

Tools and techniques for single-cell RNA sequencing data

Luke Zappia

ORCID ID: 0000-0001-7744-8565

Doctor of Philosophy

August 2019

School of Biosciences
The University of Melbourne

Submitted in Total Fulfillment of the Requirements of the Degree of Doctor of
Philosophy

Abstract

RNA sequencing of individual cells allows us to take a snapshot of the dynamic processes within a cell and explore differences between cell types. As this technology has developed over the last few years it has been rapidly adopted by researchers in areas such as developmental biology, and many single-cell RNA sequencing datasets are now available. Coinciding with the development of protocols for producing single-cell RNA sequencing data there has been a simultaneous burst in the development of computational analysis methods. My thesis explores the computational tools and techniques for analysing single-cell RNA sequencing data. I present a database that charts the release of analysis software, where it has been published and what it can be used for, as well as a website that makes this information publicly available. I also present two of my own tools and techniques including Splatter, a software package for easily simulating single-cell datasets from multiple models, and clustering trees, a visualisation approach for inspecting clustering at multiple resolutions. In the final part of my thesis I perform analysis of a dataset from kidney organoids to demonstrate and compare some current analysis methods. Taken together, my thesis covers many aspects of the tools and techniques for single-cell RNA sequencing by describing the approaches that are available, presenting software that can help in developing and evaluating methods, introducing an approach for aiding one of the most common analysis tasks, and showing how tools can be used to extract meaning from a real dataset.

Declaration

This is to certify that:

- i. this thesis comprises only my original work towards the degree of Doctor of Philosophy except where indicated in the preface;
- ii. due acknowledgement has been made in the text to all other material used; and
- iii. this thesis is fewer than the 100,000 words in length, exclusive of tables, maps, bibliographies and appendices

Luke Zappia B.Sc, M.Sc

August 2019

Preface

This preface provides a summary of the chapters in this thesis and describes my contribution to them as well as the contributions of my collaborators and supervisors. This is a thesis *with* publication and where publications form part of a chapter they are listed here. The following publications are included as part of this thesis:

Zappia L, Phipson B, Oshlack A. “Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database.” *PLoS Computational Biology*. 2018. DOI: 10.1371/journal.pcbi.1006245

Zappia L, Phipson B, Oshlack A. “Splatter: simulation of single-cell RNA sequencing data.” *Genome Biology*. 2017. DOI: 10.1186/s13059-017-1305-0

Zappia L, Oshlack A. “Clustering trees: a visualization for evaluating clusterings at multiple resolutions.” *GigaScience*. 2018. DOI: 10.1093/gigascience/giy083

These publications are included as they appear online and are designed to be read as stand-alone documents. Sections within these publications are not included in the table of contents, and references are available at the end of each publication rather than in the reference list for this thesis. The contributions of authors to these papers are explained below. I am the first author on these publications and contributed more than 50 percent of the work towards them including drafting, editing and revising the manuscripts. My co-authors have provided signed declarations acknowledging and supporting my contributions which have been submitted along with this thesis. Where publicly available datasets have been used these have been appropriately cited.

Chapter 1: Introduction is an original work providing a background and overview relevant to understanding my work in this thesis including an introduction to RNA sequencing, single-cell RNA sequencing and kidney function and development.

Chapter 2: The scRNA-seq tools landscape is an original work describing a database of software tools for analysing single-cell RNA sequencing data which has been published in *PLoS Computational Biology* as “Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database”. In addition to this publication I developed a website displaying the information in this database which is available at <https://www.scRNA-tools.org>. The database and code for building the website is available on GitHub at <https://github.com/Oshlack/scRNA-tools> under an MIT license.

Contributions to the work in this chapter:

- I compiled and regularly updated the database of tools.
- I designed and built the public website used to display the database. Breon Schmidt provided assistance with implementing some of the website functionality. Some of the code for processing the database was based on a script written by Sean Davis.
- I performed the analysis of the database presented in the publication.

- I wrote the first draft of the manuscript and produced all the figures in the publication.
- Alicia Oshlack provided advice on planning the manuscript and edited draft versions.
- Belinda Phipson contributed to writing the manuscript.

Chapter 3: Simulating scRNA-seq data is an original work describing a software package for simulating single-cell RNA sequencing expression data. This work was published in *Genome Biology* as “Splatter: simulation of single-cell RNA sequencing data”. The software package described in this publication is available through Bioconductor at <https://bioconductor.org/packages/splatter> and the code is shared on GitHub at <https://github.com/Oshlack/splatter> under a GPL-3.0 license.

Contributions to the work in this chapter:

- I designed and implemented the Splatter R package described in this chapter.
- Belinda Phipson contributed to the design of the Splat simulation method described in the publication and provided statistical advice.
- I conducted the analysis presented in the publication and produced the figures shown.
- Belinda Phipson performed pre-processing for some of the public datasets used.
- Alicia Oshlack helped to design and plan the analysis presented in the publication.
- I wrote the first draft of the manuscript and performed revisions.
- Alicia Oshlack assisted with planning the manuscript and edited drafts.
- Belinda Phipson helped write sections of the manuscript and edited drafts.
- Jovana Maksimovic and Sarah Blood proofread a draft of the manuscript and provided comments.

Chapter 4: Visualising clustering across resolutions is an original work describing a visualisation for showing clustering results across multiple resolutions and helping to select a clustering resolution to use. This work has been published in *GigaScience* as “Clustering trees: a visualization for evaluating clusterings at multiple resolutions” and a software package implementing the algorithm described is available from CRAN at <https://cran.r-project.org/package=clustree>. The source code for this package can be found on GitHub at <https://github.com/lazappi/clustree> under a GPL-3.0 license.

Contributions to the work in this chapter:

- I designed the clustering trees algorithm described in this chapter.
- I designed and built the clustree R package that implements this algorithm.
- I performed the analysis presented in the publication and designed and produced the figures shown.
- Alicia Oshlack provided advice on the design and planning of the analysis to present.
- I planned and wrote the first draft of the manuscript.
- Alicia Oshlack provided advice on the structure of the manuscript and edited draft versions.
- I performed revisions and drafted responses to reviewers.
- Marek Cmero read and provided comments on a draft of the manuscript.

Chapter 5: Analysis of kidney organoid scRNA-seq data is an original work where I performed a re-analysis of a previously published single-cell RNA sequencing experiment

from kidney organoids in order to demonstrate a range of analysis tools and decisions during analysis.

Contributions to the work in this chapter:

- The dataset is publicly available from the Gene Expression Omnibus under accession GSE114802.
- I performed pre-processing of the dataset
- I designed and performed the analysis with input from Alicia Oshlack, Belinda Phipson, Melissa Little and Alex Combes.
- Alex Combes helped with interpreting gene lists describing cell types.
- I designed and created the figures shown in this chapter.

Chapter 6: Conclusion is an original work summarising the work in this thesis, placing it in the wider context of single-cell RNA sequencing analysis and outlining potential directions of the field.

Other publications that I have contributed to during my candidature but are not presented in this thesis

Phipson B, **Zappia L**, Oshlack A. "Gene length and detection bias in single cell RNA sequencing protocols." *F1000 Research*. DOI: 10.12688/f1000research.11290.1.

Phipson B, Er PX, Combes AN, Forbes TA, Howden SE, **Zappia L**, Yen HJ, Lawlor KT, Hale LJ, Sun J, Wolvetang E, Takasato M, Oshlack A, Little MH. "Evaluation of variability in human kidney organoids." *Nature Methods*. 2019. DOI: 10.1038/s41592-018-0253-2.

Combes AN⁺, **Zappia L**⁺, Er PX, Oshlack A, Little MH. "Single-cell analysis reveals congruence between kidney organoids and human fetal kidney." *Genome Medicine*. 2019. DOI: 10.1186/s13073-019-0615-0

Kumar SV, Er PX, Lawlor KT, Motazedian A, Scurr M, Ghobrial I, Combes AN, **Zappia L**, Oshlack A, Stanley EG, Little MH. "Kidney micro-organoids in suspension culture as a scalable source of human pluripotent stem cell-derived kidney cells." *Development*. 2019. DOI: 10.1242/dev.172361

Combes AN, Phipson B, Lawlor KT, Dorison A, Patrick R, **Zappia L**, Harvey RP, Oshlack A, Little MH. "Single cell analysis of the developing mouse kidney provides deeper insight into marker gene expression and ligand-receptor crosstalk." *Development*. 2019. DOI: 10.1242/dev.178673

⁺ Authors contributed equally.

Acknowledgements

Thank you to my supervisors Alicia Oshlack and Melissa Little, it has been a pleasure and a privilege to work with you over the last few years. You both provide supportive environments that let people get the most out of themselves and I am fortunate to have had you to guide me through my PhD.

That support extends to the other members of the Oshlack lab. It's easy to come into work every day when you get on with your colleagues and enjoy spending time with them. A special thanks to Belinda Phipson who as being an amazing source of advice about statistics and analysis but also just life in general. The Little kidney development lab has also been extremely welcoming. Thank you to all of them for making me feel a part of their group and sharing their knowledge, particularly Alex Combes who it has been fantastic to work closely with.

I would also like to thank the member of my advisory committee Andrew Pask, Christine Wells and Edmund Crampin for their advice and encouragement.

Outside my work a big part of my PhD has been my involvement with COMBINE. Thank you to everyone who has contributed to building and continuing to develop the organisation, particularly Leah Roberts who has guided me through everything. The Australian bioinformatics student community is extremely fortunate to have such an organisation and I hope students and supervisors continue to support COMBINE and make it even better.

Thank you to my friends and family for the support you have provided outside my PhD, it's important to have other things going on in the rest of my life. The biggest thanks to my partner Sarah for her continued love and care (and proofreading expertise).

Lastly I want to thank the wider scientific and programming communities. Everyone who makes their code, software, data and analysis available makes this kind of research possible, easier and more fun.

Table of Contents

1	Introduction	1
1.1	The central dogma	3
1.2	RNA sequencing	5
1.2.1	Library preparation	5
1.2.2	High-throughput sequencing	6
1.2.3	Analysis of RNA-seq data	8
1.3	Single-cell RNA sequencing	9
1.3.1	Early single-cell capture technologies	10
1.3.2	Droplet-based cell capture	11
1.3.3	Unique Molecular Identifiers	11
1.3.4	Recent advances in scRNA-seq protocols	13
1.4	Analysing scRNA-seq data	15
1.4.1	Pre-processing and quality control	16
1.4.2	Normalisation and integration	18
1.4.3	Grouping cells	19
1.4.4	Ordering cells	20
1.4.5	Gene detection and interpretation	21
1.4.6	Alternative analyses	22
1.4.7	Evaluation of scRNA-seq analysis methods	22
1.5	Kidney development	22
1.5.1	Structure and function	23
1.5.2	Stages of development	23
1.5.3	Growing kidney organoids	25
1.5.4	Kidney scRNA-seq studies	27
1.6	Thesis overview and aims	27
2	The scRNA-seq tools landscape	29
2.1	Introduction	31
2.2	scRNA-tools publication	32
2.3	The current scRNA-tools database	46
2.4	Usage of the scRNA-tools website	48
3	Simulating scRNA-seq data	49
3.1	Introduction	51
3.2	Splatter publication	52
3.3	Updates to Splatter	67
3.3.1	Performance of current simulations	68
4	Visualising clustering across resolutions	71
4.1	Introduction	73
4.2	Clustering trees publication	74

4.3	Overlaying clustering trees	83
5	Analysis of kidney organoid scRNA-seq data	85
5.1	Introduction	87
5.2	Summary of published work	88
5.3	Outline and motivation	89
5.4	Pre-processing	92
5.4.1	Droplet selection	92
5.4.2	Alevin comparison	94
5.5	Quality control	96
5.6	Clustering	99
5.6.1	Gene selection	99
5.6.2	Resolution selection	100
5.6.3	Cluster validation	101
5.6.4	Comparison to published clusters	103
5.7	Marker gene detection	104
5.8	Connecting clusters	107
5.8.1	Partition-based graph abstraction	107
5.8.2	Cell velocity	109
5.9	Discussion	110
6	Conclusion	113
References		119
A	Splatter additional files	135
A.1	Additional figures	136
A.2	Session information	153
B	Splatter documentation	159
B.1	Splatter vignette	161
B.2	Splat parameters vignette	185
B.3	Splatter manual	205
C	Simulation comparison	263
C.1	Timings	264
C.2	Package versions	264
D	clustree documentation	265
D.1	clustree vignette	266
D.2	clustree manual	289
E	Kidney organoid publication	311
F	Analysis methods	327
F.1	Pre-processing	328
F.1.1	Droplet selection	328
F.1.2	Alevin	328
F.2	Quality control	328
F.3	Clustering	329
F.3.1	Gene selection	329
F.3.2	Graph-based clustering	329
F.4	Marker genes	329
F.5	Partition-based graph abstraction (PAGA)	329
F.6	Cell velocity	329

F.7	Other packages	330
G	Session information	331
G.1	Important packages	332
G.2	Full session information	332

List of Figures

1.1	The central dogma of molecular biology.	4
1.2	Illustration of adapter arrangements for the Illumina platform.	6
1.3	The Illumina Sequencing by Synthesis process.	7
1.4	A typical RNA-seq differential expression testing workflow.	8
1.5	Exponential increase in the scale of scRNA-seq experiments.	10
1.6	Diagram of the 10x Genomics Chromium cell capture process.	12
1.7	Unique Molecular Identifiers (UMIs) can improve scRNA-seq quantification. .	12
1.8	The range of multimodal scRNA-seq technologies.	14
1.9	Extensions to droplet-based capture protocols using nucleotide tagged antibodies. .	15
1.10	Phases of a standard scRNA-seq analysis workflow.	17
1.11	Trajectory inference framework as described by Cannoodt, Saelens and Saeys. .	20
1.12	Structure of the kidney and nephron.	24
1.13	Diagram of the stages of nephron maturation.	25
1.14	Diagram of the Takasato et al. kidney organoid protocol.	26
2.1	Current state of the scRNA-tools database.	47
2.2	Usage of the scRNA-tools website.	48
3.1	Performance of current Splatter simulations.	68
4.1	Clustering tree of the iris dataset overlaid on a PCA plot.	83
5.1	Outline of analysis of the kidney organoid dataset.	91
5.2	Droplet selection using EmptyDrops.	93
5.3	UpSet plot comparing droplet selection methods.	93
5.4	Comparison to quantification using alevin.	95
5.5	Thresholds used for quality control.	97
5.6	Validation of quality control.	98
5.7	Selecting genes for clustering.	100
5.8	Selecting a clustering resolution.	101
5.9	Validation of identified clusters.	102
5.10	Comparison to previously published clusters.	104
5.11	Results of edgeR differential expression analysis.	106
5.12	Results of partition-based graph abstraction (PAGA).	108
5.13	Cell velocity field calculated by velocityo.	110

List of Copyright Material

The copyright of the publication “Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database” included in **Chapter 2** is held by the authors including myself under a Creative Commons Attribution License (CC BY 4.0) (<https://creativecommons.org/licenses/by/4.0/>). This license permits unrestricted use, distribution and reproduction in any medium, provided the original authors and source are credited.

The copyright of the publication “Splatter: simulation of single-cell RNA sequencing data” included in **Chapter 3** is held by the authors including myself under a Creative Commons Attribution License (CC BY 4.0). The additional files for this publication provided in **Appendix A** are covered by the same license.

The copyright of the Splatter R software package and the documentation included in **Appendix B** is held by the authors including myself under a GNU General Public License v3.0 (GPL-3) license (<https://choosealicense.com/licenses/gpl-3.0/>). This license allows me to reproduce the documentation in both print and electronics versions of this thesis.

The copyright of the publication “Clustering trees: a visualization for evaluating clusterings at multiple resolutions” included in **Chapter 4** is held by the authors including myself under a Creative Commons Attribution License (CC BY 4.0).

The copyright of the clustree R software package and the documentation included in **Appendix D** is held by the authors including myself under a GNU General Public License v3.0 (GPL-3) license.

The copyright of the publication “Single-cell analysis reveals congruence between kidney organoids and human fetal kidney” included in **Appendix E** is held by the authors including myself under a Creative Commons Attribution License (CC BY 4.0).

1

Introduction

See, people they don't understand
No, girlfriends, they can't understand
Your grandsons, they won't understand
On top of this, I ain't ever gonna understand...

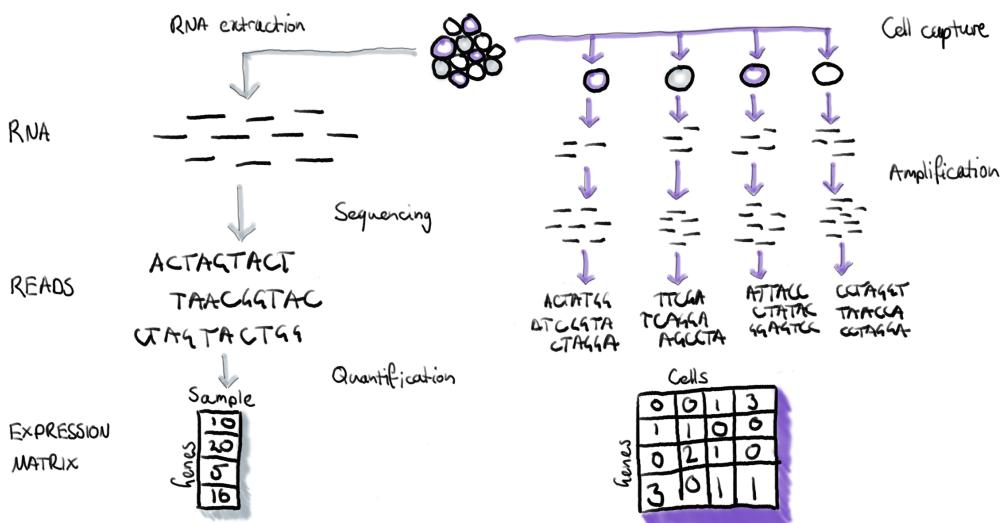
— The Strokes

Last Nite, 2001

1. INTRODUCTION

The central dogma

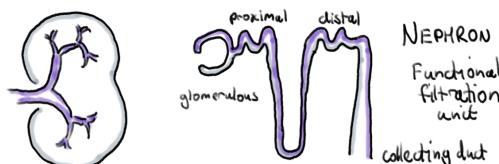
DNA $\xrightarrow{\text{transcription}}$ RNA $\xrightarrow{\text{translation}}$ PROTEIN



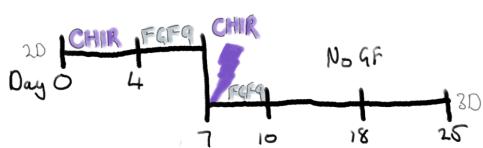
PHASES OF ANALYSIS

1. Data acquisition Alignment
Quantification
2. Data cleaning Quality control
Normalisation
3. Cell assignment Clustering
Classification
Ordering
4. Gene identification Differential expression
Expression patterns

KIDNEY DEVELOPMENT



ORGANOID PROTOCOL



This introduction chapter provides the background and overview necessary for understanding my work in this thesis, including an introduction to basic cell biology and kidney development, the technologies we use to measure those processes in the cell and the computational methods we use to analyse the data these technologies produce. **Section 1.1 - The central dogma** describes how information flows within a cell and the molecules involved in these processes. **Section 1.2 - RNA sequencing** covers the technologies used to measure the dynamic changes in some of these molecules and the established computational methods that are commonly used to extract meaning from this kind of data. In **Section 1.3 - Single-cell RNA sequencing** I discuss new technologies that have enabled measurements on the level of individual cells while **Section 1.4 - Analysing scRNA-seq data** outlines the types of analysis that can be performed on this data and the tools and methods that perform them. **Section 1.5 - Kidney development** provides a brief introduction to kidney structure and development and how it can be studied using stem cell technologies. This section provides background for **Chapter 5** where I describe an analysis of a dataset from developing kidney organoids. My introduction ends with **Section 1.6 - Thesis overview and aims** which presents an overview of my thesis aims and how I have completed them in the following chapters. An online version of this thesis is available at <https://lazappi.github.io/phd-thesis/>.

1.1 The central dogma

The central dogma of molecular biology describes the flow of information within a cell, from DNA to RNA to protein (Figure 1.1). Deoxyribonucleic acid (DNA) is the long-term data storage of the cell and has a well-known double helix structure [1–3]. Each strand of the double helix consists of a series of nucleotide molecules linked by phosphate groups. These nucleotides are made of three subunits: a nitrogenous base, a deoxyribose sugar and a phosphate group. DNA nucleotides come in four species, adenine, cytosine, thymine and guanine, which have different base subunits. The sugar and phosphate groups form the backbone of each DNA strand and the two strands are bound together through hydrogen bonds between matching nucleotides known as base pairs. Guanine forms three hydrogen bonds with cytosine and adenine forms two with thymine. In computing terms DNA is similar to a hard drive in that it provides stable, consistent storage of important information.

When the cell wants to use some of this information it produces a copy of it in the form of a ribonucleic acid (RNA) molecule through a process known as transcription, similar to a computer loading information it wants to use into its random access memory. Sections of DNA that encode functional information are called genes. RNA is similar to a single strand of DNA except that the deoxyribose sugar is replaced with ribose and the thymine nucleotide is replaced with another nucleotide called uracil. Because it is single-stranded, RNA does not have a double helix structure but it can form complex shapes by binding to itself. A cell is able to create many copies of the same RNA molecule and this is referred to as the expression level. By varying RNA expression the activity of processes within a cell can be regulated. There are several different types of RNA that serve different purposes. RNA molecules that are transcribed from protein-coding genes are known as messenger RNA (mRNA). Other types of RNA include ribosomal RNA (rRNA) which forms part of the ribosome (the molecular machinery that manufactures proteins), transfer RNA (tRNA) which carries amino acids to the ribosome, micro RNA (miRNA) which have a role in regulating gene expression and long non-coding RNA (lncRNA) which are also involved in regulation among other processes. Genes are made up of regions that encode information (known as exons) that alternate with much larger non-coding regions (introns).

When an mRNA molecule is transcribed it initially contains the intronic sequences but these are removed through a process known as RNA splicing and a sequence of adenine nucleotides (a poly(A) tail) is added where transcription ends (the 3' end) to mark a mature mRNA molecule.

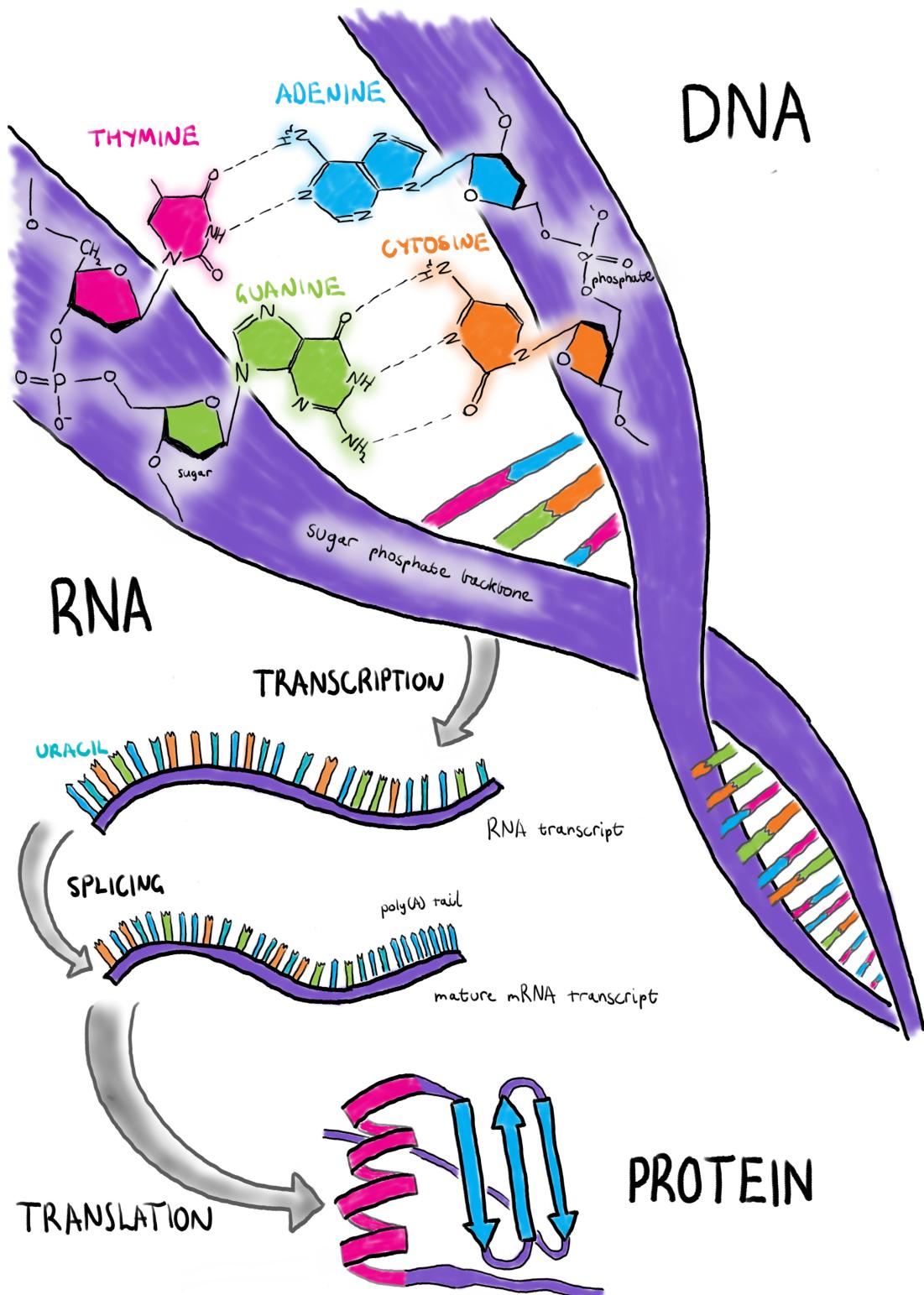


Figure 1.1: The central dogma of molecular biology. Deoxyribonucleic acid (DNA) is the long-term data storage of the cell. To use this information a ribonucleic acid (RNA) molecule is produced through a process called transcription. A mature messenger RNA (mRNA) requires splicing and addition of a ploy(A) tail. Proteins are created from mRNA through translation in the ribosome.

This splicing process allows multiple forms of a protein to be produced from a single gene by selecting which exons are retained or removed. A mature mRNA transcript is converted to a protein in a complex structure called the ribosome which is made up of specialised RNA and proteins. The conversion process is known as translation because the information encoded by nucleic acids in RNA is translated to information stored as amino acids in the protein. Proteins complete most of the work required to keep a cell functioning and can be compared to the programs running on a computer. These functions require complex three dimensional structures and include tasks such as sensing things in the external environment, transporting nutrients into the cell, regulating the expression of genes, constructing new proteins, recycling molecules and facilitating metabolism. Understanding the molecules involved in the central dogma is central to our understanding of how a cell works.

1.2 RNA sequencing

By looking at DNA we can see what versions of genes are present in a cell but we cannot tell which of them are active and what processes they might be involved in. To do that we need to inspect the parts of the system that change dynamically. Ideally we might want to interrogate the proteins that are present, as they provide most of the functionality. However, while it is possible to do this using technologies such as mass spectrometry the readout produced is more difficult to interpret and the encoding is much more complex as there are 20 different amino acids compared to only four nucleotides. In contrast, RNA molecules are much easier to measure. High-throughput RNA sequencing (RNA-seq) provides a reliable method for high-quality measurement of RNA expression levels. RNA is isolated from a biological sample, converted to complementary DNA (cDNA) and provided as input to a sequencing machine. The output of an RNA-seq experiment is millions of short nucleotide sequences originating from the RNA transcripts present in the sample. Compared to older techniques for measuring RNA, such as probe-based microarrays, RNA-seq requires no prior knowledge of existing sequences in order to measure a sample and is effective over a much greater range of expression levels.

1.2.1 Library preparation

The first step in preparing a sample for RNA-seq is to chemically lyse the cells, disrupting the structure of the cell wall and releasing the molecules inside. RNA molecules can then be isolated, typically using a chemical process called phenol/chloroform extraction, although this can also be done by physically separating different types of molecules by passing the sample through a silica column. The majority of the RNA in a cell is ribosomal RNA, usually more than 80 percent [4]. Most of the time this type of RNA is not of interest and because the total amount of sequencing in an experiment is limited sequencing rRNA would reduce the ability to detect less abundant species. To select mature mRNA transcripts, oligonucleotide probes that bind to the poly(A) tail can be used, but a downside of this approach is that it won't capture immature mRNA or other types of RNA molecules. An alternative method is ribosomal RNA depletion where a different kind of probe specific to each species is used that binds to the rRNA allowing it to be removed. The choice of selection method has been shown to introduce different biases into the resulting data [5].

The Illumina sequencing typically used for RNA-seq experiments can only read short sequences of nucleotides of approximately 40–400 base pairs. Most mRNA molecules are longer than this so to read the full length of a transcript it must first be fragmented into smaller parts. Most sequencing machines also only work with DNA, not directly with RNA, so the sample must first be reverse-transcribed using a retroviral enzyme to produce a single strand of cDNA. Many protocols have been designed for this step with each requiring a specific primer sequence to be joined to the RNA molecules. The complementary strand of cDNA is produced using a

second enzyme that is usually involved in copying DNA for cell division. For some protocols fragmentation is performed after conversion to cDNA rather than at the RNA stage.

Once the cDNA has been produced it is usually necessary to attach adaptor sequences that are used to bind the molecules and initiate sequencing (Figure 1.2). These adaptors may also contain multiplexing barcodes that tag all the molecules in a sample and allow multiple samples to be sequenced at once. It has become standard practice to perform paired-end sequencing, where a section of nucleotides is read from one end of a fragment before it is flipped and the other end read, and this process requires an additional set of adaptors. At each of the stages of library preparation there are quality control steps to be performed to make sure a high-quality cDNA sample is loaded on to the sequencing machine.

1.2.2 High-throughput sequencing

Most RNA-seq experiments are sequenced on an Illumina machine using their Sequence by Synthesis technology (Figure 1.3). In this process the two strands of cDNA fragments are separated and the adaptors bind to oligonucleotides probes coating a flow cell. The other end of the fragment can bind to a second oligonucleotide forming a bridge structure where an enzyme synthesises the complementary DNA strand. This process of separation of strands and synthesis of new complementary strands is repeated until clusters of DNA fragments with the same sequence are formed. Once the clusters contain enough fragments the adaptor at one end of each fragment is cleaved leaving single-stranded DNA attached to the flow cell at one end.

The sequencing process now begins. Nucleotides tagged with fluorescent markers are added and can bind to the next available position on a fragment if they are complementary. By

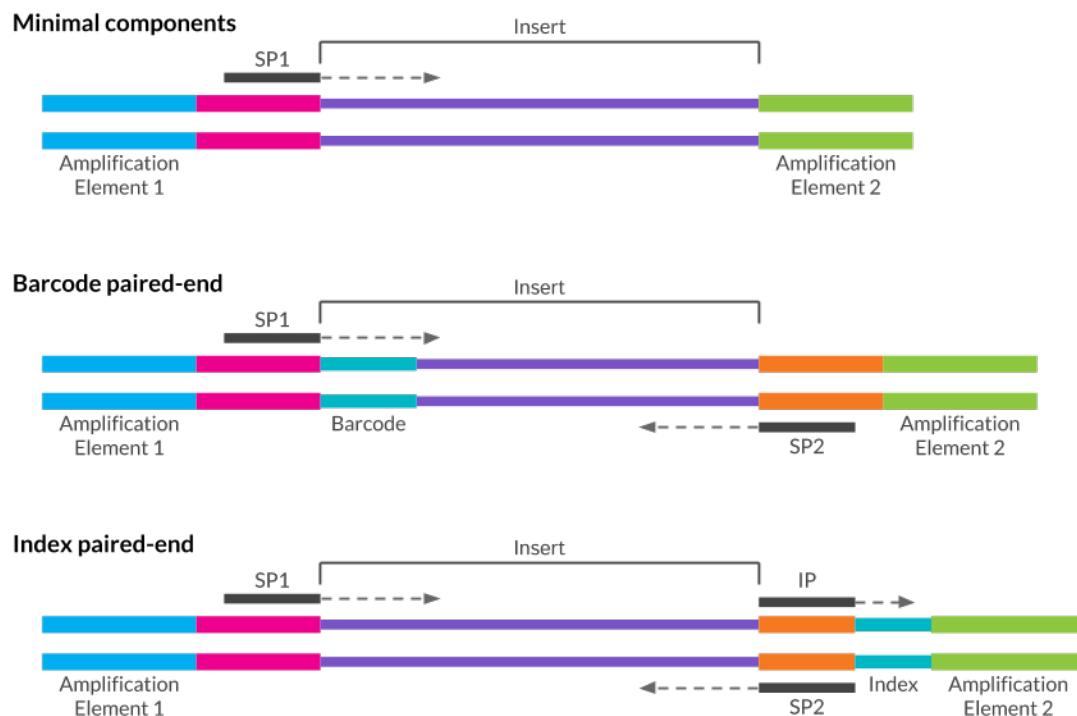


Figure 1.2: Illustration of adapter arrangements for the Illumina platform ((A) minimal adapter components, (B) “in-line” barcode configuration for paired-end sequencing, (C) index configuration for paired-end sequencing). Sequencing primers SP1 (primary) and SP2 (paired-end) allow initiation of synthesis of the insert sequence. Index or barcode sequences allow multiplexing of multiple samples with the index being sequenced from a separate index primer (IP). (Image adapted from RNA-seqlopedia, <http://rnaseq.uoregon.edu> [6]).

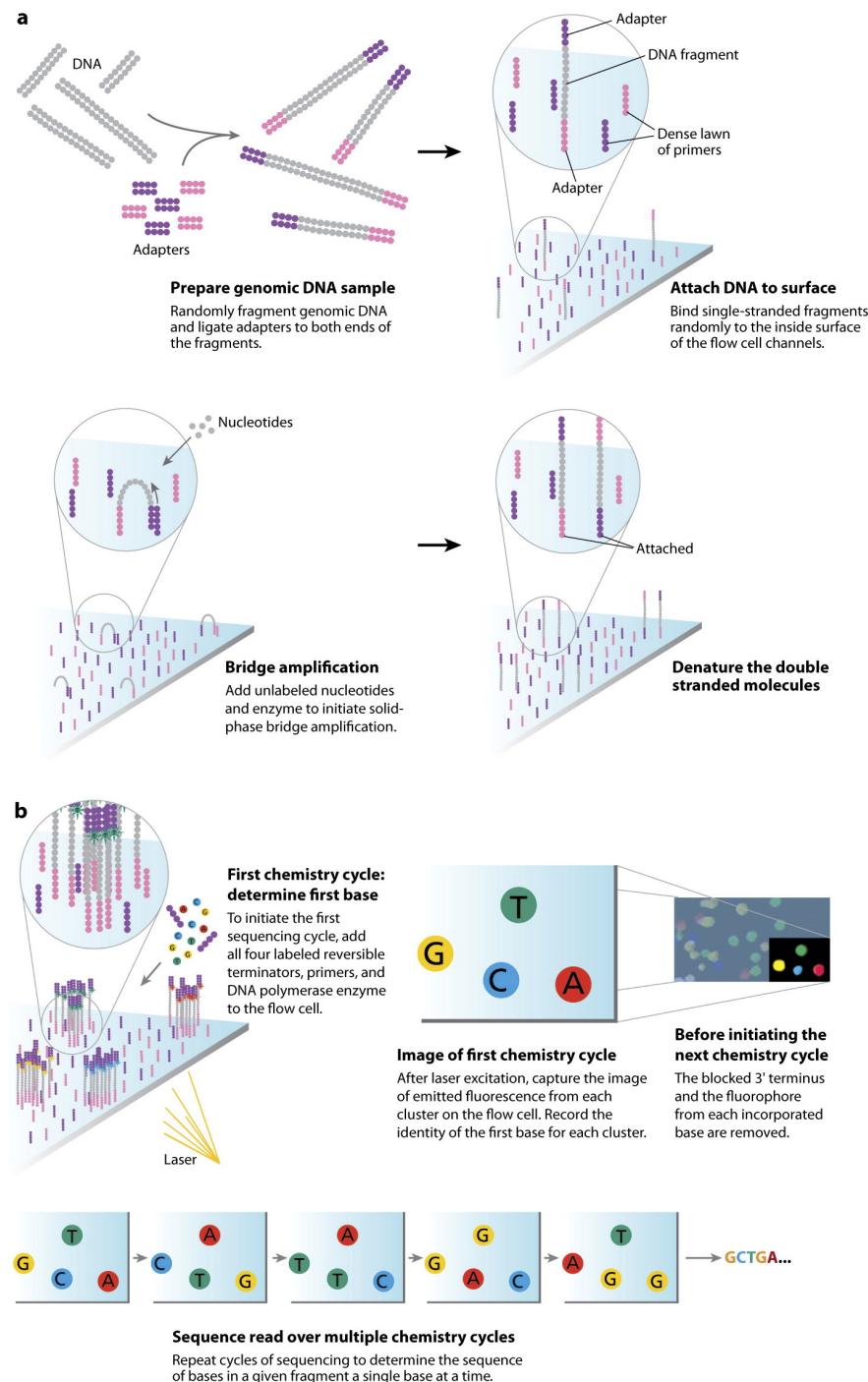


Figure 1.3: The Illumina Sequencing by Synthesis process. Nucleotide fragments bind to a flow cell and clusters are formed through bridge amplification. Strands are primed and nucleotides tagged with fluorescent labels are added to the flow cell along with a DNA polymerase enzyme. The lanes of the flow cell are then scanned to produce an image. The fluorescent tags are cleaved and 3'-OH blocking groups are added, preparing the strands for another round of nucleotide incorporation. Image adapted from Mardis, “Next-Generation DNA Sequencing Methods” [7].

adding all four nucleotides at once they compete for each position, reducing the chance of an incorrect match. Any unbound nucleotides are washed away before a laser excites the fluorescent tags and an image is taken. Each nucleotide is tagged with a different colour and the order of colours produced by a cluster shows the sequence of nucleotides in a fragment. For paired-end sequencing the fragments can be flipped and the sequencing process repeated at the other end. The images from the sequencing machine are processed to produce millions of short nucleotide sequences known as reads that are the starting point for computational analysis.

