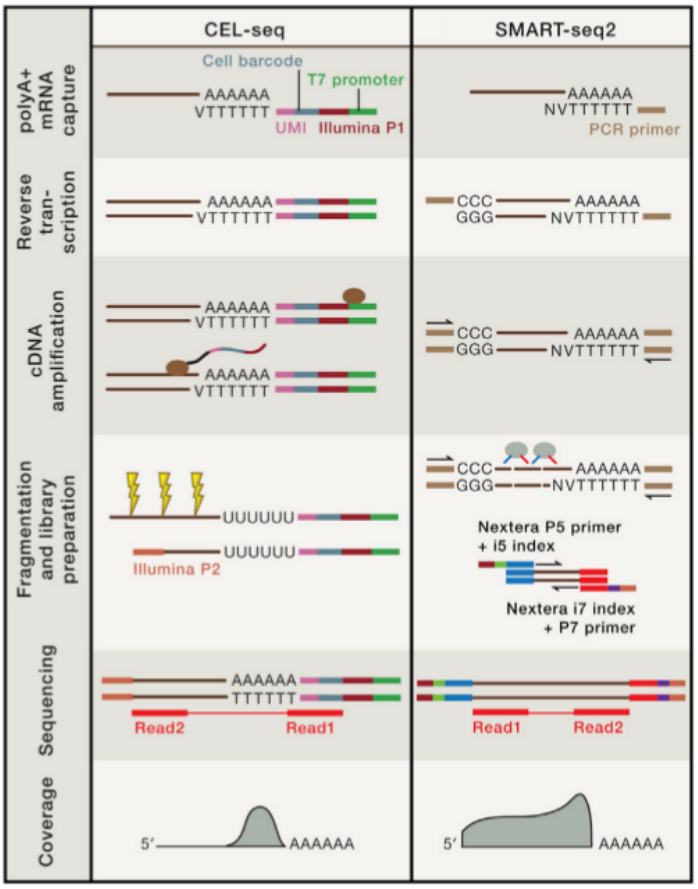


**Figure 4** – *Résumé schématique des méthodes CelSeq et SmartSeq [Grün and Van Oudenaarden, 2015].*

Le CelSeq est l'une des techniques les plus anciennes du Single-Cell [Tang et al., 2010]. Son principe se base sur l'utilisation d'un long oligonucléotide contenant une séquence poly T suivie d'une séquence servant de code-barres spécifique pour chaque cellule, d'une séquence «Unique Molecular Identifiers» (UMI), d'un adaptateur de séquençage ILLUMINA et finalement d'une séquence promotrice T7. Cette amorce permet d'initier la rétrotranscription du transcrit en cDNA à partir de l'extrémité 3' d'ARN porteur de queue poly A. Après la synthèse du second brin, les ADNc sont amplifiés de manière linéaire par transcription à partir du promoteur T7. Les

ARN synthétisés sont fragmentés et le second adaptateur ILLUMINA nécessaire à la préparation de la librairie et au séquençage en ≪paired-end≫ est inséré à l'extrémité 3'.

L'un des avantages de cette méthode est l'incorporation de la séquence UMI, courte séquence unique pour chaque primer. Après amplification et séquençage, il est dès lors possible d'identifier le nombre de transcrits originels en ne comptant plus le nombre de ≪reads≫ alignés par gène, mais le nombre d'UMI différents alignés par gène. Cette subtilité permet de réduire le bruit de fond technique inséré par l'étape d'amplification PCR qui peut biaiser la véritable représentation des proportions de transcrits dans une cellule. Par contre, un désavantage de cette méthode est lié à l'information portée par les ≪reads≫ qu'elle génère. En effet, ces ≪reads≫ ne renseignent essentiellement que sur la région 3' des transcrits. Si cette technique est optimale pour l'analyse du niveau d'expression des gènes entre des types cellulaires, elle est néanmoins moins adaptée à l'étude des profils d'épissage. Le SmartSeq s'avère plus adapté pour de telles études [Grün and Van Oudenaarden, 2015]).

Le SmartSeq est une technique plus récente [Picelli et al., 2013]. Comme le CelSeq, cette méthode commence par une rétrotranscription d'un ARNm par une amorce poly T, mais cette dernière est cette fois greffée d'une séquence d'amorçage similaire à celle qui sera insérée en amont par l'activité ≪switching template≫ de la rétrotranscriptase. Cette étape de rétrotranscription est suivie d'une amplification PCR sur base des sites d'amorçage insérés de part et d'autre du cDNA. Ces deux étapes sont communément regroupées sous l'appellation ≪Template-switching polymerase chain reaction≫ (TS-PCR). Les cDNA ainsi amplifiés sont ensuite fragmentés sous l'action de la transposase tn5 qui insère en même temps des amorces Nextera aux extrémités des séquences. Ces amorces permettent l'insertion des adaptateurs de séquençage ILLUMINA aux extrémités des brins après une seconde étape d'amplification PCR.

La technique SmartSeq permet donc d'augmenter la couverture de séquençage sur toute la longueur des transcrits et ainsi d'analyser l'épissage des ARN. En revanche, la double étape de PCR ne fait qu'accentuer l'insertion de bruit de fond technique pour l'étude des proportions de transcrits originels. De plus, l'absence de séquence UMI dans l'amorce de départ ne permet pas de compter les transcrits comme pour la technique CELseq.

Dans le cadre de cette étude, quatre sources de données SmartSeq ont été employées :
[Segerstolpe et al., 2016], [Xin et al., 2016], [Nathan Lawlor et al., 2017],
[M. Enge , H. Efsun Arda , Marco Mignardi , John Beausang , Rita Bottino, K. Kim, 2015].
Deux sources de données ont été utilisées pour le CelSeq : [Muraro et al., 2016], [Baron et al., 2016].

Ces différentes études ont permis de déterminer les signatures transcriptomiques pour chaque type cellulaire, d'évaluer l'hétérogénéité au sein de certains types et de mesurer les différences entre des cellules provenant de personnes saines, diabétiques ou obèses. Néanmoins, ces études se distinguent sur le nombre de gènes marqueurs (voire parfois leurs identités) qui caractérisent les types cellulaires. Que cela soit dû à des différences de ≪treshold≫ appliqués sur les différents programmes employés par les laboratoires ou à des variations des données brutes initiales, ceci soulève la question de la reproductibilité et de la similarité des résultats obtenus par des laboratoires qui n'utilisent pas les mêmes techniques.

## 1.5 Comparaison méthodologique

Diverses méthodologies existent pour exploiter des données issues de sources différentes. Il est possible de procéder par ≪data merging≫, qui consiste en une agglomération des données en un seul jeu (≪dataset≫) dès le départ de l'analyse (Figure 5). Ce système part donc du principe que les données de sources différentes sont scrupuleusement du même ordre. Néanmoins, dans le cas des données RNAseq, cela est rarement le cas : en effet, des différences apparaissent entre les sources de données, ne serait-ce que par le changement de manipulateur ou de matériel utilisé. De plus, des méthodes différentes ont pu être utilisées, comme le CELseq ou Smartseq. Des variations entre des études peuvent donc provenir d'une variation de type ≪technique≫, appelée ≪effet batch≫. Ainsi, lorsqu'on compare des données provenant de laboratoires différents, il est donc important d'analyser l'effet ≪batch≫ afin de s'assurer de la véracité de l'interprétation des résultats. Le procédé de ≪data-merging≫ a donc cet inconvénient majeur de ne pas tenir compte de la variation technique, bien qu'il soit possible d'atténuer ce problème en appliquant des protocoles de corrections d'effet ≪batch≫.

Une autre approche se base sur le principe méta-analytique qui consiste à traiter toutes les sources de données de façon indépendante avec un protocole standardisé. Les résultats issus de chaque analyse sont alors comparés entre eux et permettent de valider les observations globales des ≪datasets≫. Concrètement, cela se résume à comparer les listes de gènes enrichis et les profils transcriptomiques obtenus par les différents laboratoires.

Une méthode intermédiaire entre ces deux approches est applicable. Après l'étape de ≪clustering≫, les cellules appartenant au même type sont aléatoirement séparées pour former 5 échantillons. Les niveaux d'expression de gènes sont alors additionnés au sein de chaque échantillon pour former des réplicas artificiels ≪bulkés≫. A ce niveau de l'analyse, chaque type cellulaire de

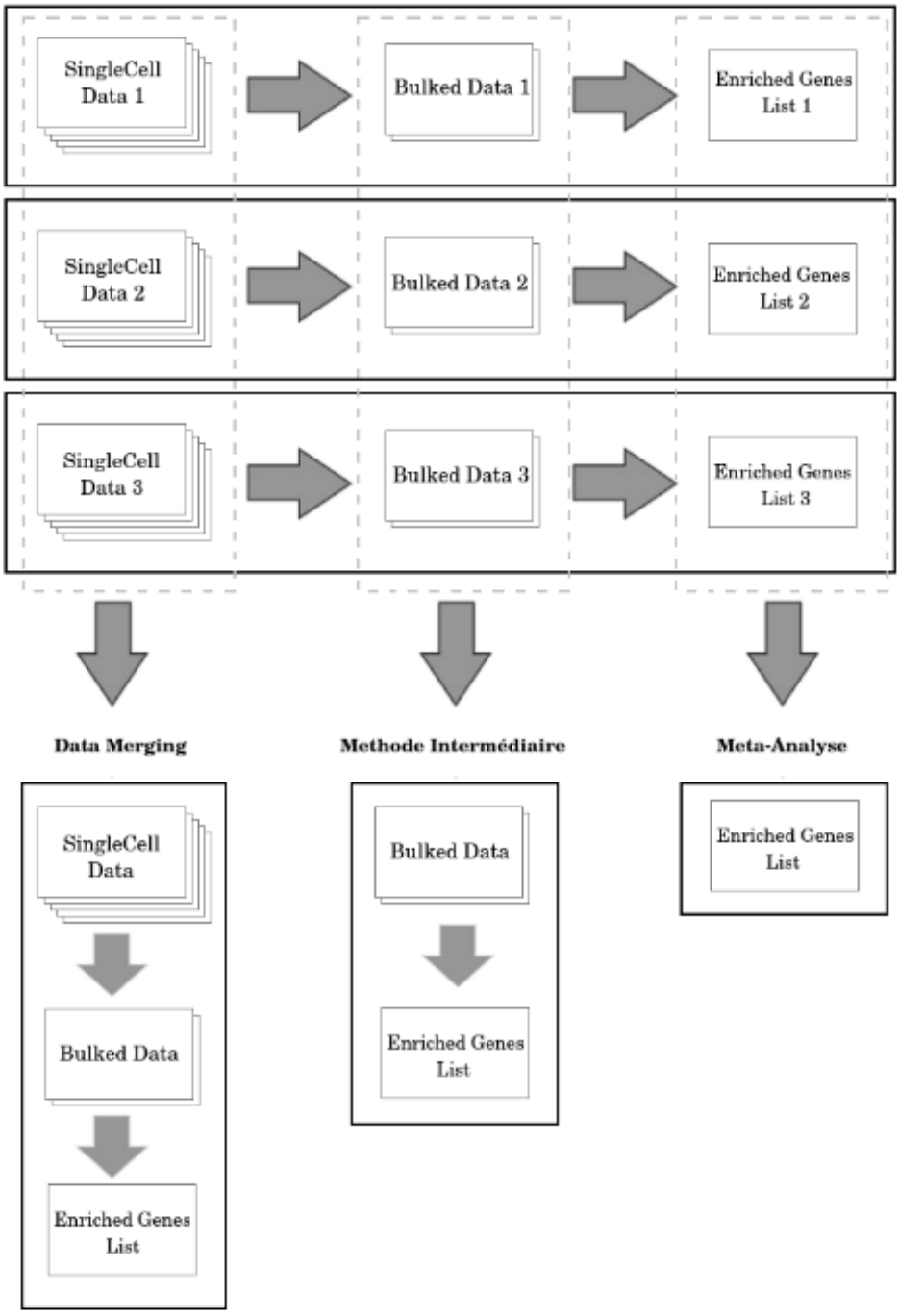


**Figure 5** – *Résumé schématique des méthodes d'exploitation de différentes sources de données.*

chaque étude est alors composé de 5 réplicas qui peuvent être dans ce cas-ci agglomérés avant de procéder à l'analyse des gènes exprimés. L'intérêt de cette méthode intermédiaire réside essentiellement au niveau de la puissance computationnelle nécessaire comparée au «data-merging». En effet, le SingleCell RNAseq fournit une représentation du transcriptome pour chaque cellule.

L'agglomération des données dès le départ de l'analyse revient à processer un fichier avec autant de colonnes qu'il y a de cellules (plus de 15 000) et autant de lignes qu'il y a de gènes exprimés (plus de 18 000). La charge mémoire requise pour cette opération dépasse alors largement les capacités de l'appareil utilisé dans le cadre de ce mémoire.

Ces différentes approches sont utilisées et appliquées pour l'inférence de réseau de régulation génique à partir de données de micro-array [Pham et al., 2017].

## 1.6 Comparaison inter-espèces

Comme énoncé précédemment, des différences structurelles pancréatiques sont perceptibles entre des espèces comme l'homme et la souris [Levetan, C. and Pierce, 2013], mais la comparaison de l'information révélée par des études RNAseq des îlots pancréatiques de différentes espèces peut mettre en évidence les gènes ayant conservés leur expression pancréatique à travers l'évolution. Cette comparaison inter-espèce est intéressante à réaliser car une forte pression de sélection s'exerce sur les gènes possédant une fonction vitale pour l'organisme.

La mise en évidence de schémas d'expression conservés est donc une approche intéressante pour révéler les acteurs indispensables à la mise en place des fonctions tissulaires et à l'identité de cellules spécifiques. Une analyse transcriptomique inter-espèces des cellules pancréatiques de l'homme, de la souris et du poisson zèbre (Danio rerio) a été réalisée en 2017 par le laboratoire ZDDM (≪Zebrafish Development and Disease Models≫) au GIGA à l'Université de Liège [Tarifeño-Saldivia et al., 2017]. Cette étude a permis de révéler des caractéristiques spécifiques ou conservées chez ces différentes espèces. Plusieurs centaines de gènes présentent une expression spécifique pour les cellules endocrines ou pour les cellules exocrines chez ces trois vertébrés, dont notamment les gènes impliqués dans la régulation de la sécrétion hormonale, dans la production d'enzymes digestives ou dans la différenciation cellulaire. D'autre part, et de manière surprenante, les gènes présentant une expression spécifique des sous-types cellulaires endocriniens alphas et bêtas semblent très peu conservés entre les trois espèces, y compris entre l'homme et la souris (Figure 6).

Cette étude est basée sur des données RNAseq classiques (≪bulk RNAseq≫) et dans lesquelles la séparation des types cellulaires peut être non-optimale. Ceci peut éventuellement expliquer le nombre peu élevé de gènes présentant une expression conservée et enrichie dans les cellules alphas et bêtas. Les données scRNAseq pancréatiques publiées ces 2 dernières années permettent d'obtenir le transcriptome de chaque type cellulaire avec une grande précision et pureté. Ce type

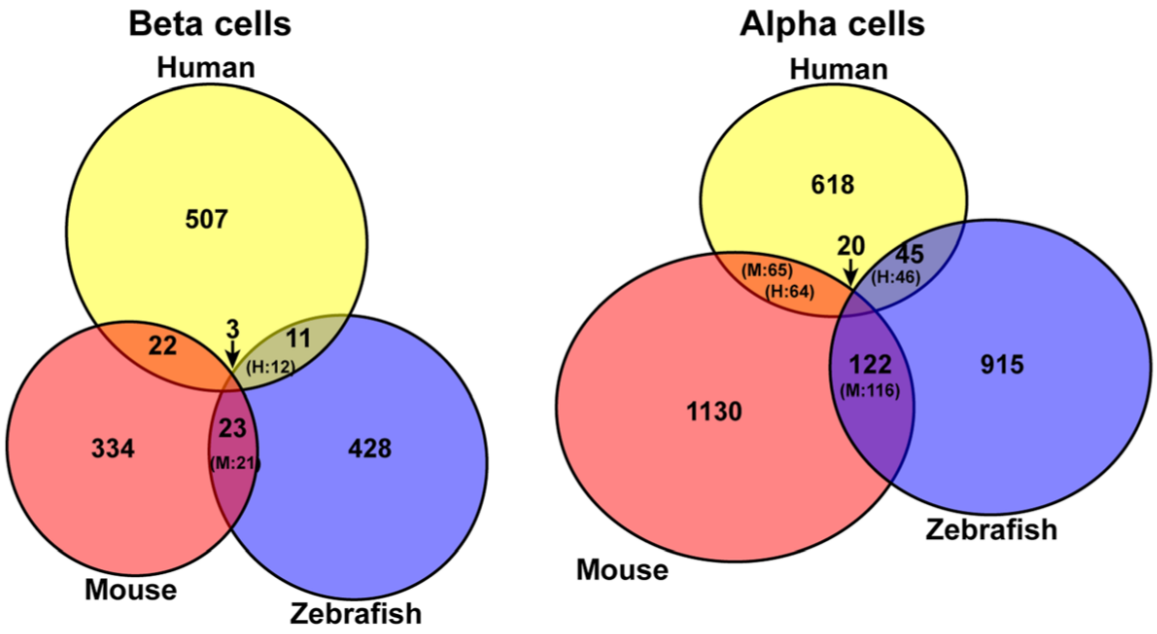


**Figure 6** – *Comparaison des gènes enrichis entre l'homme, la souris et le poisson zèbre pour les cellules bêtas et alphas.*

de données permet donc de réaliser une analyse inter-espèce optimale et d'identifier les gènes ayant conservé un profil d'expression pancréatique durant l'évolution.

## 1.7 Objectifs

Ce mémoire vise à atteindre plusieurs objectifs. Dans un premier temps, les différentes études scRNAseq seront comparées afin de vérifier leurs reproductibilités et d'estimer les variations techniques qui les distinguent. L'impact d'une correction d'effet ≪batch≫ par différents algorithmes sera par ailleurs évalué avec pour but d'exploiter tous les jeux de données en même temps.

Dans un second temps, les données scRNAseq seront comparées aux données RNAseq ≪bulk≫ disponibles pour certains types cellulaires pancréatiques afin d'évaluer la qualité des études antérieures.

Enfin, les conclusions de [Tarifeño-Saldivia et al., 2017] devront être confirmées et l'analyse sera étendue à davantage de types cellulaires. Les nouvelles données devraient permettre de visualiser un éventuel apport d'informations pour les types cellulaires déjà étudiés, et offrir de nouvelles observations pour les types cellulaires n'ayant pas encore fait l'objet de comparaison.

# 2 Méthodes

## 2.1 Analyse des données RNAseq classiques («Pipeline Bulk»)

Les données RNAseq «Bulk» employées dans ce mémoire sont les mêmes que celles utilisées par Tarifeno et al. Pour les données RNAseq des cellules alphas et bêtas humaines, les résultats de [Blodgett et al., 2015] ont été récupérés. Concernant la souris, ce sont les données de [DiGruccio et al., 2016] (alpha, bêta et delta) qui ont été rapatriées. Et enfin, pour le Zebrafish, ce sont les données générées par le laboratoire d'accueil (ZDDM) qui ont été analysées.

Les données de Blodgett et DiGrucio sont disponibles sur le site du NCBI sous les numéros d'accession GSE67543 et GSE80673 respectivement. L'outil sraToolkit fourni par ce site permet de télécharger les données de format SRA en les convertissant au format fastq (directement compressible en gz avec l'option –gzip) et en les séparant directement en 2 fichiers de «reads» en cas de séquençage «paired-end» (option – split-files).

Les données ainsi obtenues correspondent aux résultats bruts du séquençage. Autrement dit, il ne s'agit ici que de «reads» de cDNA amplifiés (séquences partielles de cDNA) obtenus après rétro-transcription d'ARN purifié. Pour obtenir l'information transcriptomique d'intérêt, il est nécessaire d'aligner ces «reads» contre un génome de référence en vue d'obtenir un fichier de comptage de «reads» cartographiés par gène («Reads Count»). Pour ce faire, le programme d'alignement STAR [Dobin et al., 2013] a été utilisé dans cette étude. Outre le simple fait qu'il soit largement employé par la communauté scientifique, cet aligneur présente l'avantage d'être rapide tout en offrant un panel d'options intéressant, avec notamment la possibilité de générer directement un fichier de Reads Count. En revanche, cet aligneur nécessite davantage de mémoire vive que ses concurrents. Un minimum de 30 Go est nécessaire pour l'indexation du génome humain d'après les développeurs. Néanmoins, même une disponibilité de 32 Go n'a pas été suffisante pour réaliser cette opération avec le génome humain de référence GRCh38.p10 d'ENSEMBL. Il s'est donc avéré nécessaire d'accéder à un serveur de calcul aux capacités plus importantes et le système Durandal du groupe PHYTOSYSTEMS de l'Université de Liège a pu être utilisé. Il a également été employé pour les étapes de filtrage des «reads» en pré-alignement. En effet, il est conseillé de filtrer les données afin d'éliminer des «reads» ne répondant pas aux critères de qualité de base (longueur de la séquence, code de fiabilité post-séquençage, etc.), et ce afin d'assurer une meilleure qualité du mapping.

Le programme Trimgalor [[F., 2012] F., ] a été utilisé à cet effet dans le cadre de cette étude. Il permet d'exploiter facilement le système Cutadapt afin de retirer les reads de faible qualité,

que cela soit en single-end ou en paired-end, mais également d'éliminer les tronçons de séquences correspondant à des résidus d'adaptateur de séquençage ILLUMINA. De plus, ce programme est en mesure d'effectuer une analyse systématique du fichier de reads en fournissant en sortie un rapport sur la qualité et le contenu en exploitant le système Fastqc [Andrews, 2013] (Figure 7).

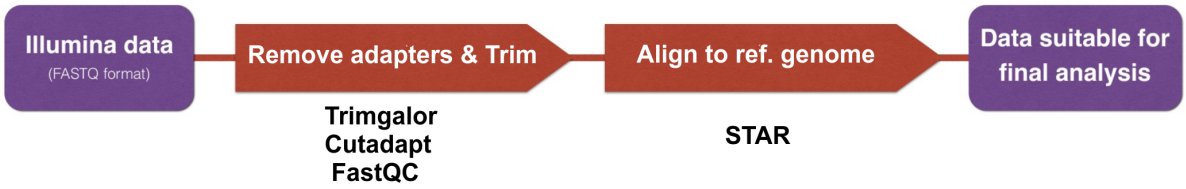


**Figure 7** – *Schéma du pipeline standard pour l'alignement.*

Les rapports des processus effectués par les différents programmes cités précédemment peuvent être résumés en un seul fichier html via le programme MultiQC [Ewels et al., 2016] qui fournit ainsi une carte d'identité de nos données.

L'analyse de l'expression différentielle à partir des fichiers de comptage de reads est réalisée avec Deseq2 [Love et al., 2014]. Avec EdgeR, il fait partie des packages R les plus communément utilisés par la communauté scientifique pour le traitement de données RNAseq. Le choix de son utilisation se défend par son efficacité comparée aux programmes similaires ([Seyednasrollah et al., 2013] et [Maza, 2016]).

Deseq2 permet l'analyse en «pairwise» de données de «Reads Counts» (associées en une matrice de comptage) sur base d'un modèle linéaire généralisé (GLM) d'une distribution binomiale négative en passant par une étape de normalisation. Le package SARTools [Varet et al., 2016] exploite l'environnement des objets R de Deseq2 afin d'offrir des outils de visualisation de données, et génère également un rapport global sur les données transcriptomiques. Avec MultiQc, ce système permet donc d'obtenir une bonne vue d'ensemble des données disponibles.

## 2.2 Analyse des données scRNAseq («Pipeline SingleCell»)

Les données SingleCell sont disponibles sous deux formes distinctes. Soit les reads ont été triés avant la mise en ligne des données, auquel cas il y aura un fichier SRA de «reads single-end» disponible par cellule, soit les reads n'ont pas été triés avant la mise en ligne. Dans ce cas, seul quelques fichiers SRA de grande taille composés de «reads paired-end» sont disponibles. Ces fichiers sont accompagnés de la liste des code-barres utilisés pour marquer les librairies produites

à partir des différentes cellules et rassemblées en un seul échantillon pour le séquençage. Dans ces fichiers, le read1 contient le code-barre de la cellule analysée et le read2 consiste à la séquence du cDNA identifié. Il est donc nécessaire de trier les reads2 du paired-end selon le code-barre porté par les reads1 afin de générer un fichier de séquences par cellule identifiée (Figure 8). Dans tous les cas, l'outil sraToolkit a été employé pour importer et convertir les données au format fastq.

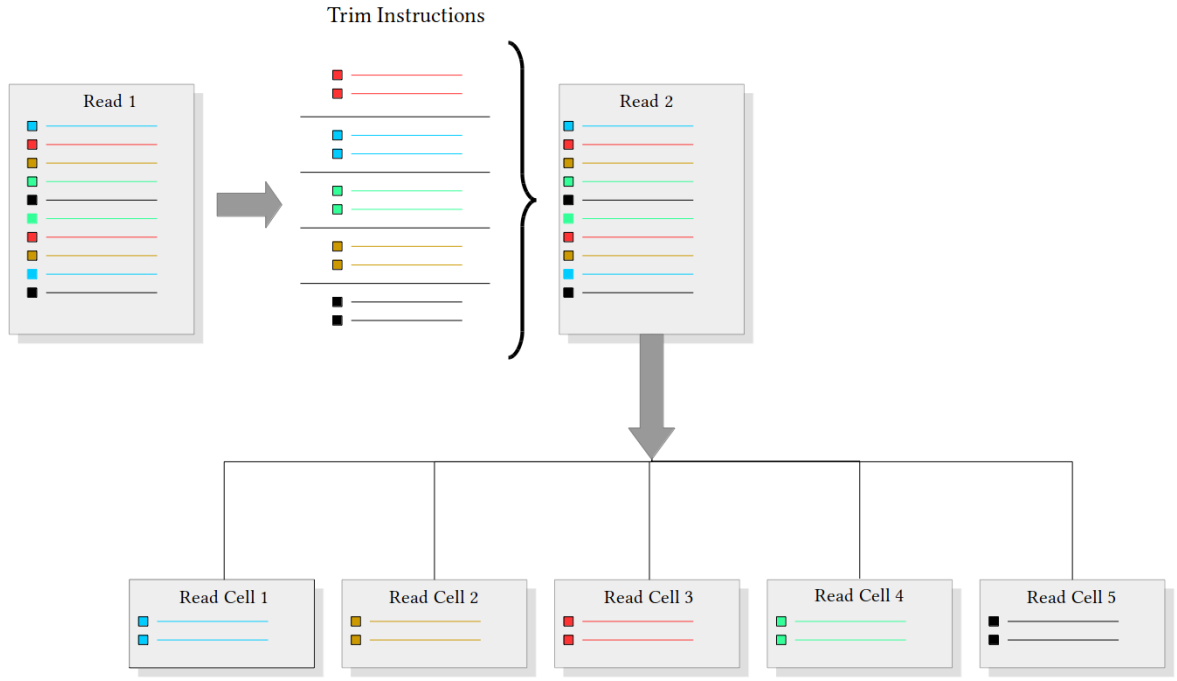


**Figure 8** – *Illustration du procédé de tri des reads selon la reconnaissance des codes-barres.*

La plupart des programmes permettant de réaliser le tri des reads sont fournis avec l'équipement de séquençage et sont donc sous licences payantes. Ces programmes, en plus des systèmes libres, nécessitent généralement des fichiers additionnels dans des formats particuliers pour être utilisables. Ces fichiers ne sont pas disponibles sur les sites de dépôt et ne sont pas générables sur base des informations disponibles. Il s'est donc avéré nécessaire de développer un programme dédié au tri des reads selon leur origine définie par les code-barres.

J'ai donc créé le script perl ScTrimReads qui prend en entrée une liste de code-barres, un fichier de reads porteur de code-barres (fichier fastq de premiers reads du paired end) et un fichier de reads portant l'information sur la séquence des transcrits (fichier fastq de seconds reads du paired end). En sortie, ce programme génère un fichier fastq de second reads par cellule identifiée. Ce système possède également une option permettant de paramétrer le nombre de

≪mismatch≫ autorisé lors de la reconnaissance des code-barres. Cette permissivité peut s'avérer essentielle pour obtenir une proportion de reads identifiés satisfaisante pour certaines sources de données. Cependant, cette option complexifie légèrement la recherche de code-barres dans un délai raisonnable et force le stockage des seconds reads identifiés dans la mémoire vive avant de les exporter dans un fichier de sortie. Dès lors, il peut être nécessaire de morceler la liste de code-barres et procéder en plusieurs étapes afin de ne pas saturer la ≪ram≫.

Comme pour la méthode ≪bulk≫, chaque fichier de séquences est trié selon la qualité des reads avec le programme Trimgalor et aligné contre un génome de référence avec le programme STAR. A terme, on obtient un fichier de comptage de ≪reads par gène≫ pour chaque cellule disponible.

A ce stade, les différents types cellulaires en présence ne sont pas encore identifiés. Autrement dit, il n'est pas possible de savoir à quel type cellulaire se rapportent nos différents transcriptomes. Le système RaceID développé par [Grün and Van Oudenaarden, 2015] utilise le ≪clustering≫ par ≪kmeans≫ ou ≪kmedoids≫ avec des corrélations de spearman ou de pearson pour produire une représentation à deux dimensions des types cellulaires à l'aide d'un graphique t-SNE (t-distributed stochastic neighbor embedding) (Figure 9). Concrètement, ce système a pour but d'identifier les différents groupes de cellules présentant des similarités notables au sein de leurs transcriptomes. Pour ce faire, les fichiers de comptage de reads par gène sont associés en une matrice de comptage et l'ensemble est soumis à une étape de filtration afin d'éliminer les transcriptomes avec trop peu de reads ainsi que les gènes avec un compte nul ou proche de zéro trop fréquent. Ce tri permet de réduire la taille de la matrice de comptage en se focalisant sur l'information significative présente dans chaque transcriptome.

Dans le cadre de cette étude, les transcriptomes avec moins de 10.000 reads ont été écartés et les gènes avec un compte inférieur à 5 pour plus de 20 cellules ont également été retirés. Le ≪clustering≫ a été réalisé par la méthode kmedoid génèrant au moins 8 clusters attendus.

La représentation graphique t-SNE permet de visualiser les différents clusters identifiés mais permet également, à l'image d'un heatmap, de mettre en évidence le niveau d'expression d'un gène d'intérêt. Il est donc possible d'identifier les populations de cellules selon leur niveau d'expression de transcrit lié à un gène marqueur connu. Par exemple, le cluster des cellules bêtas devrait présenter une expression de transcrit du gène de l'insuline largement supérieure aux autres types cellulaires. Grâce à ce système, il est donc possible d'identifier le type cellulaire propre à chaque cellule disponible.

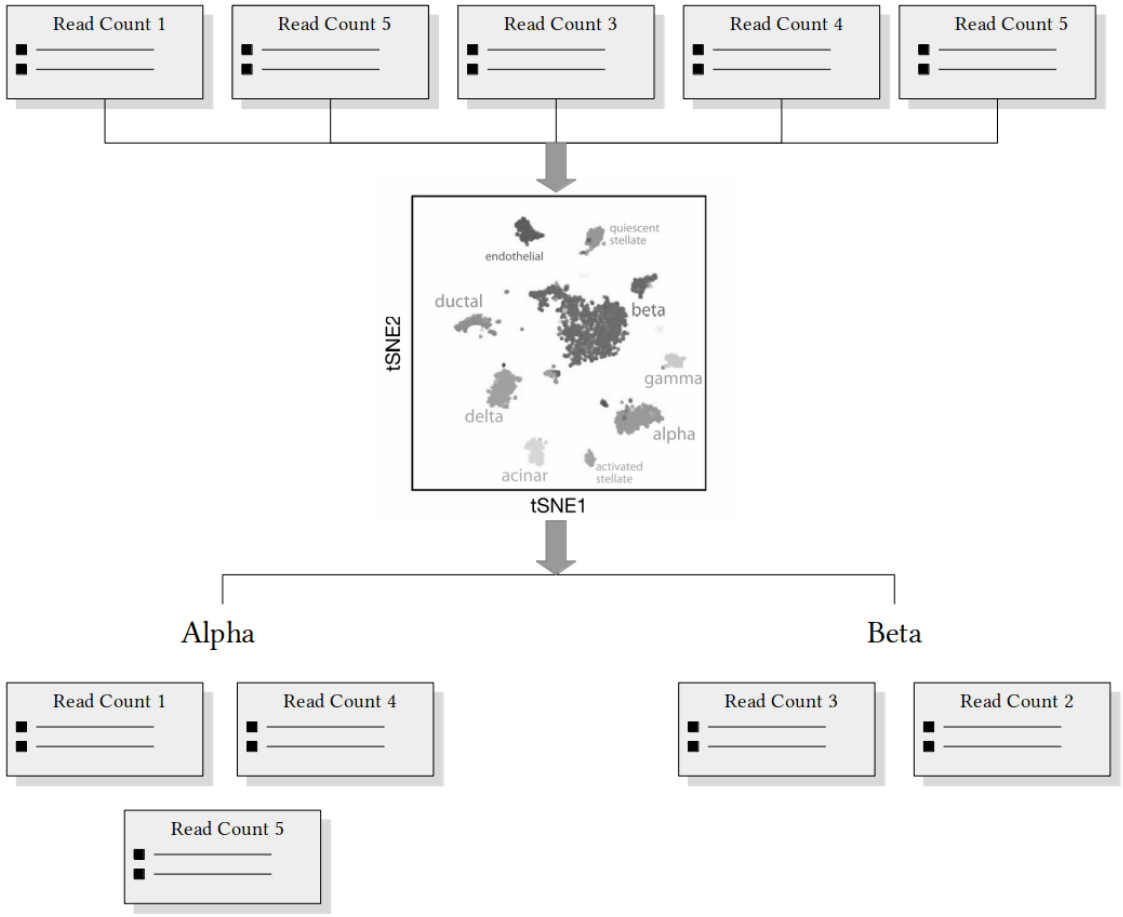


**Figure 9** – *Illustration du procédé de clustering des profils transcriptomiques des cellules*

Il s'ensuit une analyse des gènes enrichis pour chaque type cellulaire avec Deseq2 comme pour la méthode «bulk». Néanmoins, on considère que la représentativité du type cellulaire pour une cellule unique est limitée, étant donné que seul 10 à 20 % des ARNm sont capturés lors de la production des librairies. Il est donc nécessaire d'associer l'information portée par plusieurs cellules pour assurer une couverture suffisamment représentative d'un type cellulaire. De plus, comme décrit par Baron et al,., une grande quantité de cellules est compliquée à analyser si l'on considère chaque cellule comme un réplica. Nous avons donc généré au moins 5 réplicas pour chaque type cellulaire en regroupant les transcriptomes d'au moins 15 cellules..

Etant donné que le système Deseq2 et les outils de correction d'effet batch sont basés sur des modèles linéaires, un nombre important de réplicas est recommandé pour rendre les résultats plus robustes. Comme pour le «pipeline bulk», le package SARTools permet de visualiser les caractéristiques des données ainsi générées.

## 2.3 Identification des gènes présentant une expression enrichie dans chaque type cellulaire

Le système Deseq2 permet d'évaluer l'expression enrichie de certains gènes en comparant deux conditions différentes (mutant versus sauvage par exemple). Dans notre cas, nous disposons de plus de deux conditions, ce qui augmente le nombre de comparaisons à réaliser pour obtenir une liste de gènes enrichis spécifique à un type cellulaire. Deseq2 est incapable de croiser ces différentes comparaisons, il s'est donc avéré nécessaire de développer un script R capable d'exécuter cette tâche.

J'ai créé le script EGMA (Enriched Gene Multifactorial Analyser)(voir annexe) qui permet de générer automatiquement toutes les évaluations de surexpression en ≪pairwise≫ pour tous les types cellulaires et de croiser ces résultats afin de générer une liste de gènes avec une expression enrichie dans chaque type cellulaire. Par exemple, toutes les comparaisons possibles du transcriptome de cellules alphas avec les autres types cellulaires génèreront 5 listes de gènes enrichis. Le croisement de ces listes permet de ne garder que les gènes communs entre elles et qui, autrement dit, présentent une surexpression systématiquement significative dans les cellules alphas (Figure 10). Des options de ≪threshold≫ sont disponibles sur la ≪p-adjust≫, qui permet de sélectionner la significativité de la surexpression, et sur le Log Fold Enrichment qui permet de sélectionner le niveau de surexpression minimum désiré. Le script EGMA possède également une fonction de traduction qui permet de convertir les identifiants de gènes en leurs dénominations communes afin d'améliorer la lisibilité des listes finales.

## 2.4 Comparaison inter-études

Dans le cadre de ce mémoire, nous disposons d'un grand nombre de sources de données différentes avec d'une part les données RNAseq de type ≪bulk≫ et d'autre part les données ≪SingleCell≫. Il est donc intéressant de voir dans quelle mesure des données ≪SingleCell≫ confirment les résultats obtenus à partir de données ≪Bulk≫. De plus, les données SingleCell humaines proviennent de différents laboratoires utilisant éventuellement des techniques différentes (CelSeq ou SmartSeq). Il est donc également intéressant de voir quelle est la variabilité transcriptomique entre ces données issues de laboratoires différents. Toutes ces données peuvent être comparées de deux façons sur un principe méta-analytique.

Tout d'abord, il est possible de comparer les listes de gènes enrichis générées par le pipeline

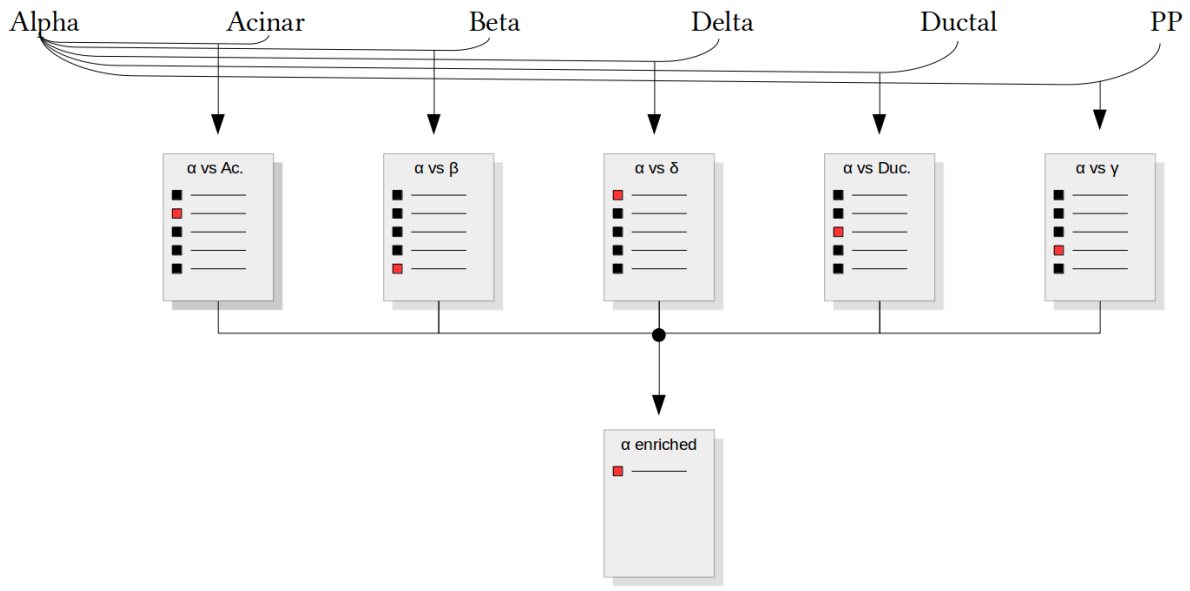


**Figure 10** – *Illustration du procédé de construction des listes de gènes enrichis pour le type cellulaire alpha.*

d'analyse SingleCell standardisé appliqué sur chaque étude prise séparément. La comparaison des listes de gènes pour chaque étude peut être visualisées avec des diagrammes de Venn et quantifier avec des indices de Jackard. Nous avons ajouté au script R EGMA une fonction de comparaison de listes de gènes enrichis de façon à générer automatiquement les diagrammes de Venn pour chaque type cellulaire. Les indices de Jackard sont, quant à eux, obtenus grâce au package R GeneOverlap et le système Gorilla a été employé pour les analyses de Gene Ontology. La seconde méthode de comparaison se base sur l'analyse des données transcriptomiques brutes à l'aide d'analyses par composant principal (PCA), de heatmap de corrélation et de dendrogramme. Ces trois outils d'analyse ont été implantés dans le script EGMA à partir de fonctions modifiées et optimisées provenant de différents packages (SARTools, heatmap2,...).

Le mode de corrélation utilisé est le Simple Error Ratio Estimate (SERE). Il ne s'agit pas d'un calcul de corrélation au sens strict mais plutôt d'un indice de divergence entre les conditions. Cet outil décrit par [Schulze et al., 2012] fait l'hypothèse nulle que les différentes conditions sont fortement similaires contrairement aux méthodes de corrélation conventionnelles qui partent du principe que les conditions sont indépendantes. Le SERE paraît plus sensible et offre des résultats plus interprétables que ses concurrents. Le score qu'il génère est proportionnel aux différences entre les conditions. Plus ce score sera élevé, plus les conditions seront donc différentes.

Des effets «batch» sont à prévoir entre des données de sources différentes et peuvent être

atténués en utilisant le package R Combat [Leek et al., 2012].

## 2.5 Comparaison inter-espèces

La comparaison des résultats entre les espèces nécessite la génération d'un tableau indiquant tous les gènes orthologues pour les trois espèces utilisées (homme, souris et zebrafish)

Une complication apparaît lors de la création de tables d'orthologie à partir de l'outil en ligne biomart d'ENSEMBL. Si ce système fonctionne bien pour générer une liste de gènes orthologues entre deux espèces, il semblerait que des liens erronés se glissent dans la construction de tables à trois espèces. Une table plus fiable a donc été créée au laboratoire sur base d'une comparaison des trois tables d'orthologie pairwise de biomart (homme-zebrafish vs homme-souris vs souris-zebrafish).

# 3 Résultats

## 3.1 Analyse des données SingleCell RNAseq

Plusieurs études scRNAseq ont été publiées ces deux dernères années décrivant le transcriptome des différentes cellules pancréatiques. Nous avons sélectionné 6 études afin de comparer leur données et de réaliser la comparaison transcriptomique interespèce.

Les données sources de Muraro, Enge et Baron ont nécessité un tri des reads pour générer un fichier de séquences par cellule. Les pourcentages d'attribution des reads pour ces différentes études sont respectivement de 95 %, 98 % et 54 % ; ces pourcentages indiquent la proportion de reads initiale qui ont pu être attribués à une cellule grâce à la reconnaissance d'un code-barres. Les données de Segerstolpe, Lawlor et Xin étaient déjà disponibles sous forme d'un fichier disponible par cellule. La table 1 présente un résumé des données disponibles pour ces études. Les données de Xin et Lawlor sont celles qui offrent le moins de cellules (inférieur à 1000). Les données de Baron en revanche, malgré un taux d'attribution de reads limité, offrent le plus grand nombre de cellules (9133 cellules) (Table 1).

| Source | Sc Type | Start Cell | Trimed Cell | Mean Read Count % | Trimed Genes | Cell Type |
|---|---|---|---|---|---|---|
| Baron Human | CellSeq | 9133 | 7501 | 149 492 | 12 845 | 6 |
| Baron Mouse | CellSeq | 1768 | 1694 | 151 229 | 8 199 | 5 |
| Enge | SmartSeq | 1805 | 1805 | 785 212 | 8 390 | 6 |
| Lawlor | SmartSeq | 478 | 478 | 1 647 658 | 4 867 | 5 |
| Muraro | CellSeq | 3022 | 2356 | 112 255 | 3 407 | 6 |
| Segerstolpe | SmartSeq | 1096 | 1096 | 577 036 | 8 390 | 6 |
| Xin | SmartSeq | 651 | 651 | 1 151 275 | 13 551 | 0 |

**Table 1** – *Description générale des données issues de différents laboratoires.*

Les cellules présentant moins de 10.000 reads ont été éliminées des analyses. Seules les données de type CelSeq semblent impactées par cette restriction. De plus, le nombre moyen de reads par cellule est nettement moins important dans les études CellSeq que dans les études SmartSeq. Ce constat n'est guère étonnant au regard des différents modes opératoires. Le SmartSeq fait en effet appel à deux étapes de PCR contre une seule pour le CelSeq lors de la préparation des librairies ; de plus, le Smartseq génère plusieurs reads par transcrit alors que le CelSeq ne génère qu'un seul read par transcrit.

L'élimination des gènes présentant un compte inférieur à 5 pour un groupe d'au moins 15 cellules laisse un nombre de gènes valides assez variable entre les études sans liens discernables selon la technique.

L'analyse de clustering par k-medoid et t-SNE permet de relativement bien séparer les différentes populations cellulaires dans certaines études. Cette étape est cruciale pour la suite des analyses car un mauvais clustering peut provoquer une contamination d'un groupe cellulaire avec un autre et, par la suite, complètement biaiser la représentation transcriptomique des types cellulaires avec un impact non négligeable sur les listes de gènes enrichis finales. A titre d'illustration, la présence de 4 cellules epsilon (qui sont par ailleurs les plus difficiles à isoler de par leur faible proportion) dans un cluster bêta suffit à placer le gène de la ghréline en tête de liste de gènes enrichis pour le type cellulaire bêta.

Une première étape de clustering permet généralement de distinguer un pool de cellules endocrines ainsi qu'un pool de cellules exocrines. Les cellules exocrines peuvent par ailleurs déjà se distinguer en différents clusters à cette étape avec les cellules ductales, acinaires et parfois un mélange de cellules mésenchymateuses, nerveuses et quelques macrophages (Figure 11).

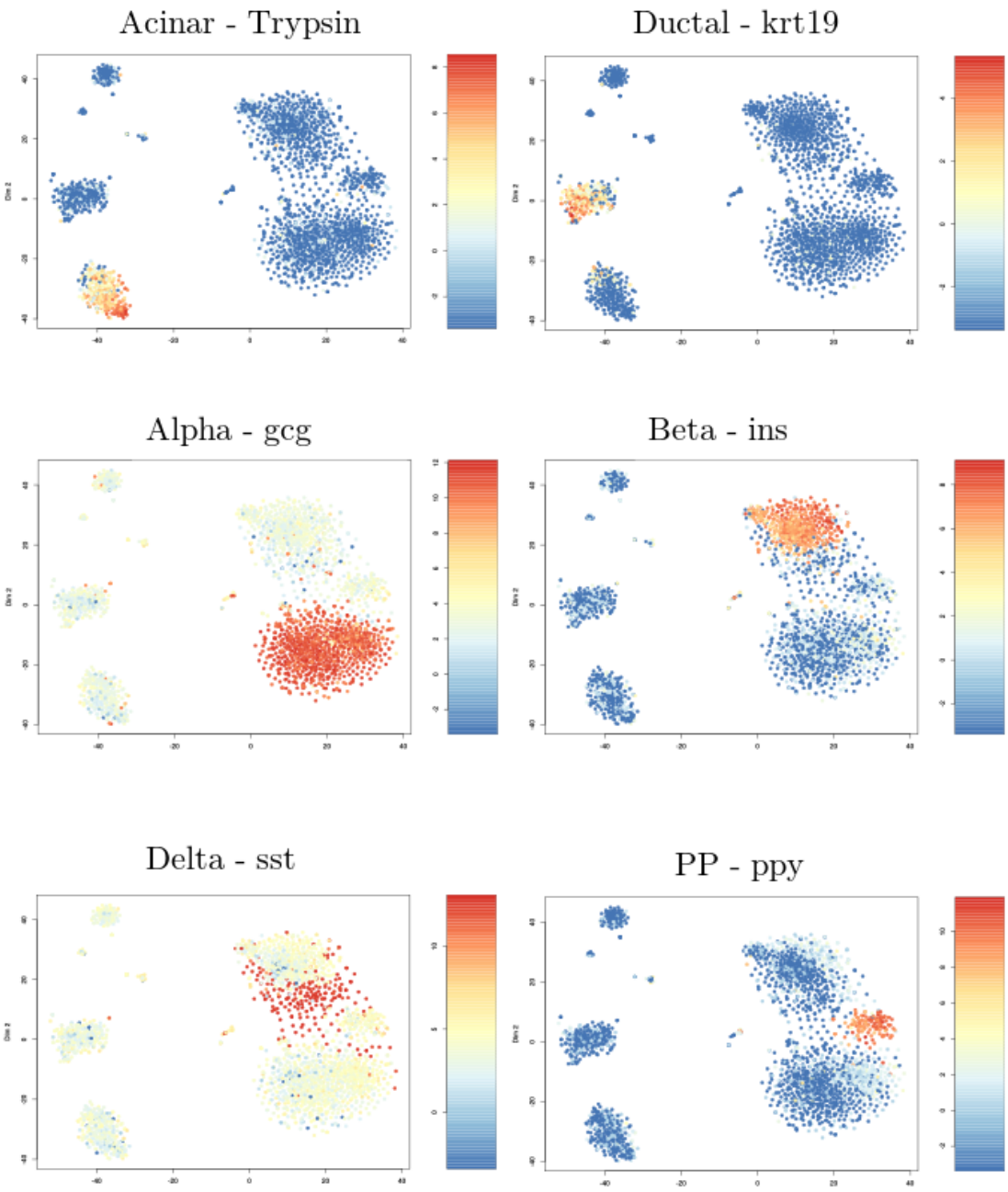


**Figure 11** – *Clustering général des données de Muraro avec mise en évidence des différentes populations cellulaires via le niveau d'expression de gènes marqueurs spécifiques. Le pool endocrine est bien délimité à droite. On retrouve sur la gauche les clusters Ductal et Acinaire ainsi que des cellules de structure dans le coin supérieur gauche.*

Un second clustering sur les cellules endocrines suivi d'un clustering sur les cellules exocrines permet de bien isoler chaque type de cellules principales (Figure 12).

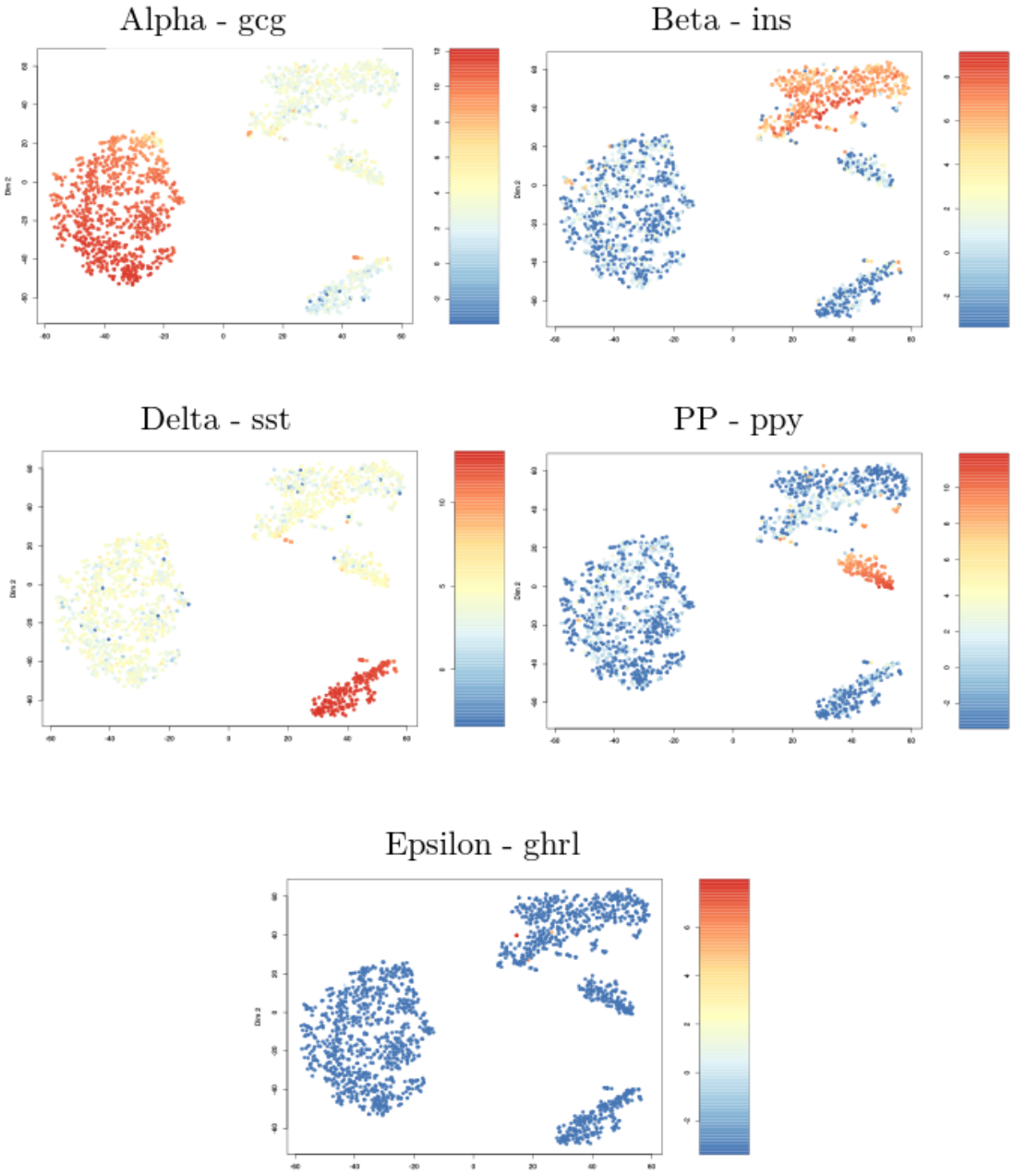


**Figure 12** – *Clustering sur le pool endocrine des données de Muraro avec mise en évidence des différentes populations cellulaires via le niveau d'expression de gènes marqueurs spécifiques. Tous les types cellulaires sont correctement isolés et identifiés.*

De tels résultats sont obtenus pour les données de Muraro et Baron (CellSeq) ainsi que pour les données de Segerstolpe ( SmartSeq) et peuvent être considérés comme optimaux. Les autres sources de données en revanche offrent des résultats sensiblement différents.

Les données de Xin présentent une piètre qualité de clustering ainsi qu'un enrichissement

massif de cellules alphas et bêtas avec une quasi-absence des autres types cellulaires (Figure 13).

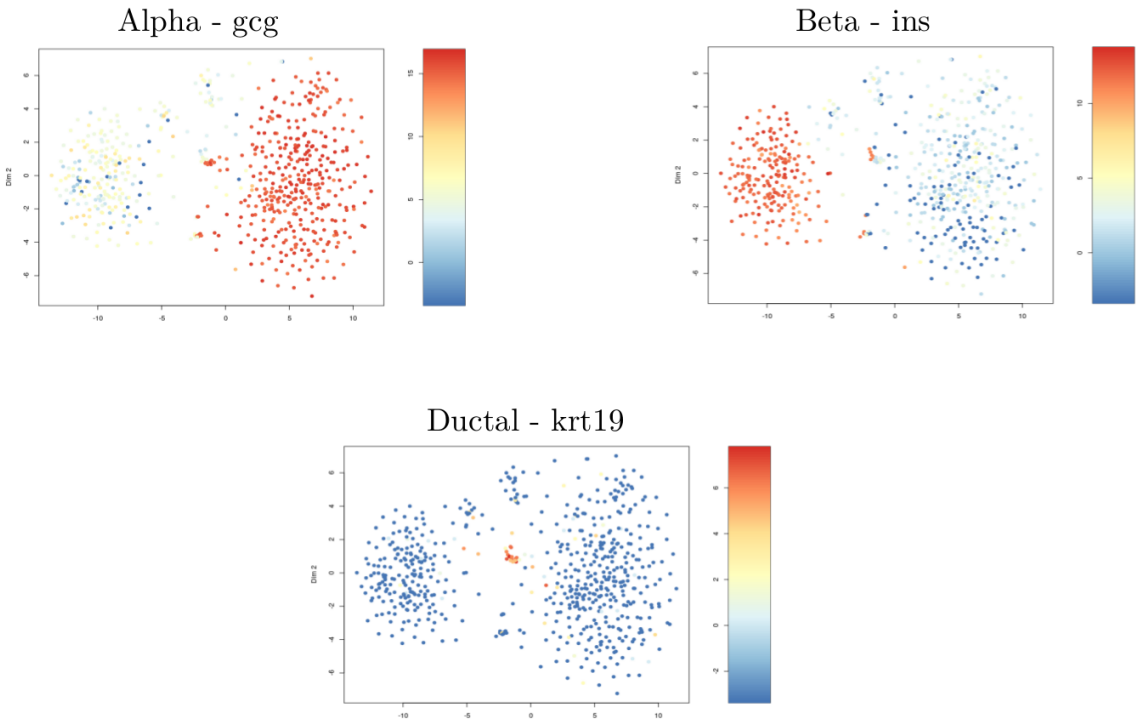


**Figure 13** – *Clustering des données de Xin avec mise en évidence des différentes populations cellulaires via le niveau d'expression de gènes marqueurs spécifiques. Seul les types cellulaires alpha et bêta ainsi que quelques cellules ductales sont décelables.*

Les données de Lawlor offrent également un clustering de qualité limité où les cellules bêta, ductales et PP tendent à se mélanger. Les cellules alphas, deltas et acinaires se distinguent correctement en revanche. On peut également noter une très faible disponibilité en cellules ductales et PP.

Concernant les données de Enge, le clustering semble relativement similaire aux données de Muraro au premier regard. Néanmoins on peut voir que les cellules ductales, acinaires, alphas et bêtas se détachent en 3 sous-clusters distincts. On pourrait s'attendre à ce genre de phénomène pour un type cellulaire selon le stade des cellules dans le cycle cellulaire ou selon des états physiologiques différents, mais retrouver cette observation pour autant de types cellulaires différents ressemble davantage à un effet «donneur» (Figure 14).

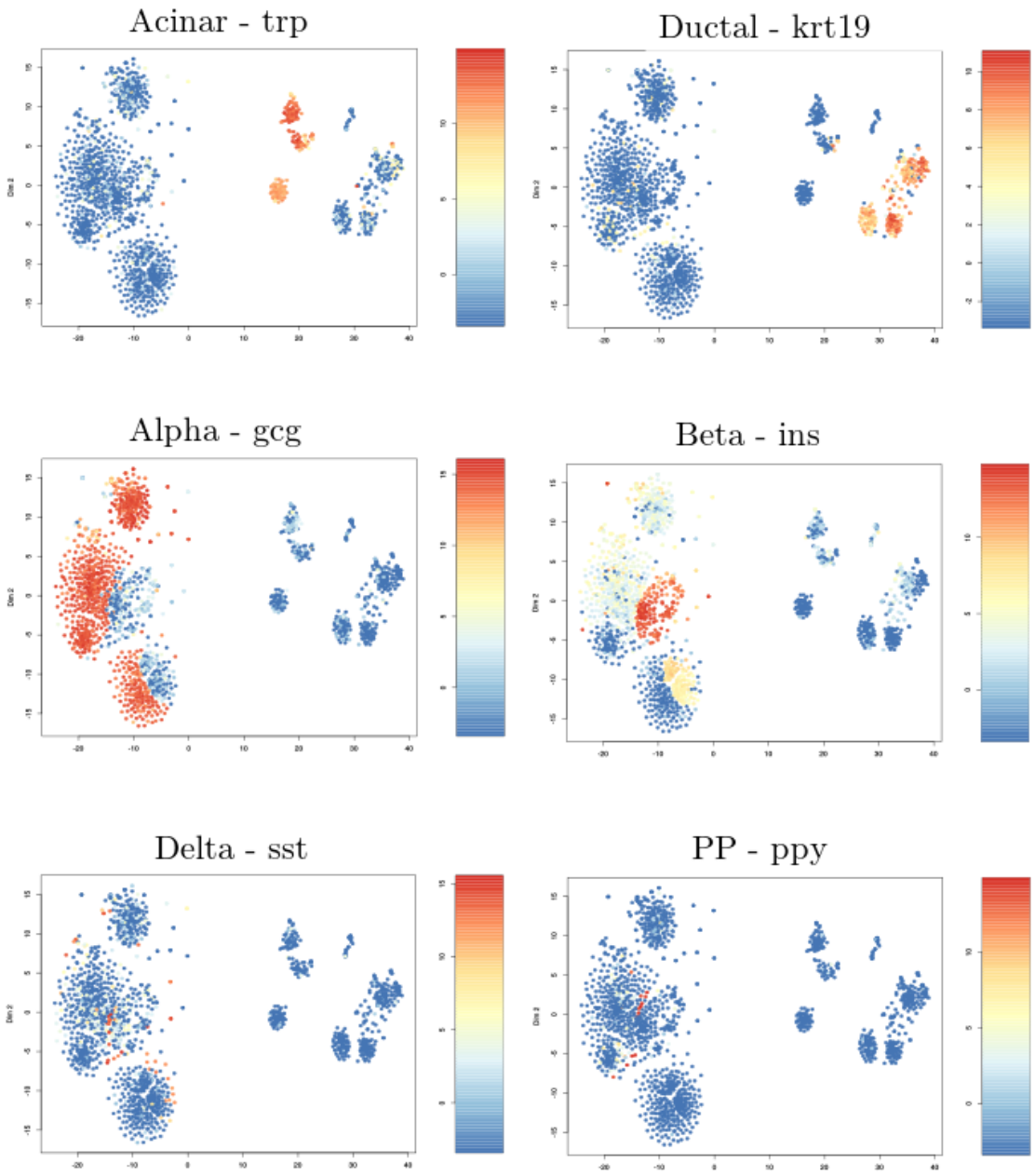


**Figure 14** – *Clustering générale des données de Enge avec mise en évidence des différentes populations cellulaires via le niveau d'expression de gènes marqueurs spécifiques. Le pool endocrine est bien délimité à gauche. On retrouve sur la droite les clusters Ductal et Acinaire ainsi que des cellules de structure.*

L'effet donneur consiste en la formation de plusieurs clusters pour un même type cellulaire provenant de différences entre les individus dont on a isolé les cellules pancréatiques. Idéalement, cet effet doit être très peu marqué, comme pour les études de Muraro, Baron et Segerstolpe, et est sans doute essentiellement dû à des différences de manipulations des échantillons en laboratoire

(temps de conservation sur glace, différents manipulateurs, analyses en plusieurs runs,...). La mise en évidence de l'origine des cellules permet de voir que les sous-clusteurs sont séparés selon les 6 donneurs. Un mélange homogène des couleurs associées aux différents donneurs pour chaque type cellulaire est effectivement absent (Figure 15).

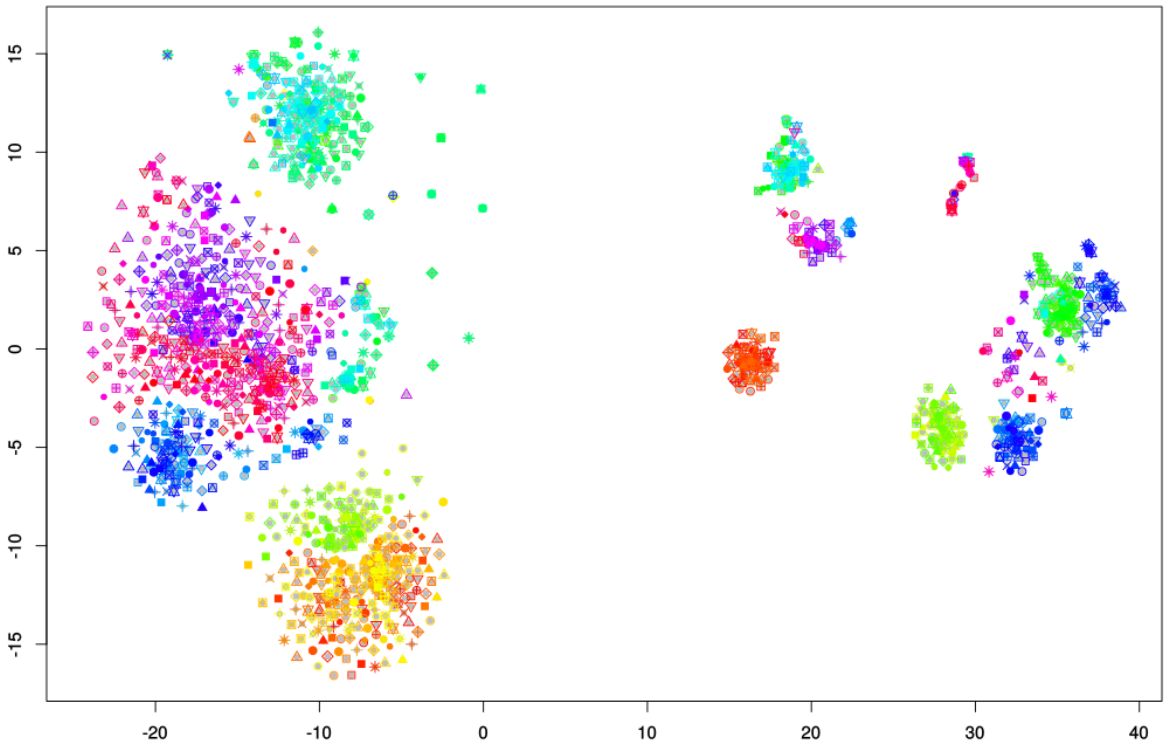


**Figure 15** – *Clustering général des données de Enge avec une mise en évidence de l'origine des cellules selon les individus donneurs. Chaque couleur est associée à un donneur.*

Le zoom de clustering sur les cellules endocrines présente une qualité limitée avec des cellules bêtas, PP et deltas qui se distinguent mal. De plus, la quantité de cellules PP et delta est extrêmement faible (Figure-Supplémentaire 1). Sur base de la qualité du clustering des différents types cellulaires, nous avons décidé de continuer l'analyse des transcriptomes cellulaires pour les données de Segerstolpe, Muraro et de Baron.

## 3.2 Comparaison inter-études

La comparaison inter-étude se fait sur un principe méta-analytique avec une comparaison des profils transcriptomiques et des listes de gènes enrichis dans le but de vérifier la variabilité des données de différents laboratoires utilisant diverses techniques

### 3.2.1 Analyse des profils transcriptomiques SingleCell RNAseq humains

Le but de l'étape suivante était de déterminer le degré de similarité des données transcriptomiques obtenues pour les 3 études scRNAseq sélectionnées. Nous avons additionné les données RNAseq pour au moins 15 cellules du même type cellulaire afin d'obtenir in silico plusieurs réplicas de données RNAseq de type «bulk» pour chaque type cellulaire.

Toutes ces données ont ensuite été comparées par une analyse de composant principal. Les différences entre les études de Baron, Muraro et Segerstolpe sur base de l'analyse par composant principal sont limitées à la seconde variable de l'analyse (Figure 16, axe PC2, 12% de la variance) . La première variable reste liée à la distinction entre les catégories cellulaires endocrine et exocrine (Figure 14, axe PC1, 42% de variance). Comme nous pouvions nous y attendre, les deux études CellSeq (Baron et Muraro) sont plus proches entre elles. L'étude SmartSeq de Segerstolpe se distingue assez clairement des études CellSeq.

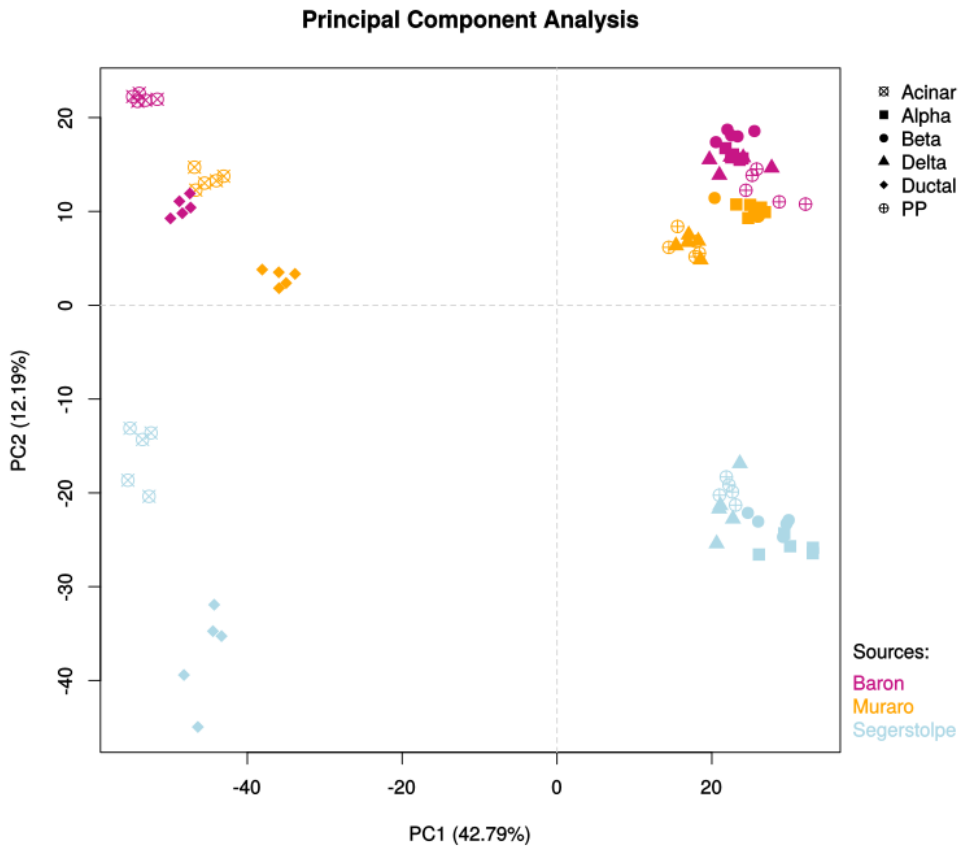


**Figure 16** – *PCA sur les données SingleCell RNAseq de Baron, Muraro et Segerstolpe à partir des réplicas artificiels générés lors du clustering.*

L'utilisation du package Combat pour la correction d'effet batch permet de réduire la distance

entre les études sur la seconde variable (Figure 17). La similarité entre les trois études semble largement plus appréciable. La focalisation de l'analyse sur le cluster endocrine corrigé permet de bien visualiser les différents types cellulaires de façon cohérente entre les études (Figure 18).

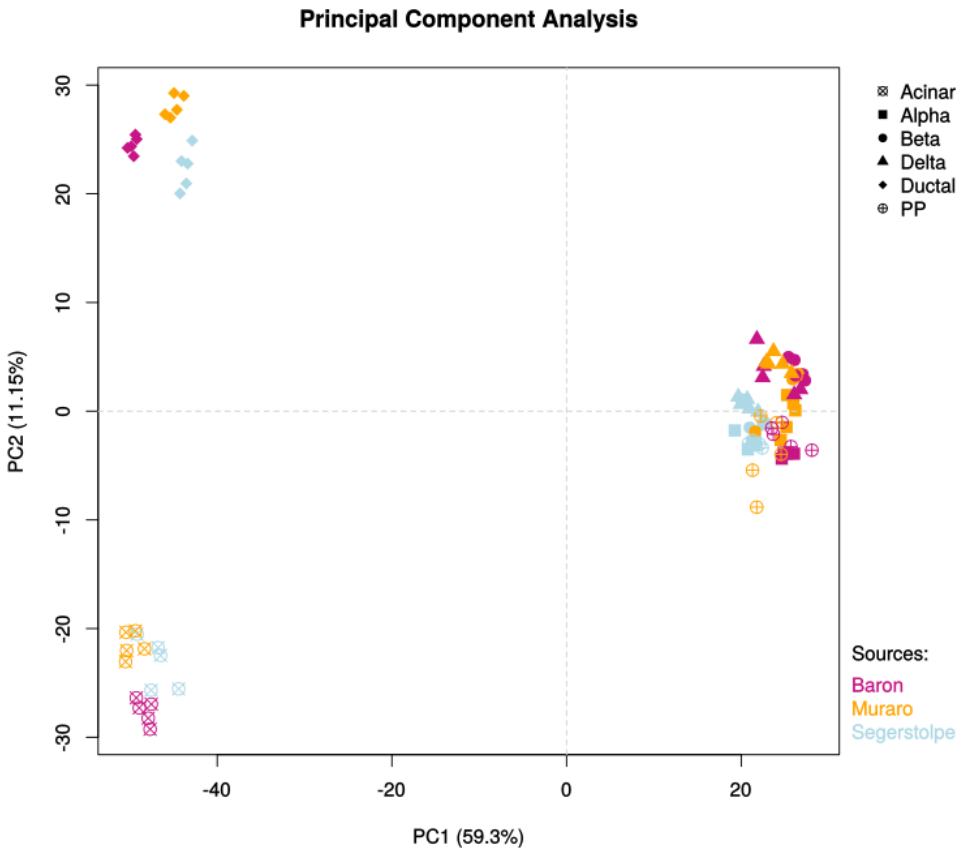


**Figure 17** – *PCA sur les données SingleCell RNAseq de Baron, Muraro et Segerstolpe après une correction d'effet Batch avec le package Combat.*

Les observations sur la correction d'effets batch sont similaires avec les analyses de SERE (à titre d'indice de corrélation) (Figure-Supplémentaire 2) et la visualisation des dendrogrammes (Figure-Supplémentaire 3). Concernant ces derniers, l'absence de correction démontre bien un rassemblement des échantillons basé sur l'origine des études et non sur les types cellulaires avec tout de même une séparation endocrine-exocrine. La correction de l'effet batch offre un dendrogramme parfaitement cohérent avec un rassemblement par type cellulaire.