1.2.3 Analysis of RNA-seq data

Many types of analyses can be performed using RNA-seq data, such as identification of variants in the genetic sequence or detection of previously unannotated transcripts, but the most common kind of analysis is to look for differences in the expression level of genes between groups of samples (Figure 1.4). To do this, reads are first aligned to a reference genome and the number of reads overlapping each gene is counted. In contrast to aligners designed for DNA sequencing, RNA-seq aligners such as STAR [8], HISAT2 [9] and subread [10] must take into account the splicing structure of mRNA transcripts which causes parts of some reads to align in different locations in the genome. The alignment step is computationally intensive and can take a significant amount of time. More recently tools such as kallisto [11] and Salmon [12] have been developed which attempt to directly quantify expression by estimating the probability that a read comes from a particular annotated transcript. These approaches are orders of magnitude faster than true genome alignment and potentially produce more accurate quantification, at the cost of having an exact genomic position for each read.

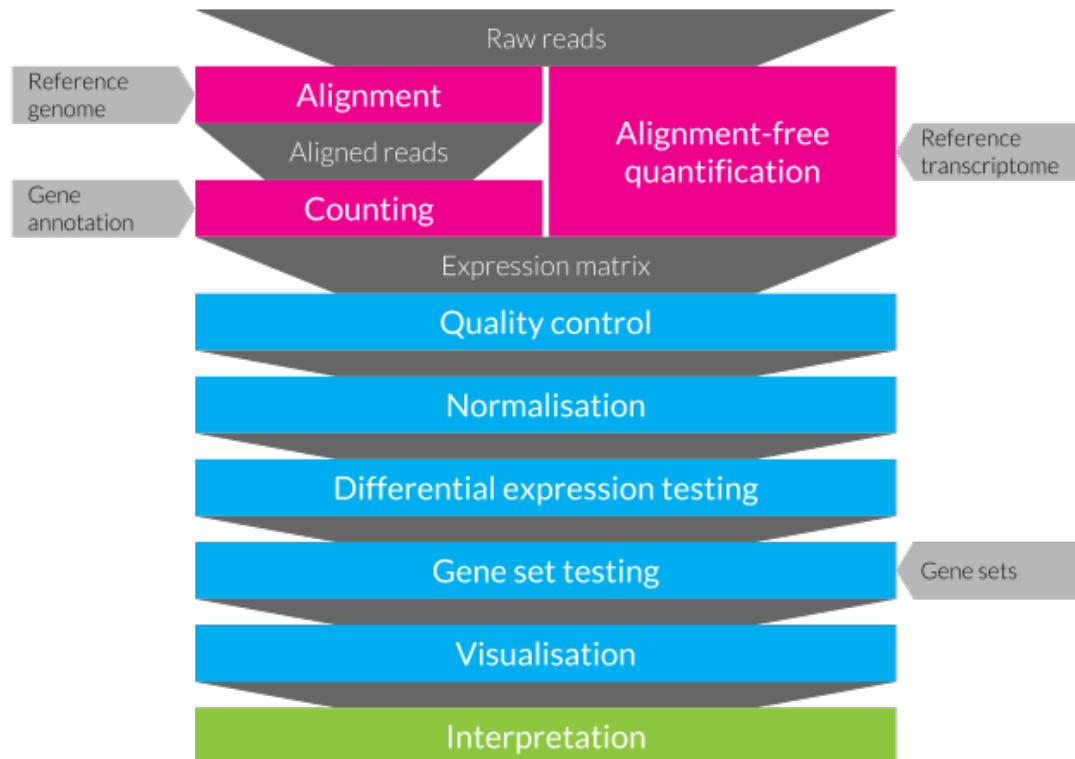


Figure 1.4: A typical RNA-seq differential expression testing workflow. An expression matrix is created from raw reads either by aligning them to a reference genome and counting those that overlap annotated genes or by alignment-free quantification (pink). Data analysis (blue) consists of several steps including quality control of samples and features, normalisation to remove technical differences between samples, testing for differential expression, gene set testing to identify enriched signatures, and visualisation. Results must be carefully interpreted (green) to extract meaning.

At this stage the result is a matrix of counts, known as an expression matrix, where the rows are features (usually genes), the columns are samples and the values show the expression level of a particular feature in a sample. As these counts result from a sampling process they can be modelled using common statistical distributions. One option is the Poisson distribution which describes the probability of a given number of events happening in a fixed amount of time, however this distribution assumes that the mean and variance of each feature is equal. This can be true for replicates of the same RNA-seq sample but does not represent the extra biological variability between different samples [13]. A better fit is the negative binomial (or Gamma-Poisson) distribution which includes an over-dispersion parameter, allowing the variance in expression to be larger than the mean. While each feature is quantified for each sample, the observed expression levels are related to the total number of reads and the expression level of other features. For example, if a gene is highly expressed in a particular sample it will soak up a large proportion of the available reads, while in another sample where it is less expressed those reads are more likely to be sampled from other genes. Another complication of RNA-seq data is that the number of features (tens of thousands) is much larger than the number of samples (usually only a few per group) which violates the assumptions of many traditional statistical techniques. Technical variation is another well-known problem in high-throughput genomics studies, for example it has been estimated that only 17.8 percent of allele-specific expression is due to biological variation with the rest being technical noise [14].

Both the edgeR [15,16] and DESeq [17] (and later the DESeq2 [18]) packages model RNA-seq data using the negative binomial distribution, but before differences in expression levels are tested the technical differences between samples should be removed through normalisation. Early approaches include simple transformations like Reads (or Fragments) Per Kilobase per Million (RPKM/FPKM) [19] or Transcripts Per Million (TPM) [20] which correct for the total number of reads per cell and gene length. The edgeR package uses the Trimmed-Mean of M values (TMM) method where a scaling factor for each sample is produced using the weighted mean of log ratios between samples for each gene, after excluding the highest expressed genes and those with the largest log ratio [13]. DESeq has a similar method that uses the median of the ratio of the expression of each gene to the geometric mean expression across all samples. The limma package [21] uses an alternative approach where a method called voom transforms the data so that it is suitable for linear modelling methods originally designed for RNA microarray technology [22]. When an experiment has been conducted in multiple batches and there are significant differences between them, alternative normalisation methods such as Remove Unwanted Variation (RUV) may be required [23]. RUV estimates unknown factors of unwanted variation using negative control genes (genes that are expected to be consistent between samples). These factors can then be included in the generalised linear modelling frameworks used for testing differential expression. Over time the methods in these packages have been refined and new tests developed allowing for the routine analysis of many RNA-seq experiments.

1.3 Single-cell RNA sequencing

Traditional bulk RNA-seq experiments average the transcriptome across the millions of cells in a sample. Recently it has become possible to perform single-cell RNA sequencing (scRNA-seq) and investigate the transcriptome at the resolution of individual cells. There are many situations where it is important to understand how specific cell types react to development or perturbations. This is often hindered in bulk analyses which may be affected by the unknown proportions of cell types in a sample. Studies into gene expression in specific cell types previously required selecting and isolating the cells of interest, which separates them from the other cell types they are usually associated with and makes it impossible to investigate interactions between them. With scRNA-seq technologies it is now possible to look at the transcriptome of all the cell types in a tissue simultaneously, which has led to a better

understanding of what makes cell types distinct and the discovery of previously unknown cell types.

1.3.1 Early single-cell capture technologies

The first scRNA-seq protocol was published in 2009 [24], just a year after the first bulk RNA-seq publication [19]. While this approach allowed measurements of the transcriptome in individual cells it required manual manipulation and was restricted to inspecting a few precious cells. Further studies quickly showed that cell types could be identified without sorting cells and approaches were developed to allow unbiased capture of the whole transcriptome. Since then, many scRNA-seq protocols have been developed (including CEL-Seq [25], CEL-Seq2 [26], Quartz-Seq [27], Quartz-Seq2 [28] and Smart-seq2 [29]) and the number of cells in scRNA-seq experiments has scaled exponentially (Figure 1.5) [30]. The first commercially available cell capture platform was the Fluidigm C1. This system uses microfluidics to passively separate cells into individual wells on a plate where they are lysed, reverse transcribed and the collected cDNA is PCR amplified. After this stage the product is extracted from the plate and libraries prepared for Illumina sequencing. Most C1 data has been produced using a 96 well plate but more recently an 800 well plate has become available, greatly increasing the number of cells that can be captured at a time. One of the disadvantages of microfluidic plate-based cell capture technologies is that the chips used have a fixed size window, meaning that only cells of a particular sizes can be captured in a single run. However, as cells are captured in individual wells they can be imaged before lysis, potentially identifying damaged or broken cells, empty wells or wells containing more than one cell. Capturing multiple cells is a known issue, with Macosko et al. finding that when preparing a mixture of mouse and human cells, 30 percent of the resulting libraries contained transcripts from both species but only about a third of these doublets were visible in microscopy images [31]. The newer Polaris system from Fluidigm also uses microfluidics to capture cells but can select particular cells based on staining or fluorescent reporter expression and then hold them for up to 24 hours while introducing various stimuli. The cells can be imaged during this time before being lysed and prepared for RNA sequencing. This platform provides opportunities for a range of experiments that aren't possible using other capture technologies.

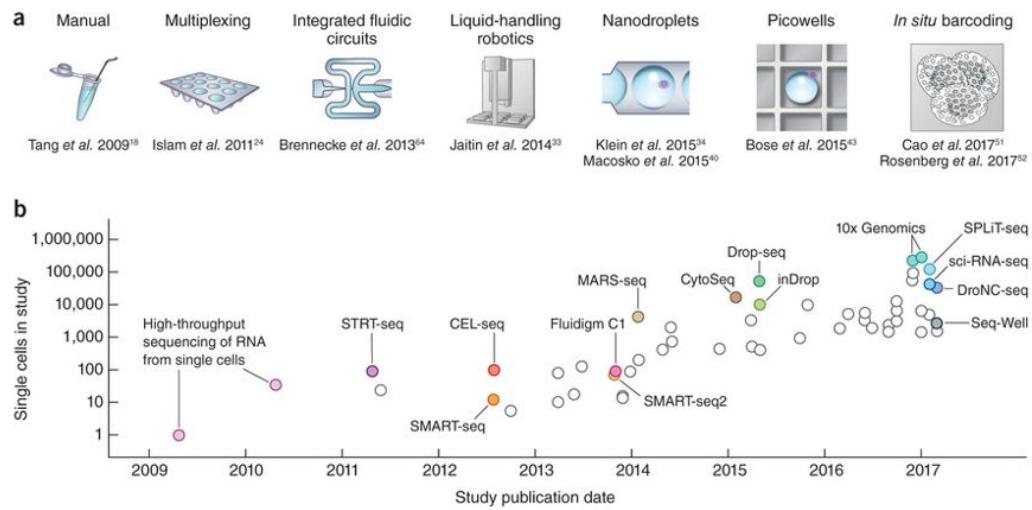


Figure 1.5: Exponential increase in the scale of scRNA-seq experiments. (a) Key jumps in scRNA-seq capture technologies including manual handling, plate-based microfluidic capture and droplet-based capture. (b) Cell numbers in representative studies using key technologies by publication date. Image from Svensson, Vento-Tormo and Teichmann “Exponential scaling of single-cell RNA-seq in the past decade” [30].

1.3.2 Droplet-based cell capture

An alternative to using microfluidics to capture cells in wells is to capture them in nanodroplets. In this process a dissociated cell mixture is fed into a microfluidic device, while beads coated in primers enter at another input. The device is designed to form aqueous droplets within mineral oil and the inputs are arranged so that cells and beads can be simultaneously captured within a droplet. When this happens the reagents carried along with the bead lyse the cell and any poly(A) tagged RNA molecules present can bind to the capture probes on the bead. Reverse transcription and PCR amplification then begins and an individual cDNA library is produced for each cell, tagged with the unique barcode sequence present on the bead. The main advantage of droplet-based capture technologies is the ability to capture many more cells at one time, up to tens of thousands. These approaches are also less selective about cell size and produce fewer doublets. As a result they are much cheaper per cell, although as sequencing costs are fixed studies using droplet-based captures typically sequence individual cells at a much lower depth.

Droplet-based capture was popularised by the publication of the Drop-seq [31] and InDrop [32] platforms in 2015 and the updated InDrops in 2017 [33]. These are both DIY systems and although they differ in how the beads are produced, when the droplets are broken and in some aspects of the chemistry they can both be constructed on a lab bench from syringes, automatic plungers, a microscope and a small custom-made microfluidic chip. A similar commercially available platform is the 10x Genomics Chromium device which automates and streamlines much of the process (Figure 1.6) [34]. This device uses droplet-based technologies for a range of applications including capture of cells for scRNA-seq. More specialised captures, such as those aimed at profiling immune cell receptors, are also possible and the company has recently announced kits for single-cell Assay for Transposase-Accessible Chromatin using sequencing (scATAC-seq) capture. A recent review has suggested that the 10x Chromium provides the most appropriate droplet-based capture for most experiments that aren't cost sensitive or require custom protocols [35].

1.3.3 Unique Molecular Identifiers

In contrast to plate-based capture methods, which usually provide reads along the length of RNA transcripts, droplet-based capture methods typically employ protocols which include short random nucleotide sequences known as Unique Molecular Identifiers (UMIs) [36]. Individual cells contain very small amounts of RNA (around 10–30 pg, less than 5 percent of which is mRNA) and to obtain enough cDNA for sequencing a PCR amplification step is necessary. Depending on their nucleotide sequence different transcripts may be amplified at different rates which can distort their relative proportions within a library. UMIs attempt to improve the quantification of gene expression by allowing the removal of PCR duplicates produced during amplification (Figure 1.7) [37]. The nucleotide probes used in droplet-based capture protocols include a poly(T) sequence which binds to mature mRNA molecules, a barcode sequence which is the same for every probe on a bead and 8–10 bases of UMI sequence which is unique to each probe. The UMI sequences are long enough that the probability of capturing two copies of a transcript on two probes with the same UMI is extremely low. After reverse transcription, amplification, sequencing and alignment, de-duplication can be performed by identifying reads with the same UMI that align to the same position and therefore should be PCR duplicates rather than truly expressed copies of a transcript.

For this method to be effective each read must be associated with a UMI which means that only a small section at the 3' end of each transcript is sequenced. This has the side effect of reducing the amount of cDNA that needs to be sequenced and therefore increasing the number of cells that can be sequenced at a time. While the improvement in quantification of gene expression levels is useful for many downstream analyses, it comes at the cost of coverage

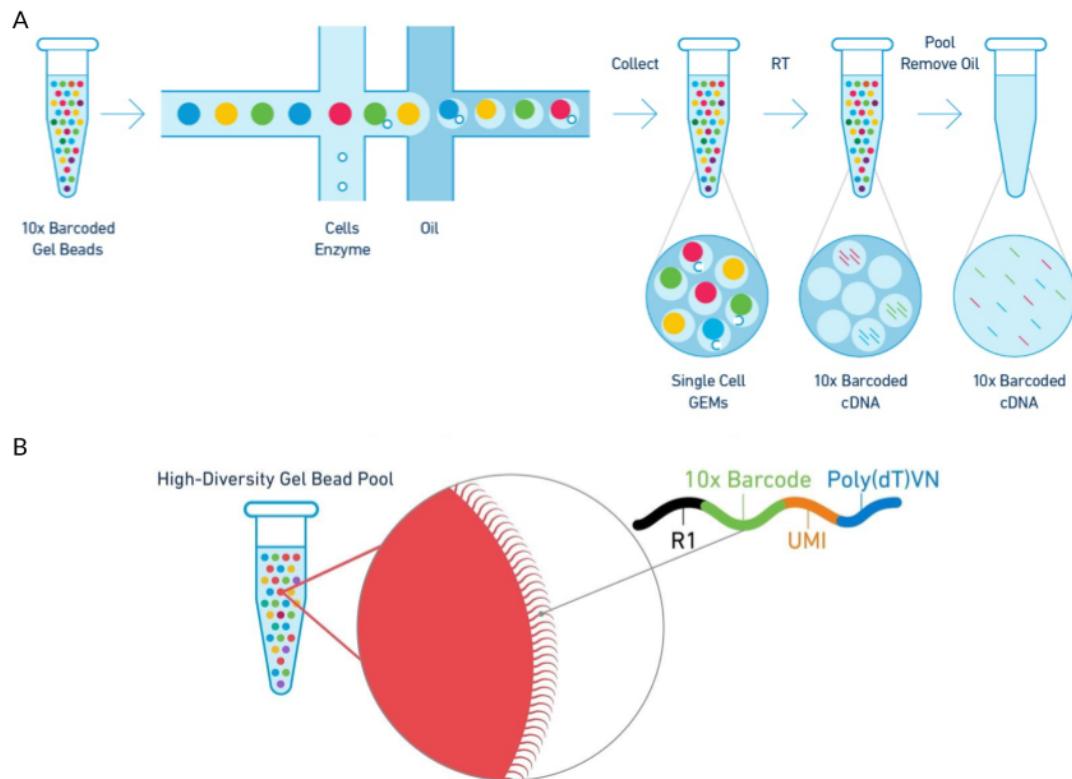


Figure 1.6: Diagram of the 10x Genomics Chromium cell capture process. (A) Steps in the cell capture process. Barcoded gel beads are passed into a microfluidic device along with dissociated cells where they are captured in aqueous droplets in an oil solution. Cells are lysed within the droplets and mRNA is reverse transcribed to produce barcoded cDNA. Droplets are then broken and the cDNA collected for sequencing. (B) Structure of the gel bead capture probe including adaptor, cell barcode, UMI and poly(T) tail. Adapted using images from 10x Genomics.

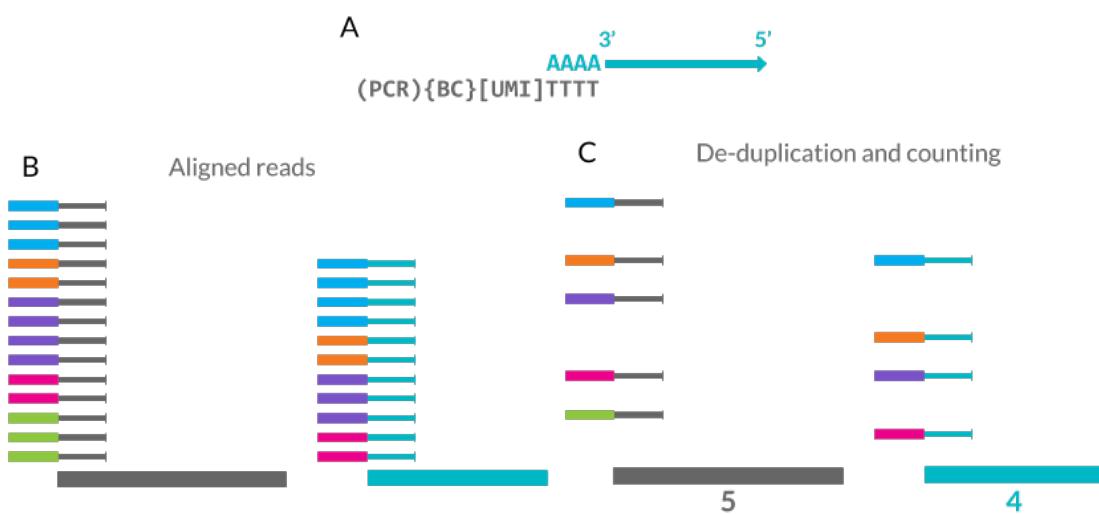


Figure 1.7: Unique Molecular Identifiers (UMIs) can improve scRNA-seq quantification. (A) UMIs are random 8–10 base pair sequences included as part of the mRNA capture probe along with the cell barcode (BC) and PCR handle. (B) The mRNA sequence can be aligned to a reference genome. (C) Each UMI is only counted once at each location, removing PCR duplicates and improving quantification.

across the length of a transcript, which is required for applications such as variant detection and de-novo assembly. However, reads along the length of genes have been observed in UMI datasets and are believed to come from unannotated transcription start sites or regions that contain enough adenine nucleotides to bind to the capture probes. Datasets with UMIs need extra processing steps which can be complicated by the possibility of sequencing errors in the UMI itself. Statistical methods designed for full-length data may also be affected by the different properties of a UMI dataset.

1.3.4 Recent advances in scRNA-seq protocols

Although droplet-based techniques are currently the most commonly used cell capture technologies, other approaches have been proposed that promise to capture even more cells at a lower cost per cell. These include approaches based around sub-nanolitre sized wells, for example the Seq-Well protocol [38]. A simple flow cell is constructed with an array of microwells made from polydimethylsiloxane (PDMS), a silicone rubber. A solution containing dissociated cells is flowed over the array and cells are captured in the wells by gravity. This is repeated to capture barcoded beads before adding reagents and sealing the wells. Once reactions are complete the beads can be retrieved and processed in a similar way to droplet-based capture [39]. Some cell types are difficult to capture due to their size, shape or other properties, and in some cases, particularly in tissues such as the brain, it has been suggested that single-nucleus rather than single-cell RNA-seq may be more effective [40–43]. In these protocols Nuclei are captured and processed in a similar way to cells, however the RNA within them is immature and unspliced so different reference annotations are required during analysis.

Extensions to the standard protocols have also been proposed that allow multiple measurements from the same cell (Figure 1.8). One such protocol is CITE-seq which enables measurement of the levels of selected proteins at the same time as the whole transcriptome (Figure 1.9A) [44]. Antibodies for the proteins of interest are labelled with short nucleotide sequences. These antibodies can then be applied to the dissociated cells and any that remain unbound are washed away before cell capture. The antibody labels are then captured along with mRNA transcripts and a size selection step is applied to separate them before library preparation. Similar antibodies can be used to allow multiplexing of samples through a process known as cell hashing (Figure 1.9B) [45]. In a typical scRNA-seq experiment each batch corresponds to a single sample. This complicates analysis as it is impossible to tell what is noise due to cells being processed in the same way and what is true biological signal. Cell hashing uses an antibody to a ubiquitously expressed protein but with a different nucleotide sequence for each sample. The samples can then be mixed, processed in batches and then the cells computationally separated based on which sequence they are associated with. An added benefit of this approach is the simple detection of doublets containing cells from different samples.

CRISPR-Cas9 gene editing has also been developed as an extension to scRNA-seq protocols. One possibility is to introduce a mutation at a known location that can then be used to demultiplex samples processed together [48]. It is possible to do this with samples from different individuals or cell lines but the advantage of a gene editing based approach is that the genetic background remains similar between samples. It is also possible to investigate the effects of introducing a mutation. Protocols like Perturb-Seq [49] introduce a range of guide RNA molecules to a cell culture, subject the cells to some stimulus then perform single-cell RNA sequencing. The introduced mutation can then be linked to the response of the cells to the stimulus and the associated broader changes in gene expression. More recently it has been proposed that CITE-seq can be combined with CRISPR-based approaches [50]. Another approach is CellTagging which introduces a barcoded lentiviral construct into cells [51]. A unique combination of barcodes is then passed on to daughter cells and can be used to trace the lineage of a cell population.

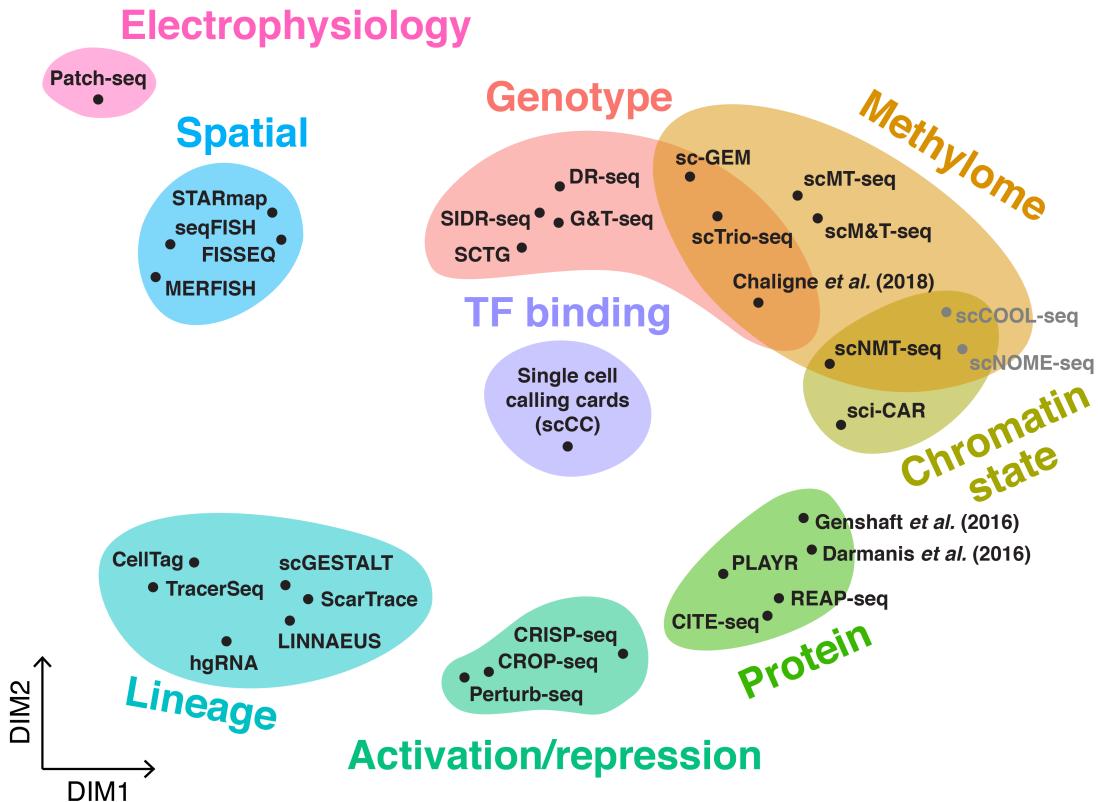


Figure 1.8: The range of multimodal scRNA-seq technologies. Many protocols have been developed to enable multiple measurements from the same individual cells with a selection shown here. These other measurements (shaded colours) include genotype (rose bud), methylation (cornsilk), chromatin state (muted lime), transcription factor binding (periwinkle), protein expression (jade lime), activation or repression (cabbage), lineage tracing (aquamarine blue), spatial location (light sky blue) or electrophysiology (cotton candy). Image from <https://github.com/arnavm/multimodal-scRNA-seq> available under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0) license [46].

Other approaches that allow multiple measurements from the same individual cells include G&T-seq [52] and SIDR [53] which measure RNA and genomic DNA, similar approaches which measure the exome (or protein-coding part of the genome) along with the transcriptome [54], scMT-seq which measures RNA and DNA methylation [55], Patch-seq which combines patch-clamp recording to measure the electrophysiology of neurons with scRNA-seq [56] and scTrio-seq which is able to measure the genome, transcriptome and methylome simultaneously [57].

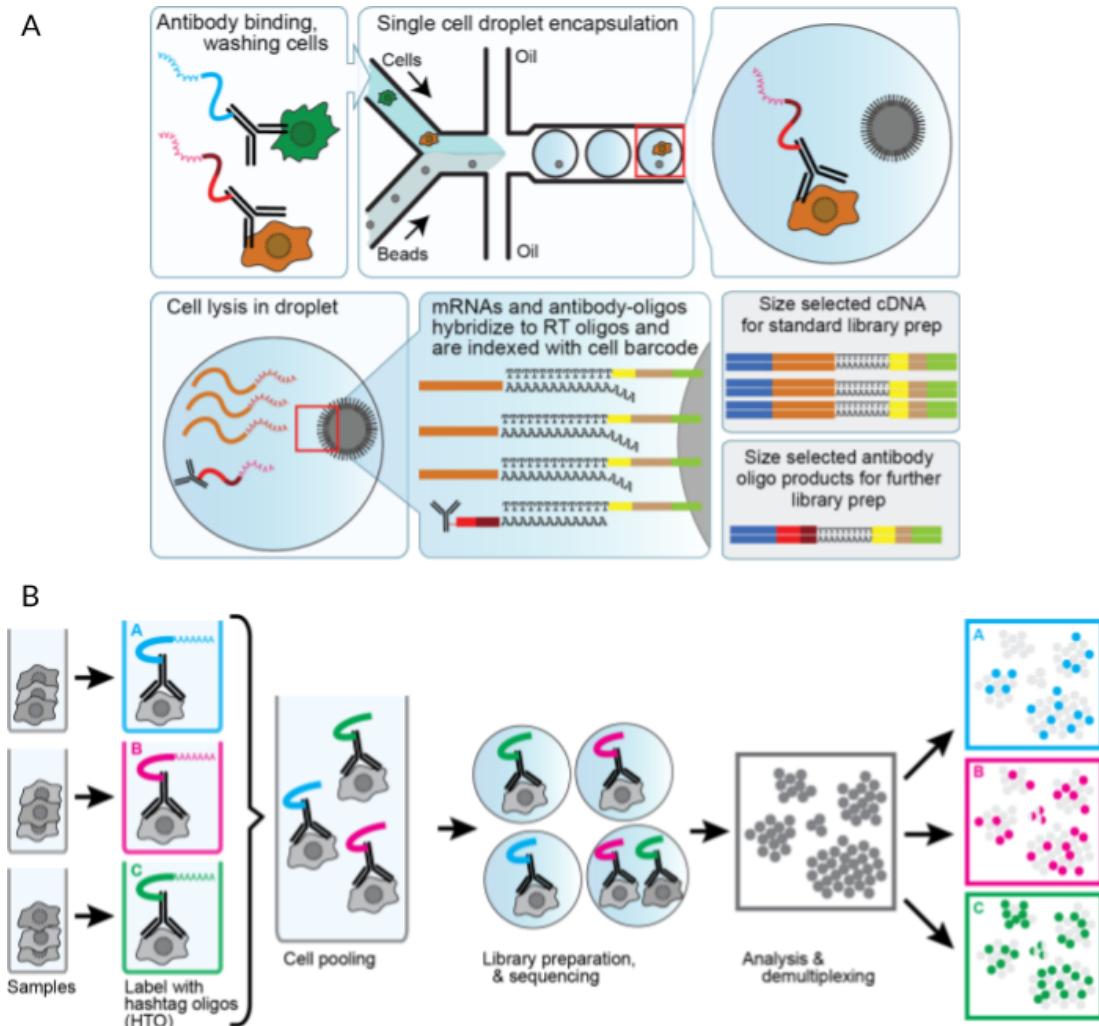


Figure 1.9: Extensions to droplet-based capture protocols using nucleotide tagged antibodies. (A) CITE-seq allows measurement of selected cell surface markers along with the transcriptome. Nucleotide tagged antibodies are applied to cells before droplet-based capture and processing. A size selection step is used to separate antibody nucleotides before library preparation. (B) Cell hashing allows multiplexing of samples. Ubiquitous antibodies are used to tag cells in each sample with a hashtag oligonucleotide. The cells are then mixed for cell capture and processing and the hashtags used to computationally identify the origin of each cell. Images from <https://cite-seq.com> [47].

1.4 Analysing scRNA-seq data

Cell capture technologies and scRNA-seq protocols have developed rapidly but the data they produce still presents a number of challenges. Existing approaches are inefficient, capturing around 10 percent of transcripts in a cell [58]. When combined with the low sequencing depth per cell this results in a limited sensitivity and an inability to detect lowly expressed transcripts. The small amount of starting material also contributes to high levels of technical noise, complicating downstream analysis and making it difficult to detect biological differences [59]. In order to capture cells they must first be dissociated into single-cell suspensions but this step can be non-trivial. Some tissues or cell types may be more difficult to separate than others and the treatments required to break them apart may affect the health of the cells and their transcriptional profiles. It has been suggested that damage caused by dissociation can be minimised by using a cold-active protease to separate cells [60]. Other cell types may be too big or have other characteristics that prevent them being captured. Multiple cells may be captured together or empty wells or droplets sequenced making quality control of datasets an important consideration.

As well as increasing technical noise the small amounts of starting material and low sequencing depth mean there are many occasions where zero counts are recorded, indicating no measured expression for a particular gene in a particular cell. These zero counts often represent the true biological state we are interested in as we that different genes will be expressed by different cell types. However, zeros can also be the result of confounding biological factors such as stage in the cell cycle, transcriptional bursting and environmental interactions which cause genuine changes in expression but that might not be of interest to a particular study. On top of this there are effects that are purely technical factors. In particular, sampling effects, which can result in “dropout” events where a transcript is truly expressed in a sample but is not observed in the sequencing data [61]. In bulk experiments these effects are limited by averaging across the cells in a sample and by the greater sequencing depth, but for single-cell experiments they can present a significant challenge for analysis as methods must account for the missing information and the large number of zeros may cause the assumptions of existing methods to be violated.

One approach to tackling the problem of too many zeros is to use zero-inflated versions of common distributions [62–65]. However, it is still debatable whether scRNA-seq datasets, particularly those from droplet-based capture protocols [66], are truly zero inflated or if the additional zeros are better modelled with standard distributions with lower means. Another approach to analysis is to impute some of the zeros, replacing them with estimates of how expressed those genes truly are based on their expression in similar cells using methods such as MAGIC [67], SAVER [68,69] or scImpute [70]. However, imputation comes with the risk of introducing false structure that is not actually present in the samples [71].

Bulk RNA-seq experiments usually involve predefined groups of samples, for example cancer cells and normal tissue, different tissue types or treatment and control groups. It is possible to design scRNA-seq experiments in the same way by sorting cells into known groups based on surface markers, sampling them at a series of time points or comparing treatment groups, but often single-cell experiments are more exploratory. Many of the single-cell studies to date have sampled developing or mature tissues and attempted to profile the cell types that are present [32,72–77]. This approach is best exemplified by the Human Cell Atlas project which is attempting to produce a reference of the transcriptional profiles of all the cell types in the human body [78]. Similar projects exist for other species and specific tissues [79–88].

As scRNA-seq datasets have become more widely available, a standard analysis workflow has developed which can be applied to many experiments [89,90]. This workflow can be divided into four phases shown in Figure 1.10: 1) Data acquisition, pre-processing of samples to produce a cell by gene expression matrix, 2) Data cleaning, quality control to refine the dataset used for analysis, 3) Cell assignment, grouping or ordering of cells based on their transcriptional profile, and 4) Gene identification to find genes that represent particular groups and can be used to interpret them. Within each phase a range of processes may be used and there are now many tools available for completing each of them, with over 450 tools currently available. An introduction to the phases of scRNA-seq analysis is provided here but the analysis tools landscape is more fully explored in **Chapter 2**.

1.4.1 Pre-processing and quality control

The result of a sequencing experiment is typically a set of image files from the sequencer or a FASTQ file containing nucleotide reads but for most analyses we use an expression matrix where each column is a cell, each row is a feature and the values indicate expression levels. To produce this matrix there is a series of pre-processing steps, typically beginning will some quality control of the raw reads. Reads are then aligned to a reference genome and the number of reads overlapping annotated features (genes or transcripts) is counted. Probabilistic quantification methods can be applied to full-length scRNA-seq datasets but have

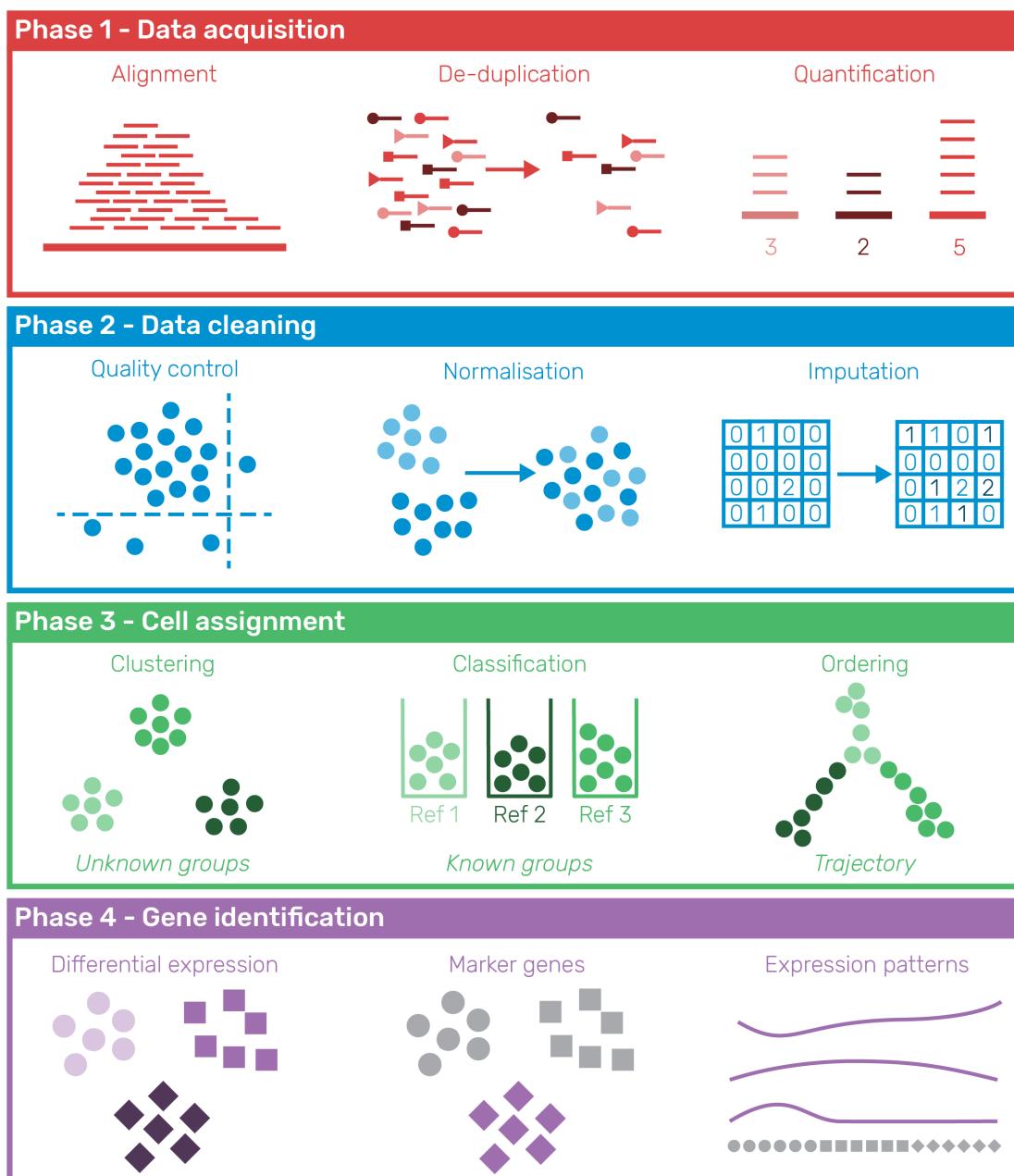


Figure 1.10: Phases of a standard scRNA-seq analysis workflow. In Phase 1 (Data acquisition) samples are pre-processed to produce an expression matrix that undergoes quality control and filtering in Phase 2 (Data cleaning). Phase 3 (Cell assignment) groups or orders cells according to their transcriptional profile and in Phase 4 (Gene identification) genes describing those groups are discovered and used to interpret them.

required adaptations such as the alevin method for UMI-based datasets [91] in the Salmon package. When using conventional alignment, UMI samples need extra processing with tools like UMI-tools [92], umis [93] or zUMIs [94] in order to assign cell barcodes and deduplicate UMIs. For datasets produced using the 10x Chromium platform the company provides the Cell Ranger software which is a complete pre-processing pipeline that also includes an automated downstream analysis. Other packages such as scPipe [95] also aim to streamline this process with some such as Falco [96] designed to work on scalable cloud-based infrastructure which may be required as bigger datasets continue to be produced.