Afin de déterminer si les trois études scRNAseq génèrent des résultats similaires, nous avons comparé la liste des gènes présentant une expression enrichie pour chaque type cellulaire pour les trois études. Nous avons donc sélectionné par l'utilisation du programme Deseq2 les gènes ayant

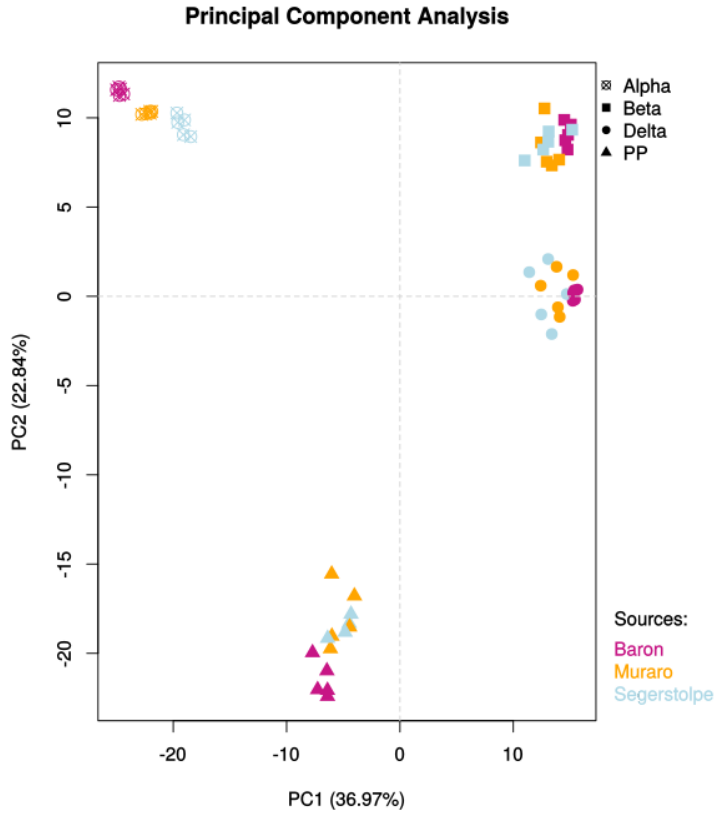


**Figure 18** – *PCA sur les données endocrines SingleCell RNAseq de Baron, Muraro et Segerstolpe après une correction d'effet batch avec le package Combat.*

une expression enrichie avec une p-djust inférieur à 0.1 et un ≪Fold Enrichment≫ positif. Un nombre similaire de gènes a été obtenu avec les trois études pour les différents types cellulaires. Cependant, la comparaison de ces trois listes de gènes montre que seulement 18-20 % sont communs aux trois études. Si on effectue la même analyse après correction de l'effet batch, le nombre de gènes communs diminue légèrement. Le package Combat permet donc d'atténuer les variations techniques entre les études et permet une meilleure comparaison graphique de celles-ci. Par contre, Combat n'améliore pas le nombre de gènes ayant une expression enrichie pour les 3 études (ou communs à 2 études). Les listes avant et après correction présentent tout de même plus de 70 % de similitude.

Dans tous les cas, l'ensemble des gènes spécifiques à chaque type cellulaire (gènes marqueurs) est commun à toutes les études. Ces marqueurs ont été sélectionnés à partir de la littérature selon leur spécificité à un type cellulaire (Table 2). La correction peut réduire le nombre de gènes associé à un GO Term connu mais en améliore néanmoins l'enrichissement (Figure 19-20). Le réarrangement minime de la liste de gènes communs semble offrir une structure plus cohérente

| Acinar | Alpha | Bêta | Delta | Ductal | PP |
|---|---|---|---|---|---|
| RNASE1 | GC | INS | SST | CFTR | PPY |
| CELA3B | TTR | MAFA | BCHE | SPARC | SERTM1 |
| CPA1 | CRYBA2 | IAPP | LEPR | RGS5 | CARTPT |
| PRSS2 | PLCE1 | DLK1 | FRZB | KRT19 | ETV1 |
| CLPS | PTGER3 | ADCYAP1 | HHEX | VWF | THSD7A |
| PRSS1 | LOXL4 | NPTX2 | RGS2 | PLAT | AQP3 |
| PLA2G1B | IRX2 | SIX3 | RBP4 | PDGFRB | MEIS2 |
| REG1B | KLHL41 | NKX6.1 | GABRG2 | C10orf128 | ID2 |
| REG1A | GCG | PFKFB2 | ERBB4 | SDC4 | ID4 |
| CPA2 | IRX1 | PDX1 | | | LMO3 |
| CTRB1 | | PIR | | | ENTPD2 |
| CXCL17 | | | | | |
| CTRB2 | | | | | |
| ALDOB | | | | | |
| IL32 | | | | | |
| SPINK1 | | | | | |
| DUOX2 | | | | | |
| MUC1 | | | | | |

**Table 2** – *Table récapitulative de gènes marqueurs par type cellulaire*

autour des voies métaboliques connues.

Il y a donc des différences entre les trois études scRNAseq mais les gènes communs aux trois études (ou à deux études) peuvent être considérés comme différentiellement surexprimés.

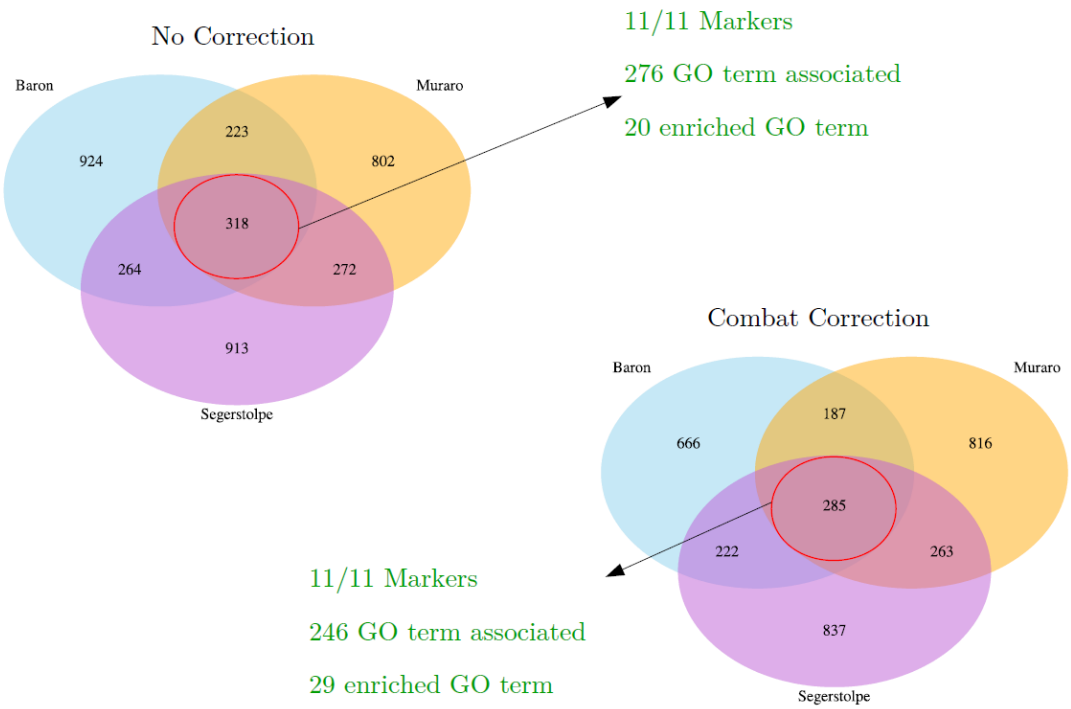


**Figure 19** – *Comparaison type entre listes de gènes enrichis à partir des différentes études. Il s'agit ici d'un croisement des listes pour les cellules bêtas qui illustre les observations générales des comparaisons des autres types cellulaires.*

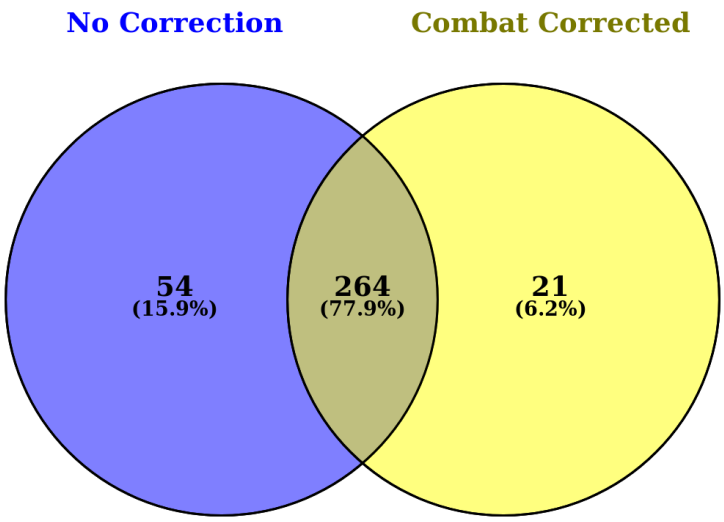


**Figure 20** – *Intersection entre la liste de gènes enrichis communs aux trois études avant et après correction d'effet «batch» par le package Combat.*

### 3.2.2 Comparaison des profils transcriptomiques SingleCell RNAseq et Bulk RNA-seq humains

Dans une même approche que pour le point précédent, le but de cette étape était de déterminer le degré de similarité des données transcriptomiques obtenues pour les trois études scRNAseq sélectionnées avec l'étude ≪bulk≫ RNAseq utilisée par [Tarifeño-Saldivia et al., 2017] ([Blodgett et al., 2015]). Pour rappel, seul des données alphas et bêtas étaient disponibles en Bulk RNAseq humain à partir de cette source.

La première variable de l'analyse PCA est cette fois-ci liée à l'origine des données. La seconde variable est quant à elle liée à la distinction des types cellulaires. (Figure 21). Une grande variabilité est observée entre les réplicas de l'analyse Bulk comparée au Single Cell. La distinction entre les deux types cellulaires semble néanmoins similaire, quelle que soit la technique de séquençage employée.

Une correction d'effet batch relie la première variable du PCA à la distinction des types cellulaires (Figure 22). Les réplicas des données Bulk de Blodget sont localisés plus centralement sur l'axe PC1 alors que les réplicas Single Cell sont aux extrémités de cet axe. Ceci pourrait suggérer que la séparation des cellules alphas et bêtas n'était pas optimale dans l'étude de Blodget et que ses préparations de cellule n'étaient pas complètement pures.

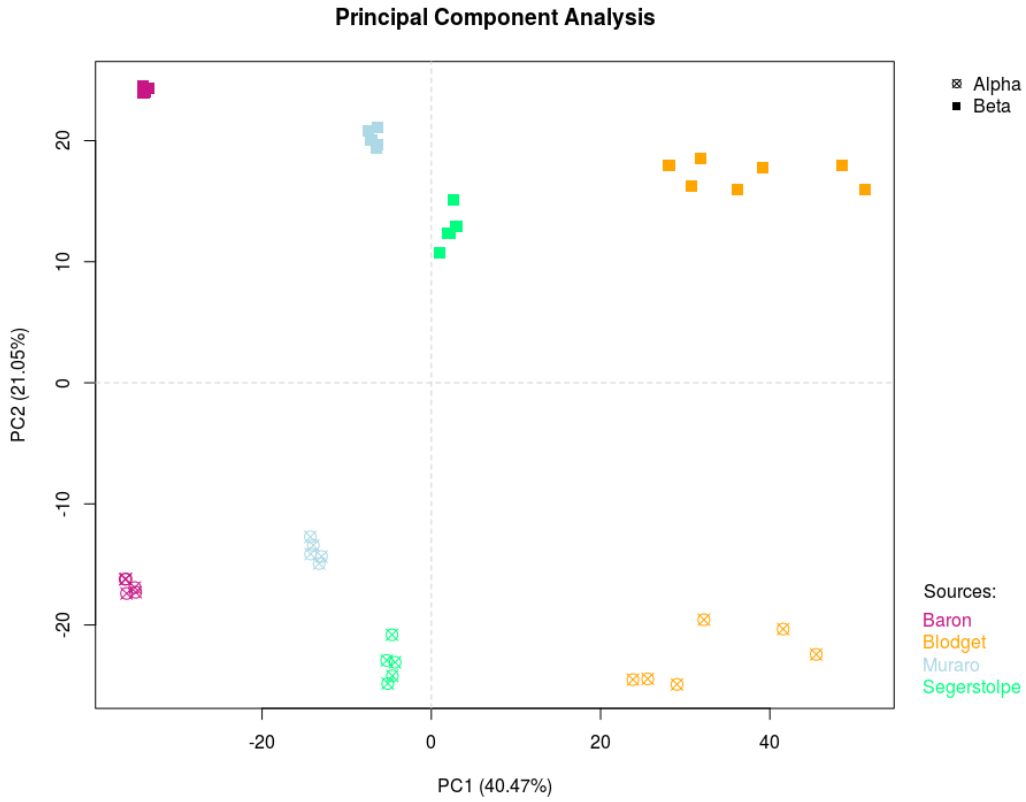


**Figure 21** – *PCA sur les données SingleCell RNAseq (Baron, Muraro et Segerstolpe) et Bulk RNAseq (Blodget) humaines.*

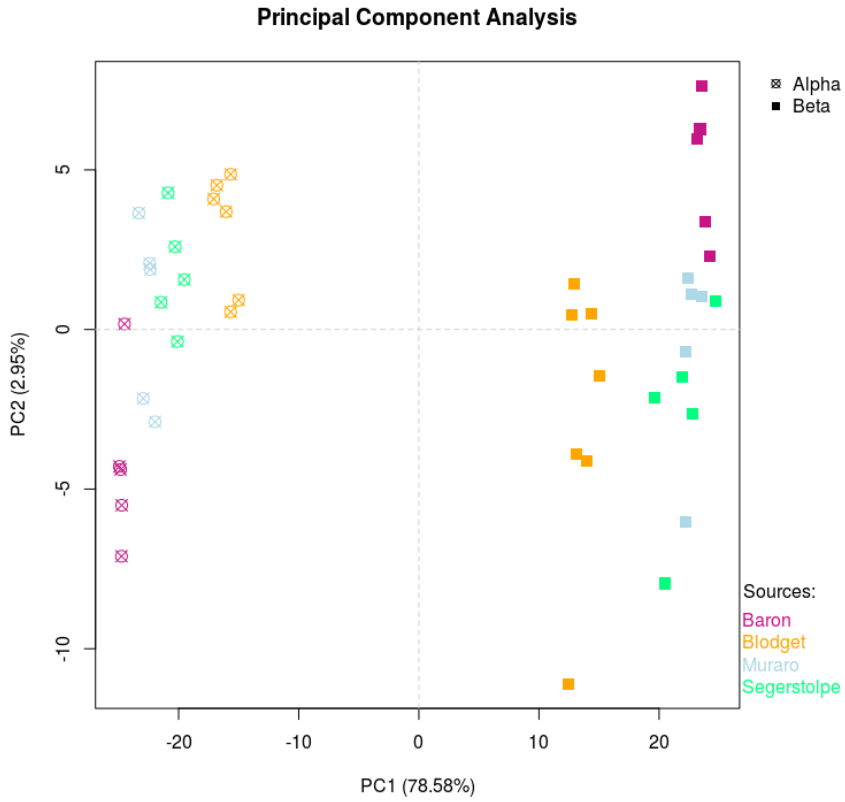


**Figure 22** – *PCA sur les données SingleCell RNAseq (Baron, Muraro et Segerstolpe) et Bulk RNAseq (Blodget) humaines après une correction d'effet batch par le package Combat.*

Les analyses par SERE (Figure-Supplémentaire 4) et dendrogramme (Figure-Supplémentaire 5) confirment la forte variabilité transcriptomique au sein même des réplicas de l'étude Bulk RNAseq. Les corrélations relativement élevées détectées entre les cellules alphas et bêtas pour les données de Blodget suggèrent une contamination par l'autre type cellulaire lors de leur préparation en laboratoire. Même la correction d'effet Batch ne permet pas d'éliminer les biais de ces contaminations. Le dendrogramme de la indique que le réplica 3 des cellules bêtas est fortement contaminé par des cellules alphas.

Une analyse d'expression différentielle entre les cellules alphas et cellules bêtas a été réalisée pour chaque étude. La comparaison de ces listes de gènes révèle une nette dégradation de l'information connue portée par ces listes. Une chute du nombre de marqueurs de référence est observée et il en va de même pour le nombre de GO Term enrichis (Figure 23). Le package Combat fonctionnant sur un principe de modèle linéaire, il est probable que la variabilité des données Bulk impacte les données SingleCell avec un effet néfaste évident.

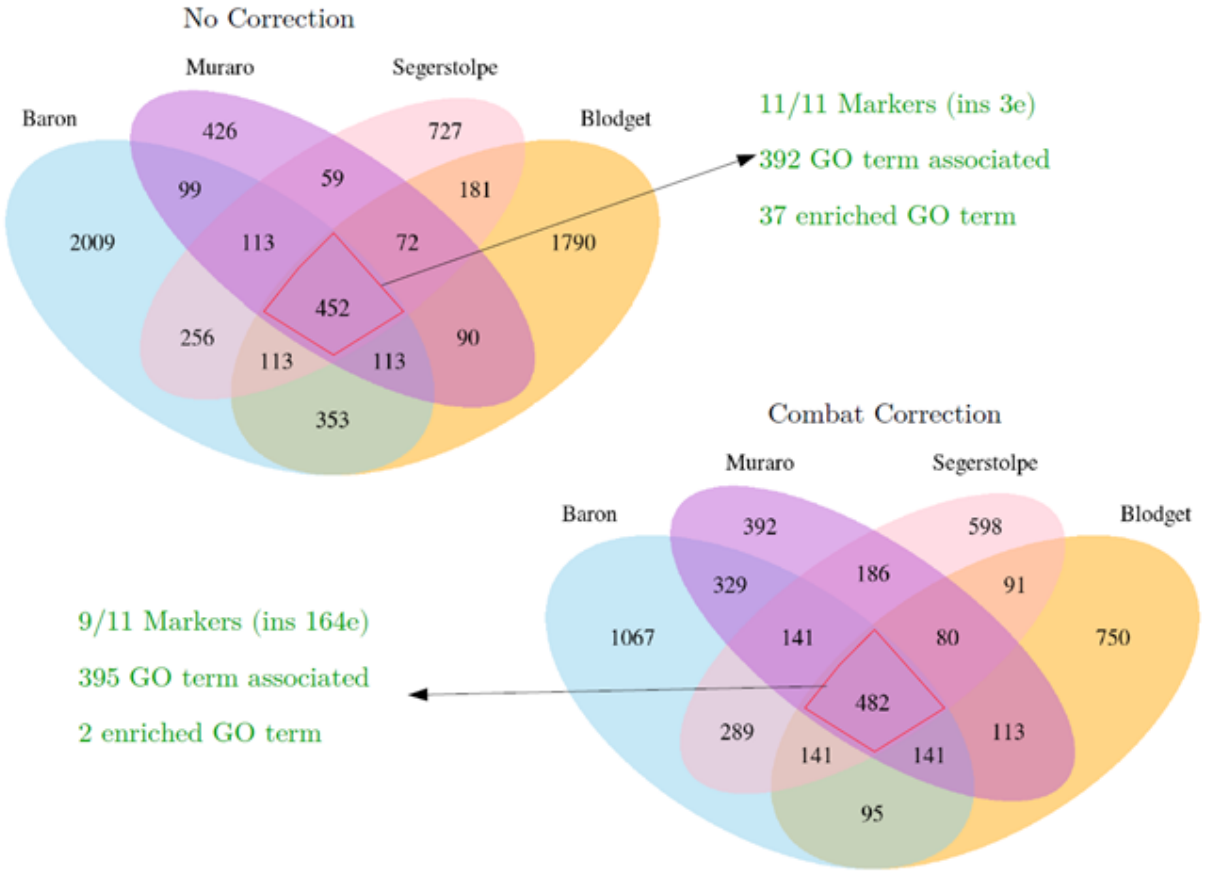


**Figure 23** – *Comparaison type entre listes de gènes enrichis à partir des différentes études "SingleCell" et "Bulk". Il s'agit ici d'un croisement des listes pour les cellules bêtas qui illustre les observations générales des comparaisons des autres types cellulaires.*

Ces résultats démontrent par ailleurs l'intérêt de reproduire les analyses de [Tarifeño-Saldivia et al., 2017] à partir de données SingleCell au vu de la qualité limitée des données Bulk humaines.

## 3.3 Comparaison des profils transcriptomiques SingleCell RNAseq vs Bulk RNAseq de souris

Des données Bulk RNAseq étaient disponibles pour les cellules alphas, bêtas et deltas de souris ([DiGruccio et al., 2016]). Nous avons comparé dans un graphe de ≪principal component analysis≫ ces données aux scRNAseq de souris publié par [Baron et al., 2016].Comme pour l'analyse sur données humaines, la première variable du PCA est liée à l'origine des échantillons. La distinction des types cellulaires sur la seconde variable est similaire entre les deux études (Figure 24). La correction d'effet batch permet de bien regrouper les différents types cellulaires (Figure 25) et offre des résultats cohérents avec les analyses par SERE et dendrogramme (Figure-Supplémentaire 6-7).

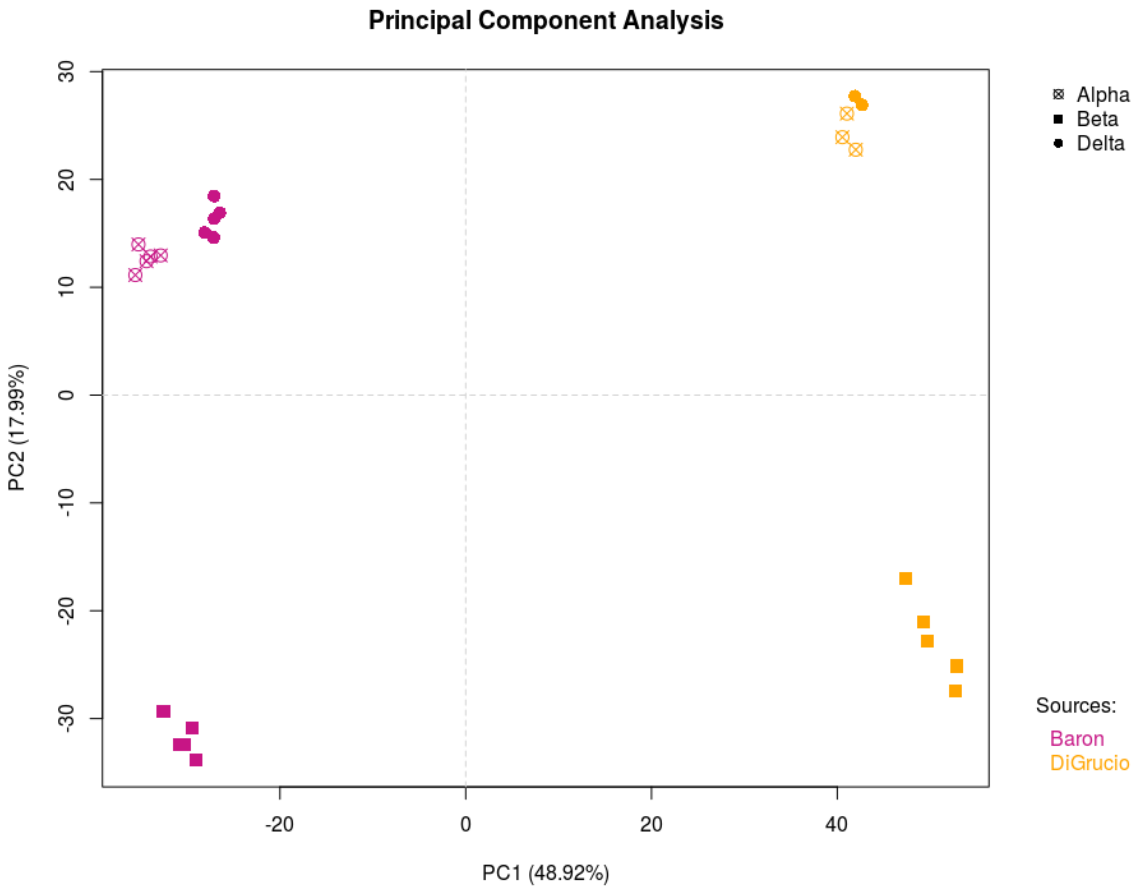


**Figure 24** – *PCA sur les données SingleCell RNAseq (Baron) et Bulk RNAseq (DiGrucio) chez la souris.*

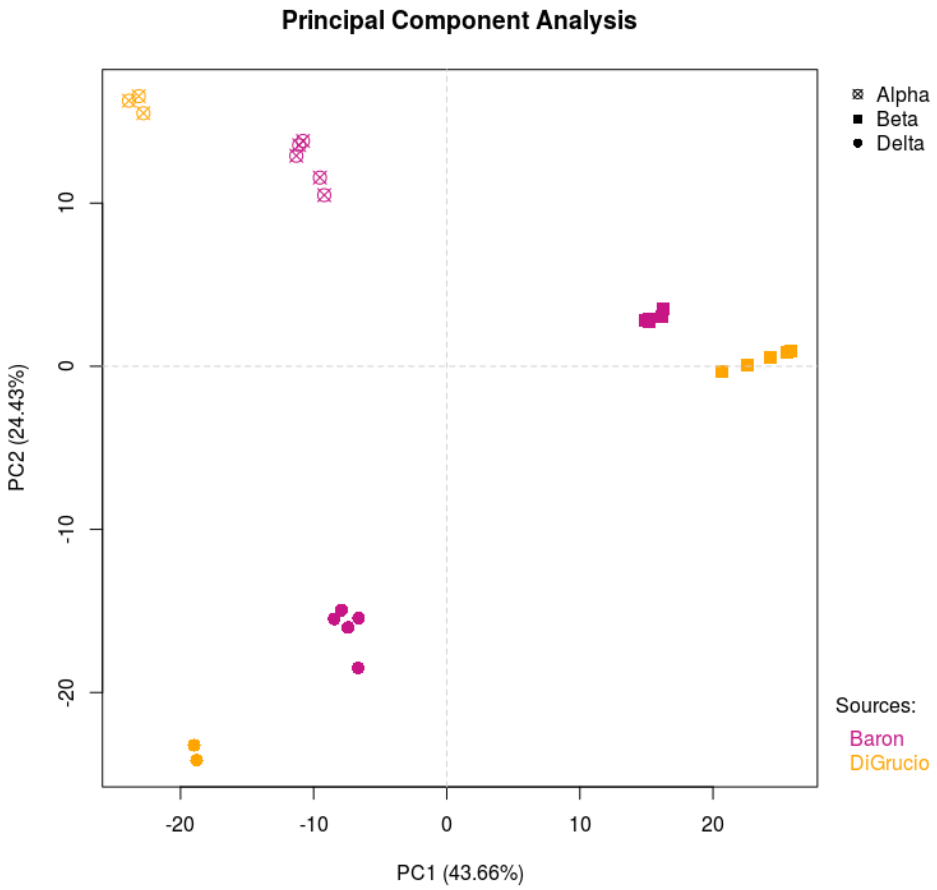


**Figure 25** – *PCA sur les données SingleCell RNAseq (Baron) et Bulk RNAseq (DiGrucio) chez la souris après une correction d'effet batch avec le package Combat.*

Il en va de même pour la comparaison des listes de gènes dont l'expression est enrichie dans les cellules alphas, bêtas ou deltas, et où la correction de l'effet batch dans ce cas met en évidence beaucoup plus d'informations communes entre les deux études (Figure 26).

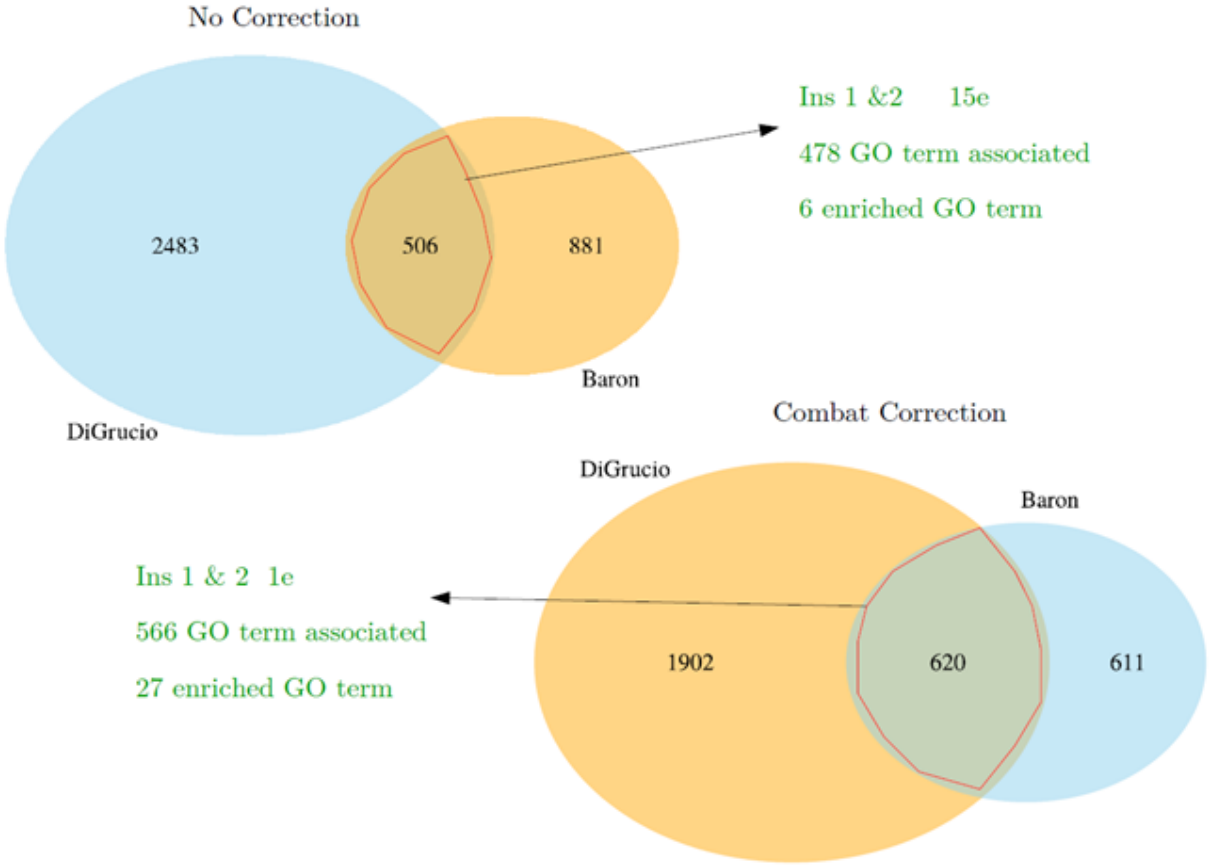


**Figure 26** – *Comparaison type entre listes de gènes enrichis à partir des études "SingleCell" et "bulk" de souris. Il s'agit ici d'un croisement des listes pour les cellules bêtas qui illustre les observations générales des comparaisons des autres types cellulaires.*

## 3.4 Choix des jeux de données RNA-seq pour réaliser les comparaisons «inter-espèces»

Une seule source de données est disponible pour la souris en «SingleCell» et seules les données «Bulk» du laboratoire décrivent le transcriptome du pancréas du Zebrafish. Ces deux jeux de données fournissent des listes de gènes enrichis pour chaque type cellulaire par l'utilisation du «package» DESeq2 comme décrit dans les méthodes. En revanche, pour les données humaines «SingleCell», trois études présentent un intérêt similaire (Baron, Muraro et Segerstolpe). La question se pose donc de quelle source choisir ou de comment générer des listes de gènes enrichis à partir des trois «datasets» différents pour les analyses inter-espèces. Trois méthodes de production de listes ont été comparées sur base de ces données à savoir le «data-merging», la méta-analyse et la méthode intermédiaire telle que décrite précédemment avec une p-adjust inférieur à 0.1, un LogFoldEnrchment supérieur à 1 (section 1.5 - Comparaison méthodologique, Figure 5). Il n'est

pas aisé de définir dans quelle mesure une liste de gènes enrichis est plus valable qu'une autre. Pour évaluer la qualité des listes, une recherche de gènes marqueurs de référence (Table 2) a été effectuée. De plus, une analyse par Gene Ontology avec le système Gorilla permet d'estimer la proportion d'information connue dans ces listes (rapport du nombre de gènes associé à un GO avec le nombre total de gènes de la liste).

Dans le cadre de l'approche méta-analytique, seuls les gènes communs à au moins deux études ont été sélectionnés pour produire les listes. Cela correspond à sélectionner toutes les intersections des diagrammes de Ven produit selon le même principe que pour l'analyse des profils transcriptomiques SingleCell RNAseq humains (section 3.2.1).

L'approche intermédiaire se base sur l'analyse par DESeq2 de tous les répliquas des différents types cellulaires sans distinction selon l'origine des données. Cela revient donc à mélanger toute nos données au stade de répliquas et cela génère directement une liste de gènes enrichis par type cellulaire.

Pour le data-merging, les données sont mélangées au stade des profils transcriptomiques de chaque cellule individuelle au départ de l'analyse. Cette méthode génère également directement une liste de gènes enrichis par type cellulaire mais fait néanmoins appel à un outil de correction d'effet batch particulier : le ≪Mutal Nearest Neighbor≫ (MNN). Ce système recherche, pour chaque cellule d'une étude, la cellule d'une autre étude avec le profil transcriptomique le plus proche. Cette recherche en ≪pairwise≫ permet de définir des vecteurs de corrections basés sur les différences d'expression des cellules les plus proches issues de sources différentes. Cet outil implémenté dans le package bioconductor ≪scran≫ pour le traitement de données ≪SingleCell≫ est considéré comme plus efficace que l'algorithme Combat à ce niveau de correction [Haghverdi et al., 2017].

En absence de ce type de correction, le ≪clustering≫ par T-SNE sépare clairement des groupes de cellules selon leur étude d'origine (Figure 27). L'application du script de correction permet d'isoler rapidement les types cellulaires sans effets batch visibles (Figure 28). Néanmoins, les cellules bêtas et ductales sont difficiles à distinguer les unes des autres ce qui pourrait biaiser les listes de gènes enrichis finales.