Quality control of individual cells is important as experiments will contain low-quality cells that can be uninformative or lead to misleading results. Particular types of cells that are commonly removed include damaged cells, doublets where multiple cells have been captured together [97,98] and empty droplets or wells that have been sequenced but do not contain a cell [99]. Quality control can be performed on various levels including the quality scores of the reads themselves and how or where they align to features of the expression matrix. The Cellity package attempts to automate this process by inspecting a series of biological and technical features and using machine learning methods to distinguish between high and low-quality cells [100]. However, the authors found that many of the features were cell type specific and more work needs to be done to make this approach more generally applicable. The scater package [101] emphasises a more exploratory approach to quality control at the expression matrix level by providing a series of functions for visualising various features of a dataset. These plots can then be used for selecting thresholds for removing cells. Plate-based capture platforms can produce additional biases based on the location of individual wells, a problem which is addressed by the OEFinder package which attempts to identify and visualise these “ordering effects”[102].

Filtering and selection of features also deserves attention. Genes or transcripts that are lowly expressed are typically removed from datasets in order to reduce computational time and the effect of multiple-testing correction but it is unclear how many counts indicate that a gene is truly “expressed”. Many downstream analyses operate on a selected subset of genes and how they are selected can have a dramatic effect on their results [103]. These features are often selected based on how variable they are across the dataset but this may be a result of noise rather than biological importance. Alternative selection methods have been proposed such as M3Drop which identifies genes that have more zeros than would be expected based on their mean expression [104].

1.4.2 Normalisation and integration

Effective normalisation is just as crucial for single-cell experiments as it is for bulk RNA-seq datasets. FPKM or TPM transformations can be used, but for UMI data the gene length correction is not required as reads only come from the ends of transcripts [105]. Normalisation methods designed for detecting differential expression between bulk samples such as TMM or the DESeq method can be applied, but it is unclear how suitable they in the single-cell context. Many of the early normalisation methods developed specifically for scRNA-seq data made use of spike-ins. These are synthetic RNA sequences added to cells in known quantities such as the External RNA Controls Consortium (ERCC) control mixes, a set of 92 transcripts from 250 to 2000 base pairs long based on bacterial plasmids. Brennecke et al. [106], Ding et al. [107] and Grün, Kester and van Oudenaarden [58] all propose methods for estimating technical variance using spike-ins, as does Bayesian Analysis of Single-Cell Sequencing data (BASiCS) [108]. Using spike-ins for normalisation assumes that they properly capture the dynamics of the underlying dataset and even if this is the case it is restricted to protocols where they can be added which does not include droplet-based capture techniques. The scran package implements a method that doesn’t rely on spike-ins, instead using a pooling approach to compensate for the large number of zero counts where expression levels are summed across

similar cells before calculating size factors that are deconvolved back to the original cells [109]. The BASiCS method has also been adapted to experiments without spike-ins by integrating data replicated across batches [110], but only for designed experiments where groups are known in advance.

Early scRNA-seq studies often made use of only a single sample but as technologies have become cheaper and more widely available it is common to see studies with multiple batches or making use of publicly available data produced by other groups. While this expands the potential insights to be gained it presents a problem as to how to integrate these datasets [111]. A range of computational approaches for performing integration have been developed including bbknn [112], ClusterMap [113], kBET [114], LIGER [115], matchSCore [116], Scanorama [117] and scMerge [118]. The alignment approach in the Seurat package [119] uses Canonical Correlation Analysis (CCA) [120] to identify a multi-dimensional subspace that is consistent between datasets. Dynamic Time Warping (DTW) [122] is then used to stretch and align these dimensions so that the datasets are similarly spread along them. Some tasks can be performed using these aligned dimensions, but as the original expression matrix is unchanged the integration is not used for other tasks such as differential expression testing. The authors of scran use a Mutual Nearest Neighbours (MNN) approach that calculates a cosine distance between cells in different datasets then identifies those that share a neighbourhood [123]. Batch correction vectors can then be calculated and subtracted from one dataset to overlay them. A recent update to the Seurat method combines these approaches by applying CCA before using the MNN approach to identify “anchor” cells that have common features in the different datasets [124].

1.4.3 Grouping cells

Grouping similar cells is a key step in analysing scRNA-seq datasets that is not usually required for bulk experiments and as such it has been a key focus of methods development. Over one hundred tools have been released for clustering cells which attempt to address a range of technical and biological challenges [125]. Some of these methods include SINgle CELL RNA-seq profiling Analysis (SINCERA) [126], Single-Cell Consensus Clustering (SC3) [127], single-cell latent variable model (scLVM) [76] and Spanning-tree Progression Analysis of Density-normalised Events (SPADE) [128], as well as BackSPIN which was used to identify nine cell types and 47 distinct subclasses in the mouse cortex and hippocampus in one of the earliest studies to demonstrate the possibilities of scRNA-seq [72]. All of these tools attempt to cluster similar cells together based on their expression profiles, forming groups of cells of the same type. The clustering method in the Seurat package [129] has become particularly popular and has been shown to perform well, particularly on UMI datasets [130,131]. This method begins by selecting a set of highly variable genes then performing PCA on them. A set of dimensions is then selected that contains most of the variation in the dataset. Alternatively, if Seurat’s alignment method has been used to integrate datasets the aligned CCA dimensions are used instead. Next, a Shared Nearest Neighbours (SNN) graph is constructed by considering the distance between cells in this multidimensional space and the overlap between shared neighbourhoods. Seurat uses a Euclidean distance but it has been suggested that correlations can provide better results [132]. In order to separate cells into clusters, a community detection algorithm such as Louvain optimisation [133] is run on the graph with a resolution parameter that controls the number of clusters that are produced. Selecting this parameter and similar parameters in other methods is difficult but important as the number of clusters selected can affect the interpretation of results. I address this problem with a visualisation-based approach in **Chapter 4**.

For tissue types that are well understood or where comprehensive references are available an alternative to unsupervised clustering is to directly classify cells. This can be done using a gating approach based on the expression of known marker genes similar to that commonly

used for flow cytometry experiments. Alternatively, machine learning algorithms can be used to perform classification based on the overall expression profile. Methods such as scmap [134], scPred [135], CaSTLe [136] and Moana [137] take this approach. Classification has the advantage of making use of existing knowledge and avoids manual annotation and interpretation of clusters which can often be difficult and time consuming. However, it is biased by what is present in the reference datasets used and typically cannot reveal previously unknown cell types or states. As projects like the Human Cell Atlas produce well-annotated references based on scRNA-seq data the viability of classification and other reference-based methods will improve.

1.4.4 Ordering cells

In some studies, for example in development where stem cells are differentiating into mature cell types, it may make sense to order cells along a continuous trajectory from one cell type to another instead of assigning them to distinct groups. Trajectory analysis was pioneered by the Monocle package which used dimensionality reduction and computation of a minimum spanning tree to explore a model of skeletal muscle differentiation [77]. Since then the Monocle algorithm has been updated [138] and a range of others developed including TSCAN [139], SLICER [140], CellTree [141], Sincell [142], Mpath [143] and Slingshot [144]. In their review of trajectory inference methods, Cannoodt, Saelens and Saeys break the process into two steps (Figure 1.11) [145]. In the first step, dimensionality reduction techniques such as principal component analysis (PCA) [146], t-distributed stochastic neighbourhood embedding (t-SNE) [147] or uniform manifold approximation and projection (UMAP) [148] are used to project cells into lower dimensions where the cells are clustered or a graph constructed between them. The trajectory is then created by finding a path through the cells and ordering the cells along it. The same group of authors have conducted a comprehensive comparison of ordering methods evaluating their performance on a range of real and synthetic datasets [149]. To do so the authors had to develop metrics for comparing trajectories from different methods and built a comprehensive infrastructure for running and evaluating software tools. They found that the performance of methods depended on the underlying topology of the data and that multiple complementary trajectory inference approaches should be used to better what is occurring.

An alternative continuous approach is the cell velocity technique [150] introduced for scRNA-seq data in the velocyo package [151]. RNA-seq studies typically focus on the expression of mature mRNA molecules but a sample will also contain immature mRNA that are yet to be spliced. Examining the reads assigned to introns can indicate newly transcribed mRNA

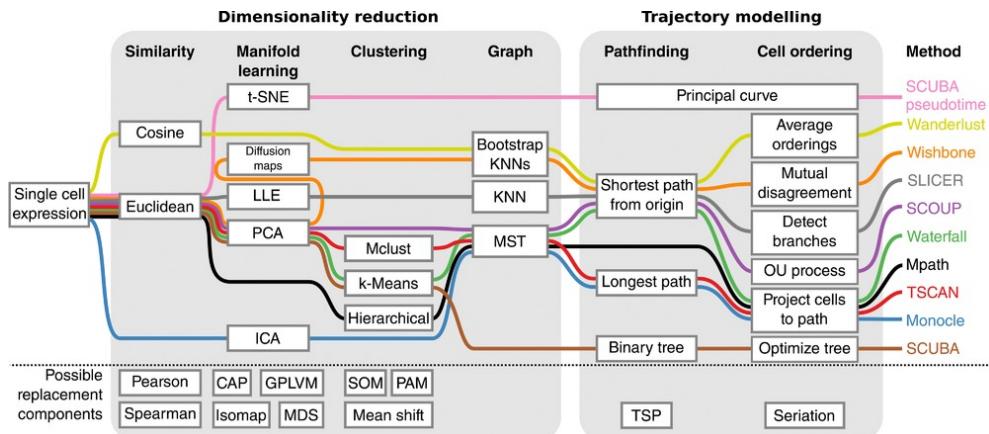


Figure 1.11: Trajectory inference framework as described by Cannoodt, Saelens and Saeys. In the first stage, dimensionality reduction is used to convert the data to a simpler representation. In the second stage a trajectory is identified and the cells ordered. Image from Cannoodt, Saelens and Saeys “Computational methods for trajectory inference from single-cell transcriptomics” [145].

molecules and therefore which genes are currently active. Instead of assigning cells to discrete groups or along a continuous path velocito uses reads from unspliced regions to place them in a space and create a vector indicating the direction in which the transcriptional profile is heading. This vector can show that a cell is differentiating in a particular way or that a specific transcriptional program has been activated.

Deciding on which cell assignment approach to use depends on the source of the data, the goals of the study and the questions that are being asked. Both grouping and ordering can be informative and it is often useful to attempt both on a dataset and see how they compare.

1.4.5 Gene detection and interpretation

Once cells are assigned by clustering or ordering the problem is to interpret what these groups represent. For clustered datasets this is usually done by identifying genes that are differentially expressed across the groups or marker genes that are expressed in a single cluster. Many methods have been suggested for testing differential expression, some of which take in to account the unique features of scRNA-seq data, for example D3E [152], DEsingle [65], MAST [153], scDD [154] and SCDE [155]. The large number of cells in scRNA-seq datasets means that some of the problems that made standard statistical tests unsuitable for bulk RNA-seq experiments do not apply. Simple methods like the unpaired Wilcoxon rank-sum test (or Mann-Whitney U test), Student's t-test or logistic regression may give reasonable results in this setting. Methods originally developed for bulk experiments have also been applied to scRNA-seq datasets but the assumptions they make may not be appropriate for single-cell data. Methods such as ZINB-WaVe [64] may be required to transform the data so that bulk testing methods are appropriate. A comprehensive evaluation of 36 differential expression testing approaches found that methods developed for bulk RNA-seq did not perform worse than scRNA-seq specific methods, however, the performance of bulk methods depended on how lowly-expressed genes were filtered [156]. The authors also observed that some methods were biased in the types of features they tended to detect as differentially expressed. Often the goal is not to find all the genes that are differentially expressed between groups but to identify genes which uniquely mark particular clusters. This goal is open to alternative approaches, such as the Gini coefficient which measures unequal distribution across a population. Another approach is to construct machine learning classifiers for each gene to distinguish between one group and all other cells. Genes that give good classification performance should be good indicators of what is specific to that cluster. It is also possible to identify genes that might have the same mean between groups but differ in variance or other characteristics of their expression distribution.

When cells have been ordered along a continuous trajectory the task is slightly different. Instead of testing for a difference in means between two groups, the goal is to find genes that have a relationship between expression and pseudotime. This can be accomplished by fitting splines to the relationship between pseudotime and expression and testing the fitting coefficients. For more complex trajectories it can also be useful to find genes that are differently expressed along each side of a branch point. Monocle's BEAM (Branch Expression Analysis Modelling) method does this using a likelihood ratio test between splines where the branch assignments are known or unknown [157]. Genes that are associated with a trajectory are important in their own right as they describe the biology along a path but they can also be used to identify cell types at end points.

Interpreting the meaning of detected marker genes is a difficult task and is likely to remain so. Some methods have been developed for this task such as celaref which suggests cluster labels based on similarity of marker genes to an already characterised reference dataset. Gene set testing to identify related categories such as Gene Ontology terms can also help but often it is necessary to rely on the results of previous functional studies. Ultimately this can only

be reliably done by working closely with experts who have significant domain knowledge in the cell types being studied. An additional concern for unsupervised scRNA-seq studies is that the same genes are used for clustering or ordering and determining what those clusters or trajectories mean. This is a problem addressed by Zhang, Kamath and Tse who suggest a differential expression test using a long-tailed distribution for testing genes following clustering [158].

1.4.6 Alternative analyses

Some uses of scRNA-seq data fall outside the most common workflow and methods have been developed for a range of other purposes. For example, methods have been designed for assigning haplotypes to cells [159], detecting allele-specific expression [160–162], identifying alternative splicing [163–165] or calling single nucleotide or complex genomic variants [166–168]. Other methods have been designed for specific cell types or tissues such as reconstructing immune cell receptors on B-cells (BraCeR [169], BRAPeS [170], VDJPuzzle [171]) or T-cells (TraCeR [172], TRAPeS [173]) or interrogating the development of cancer samples (HoneyBADGER [174], SSrGE [166]). Most future studies can be expected to continue to follow common practice but it is also expected that researchers will continue to push the boundaries of what it is possible to study using scRNA-seq technologies.

1.4.7 Evaluation of scRNA-seq analysis methods

Although the analysis for many scRNA-seq studies follow a standard workflow there is wide variation in the tools that are used. We are now at the stage where there are multiple software packages for completing every stage of analysis. Deciding which tools to use can be difficult and depends on a number of factors including effectiveness, robustness, scalability, availability, ease of use and quality of documentation. In terms of effectiveness, publications describing analysis methods should demonstrate two things: 1) they can perform the task they are designed for (at least as well as existing methods) and 2) performing that task leads to biological insights. Answering the first question is difficult using real datasets as often the underlying truth is not known. For this reason simulation techniques are commonly used to produce synthetic datasets in order to evaluate methods. I present a software package for simulating scRNA-seq data in **Chapter 3**. Simulation is an efficient and flexible approach for producing gold standard datasets but synthetic data can never fully reproduce a real dataset. An alternative approach is to carefully construct a gold standard biological dataset by combining samples where the true differences are known [175]. These datasets can be extremely useful but are difficult, time-consuming and expensive to produce and can only reproduce a limited set of scenarios. Comprehensive evaluations of methods have already been conducted for some aspects of scRNA-seq analysis including clustering [130,131], trajectory inference [149] and differential expression testing [156] but these will need to continue to be performed and updated as the field matures and new methods are developed.

1.5 Kidney development

One area that has particularly benefitted from the possibilities created by scRNA-seq technology is developmental biology. Although the genes involved in the development of many organs are now well understood, arriving at this knowledge has required many painstaking experiments to investigate a single gene at a time. During development cells are participating in a continuous dynamic process involving the maturation from one cell type to another and the creation of new cell types. Single-cell RNA-seq captures a snapshot of the expression of all the genes involved in this process, allowing the transcriptome of intermediate and mature cells to be

studied. This has revealed that some of the genes thought to be markers of specific cell types are more widely expressed or involved in other processes.

1.5.1 Structure and function

The kidney is the organ responsible for filtering the blood in order to remove waste products. In humans, kidneys grow as a pair of functional organs with each being around the size of an adult fist and weighing about 150 grams. Blood flows into the kidney via the renal artery and the blood vessels form a tree-like branching structure with ever smaller capillaries (Figure 1.12A). At the end of these branches are nephrons, the functional filtration unit of the kidney (Figure 1.12B). Humans can have around 1 million nephrons [176] that are formed during development and just after birth, however they cannot be regenerated during adulthood. A capillary loop is formed inside a structure at the end of the nephron called a glomerulus and surrounded by Bowman's capsule. Here specialised cells called podocytes create a structure called the slit diaphragm that allows water, metal ions and small molecules to be filtered while keeping blood cells and larger species such as proteins trapped within the bloodstream. The rest of the nephron is divided into segments that are responsible for different processes involved in balancing the concentration of these species in the filtrate (Figure 1.12C). The tubular segments of the nephron are surrounded by capillaries, allowing molecules to be transferred between the filtrate and blood as required. The first segment of the nephron is the proximal tubule. Here common biomolecules such as glucose, amino acids and bicarbonate are reabsorbed into the bloodstream, as is most of the water present. Other molecules including urea and ammonium ions are secreted from the blood into the filtrate at this stage. The proximal tubule is followed by the loop of Henle and the distal tubule where ions are reabsorbed including potassium, chlorine, magnesium and calcium. The final segment is the collecting duct where salt concentrations are balanced by exchanging sodium in the filtrate for potassium in the bloodstream using a process controlled by the hormone aldosterone. The remaining filtrate is then passed to the ureter where it is carried to the bladder and collected as urine while the blood leaves via the renal vein. In order to perform this complex series of reabsorption and secretion steps, each segment of the nephron is made up of specialised cell types with their own sets of signalling and transporter proteins. The filtration process is repeated about 12 times every hour with around 200 litres of blood being filtered during a day. Aside from removing waste and maintaining the balance of molecular species in the bloodstream, the kidneys also play a role in the activation of vitamin D. Other important functions include synthesising the hormones erythropoietin, which stimulates red blood cell production, and renin, which is part of the pathway that controls fluid volume and the constriction of arteries to regulate blood pressure.

Chronic kidney disease is a major health problem in Australia with 1 percent of the population (237 800 people) diagnosed with it in 2017 [177] and it being considered a contributory factor for 13 percent of deaths [178]. Early stages of the disease can be managed but once it becomes severe the only treatment options are dialysis, which is expensive, time consuming and unpleasant, or a kidney transplant. There are also a range of genetic developmental kidney disorders that have limited treatment options and can profoundly affect quality of life. Understanding how the kidney grows and develops is key to developing new treatments that may improve kidney function or repair damage.

1.5.2 Stages of development

The kidney develops from a region of the early embryo called the intermediate mesoderm and occurs in three phases with a specific spatial and temporal order [179]. The first phase results in the pronephros which consists of 6–10 pairs of tubules that forms the mature kidney in most primitive vertebrates such as hagfish. By about the fourth week of human embryonic

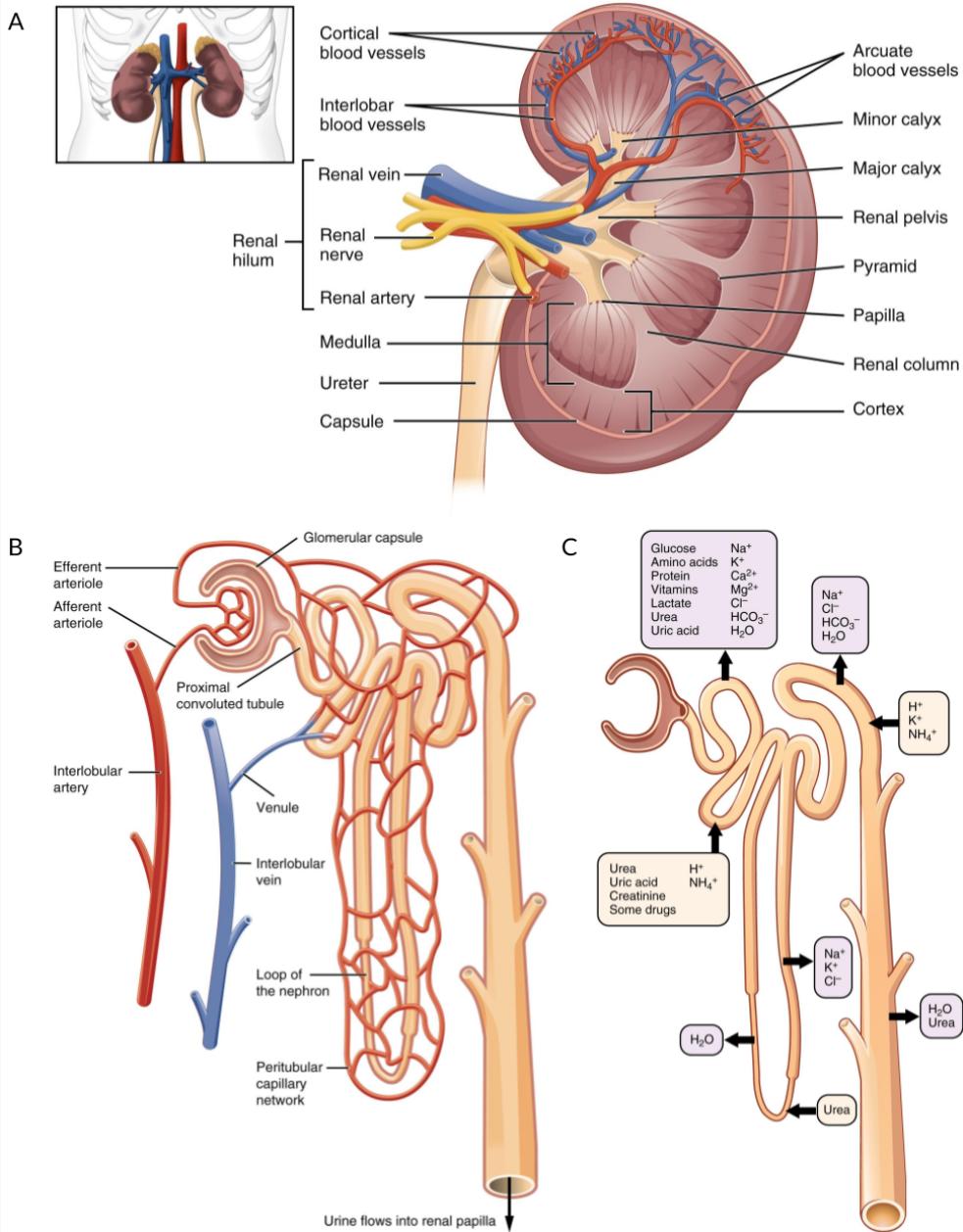


Figure 1.12: Structure of the kidney and nephron. (A) Blood flows into the kidney via the renal artery which branches into ever smaller capillaries. (B) At the end of these capillaries is the nephron, the functional filtration unit of the kidney, which is surrounded by a capillary network. (C) Different segments of the kidney are responsible for transferring specific molecular species between the bloodstream and the filtrate. Figure adapted using images from OpenStax College via Wikimedia Commons under a CC BY 3.0 license.

development this structure dies off and is replaced by the mesonephros, which is the form of kidney present in most fish and amphibians. The mesonephros is functional during weeks 4–8 of human embryonic development before degenerating, although parts of its duct system go on to form part of the male reproductive system. The final phase of human kidney development results in the metanephros which begins developing at around five weeks and continues to around week 36 to become the permanent and functional kidney [180]. Individual nephrons grow in a similar series of stages. Cells from the duct that will become the ureter begin to invade the surrounding metanephric mesenchyme forming a ureteric bud. Interactions between these cell types, including Wnt signalling, cause mesenchymal cells to condense around the ureteric bud forming a stem cell population known as the cap mesenchyme, which expresses genes such as Six2 and Cited1. Cells from the cap mesenchyme first form a renal vesicle, a primitive structure with a lumen, which extends to form an S-shaped body (Figure 1.13). By this stage the lumen has joined with the ureteric bud to form a continuous tubule. The s-shaped body continues to elongate with podocytes beginning to develop and form a glomerulus at one end and other specialised cells arising along the length of the tubule to form the various nephron segments. Several signalling pathways and cell–cell interactions are involved in this process, including Notch signalling. While all nephrons form before birth they continue to elongate and mature postnatally.

Most of our understanding of kidney development comes from studies using mouse models and other model species. While these have greatly added to our knowledge they do not completely replicate human kidney development and there are known to be significant differences in the developmental timeline, signalling pathways and gene expression between species [181]. To better understand human kidney development we need models that reproduce the human version of this process.

1.5.3 Growing kidney organoids

One alternative model of human kidney development is to grow miniature organs in a lab. Known as organoids, these tissues are grown from pluripotent stem cells provided with the right sequence of conditions and growth factors [182]. Naturally occurring embryonic stem

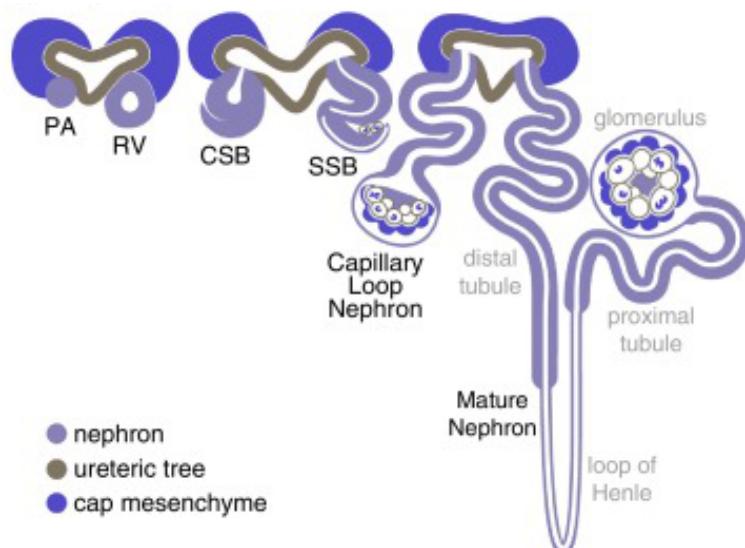


Figure 1.13: Diagram of the stages of nephron maturation. The nephron begins as a pre-tubular aggregate (PA) which forms a renal vesicle (RV), comma-shaped body (CSB), S-shaped body (SSB), capillary loop nephron and mature nephron. The connection between the forming nephron and the lumen of the adjacent ureteric epithelium forms at late renal vesicle stage. Adapted from Little “Improving our resolution of kidney morphogenesis across time and space” [180].

cells can be used but a more feasible approach is to reprogram mature cell types (typically fibroblasts from skin samples) using a method discovered by Takahashi and Yamanaka [183,184]. Under this protocol, cells are supplied with the transcription factors Oct3/4, Sox2 and Klf4 using retroviral transduction. The resulting cells have the ability to differentiate into any cell type and are known as induced pluripotent stem cells (iPSCs). By culturing iPSCs under the right conditions the course of differentiation can be directed, and protocols for growing a range of tissues have been developed including stomach, intestine, liver, lung and brain [185]. The first protocol for growing kidney organoids was published in 2015 by Takasato et al. (Figure 1.14) [186].

Using this protocol, iPSCs are first grown on a plate where Wnt signalling is induced by the presence of CHIR, an inhibitor of glycogen kinase synthase 3. After several days of growth the growth factor FGF9 is added which is required to form the intermediate mesoderm. Following several more days of growth the cells are removed from the plate and formed into three dimensional pellets. A short pulse of CHIR is added to again induce Wnt signalling and the pellets continue to be cultured in the presence of FGF9. Growth factors are removed after about five days of 3D culture and the organoids continue to grow for a further two weeks, at which point tubular structures have formed. These kidney organoids have been extensively characterised using both immunofluorescence imaging and transcriptional profiling by RNA-seq. Imaging showed that the tubules are segmented and express markers of podocytes, proximal tubule, distal tubule and collecting duct, however individual tubules are not connected in the same way they would be in a real kidney. By comparing RNA-seq profiles with those from a range of developing tissues, the organoids from this protocol were found to be most similar to trimester one and two foetal kidney [186]. While the bulk transcriptional profiles may be similar, this analysis does not confirm that individual cell types are the same between lab-grown kidney organoids and the true developing kidney. Further studies using this protocol have shown that it is reproducible with organoids grown at the same time having very similar transcriptional profiles, but organoids from different batches can be significantly different, potentially due to differences in the rate at which they develop [187].

While they are not a perfect model of a developing human kidney, organoids have several advantages over other models [188]. In particular, they have great potential for uses in the modelling of developmental kidney diseases [189]. Cells from a patient with a particular mutation can be reprogrammed and used to grow organoids that can then be used for functional studies or drug screening. Alternatively, gene editing techniques can be used to insert the mutation

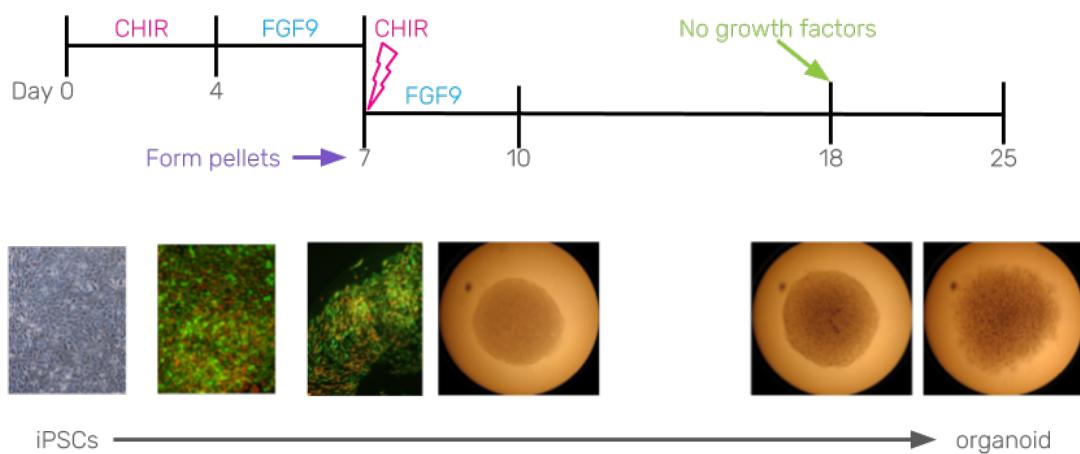


Figure 1.14: Diagram of the Takasato et al. kidney organoid protocol. iPSCs are cultured on a plate in the presence of CHIR followed by FGF9. After about a week cells are formed into three-dimensional pellets and a short CHIR pulse is applied. The pellets continue to be cultured in FGF9 for a further 10 days before all growth factors are removed. The organoids now contain tubular structures replicating human nephrons. Adapted from Takasato et al. “Kidney organoids from human iPS cells contain multiple lineages and model human nephrogenesis” [186].

into an existing cell line or correct the mutation in the patient line, allowing comparisons on the same genetic background. Modified versions of the protocol that can produce much larger numbers of organoids, for example by growing them in swirler cultures [190], could potentially be used to produce cells in sufficient numbers for cellular therapies. Extensive work has been done to improve the protocol in other ways as well, such as improving the maturation of the organoids or directing them more towards particular segments. Overall, kidney organoids open up many possibilities for studies to better understand kidney development and potentially help develop new treatments for kidney disease.

1.5.4 Kidney scRNA-seq studies

Several analyses of scRNA-seq from kidney tissues have already been performed, including of developing and adult organs from human and mouse as well as kidney organoids and specific kidney cell types [191,192]. Park et al. analysed almost 58 000 cells from mouse kidneys and found that disease-causing genetic mutations with similar phenotypes arise in the same cell types [193]. They also identified a new transitional cell type in the collecting duct. Young et al. compared human foetal, paediatric and adult kidney cells to cells from renal cancers and identified links to specific cell types, including that Wilms tumours are similar to specific foetal cell types and that adult renal cell carcinoma is related to a little-known proximal tubule cell type [194]. Lindström et al. focused on the developing mouse and human kidney [195] while Wu et al. examine kidney organoids [196]. They found that organoid protocols produced a range of cell types but that those cells were immature and 10–20 percent of cells came from non-kidney lineages. Many of these were neuronal and their formation could be limited by inhibiting the BDNF/NTRK2 pathway. In **Chapter 5** I analyse another organoid dataset generated by collaborators using the Takasato protocol for the purposes of profiling the cell types present. I demonstrate a range of analysis tools, focusing on the decisions that need to be made during the analysis and the effects they might have on the results.

1.6 Thesis overview and aims

The tools and techniques for analysing scRNA-seq data have developed rapidly along with the technology. My thesis aims to chart the progress of these methods over the last few years, contribute to the development in the field and apply those methods to a scRNA-seq dataset to better understand the cellular composition of kidney organoids. I have completed these aims in the following ways:

1. Construction and maintenance of a database recording the development of scRNA-seq analysis methods and analysis of its contents (**Chapter 2**).
2. Development of a software package and method for easily simulating realistic scRNA-seq datasets that can be used for the evaluation and development of analysis methods (**Chapter 3**).
3. Development of an algorithm for visualising clustering results at multiple resolutions that can be used to select clustering parameters and a software package that implements it (**Chapter 4**).
4. Application of scRNA-seq analysis methods to a kidney organoid dataset in order to uncover and better understand the cell types that are present and demonstrate differences between approaches (**Chapter 5**).

2

The scRNA-seq tools landscape

Have you ever seen a sound
Have you listened to an image
Have you ever touched a thought
Have you ever tasted nothing
Have you ever told a lie
That was true more than truth
Because truth it had lied
All its life when it spoke to you?

— The Cat Empire

Miserere, 2005

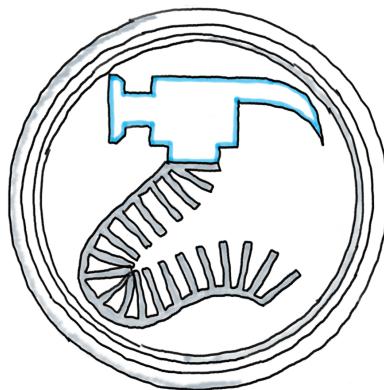
2. TOOLS

What
tools are
there?

What
can they
do?

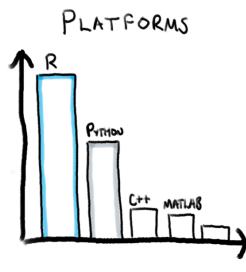
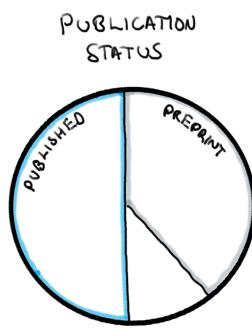
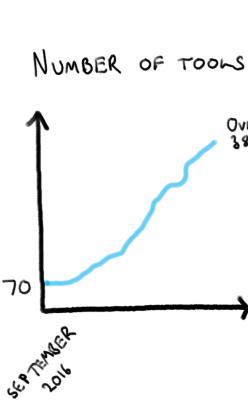
Where
can I
find them?

Online
database
of tools



Search
by
Category

scRNA-tools



2.1 Introduction

When I began my PhD in early 2016, single-cell RNA sequencing technologies were just beginning to become widely available. Since then there has been a rapid uptake of the technologies and there are now many studies using this approach. Along with the growth in the adoption of scRNA-seq technologies, there has been an explosion in the number of software tools for analysing these datasets. This chapter charts the growth in the scRNA-seq analysis landscape over time.

In 2016 there were relatively few analysis methods available and to answer questions like how many tools perform a particular task, which areas are developers focusing on or is there a tool for doing a particular task, I began to record details about them. Inspired by similar projects such as Sean Davis's Awesome Single Cell page [197] I decided to make this collection public. This turned out to be useful to other researchers and over time a simple spreadsheet became the scRNA-tools database and website (<https://scRNA-tools.org>). Our paper published in *PLoS Computational Biology* describing this resource forms the main part of this chapter [198].

By having access to details about existing analysis tools we were able to explore how the field has developed. We found that computational researchers had focused their efforts on analysis tasks specific to scRNA-seq data, such as clustering and ordering of cells or handling the larger numbers of zero counts. We also saw that many of the tools performed tasks common to several stages of analysis, including various approaches to dimensionality reduction and visualisation of data and results. Details recorded in the database show that developers of scRNA-seq analysis tools tend to embrace an open-source and open-science approach, for example most tools are developed on GitHub where others can ask questions and submit improvements. The majority of tools are also available under open-source licenses allowing their code to be reused for other purposes, although there is also a significant proportion that do not have any associated license. Tools are commonly made public by releasing a preprint publication, making them quickly available to the community and giving early adopters a chance to contribute to their development.

Sections at the end of this chapter present a version of some the analysis presented in the paper based on the most recent version of the database (**Section 2.3**), as well as details about the usage of the scRNA-tools website over time (**Section 2.4**).

2.2 scRNA-tools publication

PLOS COMPUTATIONAL BIOLOGY

RESEARCH ARTICLE

Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database

Luke Zappia^{1,2}, Belinda Phipson¹, Alicia Oshlack^{1,2*}

1 Bioinformatics, Murdoch Children's Research Institute, Melbourne, Victoria, Australia, **2** School of Biosciences, Faculty of Science, University of Melbourne, Melbourne, Victoria, Australia

* alicia.oshlack@mcri.edu.au



OPEN ACCESS

Citation: Zappia L, Phipson B, Oshlack A (2018) Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database. PLoS Comput Biol 14(6): e1006245. <https://doi.org/10.1371/journal.pcbi.1006245>

Editor: Dina Schneidman, Hebrew University of Jerusalem, ISRAEL

Received: December 6, 2017

Accepted: May 30, 2018

Published: June 25, 2018

Copyright: © 2018 Zappia et al. This is an open access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Data Availability Statement: All relevant data are within the paper and its Supporting Information files.

Funding: Luke Zappia is supported by an Australian Government Research Training Program (RTP) Scholarship. Alicia Oshlack is supported through a National Health and Medical Research Council Career Development Fellowship APP1126157. MCRI is supported by the Victorian Government's Operational Infrastructure Support Program. The funders had no role in study design,

Abstract

As single-cell RNA-sequencing (scRNA-seq) datasets have become more widespread the number of tools designed to analyse these data has dramatically increased. Navigating the vast sea of tools now available is becoming increasingly challenging for researchers. In order to better facilitate selection of appropriate analysis tools we have created the scRNA-tools database (www.scRNA-tools.org) to catalogue and curate analysis tools as they become available. Our database collects a range of information on each scRNA-seq analysis tool and categorises them according to the analysis tasks they perform. Exploration of this database gives insights into the areas of rapid development of analysis methods for scRNA-seq data. We see that many tools perform tasks specific to scRNA-seq analysis, particularly clustering and ordering of cells. We also find that the scRNA-seq community embraces an open-source and open-science approach, with most tools available under open-source licenses and preprints being extensively used as a means to describe methods. The scRNA-tools database provides a valuable resource for researchers embarking on scRNA-seq analysis and records the growth of the field over time.

Author summary

In recent years single-cell RNA-sequencing technologies have emerged that allow scientists to measure the activity of genes in thousands of individual cells simultaneously. This means we can start to look at what each cell in a sample is doing instead of considering an average across all cells in a sample, as was the case with older technologies. However, while access to this kind of data presents a wealth of opportunities it comes with a new set of challenges. Researchers across the world have developed new methods and software tools to make the most of these datasets but the field is moving at such a rapid pace it is difficult to keep up with what is currently available. To make this easier we have developed the scRNA-tools database and website (www.scRNA-tools.org). Our database catalogues analysis tools, recording the tasks they can be used for, where they can be downloaded from and the publications that describe how they work. By looking at this database we can see that developers have focused on methods specific to single-cell data and that they

PLOS Computational Biology | <https://doi.org/10.1371/journal.pcbi.1006245> June 25, 2018

1 / 14

PLOS | COMPUTATIONAL
BIOLOGY

Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database

data collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: The authors declare that no competing interests exist.

embrace an open-source approach with permissive licensing, sharing of code and release of preprint publications.

This is a *PLOS Computational Biology* Software paper.

Introduction

Single-cell RNA-sequencing (scRNA-seq) has rapidly gained traction as an effective tool for interrogating the transcriptome at the resolution of individual cells. Since the first protocols were published in 2009 [1] the number of cells profiled in individual scRNA-seq experiments has increased exponentially, outstripping Moore's Law [2]. This new kind of transcriptomic data brings a demand for new analysis methods. Not only is the scale of scRNA-seq datasets much greater than that of bulk experiments but there are also a variety of challenges unique to the single-cell context [3]. Specifically, scRNA-seq data is extremely sparse (there is no expression measured for many genes in most cells), it can have technical artefacts such as low-quality cells or differences between sequencing batches and the scientific questions of interest are often different to those asked of bulk RNA-seq datasets. For example many bulk RNA-seq datasets are generated to discover differentially expressed genes through a designed experiment while many scRNA-seq experiments aim to identify or classify cell types in complex tissues.

The bioinformatics community has embraced this new type of data at an astonishing rate, designing a plethora of methods for the analysis of scRNA-seq data. Keeping up with the current state of scRNA-seq analysis is now a significant challenge as the field is presented with a huge number of choices for analysing a dataset. Since September 2016 we have collated and categorised scRNA-seq analysis tools as they have become available. This database is being continually updated and is publicly available at www.scRNA-tools.org. In order to help researchers navigate the vast ocean of analysis tools we categorise tools in the database in the context of the typical phases of an scRNA-seq analysis. Through the analysis of this database we show trends in not only the analysis applications these methods address but how they are published and licensed, and the platforms they use. Based on this database we gain insight into the current state of current tools in this rapidly developing field.

Design and implementation

Database

The scRNA-tools database contains information on software tools specifically designed for the analysis of scRNA-seq data. For a tool to be eligible for inclusion in the database it must be available for download and public use. This can be from a software package repository (such as Bioconductor [4], CRAN or PyPI), a code sharing website such as GitHub or directly from a private website. When new tools come to our attention they are added to the scRNA-tools database. DOIs and publication dates are recorded for any associated publications. As preprints may be frequently updated they are marked as a preprint instead of recording a date. The platform used to build the tool, links to code repositories, associated licenses and a short description are also recorded. Each tool is categorised according to the analysis tasks it can perform, receiving a true or false for each category based on what is described in the accompanying paper or



documentation. We also record the date that each entry was added to the database and the date that it was last updated. Most tools are added after a preprint or publication becomes available but some have been added after being mentioned on social media or in similar collections such as Sean Davis' awesome-single-cell page (<https://github.com/seandavi/awesome-single-cell>).

Website

To build the website we start with the table described above as a CSV file which is processed using an R script. The lists of packages available in the CRAN, Bioconductor, PyPI and Anaconda software repositories are downloaded and matched with tools in the database. For tools with associated publications the number of citations they have received is retrieved from the Crossref database (www.crossref.org) using the rcrossref package (v0.8.0) [5]. We also make use of the arXiv package (v0.5.16) [6] to retrieve information about arXiv preprints. JSON files describing the complete table, tools and categories are produced and used to populate the website.

The website consists of three main pages. The home page shows an interactive table with the ability to sort, filter and download the database. The second page shows an entry for each tool, giving the description, details of publications, details of the software code and license and the associated software categories. Badges are added to tools to provide clearly visible details of any associated software or GitHub repositories. The final page describes the categories, providing easy access to the tools associated with them. Both the tools and categories pages can be sorted in a variety of ways, including by the number of associated publications or citations. An additional page shows a live and up-to-date version of some of the analysis presented here with visualisations produced using ggplot2 (v2.2.1.9000) [7] and plotly (v4.7.1) [8]. We welcome contributions to the database from the wider community via submitting an issue to the project GitHub page (<https://github.com/Oshlack/scRNA-tools>) or by filling in the submission form on the scRNA-tools website.

Analysis

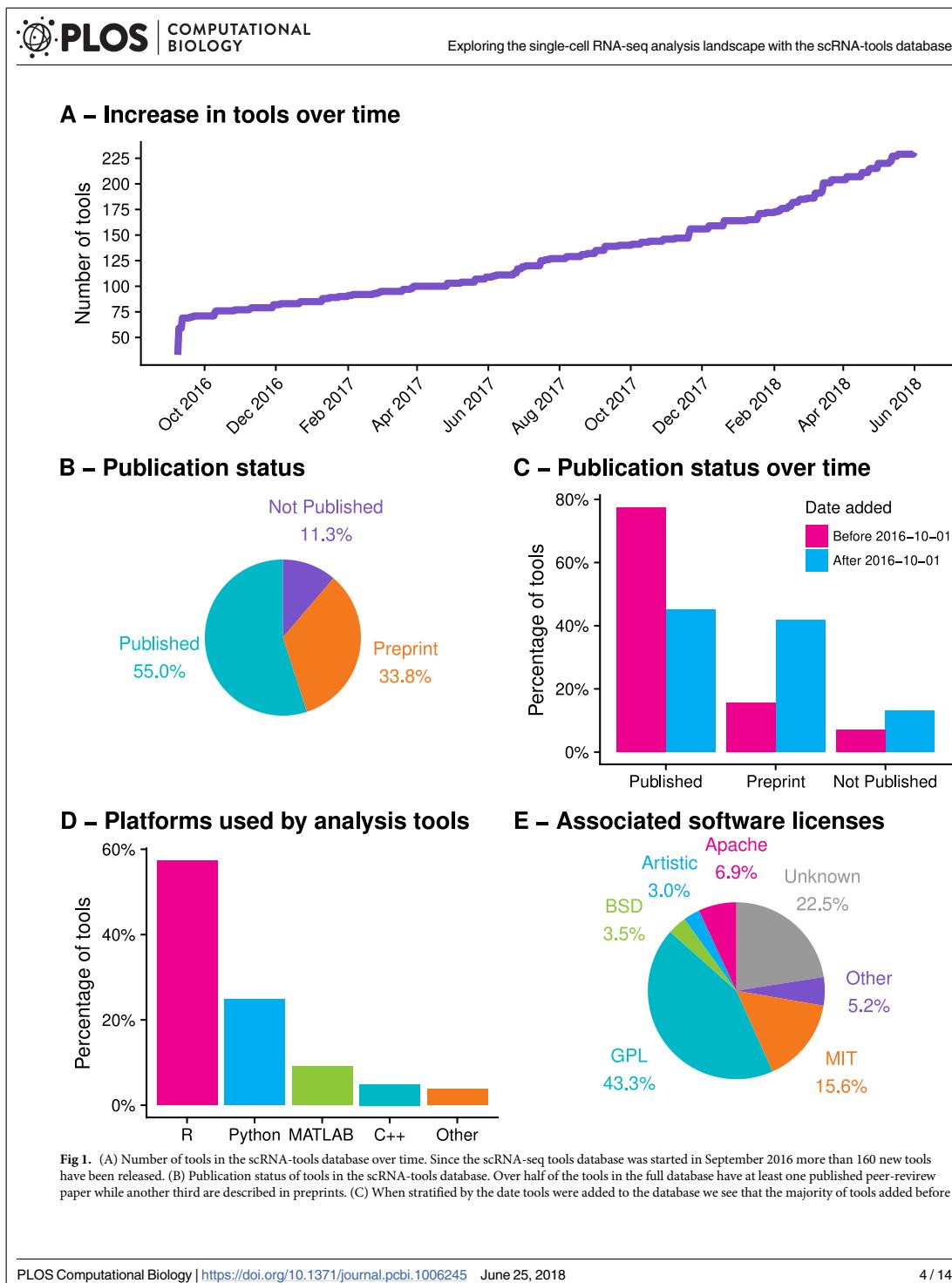
The most recent version of the scRNA-tools database as of 6 June 2018 was used for the analysis presented in this paper. Data was manipulated in R (v3.5.0) using the dplyr package (v0.7.5) [9] and plots produced using the ggplot2 (v2.2.1.9000) and cowplot (v0.9.2) [10] packages.

Results

Overview of the scRNA-tools database

When the database was first constructed it contained 70 scRNA-seq analysis tools representing the majority of work in the field during the three years from the publication of SAMstrt [11] in November 2013 up to September 2016. In the time since then over 160 new tools have been added (Fig 1A). The almost tripling of the number of available tools in such a short time demonstrates the booming interest in scRNA-seq and its maturation from a technique requiring custom-built equipment with specialised protocols to a commercially available product.

Publication status. Most tools have been added to the scRNA-tools database after coming to our attention in a publication or preprint describing their method and use. Of all the tools in the database about half have at least one publication in a peer-reviewed journal and another third are described in preprint articles, typically on the bioRxiv preprint server (Fig 1B). Tools can be split into those that were available when the database was created and those that have been added since. We can see that the majority of older tools have been published while more recent tools are more likely to only be available as preprints (Fig 1C). This is a good



PLOS COMPUTATIONAL BIOLOGY

Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database

October 2016 are published, while around half of newer tools are available only as preprints. Newer tools are also more likely to be unpublished in any form. (D) The majority of tools are available using either the R or Python programming languages. (E) Most tools are released under a standard open-source software license, with variants of the GNU Public License (GPL) being the most common. However licenses could not be found for a large proportion of tools. Up-to-date versions of these plots (with the exception of C) are available on the analysis page of the scRNA-tools website (<https://www.scRNA-tools.org/analysis>).

<https://doi.org/10.1371/journal.pcbi.1006245.g001>

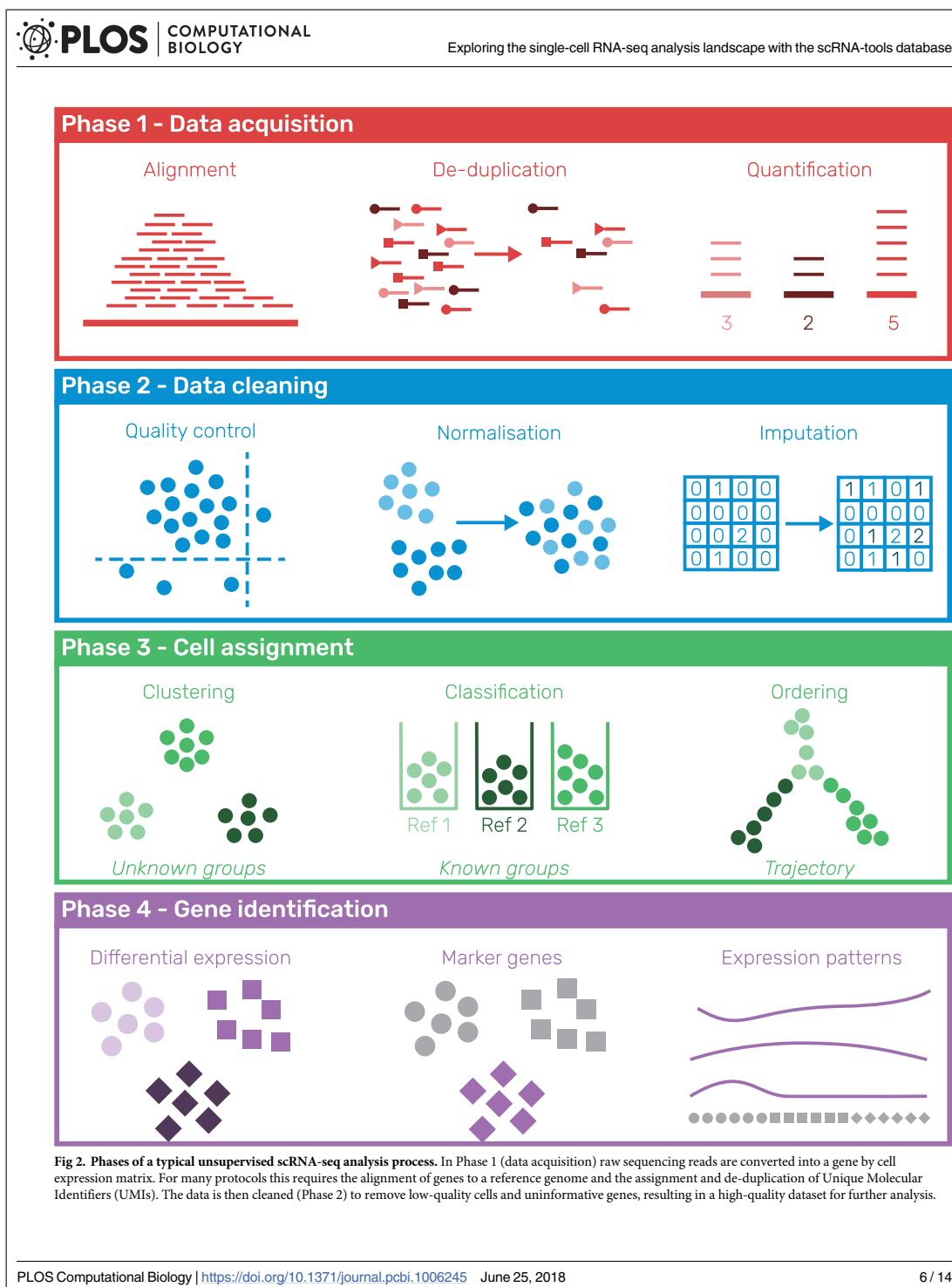
demonstration of the delay imposed by the traditional publication process. By publishing pre-prints and releasing software via repositories such as GitHub, scRNA-seq tool developers make their tools available to the community much earlier, allowing them to be used for analysis and their methods improved prior to formal publication [12].

Platforms and licensing. Developers of scRNA-seq analysis tools have choices to make about what platforms they use to create their tools, how they make them available to the community and whether they share the source code. We find that the most commonly used platform for creating scRNA-seq analysis tools is the R statistical programming language, with many tools made available through the Bioconductor or CRAN repositories (Fig 1D). Python is the second most popular language, followed by MATLAB, a proprietary programming language, and the lower-level C++. The use of R and Python is consistent with their popularity across a range of data science fields. In particular the popularity of R reflects its history as the language of choice for the analysis of bulk RNA-seq datasets and a range of other biological data types.

The majority of tools in the scRNA-tools database have been developed with an open-source approach, making their code available under permissive software licenses (Fig 1E). We feel this reflects the general underlying sentiment and willingness of the bioinformatics community to share and build upon the work of others. Variations of the GNU Public License (GPL) are the most common, covering almost half of tools. This license allows free use, modification and distribution of source code, but also has a “copyleft” nature which requires any derivatives to disclose their source code and use the same license. The MIT license is the second most popular which also allows use of code for any purpose but without any restrictions on distribution or licensing. The appropriate license could not be identified for almost a quarter of tools. This is problematic as fellow developers must assume that source code cannot be reused, potentially limiting the usefulness of the methods in those tools. We strongly encourage tool developers to clearly display their license in source code and documentation to provide certainty to the community as to any restrictions on the use of their work.

Categories of scRNA-seq analysis

Single-cell RNA-sequencing is often used to explore complex mixtures of cell types in an unsupervised manner. As has been described in previous reviews a standard scRNA-seq analysis in this setting consists of several tasks which can be completed using various tools [13–17]. In the scRNA-tools database we categorise tools based on the analysis tasks they perform. Here we group these tasks into four broad phases of analysis: data acquisition, data cleaning, cell assignment and gene identification (Fig 2). The data acquisition phase (Phase 1) takes the raw nucleotide sequences from the sequencing experiment and returns a matrix describing the expression of each gene in each cell. This phase consists of tasks common to bulk RNA-seq experiments, such as alignment to a reference genome or transcriptome and quantification of expression, but is often extended to handle Unique Molecular Identifiers (UMIs) [18]. Once an expression matrix has been obtained it is vital to make sure the resulting data is of high enough quality. In the data cleaning phase (Phase 2) quality control of cells is performed as well as filtering of uninformative genes. Additional tasks may be performed to normalise the



PLOS COMPUTATIONAL BIOLOGY

Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database

The data can also be normalised and missing values imputed during this phase. Phase 3 assigns cells, either in a discrete manner to known (classification) or unknown (clustering) groups or to a position on a continuous trajectory. Interesting genes (eg. differentially expressed, markers, specific patterns of expression) are then identified to explain these groups or trajectories (Phase 4).

<https://doi.org/10.1371/journal.pcbi.1006245.g002>

data or impute missing values. Exploratory data analysis tasks are often performed in this phase, such as viewing the datasets in reduced dimensions to look for underlying structure.

The high-quality expression matrix is the focus of the next phases of analysis. In Phase 3 cells are assigned, either to discrete groups via clustering or along a continuous trajectory from one cell type to another. As high-quality reference datasets become available it will also become feasible to classify cells directly into different cell types. Once cells have been assigned the focus of analysis turns to interpreting what those assignments mean. Identifying interesting genes (Phase 4), such as those that are differentially expressed across groups, marker genes expressed in a single group or genes that change expression along a trajectory, is the typical way to do this. The biological significance of those genes can then be interpreted to give meaning to the experiment, either by investigating the genes themselves or by getting a higher-level view through techniques such as gene set testing.

While there are other approaches that could be taken to analyse scRNA-seq data these phases represent the most common path from raw sequencing reads to biological insight applicable to many studies. An exception to this may be experiments designed to test a specific hypothesis where cell populations may have been sorted or the interest lies in differences between experimental conditions rather than cell types. In this case Phase 3 may not be required, and slightly different tools or approaches may be used, but many of the same challenges will apply. In addition, as the field expands and develops it is likely that data will be used in new ways to answer other biological questions, requiring new analysis techniques. Descriptions of the categories in the scRNA-tools database are given in [Table 1](#), along with the associated analysis phases.

Trends in scRNA-seq analysis tasks. Each of the tools in the database is assigned to one or more analysis categories. We investigated these categories in further detail to give insight into the trends in scRNA-seq analysis. [Fig 3A](#) shows the frequency of tools performing each of the analysis tasks. Visualisation is the most commonly included task and is important across all stages of analysis for exploring and displaying data and results. Tasks for assigning cells (ordering and clustering) are the next most common. This has been the biggest area of development in single-cell analysis with clustering tools such as Seurat [\[19,20\]](#), SC3 [\[21\]](#) and Back-SPIN [\[22\]](#) being used to identify cell types in a sample and trajectory analysis tools (for example Monocle [\[23–25\]](#), Wishbone [\[26\]](#) and DPT [\[27\]](#)) being used to investigate how genes change across developmental processes. These areas reflect the new opportunities for analysis provided by single-cell data that are not possible with bulk RNA-seq experiments.

Dimensionality reduction is also a common task and has applications in visualisation (via techniques such as t-SNE [\[28\]](#)), quality control and as a starting point for analysis. Testing for differential expression (DE) is perhaps the most common analysis performed on bulk RNA-seq datasets and it is also commonly applied by many scRNA-seq analysis tools, typically to identify genes that are different in one group of cells compared to the rest. However it should be noted that the DE testing applied by scRNA-seq tools is often not as sophisticated as the rigorous statistical frameworks of tools developed for bulk RNA-seq such as edgeR [\[29,30\]](#), DESeq2 [\[31\]](#) and limma [\[32\]](#), often using simple statistical tests such as the likelihood ratio test. While methods designed to test DE specifically in single-cell datasets do exist (such as SCDE [\[33\]](#), and scDD [\[34\]](#)) it is still unclear whether they improve on methods that have been established for bulk data [\[35–37\]](#), with the most comprehensive comparison to date finding

PLOS | COMPUTATIONAL
BIOLOGY

Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database

Table 1. Descriptions of categories for tools in the scRNA-tools database.

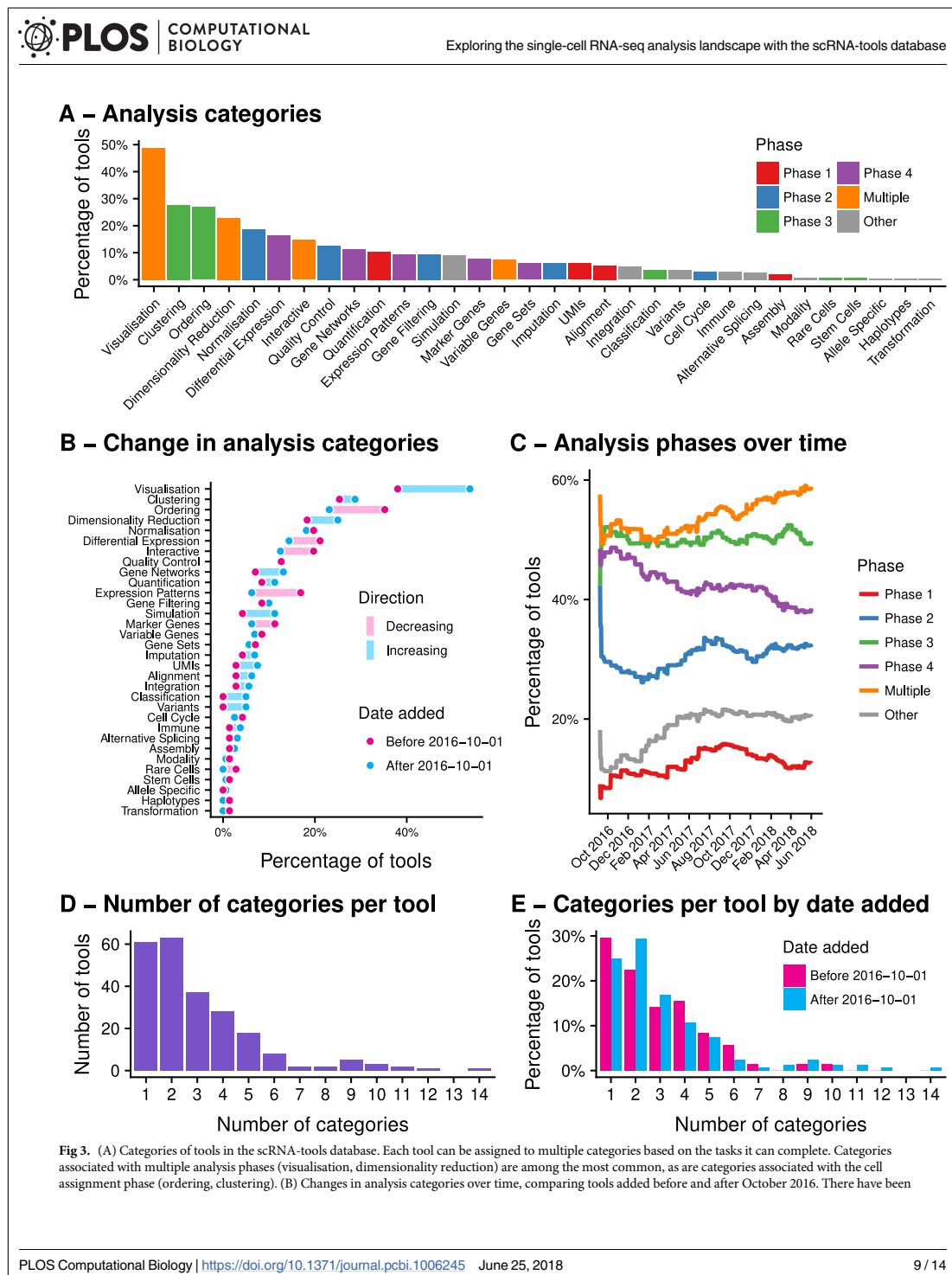
Phase	Category	Description
Phase 1	Alignment	Alignment of sequencing reads to a reference
Phase 1	Assembly	Tools that perform assembly of scRNA-seq reads
Phase 1	UMIs	Processing of Unique Molecular Identifiers
Phase 1	Quantification	Quantification of expression from reads
Phase 2	Quality Control	Removal of low-quality cells
Phase 2	Gene Filtering	Removal of lowly expressed or otherwise uninformative genes
Phase 2	Imputation	Estimation of expression where zeros have been observed
Phase 2	Normalisation	Removal of unwanted variation that may affect results
Phase 2	Cell Cycle	Assignment or correction of stages of the cell cycle, or other uses of cell cycle genes, or genes associated with similar processes
Phase 3	Classification	Assignment of cell types based on a reference dataset
Phase 3	Clustering	Unsupervised grouping of cells based on expression profiles
Phase 3	Ordering	Ordering of cells along a trajectory
Phase 3	Rare Cells	Identification of rare cell populations
Phase 3	Stem Cells	Identification of cells with stem-like characteristics
Phase 4	Differential Expression	Testing of differential expression across groups of cells
Phase 4	Expression Patterns	Detection of genes that change expression across a trajectory
Phase 4	Gene Networks	Identification or use of co-regulated gene networks
Phase 4	Gene Sets	Testing for over representation or other uses of annotated gene sets
Phase 4	Marker Genes	Identification or use of genes that mark cell populations
Multiple	Dimensionality Reduction	Projection of cells into a lower dimensional space
Multiple	Interactive	Tools with an interactive component or a graphical user interface
Multiple	Variable Genes	Identification or use of highly (or lowly) variable genes
Multiple	Visualisation	Functions for visualising some aspect of scRNA-seq data or analysis
Other	Allele Specific	Detection of allele-specific expression
Other	Alternative Splicing	Detection of alternative splicing
Other	Haplotypes	Use or assignment of haplotypes
Other	Immune	Assignment of receptor sequences and immune cell clonality
Other	Integration	Combining of scRNA-seq datasets or integration with other single-cell data types
Other	Modality	Identification or use of modality in gene expression
Other	Simulation	Generation of synthetic scRNA-seq datasets
Other	Transformation	Transformation between expression levels and some other measure
Other	Variants	Detection or use of nucleotide variants

<https://doi.org/10.1371/journal.pcbi.1006245.t001>

that bulk methods do not perform significantly worse than those designed for scRNA-seq data [38].

To investigate how the focus of scRNA-seq tool development has changed over time we again divided the scRNA-tools database into tools added before and after October 2016. This allowed us to see which analysis tasks are more common in recently released tools. We looked at the percentage of tools in each time period that performed tasks in the different analysis categories (Fig 3B). Some categories show little change in the proportion of tools that perform them while other areas have changed significantly. Specifically, both visualisation and dimensionality reduction are more commonly addressed by recent tools. The UMIs category has also seen a big increase recently as UMI based protocols have become commonly used and tools designed to handle the extra processing steps required have been developed (e.g. UMI-

PLOS Computational Biology | <https://doi.org/10.1371/journal.pcbi.1006245> June 25, 2018 8 / 14



PLOS | COMPUTATIONAL
BIOLOGY

Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database

significant increases in the percentage of tools associated with visualisation, dimensionality reduction, gene networks and simulation. Categories including expression patterns, ordering and interactivity have seen relative decreases. (C) Changes in the percentage of tools associated with analysis phases over time. The percentage of tools involved in the data acquisition and data cleaning phases have increased, as have tools designed for alternative analysis tasks. The gene identification phase has seen a relative decrease in the number of tools. (D) The number of categories associated with each tools in the scRNA-tools database. The majority of tools perform few tasks. (E) Most tools that complete many tasks are relatively recent.

<https://doi.org/10.1371/journal.pcbi.1006245.g003>

tools [39], umis [40], zUMIs [41]). Simulation is a valuable technique for developing, testing and validating scRNA-seq tools. More packages are now including their simulation functions and some tools have been developed for the specific purpose of generating realistic synthetic scRNA-seq datasets (e.g. powsimR [42], Splatter [43]). Classification of cells into known groups has also increased as reference datasets become available and more tools are identifying or making use of co-regulated gene networks.

Some categories have seen a decrease in the proportion of tools they represent, most strikingly testing for expression patterns along a trajectory. This is likely related to the change in cell ordering analysis, which is the focus of a lower percentage of tools added after October 2016. The ordering of cells along a trajectory was one of the first developments in scRNA-seq analysis and a decrease in the development of these tools could indicate that researchers have moved on to other techniques or that use has converged on a set of mature tools.

By grouping categories based on their associated analysis phases we see similar trends over time (Fig 3C). We see increases in the percentage of tools performing tasks in Phase 1 (quantification), across multiple phases (such as visualisation and dimensionality reduction) and alternative analysis tasks. In contrast the percentage of tools that perform gene identification tasks (Phase 4) has decreased and the percentage assigning cells (Phase 3) has remained steady. Phase 2 (quality control and filtering) has fluctuated over time but currently sits at a level slightly above when the database was first created. This also indicates a maturation of the analysis space as developers shift away from the tasks that were the focus of bulk RNA-seq analysis and continue to focus on those specific to scRNA-seq while working on methods for handling data from new protocols and performing alternative analysis tasks.

Pipelines and toolboxes. While there are a considerable number of scRNA-seq tools that only perform a single analysis task, many perform at least two (Fig 3D). Some tools (dropEst [44], DrSeq2 [45], scPipe [46]) are pre-processing pipelines, taking raw sequencing reads and producing an expression matrix. Others, such as Scampy [47], SCell [48], Seurat, Monocle and scater [49] can be thought of as analysis toolboxes, able to complete a range of complex analyses starting with a gene expression matrix. Most of the tools that complete many tasks are relatively more recent (Fig 3E). Being able to complete multiple tasks using a single tool can simplify analysis as problems with converting between different data formats can be avoided. However it is important to remember that it is difficult for a tool with many functions to continue to represent the state of the art in all of them. Support for common data formats, such as the recently released SingleCellExperiment [50], annData [47] or loom (<http://loompy.org>) objects provides another way for developers to allow easy use of their tools and for users to build custom workflows from specialised tools.

Alternative analyses. Some tools perform analyses that lie outside the common tasks performed on scRNA-seq data described above. Simulation is one alternative task that has already been mentioned but there is also a group of tools designed to detect biological signals in scRNA-seq data apart from changes in expression. For example identifying alternative splicing (BRIE [51], Outrigger [52], SingleSplice [53]), single nucleotide variants (SSrGE [54]), copy number variants (inferCNV [55]) and allele-specific expression (SCALE [56]). Reconstruction of immune cell receptors is another area that has received considerable attention from tools such as BASIC [57], TraCeR [58] and TRAPeS [59]. While tools that complete these tasks are

PLOS Computational Biology | <https://doi.org/10.1371/journal.pcbi.1006245> June 25, 2018

10 / 14



unlikely to ever dominate scRNA-seq analysis we expect to see an increase in methods for tackling specialised analyses as researchers continue to push the boundaries of what can be observed using scRNA-seq data.

Availability and future directions

Since October 2016 we have seen the number of software tools for analysing single-cell RNA-seq data more than triple, with more than 230 analysis tools now available. As new tools have become available we have curated and catalogued them in the scRNA-tools database where we record the analysis tasks that they can complete, along with additional information such as any associated publications. By analysing this database we have found that tool developers have focused much of their efforts on methods for handling new problems specific to scRNA-seq data, in particular clustering cells into groups or ordering them along a trajectory. We have also seen that the scRNA-seq community is generally open and willing to share their methods which are often described in preprints prior to peer-reviewed publication and released under permissive open-source licenses for other researchers to re-use.

The next few years promise to produce significant new developments in scRNA-seq analysis. New tools will continue to be produced, becoming increasingly sophisticated and aiming to address more of the questions made possible by scRNA-seq data. We anticipate that some existing tools will continue to improve and expand their functionality while others will cease to be updated and maintained. Detailed benchmarking and comparisons will show how tools perform in different situations and those that perform well, continue to be developed and provide a good user experience will become preferred for standard analyses. As single-cell capture and sequencing technology continues to improve analysis tools will have to adapt to significantly larger datasets (in the millions of cells) which may require specialised data structures and algorithms. Methods for combining multiple scRNA-seq datasets as well as integration of scRNA-seq data with other single-cell data types, such as DNA-seq, ATAC-seq or methylation, will be another area of growth. In addition, projects such as the Human Cell Atlas [60] will provide comprehensive cell type references which will open up new avenues for analysis.

As the field expands the scRNA-tools database will continue to be updated with support from the community. We hope that it provides a resource for researchers to explore when approaching scRNA-seq analyses as well as providing a record of the analysis landscape and how it changes over time.

Availability

The scRNA-tools databases is publicly accessible via the website at www.scRNA-tools.org. Suggestions for additions, updates and improvements are warmly welcomed at the associated GitHub repository (<https://github.com/Oshlack/scRNA-tools>) or via the submission form on the website. The code and datasets used for the analysis in this paper are available from <https://github.com/Oshlack/scRNAtools-paper>.

Acknowledgments

We would like to acknowledge Sean Davis' work in managing the awesome-single-cell page and producing a prototype of the script used to process the database. Daniel Wells had the idea for recording software licenses and provided licenses for the tools in the database at that time. Breon Schmidt designed a prototype of the scRNA-tools website and answered many questions about HTML and Javascript. Our thanks also to Matt Ritchie for his thoughts on early versions of the manuscript.

PLOS COMPUTATIONAL BIOLOGY Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database

Author Contributions

Conceptualization: Luke Zappia.

Formal analysis: Luke Zappia.

Investigation: Luke Zappia.

Methodology: Luke Zappia.

Software: Luke Zappia.

Supervision: Belinda Phipson, Alicia Oshlack.

Writing – original draft: Luke Zappia.

Writing – review & editing: Luke Zappia, Belinda Phipson, Alicia Oshlack.