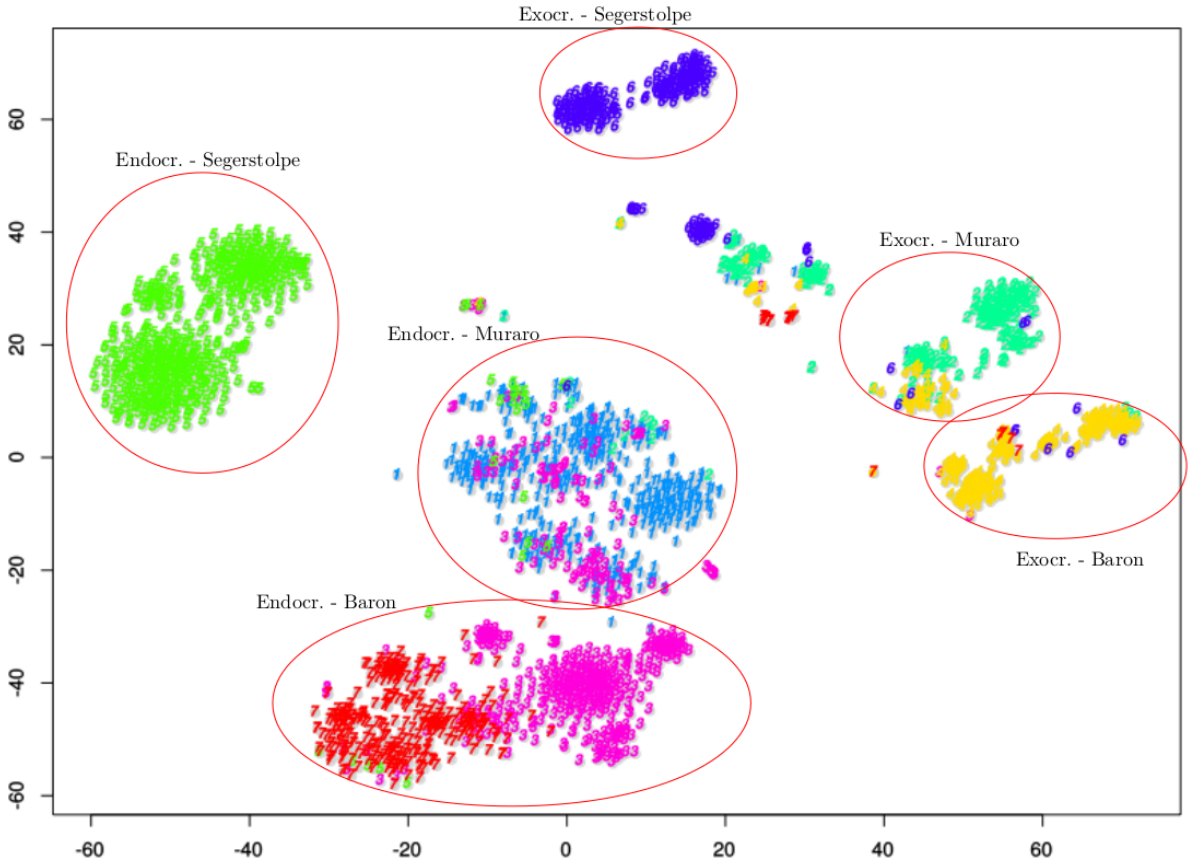


**Figure 27** – *Clustering par t-SNE sur une mélange de données issues des études de Baron, Muraro et Segerstolpe sans correction d'effet "batch". Les couleurs et numérotations sont générées automatiquement par le programme. Les annotations des différents clusters ont été obtenues après observation du niveau d'expression des gènes marqueurs de références cumulé à l'origine des données.*

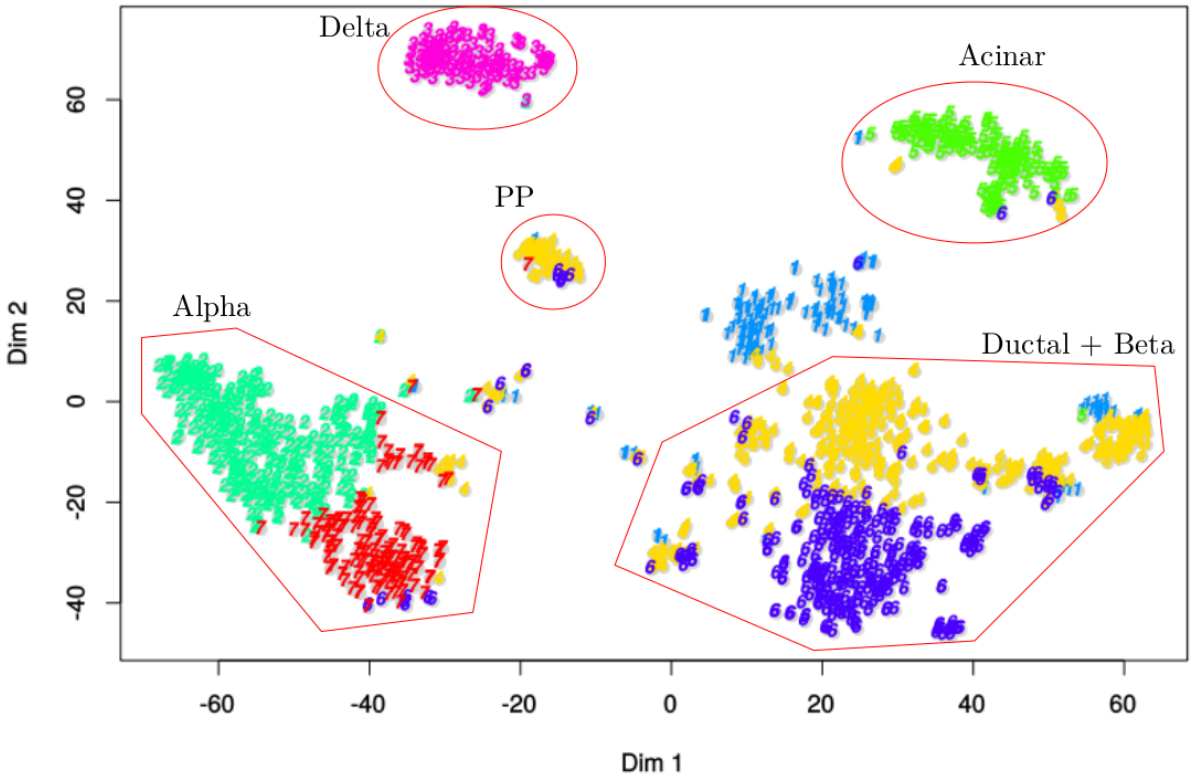


**Figure 28** – *Clustering par t-SNE sur une mélange de données issues des études de Baron, Muraro et Segerstolpe avec correction d'effet "batch" par "Mutual Nearest Neighbor". Les couleurs et numérotations sont générées automatiquement par le programme. Les annotations des différents clusters ont été obtenues après observation du niveau d'expression des gènes marqueurs de références cumulé à l'origine des données.*

A partir de ce mélange de données, nous avons utilisé le programme DEseq2 pour générer la liste des gènes ayant une expression enrichie dans chaque type cellulaire.

Comparativement, l'approche «data-merging», comme l'approche méta-analytique, permet de retrouver tous les gènes marqueurs des différents types cellulaires, ce qui n'est pas le cas de l'approche intermédiaire (lacunes pour les cellules bêtas, acinaires et ductales). En termes de proportion d'information connue par GO, l'approche méta-analytique semble systématiquement meilleure (Figure 29).

Meta Analytic

| | Markers | Nb Genes | Associated GO | Information (%) |
|---|---|---|---|---|
| Alpha | 10/10 | 115 | 98 | 85 |
| Beta | 11/11 | 116 | 99 | 85 |
| Delta | 9/9 | 161 | 127 | 79 |
| PP | 11/11 | 244 | 144 | 59 |
| Acinar | 18/18 | 466 | 383 | 82 |
| Ductal | 9/9 | 698 | 623 | 89 |

Intermédiaire

| | Markers | Nb Genes | Associated GO | Information (%) |
|---|---|---|---|---|
| Alpha | 10/10 | 329 | 212 | 64 |
| Beta | 10/11 | 241 | 156 | 65 |
| Delta | 9/9 | 595 | 371 | 62 |
| PP | 11/11 | 796 | 373 | 46 |
| Acinar | 16/18 | 950 | 562 | 59 |
| Ductal | 5/9 | 1011 | 818 | 81 |

Data Merging

| | Markers | Nb Genes | Associated GO | Information (%) |
|---|---|---|---|---|
| Alpha | 10/10 | 198 | 151 | 76 |
| Beta | 11/11 | 153 | 122 | 80 |
| Delta | 9/9 | 216 | 158 | 73 |
| PP | 11/11 | 333 | 157 | 47 |
| Acinar | 18/18 | 576 | 421 | 73 |
| Ductal | 9/9 | 905 | 796 | 88 |

**Figure 29** – *Table récapitulative des gènes marqueurs retrouvés selon les méthodes de production de listes de gènes enrichis.*

Étant donné que l'approche méta-analytique identifie tous les marqueurs de référence sans introduire de correction de l'effet «batch», nous avons décidé d'utiliser les résultats de cette méthode pour effectuer les comparaisons inter-espèces.

## 3.5 Comparaison inter-espèces et détermination des signatures transcriptomiques conservées pour les types cellulaires pancréatiques

L'étape suivante de notre travail était de comparer la liste des gènes ayant une expression enrichie dans chaque type cellulaire pour l'homme, la souris et le zebrafish afin de déterminer la signature transcriptomique des types cellulaires pancréatiques commune aux vertébrés. Ceci permettait de vérifier les observations de Tarifeno et al,. concernant les cellules alphas et bêtas, et d'élargir l'analyse à de nouveaux types cellulaires. Une comparaison inter-espèces a donc été réalisée à partir de données «SingleCell» de souris (Baron et al.,), de données «SingleCell» humaines (gènes enrichis confirmés par au moins 2 études) et de données «bulk» de

poissons zèbres. Le Zebrafish ne dispose pas de cellules PP et les données de souris ne contiennent pas de cellules acinaires. La comparaison s'est donc limitée aux cellules alphas, bêtas, deltas et ductales.

La comparaison des listes de gènes enrichis des différentes espèces se fait sur base de table d'orthologie en «pairwise». Cette comparaison inter-espèces n'est pas simple à réaliser car certains gènes sont dupliqués dans une espèce rendant les relations d'orthologie complexes. Par exemple, un seul gène codant pour l'insuline a été découvert chez l'homme alors que la souris et le zebrafish possèdent deux gènes insuline.

La comparaison des listes de gènes enrichis pour les cellules alphas montre que les signatures transcriptomiques sont majoritairement spécifiques de l'espèce (Figure 30). En effet, il n'y a que trois gènes conservés entre les trois espèces à savoir respectivement gcg (glucagon), arx et ldb2. Ces trois correspondances sont décrites comme des marqueurs spécifiques aux cellules alphas et sont spécifiquement enrichies et référencées dans les études SingleCell humaines (Muraro - Segerstolpe). Le gène arx est connu pour être crucial dans la différenciation des cellules alphas chez la souris et le zebrafish [Courtney et al., 2013].

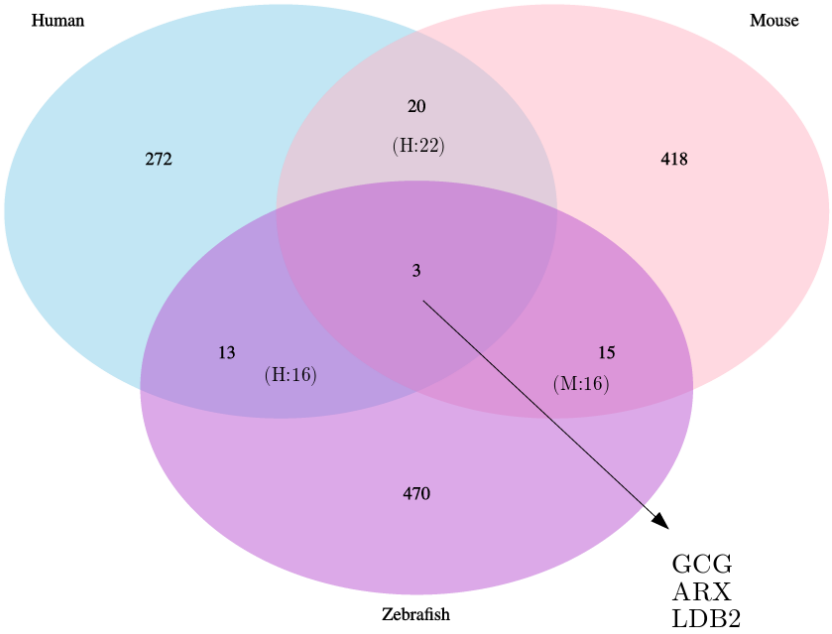


**Figure 30** – *Comparaison par diagramme de Venne des listes des gènes enrichis issues des trois espèces pour les cellules alphas.*

La même conclusion peut être faite pour la comparaison des gènes enrichis des cellules bêtas :

seul l'insuline ressort en tant que gène conservé entre les trois espèces (Figure 31). Comme attendu, on observe un plus grand nombre de marqueurs communs entre la souris et l'homme par rapport aux gènes communs homme-zebrafish ou souris-zebrafish. Par contre, le nombre de marqueurs communs à deux espèces reste faible.

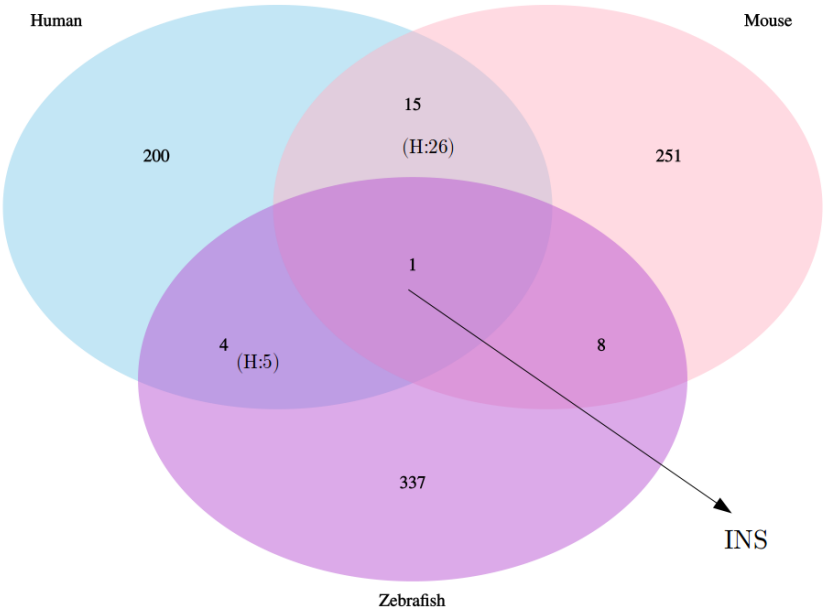


**Figure 31** – *Comparaison par diagramme de Venne des listes des gènes enrichis issues des trois espèces pour les cellules bêtas.*

Concernant les cellules deltas, 4 correspondances sont obtenues (Figure 32). Seule PCP4 (≪Purkinje cell protein 4≫) est décrite comme spécifique à ce type cellulaire (uniquement par Segerstolpe). Les études de Muraro et Segerstolpe décrivent SLITRK6 (codant pour ≪SLIT and NTRK-like protein 6≫) comme un marqueur spécifique des cellules PP. Néanmoins les niveaux d'expression de ce gène dans ces deux études révèlent également une légère surexpression dans les cellules deltas. En l'absence de cellule PP chez le zebrafish, cette surexpression devient significative et commune pour les 3 espèces.

Les gènes TOX2 (activateur transcriptionnel dans les glandes endocrines) et SFXN5 (acteur dans l'homéostasie cellulaire du fer) n'ont encore jamais été décrits comme marqueurs spécifiques deltas et présentent un niveau d'expression faible à la limite des ≪treshold≫ fixés.

Il est étonnant de ne pas retrouver la somatostatine comme gène commun entre les espèces. Cette correspondance n'existe qu'entre l'homme et la souris parmi les 31 gènes qu'ils partagent spécifiquement. L'absence de somatostatine chez le zebrafish provient du fait que les gènes

somatostatine 2 et somatostatine 1.2, qui sont surexprimés dans les cellules deltas chez le zebrafish, ne sont pas considérés comme les orthologues des gènes somatostatines de la souris et de l'homme (voir discussion).

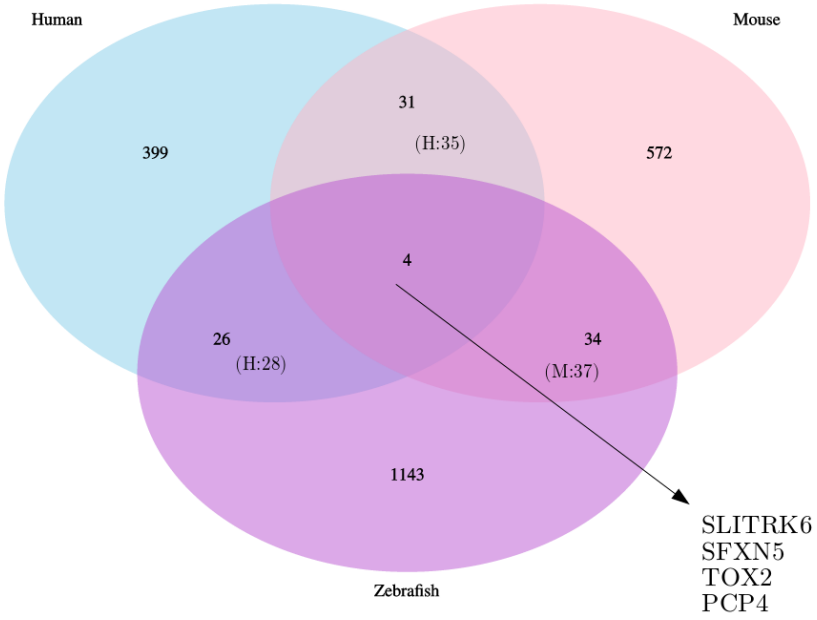


**Figure 32** – *Comparaison par diagramme de Venne des listes des gènes enrichis issues des trois espèces pour les cellules deltas.*

Enfin, les cellules ductales présentent un grand nombre de similitudes entre les trois espèces (Figure 33). Cette importante correspondance va de pair avec la taille des listes de gènes enrichis qui est également plus importante pour chaque espèce. Ceci peut être expliqué par le fait que les cellules ductales sont de type exocrine fortement différentes des cellules endocrines auxquelles elles ont été comparées pour générer la liste de gènes enrichis. Les analyses comparatives tenant compte des cellules acinaires (autre type cellulaire exocrine) permettraient de réduire les listes «ductales» en éliminant les gènes liés aux voies communes d'excrétion mais aucune cellule acinaire n'était disponible chez la souris. Néanmoins, on peut observer que la signature des cellules ductales/exocrines est mieux conservée que celle des sous-types cellulaires endocrines alphas, bêtas et deltas.

Parmi les 308 correspondances se trouvent trois gènes marqueurs connus pour les cellules ductales. Une analyse par «Gene Onthology» sur ces gènes communs met en évidence la voie «Notch signaling pathway» connue pour son implication dans le développement des cellules ductales [Greenwood et al., 2007] et des cancers du pancréas [Gao et al., 2017].

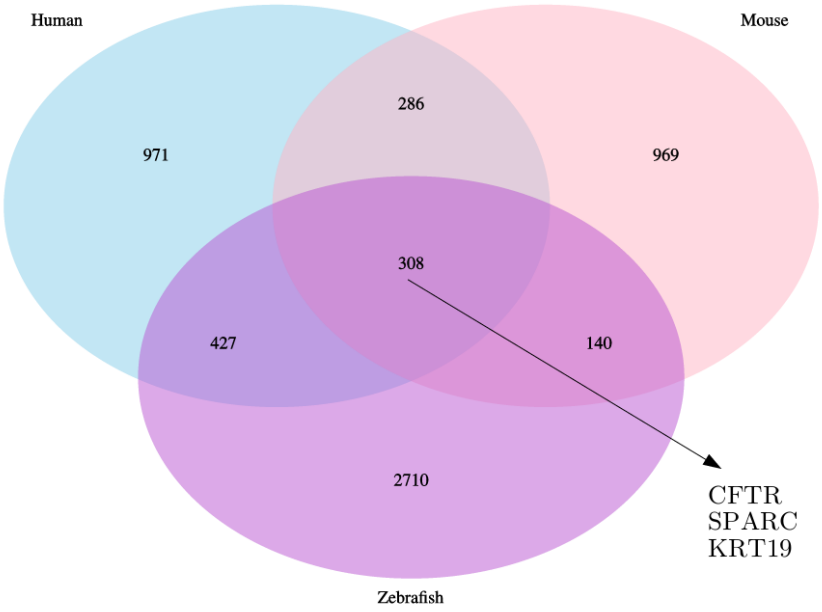


**Figure 33** – *Comparaison par diagramme de Venne des listes des gènes enrichis issues des trois espèces pour les cellules ductales.*

Comme le transcriptome des cellules ductales et acinaires a été obtenu pour l'homme et le zebrafish, la signature des cellules ductales a pu être déduite pour ces deux espèces en retirant les marqueurs propres à la voie exocrine (Figure 34). Comme on peut s'y attendre, moins de gènes communs sont détectés pour cette comparaison. Néanmoins, on retrouve encore les gènes marqueurs SPARC et KRT19 comme gènes conservés.

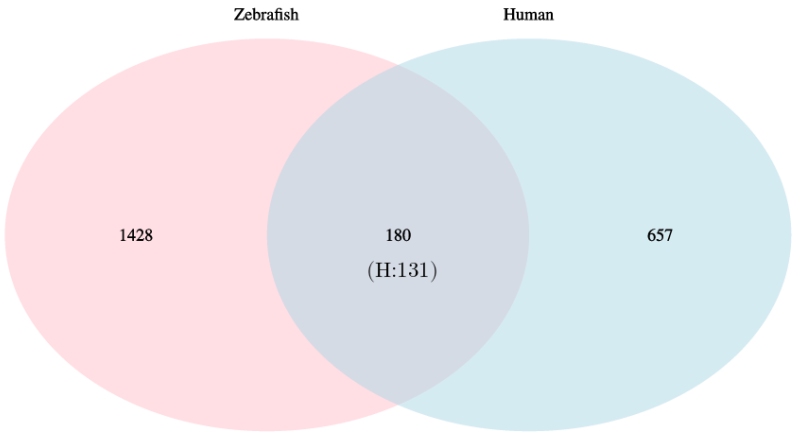


**Figure 34** – *Comparaison par diagramme de Venne des listes des gènes enrichis issues des données humaines et de poissons zèbres pour les cellules ductales en tenant compte de la population cellulaire alpha.*

# 4 Discussion

L'objectif principal de ce mémoire était d'analyser et de comparer les données scRNA-seq pancréatiques afin de déterminer les signatures transcriptomiques des types cellulaires pancréatiques au sein de chaque espèce et également entre les espèces. Ces analyses nous ont également permis de comparer les données scRNAseq obtenues par différents laboratoires, et de comparer les données scRNAseq aux données «bulk RNAseq».

## 4.1 Comparaison des données scRNAseq obtenues par différents groupes de recherche

En tenant compte de toutes les études SingleCell humaines analysées dans le cadre de ce mémoire, il semble clair qu'une grande variabilité existe entre elles. Que cela soit lié au nombre de cellules disponibles, aux proportions de cellules pour les différents types cellulaires ou à la qualité des données (impactant la qualité des «clustering» comme avec l'effet donneur par exemple), les différences inter-études ont mené à l'élimination de certaines d'entre elles pour la suite des analyses. Ainsi, les données de Lawlor et Xin ont été rejetées à cause d'une faible qualité des données menant à un mauvais clustering. Les données de Enge ont également été écartées vu l'importance de l'effet donneur ainsi que le peu de cellules disponibles pour les types PP et delta. Il est intéressant de noter que ces trois études emploient toutes un protocole de type SmartSeq. Il serait donc possible que cette technique induise davantage de variation technique et soit moins robuste que le CELseq.

Un léger effet «batch» est perceptible entre les trois études restantes avec une analyse méta-analytique (Baron, Muraro et Segerstolpe). La correction de cet effet par le programme Combat génère des résultats plus cohérents pour la comparaison graphique des profils transcriptomiques mais en ce qui concerne les gènes marqueurs communs entre les études, il n'y a pas d'amélioration évidente.

En comparant les trois méthodes de production de listes de gènes enrichis (méta-analytique, intermédiaire et «data-merging») à partir de ces trois sources de données, il s'est avéré que l'approche méta-analytique et le «data-merging» permettent de retrouver tous les gènes marqueurs pour tous les types cellulaires. Ce n'est pas le cas de la méthode intermédiaire qui présente des lacunes dans ces listes. La méthode du «data-merging» permet d'obtenir plus de gènes enrichis mais une plus faible proportion d'entre eux sont associés à un GO term en comparaison avec la méta-analyse. De plus, le «data-merging» applique une correction d'effet batch qui modifie les

niveaux d'expression dans les données d'origine. Pour cette raison, l'approche méta-analytique a été sélectionnée pour produire les listes de gènes enrichis nécessaires pour la suite de nos analyses.

## 4.2 Comparaison des données scRNAseq et «bulk»

Une forte variabilité inter-réplicas a été observée dans les données «bulk» humaines. Cela n'est pas le cas pour les données «bulk» de souris. Cette variabilité est probablement due à des contaminations pour les données humaines étant donné la méthode d'isolement des cellules employée. En effet, pour la souris et le poisson zèbre, des lignées transgéniques sont disponibles et permettent d'isoler précisément les cellules exprimant un fluorochrome de nature protéique. Il n'existe évidemment pas de lignées transgéniques chez l'homme. L'isolement des cellules se fait donc sur base de la fixation d'un anticorps fluorescent spécifique à un marqueur de surface qui serait exprimé par le type cellulaire à isoler. Cette méthode semble moins précise que l'expression d'un transgène fluorescent. La méthode de scRNAseq permet donc de résoudre ce problème et d'obtenir le transcriptome de chaque type cellulaire sans aucune contamination.

## 4.3 Comparaison inter-espèces

Pour les cellules endocrines alphas, bêtas et deltas, les signatures transcriptomiques sont très différentes entre les espèces : la grande majorité des gènes ayant une expression enrichie dans un type cellulaire sont spécifiques de chaque espèce (voir figure 32 à 33). Pour les cellules ductales en revanche, il y a beaucoup plus de marqueurs communs entre les études. Cela est dû à la comparaison endocrine-exocrine qui est plus conservée entre les espèces, comme l'a démontré l'étude de Tarifeno et al., montrant une conservation des marqueurs endocrines et exocrines mais peu de conservation pour les marqueurs alphas et bêtas.

De plus, l'homme et la souris présentent davantage de similitudes ensemble que comparés au poisson zèbre pour les cellules alphas et bêtas comme cela a également été observé. Cela paraît cohérent au vu de leur plus grande proximité évolutive par rapport au poisson zèbre. Cette proximité n'a néanmoins pas été observée pour les nouveaux types cellulaires étudiés (delta et ductale). Cela pourrait par ailleurs s'expliquer par un plus grand nombre de gènes enrichis disponibles chez le poisson zèbre pour ces deux catégories de cellules.

Comme nous pouvions l'espérer, les résultats des analyses inter-espèces confirment donc les observations de Tarifeno et al., concernant le faible nombre de gènes conservés entre les espèces pour les différents types cellulaires du pancréas. Ce nombre est évidemment encore plus restreint dans le cadre de ce mémoire étant donné le nombre plus important de types cellulaires analysés.

Par exemple, le gène PDX1 avait été identifié par Tarifeno comme faisant partie de la signature conservée des cellules bêtas. En effet, ce gène joue un rôle crucial dans la différenciation des cellules bêtas chez la souris et le zebrafish (ref). Par contre, notre étude n'identifie pas ce facteur car il est exprimé à un niveau élevé dans les cellules ductales. De même, le facteur hhex est important pour la différenciation des cellules deltas ; cependant, ce facteur n'est pas présent dans la signature des cellules deltas car il est également fort exprimé dans les cellules ductales. Il serait donc probablement plus judicieux de faire l'analyse d'expression différentielle entre les 3 types cellulaires endocrines, qui présentent un transcriptome similaire, sans inclure les cellules ductales.

Il reste néanmoins surprenant qu'une si faible proportion de gènes soient exprimés spécifiquement dans les différentes populations de cellules. On s'attendrait en effet à avoir davantage de correspondances liées aux voies de perception du glucose et à la régulation de la sécrétion des hormones, par exemple. En effet, chez tous les vertébrés, l'hyperglycémie stimule la sécrétion de l'insuline et l'hypoglycémie augmente la sécrétion de glucagon. On s'attendait donc à détecter plus de similarité dans les transcriptomes des cellules pancréatiques entre les 3 espèces.

Plusieurs hypothèses peuvent être proposées pour expliquer ce phénomène. Tout d'abord, il est possible que ces voies subissent de fortes variations au cours de l'évolution, où des gènes homologues, mais non orthologues, prennent la même fonction dans les différentes espèces. Par exemple, le transporteur au zinc slc est surexprimé dans les cellules bêtas humaines et de souris, par contre c'est un autre transporteur au zinc qui est surexprimé dans les cellules bêtas de zebrafish.

Il est également possible que la méthode de comparaison par table d'orthologie souffre de lacunes. Si cela peut être compréhensible entre l'homme et le poisson zèbre, cela reste étonnant concernant l'orthologie entre l'homme et la souris. Un tel problème dans les tables d'orthologie peut être mis en évidence avec le gène de la somatostatine exprimé spécifiquement par les cellules deltas. Cette hormone est bien conservée entre l'homme et la souris mais le poisson zèbre possède trois gènes différents (appelés paralogues) : sst1.1, sst1.2 et sst2. Les données ≪bulk≫ de poisson zèbre utilisées pour nos analyses ont été produites à partir de cellules deltas exprimant un transgène lié à l'expression de la sst2 ; ces cellules expriment de grande quantité de sst2 et sst1.2. Cependant, les tables d'orthologie générées à partir des données ≪Ensembl≫ indiquent que le gène orthologue chez le zebrafish correspondant au gène somatostatine de l'homme et de la souris est le gène sst1.1. Cela explique donc l'absence de ce marqueur dans la signature conservée des cellules deltas et illustre bien le point faible des tables d'orthologie.

En termes de perspectives, la parution de données scRNAseq sur des cellules pancréatiques acinaires de souris permettra de réaliser des comparaisons pour tous les types cellulaires y compris de refaire une comparaison endocrine-exocrine.

# 5 Conclusion

Ces dernières années ont vu l'émergence de nombreuses études scRNAseq sur le pancréas. Afin de vérifier et d'approfondir les observations réalisées par Tarifeno et al., basées sur des données de type «bulk», ces différentes études ont été exploitées dans le cadre d'une comparaison inter-espèces entre l'homme, la souris et le poisson zèbre. Ces nombreuses études, bien qu'ayant des objectifs similaires, présentent certaines différences. Tout d'abord, certaines d'entre elles se basent sur un protocole «CelSeq» et d'autres sur un protocole «SmartSeq». Ensuite, il existe des variations quant aux nombres et à l'identité des gènes marqueurs propres à chaque type cellulaire qui compose le pancréas.

Une comparaison approfondie des études scRNAseq a permis de mettre en évidence une variabilité non négligeable entre elles, essentiellement basée sur le nombre de cellules disponibles et leurs proportions par type cellulaire ainsi que sur la qualité des données (comme l'effet donneur observé dans l'étude de Enge et al par exemple).

Seules les trois études de Baron, Muraro et Segerstolpe se sont avérées être de très bonne qualité et ont été sélectionnées pour poursuivre les analyses. Un net avantage de la technique «SingleCell» a pu être démontré en comparaison avec la technique RNAseq de type «bulk», notamment sur le point de la variabilité inter-réplicas.

En règle générale, très peu de gènes enrichis sont conservés entre les espèces pour les différents types cellulaires ce qui confirme les observations de Tarifeno et al., Les gènes conservés font généralement partie des gènes marqueurs déjà bien connus pour caractériser les cellules.

D'autres analyses seront à prévoir avec l'arrivée de nouvelles données «SingleCell» pour de nouveaux types cellulaires.

## Références

[Andrews, 2013] Andrews, S. (2013). FastQC : a quality control tool for high throughput sequence data. *Available online at : http ://www.bioinformatics.babraham.ac.uk/projects/fastqc.*

[Baker and Thummel, 2007] Baker, K. D. and Thummel, C. S. (2007). Diabetic Larvae and Obese Flies-Emerging Studies of Metabolism in Drosophila. *Cell Metabolism*, 6(4) :257–266.

[Baron et al., 2016] Baron, M., Veres, A., Wolock, S. L., Faust, A. L., Gaujoux, R., Vetere, A., Ryu, J. H., Wagner, B. K., Shen-Orr, S. S., Klein, A. M., Melton, D. A., and Yanai, I. (2016). A Single-Cell Transcriptomic Map of the Human and Mouse Pancreas Reveals Inter- and Intra-cell Population Structure. *Cell Systems*, 3(4) :346–360.

[Blodgett et al., 2015] Blodgett, D. M., Nowosielska, A., Afik, S., Pechhold, S., Cura, A. J., Kennedy, N. J., Kim, S., Kucukural, A., Davis, R. J., Kent, S. C., Greiner, D. L., Garber, M. G., Harlan, D. M., and DiIorio, P. (2015). Novel observations from next-generation RNA sequencing of highly purified human adult and fetal islet cell subsets. *Diabetes*, 64(9) :3172–3181.

[Butler et al., 2013] Butler, A. E., Campbell-Thompson, M. C., Gurlo, T., Dawson, D. W., Atkinson, M., and Butler, P. C. (2013). Marked expansion of exocrine and endocrine pancreas with incretin therapy in humans with increased exocrine pancreas dysplasia and the potential for glucagon-Producing neuroendocrine tumors. *Diabetes*, 62(7) :2595–2604.

[Cabrera et al., 2006] Cabrera, O., Berman, D. M., Kenyon, N. S., Ricordi, C., Berggren, P.-O., and Caicedo, A. (2006). The unique cytoarchitecture of human pancreatic islets has implications for islet cell function. *Proceedings of the National Academy of Sciences*, 103(7) :2334–2339.

[Courtney et al., 2013] Courtney, M., Gjernes, E., Druelle, N., Ravaud, C., Vieira, A., Ben-Othman, N., Pfeifer, A., Avolio, F., Leuckx, G., Lacas-Gervais, S., Burel-Vandenbos, F., Ambrosetti, D., Hecksher-Sorensen, J., Ravassard, P., Heimberg, H., Mansouri, A., and Collombat, P. (2013). The Inactivation of Arx in Pancreatic $\alpha$-Cells Triggers Their Neogenesis and Conversion into Functional $\beta$-Like Cells. *PLoS Genetics*, 9(10) :1–18.

[DiGruccio et al., 2016] DiGruccio, M. R., Mawla, A. M., Donaldson, C. J., Noguchi, G. M., Vaughan, J., Cowing-Zitron, C., van der Meulen, T., and Huising, M. O. (2016). Comprehensive alpha, beta and delta cell transcriptomes reveal that ghrelin selectively activates delta cells and promotes somatostatin release from pancreatic islets. *Molecular Metabolism*, 5(7) :449–458.

[Dobin et al., 2013] Dobin, A., Davis, C. A., Schlesinger, F., Drenkow, J., Zaleski, C., Jha, S., Batut, P., Chaisson, M., and Gingeras, T. R. (2013). STAR : Ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1) :15–21.

[Eizirik et al., 2012] Eizirik, D. L., Sammeth, M., Bouckenooghe, T., Bottu, G., Sisino, G., Igoillo-Esteve, M., Ortis, F., Santin, I., Colli, M. L., Barthson, J., Bouwens, L., Hughes, L., Gregory, L., Lunter, G., Marselli, L., Marchetti, P., McCarthy, M. I., and Cnop, M. (2012). The human pancreatic islet transcriptome : Expression of candidate genes for type 1 diabetes and the impact of pro-inflammatory cytokines. *PLoS Genetics*, 8(3).

[Ewels et al., 2016] Ewels, P., Magnusson, M., Lundin, S., and Käller, M. (2016). MultiQC : Summarize analysis results for multiple tools and samples in a single report. *Bioinformatics*, 32(19) :3047–3048.

[[F., 2012] F., ] [F., 2012] F., K. . Trim Galore : a wrapper tool around Cutadapt and FastQC to consistently apply quality and adapter trimming to FastQ files.

[Falkmer S, 1985] Falkmer S, O. Y. (1985). Comparative morphology of pancreatic islets in animals. *The Diabetic Pancreas*, Volk BW an.

[Gao et al., 2017] Gao, J., Long, B., and Wang, Z. (2017). Role of Notch signaling pathway in pancreatic cancer. *American Journal of Cancer Research*, 7(2) :173–186.

[Greenwood et al., 2007] Greenwood, A. L., Li, S., Jones, K., and Melton, D. A. (2007). Notch signaling reveals developmental plasticity of Pax4+pancreatic endocrine progenitors and shunts them to a duct fate. *Mechanisms of Development*, 124(2) :97–107.

[Grün and Van Oudenaarden, 2015] Grün, D. and Van Oudenaarden, A. (2015). Design and Analysis of Single-Cell Sequencing Experiments. *Cell*, 163(4) :799–810.

[Haghverdi et al., 2017] Haghverdi, L., Lun, A. T. L., Morgan, M. D., and Marioni, J. C. (2017). Correcting batch effects in single-cell RNA sequencing data by matching mutual nearest neighbours. *bioRxiv*, page 165118.

[Heller, 2010] Heller, R. S. (2010). The Islets of Langerhans. 654 :21–37.

[Klein et al., 2015] Klein, A. M., Mazutis, L., Weitz, D. A., Kirschner, M. W., Klein, A. M., Mazutis, L., Akartuna, I., Tallapragada, N., Veres, A., Li, V., and Peshkin, L. (2015). Droplet Barcoding for Single-Cell Transcriptomics Applied to Embryonic Stem Cells Resource Droplet Barcoding for Single-Cell Transcriptomics Applied to Embryonic Stem Cells. *Cell*, 161(5) :1187–1201.

[Klonoff, 2009] Klonoff, D. C. (2009). The increasing incidence of diabetes in the 21st century. *Journal of Diabetes Science and Technology*, 3(1) :1–2.

[Leal et al., 2009] Leal, J., Gray, A. M., and Clarke, P. M. (2009). Development of life-expectancy tables for people with type 2 diabetes. *European Heart Journal*, 30(7) :834–839.

[Leek et al., 2012] Leek, J. T., Johnson, W. E., Parker, H. S., Jaffe, A. E., and Storey, J. D. (2012). The SVA package for removing batch effects and other unwanted variation in high-throughput experiments. *Bioinformatics*, 28(6) :882–883.

[Levetan, C. and Pierce, 2013] Levetan, C. and Pierce, S. (2013). Distinctions Between the Islets of Mice and Men : Implications for New Therapies for Type 1 and 2 Diabetes. *Endocrine Practice*, 19(2) :301.

[Love et al., 2014] Love, M. I., Huber, W., and Anders, S. (2014). Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biology*, 15(12) :1–21.

[M. Enge , H. Efsun Arda , Marco Mignardi , John Beausang , Rita Bottino, K. Kim, 2015] M. Enge , H. Efsun Arda , Marco Mignardi , John Beausang , Rita Bottino, K. Kim, R. S. (2015). HHS Public Access. *Anal Chem.*, 25(4) :368–379.

[Maza, 2016] Maza, E. (2016). In papyro comparison of TMM (edgeR), RLE (DESeq2), and MRN normalization methods for a simple two-conditions-without-replicates RNA-seq experimental design. *Frontiers in Genetics*, 7(SEP) :1–8.

[Muraro et al., 2016] Muraro, M. J., Dharmadhikari, G., Grün, D., Groen, N., Dielen, T., Jansen, E., van Gurp, L., Engelse, M. A., Carlotti, F., de Koning, E. J., and van Oudenaarden, A. (2016). A Single-Cell Transcriptome Atlas of the Human Pancreas. *Cell Systems*, 3(4) :385–394.

[Nathan Lawlor et al., 2017] Nathan Lawlor, 1, ., Joshy George, 1, ., Mohan Bolisetty, 1, Romy Kursawe, 1, Lili Sun, 1, V. Sivakamasundari, 1, Ina Kycia, 1, Paul Robson, 1, 2, ., and and Michael L. Stitzel (2017). Single-cell transcriptomes identify human islet cell signatures and reveal cell-type – specific expression changes in type 2 diabetes. *Genome Research*, pages 208–222.

[Pham et al., 2017] Pham, N. C., Haibe-Kains, B., Bellot, P., Bontempi, G., and Meyer, P. E. (2017). Study of Meta-analysis Strategies for Network Inference Using Information-Theoretic Approaches. *Proceedings - International Workshop on Database and Expert Systems Applications, DEXA*, pages 76–83.

[Picelli et al., 2013] Picelli, S., Picelli, S., Björklund, Å. K., Faridani, O. R., Sagasser, S., Winberg, G., and Sandberg, R. (2013). Smart-seq2 improves yield and length in single cell-derived cDNA libraries and uses off-the-shelf reagents. *Nature Methods*, 10(SEPTEMBER) :1096–1098.

[Rahier et al., 2008] Rahier, J., Guiot, Y., Goebbels, R. M., Sempoux, C., and Henquin, J. C. (2008). Pancreatic $\beta$-cell mass in European subjects with type 2 diabetes. *Diabetes, Obesity and Metabolism*, 10 :32–42.

[Schulze et al., 2012] Schulze, S. K., Kanwar, R., Gölzenleuchter, M., Therneau, T. M., and Beutler, A. S. (2012). SERE : Single-parameter quality control and sample comparison for RNA-Seq. *BMC Genomics*, 13(1) :7–9.

[Segerstolpe et al., 2016] Segerstolpe, Å., Palasantza, A., Eliasson, P., Andersson, E. M., Andréasson, A. C., Sun, X., Picelli, S., Sabirsh, A., Clausen, M., Bjursell, M. K., Smith, D. M., Kasper, M., Ämmälä, C., and Sandberg, R. (2016). Single-Cell Transcriptome Profiling of Human Pancreatic Islets in Health and Type 2 Diabetes. *Cell Metabolism*, 24(4) :593–607.

[Seyednasrollah et al., 2013] Seyednasrollah, F., Laiho, A., and Elo, L. L. (2013). Comparison of software packages for detecting differential expression in RNA-seq studies. *Briefings in Bioinformatics*, 16(1) :59–70.

[Tang et al., 2010] Tang, F., Barbacioru, C., Bao, S., Lee, C., Nordman, E., Wang, X., Lao, K., and Surani, M. A. (2010). Tracing the derivation of embryonic stem cells from the inner cell mass by single-cell RNA-seq analysis. *Cell Stem Cell*, 6(5) :468–478.

[Tarifeño-Saldivia et al., 2017] Tarifeño-Saldivia, E., Lavergne, A., Bernard, A., Padamata, K., Bergemann, D., Voz, M. L., Manfroid, I., and Peers, B. (2017). Transcriptome analysis of pancreatic cells across distant species highlights novel important regulator genes. *BMC Biology*, 15(1) :1–19.

[Varet et al., 2016] Varet, H., Brillet-Guéguen, L., Coppée, J. Y., and Dillies, M. A. (2016). SARTools : A DESeq2- and edgeR-based R pipeline for comprehensive differential analysis of RNA-Seq data. *PLoS ONE*, 11(6) :1–8.

[Xin et al., 2016] Xin, Y., Kim, J., Okamoto, H., Ni, M., Wei, Y., Adler, C., Murphy, A. J., Yancopoulos, G. D., Lin, C., and Gromada, J. (2016). RNA Sequencing of Single Human Islet Cells Reveals Type 2 Diabetes Genes. *Cell Metabolism*, 24(4) :608–615.

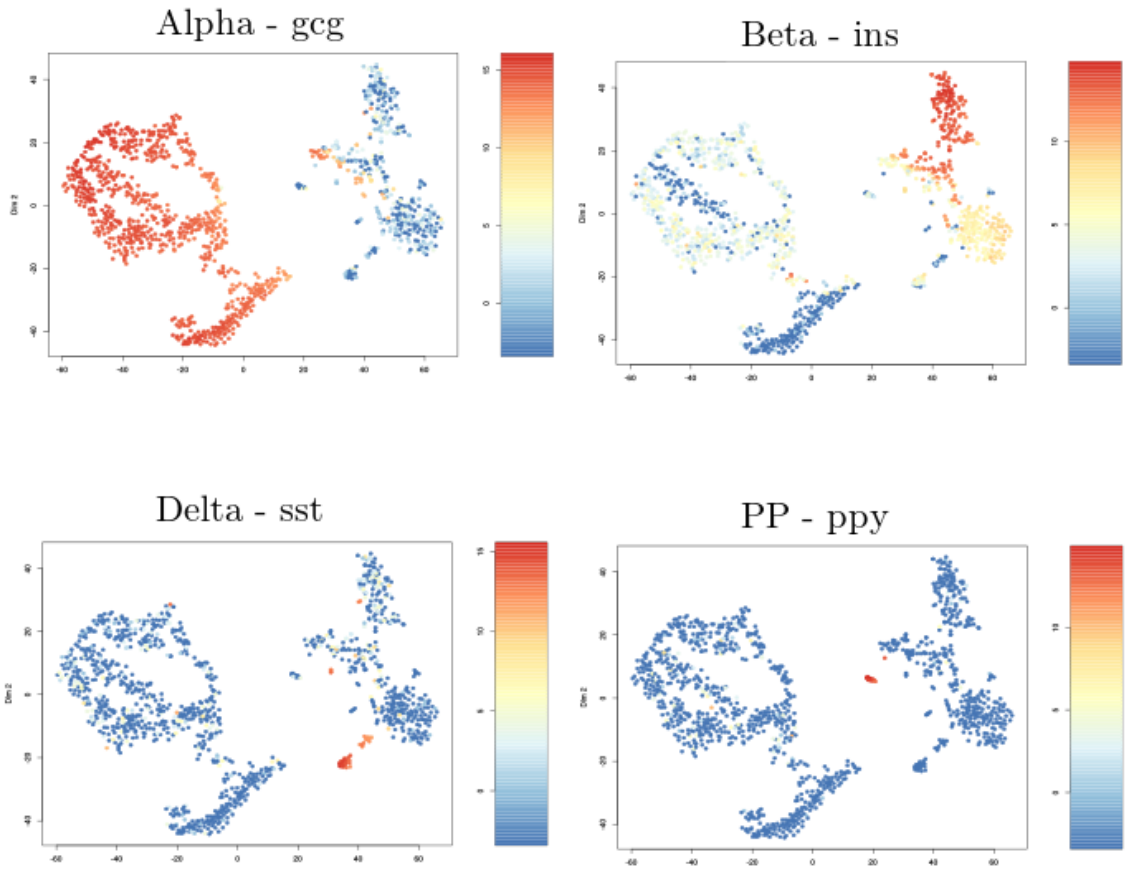


**Figure-Supplémentaire 1** – *Clustering des données endocrines de Enge avec mise en évidence des différentes populations cellulaires via le niveau d'expression de gènes marqueurs spécifiques. Les cellules alphas, bêtas, deltas et PP sont respectivement mises en évidence.*

# 6 Annexes

## 6.1 Figures supplémentaires

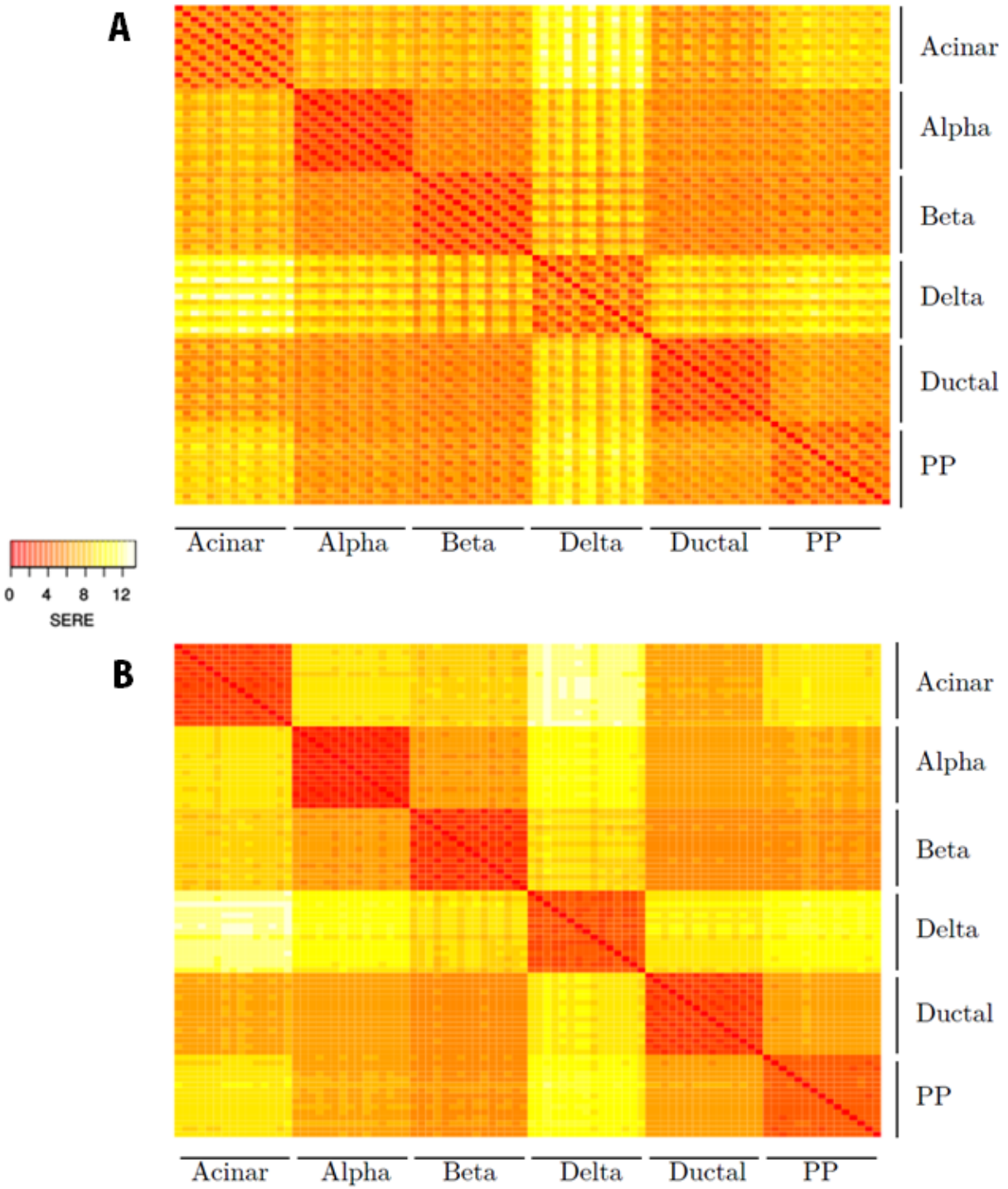


**Figure-Supplémentaire 2** – *A : Heatmap de Simple Error Ratio Estimates des réplicas des différentes études SingleCell. B : Heatmap de Simple Error Ratio Estimates des réplicas des différentes études SingleCell après une correction d'effet Batch avec le package Combat.*

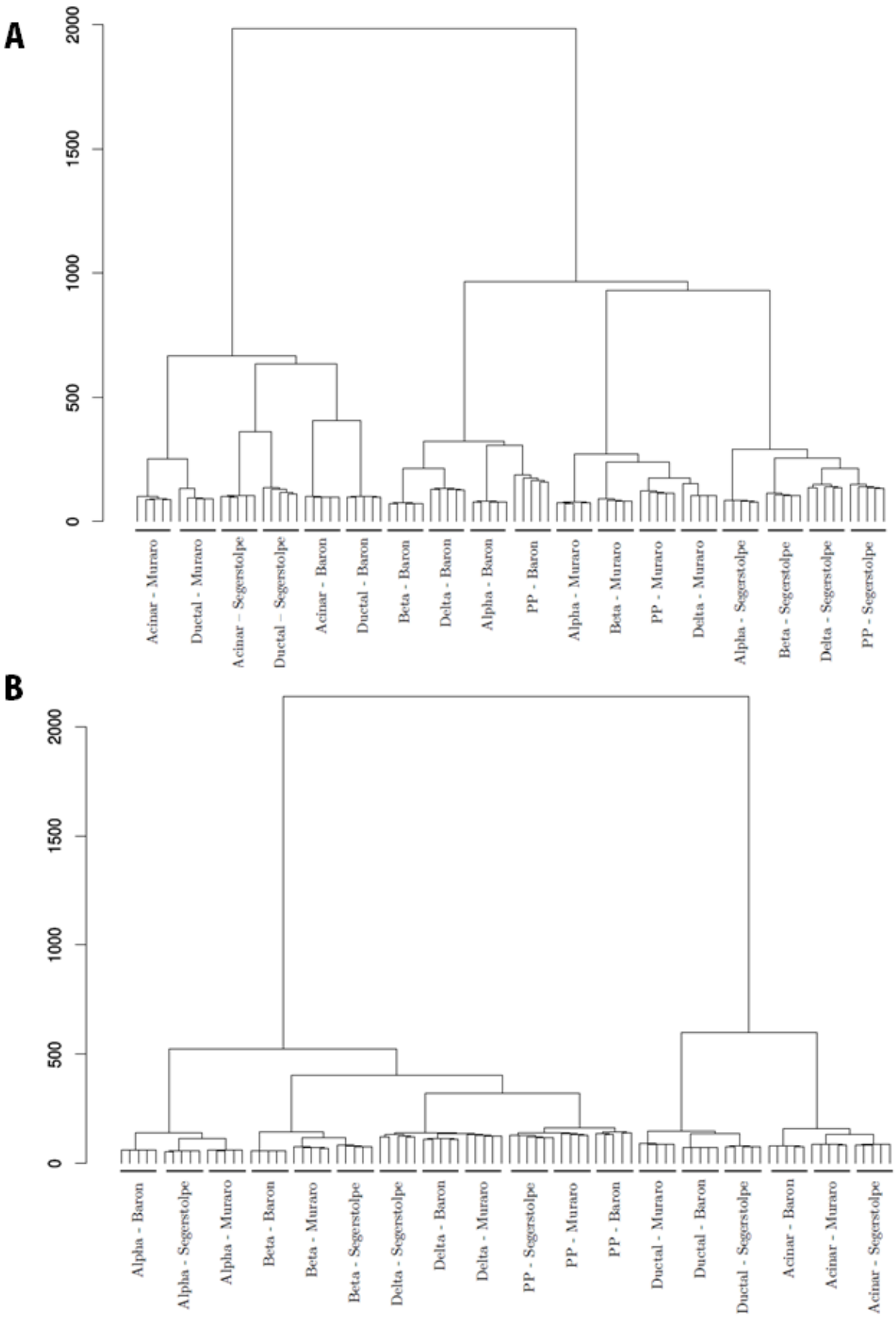


**Figure-Supplémentaire 3** – *A : Dendrogramme des réplicas des différentes études SingleCell. B : Dendrogramme des réplicas des différentes études SingleCell après une correction d'effet Batch avec le package Combat.*

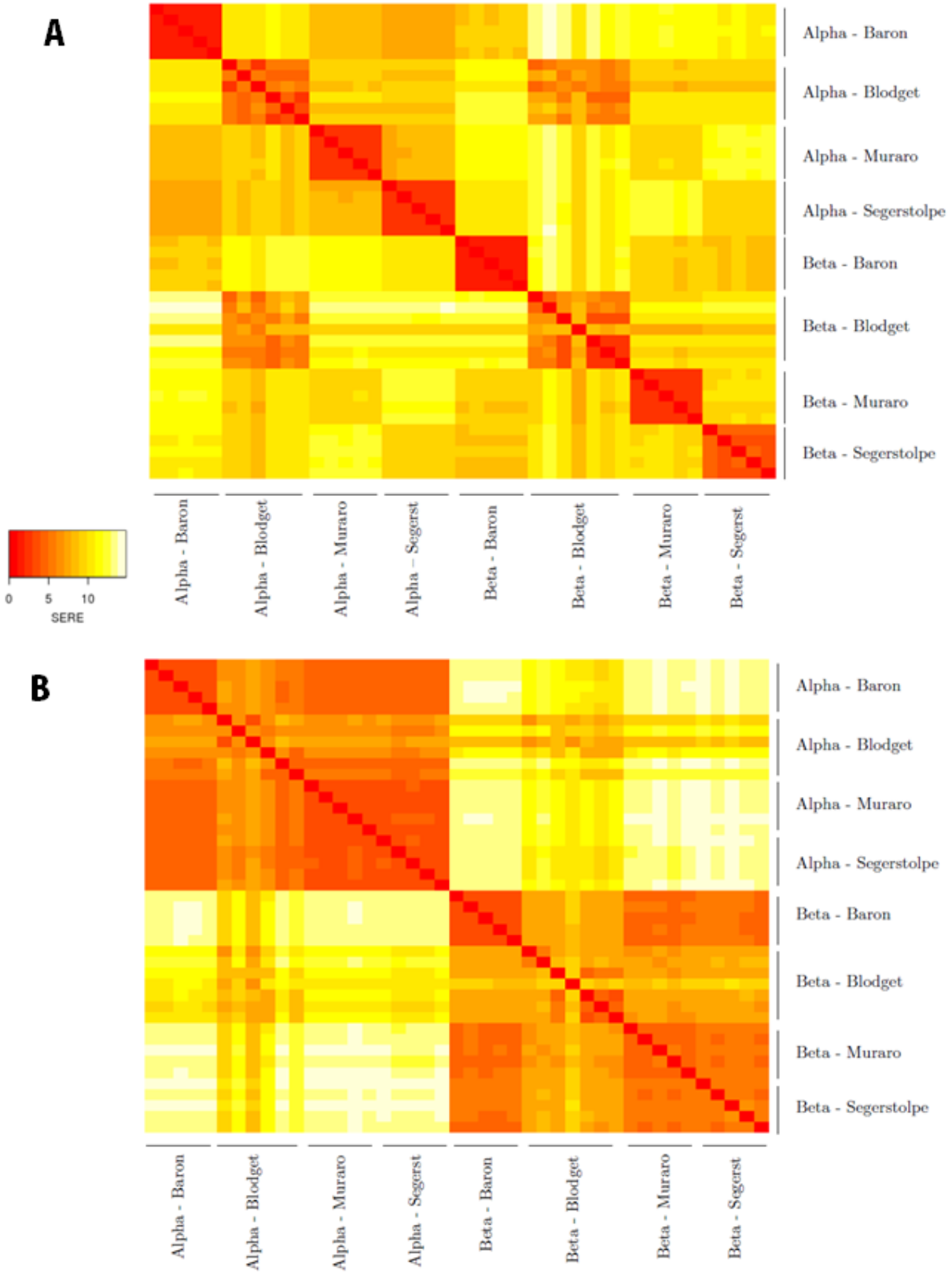


**Figure-Supplémentaire 4** – *A : Heatmap de Simple Error Ratio Estimates des réplicas des différentes études SingleCell RNAseq (Baron, Muraro et Segerstolpe) et Bulk RNAseq (Blodget). B : Heatmap de Simple Error Ratio Estimates des réplicas des différentes études SingleCell et Bulk après une correction d'effet Batch avec le package Combat.*

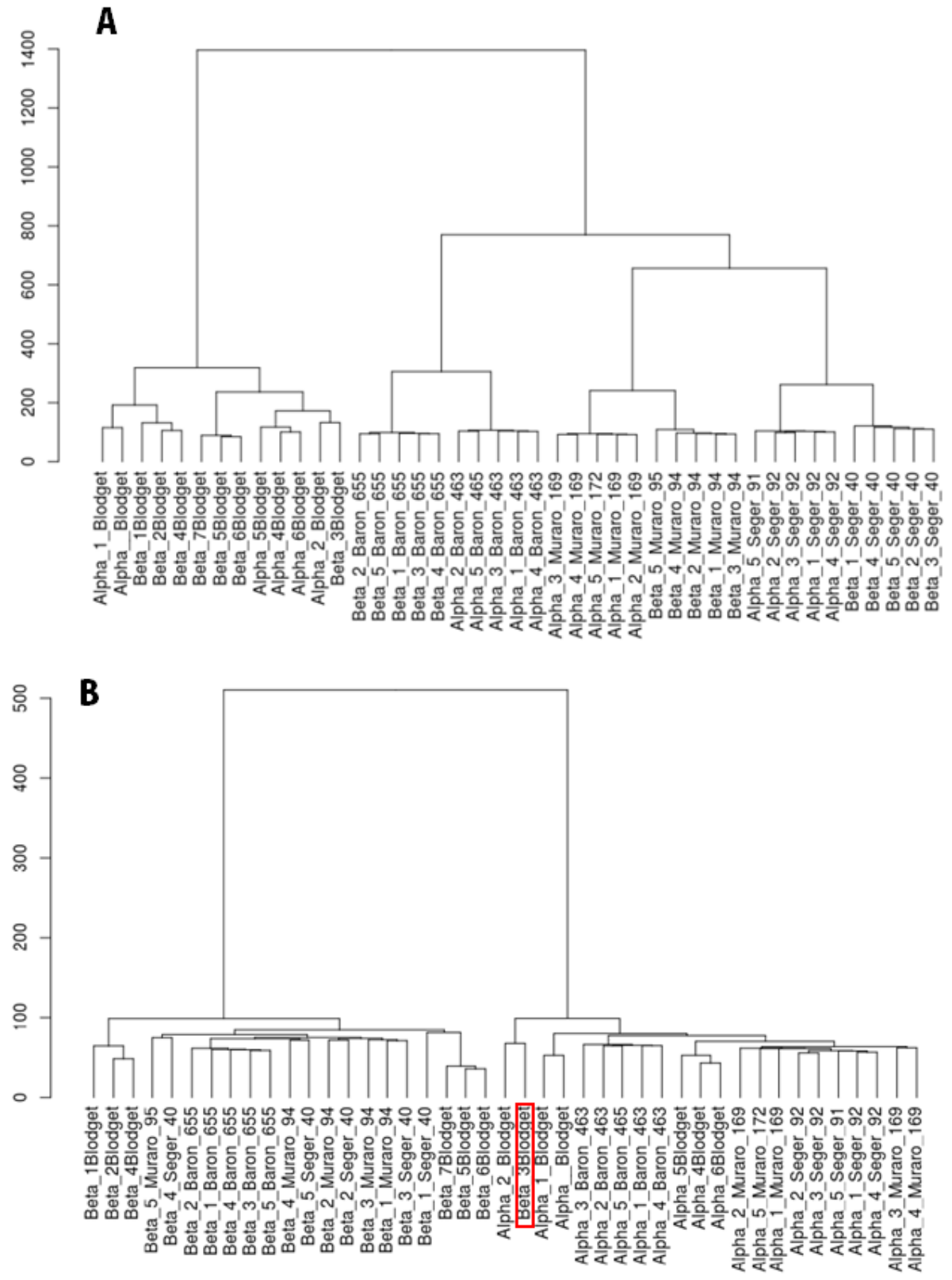


**Figure-Supplémentaire 5** – *A : Dendrogramme des réplicas des différentes études SingleCell et Bulk. B : Dendrogramme des réplicas des différentes études SingleCell et Bulk après une correction d'effet Batch avec le package Combat.*

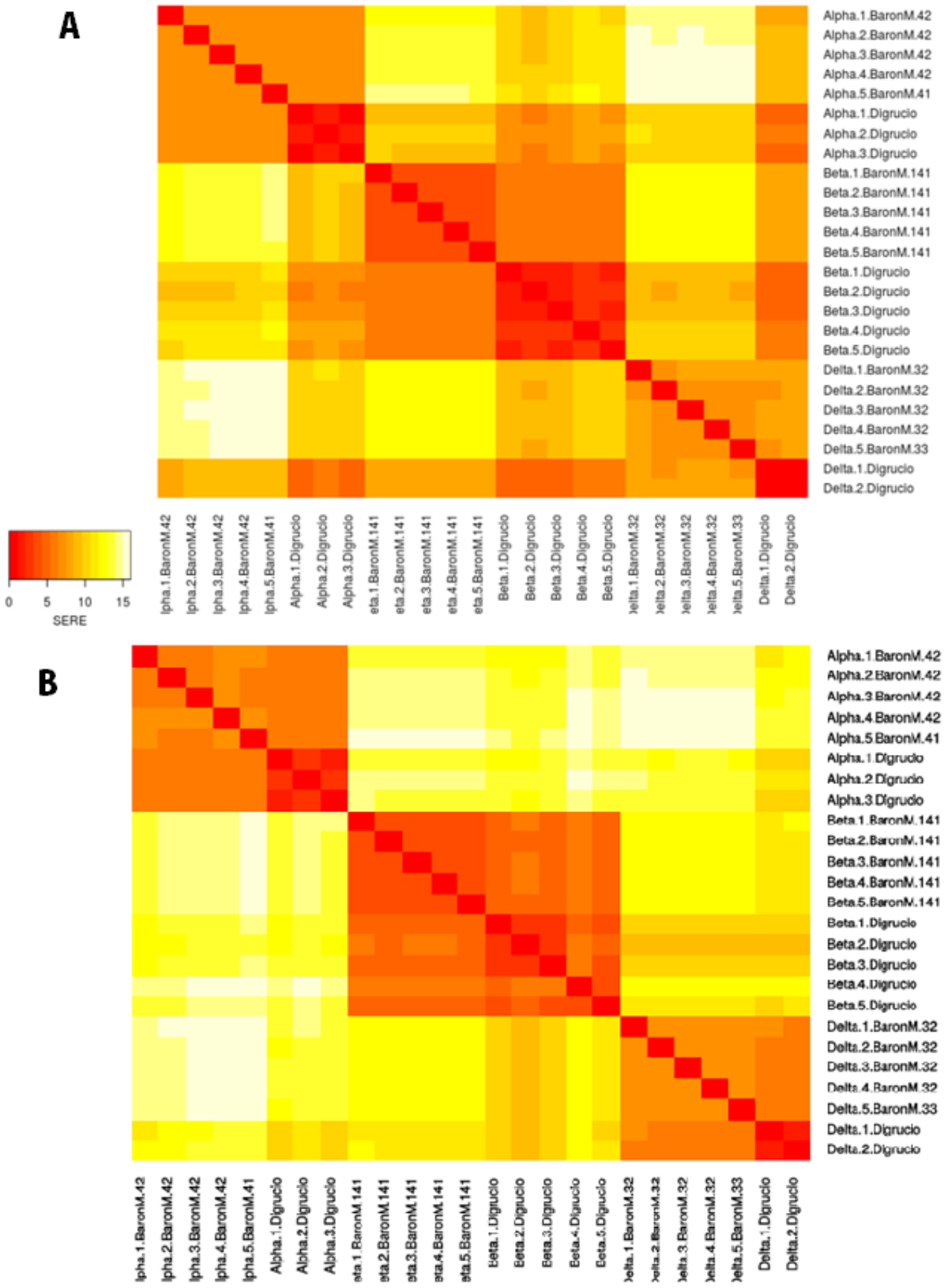


**Figure-Supplémentaire 6** – *A : Heatmap de Simple Error Ratio Estimates des réplicas SingleCell RNAseq (Baron) et Bulk RNAseq (DiGrucio). B : Heatmap de Simple Error Ratio Estimates des réplicas SingleCell et Bulk après une correction d'effet Batch avec le package Combat.*

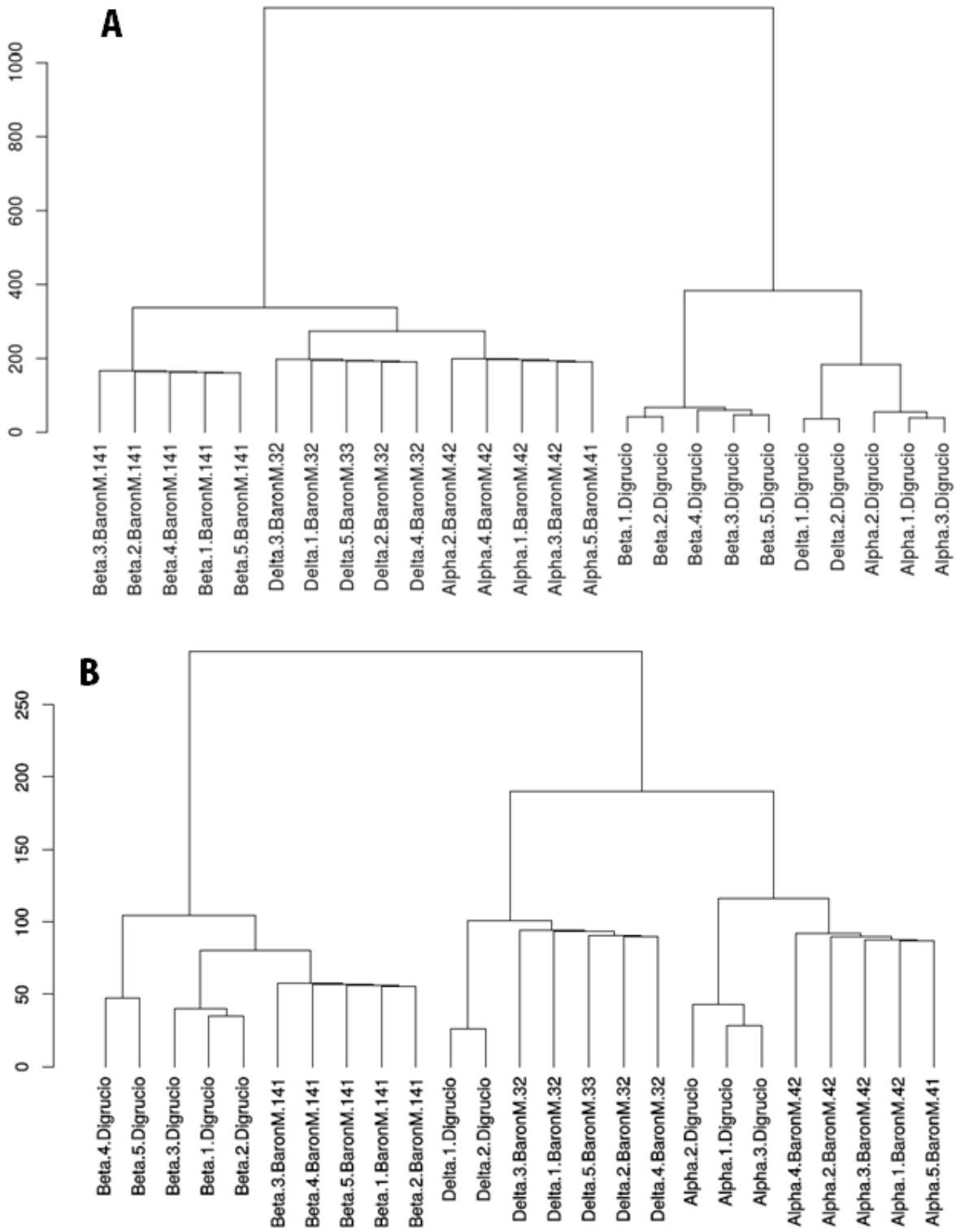


**Figure-Supplémentaire 7** – *A : Dendrogramme des réplicas des études SingleCell et Bulk. B : Dendrogramme des réplicas des études SingleCell et Bulk après une correction d'effet Batch avec le package Combat pour les données de souris.*