References

1. Tang F, Barbacioru C, Wang Y, Nordman E, Lee C, Xu N, et al. mRNA-Seq whole-transcriptome analysis of a single cell. *Nat Methods*. 2009; 6: 377–382. <https://doi.org/10.1038/nmeth.1315> PMID: 19349980
2. Svensson V, Vento-Tormo R, Teichmann SA. Exponential scaling of single-cell RNA-seq in the past decade. *Nat Protoc*. 2018; 13: 599. <https://doi.org/10.1038/nprot.2017.149> PMID: 29494575
3. Stegle O, Teichmann SA, Marioni JC. Computational and analytical challenges in single-cell transcriptomics. *Nat Rev Genet*. 2015; 16: 133–145. <https://doi.org/10.1038/nrg3833> PMID: 25628217
4. Huber W, Carey VJ, Gentleman R, Anders S, Carlson M, Carvalho BS, et al. Orchestrating high-throughput genomic analysis with Bioconductor. *Nat Methods*. 2015; 12: 115–121. <https://doi.org/10.1038/nmeth.3252> PMID: 25633503
5. Chamberlain S, Boettiger C, Hart T, Ram K. rcrossref: Client for Various ‘CrossRef’ APIs’. 2017. <https://CRAN.R-project.org/package=rcrossref>
6. Ram K, Broman K. aRxiv: Interface to the arXiv API. 2017. <https://CRAN.R-project.org/package=aRxiv>
7. Wickham H. ggplot2: Elegant Graphics for Data Analysis. 2010.
8. Sievert C, Parmer C, Hocking T, Chamberlain S, Ram K, Corvellec M, et al. plotly: Create Interactive Web Graphics via ‘plotly.js’. 2017. <https://CRAN.R-project.org/package=plotly>
9. Wickham H, Francois R, Henry L, Müller K. dplyr: A Grammar of Data Manipulation. 2017. <https://CRAN.R-project.org/package=dplyr>
10. Wilke CO. cowplot: Streamlined Plot Theme and Plot Annotations for ‘ggplot2’. 2017. <https://CRAN.R-project.org/package=cowplot>
11. Katayama S, Töhönen V, Linnarsson S, Kere J. SAMstr: statistical test for differential expression in single-cell transcriptome with spike-in normalization. *Bioinformatics*. 2013; 29: 2943–2945. <https://doi.org/10.1093/bioinformatics/btt511> PMID: 23995393
12. Bourne PE, Polka JK, Vale RD, Kiley R. Ten simple rules to consider regarding preprint submission. *PLoS Comput Biol*. 2017; 13: e1005473. <https://doi.org/10.1371/journal.pcbi.1005473> PMID: 28472041
13. Bacher R, Kendziora C. Design and computational analysis of single-cell RNA-sequencing experiments. *Genome Biol*. 2016; 17: 63. <https://doi.org/10.1186/s13059-016-0927-y> PMID: 27052890
14. Wagner A, Regev A, Yosef N. Revealing the vectors of cellular identity with single-cell genomics. *Nat Biotechnol*. 2016; 34: 1145–1160. <https://doi.org/10.1038/nbt.3711> PMID: 27824854
15. Miragaia RJ, Teichmann SA, Hagai T. Single-cell insights into transcriptomic diversity in immunity. *Curr Opin in Systems Biology*. 2017; 5: 63–71. <https://doi.org/10.1016/j.coisb.2017.08.003>
16. Poirion OB, Zhu X, Ching T, Garmire L. Single-cell transcriptomics bioinformatics and computational challenges. *Front Genet*. 2016; 7. <https://doi.org/10.3389/fgene.2016.00163> PMID: 27708664
17. Rostom R, Svensson V, Teichmann SA, Kar G. Computational approaches for interpreting scRNA-seq data. *FEBS Lett*. 2017; <https://doi.org/10.1002/1873-3468.12684> PMID: 28524227
18. Kivioja T, Vähärautio A, Karlsson K, Bonke M, Enge M, Linnarsson S, et al. Counting absolute numbers of molecules using unique molecular identifiers. *Nat Methods*. 2012; 9: 72–74. <https://doi.org/10.1038/nmeth.1778> PMID: 22101854
19. Satija R, Farrell JA, Gennert D, Schier AF, Regev A. Spatial reconstruction of single-cell gene expression data. *Nat Biotechnol*. 2015; 33: 495–502. <https://doi.org/10.1038/nbt.3192> PMID: 25867923

20. Butler A, Hoffman P, Smibert P, Papalexi E, Satija R. Integrating single-cell transcriptomic data across different conditions, technologies, and species. *Nat Biotechnol.* 2018; <https://doi.org/10.1038/nbt.4096> PMID: 29608179
21. Kiselev VY, Kirschner K, Schaub MT, Andrews T, Yiu A, Chandra T, et al. SC3: consensus clustering of single-cell RNA-seq data. *Nat Methods.* 2017; 14: 483–486. <https://doi.org/10.1038/nmeth.4236> PMID: 28346451
22. Zeisel A, Muñoz-Manchado AB, Codeluppi S, Lönnerberg P, La Manno G, Juréus A, et al. Brain structure. Cell types in the mouse cortex and hippocampus revealed by single-cell RNA-seq. *Science.* 2015; 347: 1138–1142. <https://doi.org/10.1126/science.aaa1934> PMID: 25700174
23. Trapnell C, Cacchiarelli D, Grimsby J, Pokharel P, Li S, Morse M, et al. The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells. *Nat Biotechnol.* 2014; 32: 381–386. <https://doi.org/10.1038/nbt.2859> PMID: 24658644
24. Qiu X, Hill A, Packer J, Lin D, Ma Y-A, Trapnell C. Single-cell mRNA quantification and differential analysis with Census. *Nat Methods.* 2017; <https://doi.org/10.1038/nmeth.4150> PMID: 28114287
25. Qiu X, Mao Q, Tang Y, Wang L, Chawla R, Pliner HA, et al. Reversed graph embedding resolves complex single-cell trajectories. *Nat Methods.* 2017; <https://doi.org/10.1038/nmeth.4402> PMID: 28825705
26. Setty M, Tadmor MD, Reich-Zeliger S, Angel O, Salame TM, Kathail P, et al. Wishbone identifies bifurcating developmental trajectories from single-cell data. *Nat Biotechnol.* 2016; 34: 637–645. <https://doi.org/10.1038/nbt.3569> PMID: 27136076
27. Haghverdi L, Büttner M, Wolf FA, Buettner F, Theis FJ. Diffusion pseudotime robustly reconstructs lineage branching. *Nat Methods.* 2016; <https://doi.org/10.1038/nmeth.3971> PMID: 27571553
28. van der Maaten L, Hinton G. Visualizing Data using t-SNE. *J Mach Learn Res.* 2008; 9: 2579–2605. Available: <http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf>
29. Robinson MD, McCarthy DJ, Smyth GK. edgeR: A Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics.* 2010; 26: 139–140. <https://doi.org/10.1093/bioinformatics/btp616> PMID: 19910308
30. McCarthy DJ, Chen Y, Smyth GK. Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Res.* 2012; 40: 4288–4297. <https://doi.org/10.1093/nar/gks042> PMID: 22287627
31. Love MI, Huber W, Anders S. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biol.* 2014; 15: 550. <https://doi.org/10.1186/s13059-014-0550-8> PMID: 25516281
32. Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, et al. limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Res.* 2015; 43: e47. <https://doi.org/10.1093/nar/gkv007> PMID: 25605792
33. Kharchenko PV, Silberstein L, Scadden DT. Bayesian approach to single-cell differential expression analysis. *Nat Methods.* 2014; 11: 740–742. <https://doi.org/10.1038/nmeth.2967> PMID: 24836921
34. Korthauer KD, Chu L-F, Newton MA, Li Y, Thomson J, Stewart R, et al. A statistical approach for identifying differential distributions in single-cell RNA-seq experiments. *Genome Biol.* 2016; 17: 222. <https://doi.org/10.1186/s13059-016-1077-y> PMID: 27782827
35. Jaakkola MK, Seyednasrullah F, Mahmood A, Elo LL. Comparison of methods to detect differentially expressed genes between single-cell populations. *Brief Bioinform.* 2016; <https://doi.org/10.1093/bib/bbw057> PMID: 27373736
36. Miao Z, Zhang X. Differential expression analyses for single-cell RNA-Seq: old questions on new data. *Quant Biol.* 2016; 4: 243–260. <https://doi.org/10.1007/s40484-016-0089-7>
37. Dal Molin A, Baruzzo G, Di Camillo B. Single-cell RNA-sequencing: assessment of differential expression analysis methods. *Front Genet.* 2017; 8: 62. <https://doi.org/10.3389/fgene.2017.00062> PMID: 28588607
38. Soneson C, Robinson MD. Bias, robustness and scalability in single-cell differential expression analysis. *Nat Methods.* 2018; <https://doi.org/10.1038/nmeth.4612> PMID: 29481549
39. Smith T, Heger A, Sudbery I. UMI-tools: modeling sequencing errors in Unique Molecular Identifiers to improve quantification accuracy. *Genome Res.* 2017; 27: 491–499. <https://doi.org/10.1101/gr.209601.116> PMID: 28100584
40. Svensson V, Natarajan KN, Ly L-H, Miragaia RJ, Labalette C, Macaulay IC, et al. Power analysis of single-cell RNA-sequencing experiments. *Nat Methods.* 2017; <https://doi.org/10.1038/nmeth.4220> PMID: 28263961
41. Parekh S, Ziegenhain C, Vieth B, Enard W, Hellmann I. zUMIs—A fast and flexible pipeline to process RNA sequencing data with UMIs. *Gigascience.* 2018; <https://doi.org/10.1093/gigascience/giy059> PMID: 29846586

PLOS COMPUTATIONAL BIOLOGY Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database

42. Vieith B, Ziegenhain C, Parekh S, Enard W, Hellmann I. powsimR: power analysis for bulk and single cell RNA-seq experiments. *Bioinformatics*. 2017; 33: 3486–3488. <https://doi.org/10.1093/bioinformatics/btx435> PMID: 29036287

43. Zappia L, Phipson B, Oshlack A. Splatter: simulation of single-cell RNA sequencing data. *Genome Biol.* 2017; 18: 174. <https://doi.org/10.1186/s13059-017-1305-0> PMID: 28899397

44. Petukhov V, Guo J, Baryawno N, Severe N, Scadden D, Kharchenko PV. Accurate estimation of molecular counts in droplet-based single-cell RNA-seq experiments. *bioRxiv*. 2017. p. 171496. 10.1101/171496

45. Zhao C, Hu S, Huo X, Zhang Y. Dr.seq2: A quality control and analysis pipeline for parallel single cell transcriptome and epigenome data. *PLoS One*. 2017; 12: e0180583. <https://doi.org/10.1371/journal.pone.0180583> PMID: 28671995

46. Tian L, Su S, Armann-Zalcenstein D, Biben C, Naik SH, Ritchie ME. scPipe: a flexible data preprocessing pipeline for single-cell RNA-sequencing data. *bioRxiv*. 2017. p. 175927. 10.1101/175927

47. Wolf FA, Angerer P, Theis FJ. SCANPY: large-scale single-cell gene expression data analysis. *Genome Biol.* 2018; 19: 15. <https://doi.org/10.1186/s13059-017-1382-0> PMID: 29409532

48. Diaz A, Liu SJ, Sandoval C, Pollen A, Nowakowski TJ, Lim DA, et al. SCell: integrated analysis of single-cell RNA-seq data. *Bioinformatics*. 2016; <https://doi.org/10.1093/bioinformatics/btw201> PMID: 27153637

49. McCarthy DJ, Campbell KR, Lun ATL, Wills QF. Scater: pre-processing, quality control, normalization and visualization of single-cell RNA-seq data in R. *Bioinformatics*. 2017; 33: 1179–1186. <https://doi.org/10.1093/bioinformatics/btw777> PMID: 28088763

50. Lun A, Risso D. SingleCellExperiment: S4 Classes for Single Cell Data. 2017.

51. Huang Y, Sanguinetti G. BRIE: transcriptome-wide splicing quantification in single cells. *Genome Biol.* 2017; 18: 123. <https://doi.org/10.1186/s13059-017-1248-5> PMID: 28655331

52. Song Y, Botvinnik OB, Lovci MT, Kakaradov B, Liu P, Xu JL, et al. Single-cell alternative splicing analysis with Expedition reveals splicing dynamics during neuron differentiation. *Mol Cell*. 2017; <https://doi.org/10.1016/j.molcel.2017.06.003> PMID: 28673540

53. Welch JD, Hu Y, Prins JF. Robust detection of alternative splicing in a population of single cells. *Nucleic Acids Res.* 2016; 44: e73. <https://doi.org/10.1093/nar/gkv1525> PMID: 26740580

54. Poirion OB, Zhu X, Ching T, Garmire LX. Using single nucleotide variations in cancer single-cell RNA-seq data for subpopulation identification and genotype-phenotype linkage analysis. *bioRxiv*. 2016. p. 095810. 10.1101/095810

55. Patel AP, Tirosh I, Trombetta JJ, Shalek AK, Gillespie SM, Wakimoto H, et al. Single-cell RNA-seq highlights intratumoral heterogeneity in primary glioblastoma. *Science*. 2014; 344: 1396–1401. <https://doi.org/10.1126/science.1254257> PMID: 24925914

56. Jiang Y, Zhang NR, Li M. SCALE: modeling allele-specific gene expression by single-cell RNA sequencing. *Genome Biol.* 2017; 18: 74. <https://doi.org/10.1186/s13059-017-1200-8> PMID: 28446220

57. Canzar S, Neu KE, Tang Q, Wilson PC, Khan AA. BASIC: BCR assembly from single cells. *Bioinformatics*. 2017; 33: 425–427. <https://doi.org/10.1093/bioinformatics/btw631> PMID: 28172415

58. Stubbington MJT, Lönnberg T, Prosperio V, Clare S, Speak AO, Dougan G, et al. T cell fate and clonality inference from single-cell transcriptomes. *Nat Methods*. 2016; 13: 329–332. <https://doi.org/10.1038/nmeth.3800> PMID: 26950746

59. Afik S, Yates KB, Bi K, Darko S, Godec J, Gerdemann U, et al. Targeted reconstruction of T cell receptor sequence from single cell RNA-seq links CDR3 length to T cell differentiation state. *Nucleic Acids Res.* 2017; <https://doi.org/10.1093/nar/gkw615> PMID: 28934479

60. Regev A, Teichmann S, Lander ES, Amit I, Benoist C, Birney E, et al. The Human Cell Atlas. *bioRxiv*. 2017. p. 121202. 10.1101/121202

2.3 The current scRNA-tools database

Since the paper was published in June 2018, 227 tools have been added to the scRNA-tools database, taking the total to 459. Figure 2.1 shows an overview of the current database as of 1 August 2019. Overall the analysis landscape has not changed significantly over the last few months.

Over half of tools are still associated with a peer-reviewed publication with another third described in preprints (Figure 2.1B). Versions of the GNU Public License (GPL) continue to be the most common license for scRNA-seq software tools, followed by the more permissive MIT license (Figure 2.1C). The use of preprints to quickly share methods and observations and open-source licenses which allow code to be reused and adapted reflects the general willingness to share that I have observed in the scRNA-seq bioinformatics community during my PhD. Unfortunately, around a quarter of tools do not have associated licenses, which can limit their usefulness for other developers. I believe that in most cases this is not an attempt by authors to restrict how their work is used, but a lack of awareness about the importance of clear licensing. By displaying licenses on the scRNA-tools website I hope to encourage developers to consider how they license their work in the future.

R continues to be the most popular programming platform and is used in some way by around two thirds of scRNA-seq analysis tools (Figure 2.1D). Around a third of tools are built using Python which is the next most common programming language, followed by C++ and MATLAB which are each used by just under 10 percent of tools. The dominance of R reflects the statistical nature of many of the tasks involved in analysing scRNA-seq data as well as the historical usage of the platform for bulk RNA-seq and other genomic data types. The Bioconductor project [199] has been integral to promoting the use of R for analysing genomic data and already includes a range of tools for scRNA-seq analysis [200]. However, as the size of scRNA-seq datasets continues to increase and approaches to analysis change to include more machine learning methods I expect the platforms used to change to some degree. In particular we will see more use of compiled languages like C++, Swift, Julia and Go used for their improved performance. While MATLAB is commonly used within a small section of the academic community, it is proprietary software that requires a paid license and tools developed in MATLAB cannot be used by the majority of scRNA-seq analysts.

The analysis tasks that tools perform reflect the advantages and uses of scRNA-seq data (Figure 2.1E). The most common categories include those that are associated with multiple phases of analysis, particularly visualisation, for displaying results and checking distributions, and dimensionality reduction. The dimensionality reduction category includes tools that make use of a lower dimensional space as part of their analysis approach, as well as several techniques that have been specifically developed for projecting scRNA-seq data. Clustering and ordering of cells, which are tasks specific to scRNA-seq analysis, are also common categories. I expect these tasks to continue to be included in new software tools as they are important steps in scRNA-seq analysis, however I hope that the focus turns from developing new clustering and ordering methods to identifying those that are consistently effective and developing robust, easy to use and well-documented software tools that implement them. The availability of reliable and complete reference datasets should lead to the development of more that make use of references such as the classification of cells, and developers will continue to push the boundaries of what is possible using scRNA-seq data.

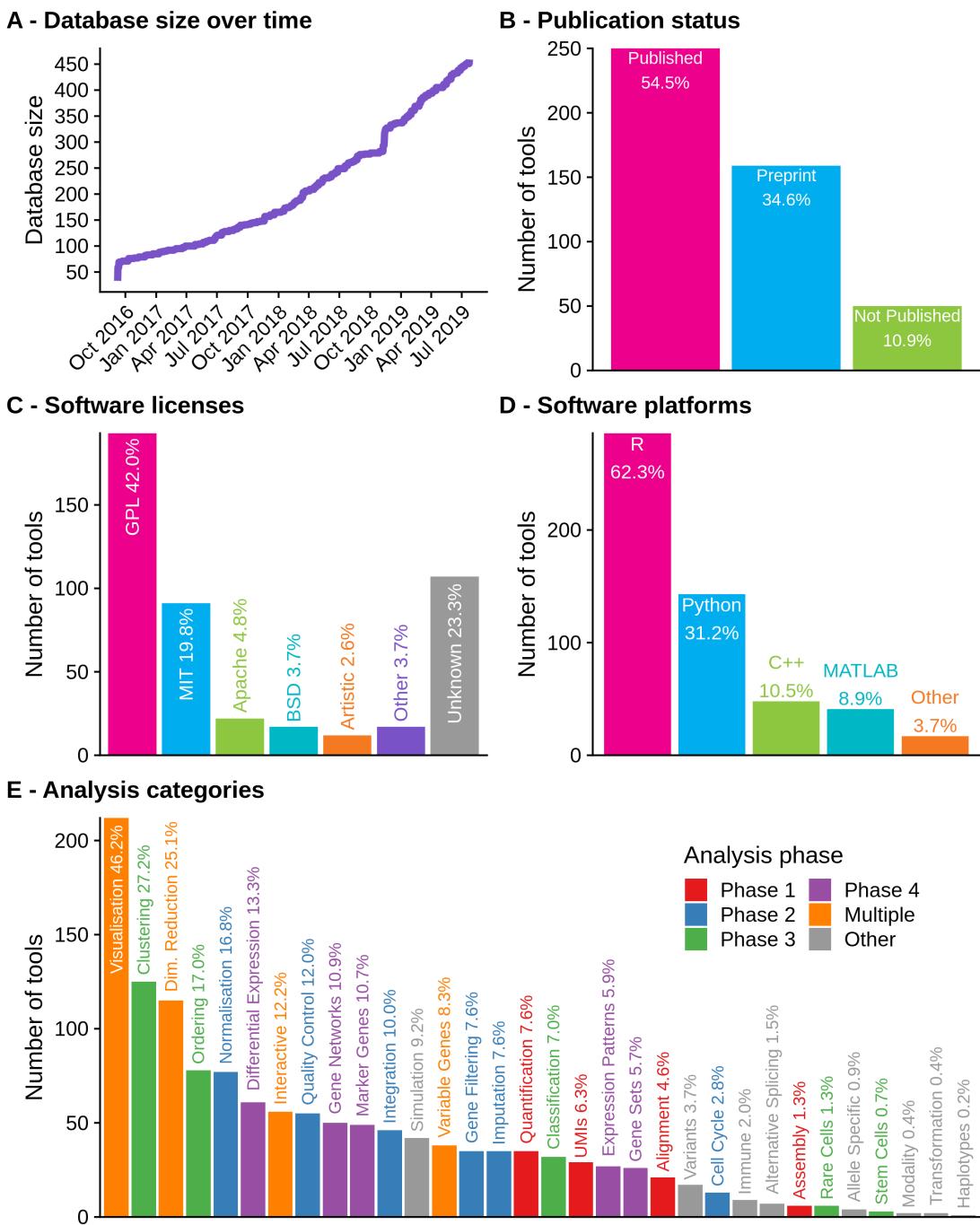


Figure 2.1: Current state of the scRNA-tools database. (A) Size of the database over time. (B) Publication status of tools in the database. Most tools are published as either peer-reviewed publications or preprints. (C) Associated software licenses. Most tools are covered by open-source licenses but around a quarter of tools still have no license. (D) Software platforms used by tools, some tools use more than one language. R is the most common programming language followed by Python. (E) Number of tools that complete each analysis task. Tasks associated with multiple analysis phases are the most common, along with those that assign cells.

2.4 Usage of the scRNA-tools website

Since it was launched in July 2017 the scRNA-tools website has been used by people from around the world (Figure 2.2). Usage has increased since release of the publication and the website now receives an average of over 1000 visitors per month (Figure 2.2A). These users come from over 90 countries around the world (Figure 2.2B) with the USA being by far the most common location, followed by China, Japan, Australia, the United Kingdom and countries in Western Europe (Figure 2.2C). Aggregating by continent (Figure 2.2D) better reflects population size and the size of academic communities, with the Americas responsible for the majority of traffic followed by Europe and Asia.

Although these figures may not seem large when compared to an average website, they represent a significant part of the scRNA-seq community. Seeing usage of the website from so many countries sustained over this period of time suggests that the database is a useful resource. I intend to continue maintain and grow the resource by adding new tools and updating those in the database. I hope to encourage contributions from the community who can easily submit changes via a form on the website.

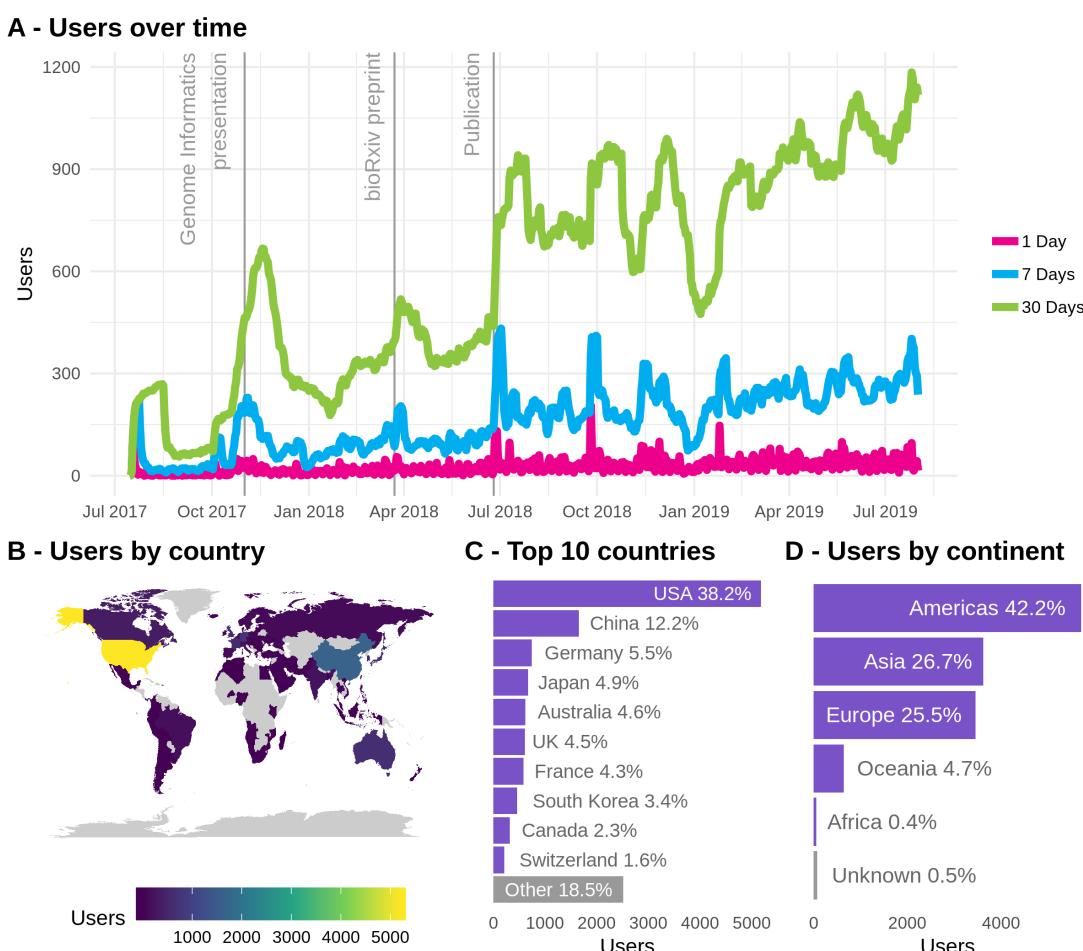


Figure 2.2: Usage of the scRNA-tools website. (A) Usage over time showing the number of users per day (pink), week (blue) or 30 days (green). Labelled vertical lines indicate events that may have influenced traffic. (B) World map coloured according to number of users from low (blue) to high (yellow). (C) Number of users from the top 10 most common countries. (D) Number of users by continent.

3

Simulating scRNA-seq data

“Essentially, all models are wrong, but some are useful.”

— George E. P. Box and Norman R. Draper

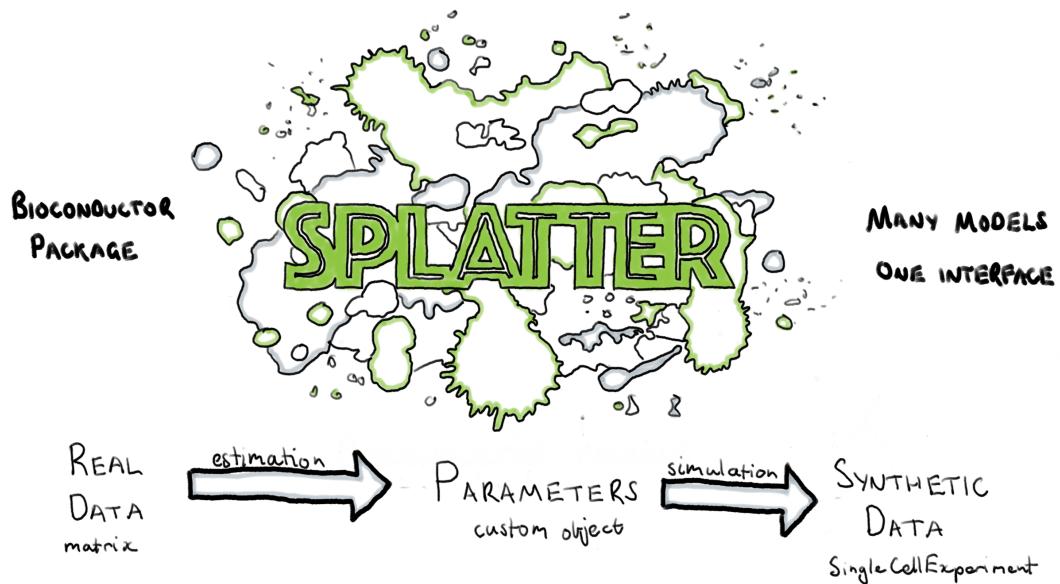
Empirical Model-Building and Response Surfaces, 1987

3. Simulation

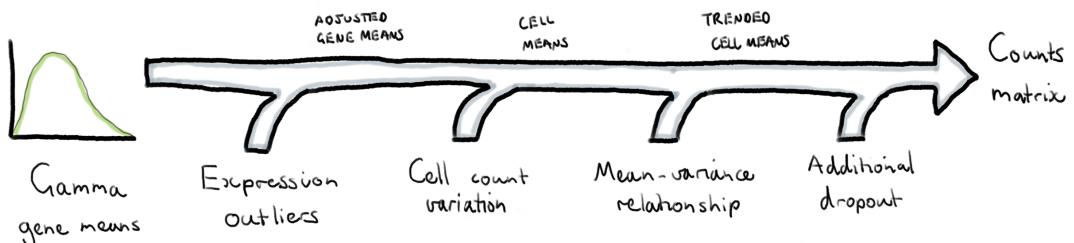
Methods should
show that
they work

How?

SIMULATIONS!



The SPLAT Model



3.1 Introduction

For a computational analysis method to be accepted and used by the community it needs to demonstrate that it is effective at the task it aims to complete. Ideally this can be achieved by evaluating the performance of the method on a real dataset where the results are already known. Unfortunately, in many cases such gold standard datasets are not available. This is particularly true for genomic data where it is difficult to know what the truth is or the known truth is limited to only small sections of the genome. It is possible to create some genomic datasets where the truth is known, for example through carefully performed mixing experiments [175], but these often do not capture the true biological complexity. In many cases the most effective way to evaluate an analysis method during development is by testing it on simulated datasets. Simulations have the additional advantage of being relatively cheap and easy to produce, allowing exploration of a wide range of possible parameters. This is the approach taken by many early methods for scRNA-seq analysis. Unfortunately, these simulations were often not well explained, code for reproducing them was not available and perhaps most importantly they didn't show that the synthetic datasets were similar to real scRNA-seq data.

This chapter presents Splatter, a software package for simulating scRNA-seq datasets presented in a publication in *Genome Biology* [201]. Splatter is designed to provide a consistent, easy-to-use interface for multiple previously developed scRNA-seq simulation models. We do this by providing a consistent framework for each model defined by two functions: one which estimates parameters from a real dataset and a second that generates a synthetic dataset using those parameters. Each simulation model has different assumptions and reproduces different aspects of scRNA-seq data, and we explain these differences in our publication. We also present Splat, our own simulation model based on the Gamma-Poisson distribution. This model includes several important aspects of scRNA-seq data including highly expressed outliers genes, differences in library sizes between cells, a relationship between the mean and the variance of each gene and the ability to add a dropout effect linked to gene expression. When designing the Splat simulation our goal was to accurately reproduce scRNA-seq count data rather than test a specific method, with the result being that the model is highly flexible and able to generate a range of scenarios including datasets with multiple groups of cells, batch effects and continuous trajectories.

In the publication we compare how well each simulation reproduces a range of scRNA-seq datasets, including UMI and full-length protocols, different capture methods, and homogenous and complex tissues. We found that the Splat simulation was a good match for some of these datasets across a range of methods, however it was also clear that some models more faithfully reproduced different aspects of the data, particularly for datasets from different sources. The Splatter R package is available for download from Bioconductor (<https://bioconductor.org/packages/splatter>). Additional files for the publication are provided in **Appendix A**. Following the publication (**Section 3.2**) I discuss updates to the software package and provide an updated comparison of the current simulation models (**Section 3.3**). Vignettes describing the use of Splatter (**Appendix B.1**) and the parameters of the Splat model (**Appendix B.2**) are included as appendices, as is the full Splatter manual (**Appendix B.3**).

3.2 Splatter publication

Zappia et al. *Genome Biology* (2017) 18:174
DOI 10.1186/s13059-017-1305-0

Genome Biology

METHOD

Open Access



CrossMark

Splatter: simulation of single-cell RNA sequencing data

Luke Zappia^{1,2} , Belinda Phipson¹ and Alicia Oshlack^{1,2*}

Abstract

As single-cell RNA sequencing (scRNA-seq) technologies have rapidly developed, so have analysis methods. Many methods have been tested, developed, and validated using simulated datasets. Unfortunately, current simulations are often poorly documented, their similarity to real data is not demonstrated, or reproducible code is not available. Here, we present the Splatter Bioconductor package for simple, reproducible, and well-documented simulation of scRNA-seq data. Splatter provides an interface to multiple simulation methods including Splat, our own simulation, based on a gamma-Poisson distribution. Splat can simulate single populations of cells, populations with multiple cell types, or differentiation paths.

Keywords: Single-cell, RNA-seq, Simulation, Software

Background

The first decade of next-generation sequencing has seen an explosion in our understanding of the genome [1]. In particular, the development of RNA sequencing (RNA-seq) has enabled unprecedented insight into the dynamics of gene expression [2]. Researchers now routinely conduct experiments designed to test how gene expression is affected by various stimuli. One limitation of bulk RNA-seq experiments is that they measure the average expression level of genes across the many cells in a sample. However, recent technological developments have enabled the extraction and amplification of minute quantities of RNA, allowing sequencing to be conducted on the level of single cells [3]. The increased resolution of single-cell RNA-seq (scRNA-seq) data has made a range of new analyses possible.

As scRNA-seq data have become available there has been a rapid development of new bioinformatics tools attempting to unlock its potential. Currently there are at least 120 software packages that have been designed specifically for the analysis of scRNA-seq data, the majority of which have been published in peer-reviewed journals or as preprints [4]. The focus of these tools is often different from those designed for the analysis of a

bulk RNA-seq experiment. In a bulk experiment, the groups of samples are known and a common task is to test for genes that are differentially expressed (DE) between two or more groups. In contrast, the groups in a single-cell experiment are usually unknown and the analysis is often more exploratory.

Much of the existing software focuses on assigning cells to groups based on their expression profiles (clustering) before applying more traditional DE testing. This approach is taken by tools such as SC3 [5], CIDR [6], and Seurat [7] and is appropriate for a sample of mature cells where it is reasonable to expect cells to have a particular type. In a developmental setting, for example, where stem cells are differentiating into mature cells, it may be more appropriate to order cells along a continuous trajectory from one cell type to another. Tools such as Monocle [8], CellTree [9], and Sincell [10] take this approach, ordering cells along a path, then identifying patterns in the changes of gene expression along that path.

Another defining characteristic of scRNA-seq data is its sparsity; typically expression is only observed for relatively few genes in each cell. The observed zero counts have both biological (different cell types express different genes) and technical (an expressed RNA molecule might not be captured) causes, with technical zeros often referred to as “dropout”. Some analysis methods (ZIFA [11], MAST [12], ZINB-WaVE [13])

* Correspondence: alicia.oshlack@mcri.edu.au

¹Murdoch Childrens Research Institute, Royal Children's Hospital, 50 Flemington Rd, Parkville, VIC 3052, Australia

²School of Biosciences, The University of Melbourne, Parkville, VIC 3010, Australia

incorporate dropout into their models while others (MAGIC [14], SAVER [15], scImpute [16]) attempt to infer what the true expression levels should be.

Existing scRNA-seq analysis packages, and any new methods that are being developed, should demonstrate two properties: first that they can do what they claim to do, whether that is clustering, lineage tracing, differential expression testing or improved performance compared to other methods; and second that they produce some meaningful biological insight. The second criterion is specific to particular studies but it should be possible to address the first point in a more general way.

A common way to test the performance of an analysis method is through a simulation. Simulated data provide a known truth to test against, making it possible to assess whether a method has been implemented correctly, whether the assumptions of the method are appropriate, and demonstrating the method's limitations. Such tests are often difficult with real biological data, as an experiment must be specifically designed, or results from an appropriate orthogonal test taken as the truth. Simulations, however, easily allow access to a range of metrics for assessing the performance of an analysis method. An additional advantage of evaluating methods using simulated data is that many datasets, with different parameters and assumptions, can be rapidly generated at minimal cost. As such, many of the scRNA-seq analysis packages that are currently available have used simulations to demonstrate their effectiveness. These simulations, however, are often not described in a reproducible or reusable way and the code to construct them may not be readily available. When code is available it may be poorly documented or written specifically for the computing environment used by the authors, limiting its reproducibility and making it difficult for other researchers to reuse. Most importantly, publications do not usually provide sufficient detail demonstrating that a simulation is similar to real datasets, or in what ways it differs.

In this paper we present Splatter, an R Bioconductor package for reproducible and accurate simulation of single-cell RNA sequencing data. Splatter is a framework designed to provide a consistent interface to multiple published simulations, enabling researchers to quickly simulate scRNA-seq count data in a reproducible fashion and make comparisons between simulations and real data. Along with the framework we have developed our own simulation model, Splat, and show how it compares to previously published simulations based on real datasets. We also provide a short example of how simulations can be used for assessing analysis methods.

Results

The Splatter framework

Currently, Splatter implements six different simulation models, each with their own assumptions but accessed through a consistent, easy-to-use interface. These simulations are described in more detail in the following sections and in the documentation for each simulation in Splatter, which also describes the required input parameters.

The Splatter simulation process consists of two steps. The first step estimates the parameters required for the simulation from a real dataset. The result of the first step is a parameters object unique to each simulation model. These objects have been designed to hold the information required for the specific simulation and display details such as which parameters can be estimated and which have been changed from the default value. It is important that each simulation has its own object for storing parameters as different simulations can vary greatly in the information they require. For example, some simulations only need parameters for well-known statistical distributions while others require large vectors or matrices of data sampled from real datasets.

In the second step, Splatter uses the estimated parameters, along with any additional parameters that cannot be estimated or are overridden by the user, to generate a synthetic scRNA-seq dataset. If there is no relevant real data to estimate parameters from, a synthetic dataset can still be generated using default parameters that can be manually modified by the user. Additional parameters that may be required depend on the simulation; these could include parameters indicating whether to use a zero-inflated model or the number of genes and cells to simulate. The main result of the simulation step is a matrix of counts, which is returned as an SCESet object as defined by the scater package. Scater is a low-level analysis package that provides various functions for quality control, visualization, and preprocessing of scRNA-seq data [17]. Briefly, the structure of the SCESet combines cell by feature (gene) matrices for storing expression values along with tables for storing metadata about cells and features (further details are described in the scater documentation and the accompanying paper). This is a convenient format for returning intermediate values created during simulation as well as the final expression matrix. For example, the underlying gene expression means in different groups of cells are returned and could be used as a truth when evaluating differential expression testing. Using an SCESet also provides easy access to scater's functionality.