## 6.2 Codes Sources

### 6.2.1 EGMA

```
library("DESeq2")
library("devtools")
library("tcltk2")
library("VennDiagram")
library("ggpubr")
library("sva")
library("SARTools")
library("gplots")
library("reshape2")
library(ggplot2)
library("GeneOverlap")


#------------ Fonction de comparaison  automatique de profils transcriptomique ----------------
EGMATransComp <- function(countdata, coldata,padj = 0.01, log2fold = 2, ConversionDataFrame,batch = FALSE, cormethod = "SERE",
                          PCA =TRUE){
  crossdata <- new.env()
  EGMA_Cells_Comp <- new.env()
  assign("EGMA_Cells_Comp", EGMA_Cells_Comp, envir = crossdata)
  if (isTRUE(batch)){
    message("---------------------------------- Batch Correction ----------------------------------")
    dds <- DESeqDataSetFromMatrix(countData = countdata,
                                  colData = coldata,
                                  design = ~ CellType)
    dds <- estimateSizeFactors(dds)
    y <- counts(dds, normalized = TRUE)
    myTmpVar <- as.matrix(y)
    keep <- rowSums(myTmpVar) >= 1
    myTmpVar <- myTmpVar[keep,]

    mod0 = model.matrix(~1,data=dds$Sample_Name)
    batch = as.numeric(dds$Source)
    coreect <- ComBat(myTmpVar, batch = batch, mod = mod0)
    coreect[coreect < 0] <- 0.0
    coreect <- data.frame(coreect)
```

```
36        coreect[,1:ncol(coreect)] <- lapply(coreect[,1:ncol(coreect)], as.integer)
37        dds <- data.frame()
38        dds <- DESeqDataSetFromMatrix(countData = coreect,
39                                      colData = coldata,
40                                      design = ~ CellType)
41      }
42      else{   # simple preparation de l'objet d'entree de DESeq en absence de
            demande de correction batch
43        dds <- DESeqDataSetFromMatrix(countData = countdata,
44                                      colData = coldata,
45                                      design = ~ CellType)
46      }
47
48      # Verification que les donnees d'entrees ne melange pas plusieurs especes
49      if (length(unique.Vector(dds$Species)) > 1){
50        warning("Several species detected. Keep only one species in dataset for
              intra-species cross data or use crossspecies function!")
51      }
52      else{
53      message("---------------------------------- DeSeq2 is running
            ---------------------------------- ")
54      dds <- DESeq(dds)
55      assign("dds", dds, envir = crossdata)
56
57      if (isTRUE(PCA)){
58      message("---------------------------------- EGMApcaPLOT is running
            ---------------------------------- ")
59      counts.trans <<- assay(rlogTransformation(dds))
60      EGMApcaPLOT(counts.trans, group = dds$CellType,source = dds$Source ,n = min
            (500, nrow(counts(dds, normalized =T))))
61      assign("PCAplot",recordPlot(), envir = crossdata)
62      assign("counts.trans", counts.trans, envir = crossdata)
63      }
64
65
66        # Genere tout les scaterPLot automatiquement
67        message("---------------------------------- Correlation is running
              ---------------------------------- ")
68        #scpenv <- new.env()
69        #assign("ScatterPlots", scpenv, envir = crossdata)
70        #combi <- combn(as.integer(dds$Sample_Name),2)
```

```
71      #pb <- txtProgressBar(min = 0, max = ncol(combi) || 1, style = 3)
72      #dev.control('inhibit')
73      #for (o in 1:ncol(combi)){
74      #  dev.off()
75      #  EGMAscaterplot(counts(dds)[,c(combi[1,o],combi[2,o])], outfile = F,
            method = cormethod)
76      #  assign(paste(as.character(dds$Sample_Name[combi[1,o]]), as.character(dds$
            Sample_Name[combi[2,o]]), sep = "vs"),
77      #         recordPlot(), envir = scpenv)
78      #  setTxtProgressBar(pb, o)
79      #}
80      #close(pb)
81
82
83
84      # Selection du mode e corelation et production automatique des heatmaps
85    if (cormethod == "SERE"){
86      cortab <- tabSERE(counts(dds, normalized =T))
87      assign("Correlation_Table", cortab, envir = crossdata)
88    }
89    if (cormethod == "pearson" || cormethod == "spearman" || cormethod == "kendall
        "){
90     cortab <- cor(counts(dds, normalized = TRUE), method = cormethod, use = "
          complete.obs" )
91     assign("Correlation_Table", cortab, envir = crossdata)
92    }
93      myheat2(as.matrix(get( "Correlation_Table", envir =crossdata )), scale = "
          none",
94              trace = "none", density.info = "none", Colv = NA, key.title = "SERE"
                  , keysize = 1, margins = c(7,8),
95              key.xlab = "SERE")
96
97    # Production des listes de genes enrichis en mode intermediaire
98      message("------------------------------------ EGMA is running (Transcriptomic
          Comparison mode) ---------------------- ")
99    GeneralMarkers <- markerList(dds,padj,log2fold)
100   traductor(GeneralMarkers,myconv)
101   assign("GeneralMarkers", GeneralMarkers, envir = crossdata)
102   return(crossdata)
103   message("DONE")
104   }
```

```
105 }
106
107
108
109 EGMAMetaAnalysis <- function(countdata, coldata,padj = 0.01, log2fold = 2,
        ConversionDataFrame,batch = FALSE){
110   crossdata <- new.env()
111   EGMA_Cells_Comp <- new.env()
112   assign("EGMA_Cells_Comp", EGMA_Cells_Comp, envir = crossdata)
113   if (isTRUE(batch)){
114     message("------------------------------------ Batch Correction
          ------------------------------------ ")
115     dds <- DESeqDataSetFromMatrix(countData = countdata,
116                                   colData = coldata,
117                                   design = ~ CellType)
118     dds <- estimateSizeFactors(dds)
119     y <- counts(dds, normalized = TRUE)
120     myTmpVar <- as.matrix(y)
121     keep <- rowSums(myTmpVar) >= 1
122     myTmpVar <- myTmpVar[keep,]
123
124     mod0 = model.matrix(~1,data=dds$Sample_Name)
125     batch = as.numeric(dds$Source)
126     coreect <- ComBat(myTmpVar, batch = batch, mod = mod0)
127     coreect[coreect < 0] <- 0.0
128     coreect <- ComBat(coreect,batch = batch, mod = mod0)
129     coreect[coreect < 0] <- 0.0
130     coreect <- data.frame(coreect)
131     coreect[,1:ncol(coreect)] <- lapply(coreect[,1:ncol(coreect)], as.integer)
132     dds <- data.frame()
133     dds <- DESeqDataSetFromMatrix(countData = coreect,
134                                   colData = coldata,
135                                   design = ~ CellType)
136
137     message("------------------------------------ EGMA is running (Meta-analysis mode)
          ---------------------- ")
138     # Defini le nombre d'etude a analyser
139     for (i in 1:length(unique.Vector(dds$Source))){
140       mymin <- min(which(as.integer(dds$Source) == i))
141       mymax <- max(which(as.integer(dds$Source) == i))
142
```

```
143
144       tmpcount <- coreect[,mymin:mymax]
145       tmpcoldata <- coldata[mymin:mymax,]
146
147       tmpdds <- DESeqDataSetFromMatrix(countData = tmpcount,
148                                        colData = tmpcoldata,
149                                        design = ~ CellType)
150       tmpdds <- DESeq(tmpdds)
151       assign(as.character(unique.Vector(dds$Source)[i]),tmpdds, envir = crossdata)
152       tmpMarker <- markerList(tmpdds,padj,log2fold)
153       traductor(tmpMarker,myconv)
154       assign(paste(as.character(unique.Vector(dds$Source)[i]),"EG",sep = ""),
              tmpMarker, envir = crossdata)
155     }
156   }
157   else {
158     dds <- DESeqDataSetFromMatrix(countData = countdata,
159                                   colData = coldata,
160                                   design = ~ CellType)
161
162     message("------------------------------------ EGMA is running (Meta-analysis mode)
          ---------------------- ")
163     # Defini le nombre d'etude a analyser
164     for (i in 1:length(unique.Vector(dds$Source))){
165       mymin <- min(which(as.integer(dds$Source) == i))
166       mymax <- max(which(as.integer(dds$Source) == i))
167
168
169       tmpcount <- countdata[,mymin:mymax]
170       tmpcoldata <- coldata[mymin:mymax,]
171
172       tmpdds <- DESeqDataSetFromMatrix(countData = tmpcount,
173                                        colData = tmpcoldata,
174                                        design = ~ CellType)
175       tmpdds <- DESeq(tmpdds)
176       # Genere les listes de genes enrichis pour tout les types cellulaires
            detectes
177       assign(as.character(dds$Source)[mymin],tmpdds, envir = crossdata)
178       tmpMarker <- markerList(tmpdds,padj,log2fold)
179       traductor(tmpMarker,myconv)
```

```
180         assign(paste(as.character(dds$Source)[mymin],"EG",sep = ""), tmpMarker,
                envir = crossdata)
181         }
182     }
183
184     # Genere tout les diagramme de Ven entre les etudes pour tout les types
            cellulaires detectes
185     message("---------------------------------- CrossData is running
            ---------------------- ")
186     if (length(unique.Vector(dds$Source)) == 2){
187       for (i in 1:length(unique.Vector(dds$CellType))){
188         kill <- new.env()
189         assign(paste(unique.Vector(dds$CellType)[i]), kill, envir = EGMA_Cells_Comp)
190         tmp1 <- get(paste(as.character(unique.Vector(dds$Source)[1]),"EG",sep = ""),
                envir = crossdata)
191         tmp2 <- get(paste(as.character(unique.Vector(dds$Source)[2]),"EG",sep = ""),
                envir = crossdata)
192
193         list1 <- get(as.character(unique.Vector(dds$CellType)[i]), envir = tmp1)
194         list2 <- get(as.character(unique.Vector(dds$CellType)[i]), envir = tmp2)
195
196         tmpcomp <- comparator(list1,list2)
197         assign(paste(as.character(unique.Vector(dds$CellType)[i]), "EGMA_Comp",
198                      sep = "_"), tmpcomp, envir = get(paste(unique.Vector(dds$
                         CellType)[i]), envir = EGMA_Cells_Comp))
199         x <- plot.new()
200         dev.off()
201         x <- plot.new()
202         draw.pairwise.venn(nrow(list1),nrow(list2), nrow(tmpcomp),
203                            c(as.character(unique.Vector(dds$Source)[1]),as.character
                               (unique.Vector(dds$Source)[2])),
204                            lty = rep("blank",2), fill = c("skyblue", "orange"),
205                            alpha = rep(0.5, 2), cat.pos = c(210,150), cat.dist = rep
                               (0.025, 2),cex = 2,cat.cex = 2)
206         assign(paste(as.character(unique.Vector(dds$CellType)[i]),"VenPlot", sep = "
                _"), recordPlot(x), envir = crossdata)
207       }
208     }
209     if (length(unique.Vector(dds$Source)) == 3){
210       for (i in 1:length(unique.Vector(dds$CellType))){
211         kill <- new.env()
```

```
212       assign(paste(unique.Vector(dds$CellType)[i]), kill, envir = EGMA_Cells_Comp)
213
214       tmp1 <- get(paste(as.character(unique.Vector(dds$Source)[1]),"EG",sep = ""),
              envir = crossdata)
215       tmp2 <- get(paste(as.character(unique.Vector(dds$Source)[2]),"EG",sep = ""),
              envir = crossdata)
216       tmp3 <- get(paste(as.character(unique.Vector(dds$Source)[3]),"EG",sep = ""),
              envir = crossdata)
217
218       list1 <- get(as.character(unique.Vector(dds$CellType)[i]), envir = tmp1)
219       list2 <- get(as.character(unique.Vector(dds$CellType)[i]), envir = tmp2)
220       list3 <- get(as.character(unique.Vector(dds$CellType)[i]), envir = tmp3)
221
222       tmpcomp1 <- comparator(list1,list2)
223       assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
              unique.Vector(dds$Source)[1]), "vs",
224                 as.character(unique.Vector(dds$Source)[2]), sep = "_"),
                       tmpcomp1, envir = get(paste(unique.Vector(dds$CellType)[i
                       ]), envir = EGMA_Cells_Comp))
225       tmpcomp2 <- comparator(list1,list3)
226       assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
              unique.Vector(dds$Source)[1]), "vs",
227                 as.character(unique.Vector(dds$Source)[3]), sep = "_"),
                       tmpcomp2, envir = get(paste(unique.Vector(dds$CellType)[i
                       ]), envir = EGMA_Cells_Comp))
228       tmpcomp3 <- comparator(list2,list3)
229       assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
              unique.Vector(dds$Source)[2]), "vs",
230                 as.character(unique.Vector(dds$Source)[3]), sep = "_"),
                       tmpcomp3, envir = get(paste(unique.Vector(dds$CellType)[i
                       ]), envir = EGMA_Cells_Comp))
231
232       tmpcomp4 <- comparator(tmpcomp1,tmpcomp2)
233       assign(paste( as.character(unique.Vector(dds$CellType)[i]), "EGMA_Comp", sep
              = "_"),
234          tmpcomp4, envir = get(paste(unique.Vector(dds$CellType)[i]), envir =
                 EGMA_Cells_Comp))
235
236
237       x <- plot.new()
238       dev.off()
```

```
239       x <- plot.new()
240       draw.triple.venn(area1 = nrow(list1), area2 = nrow(list2), area3 = nrow(
              list3), n12 = nrow(tmpcomp1),
241                        n23 = nrow(tmpcomp3), n13 = nrow(tmpcomp2), n123 = nrow(
                               tmpcomp4),
242                        category = c(as.character(unique.Vector(dds$Source)[1]),as.
                               character(unique.Vector(dds$Source)[2]),
243                                     as.character(unique.Vector(dds$Source)[3])),
244                        lty = "blank", fill = c("skyblue", "orange", "mediumorchid"
                               ),cex = 2,cat.cex = 2)
245       assign(paste(as.character(unique.Vector(dds$CellType)[i]),"VenPlot", sep = "
              _"), recordPlot(x), envir = crossdata)
246     }
247   }
248   if (length(unique.Vector(dds$Source)) == 4){
249     for (i in 1:length(unique.Vector(dds$CellType))){
250       kill <- new.env()
251       assign(paste(unique.Vector(dds$CellType)[i]), kill, envir = EGMA_Cells_Comp)
252
253       tmp1 <- get(paste(as.character(unique.Vector(dds$Source)[1]),"EG",sep = ""),
              envir = crossdata)
254       tmp2 <- get(paste(as.character(unique.Vector(dds$Source)[2]),"EG",sep = ""),
              envir = crossdata)
255       tmp3 <- get(paste(as.character(unique.Vector(dds$Source)[3]),"EG",sep = ""),
              envir = crossdata)
256       tmp4 <- get(paste(as.character(unique.Vector(dds$Source)[4]),"EG",sep = ""),
              envir = crossdata)
257
258
259       list1 <- get(as.character(unique.Vector(dds$CellType)[i]), envir = tmp1)
260       list2 <- get(as.character(unique.Vector(dds$CellType)[i]), envir = tmp2)
261       list3 <- get(as.character(unique.Vector(dds$CellType)[i]), envir = tmp3)
262       list4 <- get(as.character(unique.Vector(dds$CellType)[i]), envir = tmp4)
263
264       tmpcomp1 <- comparator(list1,list2)
265       assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
              unique.Vector(dds$Source)[1]), "vs",
266                     as.character(unique.Vector(dds$Source)[2]), sep = "_"),
                            tmpcomp1, envir = get(paste(unique.Vector(dds$CellType)[i
                            ]), envir = EGMA_Cells_Comp))
267       tmpcomp2 <- comparator(list1,list3)
```

```
268         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[1]), "vs",
269                       as.character(unique.Vector(dds$Source)[3]), sep = "_"),
                              tmpcomp2, envir = get(paste(unique.Vector(dds$CellType)[i
                              ]), envir = EGMA_Cells_Comp))
270         tmpcomp3 <- comparator(list1,list4)
271         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[1]), "vs",
272                       as.character(unique.Vector(dds$Source)[4]), sep = "_"),
                              tmpcomp3, envir = get(paste(unique.Vector(dds$CellType)[i
                              ]), envir = EGMA_Cells_Comp))
273         tmpcomp4 <- comparator(list2,list3)
274         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[2]), "vs",
275                       as.character(unique.Vector(dds$Source)[3]), sep = "_"),
                              tmpcomp4, envir = get(paste(unique.Vector(dds$CellType)[i
                              ]), envir = EGMA_Cells_Comp))
276         tmpcomp5 <- comparator(list2,list4)
277         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[2]), "vs",
278                       as.character(unique.Vector(dds$Source)[4]), sep = "_"),
                              tmpcomp5, envir = get(paste(unique.Vector(dds$CellType)[i
                              ]), envir = EGMA_Cells_Comp))
279         tmpcomp6 <- comparator(list3,list4)
280         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[3]), "vs",
281                       as.character(unique.Vector(dds$Source)[4]), sep = "_"),
                              tmpcomp6, envir = get(paste(unique.Vector(dds$CellType)[i
                              ]), envir = EGMA_Cells_Comp))
282
283         tmpcomp7 <- comparator(tmpcomp1,tmpcomp2)
284         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[1]), "vs",
285                       as.character(unique.Vector(dds$Source)[2]), "vs",as.character(
                              unique.Vector(dds$Source)[3]),
286                       sep = "_"), tmpcomp7, envir = get(paste(unique.Vector(dds$
                              CellType)[i]), envir = EGMA_Cells_Comp))
287         tmpcomp8 <- comparator(tmpcomp1,tmpcomp4)
288         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[1]), "vs",
```

```
289                     as.character(unique.Vector(dds$Source)[2]), "vs",as.character(
                            unique.Vector(dds$Source)[4]),
290                     sep = "_"), tmpcomp8, envir = get(paste(unique.Vector(dds$
                            CellType)[i]), envir = EGMA_Cells_Comp))
291         tmpcomp9 <- comparator(tmpcomp2,tmpcomp4)
292         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[1]), "vs",
293                     as.character(unique.Vector(dds$Source)[3]), "vs",as.character(
                            unique.Vector(dds$Source)[4]),
294                     sep = "_"), tmpcomp9, envir = get(paste(unique.Vector(dds$
                            CellType)[i]), envir = EGMA_Cells_Comp))
295         tmpcomp10 <- comparator(tmpcomp4,tmpcomp5)
296         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[2]), "vs",
297                     as.character(unique.Vector(dds$Source)[3]), "vs",as.character(
                            unique.Vector(dds$Source)[4]),
298                     sep = "_"), tmpcomp10, envir = get(paste(unique.Vector(dds$
                            CellType)[i]), envir = EGMA_Cells_Comp))
299
300         tmpcomp11 <- comparator(tmpcomp8,tmpcomp10)
301         assign(paste( as.character(unique.Vector(dds$CellType)[i]), "EGMA_Comp", sep
                = "_"),
302             tmpcomp11, envir = get(paste(unique.Vector(dds$CellType)[i]), envir =
                    EGMA_Cells_Comp))
303
304         x <- plot.new()
305         dev.off()
306         x <- plot.new()
307         draw.quad.venn(nrow(list1), nrow(list2), nrow(list3), nrow(list4), nrow(
                tmpcomp1), nrow(tmpcomp2), nrow(tmpcomp3),
308                     nrow(tmpcomp4), nrow(tmpcomp5),nrow(tmpcomp6), nrow(tmpcomp7)
                            , nrow(tmpcomp8), nrow(tmpcomp9),
309                     nrow(tmpcomp10), nrow(tmpcomp11),
310                     category = c(as.character(unique.Vector(dds$Source)[1]),as.
                            character(unique.Vector(dds$Source)[2]),
311                                 as.character(unique.Vector(dds$Source)[3]),as.
                                        character(unique.Vector(dds$Source)[4])),
312                     lty = "blank", fill = c("skyblue", "orange", "mediumorchid","
                            pink1"),cex = 2,cat.cex = 2)
313         assign(paste(as.character(unique.Vector(dds$CellType)[i]),"VenPlot", sep = "
                _"), recordPlot(x), envir = crossdata)
```

```
314     }
315   }
316   if (length(unique.Vector(dds$Source)) == 5){
317     for (i in 1:length(unique.Vector(dds$CellType))){
318       kill <- new.env()
319       assign(paste(unique.Vector(dds$CellType)[i]), kill, envir = EGMA_Cells_Comp)
320
321       tmp1 <- get(paste(as.character(unique.Vector(dds$Source)[1]),"EG",sep = ""),
              envir = crossdata)
322       tmp2 <- get(paste(as.character(unique.Vector(dds$Source)[2]),"EG",sep = ""),
              envir = crossdata)
323       tmp3 <- get(paste(as.character(unique.Vector(dds$Source)[3]),"EG",sep = ""),
              envir = crossdata)
324       tmp4 <- get(paste(as.character(unique.Vector(dds$Source)[4]),"EG",sep = ""),
              envir = crossdata)
325       tmp5 <- get(paste(as.character(unique.Vector(dds$Source)[5]),"EG",sep = ""),
              envir = crossdata)
326
327
328       list1 <- get(as.character(unique.Vector(dds$CellType)[i]), envir = tmp1)
329       list2 <- get(as.character(unique.Vector(dds$CellType)[i]), envir = tmp2)
330       list3 <- get(as.character(unique.Vector(dds$CellType)[i]), envir = tmp3)
331       list4 <- get(as.character(unique.Vector(dds$CellType)[i]), envir = tmp4)
332       list5 <- get(as.character(unique.Vector(dds$CellType)[i]), envir = tmp5)
333
334       tmpcomp1 <- comparator(list1,list2)
335       assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
            unique.Vector(dds$Source)[1]), "vs",
336                     as.character(unique.Vector(dds$Source)[2]), sep = "_"),
                          tmpcomp1, envir = get(paste(unique.Vector(dds$CellType)[i
                          ]), envir = EGMA_Cells_Comp))
337       tmpcomp2 <- comparator(list1,list3)
338       assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
            unique.Vector(dds$Source)[1]), "vs",
339                     as.character(unique.Vector(dds$Source)[3]), sep = "_"),
                          tmpcomp2, envir = get(paste(unique.Vector(dds$CellType)[i
                          ]), envir = EGMA_Cells_Comp))
340       tmpcomp3 <- comparator(list1,list4)
341       assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
            unique.Vector(dds$Source)[1]), "vs",
```

```
342                     as.character(unique.Vector(dds$Source)[4]), sep = "_"),
                            tmpcomp3, envir = get(paste(unique.Vector(dds$CellType)[i
                            ]), envir = EGMA_Cells_Comp))
343         tmpcomp4 <- comparator(list1,list5)
344         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[1]), "vs",
345                     as.character(unique.Vector(dds$Source)[5]), sep = "_"),
                            tmpcomp4, envir = get(paste(unique.Vector(dds$CellType)[i
                            ]), envir = EGMA_Cells_Comp))
346         tmpcomp5 <- comparator(list2,list3)
347         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[2]), "vs",
348                     as.character(unique.Vector(dds$Source)[3]), sep = "_"),
                            tmpcomp5, envir = get(paste(unique.Vector(dds$CellType)[i
                            ]), envir = EGMA_Cells_Comp))
349         tmpcomp6 <- comparator(list2,list4)
350         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[2]), "vs",
351                     as.character(unique.Vector(dds$Source)[4]), sep = "_"),
                            tmpcomp6, envir = get(paste(unique.Vector(dds$CellType)[i
                            ]), envir = EGMA_Cells_Comp))
352         tmpcomp7 <- comparator(list2,list5)
353         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[2]), "vs",
354                     as.character(unique.Vector(dds$Source)[5]), sep = "_"),
                            tmpcomp7, envir = get(paste(unique.Vector(dds$CellType)[i
                            ]), envir = EGMA_Cells_Comp))
355         tmpcomp8 <- comparator(list3,list4)
356         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[3]), "vs",
357                     as.character(unique.Vector(dds$Source)[4]), sep = "_"),
                            tmpcomp8, envir = get(paste(unique.Vector(dds$CellType)[i
                            ]), envir = EGMA_Cells_Comp))
358         tmpcomp9 <- comparator(list3,list5)
359         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[3]), "vs",
360                     as.character(unique.Vector(dds$Source)[5]), sep = "_"),
                            tmpcomp9, envir = get(paste(unique.Vector(dds$CellType)[i
                            ]), envir = EGMA_Cells_Comp))
361         tmpcomp10 <- comparator(list4,list5)
```

```
362         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[4]), "vs",
363                       as.character(unique.Vector(dds$Source)[5]), sep = "_"),
                              tmpcomp10, envir = get(paste(unique.Vector(dds$CellType)[i
                              ]), envir = EGMA_Cells_Comp))
364
365
366         ctmpcomp1 <- comparator(tmpcomp1,tmpcomp2) #123
367         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[1]), "vs",
368                       as.character(unique.Vector(dds$Source)[2]), "vs",as.character(
                              unique.Vector(dds$Source)[3]),
369                       sep = "_"), ctmpcomp1, envir = get(paste(unique.Vector(dds$
                              CellType)[i]), envir = EGMA_Cells_Comp))
370         ctmpcomp2 <- comparator(tmpcomp1,tmpcomp3) #124
371         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[1]), "vs",
372                       as.character(unique.Vector(dds$Source)[2]), "vs",as.character(
                              unique.Vector(dds$Source)[4]),
373                       sep = "_"), ctmpcomp2, envir = get(paste(unique.Vector(dds$
                              CellType)[i]), envir = EGMA_Cells_Comp))
374         ctmpcomp3 <- comparator(tmpcomp1,tmpcomp4) #125
375         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[1]), "vs",
376                       as.character(unique.Vector(dds$Source)[2]), "vs",as.character(
                              unique.Vector(dds$Source)[5]),
377                       sep = "_"), ctmpcomp3, envir = get(paste(unique.Vector(dds$
                              CellType)[i]), envir = EGMA_Cells_Comp))
378         ctmpcomp4 <- comparator(tmpcomp2,tmpcomp3) #134
379         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[1]), "vs",
380                       as.character(unique.Vector(dds$Source)[3]), "vs",as.character(
                              unique.Vector(dds$Source)[4]),
381                       sep = "_"), ctmpcomp4, envir = get(paste(unique.Vector(dds$
                              CellType)[i]), envir = EGMA_Cells_Comp))
382         ctmpcomp5 <- comparator(tmpcomp2,tmpcomp4) #135
383         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[1]), "vs",
384                       as.character(unique.Vector(dds$Source)[3]), "vs",as.character(
                              unique.Vector(dds$Source)[5]),
```

```
385                     sep = "_"), ctmpcomp5, envir = get(paste(unique.Vector(dds$
                            CellType)[i]), envir = EGMA_Cells_Comp))
386         ctmpcomp6 <- comparator(tmpcomp3,tmpcomp4) #145
387         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[1]), "vs",
388                     as.character(unique.Vector(dds$Source)[4]), "vs",as.character(
                            unique.Vector(dds$Source)[5]),
389                     sep = "_"), ctmpcomp6, envir = get(paste(unique.Vector(dds$
                            CellType)[i]), envir = EGMA_Cells_Comp))
390         ctmpcomp7 <- comparator(tmpcomp5,tmpcomp6) #234
391         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[2]), "vs",
392                     as.character(unique.Vector(dds$Source)[3]), "vs",as.character(
                            unique.Vector(dds$Source)[4]),
393                     sep = "_"), ctmpcomp7, envir = get(paste(unique.Vector(dds$
                            CellType)[i]), envir = EGMA_Cells_Comp))
394         ctmpcomp8 <- comparator(tmpcomp5,tmpcomp7) #235
395         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[2]), "vs",
396                     as.character(unique.Vector(dds$Source)[3]), "vs",as.character(
                            unique.Vector(dds$Source)[5]),
397                     sep = "_"), ctmpcomp8, envir = get(paste(unique.Vector(dds$
                            CellType)[i]), envir = EGMA_Cells_Comp))
398         ctmpcomp9 <- comparator(tmpcomp6,tmpcomp7) #245
399         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[2]), "vs",
400                     as.character(unique.Vector(dds$Source)[4]), "vs",as.character(
                            unique.Vector(dds$Source)[5]),
401                     sep = "_"), ctmpcomp9, envir = get(paste(unique.Vector(dds$
                            CellType)[i]), envir = EGMA_Cells_Comp))
402         ctmpcomp10 <- comparator(tmpcomp8,tmpcomp9)#345
403         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[3]), "vs",
404                     as.character(unique.Vector(dds$Source)[4]), "vs",as.character(
                            unique.Vector(dds$Source)[5]),
405                     sep = "_"), ctmpcomp10, envir = get(paste(unique.Vector(dds$
                            CellType)[i]), envir = EGMA_Cells_Comp))
406
407
408         sctmpcomp1 <- comparator(ctmpcomp1,ctmpcomp7)  #1234
```

```
409         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[1]), "vs",
410                       as.character(unique.Vector(dds$Source)[2]), "vs",as.character(
                              unique.Vector(dds$Source)[3]),
411                       "vs", as.character(unique.Vector(dds$Source)[4]),
412                       sep = "_"), sctmpcomp1, envir = get(paste(unique.Vector(dds$
                              CellType)[i]), envir = EGMA_Cells_Comp))
413         sctmpcomp2 <- comparator(ctmpcomp1,ctmpcomp8)  #1235
414         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[1]), "vs",
415                       as.character(unique.Vector(dds$Source)[2]), "vs",as.character(
                              unique.Vector(dds$Source)[3]),
416                       "vs", as.character(unique.Vector(dds$Source)[5]),
417                       sep = "_"), sctmpcomp2, envir = get(paste(unique.Vector(dds$
                              CellType)[i]), envir = EGMA_Cells_Comp))
418         sctmpcomp3 <- comparator(ctmpcomp2,ctmpcomp9)  #1245
419         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[1]), "vs",
420                       as.character(unique.Vector(dds$Source)[2]), "vs",as.character(
                              unique.Vector(dds$Source)[4]),
421                       "vs", as.character(unique.Vector(dds$Source)[5]),
422                       sep = "_"), sctmpcomp3, envir = get(paste(unique.Vector(dds$
                              CellType)[i]), envir = EGMA_Cells_Comp))
423         sctmpcomp4 <- comparator(ctmpcomp4,ctmpcomp10) #1345
424         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[1]), "vs",
425                       as.character(unique.Vector(dds$Source)[3]), "vs",as.character(
                              unique.Vector(dds$Source)[4]),
426                       "vs", as.character(unique.Vector(dds$Source)[5]),
427                       sep = "_"), sctmpcomp4, envir = get(paste(unique.Vector(dds$
                              CellType)[i]), envir = EGMA_Cells_Comp))
428         sctmpcomp5 <- comparator(ctmpcomp7,ctmpcomp10) #2345
429         assign(paste( as.character(unique.Vector(dds$CellType)[i]), as.character(
                unique.Vector(dds$Source)[2]), "vs",
430                       as.character(unique.Vector(dds$Source)[3]), "vs",as.character(
                              unique.Vector(dds$Source)[4]),
431                       "vs", as.character(unique.Vector(dds$Source)[5]),
432                       sep = "_"), sctmpcomp5, envir = get(paste(unique.Vector(dds$
                              CellType)[i]), envir = EGMA_Cells_Comp))
433
434         suctmpcomp <- comparator(sctmpcomp1,sctmpcomp2)#12345
```

```
435         assign(paste( as.character(unique.Vector(dds$CellType)[i]), "EGMA_Comp", sep
                = "_"),
436                 suctmpcomp, envir = get(paste(unique.Vector(dds$CellType)[i]), envir
                    = EGMA_Cells_Comp))
437         x <- plot.new()
438         dev.off()
439         x <- plot.new()
440         draw.quintuple.venn(nrow(list1), nrow(list2), nrow(list3), nrow(list4), nrow
                (list5),nrow(tmpcomp1), nrow(tmpcomp2), nrow(tmpcomp3), nrow(tmpcomp4),
441                             nrow(tmpcomp5), nrow(tmpcomp6), nrow(tmpcomp7), nrow(
                                tmpcomp8), nrow(tmpcomp9), nrow(tmpcomp10),
442                             nrow(ctmpcomp1), nrow(ctmpcomp2), nrow(ctmpcomp3), nrow(
                                ctmpcomp4), nrow(ctmpcomp5), nrow(ctmpcomp6),
443                             nrow(ctmpcomp7), nrow(ctmpcomp8), nrow(ctmpcomp9), nrow(
                                ctmpcomp10),nrow(sctmpcomp1),
444                             nrow(sctmpcomp2), nrow(sctmpcomp3), nrow(sctmpcomp4),
                                nrow(sctmpcomp5),
445                             nrow(suctmpcomp),
446                             category = c(as.character(unique.Vector(dds$Source)[1]),
                                as.character(unique.Vector(dds$Source)[2]),
447                                          as.character(unique.Vector(dds$Source)[3]),
                                              as.character(unique.Vector(dds$Source)
                                              [4]),
448                                          as.character(unique.Vector(dds$Source)[4]))
                                              ,
449                             lty = "blank", fill = c("skyblue", "orange", "
                                mediumorchid","pink1","red"),cex = 2,cat.cex = 2)
450
451         title(paste(as.character(unique.Vector(dds$CellType)[i]),"VenPlot", sep = "
                "), cex.main =2)
452         assign(paste(as.character(unique.Vector(dds$CellType)[i]),"VenPlot", sep = "
                _"), recordPlot(x), envir = crossdata)
453       }
454     }
455     return(crossdata)
456     message("DONE")
457 }
458
459
460 # Fonction de comparaison des listes de niveaux d'expression
461 comparator <- function(list1,list2){
```

```
462     copy <- data.frame()
463     s <-1
464     rname1 <- rownames(data.frame(list1))
465     rname2 <- rownames(data.frame(list2))
466     for (q in 1:length(rname2)){
467       a <- which(rname1 == rname2[q])
468       if (length(a) > 0 ){
469         copy[s,1] <- sum(as.numeric(list1[a,1])+as.numeric(list2[q,1]))
470         copy[s,2] <- as.character(list1[a,which(colnames(list1) == "GeneName")])
471         rownames(copy)[s] <- as.character(rname2[q])
472         s<- s+1
473       }
474     }
475     copy <- as.data.frame(copy[order(copy[1], decreasing = TRUE),])
476     colnames(copy)[1] <- "SumLog2"
477     colnames(copy)[2] <- "GeneName"
478   return(copy)
479 }
480
481
482 # Fonction de Plot de PCA derivee du package SARTools
483 EGMApcaPLOT <-function (counts.trans, group ,source, n = min(500, nrow(counts.
      trans)),
484           colo = c("MediumVioletRed", "orange", "lightblue", "SpringGreen"),pch =
                c(13,15,16,17,18,10),
485           outfile = F) {
486   rv = apply(counts.trans, 1, var, na.rm = TRUE)
487   pca = prcomp(t(counts.trans[order(rv, decreasing = TRUE),
488                               ][1:n, ]))
489   par(mfrow = c(1, 1))
490   prp <- pca$sdev^2 * 100/sum(pca$sdev^2)
491   prp <- round(prp[1:3], 2)
492   if (outfile)
493     png(filename = "figures/PCA.png", width = 1800 * 2, height = 1800,
494         res = 300)
495   par(mar = par()$mar + c(0,0,0,7))
496   plot(pca$x[, 1], pca$x[, 2], cex = 1.4, pch = pch[as.numeric(group)],
497        col = colo[as.numeric(source)], xlab = paste0("PC1 (",
498                                                      prp[1], "%)"), ylab = paste0("
                                                          PC2 (", prp[2], "%)"),
499        main = "Principal Component Analysis")
```

```
500   colo = c("MediumVioletRed", "orange", "lightblue", "SpringGreen")
501   abline(h = 0, v = 0, lty = 2, col = "lightgray")
502   legend("topright",inset=c(-0.17,0),xpd = T,bty="n", legend = unique.Vector(group
          ), pch = pch[1:length(unique.Vector(group))])
503   legend("bottomright",inset=c(-0.17,0),xpd = T,bty="n", title.col = "black",
          title = "Sources:",legend = unique.Vector(source), text.col  = colo[1:length
          (unique.Vector(source))])
504   if (outfile)
505     dev.off()
506   return(invisible(pca$x))
507 }
508
509 # Fonction de production des listes de genes enrichis
510 markerList <- function(deseqObject,padj = 0.01, log2fold = 1){
511     output <- new.env()
512     outpos <<- new.env()
513     mylist <- unique(deseqObject@colData[,2])
514     message("To Process:")
515     message(paste(mylist,sep = " "))
516     level <- length(mylist)
517     combi <- combn(mylist,2)
518     filelist <-c()
519     for ( m in 1:ncol(combi)){ # Generate all possible combination of comparison
            with p-adj treshold
520       res <- results(deseqObject,alpha = padj ,cooksCutoff=F, lfcThreshold = 0,
              contrast = c(as.character(colnames(deseqObject@colData)[2]),as.character
              (combi[1,m]),as.character(combi[2,m])))
521       res <- res[complete.cases(res$padj),]
522       resOrdered <- res[order(res$log2FoldChange, decreasing = TRUE),]
523       assign(paste(as.character(combi[1,m]), as.character(combi[2,m]), sep="VS"),
              data.frame(resOrdered$log2FoldChange, row.names = resOrdered@rownames))
524       filelist <- c(filelist,paste(as.character(combi[1,m]), as.character(combi[2,
              m]), sep="VS"))
525     }
526    for (i in 1:length(mylist)){      # for each CellType
527       myTmpVar <- filelist[grep(mylist[i],filelist)]      # find which list to use
              for current Celltype
528       message("Running:")
529       message(mylist[i])
530       print(myTmpVar)
531       worklist <- c()
```

```
532       for (s in 1:length(myTmpVar)){        # Invert list values if needed and
              treshold on Log2Fold
533         if (substr(mylist[i],0,4) != substr(myTmpVar[s],0,4)){
534           tmp <- data.frame(get(myTmpVar[s]))
535           tmpneg <- data.frame(-tmp)
536           rname <- c(rownames(tmp))
537           rm <- which(as.numeric(tmpneg[,1])  < log2fold) #
538           rname <- rname[-rm]
539           tmpneg <-data.frame(tmpneg[-rm,], row.names = rname)
540           assign(paste(as.character(myTmpVar[s]),"inverted",sep = ""), tmpneg)
541           worklist[s] <- as.character(paste(as.character(myTmpVar[s]),"inverted",
                  sep = ""))
542         }
543         else{                           # Treshold on Log2Fold
544           tmp <- data.frame(get(myTmpVar[s]))
545           rname <- c(rownames(data.frame(get(myTmpVar[s]))))
546           rm <- which(tmp[,1]  < log2fold)
547           rname <- rname[-rm]
548           tmp <-data.frame(tmp[-rm,], row.names = rname)
549           assign(paste(as.character(myTmpVar[s]),"Tr",sep = ""), tmp)
550           worklist[s] <- as.character(paste(as.character(myTmpVar[s]),"Tr",sep = "
                  "))
551         }
552         tmp <- data.frame()
553         rname <- c()
554       }
555       print(worklist)
556       message("Treshold passed:")
557       for (s in 1:length(worklist)){
558       message(nrow(get(worklist[s])))
559       }
560       if (length(worklist) > 2){
561         copy <- data.frame()
562         s <-1
563         rname1 <- rownames(data.frame(get(worklist[2])))
564         rname2 <- rownames(data.frame(get(worklist[1])))
565         pb <- txtProgressBar(min = 0, max = length(rname2) || 1, style = 3)
566         for (q in 1:length(rname2)){
567           a <- which(rname1 == rname2[q])
568           if (length(a) > 0 ){
569             rownames(copy[s,]) <- rname2[q]
```

```
570             copy[s,1] <- data.frame(get(worklist[1])[q,1])
571             copy[s,2] <- data.frame(get(worklist[2])[a,1])
572             s<- s+1
573           }
574           setTxtProgressBar(pb, q)
575         }
576         close(pb)
577         copy2 <- data.frame()
578         for (c in 3:(length(worklist))){
579           s <-1
580           rname1 <- rownames(data.frame(get(worklist[c])))
581           rname2 <- rownames(data.frame(copy))
582           pb <- txtProgressBar(min = 0, max = nrow(copy) || 1, style = 3)
583           for (q in 1:length(rname2)){
584             a <- which(rname1 == rname2[q])
585             if (length(a) > 0 ){
586               rownames(copy2[s,]) <- rname2[q]
587               for (t in 1:(c-1)){
588                 copy2[s,t] <- data.frame(copy[q,t])
589               }
590               copy2[s,c] <- data.frame(get(worklist[c])[a,1])
591               s<- s+1
592             }
593             setTxtProgressBar(pb, q)
594           }
595           close(pb)
596           #print(copy2)
597           copy <- data.frame(copy2)
598           copy2 <- data.frame()
599         }
600         if (nrow(copy) > 0){
601         colnames(copy) <- c(worklist)
602         n <- ncol(copy)+1
603         trc <- c()
604         for(k in 1:nrow(copy)){
605           trc <- c(trc,sum(copy[k,]))
606         }
607         copy[,n] <- trc
608         colnames(copy)[n] <- "Sum"
609         copy <- copy[order(copy$Sum, decreasing = TRUE),]
610         }
```

```
611         message("Nb Enriched Genes:")
612         message(nrow(copy))
613         message("------------------------------------------------------------")
614         posix <- c()
615         for (t in 1:nrow(copy)){
616           posix <- c(posix,which(rownames(data.frame(get(worklist[1]))) ==
                rownames(data.frame(copy)[t,])))
617         }
618         assign(as.character(mylist[i]), posix, envir = outpos)
619         assign(as.character(mylist[i]), copy, envir = output)
620       }
621       else {
622         assign(as.character(mylist[i]), get(worklist[1]), envir = output)
623       }
624     }
625   return(output)
626 }
627
628
629 # Fonction de traduction des identifiant ENSEMBL en nom de genes
630 traductor <- function(MarkerListObject, ConversionDataFrame){
631   if (is.environment(MarkerListObject)){
632     for (i in names(MarkerListObject)){
633       nc <- (ncol(i)+1)
634       GeneName <- c()
635       rname1 <- rownames(data.frame(get(i, envir = MarkerListObject)))
636       rname2 <- rownames(data.frame(ConversionDataFrame))
637       for (q in 1:length(rname1)){
638         a <- which(rname2 == rname1[q])
639         if (length(a) > 0 ){
640           GeneName <- c(GeneName,as.character(ConversionDataFrame[a,1]))
641         }
642         else{
643           GeneName <- c(GeneName,"undef")
644         }
645       }
646       tmp<- cbind(get(i,envir = MarkerListObject),GeneName)
647       assign(as.character(i),envir = MarkerListObject,tmp)
648       message(paste(i, "done",sep = " "))
649     }
650   }
```

```
651   else{
652     i <- MarkerListObject
653     nc <- (ncol(i)+1)
654     GeneName <- c()
655     rname1 <- rownames(data.frame(i))
656     rname2 <- rownames(data.frame(ConversionDataFrame))
657     for (q in 1:length(rname1)){
658       a <- which(rname2 == rname1[q])
659       if (length(a) > 0 ){
660         GeneName <- c(GeneName,as.character(ConversionDataFrame[a,1]))
661       }
662     }
663     tmp<- cbind(i,GeneName)
664     return(tmp)
665     message(paste(i, "done",sep = " "))
666   }
667 }
668
669 #Fonction de production des scaterPlot base sur lepackage SARTools
670 EGMAscaterplot <- function (counts, group, outfile = TRUE, cex.lab = 1, method = "
      SERE", title = "Pairwise scatter plot")
671 {
672   ncol <- ncol(counts)
673   if (ncol <= 30) {
674     if (outfile)
675       png(filename = "figures/pairwiseScatter.png", width = 700 *
676             ncol, height = 700 * ncol, res = 300)
677     panel <- function(x, y, ...) {
678       points(x, y, pch = ".")
679       abline(a = 0, b = 1, lty = 2)
680     }
681     lower.panel <- function(x, y, ...) {
682       horizontal <- (par("usr")[1] + par("usr")[2])/2
683       vertical <- (par("usr")[3] + par("usr")[4])/2
684       if (method == "SERE"){
685         text(horizontal, vertical, round(SERE(2^cbind(x,
686                                                       y) - 1), digits = 2), cex =
                                                           cex.lab)
687       }
688       else{
689         if (method == "pearson" || method == "spearman" || method == "kendall"){
```

```
690           text(horizontal, vertical, round(data.frame(cor(cbind(x,y)), use="
                  complete.obs", method = method)[2,1],digits = 2),cex = cex.lab)
691         }
692         else{
693           warning("Wrong selected method")
694         }
695       }
696     }
697     pairs(log2(counts + 1), panel = panel, lower.panel = lower.panel,
698           las = 1, labels = paste(colnames(counts),
699                                   sep = "\n"), main = title,
700           cex.labels = cex.lab, cex.main = cex.lab)
701     if (outfile)
702       dev.off()
703   }
704   else {
705     warning("No pairwise scatter-plot produced because of a too high number of
          samples (>30).")
706     if (outfile)
707       png(filename = "figures/pairwiseScatter.png", width = 1900,
708           height = 1900, res = 300)
709     par(mar = c(1.5, 1.5, 1.5, 1.5))
710     plot(0, 0, bty = "o", pch = ".", col = "white", xaxt = "n",
711          yaxt = "n", xlab = "", ylab = "")
712     text(0, 0.3, "No pairwise scatter-plot produced because of\na too high number
          of samples (>30).",
713          pos = 1, cex = 1.5)
714     if (outfile)
715       dev.off()
716   }
717 }
718
719 # Fonction adaptee pour la production automatique des heatmaps de correlation
720 myheat2 <- function (x, Rowv = TRUE, Colv = if (symm) "Rowv" else TRUE,
721                      distfun = dist, hclustfun = hclust, dendrogram = c("both",
722                                                                         "row", "
                                                                             column
                                                                             ", "
                                                                             none")
                                                                             ,
                                                                             reorderfun
```

```
                                                                    =
                                                                    function
                                                                    (d, w)

                                                                    reorder
                                                                    (d,
723
```

```
724                 na.rm = TRUE, revC = identical(Colv, "Rowv"), add.expr,
                    breaks,
```

```
725                     symbreaks = any(x < 0, na.rm = TRUE) || scale != "none",
726                     col = "heat.colors", colsep, rowsep, sepcolor = "white",
727                     sepwidth = c(0.05, 0.05), cellnote, notecex = 1, notecol = "
                            cyan",
728                     na.color = par("bg"), trace = c("column", "row", "both",
729                                                     "none"), tracecol = "cyan",
                                                         hline = median(breaks),
                                                         vline = median(breaks),
730                     linecol = tracecol, margins = c(5, 5), ColSideColors,
                            RowSideColors,
731                     cexRow = 0.2 + 1/log10(nr), cexCol = 0.2 + 1/log10(nc),
                            labRow = NULL,
732                     labCol = NULL, srtRow = NULL, srtCol = NULL, adjRow = c(0,
733                                                                   NA),
                                                                        adjCol
                                                                        =
                                                                        c
                                                                        (
                                                                        NA
                                                                        ,
                                                                        0)
                                                                        ,
                                                                        offsetRow
                                                                        =
                                                                        0.5,
                                                                        offsetCol
                                                                        =
                                                                        0.5,
734                     colRow = NULL, colCol = NULL, key = TRUE, keysize = 1.5,
735                     density.info = c("histogram", "density", "none"), denscol =
                            tracecol,
```

```
736                         symkey = any(x < 0, na.rm = TRUE) || symbreaks, densadj =
                                0.25,
737                         key.title = NULL, key.xlab = NULL, key.ylab = NULL, key.
                                xtickfun = NULL,
738                         key.ytickfun = NULL, key.par = list(), main = NULL, xlab =
                                NULL,
739                         ylab = NULL, lmat = NULL, lhei = NULL, lwid = NULL, extrafun
                                = NULL,
740                         ...)
741 {
742   scale01 <- function(x, low = min(x), high = max(x)) {
743     x <- (x - low)/(high - low)
744     x
745   }
746   retval <- list()
747   scale <- if (symm && missing(scale))
748     "none"
749   else match.arg(scale)
750   dendrogram <- match.arg(dendrogram)
751   trace <- match.arg(trace)
752   density.info <- match.arg(density.info)
753   if (length(col) == 1 && is.character(col))
754     col <- get(col, mode = "function")
755   if (!missing(breaks) && any(duplicated(breaks)))
756     stop("breaks may not contian duplicate values")
757   if (!missing(breaks) && (scale != "none"))
758     warning("Using scale=\"row\" or scale=\"column\" when breaks are",
759             "specified can produce unpredictable results.", "Please consider using
                    only one or the other.")
760   if (is.null(Rowv) || is.na(Rowv))
761     Rowv <- FALSE
762   if (is.null(Colv) || is.na(Colv))
763     Colv <- FALSE
764   else if (all(Colv == "Rowv"))
765     Colv <- Rowv
766   if (length(di <- dim(x)) != 2 || !is.numeric(x))
767     stop("'x' must be a numeric matrix")
768   nr <- di[1]
769   nc <- di[2]
770   if (nr <= 1 || nc <= 1)
771     stop("'x' must have at least 2 rows and 2 columns")
```

```
772   if (!is.numeric(margins) || length(margins) != 2)
773     stop("'margins' must be a numeric vector of length 2")
774   if (missing(cellnote))
775     cellnote <- matrix("", ncol = ncol(x), nrow = nrow(x))
776   if (!inherits(Rowv, "dendrogram")) {
777     if (((is.logical(Rowv) && !isTRUE(Rowv)) || (is.null(Rowv))) &&
778         (dendrogram %in% c("both", "row"))) {
779       warning("Discrepancy: Rowv is FALSE, while dendrogram is `",
780               dendrogram, "'. Omitting row dendogram.")
781       if (dendrogram == "both")
782         dendrogram <- "column"
783       else dendrogram <- "none"
784     }
785   }
786   if (!inherits(Colv, "dendrogram")) {
787     if (((is.logical(Colv) && !isTRUE(Colv)) || (is.null(Colv))) &&
788         (dendrogram %in% c("both", "column"))) {
789       warning("Discrepancy: Colv is FALSE, while dendrogram is `",
790               dendrogram, "'. Omitting column dendogram.")
791       if (dendrogram == "both")
792         dendrogram <- "row"
793       else dendrogram <- "none"
794     }
795   }
796   if (inherits(Rowv, "dendrogram")) {
797     ddr <- Rowv
798     rowInd <- order.dendrogram(ddr)
799     if (length(rowInd) > nr || any(rowInd < 1 | rowInd >
800                                    nr))
801       stop("Rowv dendrogram doesn't match size of x")
802     if (length(rowInd) < nr)
803       nr <- length(rowInd)
804   }
805   else if (is.integer(Rowv)) {
806     distr <- distfun(x)
807     hcr <<- hclust(dist(t(counts.trans)), method = "ward.D")
808     ddr <- as.dendrogram(hcr)
809     ddr <- reorderfun(ddr, Rowv)
810     rowInd <- order.dendrogram(ddr)
811     if (nr != length(rowInd))
812       stop("row dendrogram ordering gave index of wrong length")
```

```
813   }
814   else if (isTRUE(Rowv)) {
815     Rowv <- rowMeans(x, na.rm = na.rm)
816     distr <- distfun(x)
817     hcr <<- hclust(dist(t(counts.trans)), method = "ward.D")
818     ddr <- as.dendrogram(hcr)
819     ddr <- reorderfun(ddr, Rowv)
820     rowInd <- order.dendrogram(ddr)
821     if (nr != length(rowInd))
822       stop("row dendrogram ordering gave index of wrong length")
823   }
824   else if (!isTRUE(Rowv)) {
825     rowInd <- nr:1
826     ddr <- as.dendrogram(hclust(dist(diag(nr))))
827   }
828   else {
829     rowInd <- nr:1
830     ddr <- as.dendrogram(Rowv)
831   }
832   if (inherits(Colv, "dendrogram")) {
833     ddc <- Colv
834     colInd <- order.dendrogram(ddc)
835     if (length(colInd) > nc || any(colInd < 1 | colInd >
836                                      nc))
837       stop("Colv dendrogram doesn't match size of x")
838     if (length(colInd) < nc)
839       nc <- length(colInd)
840   }
841   else if (identical(Colv, "Rowv")) {
842     if (nr != nc)
843       stop("Colv = \"Rowv\" but nrow(x) != ncol(x)")
844     if (exists("ddr")) {
845       ddc <- ddr
846       colInd <- order.dendrogram(ddc)
847     }
848     else colInd <- rowInd
849   }
850   else if (is.integer(Colv)) {
851     distc <- distfun(if (symm)
852       x
853       else t(x))
```

```
854     hcc <- hclustfun(distc)
855     ddc <- as.dendrogram(hcc)
856     ddc <- reorderfun(ddc, Colv)
857     colInd <- order.dendrogram(ddc)
858     if (nc != length(colInd))
859       stop("column dendrogram ordering gave index of wrong length")
860   }
861   else if (isTRUE(Colv)) {
862     Colv <- colMeans(x, na.rm = na.rm)
863     distc <- distfun(if (symm)
864       x
865       else t(x))
866     hcc <- hclustfun(distc)
867     ddc <- as.dendrogram(hcc)
868     ddc <- reorderfun(ddc, Colv)
869     colInd <- order.dendrogram(ddc)
870     if (nc != length(colInd))
871       stop("column dendrogram ordering gave index of wrong length")
872   }
873   else if (!isTRUE(Colv)) {
874     colInd <- 1:nc
875     ddc <- as.dendrogram(hclust(dist(diag(nc))))
876   }
877   else {
878     colInd <- 1:nc
879     ddc <- as.dendrogram(Colv)
880   }
881   retval$rowInd <- rowInd
882   retval$colInd <- colInd
883   retval$call <- match.call()
884   x <- x[rowInd, colInd]
885   x.unscaled <- x
886   cellnote <- cellnote[rowInd, colInd]
887   if (is.null(labRow))
888     labRow <- if (is.null(rownames(x)))
889       (1:nr)[rowInd]
890   else rownames(x)
891   else labRow <- labRow[rowInd]
892   if (is.null(labCol))
893     labCol <- if (is.null(colnames(x)))
894       (1:nc)[colInd]
```

```
895   else colnames(x)
896   else labCol <- labCol[colInd]
897   if (!is.null(colRow))
898     colRow <- colRow[rowInd]
899   if (!is.null(colCol))
900     colCol <- colCol[colInd]
901   if (scale == "row") {
902     retval$rowMeans <- rm <- rowMeans(x, na.rm = na.rm)
903     x <- sweep(x, 1, rm)
904     retval$rowSDs <- sx <- apply(x, 1, sd, na.rm = na.rm)
905     x <- sweep(x, 1, sx, "/")
906   }
907   else if (scale == "column") {
908     retval$colMeans <- rm <- colMeans(x, na.rm = na.rm)
909     x <- sweep(x, 2, rm)
910     retval$colSDs <- sx <- apply(x, 2, sd, na.rm = na.rm)
911     x <- sweep(x, 2, sx, "/")
912   }
913   if (missing(breaks) || is.null(breaks) || length(breaks) <
914       1) {
915     if (missing(col) || is.function(col))
916       breaks <- 16
917     else breaks <- length(col) + 1
918   }
919   if (length(breaks) == 1) {
920     if (!symbreaks)
921       breaks <- seq(min(x, na.rm = na.rm), max(x, na.rm = na.rm),
922                     length = breaks)
923     else {
924       extreme <- max(abs(x), na.rm = TRUE)
925       breaks <- seq(-extreme, extreme, length = breaks)
926     }
927   }
928   nbr <- length(breaks)
929   ncol <- length(breaks) - 1
930   if (class(col) == "function")
931     col <- col(ncol)
932   min.breaks <- min(breaks)
933   max.breaks <- max(breaks)
934   x[x < min.breaks] <- min.breaks
935   x[x > max.breaks] <- max.breaks
```

```
936   if (missing(lhei) || is.null(lhei))
937     lhei <- c(keysize, 4)
938   if (missing(lwid) || is.null(lwid))
939     lwid <- c(keysize, 4)
940   if (missing(lmat) || is.null(lmat)) {
941     lmat <- rbind(4:3, 2:1)
942     if (!missing(ColSideColors)) {
943       if (!is.character(ColSideColors) || length(ColSideColors) !=
944           nc)
945         stop("'ColSideColors' must be a character vector of length ncol(x)")
946       lmat <- rbind(lmat[1, ] + 1, c(NA, 1), lmat[2, ] +
947                       1)
948       lhei <- c(lhei[1], 0.2, lhei[2])
949     }
950     if (!missing(RowSideColors)) {
951       if (!is.character(RowSideColors) || length(RowSideColors) !=
952           nr)
953         stop("'RowSideColors' must be a character vector of length nrow(x)")
954       lmat <- cbind(lmat[, 1] + 1, c(rep(NA, nrow(lmat) -
955                                            1), 1), lmat[, 2] + 1)
956       lwid <- c(lwid[1], 0.2, lwid[2])
957     }
958     lmat[is.na(lmat)] <- 0
959   }
960   if (length(lhei) != nrow(lmat))
961     stop("lhei must have length = nrow(lmat) = ", nrow(lmat))
962   if (length(lwid) != ncol(lmat))
963     stop("lwid must have length = ncol(lmat) =", ncol(lmat))
964   op <- par(no.readonly = TRUE)
965   on.exit(par(op))
966   layout(lmat, widths = lwid, heights = lhei, respect = FALSE)
967   plot.index <- 1
968   if (!missing(RowSideColors)) {
969     par(mar = c(margins[1], 0, 0, 0.5))
970     image(rbind(1:nr), col = RowSideColors[rowInd], axes = FALSE)
971     plot.index <- plot.index + 1
972   }
973   if (!missing(ColSideColors)) {
974     par(mar = c(0.5, 0, 0, margins[2]))
975     image(cbind(1:nc), col = ColSideColors[colInd], axes = FALSE)
976     plot.index <- plot.index + 1
```

```
977   }
978   par(mar = c(margins[1], 0, 0, margins[2]))
979   x <- t(x)
980   cellnote <- t(cellnote)
981   if (revC) {
982     iy <- nr:1
983     if (exists("ddr"))
984       ddr <- rev(ddr)
985     x <- x[, iy]
986     cellnote <- cellnote[, iy]
987   }
988   else iy <- 1:nr
989   image(1:nc, 1:nr, x, xlim = 0.5 + c(0, nc), ylim = 0.5 +
990           c(0, nr), axes = FALSE, xlab = "", ylab = "", col = col,
991         breaks = breaks, ...)
992   retval$carpet <- x
993   if (exists("ddr"))
994     retval$rowDendrogram <- ddr
995   if (exists("ddc"))
996     retval$colDendrogram <- ddc
997   retval$breaks <- breaks
998   retval$col <- col
999   if (is.null(srtCol) && is.null(colCol))
1000    axis(1, 1:nc, labels = labCol, las = 2, line = -0.5 +
1001           offsetCol, tick = 0, cex.axis = cexCol, hadj = adjCol[1],
1002         padj = adjCol[2])
1003  else {
1004    if (is.null(srtCol) || is.numeric(srtCol)) {
1005      if (missing(adjCol) || is.null(adjCol))
1006        adjCol = c(1, NA)
1007      if (is.null(srtCol))
1008        srtCol <- 90
1009      xpd.orig <- par("xpd")
1010      par(xpd = NA)
1011      xpos <- axis(1, 1:nc, labels = rep("", nc), las = 2,
1012                   tick = 0)
1013      text(x = xpos, y = par("usr")[3] - (1 + offsetCol) *
1014             strheight("M"), labels = labCol, adj = adjCol,
1015           cex = cexCol, srt = srtCol, col = colCol)
1016      par(xpd = xpd.orig)
1017    }
```

```
1018      else warning("Invalid value for srtCol ignored.")
1019    }
1020    if (is.null(srtRow) && is.null(colRow)) {
1021      axis(4, iy, labels = labRow, las = 2, line = -0.5 + offsetRow,
1022           tick = 0, cex.axis = cexRow, hadj = adjRow[1], padj = adjRow[2])
1023    }
1024    else {
1025      if (is.null(srtRow) || is.numeric(srtRow)) {
1026        xpd.orig <- par("xpd")
1027        par(xpd = NA)
1028        ypos <- axis(4, iy, labels = rep("", nr), las = 2,
1029                     line = -0.5, tick = 0)
1030        text(x = par("usr")[2] + (1 + offsetRow) * strwidth("M"),
1031             y = ypos, labels = labRow, adj = adjRow, cex = cexRow,
1032             srt = srtRow, col = colRow)
1033        par(xpd = xpd.orig)
1034      }
1035      else warning("Invalid value for srtRow ignored.")
1036    }
1037    if (!is.null(xlab))
1038      mtext(xlab, side = 1, line = margins[1] - 1.25)
1039    if (!is.null(ylab))
1040      mtext(ylab, side = 4, line = margins[2] - 1.25)
1041    if (!missing(add.expr))
1042      eval(substitute(add.expr))
1043    if (!missing(colsep))
1044      for (csep in colsep) rect(xleft = csep + 0.5, ybottom = 0,
1045                                xright = csep + 0.5 + sepwidth[1], ytop = ncol(x) +
1046                                  1, lty = 1, lwd = 1, col = sepcolor, border =
                                      sepcolor)
1047    if (!missing(rowsep))
1048      for (rsep in rowsep) rect(xleft = 0, ybottom = (ncol(x) +
1049                                                        1 - rsep) - 0.5, xright =
                                                            nrow(x) + 1, ytop = (
                                                            ncol(x) +
1050
```

```
                                col = sepcolor, border = sepcolor)
  min.scale <- min(breaks)
  max.scale <- max(breaks)
  x.scaled <- scale01(t(x), min.scale, max.scale)
  if (trace %in% c("both", "column")) {
    retval$vline <- vline
    vline.vals <- scale01(vline, min.scale, max.scale)
    for (i in 1:length(colInd)) {
      if (!is.null(vline)) {
        abline(v = i - 0.5 + vline.vals, col = linecol,
               lty = 2)
      }
      xv <- rep(i, nrow(x.scaled)) + x.scaled[, i] - 0.5
      xv <- c(xv[1], xv)
      yv <- 1:length(xv) - 0.5
      lines(x = xv, y = yv, lwd = 1, col = tracecol, type = "s")
    }
  }
  if (trace %in% c("both", "row")) {
```

```
1070      retval$hline <- hline
1071      hline.vals <- scale01(hline, min.scale, max.scale)
1072      for (i in 1:length(rowInd)) {
1073        if (!is.null(hline)) {
1074          abline(h = i - 0.5 + hline.vals, col = linecol,
1075                 lty = 2)
1076        }
1077        yv <- rep(i, ncol(x.scaled)) + x.scaled[i, ] - 0.5
1078        yv <- rev(c(yv[1], yv))
1079        xv <- length(yv):1 - 0.5
1080        lines(x = xv, y = yv, lwd = 1, col = tracecol, type = "s")
1081      }
1082    }
1083    if (!missing(cellnote))
1084      text(x = c(row(cellnote)), y = c(col(cellnote)), labels = c(cellnote),
1085           col = notecol, cex = notecex)
1086    plot.index <- plot.index + 1
1087    par(mar = c(margins[1], 0, 0, 0))
1088    if (dendrogram %in% c("both", "row")) {
1089      flag <- try(plot.dendrogram(ddr, horiz = TRUE, axes = FALSE,
1090                                  yaxs = "i", leaflab = "none"))
1091      if ("try-error" %in% class(flag)) {
1092        cond <- attr(flag, "condition")
1093        if (!is.null(cond) && conditionMessage(cond) == "evaluation nested too
              deeply: infinite recursion / options(expressions=)?")
1094          stop("Row dendrogram too deeply nested, recursion limit exceeded.  Try
                increasing option(\"expressions\"=...).")
1095      }
1096    }
1097    else plot.new()
1098    par(mar = c(0, 0, if (!is.null(main)) 5 else 0, margins[2]))
1099    if (dendrogram %in% c("both", "column")) {
1100      flag <- try(plot.dendrogram(ddc, axes = FALSE, xaxs = "i",
1101                                  leaflab = "none"))
1102      if ("try-error" %in% class(flag)) {
1103        cond <- attr(flag, "condition")
1104        if (!is.null(cond) && conditionMessage(cond) == "evaluation nested too
              deeply: infinite recursion / options(expressions=)?")
1105          stop("Column dendrogram too deeply nested, recursion limit exceeded.  Try
                increasing option(\"expressions\"=...).")
1106      }
```

```
1107    }
1108    else plot.new()
1109    if (!is.null(main))
1110      title(main, cex.main = 1.5 * op[["cex.main"]])
1111    if (key) {
1112      mar <- c(5, 4, 2, 1)
1113      if (!is.null(key.xlab) && is.na(key.xlab))
1114        mar[1] <- 2
1115      if (!is.null(key.ylab) && is.na(key.ylab))
1116        mar[2] <- 2
1117      if (!is.null(key.title) && is.na(key.title))
1118        mar[3] <- 1
1119      par(mar = mar, cex = 0.75, mgp = c(2, 1, 0))
1120      if (length(key.par) > 0)
1121        do.call(par, key.par)
1122      tmpbreaks <- breaks
1123      if (symkey) {
1124        max.raw <- max(abs(c(x, breaks)), na.rm = TRUE)
1125        min.raw <- -max.raw
1126        tmpbreaks[1] <- -max(abs(x), na.rm = TRUE)
1127        tmpbreaks[length(tmpbreaks)] <- max(abs(x), na.rm = TRUE)
1128      }
1129      else {
1130        min.raw <- min.breaks
1131        max.raw <- max.breaks
1132      }
1133      z <- seq(min.raw, max.raw, by = min(diff(breaks)/100))
1134      image(z = matrix(z, ncol = 1), col = col, breaks = tmpbreaks,
1135            xaxt = "n", yaxt = "n")
1136      par(usr = c(0, 1, 0, 1))
1137      if (is.null(key.xtickfun)) {
1138        lv <- pretty(breaks)
1139        xv <- scale01(as.numeric(lv), min.raw, max.raw)
1140        xargs <- list(at = xv, labels = lv)
1141      }
1142      else {
1143        xargs <- key.xtickfun()
1144      }
1145      xargs$side <- 1
1146      do.call(axis, xargs)
1147      if (is.null(key.xlab)) {
```

```
1148        if (scale == "row")
1149          key.xlab <- "Row Z-Score"
1150        else if (scale == "column")
1151          key.xlab <- "Column Z-Score"
1152        else key.xlab <- "Value"
1153      }
1154      if (!is.na(key.xlab)) {
1155        mtext(side = 1, key.xlab, line = par("mgp")[1], padj = 0.5,
1156              cex = par("cex") * par("cex.lab"))
1157      }
1158      if (density.info == "density") {
1159        dens <- density(x, adjust = densadj, na.rm = TRUE,
1160                        from = min.scale, to = max.scale)
1161        omit <- dens$x < min(breaks) | dens$x > max(breaks)
1162        dens$x <- dens$x[!omit]
1163        dens$y <- dens$y[!omit]
1164        dens$x <- scale01(dens$x, min.raw, max.raw)
1165        lines(dens$x, dens$y/max(dens$y) * 0.95, col = denscol,
1166              lwd = 1)
1167        if (is.null(key.ytickfun)) {
1168          yargs <- list(at = pretty(dens$y)/max(dens$y) *
1169                          0.95, labels = pretty(dens$y))
1170        }
1171        else {
1172          yargs <- key.ytickfun()
1173        }
1174        yargs$side <- 2
1175        do.call(axis, yargs)
1176        if (is.null(key.title))
1177          key.title <- "Color Key\nand Density Plot"
1178        if (!is.na(key.title))
1179          title(key.title)
1180        par(cex = 0.5)
1181        if (is.null(key.ylab))
1182          key.ylab <- "Density"
1183        if (!is.na(key.ylab))
1184          mtext(side = 2, key.ylab, line = par("mgp")[1],
1185                padj = 0.5, cex = par("cex") * par("cex.lab"))
1186      }
1187      else if (density.info == "histogram") {
1188        h <- hist(x, plot = FALSE, breaks = breaks)
```

```
1189        hx <- scale01(breaks, min.raw, max.raw)
1190        hy <- c(h$counts, h$counts[length(h$counts)])
1191        lines(hx, hy/max(hy) * 0.95, lwd = 1, type = "s",
1192              col = denscol)
1193        if (is.null(key.ytickfun)) {
1194          yargs <- list(at = pretty(hy)/max(hy) * 0.95,
1195                        labels = pretty(hy))
1196        }
1197        else {
1198          yargs <- key.ytickfun()
1199        }
1200        yargs$side <- 2
1201        do.call(axis, yargs)
1202        if (is.null(key.title))
1203          key.title <- "Color Key\nand Histogram"
1204        if (!is.na(key.title))
1205          title(key.title)
1206        par(cex = 0.5)
1207        if (is.null(key.ylab))
1208          key.ylab <- "Count"
1209        if (!is.na(key.ylab))
1210          mtext(side = 2, key.ylab, line = par("mgp")[1],
1211                padj = 0.5, cex = par("cex") * par("cex.lab"))
1212      }
1213      else if (is.null(key.title))
1214        title("Color Key")
1215      if (trace %in% c("both", "column")) {
1216        vline.vals <- scale01(vline, min.raw, max.raw)
1217        if (!is.null(vline)) {
1218          abline(v = vline.vals, col = linecol, lty = 2)
1219        }
1220      }
1221      if (trace %in% c("both", "row")) {
1222        hline.vals <- scale01(hline, min.raw, max.raw)
1223        if (!is.null(hline)) {
1224          abline(v = hline.vals, col = linecol, lty = 2)
1225        }
1226      }
1227    }
1228    else {
1229      par(mar = c(0, 0, 0, 0))
```

```
1230      plot.new()
1231    }
1232    retval$colorTable <- data.frame(low = retval$breaks[-length(retval$breaks)],
1233                                    high = retval$breaks[-1], color = retval$col)
1234    retval$layout <- list(lmat = lmat, lhei = lhei, lwid = lwid)
1235    if (!is.null(extrafun))
1236      extrafun()
1237    invisible(retval)
1238  }
1239
1240  # Fonction modifiee pour la production automatique des dendrogramme sur base du
        package SARTools
1241  plot.dendrogram <- function (x, type = c("rectangle", "triangle"), center = FALSE,
1242                               edge.root = is.leaf(x) || !is.null(attr(x, "edgetext"
                                     )),
1243                               nodePar = NULL, edgePar = list(), leaflab = c("
                                     perpendicular",
1244                                                                             "
                                                                                   textlike
                                                                                   ",
                                                                                   "
                                                                                   none
                                                                                   "),
                                                                             dLeaf
                                                                              =
                                                                             NULL
                                                                             ,
                                                                             xlab
                                                                              =
                                                                             "",
                                                                             ylab
                                                                              =
                                                                             "",
1245                               xaxt = "n", yaxt = "s", horiz = FALSE, frame.plot =
                                     FALSE,
1246                               xlim, ylim, ...)
1247  {
1248    type <- match.arg(type)
1249    leaflab <- match.arg(leaflab)
```

```
1250    hgt <- attr(x, "height")
1251    if (edge.root && is.logical(edge.root))
1252      edge.root <- 0.0625 * if (is.leaf(x))
1253        1
1254    else hgt
1255    mem.x <- .memberDend(x)
1256    yTop <- hgt + edge.root
1257    if (center) {
1258      x1 <- 0.5
1259      x2 <- mem.x + 0.5
1260    }
1261    else {
1262      x1 <- 1
1263      x2 <- mem.x
1264    }
1265    xl. <- c(x1 - 1/2, x2 + 1/2)
1266    yl. <- c(0, yTop)
1267    if (horiz) {
1268      tmp <- xl.
1269      xl. <- rev(yl.)
1270      yl. <- tmp
1271      tmp <- xaxt
1272      xaxt <- yaxt
1273      yaxt <- tmp
1274    }
1275    if (missing(xlim) || is.null(xlim))
1276      xlim <- xl.
1277    if (missing(ylim) || is.null(ylim))
1278      ylim <- yl.
1279    dev.hold()
1280    on.exit(dev.flush())
1281    plot(0, xlim = xlim, ylim = ylim, type = "n", xlab = xlab,
1282         ylab = ylab, xaxt = xaxt, yaxt = yaxt, frame.plot = frame.plot,
1283         ...)
1284    if (is.null(dLeaf))
1285      dLeaf <- 0.75 * (if (horiz)
1286        strwidth("w")
1287        else strheight("x"))
1288    if (edge.root) {
1289      x0 <- plotNodeLimit(x1, x2, x, center)$x
1290      if (horiz)
```

```
1291        segments(hgt, x0, yTop, x0)
1292      else segments(x0, hgt, x0, yTop)
1293      if (!is.null(et <- attr(x, "edgetext"))) {
1294        my <- mean(hgt, yTop)
1295        if (horiz)
1296          text(my, x0, et)
1297        else text(x0, my, et)
1298      }
1299    }
1300    plotNode(x1, x2, x, type = type, center = center, leaflab = leaflab,
1301             dLeaf = dLeaf, nodePar = nodePar, edgePar = edgePar,
1302             horiz = horiz)
1303  }
1304
1305
1306  .memberDend <- function (x)
1307  {
1308    r <- attr(x, "x.member")
1309    if (is.null(r)) {
1310      r <- attr(x, "members")
1311      if (is.null(r))
1312        r <- 1L
1313    }
1314    r
1315  }
1316
1317  # Fonction embarquee pour la production des heatmaps
1318  plotNode <- function (x1, x2, subtree, type, center, leaflab, dLeaf, nodePar,
1319                        edgePar, horiz = FALSE)
1320  {
1321    wholetree <- subtree
1322    depth <- 0L
1323    llimit <- list()
1324    KK <- integer()
1325    kk <- integer()
1326    repeat {
1327      inner <- !is.leaf(subtree) && x1 != x2
1328      yTop <- attr(subtree, "height")
1329      bx <- plotNodeLimit(x1, x2, subtree, center)
1330      xTop <- bx$x
1331      depth <- depth + 1L
```

```
1332      llimit[[depth]] <- bx$limit
1333      hasP <- !is.null(nPar <- attr(subtree, "nodePar"))
1334      if (!hasP)
1335        nPar <- nodePar
1336      if (getOption("verbose")) {
1337        cat(if (inner)
1338          "inner node"
1339          else "leaf", ":")
1340        if (!is.null(nPar)) {
1341          cat(" with node pars\n")
1342          str(nPar)
1343        }
1344        cat(if (inner)
1345          paste(" height", formatC(yTop), "; "), "(x1,x2)= (",
1346          formatC(x1, width = 4), ",", formatC(x2, width = 4),
1347          ")", "--> xTop=", formatC(xTop, width = 8),
1348          "\n", sep = "")
1349      }
1350      Xtract <- function(nam, L, default, indx) rep(if (nam %in%
1351                                                        names(L)) L[[nam]] else
                                                            default, length.out =
                                                            indx)[indx]
1352      asTxt <- function(x) if (is.character(x) || is.expression(x) ||
1353                               is.null(x))
1354        x
1355      else as.character(x)
1356      i <- if (inner || hasP)
1357        1
1358      else 2
1359      if (!is.null(nPar)) {
1360        pch <- Xtract("pch", nPar, default = 1L:2, i)
1361        cex <- Xtract("cex", nPar, default = c(1, 1), i)
1362        col <- Xtract("col", nPar, default = par("col"),
1363                      i)
1364        bg <- Xtract("bg", nPar, default = par("bg"), i)
1365        points(if (horiz)
1366          cbind(yTop, xTop)
1367          else cbind(xTop, yTop), pch = pch, bg = bg, col = col,
1368          cex = cex)
1369      }
1370      if (leaflab == "textlike")
```

```
1371        p.col <- Xtract("p.col", nPar, default = "white",
1372                        i)
1373      lab.col <- Xtract("lab.col", nPar, default = par("col"),
1374                        i)
1375      lab.cex <- Xtract("lab.cex", nPar, default = c(1, 1),
1376                        i)
1377      lab.font <- Xtract("lab.font", nPar, default = par("font"),
1378                         i)
1379      lab.xpd <- Xtract("xpd", nPar, default = c(TRUE, TRUE),
1380                        i)
1381      if (is.leaf(subtree)) {
1382        if (leaflab == "perpendicular") {
1383          if (horiz) {
1384            X <- yTop + dLeaf * lab.cex
1385            Y <- xTop
1386            srt <- 0
1387            adj <- c(0, 0.5)
1388          }
1389          else {
1390            Y <- yTop - dLeaf * lab.cex
1391            X <- xTop
1392            srt <- 90
1393            adj <- 1
1394          }
1395          nodeText <- asTxt(attr(subtree, "label"))
1396          text(X, Y, nodeText, xpd = lab.xpd, srt = srt,
1397               adj = adj, cex = lab.cex, col = lab.col, font = lab.font)
1398        }
1399      }
1400      else if (inner) {
1401        segmentsHV <- function(x0, y0, x1, y1) {
1402          if (horiz)
1403            segments(y0, x0, y1, x1, col = col, lty = lty,
1404                     lwd = lwd)
1405          else segments(x0, y0, x1, y1, col = col, lty = lty,
1406                        lwd = lwd)
1407        }
1408        for (k in seq_along(subtree)) {
1409          child <- subtree[[k]]
1410          yBot <- attr(child, "height")
1411          if (getOption("verbose"))
```

```
1412            cat("ch.", k, "@ h=", yBot, "; ")
1413          if (is.null(yBot))
1414            yBot <- 0
1415          xBot <- if (center)
1416            mean(bx$limit[k:(k + 1)])
1417          else bx$limit[k] + .midDend(child)
1418          hasE <- !is.null(ePar <- attr(child, "edgePar"))
1419          if (!hasE)
1420            ePar <- edgePar
1421          i <- if (!is.leaf(child) || hasE)
1422            1
1423          else 2
1424          col <- Xtract("col", ePar, default = par("col"),
1425                        i)
1426          lty <- Xtract("lty", ePar, default = par("lty"),
1427                        i)
1428          lwd <- Xtract("lwd", ePar, default = par("lwd"),
1429                        i)
1430          if (type == "triangle") {
1431            segmentsHV(xTop, yTop, xBot, yBot)
1432          }
1433          else {
1434            segmentsHV(xTop, yTop, xBot, yTop)
1435            segmentsHV(xBot, yTop, xBot, yBot)
1436          }
1437          vln <- NULL
1438          if (is.leaf(child) && leaflab == "textlike") {
1439            nodeText <- asTxt(attr(child, "label"))
1440            if (getOption("verbose"))
1441              cat("-- with \"label\"", format(nodeText))
1442            hln <- 0.6 * strwidth(nodeText, cex = lab.cex)/2
1443            vln <- 1.5 * strheight(nodeText, cex = lab.cex)/2
1444            rect(xBot - hln, yBot, xBot + hln, yBot +
1445                   2 * vln, col = p.col)
1446            text(xBot, yBot + vln, nodeText, xpd = lab.xpd,
1447                 cex = lab.cex, col = lab.col, font = lab.font)
1448          }
1449          if (!is.null(attr(child, "edgetext"))) {
1450            edgeText <- asTxt(attr(child, "edgetext"))
1451            if (getOption("verbose"))
1452              cat("-- with \"edgetext\"", format(edgeText))
```

```
1453            if (!is.null(vln)) {
1454              mx <- if (type == "triangle")
1455                (xTop + xBot + ((xTop - xBot)/(yTop -
1456                                                yBot)) * vln)/2
1457                else xBot
1458              my <- (yTop + yBot + 2 * vln)/2
1459            }
1460            else {
1461              mx <- if (type == "triangle")
1462                (xTop + xBot)/2
1463                else xBot
1464              my <- (yTop + yBot)/2
1465            }
1466            p.col <- Xtract("p.col", ePar, default = "white",
1467                            i)
1468            p.border <- Xtract("p.border", ePar, default = par("fg"),
1469                               i)
1470            p.lwd <- Xtract("p.lwd", ePar, default = lwd,
1471                            i)
1472            p.lty <- Xtract("p.lty", ePar, default = lty,
1473                            i)
1474            t.col <- Xtract("t.col", ePar, default = col,
1475                            i)
1476            t.cex <- Xtract("t.cex", ePar, default = 1,
1477                            i)
1478            t.font <- Xtract("t.font", ePar, default = par("font"),
1479                             i)
1480            vlm <- strheight(c(edgeText, "h"), cex = t.cex)/2
1481            hlm <- strwidth(c(edgeText, "m"), cex = t.cex)/2
1482            hl3 <- c(hlm[1L], hlm[1L] + hlm[2L], hlm[1L])
1483            if (horiz) {
1484              polygon(my + c(-hl3, hl3), mx + sum(vlm) *
1485                        c(-1L:1L, 1L:-1L), col = p.col, border = p.border,
1486                      lty = p.lty, lwd = p.lwd)
1487              text(my, mx, edgeText, cex = t.cex, col = t.col,
1488                   font = t.font)
1489            }
1490            else {
1491              polygon(mx + c(-hl3, hl3), my + sum(vlm) *
1492                        c(-1L:1L, 1L:-1L), col = p.col, border = p.border,
1493                      lty = p.lty, lwd = p.lwd)
```

```
1494              text(mx, my, edgeText, cex = t.cex, col = t.col,
1495                   font = t.font)
1496            }
1497          }
1498        }
1499      }
1500      if (inner && length(subtree)) {
1501        KK[depth] <- length(subtree)
1502        if (storage.mode(kk) != storage.mode(KK))
1503          storage.mode(kk) <- storage.mode(KK)
1504        kk[depth] <- 1L
1505        x1 <- bx$limit[1L]
1506        x2 <- bx$limit[2L]
1507        subtree <- subtree[[1L]]
1508      }
1509      else {
1510        repeat {
1511          depth <- depth - 1L
1512          if (!depth || kk[depth] < KK[depth])
1513            break
1514        }
1515        if (!depth)
1516          break
1517        length(kk) <- depth
1518        kk[depth] <- k <- kk[depth] + 1L
1519        x1 <- llimit[[depth]][k]
1520        x2 <- llimit[[depth]][k + 1L]
1521        subtree <- wholetree[[kk]]
1522      }
1523    }
1524    invisible()
1525  }
1526
1527  # Fonction embarquee pour la production des heatmaps
1528  plotNodeLimit <- function (x1, x2, subtree, center)
1529  {
1530    inner <- !is.leaf(subtree) && x1 != x2
1531    limit <- c(x1, if (inner) {
1532      K <- length(subtree)
1533      mTop <- .memberDend(subtree)
1534      limit <- integer(K)
```

```
1535      xx1 <- x1
1536      for (k in 1L:K) {
1537        m <- .memberDend(subtree[[k]])
1538        xx1 <- xx1 + (if (center) (x2 - x1) * m/mTop else m)
1539        limit[k] <- xx1
1540      }
1541      limit
1542    } else x2)
1543    mid <- attr(subtree, "midpoint")
1544    center <- center || (inner && !is.numeric(mid))
1545    x <- if (center)
1546      mean(c(x1, x2))
1547    else x1 + (if (inner)
1548      mid
1549      else 0)
1550    list(x = x, limit = limit)
1551  }
1552
1553  # Fonction embarquee pour la production des heatmaps
1554  .midDend <- function (x) {
1555    if (is.null(mp <- attr(x, "midpoint"))) 0 else mp}
```