Splatter is also able to compare SCESet objects. These may contain simulations with different models or different parameters, or real datasets from which parameters have been estimated. The comparison function takes one

or more SCESet objects, combines them (keeping any cell or gene-level information that is present in all of them) and produces a series of diagnostic plots comparing aspects of scRNA-seq data. The combined datasets are also returned, making it easy to produce additional comparison plots or statistics. Alternatively, one SCESet can be designated as a reference, such as the real data used to estimate parameters, and the difference between the reference and the other datasets can be assessed. This approach is particularly useful for comparing how well simulations recapitulate real datasets. Examples of these comparison plots are shown in the following sections.

Simulation models

Splatter provides implementations of our own simulation model, Splat, as well as several previously published simulations. The previous simulations have either been published as R code associated with a paper or as functions in existing packages. By including them in Splatter, we have made them available in a single place in a more accessible way. If only a script was originally published, such as the Lun [18] and Lun 2 [19] simulations, the simulations have been re-implemented in Splatter. If the simulation is available in an existing R package, for example, scDD [20] and BASICS [21], we have simply written wrappers that provide consistent input and output but use the package implementation. We have endeavored to keep the simulations and estimation procedures as close as possible to what was originally published while providing a consistent interface within Splatter. The six different simulations currently available in Splatter are described below.

Simple

The negative binomial is the most common distribution used to model RNA-seq count data, as in the edgeR [22] and DESeq [23] packages. The Simple simulation is a basic implementation of this approach. A mean expression level for each gene is simulated using a gamma distribution and the negative binomial distribution is used to generate a count for each cell based on these means, with a fixed dispersion parameter (default = 0.1; Additional file 1: Figure S1). This simulation is primarily included as a baseline reference and is not intended to accurately reproduce many of the features of scRNA-seq data.

Lun

Published in “Pooling across cells to normalize single-cell RNA sequencing data with many zero counts” [18], the Lun simulation builds on the Simple simulation by adding a scaling factor for each cell (Additional file 1:

Figure S2). The cell factors are randomly sampled from a normal distribution with mean 1 and variance 0.5. The inverse- \log_2 transformed factors are used to adjust the gene means, resulting in a matrix in which each cell has a different mean. This represents the kinds of technical effects that scaling normalization aims to remove. The matrix of means is then used to sample counts from a negative binomial distribution, with a fixed dispersion parameter. This simulation can also model differential expression between multiple groups with fixed fold changes.

Lun 2

In “Overcoming confounding plate effects in differential expression analyses of single-cell RNA-seq data” [19] Lun and Marioni extended the negative binomial model from the Lun simulation. This simulation samples input parameters from real data, with very little random sampling from statistical distributions. In the Lun 2 simulation the cell factors are replaced with a library size factor and an additional level of variation is added by including a batch effects factor. While the library size factor acts on individual cells the batch effects are applied to groups of cells from the same batch. This simulation is thus highly specific to the scenario when there are known batch effects present in the data, for example, Fluidigm C1 plate effects. Differential expression can be added between two sets of batches and the user can choose to use a zero-inflated negative binomial (ZINB) model. Counts are simulated from a negative binomial using the library size and plate factor adjusted gene means and gene-wise dispersion estimates obtained from the real data. If the ZINB model is chosen, zero inflated estimates of gene means and dispersions are used instead. An additional step then randomly sets some counts to zero, based on the gene-wise proportions of zeros observed in the data. Additional file 1: Figure S3 shows the model assumptions and parameters for this simulation.

scDD

The scDD package aims to test for differential expression between two groups of cells but also more complex changes such as differential distributions or differential proportions [20]. This is reflected in the scDD simulation, which can contain a mixture of genes simulated to have different distributions, or differing proportions where the expression of the gene is multi-modal. This simulation also samples information from a real dataset. As the scDD simulation is designed to reproduce a high quality, filtered dataset, it only samples from genes with less than 75% zeros. As a result, it only simulates relatively highly expressed genes. The Splatter package

simply provides wrapper functions to the simulation function in the scDD package, while capturing the necessary inputs and outputs needed to compare to other simulations. The full details of the scDD simulation are described in the scDD package vignette [24].

BASiCS

The BASiCS package introduced a model for separating variation in scRNA-seq data into biological and technical components based on the expression of external spike-in controls [21]. This model also enables cell-specific normalization and was extended to detect differential expression between groups of cells [25]. Similar to the scDD simulation, Splatter provides a wrapper for the BASiCS simulation function, which is able to produce datasets with both endogenous and spike-in genes as well as multiple batches of cells. As the BASiCS simulation contains both biological and technical variation, it can be used to test the ability of methods to distinguish between the two.

Splat

We have developed the Splat simulation to capture many features observed in real scRNA-Seq data, including high expression outlier genes, differing sequencing depths (library sizes) between cells, trended gene-wise dispersion, and zero-inflation. Our model uses parametric distributions with hyper-parameters estimated from real data (Fig. 1). The core of the Splat simulation is the gamma-Poisson hierarchical model where the mean expression level for each gene i , $i = 1, \dots, N$, is simulated from a gamma distribution and the count for each cell j , $j = 1, \dots, M$, is subsequently sampled from a Poisson distribution, with modifications to include expression outliers and to enforce a mean-variance trend.

More specifically, the Splat simulation initially samples gene means from a Gamma distribution with shape α and rate β . While the gamma distribution is a good fit for gene means it does not always capture extreme expression levels. To counter this a probability (π^O) that a gene is a high expression outlier can be specified. We then add these outliers to the simulation by replacing the previously simulated mean with the median of the simulated gene means multiplied by an inflation factor. The inflation factor is sampled from a log-normal distribution with location μ^O and scale σ^O .

The library size (total number of counts) varies within an scRNA-seq experiment and can be very different between experiments depending on the sequencing depth. We model library size using a log-normal distribution (with location μ^L and scale σ^L) and use the simulated library sizes (L_j) to proportionally adjust the gene means for each cell. This allows us to alter the

number of counts per cell independently of the underlying gene expression levels.

It is known that there is a strong mean-variance trend in RNA-Seq data, where lowly expressed genes are more variable and highly expressed genes are more consistent [26]. In the Splat simulation we enforce this trend by simulating the biological coefficient of variation (BCV) for each gene from a scaled inverse chi-squared distribution, where the scaling factor is a function of the gene mean. After simulating the BCV values we generate a new set of means (λ_{ij}) from a gamma distribution with shape and rate parameters dependent on the simulated BCVs and previous gene means. We then generate a matrix of counts by sampling from a Poisson distribution, with lambda equal to λ_{ij} . This process is similar to the simulation of bulk RNA-seq data used by Law et al. [27].

The high proportion of zeros is another key feature of scRNA-seq data [11], one cause of which is technical dropout. We use the relationship between the mean expression of a gene and the proportion of zero counts in that gene to model this process and use a logistic function to produce a probability that a count should be zero. The logistic function is defined by a midpoint parameter (x_0), the expression level at which 50% of cells are zero, and a shape parameter (k) that controls how quickly the probabilities change from that point. The probability of a zero for each gene is then used to randomly replace some of the simulated counts with zeros using a Bernoulli distribution.

Each of the different steps in the Splat simulation outlined above are easily controlled by setting the appropriate parameters and can be turned off when they are not desirable or appropriate. The final result is a matrix of observed counts Y_{ij} where the rows are genes and the columns are cells. The full set of input parameters is shown in Table 1.

Comparison of simulations

To compare the simulation models available in Splatter we estimated parameters from several real datasets and then generated synthetic datasets using those parameters. Both the standard and zero-inflated versions of the Splat and Lun 2 simulations were included, giving a total of eight simulations. We began with the Tung dataset which contains induced pluripotent stem cells from three HapMap individuals [28].

To reduce the computational time we randomly sampled 200 cells to use for the estimation step and each simulation consisted of 200 cells. Benchmarking showed a roughly linear relationship between the number of genes or cells and the processing time required (Additional file 1: Figures S4 and S5). The estimation procedures for the Lun 2 and BASiCS simulations are particularly time consuming; however, the Lun 2 estimation can be run using

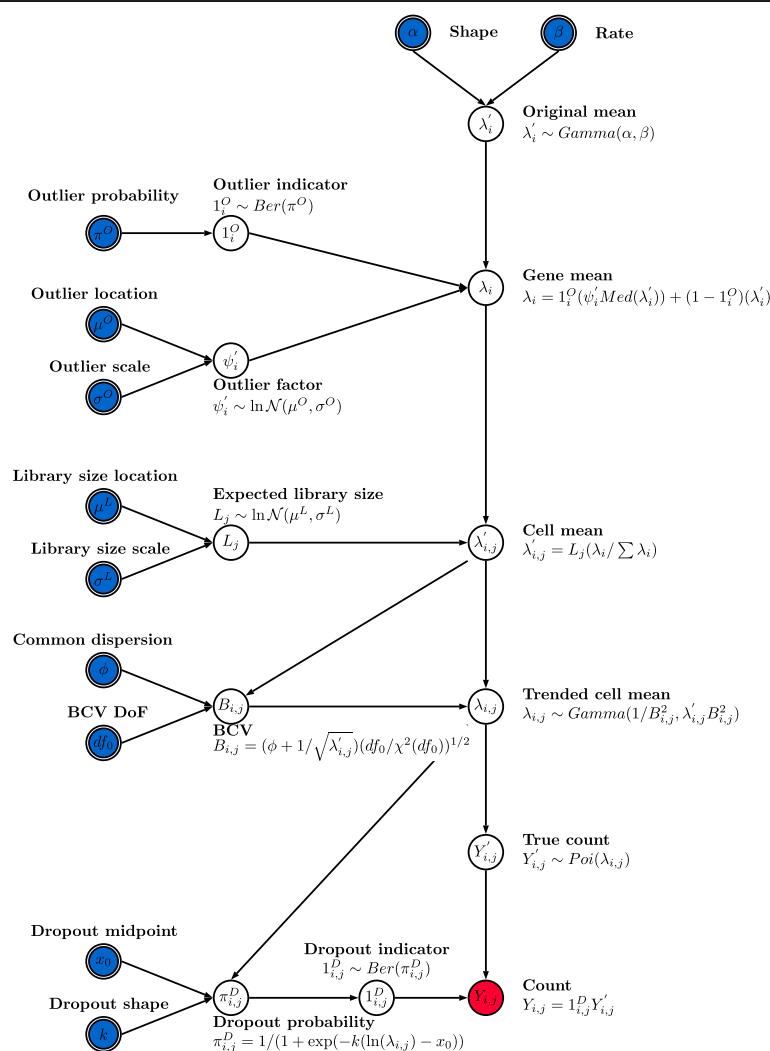


Fig. 1 The Splat simulation model. Input parameters are indicated with double borders and those that can be estimated from real data are shaded blue. Red shading indicates the final output. The simulation begins by generating means from a gamma distribution. Outlier expression genes are added by multiplying by a log-normal factor and the means are proportionally adjusted for each cell's library size. Adjusting the means using a simulated Biological Coefficient of Variation (BCV) enforces a mean-variance trend. These final means are used to generate counts from a Poisson distribution. In the final step dropout is (optionally) simulated by randomly setting some counts to zero, based on each gene's mean expression. DoF degrees of freedom

multiple cores unlike the BASiCS estimation procedure. We did not perform any quality control of cells and only removed genes that were zero in all of the selected cells. We believe this presents the most challenging situation to simulate, as there are more likely to be violations of the underlying model. This scenario is also possibly the most useful as it allows any analysis method to be evaluated, from low-level filtering to complex downstream analysis.

Figure 2 shows some of the plots produced by Splatter to compare simulations based on the Tung dataset.

We compared the gene means, variances, library sizes, and the mean–variance relationship. From these diagnostic plots, we can evaluate how well each simulation reproduces the real dataset and how it differs. One way to compare across the simulations is to look at the overall distributions (Fig. 2, left column).

Table 1 Input parameters for the Splat simulation model

Name	Symbol	Description
Mean shape	α	Shape parameter for the mean gene expression gamma distribution
Mean rate	β	Rate parameter for the mean gene expression gamma distribution
Library size location	μ^L	Location parameter for the library size log-normal distribution
Library size scale	σ^L	Scale parameter for the library size log-normal distribution
Outlier probability	π^O	Probability that a gene is an expression outlier
Outlier location	μ^O	Location parameter for the expression outlier factor log-normal distribution
Outlier scale	σ^O	Scale parameter for the expression outlier factor log-normal distribution
Common BCV	ϕ	Common BCV dispersion across all genes
BCV degrees of freedom	df	Degrees of freedom for the BCV inverse chi-squared distribution
Dropout midpoint	x_0	Midpoint for the dropout logistic function
Dropout shape	k	Shape of the dropout logistic function

Alternatively, we can choose a reference (in this case the real data) and look at departures from that data (Fig. 2, right column). Examining the mean expression levels across genes, we see that the scDD simulation is missing lowly expressed genes, as expected, as is the Lun simulation. In contrast, the Simple and Lun 2 simulations are skewed towards lower expression levels (Fig. 2a, b). The BASiCS simulation is a good match to the real data as is the Splat simulation. Both versions of the Lun 2 simulation produce some extremely highly variable genes, an effect which is also seen to a lesser extent in the Lun simulation. The difference in variance is reflected in the mean–variance relationship where genes from the Lun 2 simulation are much too variable at high expression levels for this dataset. Library size is another aspect in which the simulations differ from the real data. The simulations that do not contain a library size component (Simple, Lun, scDD) have different median library sizes and much smaller spreads. In this example, the BASiCS simulation produces too many large library sizes, as does the Lun 2 simulation to a lesser degree.

A key aspect of scRNA-seq data is the number of observed zeros. To properly recreate an scRNA-seq dataset a simulation must produce the correct number of zeros but also have them appropriately distributed across both genes and cells. In addition, there is a clear relationship between the expression level of a gene and the number of observed zeros [29] and this should be reproduced in simulations. Figure 3 shows the distribution of zeros for the simulations based on the Tung dataset.

For this dataset the Simple and Lun 2 simulations produce too many zeros across both genes and cells while the Lun and scDD simulations produce too few. Interestingly, the Splat simulation produces a better fit to this dataset when dropout is not included, suggesting that additional dropout is not present in the Tung dataset. However, this is not the case for all data and sometimes simulating additional dropout produces a better fit to the data (for example, the Camp dataset presented below). We can also consider the relationship between the expression level of a gene, calculated including cells with zero counts, and the percentage of zero counts in that gene. The Lun and scDD simulations produce too few zeros at low expression levels, while the Simple and Lun 2 simulations produce too many zeros at high expression levels. It is important to note that as the scDD simulation removes genes with more than 75% zeros prior to simulation this model can never produce genes with high numbers of zeros as shown in Fig. 3c. Both the Splat and BASiCS models are successful at distributing zeros across genes and cells as well as maintaining the mean–zeros relationship.

Although the analysis presented in Figs. 2 and 3 allows us to visually inspect how simulations compare with a single dataset, we also wished to compare simulations across a variety of datasets. To address this we performed simulations based on five different datasets (outlined in Table 2) that varied in terms of library preparation protocol, cell capture platform, species, and tissue complexity. Three of the datasets used Unique Molecular Identifiers (UMIs) [30] and two used full-length protocols. Complete comparison panels for all the datasets are provided in Additional file 1: Figures S5–S10 and processing times for all datasets are shown in Additional file 1: Figure S11.

For each dataset, we estimated parameters and produced a synthetic dataset as described previously. We then compared simulations across metrics and datasets by calculating a median absolute deviation (MAD) for each metric. For example, to get a MAD for the gene expression means, the mean expression values for both the real data and the simulations were sorted and the real values were subtracted from the simulated values. The median of these absolute differences was taken as the final statistic. To compare between simulations, we ranked the MADs for each metric with a rank of one being most similar to the real data. Figure 4 summarizes the ranked results for the five datasets as a heatmap. A heatmap of the MADs is presented in Additional file 1: Figure S12 and the values themselves in Additional file 2.

Looking across the metrics and datasets we see that the Splat simulations are consistently highly ranked. In

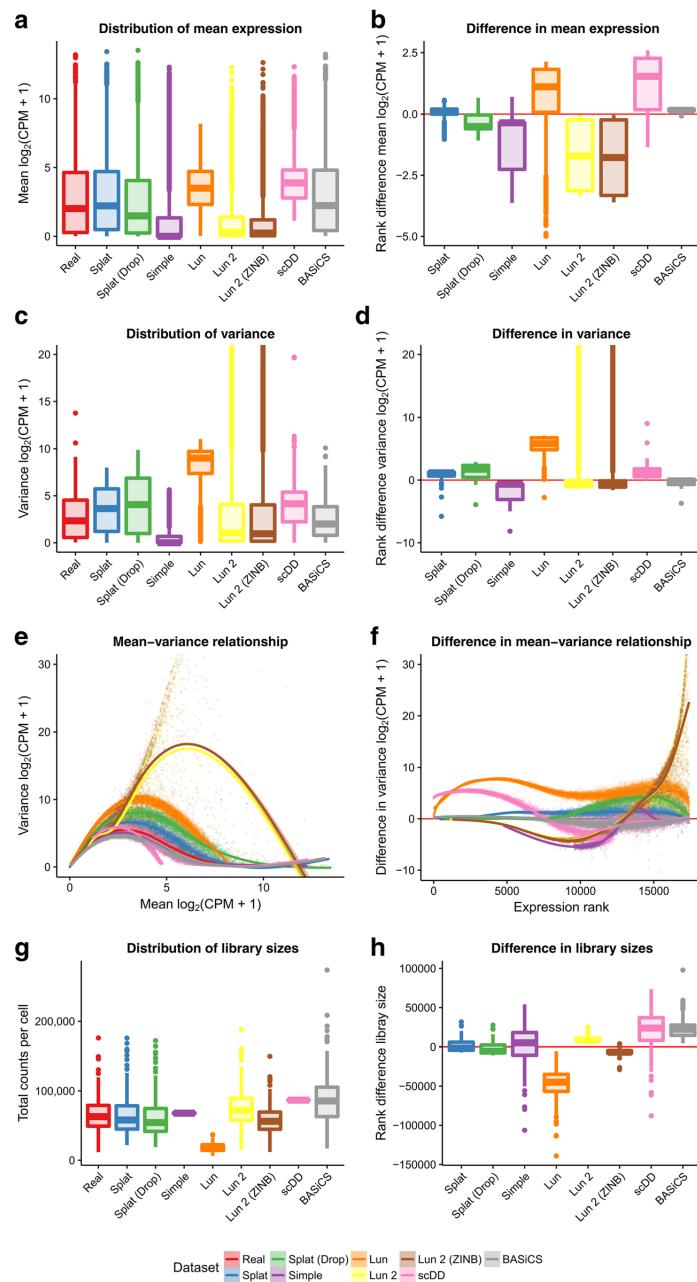


Fig. 2 Comparison of simulations based on the Tung dataset. The left column panels show the distribution of mean expression (a), variance (c) and library size (g) across the real dataset and the simulations as boxplots, along with a scatter plot of the mean–variance relationship (e). The right column shows boxplots of the ranked differences between the real data and simulations for the same statistics: mean (b), variance (d), mean–variance relationship (f), and library size (h). Note that the y-axis for plots of the variance has been limited in order to show more detail. Variances for the Lun and Lun 2 simulations extend beyond what has been shown

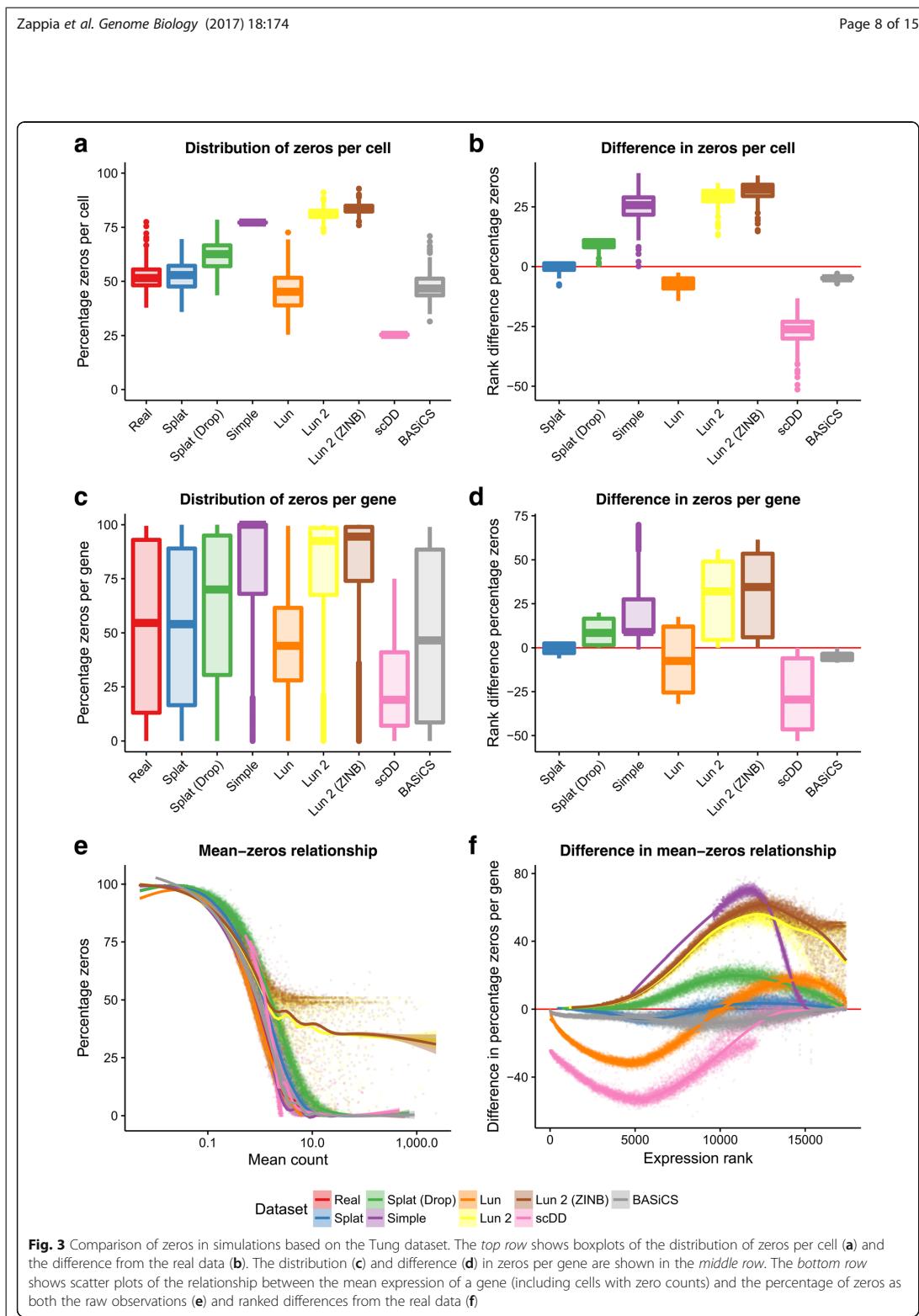
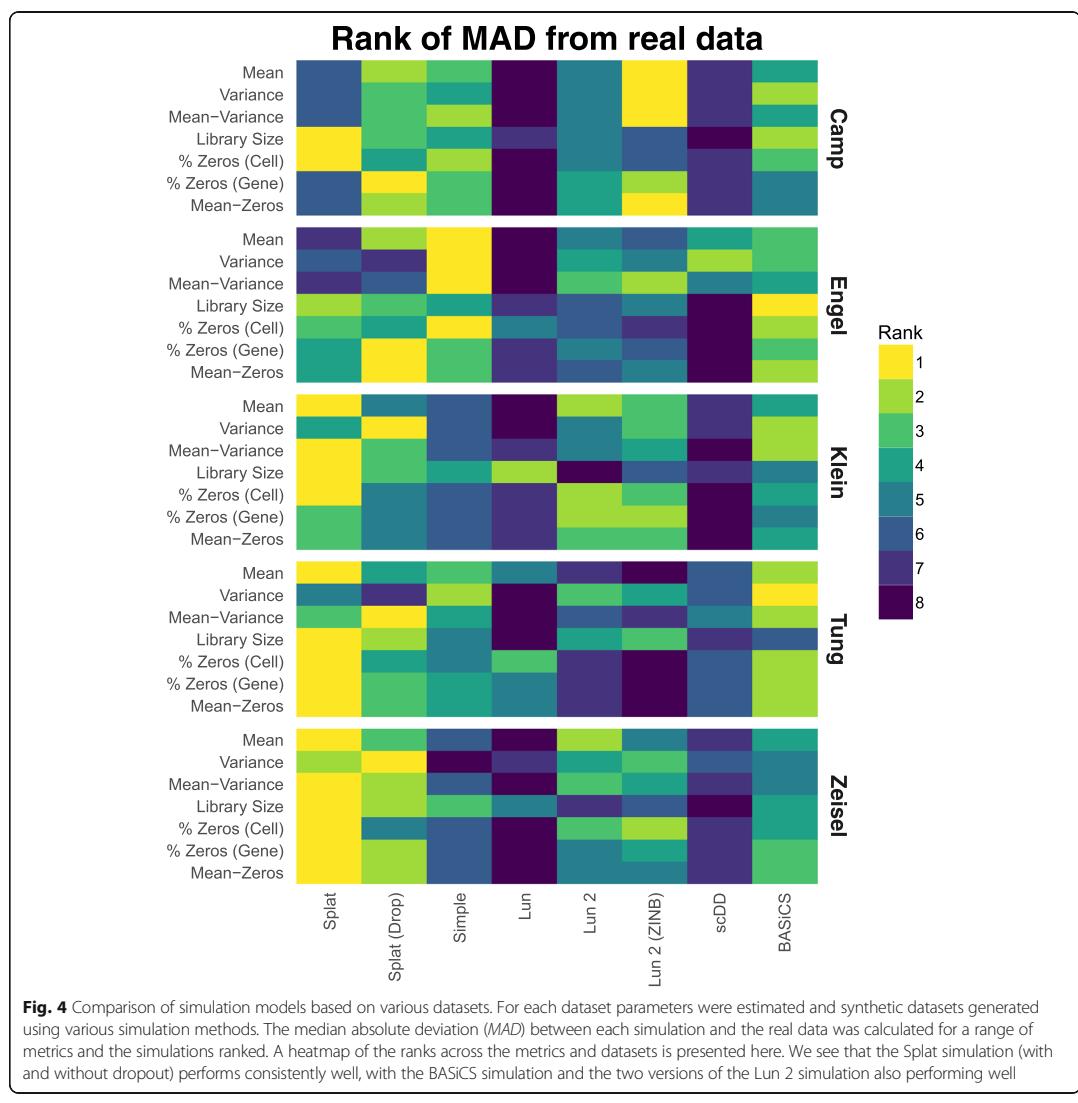


Table 2 Details of real datasets

Dataset	Species	Cell type	Platform	Protocol	UMI	Number of cells
Camp [44]	Human	Whole brain organoids	Fluidigm C1	SMARTer	No	597
Engel [45]	Mouse	Natural killer T cells	Flow cytometry	Modified Smart-seq2	No	203
Klein [46]	Human	K562 cells	InDrop	CEL-Seq	Yes	213
Tung [28]	Human	Induced pluripotent stem cells	Fluidigm C1	Modified SMARTer	Yes	564
Zeisel [47]	Mouse	Cortex and hippocampus cells	Fluidigm C1	STRT-Seq	Yes	3005



general, it seems that the datasets are not zero-inflated and thus the zero-inflated simulations do not perform as well as their regular counterparts. The Splat simulations were least successful on the Camp cerebral organoid and Engel T-cell datasets. The complex nature of the Camp data (many cell types) and the full-length protocols used by both may have contributed to Splat's poorer performance. In this situation the semi-parametric, sampling-based models may have an advantage and the Lun 2 simulation was the best performer on most aspects of the Camp data. Interestingly, the Simple simulation was the best performer on the Engel dataset. This result suggests that the additional features of the more complex simulations may be unnecessary in this case or that other models may be more appropriate. The Lun simulation is consistently among the worst performing. However, given that this model is largely similar to the others, it is likely due to the lack of an estimation procedure for most parameters rather than significant problems with the model itself. The scDD simulation also often differed significantly from the real data, which is unsurprising as this simulation is designed to produce a filtered dataset, not the raw datasets used here. A comparison based on a filtered version of the Tung dataset, showing scDD to be a better match, is provided in Additional file 1: Figure S13.

Most importantly we see that simulations perform differently on different datasets. This emphasizes the importance of evaluating different models and demonstrating their similarity to real datasets. Other comparisons may also be of interest for evaluation, such as testing each simulated gene to see if it matches known distributions, an example of which is shown in Additional file: 1 Figure S14. The Splatter framework makes these comparisons between simulation models straightforward, making it easier for researchers to choose simulations that best reflect the data they are trying to model.

Complex simulations with Splat

The simulation models described above are sufficient for simulating a single, homogeneous population but not to reproduce the more complex situations seen in some real biological samples. For example, we might wish to simulate a population of cells from a complex tissue containing multiple mature cell types or a developmental scenario where cells are transitioning between cell types. In this section, we present how the Splat simulation can be extended to reproduce these complex sample types (Fig. 5).

Simulating groups

Splat can model samples with multiple cell types by creating distinct groups of cells where several genes are differentially expressed between the different groups. Previously published simulations can reproduce this situation to

some degree but are often limited to fixed fold changes between only two groups. In the Splat simulation, however, differential expression is modeled using a process similar to that for creating expression outliers and can be used to simulate complex cell mixtures. Specifically a multiplicative differential expression factor is assigned to each gene and applied to the underlying mean. For DE genes, these factors are generated from a log-normal distribution while for other genes they are equal to one. Setting the number of groups and the probability that a cell comes from each group allows flexibility in how different groups are defined. Additionally, parameters controlling the probability that genes are differentially expressed as well as the magnitude and direction of DE factors can be set individually for each group. The resulting SCESet object contains information about which group each cell comes from as well as the factors applied to each gene in each group (Fig. 5a).

Simulating batches

A common technical problem in all sequencing experiments is batch effects, where technical variation is created during sample collection and preparation. The Splat simulation can model these effects using multiplicative factors that are applied to all genes for groups of cells. Adding this extra layer of variation allows researchers to evaluate how methods perform in the presence of unwanted variation (Fig. 5b).

Simulating paths

A common use of scRNA-seq is to study cellular development and differentiation. Instead of having groups of mature cells, individual cells are somewhere on a continuous differentiation path or lineage from one cell type to another. To model this, the Splat simulation uses the differential expression process described above to define the expression levels of a start and end cell for each path. A series of steps is then defined between the two cell types and the simulated cells are randomly assigned to one of these steps, receiving the mean expression levels at that point. Therefore, the simulation of lineages using Splat is defined by the differential expression parameters used to create the differences between the start and end of each path. It also incorporates the parameters that define the path itself, such as the length (number of steps) and skew (whether cells are more likely to come from the start or end of the path).

In real data it has been observed that expression of genes can change in more complex, non-linear ways across a differentiation trajectory. For example, a gene may be lowly expressed at the beginning of a process, highly expressed in the middle and lowly expressed at the end. Splat models these kinds of changes by

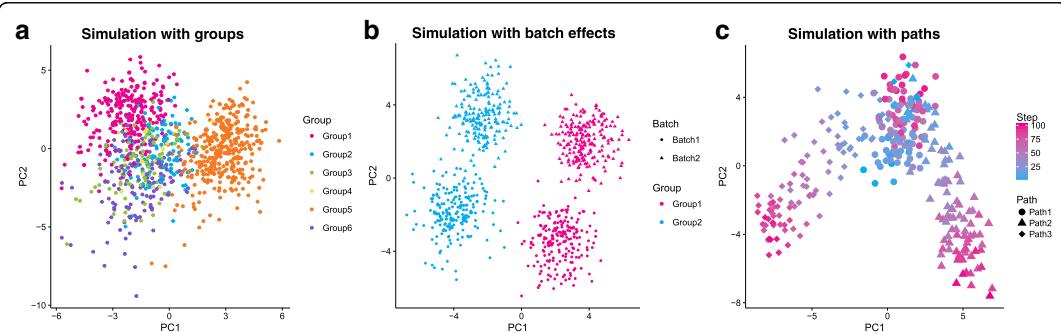


Fig. 5 Examples of complex Splat simulations. **a** A principle components analysis (PCA) plot of a simulation with six groups with varying numbers of cells and levels of differential expression. **b** A PCA plot of a simulation with two groups (pink and blue) and two batches (circle and triangle). *PC1* separates groups (wanted biological variation) while *PC2* separates batches (unwanted technical variation). **c** A PCA plot of a simulation with differentiation paths; the colored gradient indicates how far along a path each cell is from blue to pink. A progenitor cell type (blue circles) differentiates into an intermediate cell type (pink circles/blue triangles or diamonds), which becomes one of two (pink triangle or diamond) mature cell types

generating a Brownian bridge (a random walk with fixed end points) between the two end cells of a path, which is then smoothed and interpolated using an Akima spline [31, 32]. This random element allows many possible patterns of expression changes over the course of a path (Additional file 1: Figure S15). While non-linear changes are possible they are not the norm. Splat defines parameters that control the proportion of genes that are non-linear and how variable those genes can be.

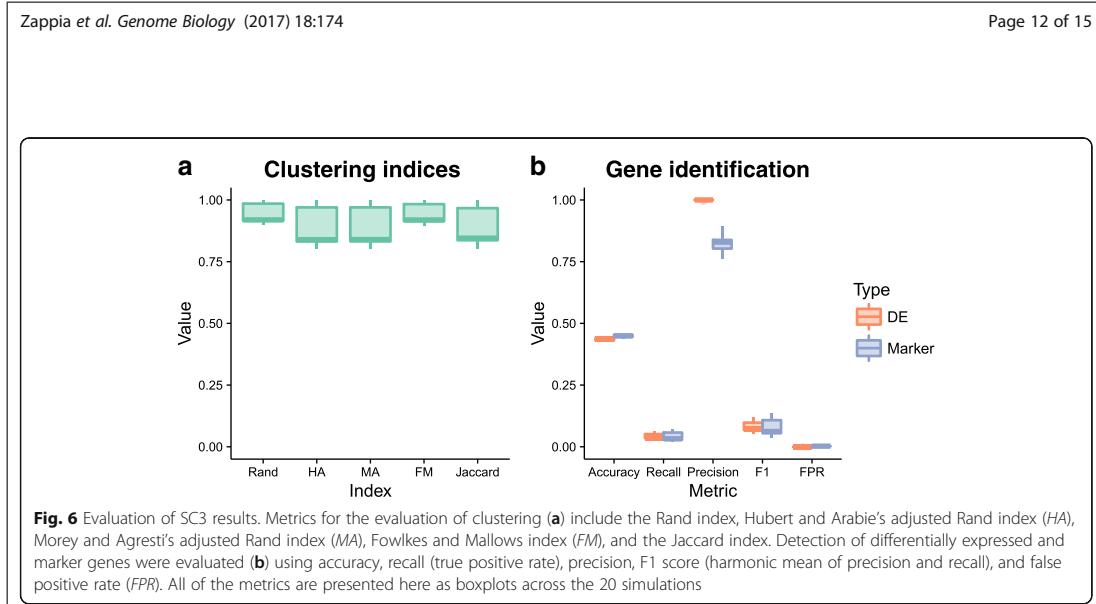
Further complexity in simulating differentiation paths can be achieved by modeling lineages with multiple steps or branches. For example, a stem cell that differentiates into an intermediate cell type that then changes into one of two mature cell types. These possibilities are enabled by allowing the user to set a starting point for each path (Fig. 5c).

Example: using Splatter simulations to evaluate a clustering method

To demonstrate how the simulations available in Splatter could be used to evaluate an analysis method we present an example of evaluating a clustering method. SC3 [5] is a consensus k -means-based approach available from Bioconductor [33]. As well as assigning cells to groups, SC3 is able to detect genes that are differentially expressed between groups and marker genes that uniquely identify each group. To test SC3 we estimated Splat simulation parameters from the Tung dataset and simulated 400 cells from three groups with probabilities of 0.6, 0.25, and 0.15. The probability of a gene being differentially expressed in a group was 0.1, resulting in approximately 1700 DE genes per group. We then ran SC3 with three clusters ($k = 3$) and compared the results to the true groupings (Fig. 6a). We also assessed the detection of DE and marker genes. True DE genes were taken as

genes with simulated DE in any group and true marker genes as the subset of DE genes that were DE in only a single group (Fig. 6b). This procedure was repeated 20 times with different random seeds to get some idea of the variability and robustness of the method.

Figure 6 shows the evaluation of SC3's clustering and gene identification on the simulated data. Five measures were used to evaluate the clustering: the Rand index (Rand), Hubert and Arabie's (HA) adjusted Rand index and Morey and Agresti's (MA) adjusted Rand index (both of which adjust for chance groupings), Fowlkes and Mallows index (FM) and the Jaccard index (Jaccard). All of these indices attempt to measure the similarity between two clusterings, in this case the clustering returned by SC3 and the true groups in the simulation. SC3 appears to identify clusters well for the majority of simulations, in some cases producing a near-perfect clustering. It may be interesting to examine individual cases further in order to identify when SC3 is able to perform better. Both the DE genes and marker genes identified by SC3 show a similar pattern across our classification metrics of accuracy, precision, recall, and F1 score. On average approximately 2700 of the truly DE genes and 2500 of the true marker genes passed SC3's automatic filtering (with additional non-DE genes). SC3 then detected around 100 DE genes per simulation, along with 99 marker genes (median values). Precision (the proportion of identified genes that are true positives) is very high while recall (the proportion of true positives that were identified, or true positive rate) is very low. This tells us that in this scenario SC3 is producing many false negatives, but that the genes that it finds to be markers or DE are correct. This result is often desirable, particularly for marker genes, and is reflected in the very low false positive rate.



While it is beyond the scope of this paper, clearly this evaluation could be extended, for example, by including more clustering methods, more variations in simulation parameters, and investigating why particular results are seen. However, these data, and the code used to produce them, are an example of how such an evaluation could be conducted using the simulations available in Splatter.