### 6.2.2 ScTrimReads

```
1  #!/usr/bin/env perl
2  use Modern::Perl '2011';
3  use autodie;
4  use Smart::Comments;
5  use Term::ProgressBar;
6  use File::Basename;
7
8   unless (@ARGV == 7) {
9    die <<"EOT";
10   Usage: $0 <barcode-fileList> <read_1.fastq> <read_2.fastq> <output_directory> <
         mismatch> <starting_barcodes> <precicion>
11   This tool separates seconds reads in separated files in accordance to barcodes
12   on corresponding firsts reads.
13   starting_barcodes define the start point in the barcode list in case of multiple
          steps trimming.
```

```
14    precision define the pre-trimming efficency of barcodes for the trimming
          acceleration.
15    Example: $0 ~/barcode_list ~/SRR089752_R1.fastq ~/SRR089752_R2.fastq ~/MyCells 2
          1 3
16  EOT
17    }
18  #------------------------ Tool-Box-----------------------------------------
19  my %seq_for;
20  my $barcodefile = shift;
21  my $readfile1 = shift;
22  my $readfile2 = shift;
23  my $outdir = shift;
24  my $mistmatch = shift;
25  my $lib = shift;
26  $mistmatch = $mistmatch +1;
27  $outdir = substr $outdir, 0 if $outdir =~ s/\/$//;    # retire le dernier slash si
        present
28  system "mkdir -p $outdir";
29  ### Estimating process time...
30  # defini le nombre de ligne dans le fichier d'entree pour la progressbar
31  my $total;
32  if ($readfile1 =~ /.gz$/) {
33    $total =  `zcat $readfile1 | wc -l` ;
34  }
35  else{
36    $total =  `wc -l $readfile1` ;
37  }
38  ($total) = split ' ', $total;
39  my $progress = Term::ProgressBar->new($total);
40  my $map = 0 ;
41  my $precision = shift;
42  #------------------------ Load Barcode File-List---------------------------
43  open my $in, '<', $barcodefile;
44  my %barcode_for;
45  my $i = $lib;
46  LINE:
47  my %supercode;
48  #Produire un hash avec le debut du barcode en clefs pour accelerer la recherche
        par categorie de barcode
49  while (my $line = <$in>){
50    chomp $line;
```

```
51    my $sub = substr $line,0,$precision;
52    $line = "$line.$i";
53    push @{$barcode_for{$sub}}, $line;
54    $i = $i+1;
55  }
56  close $in;
57  #---------------------- Load Read 1 File ---------------------------------
58  my ($in2, $in3);
59  if ($readfile1 =~ /.gz$/) {
60  open( $in2, "gunzip -c $readfile1 |") || die "cant open pipe to $readfile1";
61  }
62  else {
63  open $in2, '<', $readfile1;
64  }
65
66  if ($readfile2 =~ /.gz$/) {
67  open( $in3, "gunzip -c $readfile2 |") || die "cant open pipe to $readfile2";
68  }
69  else {
70  open $in3, '<', $readfile2;
71  }
72  #~ open my $in2, '<', $readfile1;
73  #~ open my $in3, '<', $readfile2;
74
75  my $id;
76
77  # lecture synchronisee des deux fichiers de reads
78  LINE2:
79  while (my $line = <$in2>){
80    my $read2id = <$in3>;
81    my $read2seq = <$in3>;
82    my $read2id2 = <$in3>;
83    my $read2qual = <$in3>;
84    my $read1 = <$in2>;
85    my ($number) = $line =~ /^.*\.(\d+).*$/; # pour la progressbar
86
87    my $tmpcode = substr ($read1,0,$precision); # recupere le debut du premier reads
          pour chercher dans les clefs de hash de barcode
88
89      if (exists $barcode_for{$tmpcode}){   # recherche du barcode si disponible
            selon le debut du premier reads (pour accelerer)
```

```
90
91
92        for my $barcod (@{$barcode_for{$tmpcode}}){
93        my ($code) = $barcod =~ /(^.*?)\..*$/;
94        my  $count = 0 ;
95        my $p = length $code;
96        my ($cell) = $barcod =~ /^.*\.(\d+)/;
97
98
99        if (substr ($read1,0,$p) eq $code) {
100           $progress->update($number*4);
101           $map = $map +1;
102         push @{$seq_for{$cell}}, "$read2id$read2seq$read2id2$read2qual";
103         }
104       else {
105
106
107         if ($mistmatch ne 0){  # si il y a une demande de mismatch (attention
              ralentissement)
108           LOOK:
109           for (my $i =0 ; $i <= length $code ; $i++) {
110             if ($count < $mistmatch) {
111               my $la = substr $code, $i,1;
112               my $lb = substr $read1, $i,1;
113                 if ($la ne $lb) {
114                   $count = $count + 1 ;
115                 }
116                 else {
117                 $tmpcode = "$tmpcode$la" unless $i == 0;
118                 }
119             }
120             else {
121               last LOOK;
122               }
123         }
124         if ($count < $mistmatch ) {
125           $progress->update($number*4);
126           $map = $map +1;
127         push @{$seq_for{$cell}}, "$read2id$read2seq$read2id2$read2qual";
128         }
129       }
```

```
130         }
131       }
132     }
133     my $line3 = <$in2>;
134     my $line4 = <$in2>;
135 }
136 close $in2;
137 my $accuracy = ($map/($total/4))*100; # defini la proportion de reads attribues
138 say " \n Efficiency : $accuracy % reads attributed";
139 my ($name) = fileparse($readfile2, qr{\.[^.]*}xms);
140 
141 # export des fichiers de reads 2 tries.
142 for (my $o = 1; $o <= $i-1 ; $o++) { ### Elapsed time |===[%]
143   if ( exists $seq_for{$o}) {
144     open my $out, '>>', "$outdir/$name.Cell_$o.fastq";
145     say {$out} @{$seq_for{$o}};
146     close $out;
147   }
148 }
149     open my $out2, '>>', "$outdir/accuracy5";
150     say {$out2} "$accuracy % reads attributed";
151     close $out2;
```

### 6.2.3 MapPipeline

Cet outil a été développé pour effectuer les étapes de tri de reads et de maping de façon automatisée à partir d'un très grand nombre de fichiers à processer.

```
1 #!/usr/bin/env perl
2 use Modern::Perl '2011';
3 use autodie;
4 use Smart::Comments;
5 use File::Basename;
6 use Getopt::Euclid;
7 
8 my $dirname = $ARGV{'<directory>'};
9 $dirname = substr $dirname, 0 if $dirname =~ s/\/$//;   # retire le dernier slash
      si present
10 
11 
```

```
my $espece = $ARGV{'<species>'};
my @dirs;
for my $name (@{$ARGV{'--dirs'}}){
        push @dirs, "$name";
    }
@dirs = sort @dirs;
for my $soft (@{$ARGV{'--soft'}}){
  say "\n\n\n #------------------------- $soft -------------------------#";
  trimgalor() and next if $soft eq "trimgalor";
  star()  and next    if $soft eq "star";
    }
my @proc_files;
my @auto_mode_detect;

#-------------------------------------------------------------------------TRIMGALOR
    ---------------------------------------------------------------------------

sub trimgalor {
  my @sub_dirs =  @dirs;
  my @detec_files;
  if (@sub_dirs){
        for my $dir (@sub_dirs){
        push @detec_files ,`find "$dirname/$dir"|grep -e .fq.gz -e fastq.gz`;
            push @auto_mode_detect, "\tWith Paired-end mode\n" if `find "$dirname/
                $dir"|grep -e .fq.gz -e fastq.gz | wc -l` == 2 ;
            push @auto_mode_detect, "\tWith Single-end mode\n" if `find "$dirname/
                $dir"|grep -e .fq.gz -e fastq.gz | wc -l` == 1 ;
            push @auto_mode_detect, "\tWarning With undetectable process type! \n"
                 if `find "$dirname"|grep -e .fq.gz -e fastq.gz | wc -l` < 3 ;
    }
  }
  else{                                                                  #
      Detection de tout les fichiers fastq dans le repertoire d'analyse.
      push @detec_files ,`find $dirname | grep -e .fastq -e .fq `;
  }
  for my $file (@detec_files){                    # Verifie que les fichiers
      detectes n'ont pas deja ete tries.
    chomp $file;
    my($filename,$dirs) = fileparse($file);
    ($dirs) = $dirs =~ /.*\/(.*)\/\z/;
    my $test = `find ~/WorkSpace/Data/TRIMGALOR | grep -E $dirs | grep -E '
        sortedByCoord.out.bam'`;
```

```
46      push @proc_files, "$file\n" unless $test and (say "\n !Warning! $filename has
            already been Trimmed!" and die);
47    }
48        my @tmp_proc_files = sort @proc_files;
49        my @tmp_auto_mode_detect = @auto_mode_detect;
50        splice @proc_files;
51    my $len = length (@tmp_proc_files);
52    say "\n # $len Files to Trim : \n @tmp_proc_files";
53    say "\n # Triming Mode: \n @tmp_auto_mode_detect";
54
55    while (my $detect = shift @tmp_auto_mode_detect) {
56      if ($detect =~ m/Paired-end mode/) {
57        my ($read1,$read2) = splice(@tmp_proc_files,0,2);
58                chomp $read1; chomp $read2;
59                my ($basename, $dirs) = fileparse($read1);
60                my $out_dir ;
61                if ($especе eq "Zebrafish"){$out_dir = "~/WorkSpace/Data/TRIMGALOR/
                      Zebrafish$1" if $dirs =~ /Zebrafish(\/.*)\z/};
62                if ($especе eq "Human"){$out_dir = "~/WorkSpace/Data/TRIMGALOR/Human
                      $1" if $dirs =~ /Human(\/.*)\z/;}
63                if ($especе eq "Mouse"){$out_dir = "~/WorkSpace/Data/TRIMGALOR/Mouse
                      $1" if $dirs =~ /Mouse(\/.*)\z/s;}
64                system "mkdir -p $out_dir";
65                my ($prefix) =  $dirs =~ /.*\/(.*)\/\z/;
66                system "trim_galore --gzip --paired --path_to_cutadapt /root/
                      miniconda3/bin/cutadapt --fastqc  --output_dir $out_dir $read1 $
                      read2";
67            say " \n$read1 and $read2 Trimmed";
68            ($dirname) = $out_dir =~ /(.*)\/.*\/\z/;
69            push @proc_files, "$out_dir$prefix\_val_1.fq.gz\n";
70            push @proc_files, "$out_dir$prefix\_val_2.fq.gz\n";
71
72      }
73
74      if ($detect =~ m/Single-end mode/) {
75          my $read1 = shift(@tmp_proc_files) ;
76              chomp $read1;
77              my ($basename, $dirs) = fileparse($read1);
78              my $out_dir ;
79              if ($especе eq "Zebrafish"){$out_dir = "~/WorkSpace/Data/TRIMGALOR/
                      Zebrafish$1" if $dirs =~ /Zebrafish(\/.*)\z/ ;}
```

```
80              if ($espece eq "Human"){$out_dir = "~/WorkSpace/Data/TRIMGALOR/Human$1
                    " if $dirs =~ /Human(\/.*)\z/;}
81              if ($espece eq "Mouse"){$out_dir = "~/WorkSpace/Data/TRIMGALOR/Mouse$1
                    " if $dirs =~ /Mouse(\/.*)\z/s;}
82              system "mkdir -p $out_dir";
83              $read1 = $read1 =~ tr/\n//dr;
84              my ($prefix) =  $dirs =~ /.*\/(.*)\/\z/;
85              system "trim_galore --gzip --path_to_cutadapt /root/miniconda3/bin/
                    cutadapt --fastqc  --output_dir $out_dir $read1";
86              say " $read1 Trimed";
87              ($dirname) = $out_dir =~ /(.*)\/.*\/\z/;
88              push @proc_files, "$out_dir$prefix\_trimmed.fq.gz\n";
89
90      }
91    }
92    @proc_files = sort @proc_files;
93  }
94
95  #-------------------------------------------------------------------STAR
        ------------------------------------------------------------------------------


96  sub star {
97
98    unless (@proc_files){
99        my @sub_dirs = @dirs;
100       my @detec_files;
101       say @sub_dirs;
102       say $dirname;
103       if (@sub_dirs){
              # Detection des fichiers fastq dans des dossiers specifie.
104            for my $dir (@sub_dirs){
105                push @detec_files ,`find "$dirname/$dir"|grep -e .fq.gz`;
                       @detec_files = sort @detec_files;
106                    push @auto_mode_detect, "\tWith Paired-end mode\n" if `find "$
                           dirname/$dir"|grep -e .fq.gz | wc -l` == 2 ;
107                    push @auto_mode_detect, "\tWith Single-end mode\n" if `find "$
                           dirname/$dir"|grep -e .fq.gz | wc -l` == 1 ;
108                    push @auto_mode_detect, "\tWarning With undetectable process
                           type! \n" if `find "$dirname"|grep -e .fq.gz | wc -l` < 3 ;
109           }
110       }
```

```
111       else{                                                                    #
              Detection de tout les fichiers fastq dans le repertoire d'analyse.
112           push @detec_files ,`find $dirname | grep -e .fq.gz -e fastq.gz `;
113       }
114       for my $file (@detec_files){                    # Verifie que les fichiers
              detectes n'ont pas deja ete tries.
115           chomp $file;
116             my($filename,$dirs) = fileparse($file);
117             ($dirs) = $dirs =~ /.*\/(.*)\/\z/;
118           my $test = `find ~/WorkSpace/Data/STAR | grep -E $dirs | grep -E '
                  sortedByCoord.out.bam'`;
119           push @proc_files, "$file\n" unless $test and (say "\n !Warning! $
                  filename has already been Maped!" and die);
120       }
121     }
122           my @tmp_proc_files = sort @proc_files;
123       my @tmp_auto_mode_detect = @auto_mode_detect;
124       my $len = length (@tmp_proc_files);
125
126   system "STAR --genomeDir ~/WorkSpace/SourceData/Index_Genom/$espece/ --
          runThreadN 8 --genomeLoad LoadAndExit";
127
128   say "\n # $len Files to Map : \n @tmp_proc_files";
129   say "\n # Maping Mode: \n @tmp_auto_mode_detect";
130   while (my $detect = shift @tmp_auto_mode_detect) {
131     if ($detect =~ m/Paired-end mode/) {
132       my ($read1,$read2) = splice(@tmp_proc_files,0,2);
133               chomp $read1; chomp $read2;
134               my ($basename, $dirs) = fileparse($read1);
135               my $out_dir ;
136               if ($espece eq "Zebrafish"){$out_dir = "~/WorkSpace/Data/STAR/
                      Zebrafish$1" if $dirs =~ /Zebrafish(\/.*)\z/};
137               if ($espece eq "Human"){$out_dir = "~/WorkSpace/Data/STAR/Human$1"
                      if $dirs =~ /Human(\/.*)\z/;}
138               if ($espece eq "Mouse"){$out_dir = "~/WorkSpace/Data/STAR/Mouse$1"
                      if $dirs =~ /Mouse(\/.*)\z/s;}
139               system "mkdir -p $out_dir";
140               my $prefix =  $dirs =~ /.*\/(.*)\/\z/;
141               system "STAR --genomeDir ~/WorkSpace/SourceData/Index_Genom/$espece/
                      --runThreadN 8 --genomeLoad NoSharedMemory --outSAMtype BAM
                      SortedByCoordinate --readFilesIn $read1 $read2 --
```

```
                readFilesCommand zcat --seedSearchStartLmax 12 --
                outFilterScoreMinOverLread 0.3 --alignSJoverhangMin 15 --
                outFilterMismatchNmax 33 --outFilterMatchNminOverLread 0 --
                outFilterType BySJout --outSAMunmapped Within --outSAMattributes
                 NH HI AS NM MD --outSAMstrandField intronMotif --quantMode
                TranscriptomeSAM GeneCounts --outFileNamePrefix $out_dir/$
                prefix.";
142         say " \n\n $read1 and $read2 processed \n\n";
143     }
144
145
146   if ($detect =~ m/Single-end mode/) {
147         my $read1 = shift(@tmp_proc_files) ;
148             chomp $read1;
149             my ($basename, $dirs) = fileparse($read1);
150             my $out_dir ;
151             if ($espece eq "Zebrafish"){$out_dir = "~/WorkSpace/Data/STAR/
                Zebrafish$1" if $dirs =~ /Zebrafish(\/.*)\z/ ;}
152             if ($espece eq "Human"){$out_dir = "~/WorkSpace/Data/STAR/Human$1"
                if $dirs =~ /Human(\/.*)\z/;}
153             if ($espece eq "Mouse"){$out_dir = "~/WorkSpace/Data/STAR/Mouse$1"
                if $dirs =~ /Mouse(\/.*)\z/s;}
154             system "mkdir -p $out_dir";
155             $read1 = $read1 =~ tr/\n//dr;
156             my $prefix =  $dirs =~ /.*\/(.*)\/\z/;
157             system "STAR --genomeDir ~/WorkSpace/SourceData/Index_Genom/$espece/
                 --runThreadN 8 --genomeLoad NoSharedMemory --outSAMtype BAM
                SortedByCoordinate --readFilesIn $read1 --readFilesCommand zcat
                --seedSearchStartLmax 12 --outFilterScoreMinOverLread 0.3 --
                alignSJoverhangMin 15 --outFilterMismatchNmax 33 --
                outFilterMatchNminOverLread 0 --outFilterType BySJout --
                outSAMunmapped Within --outSAMattributes NH HI AS NM MD --
                outSAMstrandField intronMotif --quantMode TranscriptomeSAM
                GeneCounts --outFileNamePrefix $out_dir/$prefix.";
158         say "\n\n $read1 processed \n\n";
159
160     }
161   }
162   system "STAR --genomeDir ~/WorkSpace/SourceData/Index_Genom/$espece/ --
        runThreadN 8 --genomeLoad Remove";
163 }
```

```
=head1 NAME

Parser - Display Blast Results from get_parser tool.

=head1 VERSION

  VERSION 0.01

=head1 USAGE

  autoSTAR <directory> <especce> [options]

=head1 REQUIRED ARGUMENTS

=over

=item <directory>

Path to general directory to scan.
Without any options, this tool will find every processable files (with
fq.gz or fastq.gz extension) files in the directory, or in all existant
subdirectories.

=for Euclid:
  directory.type: readable

=item <species>

Name of especce to refer for reference genome.

The available species to display are:

  0. Zebrafish
  1. Human
  2. Mouse

=for Euclid:
  species.type: string

=item --soft [=] <name>...

```

```
Softawre or list of software to pipe to selected datas with the selected order.

The available softwares are:

  0. fastQc
  1. trimgalor
  2. star
  3. featurecounts
  4. qualimap



=back



=head1 OPTIONS

=over
=item --dirs [=] <name>...

List of white-spaced separated subdirectories to scan.



Example:

  --dirs Acinar_1 Alpha_1 Beta_2

=item --version

=item --usage

=item --help

=item --man

Print the usual program information

=back
```

```
246
247 =head1 AUTHOR
248
249 Wayet (jwayet@student.uliege.ac.be)
250
251 =head1 BUGS
252
253 There are undoudtedly serious bgs lurking somewhere in this code.
254 Bug reports and other feedback are most welcome.
255
256 =head1 COPYRIGHT
257
258 Copyright (c) 2013, Wayet. All Rights Reserved.
259 This program is free software. It may be used, redistributed
260 and/or modified under the terms of the Perl Artistic License
261 (see http://www.perl.com/perl/misc/Artistic.html)
```

### 6.2.4 SingleSeqPrep

Cet outil a été développé afin de rassembler les données transcriptomiques de milliers de fichiers de cellules en une matrice de comptage unique qui est ensuite exploitée pour le clustering via le système StemID.

```
1  #!/usr/bin/env perl
2  use Modern::Perl '2011';
3  use autodie;
4  use Smart::Comments '###';
5  use File::Basename;
6  use LWP::Simple 'get';
7  use Path::Class 'file';
8  use Getopt::Euclid;
9
10
11
12
13 # Load Url's
   ------------------------------------------------------------------------
14 my $dirname = $ARGV{'<directory>'};
15 $dirname = substr $dirname, 0 if $dirname =~ s/\/$//;
16 my @dirs;
```

```
for my $name (@{$ARGV{'--dirs'}}){
        push @dirs, "$name";
      }
@ dirs = sort @dirs;
my $output = $ARGV{'<output_name>'};
my $regex = $ARGV{'<Regex>'};
my $headname = $ARGV{'<prefix>'};
my @headers;
my @detec_files;
if (@dirs){
  for my $dir (@dirs){
    push @detec_files , `find "$dirname/$dir"|grep -e ReadsPerGene.out.tab`;
        @detec_files = sort @detec_files;
    for my $i (@detec_files){
  my ($basename) = fileparse($i);
  my ($tmp) = $basename =~ /^.*.$regex(\d+)_.*$/;
  my $tmp2 = "$headname$tmp";
  push @headers, $tmp2;
  }
  }
}
else{
  push @detec_files , `find $dirname | grep -e ReadsPerGene.out.tab `; @detec_
      files = sort @detec_files;
  for my $i (@detec_files){
  my ($basename) = fileparse($i);
  my ($tmp) = $basename =~ /^(.*)$/;
  my $tmp2 = "$headname$tmp";
  push @headers, $tmp2;
  }
}

### Files to Prep: @headers
### $output
# Load first file
    ------------------------------------------------------------------------


my $first = shift @detec_files;
### $first
chomp $first;
my %files;
```

```
54 my $di= join "\t","                    ", @headers; # permet l'entree des headers du
       fichier dans le bon format
55 $files{"ENSMUSG00000000000"} = [$di];      # Assure que les headers ne seront pas
      detruit a cause d'un identifiant existant qui serra reecrit.
56 open my $in, '<', $first;
57 LINE:
58 while (my $line = <$in>){
59   chomp $line;
60       if  (substr($line,0,1) eq 'E'){
61         my ($id, $count) = split /\t/, $line;
62         $files{$id} = [$id, $count];
63         next LINE;
64       }
65     }
66 close $in;
67
68
69 # Load next files
       ------------------------------------------------------------------------------
70
71 for my $detec_file (@detec_files){ ### Elapsed time |===[%]
72   chomp $detec_file;
73 proc_file($detec_file);
74 }
75
76 # Zone fonction
       ------------------------------------------------------------------------------
77 sub proc_file{
78     ## Reading input file: $infile
79   open my $in, '<', $_[0];
80     my @counts;
81   LINE:
82   while (my $line = <$in>){
83     chomp $line;
84         if  (substr($line,0,1) eq 'E'){
85           my ($id, $count) = split /\t/, $line;
86           my $temp = $files{$id};
87           $temp = join "\t", @$temp, $count;
88           $files{$id} = [$temp];
```

```
89              next LINE;
90            }
91          }
92    close $in;
93    return
94  }
95
96  # Output
        ------------------------------------------------------------------------------
97
98  my $outfile = "$output.prep";
99  open my $out, '>', $outfile;
100 for my $id (sort keys %files) {
101   my $temp = $files{$id};
102   #say {$out} "\t\t\t" . join "\t", @dirs;
103   say {$out}  join "\t", @$temp;
104 }
105 close $out;
106 =head1 NAME
107
108   This tool export a file containing readcounts from several sources.
109   It requires a directory to locate sources files, coma separetade list
110   of subdirectories containing sources files and an output file name.
111
112 =head1 VERSION
113
114   VERSION 0.01
115
116 =head1 USAGE
117
118   Single_Ceq_prep <directory> <output_name> <Regex> [options]
119
120 =head1 REQUIRED ARGUMENTS
121
122 =over
123
124 =item <directory>
125
126 Path to general directory to scan.
127 Without any options, this tool will find every processable files (with
```

```
'ReadsPer' in file names) in the directory, or in all existant
subdirectories.

=for Euclid:
  directory.type: readable

=item <output_name>

Name to give to output file.

=for Euclid:
  output_name.type: string

=item <Regex>

Define a field just before the cell number to catch this id.

Example: ERR\d+

=for Euclid:
  Regex.type: string

=item <prefix>

Set the prefix fore headers followed by cell number.

Example: Muraro_D1_

=for Euclid:
  prefix.type: string


=back




=head1 OPTIONS

=over
=item --dirs [=] <name>...
```

```
169
170 List of white-spaced separated subdirectories to scan.
171
172
173
174 Example:
175
176   --dirs Acinar_1 Alpha_1 Beta_2
177
178 =item --version
179
180 =item --usage
181
182 =item --help
183
184 =item --man
185
186 Print the usual program information
187
188 =back
189
190 =head1 AUTHOR
191
192 Wayet (jwayet@student.uliege.ac.be)
193
194 =head1 BUGS
195
196 There are undoudtedly serious bgs lurking somewhere in this code.
197 Bug reports and other feedback are most welcome.
198
199 =head1 COPYRIGHT
200
201 Copyright (c) 2013, Wayet. All Rights Reserved.
202 This program is free software. It may be used, redistributed
203 and/or modified under the terms of the Perl Artistic License
204 (see http://www.perl.com/perl/misc/Artistic.html)
```

### 6.2.5 GeneIDtoName

Ce petit outil a été développé pour convertir les identifiants de gènes ENSEMBL en leurs noms correspondants.

```
#!/usr/bin/env perl
use Modern::Perl '2011';
use autodie;
use Smart::Comments '###';
use File::Basename;
use LWP::Simple 'get';
use Path::Class 'file';


unless (@ARGV == 2) {
  die <<"EOT";
  Usage: $0 <Conversion.table> <File_to_convert>
  This tool replace the IDs of recognized genes by their conventional
  names.
  Example: ./GeneIDtoName  ~/Dconvert.table  ~/MyFile

EOT
  }

my $table = shift;
my $file  = shift;

my %convert_for;
my @output;

open my $table_in, '<', $table;
TABLE:
while (my $line = <$table_in>){
  chomp $line;
  my ($id,$name) = split ',', $line;
  $convert_for{$id} = $name
}

open my $file_in, '<', $file;
TABLE:
while (my $line = <$file_in>){
  chomp $line;
```

```
38    my ($id) = split "\t", $line;
39    my ($name)= $convert_for{$id};
40    if ($name){
41      ($line) = $line =~ /^ENS.*?\t(.*)/;
42      push @output, (join "\t", $name, "$line\n");
43    }
44    else {
45      push @output, "$line\n";
46    }
47  }
48
49  my $outfile = "$file.cv";
50  open my $out, '>', $outfile;
51  say {$out}  @output;
52  close $out;
```

### 6.2.6 CrossSpeciesComparator

Cet outil a été développé pour comparer deux espèces selon une table d'orthologie en pairwise.

```
1  #!/usr/bin/env perl
2  use Modern::Perl '2011';
3  use autodie;
4  use Smart::Comments '###';
5  use File::Basename;
6  use LWP::Simple 'get';
7  use Path::Class 'file';
8
9  my $inputMan = shift;
10 my $inputMouse = shift;
11 my $inputList = shift;
12
13 my $output = shift;
14
15 my @listH;
16
17 open my $in, '<', $inputMan;
18 my $s = 0;
19 LINE:
20 while (my $line = <$in>){
```

```
21    chomp $line;
22
23    my $res = `cat $inputList | grep -e $line`;
24    if ($res =~ m/$line/){
25      $s = $s+1;
26      push @listH, split "\n", $res;
27      }
28  }
29  close $in;
30    print "$s corresponding \n";
31
32  my @listM;
33    open my $in2, '<', $inputMouse;
34   $s = 0;
35  LINE:
36  while (my $line2 = <$in2>){
37    chomp $line2;
38
39    my $res = `cat $inputList | grep -e $line2`;
40    if ($res =~ m/$line2/){
41      $s = $s+1;
42      push @listM, $line2;
43      }
44  }
45  close $in2;
46    print "$s corresponding \n";
47  open my $out, '>', "$output";
48
49  for my $test (@listM){
50    for my $line (@listH){
51      if ($line =~ m/$test/){
52        #~ print "$line\n";
53        chomp $line;
54        say {$out}  $line;
55      }
56    }
57  }
58   close $out;
```

### 6.2.7 Exemple d'utilisation du script EGMA

Voici une illustration de l'exécution du script EGMA. L'entrée prend un fichier de niveaux d'expression de réplicas par type cellulaire pour chaque étude ainsi qu'un fichier de description de l'expérience.

Le fichier de description est formaté en quatre colonne correspondant respectivement aux nom de l'échantillons/réplica, au type cellulaire, à la source des données et enfin à l'espèce étudiée.

```
source("~/EGMA.R")

#------------- analyse run-----------------------------------------
#chargement de la matrice de comptage des replicas pour les types cellulaires de
    differentes etudes.

cts <- read.table("mytable.csv", header = T)


# chargement du fichier descriptif de l'experience
coldata <- read.table("myDescr.csv", header = T)

# production des listes de genes enrichis et des VenPLot
myEGs <- EGMAMetaAnalysis(cts,coldata,padj = 0.1, log2fold = 1,myconv,batch = F)

# Comparaison des profils transcriptomiques
myTransCross <- EGMATransComp(cts,coldata,padj = 0.1, log2fold = 1,myconv,batch =
    F, cormethod = "SERE", PCA = F)
```