Discussion and conclusions

The recent development of single-cell RNA sequencing has spawned a plethora of analysis methods, and simulations can be a powerful tool for developing and evaluating them. Unfortunately, many current simulations of scRNA-seq data are poorly documented, not reproducible, or fail to demonstrate similarity to real datasets. In addition, simulations created to evaluate a specific method can sometimes fall into the trap of having the same underlying assumptions as the method that they are trying to test. An independent, reproducible, and flexible simulation framework is required in order for the scientific community to evaluate and develop sophisticated analysis methodologies.

Here we have developed Splatter, an independent framework for the reproducible simulation of scRNA-seq data. Splatter is available as an R package from Bioconductor, under a GPL-3 license, and implements a series of simulation models. Splatter can easily estimate parameters for each model from real data, generate synthetic datasets and quickly create a series of diagnostic plots comparing different simulations and datasets.

As part of Splatter we introduce our own simulation called Splat. Splat builds on the gamma-Poisson (or negative binomial) distribution commonly used to represent RNA-seq data, and adds high-expression outlier

genes, library size distributions, a mean-variance trend, and the option of expression-based dropout. Extensions to Splat include the simulation of more complex scenarios, such as multiple groups of cells with differing sizes and levels of differential expression, experiments with several batches, or differentiation trajectories with multiple paths and branches, with genes that change in non-linear ways.

We performed an evaluation of the six simulation models currently available in Splatter by comparing synthetic data generated using estimated parameters to five published datasets. Overall Splat performed well, ranking highly on most metrics. However, other simulations performed better for some metrics or better reproduced specific datasets. We found the Camp cerebral organoid dataset the most challenging to simulate, perhaps because of the complex nature of this sample, which is comprised of many different cell types. In addition, this dataset (along with the Engel data) used a full-length protocol, which may contain additional noise compared to the UMI datasets [34].

One of the key features of scRNA-seq data is the high number of zero counts where no expression is observed for a particular gene in a particular cell. This can be especially challenging to simulate as not only must there be the correct number of zeros but they must be correctly distributed across genes and cells. We found that introducing dropout (in Splat) or zero-inflation (in Lun 2) often failed to improve the match to real datasets, suggesting that they are not truly zero-inflated. Together, the results demonstrate that no simulation can accurately reproduce all scRNA-seq datasets. They also emphasize the variability in scRNA-seq data, which arises from a complex set of biological (for example, species, tissue type, cell

type, treatment, and cell cycle) and technical (for example, platform, protocol, or processing) factors. Non-parametric simulations that permute real data could potentially produce more realistic synthetic datasets but at the cost of flexibility in what can be simulated and knowledge of the underlying parameters.

Finally, we demonstrated how Splatter could be used for the development and evaluation of analysis methods, using the SC3 clustering method as an example. Splatter's flexible framework allowed us to quickly generate multiple test datasets, based on parameters from real data. The information returned about the simulations gave us a truth to test against when evaluating the method. We found that SC3 accurately clustered cells and was precise in identifying DE and marker genes.

The simulations available in Splatter are well documented, reproducible, and independent of any particular analysis method. Splatter's comparison functions also make it easy to demonstrate how similar simulations are to real datasets. Splatter provides a framework for simulation models, makes existing scRNA-seq simulations accessible to researchers and introduces Splat, a new scRNA-seq simulation model. As more simulation models become available, such as those replicating newer technologies including k-cell sequencing, they can be adapted to Splatter's framework. The Splat model will continue to be developed and may, in the future, include additional modules such as the ability to add gene lengths to differentiate between UMI and full-length data. We hope that Splatter empowers researchers to rapidly and rigorously develop new scRNA-seq analysis methods, ultimately leading to new discoveries in cell biology.

Methods

Splat parameter estimation

To easily generate a simulation that is similar to a given dataset, Splatter includes functions to estimate the parameters for each simulation from real datasets. Just as with the simulation models themselves, the estimation procedures are based on what has been published and there is variation in how many parameters can be estimated for each model. We have given significant attention to estimating the parameters for the Splat simulation. The parameters that control the mean expression of each gene (α and β) are estimated by fitting a gamma distribution to the winsorized means of the library size normalized counts using the fitdistrplus package [35]. The library size normalization is a basic normalization where the counts in the original dataset are adjusted so that each cell has the same number of total counts (in this case the median across all cells) and any genes that are all zero are removed. We found that genes with extreme means affect the fit of the gamma distribution

and that this effect was mitigated by winsorizing the top and bottom 10% of values to the 10th and 90th percentiles, respectively. Parameters for the library size distribution (μ^L and σ^L) are estimated in a similar way by fitting a log-normal distribution to the unnormalised library sizes.

The procedure for estimating expression outlier parameters is more complex. Taking the library size normalized counts, outliers are defined as genes where the mean expression is more than two MADs greater than the median of the gene expression means. The outlier probability π^O is then calculated as the proportion of genes that are outliers. Parameters for the outlier factors (μ^O and σ^O) are estimated by fitting a log-normal distribution to the ratio of the means of the outlier genes to the median of the gene expression means.

BCV parameters are estimated using the estimateDisp function in the edgeR package [22]. When testing the estimation procedure on simulated datasets we observed that the edgeR estimate of common dispersion was inflated (Additional file 1: Figure S16); therefore, we apply a linear correction to this value ($\hat{\phi} = 0.1 + 0.25\hat{\phi}_{\text{edgeR}}$).

The midpoint (x_0) and shape (k) parameters for the dropout function are estimated by fitting a logistic function to the relationship between the log means of the normalized counts and the proportion of samples that are zero for each gene (Additional file 1: Figure S17).

While we note that our estimation procedures are somewhat ad hoc, we found that these procedures are robust, efficient, and guaranteed to produce parameter estimates on all datasets we tested.

Datasets

Each of the real datasets used in the comparison of simulations is publicly available. Raw FASTQ files for the Camp dataset were downloaded from SRA (accession SRP066834) and processed using a Bpipe (v0.9.9.3) [36] pipeline that examined the quality of reads using FastQC (v0.11.4), aligned the reads to the hg38 reference genome using STAR (v2.5.2a) [37], and counted reads overlapping genes in the Gencode V22 annotation using featureCounts (v1.5.0-p3) [38]. Matrices of gene by cell expression values for the Klein (accession GSM1599500) and Zeisel (accession GSE60361) datasets were downloaded from GEO. For the Tung dataset the matrix of molecules (UMIs) aligned to each gene available from <https://github.com/jdblischak/singleCellSeq> was used. These data are also available from GEO (accession GSE77288). The Salmon [39] quantification files for the Engel dataset were download from the Conquer database (<http://imlspenticton.uzh.ch:3838/conquer/>) and converted to a gene by cell matrix using the tximport [40] package.

Simulation comparison

For each dataset the data file was read into R (v3.4.0) [41] and converted to a gene by cell matrix. We randomly selected 200 cells without replacement and filtered out any genes that had zero expression in all cells or any missing values. The parameters for each simulation were estimated from the selected cells and a synthetic dataset generated with 200 cells and the same number of genes as the real data. Simulations were limited to 200 cells (the size of the smallest dataset) to reduce the computational time required. When estimating parameters for the Lun 2, scDD, and BASiCS simulations cells were randomly assigned to two groups. For the Splat and Lun 2 simulations both the regular and zero-inflated variants were used to simulate data. The resulting eight simulations were then compared to the real data using Splatter's comparison functions and plots showing the overall comparison produced. To compare simulations across the datasets summary statistics were calculated. For each of the basic metrics (mean, variance, library size, zeros per gene, and zeros per cell) the genes were sorted individually for each simulation and the difference from the sorted values and the real data calculated. When looking at the relationship between mean expression level and other metrics (variance, zeros per gene) genes in both the real and simulated data were sorted by mean expression and the difference between the metric of interest (e.g., variance) calculated. The median absolute deviation for each metric was then calculated and ranked for each dataset to give the rankings shown in Fig. 4.

Clustering evaluation

Parameters for Splat simulations used in the example evaluation of SC3 were estimated from the Tung dataset. Twenty synthetic datasets were generated using these parameters with different random seeds. Each simulation had three groups of different cells, with probabilities of 0.6, 0.25 and 0.1, and a probability of a gene being differentially expressed of 0.1. Factors for differentially expressed genes were generated from a log-normal distribution with location parameter equal to -0.1 and scale parameter equal to 0.3. For each simulation the SC3 package was used to cluster cells with $k = 3$ and asked to detect DE and marker genes, taking those with adjusted p values less than 0.05. True DE genes were defined as genes where the simulated DE factor was not equal to 1 in one or more groups. Marker genes were defined as genes where the DE factor was not equal to 1 in a single group (and 1 in all others). Clustering metrics were calculated using the clues R package [42]. To evaluate the DE and marker gene detection we calculated the numbers of true negatives (TN), true positives (TP), false negatives (FN),

and false positives (FP). We then used these values to calculate the metrics shown in Fig. 6: accuracy ($Acc = (TP + TN) / Total\ number\ of\ genes$), recall ($Rec = TP / (TP + FN)$), precision ($Pre = TP / (TP + FP)$), F1 score ($F1 = 2 * ((Pre * Rec) / (Pre + Rec))$), and false positive rate ($FPR = FP / (FP + TN)$). Metrics were aggregated across the 20 simulations and boxplots produced using the ggplot2 package [43].

Session information describing the packages used in all analysis steps is included as Additional file 3. The code and dataset files are available at <https://github.com/Oshlack/splatter-paper> under an MIT license.

Additional files

Additional file 1: Figures S1–S17 Diagrams of other simulation models, Splatter comparison output for all datasets, example non-linear gene dispersion estimate correction, mean-zeros fit, benchmarking, and processing times (PDF 17991 kb)

Additional file 2: Table of the median absolute deviations used to produce Fig. 4 in CSV format. (CSV 37 kb)

Additional file 3: Session information. Details of the R environment and packages used for analysis. (PDF 118 kb)

Acknowledgements

We would like to thank the authors of the BASiCS and scDD packages for their responses to our questions about how to include their simulations in Splatter as well as Mark Robinson and Charlotte Soneson for discussions regarding the simulation of scRNA-seq data. Our thanks also to Jovana Maksimovic and Sarah Blood for their comments on the manuscript.

Funding

Luke Zappia is supported by an Australian Government Research Training Program (RTP) Scholarship. Alicia Oshlack is supported through a National Health and Medical Research Council Career Development Fellowship APP1126157. MCRI is supported by the Victorian Government's Operational Infrastructure Support Program.

Availability of data and materials

The datasets analyzed during the current study are available from the repositories specified in the methods. The code used to analyze them is available under an MIT license from the repository for this paper <https://github.com/Oshlack/splatter-paper> (doi: 10.5281/zenodo.833571). Copies of the datasets are also provided in this repository. The Splatter package is available from Bioconductor (<http://bioconductor.org/packages/splatter/>) and is being developed on Github (<https://github.com/Oshlack/splatter>) under a GPL-3 license. The specific version of Splatter used in this paper, which includes the BASiCS simulation, is available at <https://github.com/Oshlack/splatter/releases/tag/v1.1.3-basics> (doi: 10.5281/zenodo.833574).

Authors' contributions

LZ developed the software and performed the analysis. BP contributed to the statistics and supervision. AO oversaw all aspects of the project. All authors contributed to drafting the manuscript. All authors read and approved the final manuscript.

Ethics approval and consent to participate

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Received: 3 May 2017 Accepted: 22 August 2017
Published online: 12 September 2017

References

- Goodwin S, McPherson JD, Richard McCombie W. Coming of age: ten years of next-generation sequencing technologies. *Nat Rev Genet.* 2016;17:333–51.
- Ozsolak F, Milos PM. RNA sequencing: advances, challenges and opportunities. *Nat Rev Genet.* 2011;12:87–98.
- Tang F, Barbacioru C, Wang Y, Nordman E, Lee C, Xu N, et al. mRNA-Seq whole-transcriptome analysis of a single cell. *Nat Methods.* 2009;6:377–82.
- scRNA-tools. <http://www.scRNA-tools.org/>.
- Kiselev VY, Kirschner K, Schaub MT, Andrews T, Yu A, Chandra T, et al. SC3: consensus clustering of single-cell RNA-seq data. *Nat Methods.* 2017;14:483–6.
- Lin P, Troup M, Ho JWK. CIDR: ultrafast and accurate clustering through imputation for single-cell RNA-seq data. *Genome Biol.* 2017;18:59.
- Satija R, Farrell JA, Gennert D, Schier AF, Regev A. Spatial reconstruction of single-cell gene expression data. *Nat Biotechnol.* 2015;33:495–502.
- Trapnell C, Cacchiarelli D, Grimsby J, Pokharel P, Li S, Morse M, et al. The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells. *Nat Biotechnol.* 2014;32:381–6.
- DuVerle DA, Yotsukura S, Nomura S, Aburatani H, Tsuda K. Cell Tree: an R/bioconductor package to infer the hierarchical structure of cell populations from single-cell RNA-seq data. *BMC Bioinformatics.* 2016;17:363.
- Juliá M, Telenti A, Rausell A. Sincell: an R/Bioconductor package for statistical assessment of cell-state hierarchies from single-cell RNA-seq. *Bioinformatics.* 2015;31:3380–2.
- Pierson E, Yau C. ZIFA: dimensionality reduction for zero-inflated single-cell gene expression analysis. *Genome Biol.* 2015;16:241.
- Finak G, McDavid A, Yajima M, Deng J, Gorsuk V, Shalek AK, et al. MAST: a flexible statistical framework for assessing transcriptional changes and characterizing heterogeneity in single-cell RNA sequencing data. *Genome Biol.* 2015;16:278.
- Risso D, Perraudeau F, Gribkova S, Dudoit S, Vert J-P. ZINB-Wave: a general and flexible method for signal extraction from single-cell RNA-seq data. 2017. <http://www.biorxiv.org/content/early/2017/04/06/125112>.
- van Dijk D, Nainys J, Sharma R, Kathail P, Carr AJ, Moon KR, et al. MAGIC: a diffusion-based imputation method reveals gene-gene interactions in single-cell RNA-sequencing data. 2017. <http://www.biorxiv.org/content/early/2017/02/25/111591>.
- Huang M, Wang J, Torre E, Dueck H, Shaffer S, Bonasio R, et al. Gene expression recovery for single cell RNA sequencing. 2017. <http://www.biorxiv.org/content/early/2017/05/17/138677>.
- Li WW, Li JJ. sclmpcute: accurate and robust imputation for single cell RNA-Seq data. 2017. <http://www.biorxiv.org/content/early/2017/05/24/141598>.
- McCarthy DJ, Campbell KR, Lun ATL, Wills QF. Scater: pre-processing, quality control, normalization and visualization of single-cell RNA-seq data in R. *Bioinformatics.* 2017;33:1179–86.
- Lun ATL, Bach K, Marioni JC. Pooling across cells to normalize single-cell RNA sequencing data with many zero counts. *Genome Biol.* 2016;17:1–14.
- Lun ATL, Marioni JC. Overcoming confounding plate effects in differential expression analyses of single-cell RNA-seq data. *Biostatistics.* 2017;18:451–64.
- Korthauer KD, Chu L-F, Newton MA, Li Y, Thomson J, Stewart R, et al. A statistical approach for identifying differential distributions in single-cell RNA-seq experiments. *Genome Biol.* 2016;17:222.
- Vallejos CA, Marioni JC, Richardson S. BASICS: Bayesian analysis of single-cell sequencing data. *PLoS Comput Biol.* 2015;11:e1004333.
- Robinson MD, McCarthy DJ, Smyth GK. edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics.* 2010;26:139–40.
- Anders S, Huber W. Differential expression analysis for sequence count data. *Genome Biol.* 2010;11:R106.
- Korthauer K. scDD vignette. 2017. <https://bioconductor.org/packages/release/bioc/vignettes/scDD/inst/doc/scDD.pdf>.
- Vallejos CA, Richardson S, Marioni JC. Beyond comparisons of means: understanding changes in gene expression at the single-cell level. *Genome Biol.* 2016;17:70.
- McCarthy DJ, Chen Y, Smyth GK. Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Res.* 2012;40:4288–97.
- Law CW, Chen Y, Shi W, Smyth GK. voom: Precision weights unlock linear model analysis tools for RNA-seq read counts. *Genome Biol.* 2014;15:R29.
- Tung P-Y, Blischak JD, Hsiao CJ, Knowles DA, Burnett JE, Pritchard JK, et al. Batch effects and the effective design of single-cell gene expression studies. *Sci Rep.* 2017;7:39921.
- Andrews TS, Hemberg M. Modelling dropouts allows for unbiased identification of marker genes in scRNAseq experiments. 2016. <http://www.biorxiv.org/content/early/2016/07/21/065094>.
- Kivioja T, Vähärautio A, Karlsson K, Bonke M, Enge M, Linnarsson S, et al. Counting absolute numbers of molecules using unique molecular identifiers. *Nat Methods.* 2012;9:72–4.
- Akima H. A new method of interpolation and smooth curve fitting based on local procedures. *JACM.* 1970;17:589–602.
- Akima H, Gebhardt A. akima: interpolation of irregularly and regularly spaced data. 2016. <https://CRAN.R-project.org/package=akima>.
- Huber W, Carey JV, Gentleman R, et al. Orchestrating high-throughput genomic analysis with Bioconductor. *Nat Methods.* 2015;12:115–21.
- Phipson B, Zappia L, Oshlack A. Gene length and detection bias in single cell RNA sequencing protocols. *F1000Res.* 2017;6:595.
- Delignette-Muller M, Dutang C. ftdistrplus: an R package for fitting distributions. *J Stat Softw.* 2015;64:1–34.
- Sadedin SP, Pope B, Oshlack A. Bpipe: a tool for running and managing bioinformatics pipelines. *Bioinformatics.* 2012;28:1525–6.
- Dobin A, Davis CA, Schlesinger F, Drenkow J, Zaleski C, Jha S, et al. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics.* 2013;29:15–21.
- Liao Y, Smyth GK, Shi W. featureCounts: an efficient general purpose program for assigning sequence reads to genomic features. *Bioinformatics.* 2014;30:923–30.
- Patro R, Duggal G, Love MI, Irizarry RA, Kingsford C. Salmon provides fast and bias-aware quantification of transcript expression. *Nat Methods.* 2017;14:417–9.
- Soneson C, Love MI, Robinson MD. Differential analyses for RNA-seq: transcript-level estimates improve gene-level inferences. *F1000Res.* 2015;4:1521.
- R Core Team. R: a language and environment for statistical computing. Vienna: R Foundation for Statistical Computing; 2016. <https://www.R-project.org/>.
- Chang F, Qiu W, Zamar R, Lazarus R, Wang X. clues: an R package for nonparametric clustering based on local shrinking. *J Stat Softw.* 2010;33:1–16.
- Wickham H. ggplot2: elegant graphics for data analysis. New York: Springer; 2010.
- Camp JG, Badsha F, Florio M, Kanton S, Gerber T, Wilsch-Bräuning M, et al. Human cerebral organoids recapitulate gene expression programs of fetal neocortex development. *Proc Natl Acad Sci U S A.* 2015;112:15672–7.
- Engel I, Seimois G, Chavez L, Samaniego-Castruita D, White B, Chawla A, et al. Innate-like functions of natural killer T cell subsets result from highly divergent gene programs. *Nat Immunol.* 2016;17:728–39.
- Klein AM, Mazutis L, Akartuna I, Tallapragada N, Veres A, Li V, et al. Droplet barcoding for single-cell transcriptomics applied to embryonic stem cells. *Cell.* 2015;161:1187–201.
- Zeisel A, Muñoz-Manchado AB, Codeluppi S, Lönnberg P, La Manno G, Juréus A, et al. Brain structure. Cell types in the mouse cortex and hippocampus revealed by single-cell RNA-seq. *Science.* 2015;347:1138–42.

Submit your next manuscript to BioMed Central and we will help you at every step:

- We accept pre-submission inquiries
- Our selector tool helps you to find the most relevant journal
- We provide round the clock customer support
- Convenient online submission
- Thorough peer review
- Inclusion in PubMed and all major indexing services
- Maximum visibility for your research

Submit your manuscript at
www.biomedcentral.com/submit



3.3 Updates to Splatter

I have continued to develop the Splatter package since publication, improving the package infrastructure and implementing additional simulation models. This section briefly describes changes to the software since the publication and provides an up-to-date comparison of the current simulation models.

The most significant change to the infrastructure of Splatter is switching from using the SCESet object provided by early versions of the scater package as the output of simulation functions, to using the SingleCellExperiment object [202]. This object has been available since mid-2017 and is intended to be the core object for all Bioconductor packages working with scRNA-seq data. It is based on Bioconductor's SummarizedExperiment object which consists of feature by sample matrices linked to annotation data frames for both features and samples. The SingleCellExperiment object adds features specifically useful for scRNA-seq data, including slots for storing the results of dimensionality reduction techniques and normalisation size factors, as well as conveniently named accessors for information commonly used in analyses. In practical terms, changing objects has not changed how Splatter is used but it does ensure that the results of Splatter simulations are immediately compatible with tools in the Bioconductor ecosystem. Display of the Splatter parameters objects has also been improved, including colouring of output to highlight which parameters have been changed from the default values.

Minor changes have also been made to the Splat simulation model. At the request of users we have modified the parameters controlling additional dropout to allow them to be specified by whole experiment (the previous method), sample batch, cell group or individual cell. This allows users investigating the effects of dropout to simulate datasets where different cell types or batches are differently affected by the dropout phenomenon. Many of the other parameters could already be specified at different levels, but extending this to the dropout parameters provides extra flexibility to the Splat model. In the original model the total number of reads in each cell was sampled from a log-normal distribution, however some users have come across datasets where the normal distribution was a better fit, which caused the Splat estimation procedure to fail. We now allow either the log-normal or normal distributions to be used for this part of the model and the estimation procedure attempts to identify which is more appropriate for each input dataset.

Another model that has seen changes is that from the BASiCS package. The authors of BASiCS have made significant improvements to how they estimate parameters, particularly for datasets without internal spike-ins but where multiple batches exist. BASiCS now uses non-linear regression to capture the trend between gene mean and over-dispersion [110]. This method is now used as part of the default estimation procedure for the BASiCS model within Splatter, resulting in simulations that are a much better match for the original dataset.

I have also added four new simulation models to Splatter, bringing the number of models to 10. These new models are:

- mfa – Simulation of a bifurcating pseudotime trajectory. This simulation can optionally include genes with changes in expression along the trajectory and added dropout [203].
- PhenoPath – Pseudotime trajectory including genes that show differential expression, changes along pseudotime, interaction effects or combinations of these [204].
- ZINB-WaVE – Sophisticated zero-inflated negative-binomial distribution based simulation that includes cell and gene-level covariates [63].
- SparseDC – Simulates a set of clusters across two conditions, where some clusters may be present in only one condition [205].

More details about the models used by these packages are available in the associated publications and package documentation.

3.3.1 Performance of current simulations

To demonstrate the performance of the current simulation models I have repeated some of the analysis from the publication using the Tung dataset [206]. Parameters for each of the models were estimated from a sample of 500 cells from the Tung data and synthetic datasets were produced for each model using these parameters. The time taken for the estimation and simulation steps is shown in [Appendix C.1](#) and versions of the packages used in [Appendix C.2](#). The comparison functions within Splatter were then used to evaluate the differences between the simulated datasets and the real data they were based on. Figure 3.1 summarises this evaluation by showing the ranking of the median absolute deviation (MAD) for each model across a range of metrics.

This comparison clearly shows that the ZINB-WaVE model produces the closest match to this dataset. The similarity to the real dataset suggests that the model used by ZINB-WaVE, and particularly the procedure used to estimate the parameters in it, is effective for scRNA-seq data. ZINB-WaVE was originally designed for dimensionality reduction but also has applications in normalisation and differential expression testing [64]. In contrast to the previously published analysis, the BASiCS model is also a very good match, with the improvement in performance coming from the new regression-based estimation method. Unfortunately, the BASiCS normalisation method relies on knowing predefined groups of cells, which limits its usefulness for exploratory experiments. The Splat method does not match the performance of ZINB-WaVE and BASiCS but still does a reasonable job of reproducing this dataset, despite having a much simpler (and quicker) estimation procedure. While producing realistic datasets is extremely important, the real strength of the Splat model lies in its flexibility to simulate many kinds

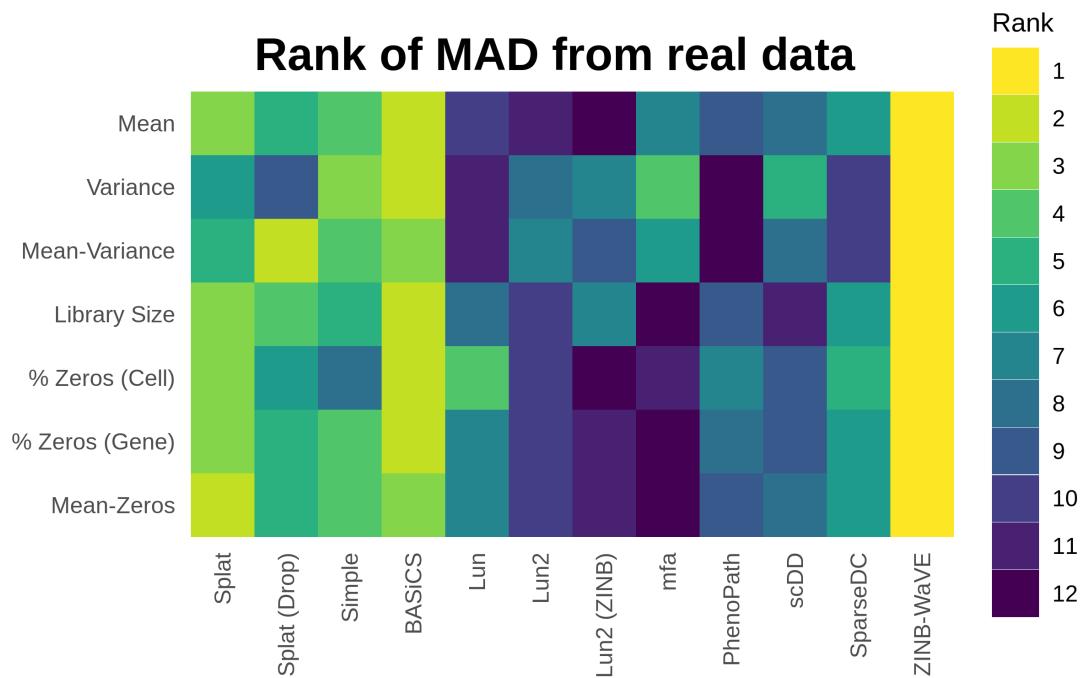


Figure 3.1: Performance of current Splatter simulations. Synthetic datasets were produced using parameters estimated from 500 random cells from the Tung dataset. Plot shows the rank of median absolute deviation (MAD) from the real dataset on a variety of measures from least similar (blue) to most similar (yellow). The ZINB-WaVE model best reproduces this dataset, followed by the updated BASiCS model. The Splat model also performs well despite having a less sophisticated estimation procedure.

of experiment and its reporting of intermediate values that can be used to evaluate methods. As observed in the publication, we again see that the zero-inflated versions of the Splat and Lun2 models perform worse than the regular versions on this UMI dataset. However, this is in contrast to the extremely good performance of ZINB-WaVE which also uses a zero-inflated model. Taken together these results suggest that both regular and zero-inflated models can be effective for this type of data, if the parameters are estimated correctly.

The other three new simulation models (mfa, PhenoPath and SparseDC) do not do a particularly good job of reproducing this dataset, and in particular mfa produces a dataset with a very different number of counts per cell. These simulations have mostly been designed to demonstrate the analysis methods in these packages and the estimation procedures associated with them are relatively simple. This result underlines that the key to producing a realistic simulation of any kind of data lies not so much in the simulation model itself but in choosing good parameters for that model. As we describe in the publication, there are caveats around the estimation procedures for some of the other models, specifically the Lun model, which only estimates the number of genes and cells, and the scDD model, which is designed for a high-quality filtered dataset unlike the raw dataset used here. Another important consideration is that the comparison shown here is based on a single dataset. As we showed in the publication, different datasets have different biological and technical features which may be reproduced more accurately by alternative simulation models.

Comparing simulations and real datasets is a difficult task. For the comparisons in the Splatter publication we focused on simple summary statistics such as mean gene expression or percentage of zeros per cell and the relationships between them. However, there are other measures that could be considered such as correlation between features or estimated dispersions. Statistical tests can also be used to formally evaluate the similarity of datasets. The `countsimQC` package [207] can be used to produce a comprehensive report comparing count datasets in a variety of ways and could be a useful tool for any future evaluations of simulation models.

4

Visualising clustering across resolutions

“Don’t do what you can’t undo, until you’ve considered what you can’t do once you’ve done it.”

— Robin Hobb

Assassin’s Apprentice, 1995

4. Clustering Trees

How many clusters to use?

A tree of clusters!

1. Cluster at multiple resolutions

2. Calculate overlap between clusters

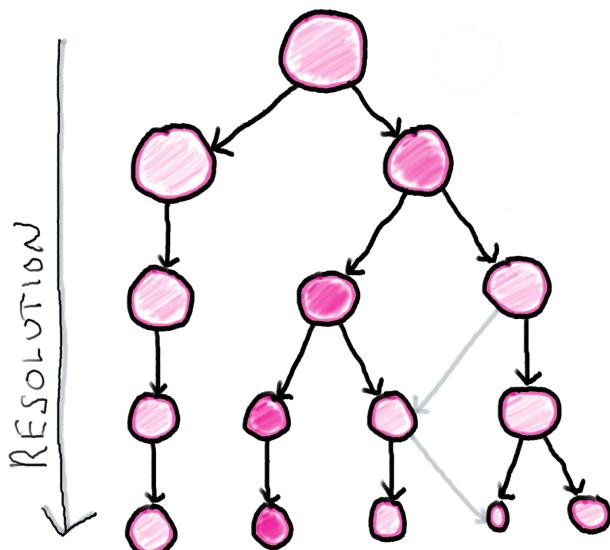
3. Build a graph

4. Weight edges

5. Visualise



$$\text{in prop} = \frac{\text{edge samples}}{\text{high res samples}}$$



Structure can show stability

Show information across resolutions

4.1 Introduction

Clustering of cells to form groups is a common task when analysing scRNA-seq data. This is different from analysis of bulk RNA-seq datasets where the groups are usually known in advance and clustering isn't required. Because clustering is a new step in analysis of scRNA-seq data it has received a lot of attention from scRNA-seq methods developers. The need to group samples is not unique to genomic data and clustering techniques are used in many other fields for a wide variety of purposes. Whatever kind of data is of interest, and whatever clustering method is being used, a question that commonly comes up is how many clusters do we want to have? Depending on the clustering method, the number of clusters can be controlled by setting an exact value or by changing a parameter that controls the clustering resolution. The number of clusters can also be indirectly affected by the values of related parameters. The number of clusters that are used in an analysis can often have a profound affect on how the results are interpreted. Existing measures of clustering typically only consider a single clustering resolution at a time or require multiple rounds of permutations and clustering which can be infeasible for large datasets. In this chapter I propose an alternative, visualisation-based aid for deciding which clustering resolution to use.

Clusterings of the same dataset at different resolutions are often related and it is common for new clusters formed at higher resolutions to form by splitting existing clusters. However, when comparing clusterings it is not always clear what those relationships are and how significant they might be. The method I describe here was published in *GigaScience* and proposes clustering datasets at multiple resolutions and considering the overlap in samples between clusters at neighbouring resolutions [208]. By doing this we can build a graph structure we call a “clustering tree”. Visualising this tree allows us to see where new clusters form, how they are related and the stability of particular clustering resolutions. In the publication we demonstrate this approach using simulated datasets, a simple dataset commonly used as an example for machine learning techniques and a complex scRNA-seq dataset from blood.

While the structure of clustering trees can help choose a clustering resolution to use for an analysis, clustering trees are more generally a compact, information-dense visualisation that can show information across clustering resolutions. This is something that is not possible with traditional visualisations used for clustering results such as t-SNE projections, and is achieved by trading individual information about each sample for summarised information about clusters and adding a resolution dimension. Overlaying important domain knowledge (such as the expression of known marker genes) onto these visualisations can be particularly informative and we also demonstrate this in our paper (**Section 4.2**). Section 4.3 demonstrates how clustering trees can be combined with other visualisations of scRNA-seq data.

Clustering trees can be produced using the `clustree` R package which is built on the `tidygraph` [209], `igraph` [210] and `ggraph` [211] packages and is available from CRAN (<https://cran.r-project.org/package=clustree>). The `clustree` vignette is included as **Appendix D.2** and the manual as **Appendix D.2**.

4.2 Clustering trees publication

GigaScience, 7, 2018, 1–9
doi: 10.1093/gigascience/giy083
Advance Access Publication Date: 11 July 2018
Research

RESEARCH

Clustering trees: a visualization for evaluating clusterings at multiple resolutions

Luke Zappia  1,2 and Alicia Oshlack  1,2,*

¹Bioinformatics, Murdoch Children's Research Institute, Flemington Road, Parkville, Victoria 3052, Australia and ²School of Biosciences, Faculty of Science, The University of Melbourne, Parkville, Victoria 3052, Australia

*Correspondence address. Alicia Oshlack, E-mail: alicia.oshlack@mcri.edu.au  <http://orcid.org/0000-0001-9788-5690> Address: Alicia Oshlack, Murdoch Children's Research Institute, Flemington Road, Parkville, Victoria 3052, Australia

Abstract

Clustering techniques are widely used in the analysis of large datasets to group together samples with similar properties. For example, clustering is often used in the field of single-cell RNA-sequencing in order to identify different cell types present in a tissue sample. There are many algorithms for performing clustering, and the results can vary substantially. In particular, the number of groups present in a dataset is often unknown, and the number of clusters identified by an algorithm can change based on the parameters used. To explore and examine the impact of varying clustering resolution, we present clustering trees. This visualization shows the relationships between clusters at multiple resolutions, allowing researchers to see how samples move as the number of clusters increases. In addition, meta-information can be overlaid on the tree to inform the choice of resolution and guide in identification of clusters. We illustrate the features of clustering trees using a series of simulations as well as two real examples, the classical iris dataset and a complex single-cell RNA-sequencing dataset. Clustering trees can be produced using the clustree R package, available from CRAN and developed on GitHub.

Keywords: clustering; visualization; scRNA-seq

Introduction

Clustering analysis is commonly used to group similar samples across a diverse range of applications. Typically, the goal of clustering is to form groups of samples that are more similar to each other than to samples in other groups. While fuzzy or soft clustering approaches assign each sample to every cluster with some probability, and hierarchical clustering forms a tree of samples, most methods form hard clusters where each sample is assigned to a single group. This goal can be achieved in a variety of ways, such as by considering the distances between samples (e.g., k-means [1–3], PAM [4]), areas of density across the dataset (e.g., DBSCAN [5]), or relationships to statistical distributions [6].

In many cases, the number of groups that should be present in a dataset is not known in advance, and deciding the correct number of clusters to use is a significant challenge. For some algorithms, such as k-means clustering, the number of clusters must be explicitly provided. Other methods have parameters that, directly or indirectly, control the clustering resolution and therefore the number of clusters produced. While there are methods and statistics (such as the elbow method [7] and silhouette plots [8]) designed to help analysts decide which clustering resolution to use, they typically produce a single score that only considers a single set of samples or clusters at a time.

An alternative approach would be to consider clusterings at multiple resolutions and examine how samples change groupings as the number of clusters increases. This has led to a range of cluster stability measures [9], many of which rely on clustering of perturbed or subsampled datasets. For example, the model explorer algorithm subsamples a dataset multiple times, clusters each subsampled dataset at various resolutions, and then calculates a similarity between clusterings at the same resolution to give a distribution of similarities that can inform the

Received: 7 March 2018; Revised: 21 May 2018; Accepted: 27 June 2018
© The Author(s) 2018. Published by Oxford University Press. This is an Open Access article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted reuse, distribution, and reproduction in any medium, provided the original work is properly cited.

2 | Clustering trees: a visualization for evaluating clusterings

choice of resolution [10]. One cluster stability measure that is not based on perturbations is that contained in the SC3 package for clustering single-cell RNA-seqencing (scRNA-seq) data [11]. Starting with a set of cluster labels at different resolutions, each cluster is scored, with clusters awarded increased stability if they share the same samples as a cluster at another resolution but penalized for being at a higher resolution.

A similar simple approach is taken by the clustering tree visualization we present here, without calculating scores: (i) a dataset is clustered using any hard clustering algorithm at multiple resolutions, producing sets of cluster nodes; (ii) the overlap between clusters at adjacent resolutions is used to build edges; and (iii) the resulting graph is presented as a tree. This tree can be used to examine how clusters are related to each other—which clusters are distinct and which are unstable. In the following sections, we describe how we construct such a tree and present examples of trees built from a classic clustering dataset and a complex scRNA-seq dataset. The figures shown here can be produced in R using our publicly available clustree package. Although clustering trees cannot directly provide a clustering resolution to use, they can be a useful tool for exploring and visualizing the range of possible choices.

Building a Clustering Tree

To build a clustering tree, we start with a set of clusterings and allocate samples to groups at several different resolutions. These could be produced using any hard-clustering algorithm that allows control of the number of clusters in some way. For example, this could be a set of samples clustered using k-means with $k = 1, 2, 3$ as shown in Fig. 1. We sort these clusterings so that they are ordered by increasing resolution (k), then consider pairs of adjacent clusterings. Each cluster $c_{k,i}$ (where $i = 1, \dots, n$ and n is the number of clusters at resolution k) is compared with each cluster $c_{k+1,j}$ (where $j = 1, \dots, m$ and m is the number of clusters at resolution $k+1$). The overlap between the two clusters is computed as the number of samples that are assigned to both $c_{k,i}$ and $c_{k+1,j}$. Next, we build a graph where each node is a cluster and each edge is an overlap between two clusters. While we refer to this graph as a tree for simplicity, it can more correctly be described as a polytree, a special case of a directed acyclic graph where the underlying undirected graph is a tree [12].

Many of the edges will be empty, e.g., in Fig. 1 no samples in cluster A at $k = 2$ end up in cluster B at $k = 3$. In some datasets there may also be edges that contain few samples. These edges are not informative and result in a cluttered tree. An obvious solution for removing uninformative, low-count edges is to filter them using a threshold on the number of samples they represent. However, in this case, the count of samples is not the correct statistic to use because it favors edges at lower resolutions and those connecting larger clusters. Instead, we define the in-proportion metric as the ratio between the number of samples on the edge and the number of samples in the cluster it goes toward. This metric shows the importance of the edge to the higher-resolution cluster independently of the cluster size. We can then apply a threshold to the in-proportion in order to remove less-informative edges.

The final graph can then be visualized. In theory, any graph layout algorithm could be used. However, for the clustree package, we have made use of the two algorithms specifically designed for tree structures available in the igraph package [13]. These are the Reingold-Tilford tree layout, which places par-

ent nodes above their children [14], and the Sugiyama layout, which places nodes of a directed acyclic graph in layers while minimizing the number of crossing edges [15]. Both of these algorithms can produce attractive layouts; as such, we have not found the need to design a specific layout algorithm for clustering trees. By default, the clustree package uses only a subset of edges when constructing a layout, specifically the highest in-proportion edges for each node. We have found that this often leads to more interpretable visualizations; however, users can choose to use all edges if desired.

Regardless of the layout used, the final visualization places the cluster nodes in a series of layers where each layer is a different clustering resolution and edges show the transition of samples through those resolutions. Edges are colored according to the number of samples they represent, and the in-proportion metric is used to control the edge transparency, highlighting more important edges. By default, the node size is adjusted according to the number of samples in the cluster, and their color indicates the clustering resolution. The clustree package also includes options for controlling the aesthetics of nodes based on the attributes of samples in the clusters they represent, as shown in the following examples.

While a clustering tree is conceptually similar to the tree produced through hierarchical clustering, there are some important differences. The most obvious are that a hierarchical clustering tree is the result of a particular clustering algorithm and shows the relationships between individual samples, while the clustering trees described here are independent of clustering method and show relationships between clusters. The branches of a hierarchical tree show how the clustering algorithm has merged samples. In contrast, edges in a clustering tree show how samples move between clusters as the resolution changes and nodes may have multiple parents. While it is possible to overlay information about samples on a hierarchical tree, this is not commonly done but is a key feature of the clustree package and how clustering trees could be used in practice.

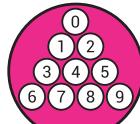
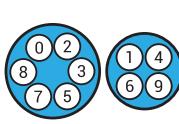
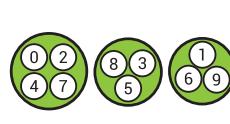
A Demonstration Using Simulations

To demonstrate what a clustering tree can look like in different situations and how it behaves as a dataset is overclustered, we present some illustrative examples using simple simulations (see Methods). We present five scenarios: random uniform noise (simulation A), a single cluster (simulation B), two clusters (simulation C), three clusters (simulation D), and four clusters (simulation E). Each cluster consists of 1,000 samples (points) generated from a 100-dimensional normal distribution, and each synthetic dataset has been clustered using k-means clustering with $k = 1, \dots, 8$. We then use the clustree package to produce clustering trees for each dataset (Fig. 2).

Looking at the first two examples (uniform noise [Fig. 2A] and a single cluster [Fig. 2B]), we can clearly see how a clustering tree behaves when a clustering algorithm returns more clusters than are truly present in a dataset. New clusters begin to form from multiple existing clusters, and many samples switch between branches of the tree, resulting in low in-proportion edges. Unstable clusters may also appear and then disappear as the resolution increases, as seen in Fig. 2E. As we add more structure to the datasets, the clustering trees begin to form clear branches and low in-proportion edges tend to be confined to sections of the tree. By looking at which clusters are stable and where low in-proportion edges arise, we can infer which areas of the tree

Zappia and Oshlack | 3

1. Cluster at multiple resolutions

k = 1***k = 2******k = 3***

2. Find overlaps and calculate in-proportion

$$\text{Overlap} = \frac{0, 2, 3}{0, 2, 3, 5, 7, 8} = \frac{6}{6} = 1.00$$

$$\text{Overlap} = \frac{1, 4}{1, 4, 6, 9} = \frac{4}{4} = 1.00$$

$$\text{Overlap} = \frac{4}{1, 4, 6, 9} = \frac{1}{4} = 0.25$$

$$\text{Overlap} = \frac{0}{1, 4, 6, 9} = \frac{0}{3} = 0.00$$

$$\text{Overlap} = \frac{1}{1, 4, 6, 9} = \frac{3}{3} = 1.00$$

Overlap

$$\text{In-proportion} = \frac{0, 2, 7}{0, 2, 4, 7} = \frac{3}{4} = 0.75$$

$$\text{Overlap} = \frac{3}{0, 2, 3, 5} = \frac{3}{3} = 1.00$$

$$\text{Overlap} = \frac{0}{0, 2, 3, 5} = \frac{0}{3} = 0.00$$

3. Filter edges and visualize tree

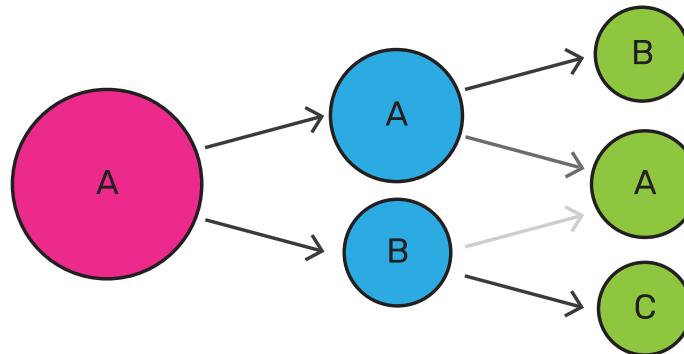


Figure 1: Illustration of the steps required to build a clustering tree. First, a dataset must be clustered at different resolutions. The overlap in samples between clusters at adjacent resolutions is computed and used to calculate the in-proportion for each edge. Finally, the edges are filtered and the graph visualized as a tree.

are likely to be the result of true clusters and which are caused by overclustering.

The second clustering tree for each dataset shows nodes colored according to the SC3 stability index for each cluster. As we

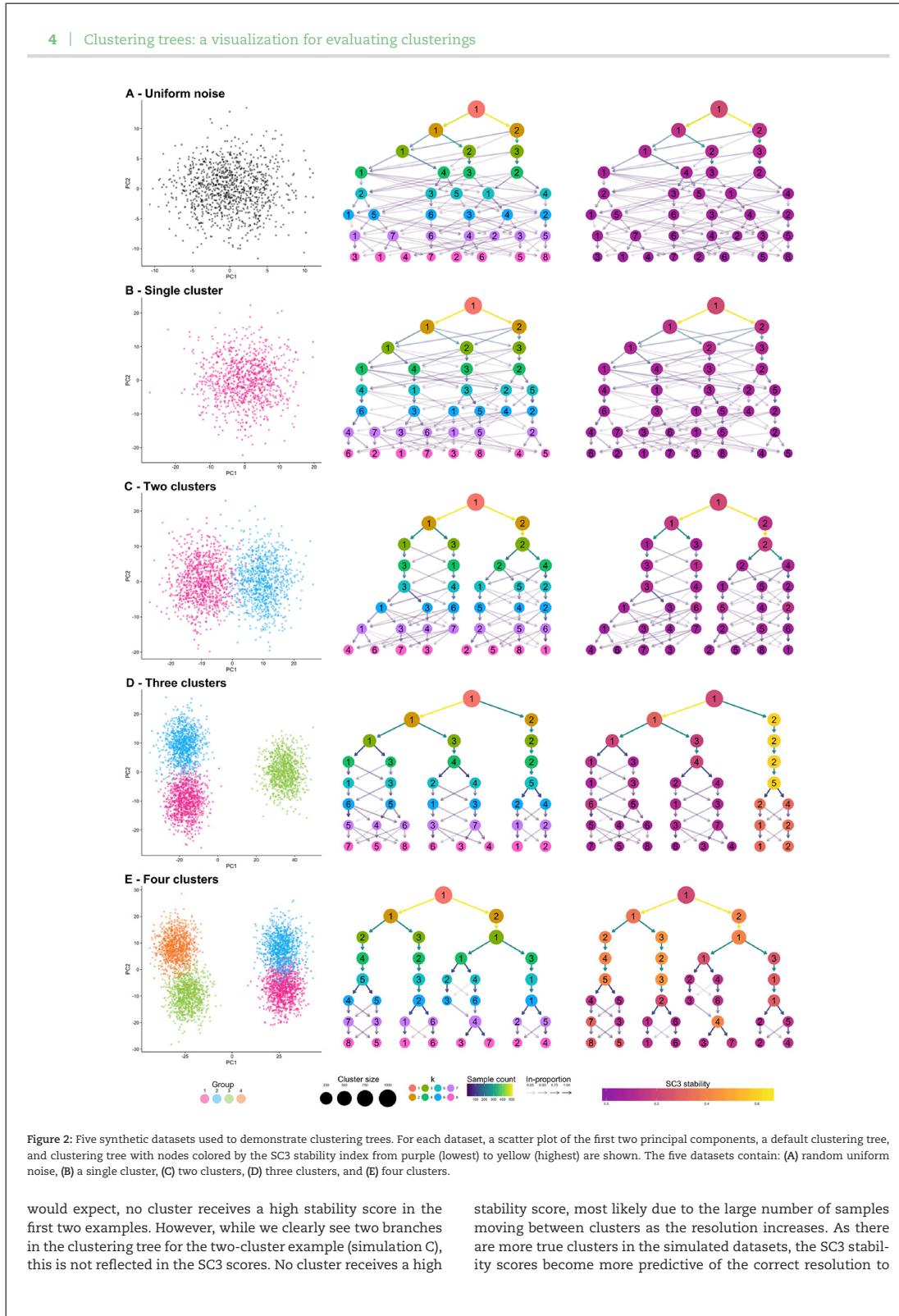


Figure 2: Five synthetic datasets used to demonstrate clustering trees. For each dataset, a scatter plot of the first two principal components, a default clustering tree, and clustering tree with nodes colored by the SC3 stability index from purple (lowest) to yellow (highest) are shown. The five datasets contain: (A) random uniform noise, (B) a single cluster, (C) two clusters, (D) three clusters, and (E) four clusters.

would expect, no cluster receives a high stability score in the first two examples. However, while we clearly see two branches in the clustering tree for the two-cluster example (simulation C), this is not reflected in the SC3 scores. No cluster receives a high

stability score, most likely due to the large number of samples moving between clusters as the resolution increases. As there are more true clusters in the simulated datasets, the SC3 stability scores become more predictive of the correct resolution to

use. However, it is important to look at the stability scores of all clusters at a particular resolution as taking the highest individual cluster stability score could lead to the incorrect resolution being used, as can be seen in the four-cluster example (simulation E). These examples show how clustering trees can be used to display existing clustering metrics in a way that can help to inform parameter choices.

A Simple Example

To further illustrate how a clustering tree is built, we will work through an example using the classic iris dataset [16, 17]. This dataset contains measurements of the sepal length, sepal width, petal length, and petal width from 150 iris flowers, 50 from each of three species: *Iris setosa*, *Iris versicolor*, and *Iris virginica*. The iris dataset is commonly used as an example for both clustering and classification problems with the *I. setosa* samples being significantly different from, and linearly separable from, the other samples. We have clustered this dataset using k-means clustering with $k = 1, \dots, 5$ and produced the clustering tree shown in Fig. 3A.

We see that one branch of the tree is clearly distinct (presumably representing *I. setosa*), remaining unchanged regardless of the number of clusters. On the other side, we see that the cluster at $k = 2$ cleanly splits into two clusters (presumably *I. versicolor* and *I. virginica*) at $k = 3$. However, as we move to $k = 4$ and $k = 5$, we see clusters being formed from multiple branches with more low-in-proportion edges. As we have seen in the simulated examples, this kind of pattern can indicate that the data have become overclustered and we have begun to introduce artificial groupings.

We can check our assumption that the distinct branch represents the *I. setosa* samples and that the other two clusters at $k = 3$ are *I. versicolor* and *I. virginica* by overlaying some known information about the samples. In Fig. 3B we have colored the nodes by the mean petal length of the samples they contain. We can now see that clusters in the distinct branch have the shortest petals, with cluster 1 at $k = 3$ having an intermediate length and cluster 3 having the longest petals. This feature is known to separate the samples into the expected species, with *I. setosa* having the shortest petals on average, *I. versicolor* an intermediate length, and *I. virginica* the longest.

Although this is a very simple example, it highlights some of the benefits of viewing a clustering tree. We get some indication of the correct clustering resolution by examining the edges, and we can overlay known information to assess the quality of the clustering. For example, if we observed that all clusters had the same mean petal length, it would suggest that the clustering has not been successful as we know this is an important feature that separates the species. We could potentially learn more by looking at which samples follow low-proportion edges or by overlaying a series of features to try and understand what causes particular clusters to split.

Clustering Trees for scRNA-seq Data

One field that has begun to make heavy use of clustering techniques is the analysis of scRNA-seq data. scRNA-seq is a recently developed technology that can measure how genes are expressed in thousands to millions of individual cells [18]. This technology has been rapidly adopted in fields such as developmental biology and immunology where it is valuable to have information from single cells rather than measurements that are

averaged across the many different cells in a sample using older RNA-seq technologies. A key use for scRNA-seq is to discover and interrogate the different cell types present in a sample of a complex tissue. In this situation, clustering is typically used to group similar cells based on their gene expression profiles. Differences in gene expression between groups can then be used to infer the identity or function of those cells [19]. The number of cell types (clusters) in an scRNA-seq dataset can vary depending on factors such as the tissue being studied, its developmental or environmental state, and the number of cells captured. Often, the number of cell types is not known before the data are generated, and some samples can contain dozens of clusters. Therefore, deciding which clustering resolution to use is an important consideration in this application.

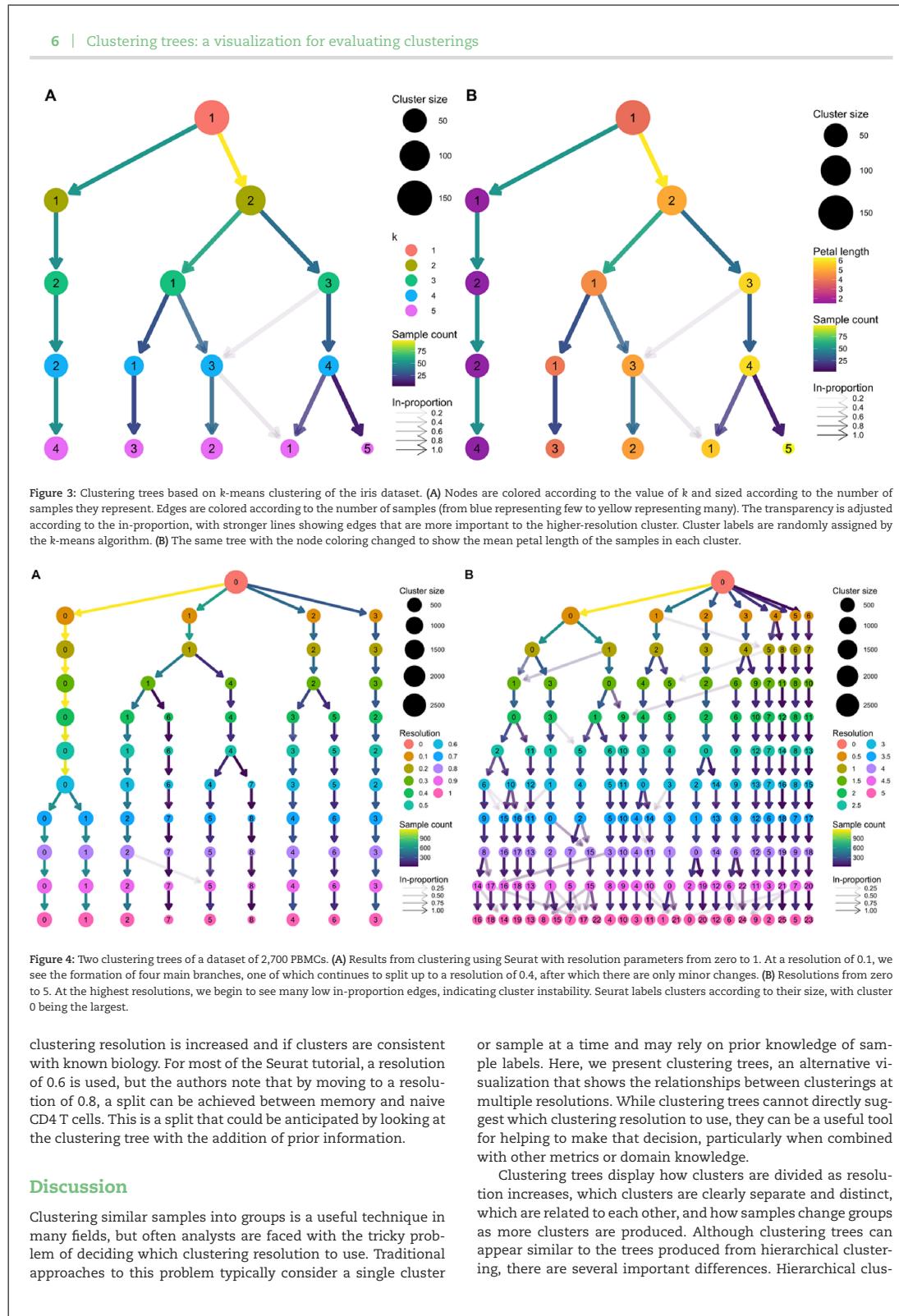
As an example of how clustering trees can be used in the scRNA-seq context, we consider a commonly used peripheral blood mononuclear cell (PBMC) dataset. This dataset was originally produced by 10x Genomics and contains 2,700 peripheral blood mononuclear cells, representing a range of well-studied immune cell types [20]. We analyzed this dataset using the Seurat package [21], a commonly used toolkit for scRNA-seq analysis, following the instructions in their tutorial with the exception of varying the clustering resolution parameter from zero to 5 (see Methods). Seurat uses a graph-based clustering algorithm, and the resolution parameter controls the partitioning of this graph, with higher values resulting in more clusters. The clustering trees produced from this analysis are shown in Fig. 4.

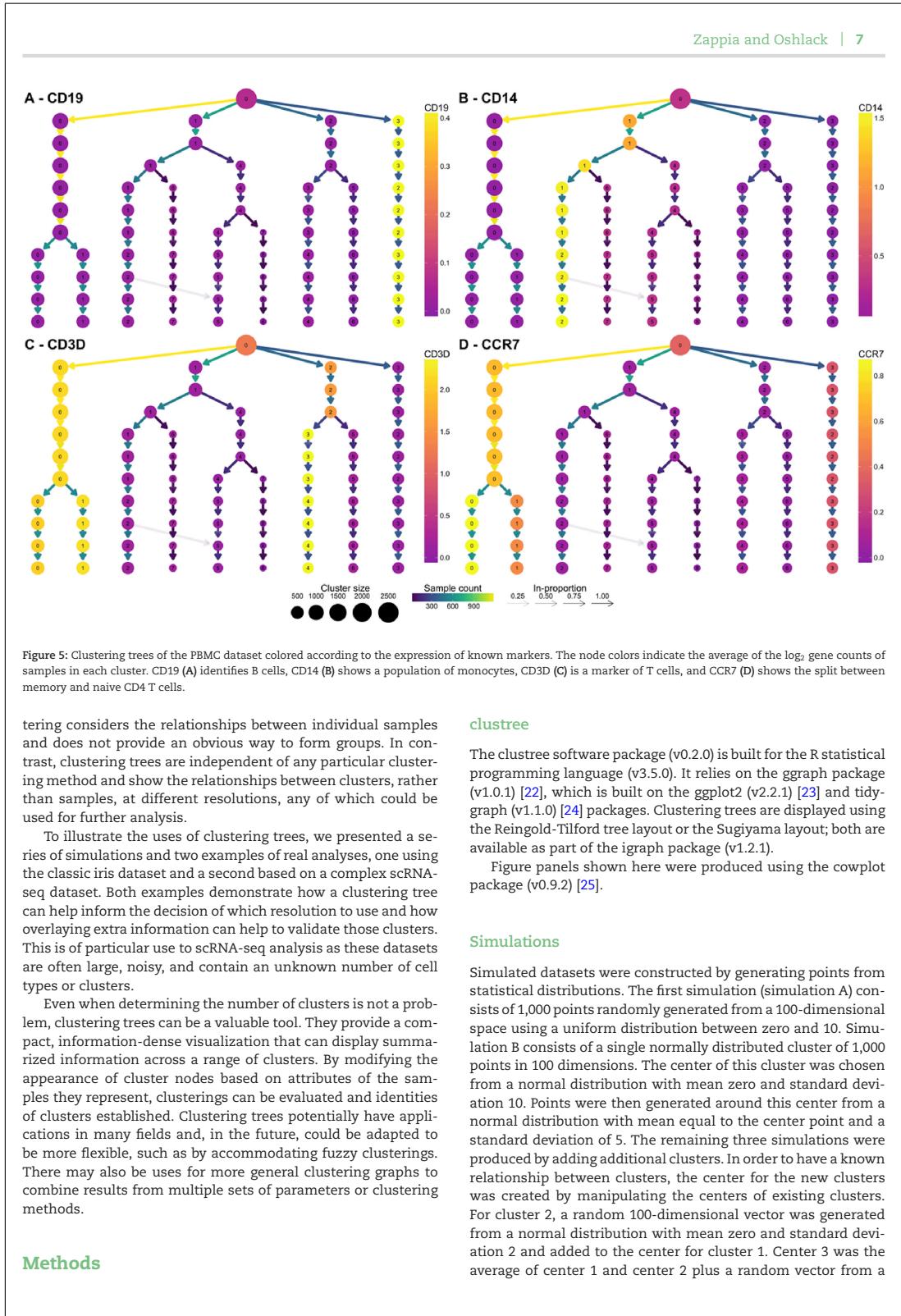
The clustering tree covering resolutions zero to 1 in steps of 0.1 (Fig. 4A) shows that four main branches form at a resolution of just 0.1. One of these branches, starting with cluster 3 at resolution 0.1, remains unchanged, while the branch starting with cluster 2 splits only once at a resolution of 0.4. Most of the branching occurs in the branch starting with cluster 1, which consistently has subbranches split off to form new clusters as the resolution increases. There are two regions of stability in this tree—at resolution 0.4–0.5 and resolution 0.7–1.0 where the branch starting at cluster 0 splits in two.

Fig. 4B shows a clustering tree with a greater range of resolutions, from zero to 5 in steps of 0.5. By looking across this range, we can see what happens when the algorithm is forced to produce more clusters than are likely to be truly present in this dataset. As overclustering occurs, we begin to see more low-in-proportion edges and new clusters forming from multiple parent clusters. This suggests that those areas of the tree are unstable and that the new clusters being formed are unlikely to represent true groups in the dataset.

Known marker genes are commonly used to identify the cell types that specific clusters correspond to. Overlaying gene expression information onto a clustering tree provides an alternative view that can help to indicate when clusters containing pure cell populations are formed. Figure 5 shows the PBMC clustering tree in Fig. 4A overlaid with the expression of some known marker genes.

By adding this extra information, we can quickly identify some of the cell types. CD19 (Fig. 5A) is a marker of B cells and is clearly expressed in the most distinct branch of the tree. CD14 (Fig. 5B) is a marker of a type of monocyte, which becomes more expressed as we follow one of the central branches, allowing us to see which resolution identifies a pure population of these cells. CD3D (Fig. 5C) is a general marker of T cells and is expressed in two separate branches, one that splits into low and high expression of CCR7 (Fig. 5D), separating memory and naive CD4 T cells. By adding expression of known genes to a clustering tree, we can see if more populations can be identified as the





8 | Clustering trees: a visualization for evaluating clusterings

normal distribution with mean zero and standard deviation 5. To ensure a similar relationship between clusters 3 and 4 as between clusters 1 and 2, center 4 was produced by adding half the vector used to produce center 2 to center 3 plus another vector from a normal distribution with mean zero and standard deviation 2. Points for each cluster were generated in the same way as for cluster 1. Simulation C consists of the points in clusters 1 and 2; simulation D consists of clusters 1, 2, and 3; and simulation E consists of clusters 1, 2, 3, and 4. Each simulated dataset was clustered using the “kmeans” function in the stats package with values of k from 1 to 8, a maximum of 100 iterations, and 10 random starting positions. The clustering tree visualizations were produced using the clustree package with the tree layout. The simulated datasets and the code used to produce them are available from the repository for this article [26].

Iris dataset

The iris dataset is available as part of R. We clustered this dataset using the “kmeans” function in the stats package with values of k from 1 to 5. Each value of k was clustered with a maximum of 100 iterations and with 10 random starting positions. The clustree package was used to visualize the results using the Sugiyama layout. The clustered iris dataset is available as part of the clustree package.

PBMC dataset

The PBMC dataset was downloaded from the Seurat tutorial page [27], and this tutorial was followed for most of the analysis using Seurat version 2.3.1. Briefly, cells were filtered based on the number of genes they express and the percentage of counts assigned to mitochondrial genes. The data were then log-normalized and 1,838 variable genes identified. Potential confounding variables (number of unique molecular identifiers and percentage mitochondrial expression) were regressed from the dataset before performing principal component analysis on the identified variable genes. The first 10 principal components were then used to build a graph that was partitioned into clusters using Louvain modularity optimization [28] with resolution parameters in the range zero to 5, in steps of 0.1 between zero and 1, and then in steps of 0.5. Clustree was then used to visualize the results using the tree layout.

Availability of source code and requirements

Project name: clustree.

Project home page: <https://github.com/lazappi/clustree>.

Operating systems(s): Linux, MacOS, Windows

Programming language: R ($>= 3.4$)

Other requirements: None

License: GPL-3

Any restrictions to use by non-academics: None

[RRID:SCR_016293](#)

Availability of supporting data

The clustree package is available from CRAN [29] and is being developed on GitHub [30]. The code and datasets used for the analysis presented here are also available from GitHub [26]. The clustered iris dataset is included as part of clustree, and the PBMC dataset can be downloaded from the Seurat tutorial page [27] or the paper GitHub repository. Snapshots of the code are available in the GigaScience repository, GigaDB [31].

Abbreviations

PBMC: peripheral blood mononuclear cell; scRNA-seq: single-cell RNA-sequencing.

Competing interests

The authors declare that they have no competing interests.

Funding

L.Z. is supported by an Australian Government Research Training Program scholarship. A.O. is supported through a National Health and Medical Research Council Career Development fellowship (APP1126157). The Murdoch Children’s Research Institute is supported by the Victorian Government’s Operational Infrastructure Support Program.

Author contributions

L.Z. designed the clustering tree algorithm, wrote the clustree software package, and drafted the manuscript. A.O. supervised the project and commented on the manuscript.

Acknowledgements

Thank you to Marek Cmero for providing comments on a draft of the manuscript and the reviewers for their comments and suggestions.

References

- Forgy WE. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics* 1965;21:768–9.
- Macqueen J. Some methods for classification and analysis of multivariate observations. In 5th Berkeley Symposium on Mathematical Statistics and Probability, 1967.
- Lloyd S. Least squares quantization in PCM. *IEEE Trans Inf Theory* 1982;28:129–37.
- Kaufman L, Rousseeuw PJ. Partitioning Around Medoids (Program PAM). Finding Groups in Data, New Jersey, USA. John Wiley & Sons, Inc. 1990. pp. 68–125.
- Ester M, Kriegel H-P, Sander J, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining. Portland, Oregon: AAAI Press; 1996. pp. 226–31. Available from:
- Fraley C, Raftery AE. Model-based clustering, discriminant analysis, and density estimation. *J Am Stat Assoc* 2002;97:611–31.
- Thorndike RL. Who belongs in the family? *Psychometrika* 1953;18:267–76.
- Rousseeuw PJ. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *J Comput Appl Math* 1987;20:53–65.
- Luxburg U von. Clustering stability: an overview. *Foundations and Trends in Machine Learning* 2010;2:235–74.
- Ben-Hur A, Elisseeff A, Guyon I. A stability based method for discovering structure in clustered data. *Pac Symp Biocomput* 2002, 7:6–17.
- Kiselev VY, Kirschner K, Schaub MT, et al. SC3: consensus clustering of single-cell RNA-seq data. *Nat Methods* 2017;14:483–6.

12. Rebane G, Pearl J. The Recovery of Causal Poly-Trees from Statistical Data. 2013; Available from: <http://arxiv.org/abs/1304.2736>, Accessed May 16, 2018.
13. Csardi G, Nepusz T. The igraph software package for complex network research. *Inter Journal Complex Systems* 2006;**1695**:1–9.
14. Reingold EM, Tilford JS. Tidier drawings of trees. *IEEE Trans Software Eng* 1981;**SE-7**:223–8.
15. Sugiyama K, Tagawa S, Toda M. Methods for visual understanding of hierarchical system structures. *IEEE Trans Syst Man Cybern* 1981;**11**:109–25.
16. Anderson E. The irises of the Gaspe Peninsula. *Bulletin of the American Iris Society* 1935;**59**:2–5.
17. Fisher RA. The use of multiple measurements in taxonomic problems. *Ann Eugen* 1936;**7**:179–88.
18. Tang F, Barbacioru C, Wang Y, et al. mRNA-seq whole-transcriptome analysis of a single cell. *Nat Methods* 2009;**6**:377–82.
19. Stegle O, Teichmann SA, Marioni JC. Computational and analytical challenges in single-cell transcriptomics. *Nat Rev Genet* 2015;**16**:133–45.
20. Zheng GXY, Terry JM, Belgrader P, et al. Massively parallel digital transcriptional profiling of single cells. *Nat Commun* 2017;**8**:14049.
21. Satija R, Farrell JA, Gennert D, et al. Spatial reconstruction of single-cell gene expression data. *Nat Biotechnol* 2015;**33**:495–502.
22. Pedersen TL. ggraph: An Implementation of Grammar of Graphics for Graphs and Networks. 2018. Available from: <https://CRAN.R-project.org/package=ggraph>, Accessed 21 May, 2018
23. Wickham H. ggplot2: Elegant Graphics for Data Analysis. New York: Springer; 2010.
24. Pedersen TL. tidygraph: A Tidy API for Graph Manipulation. 2018. Available from: <https://CRAN.R-project.org/package=tidygraph>, Accessed May 21, 2018
25. Wilke CO. cowplot: Streamlined Plot Theme and Plot Annotations for 'ggplot2.' 2018. Available from: <https://CRAN.R-project.org/package=cowplot>, Accessed May 21, 2018
26. Zappia L, Oshlack A. clustree GitHub repository, 2018. Available from: <https://github.com/Oshlack/clustree-paper>, Accessed May 21, 2018.
27. Satija Lab. Seurat PBMC3K Tutorial. ; 2018. Available from: https://satijalab.org/seurat/pbmc3k_tutorial.html, Accessed May 21, 2018
28. Blondel VD, Guillaume J-L, Lambiotte R, et al. Fast unfolding of communities in large networks. *J Stat Mech. IOP Publishing*; 2008;**2008**:P10008.
29. Zappia L, Oshlack A. clustree: Visualise Clusterings at Different Resolutions. 2018. Available from: <https://CRAN.R-project.org/package=clustree>, Accessed May 21, 2018
30. Zappia L, Oshlack A. clustree GitHub repository, ; 2018. Available from: <https://github.com/lazappi/clustree>, Accessed May 21, 2018.
31. Zappia L, Oshlack A. Supporting data for "Clustering trees: a visualization for evaluating clusterings at multiple resolutions." GigaScience Database 2018. <http://dx.doi.org/10.5524/100478>.

4.3 Overlaying clustering trees

In the clustering trees publication we focused on the use of the visualisation as a tool for selecting a clustering resolution, but more generally the clustering tree structure can be used to display information across the resolution dimension. The visualisation that is most commonly used to display scRNA-seq data is the dimensionality reduction plot produced using methods such as PCA, t-SNE [147] or UMAP [148]. In this visualisation each point is a cell placed in a two-dimensional space and coloured using factors such as the assigned cluster or the expression level of a gene. The two visualisations display complementary information. The dimensionality reduction plot shows the location of each individual sample in a low-dimensional space, while the clustering tree trades that sample level resolution for a summarised view of each cluster and swaps the spatial information for a new resolution dimension. It is possible to combine these two approaches by overlaying a clustering tree onto a dimensionality reduction plot. Figure 4.1 shows an example of this using the iris dataset [212,213].

Here instead of using a tree layout algorithm to decide where the cluster nodes are placed, we place them in the low-dimensional space at the mean position of the samples in each cluster. The easiest way to view this overlaid plot (Figure 4.1A) is to imagine looking down on a landscape from above. On the ground we have the individual sample points (here coloured by their highest resolution cluster) and rising up from that we have the clustering tree with the highest resolutions placed closest to the ground and the lowest resolution at the top. As we step down the clustering tree we can see where in this space new clusters are formed.

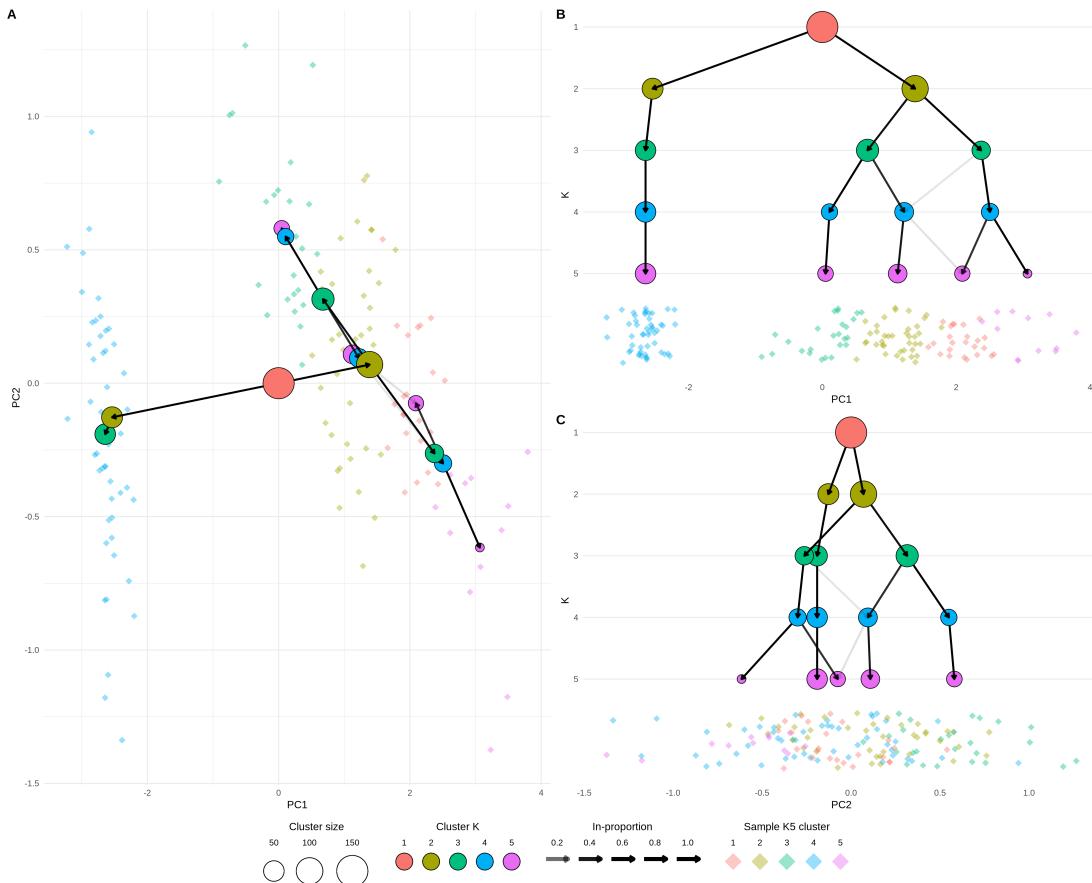


Figure 4.1: Clustering tree of the iris dataset overlaid on a PCA plot. (A) A PCA plot of the samples in the iris dataset coloured by cluster at $k = 5$ overlaid with a clustering tree showing clustering resolutions from $k = 1$ to $k = 5$. (B) The same plot shown from the direction of the x-axis (PC1), the y-axis now shows the clustering resolution and samples are jittered along the x-axis. (C) Same as (B) but from the direction of the y-axis (PC2).

Figures 4.1 B and C show this same view but from a different perspective. Here imagine standing on the ground at the same level as the samples and looking towards the clustering tree. Now the x-axis shows one of the reduced dimensions (PC1 or PC2) and the y-axis show the resolution dimension. Looking at a clustering tree in this way provides an alternative view where the focus is less on the structure of the tree itself and more on how it fits into a space that researchers are more familiar with.

5

Analysis of kidney organoid scRNA-seq data

“And the first lesson of all was the basic trust that he could learn. It’s shocking to find how many people do not believe they can learn, and how many more believe learning to be difficult.”

— Frank Herbert

Dune, 1965

5. ANALYSIS

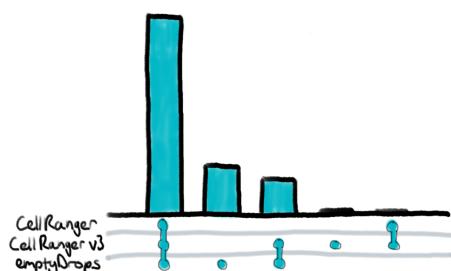
What cells are in kidney organoids?

stroma, podocytes, endothelium, epithelium
glial, neural progenitor, muscle progenitor

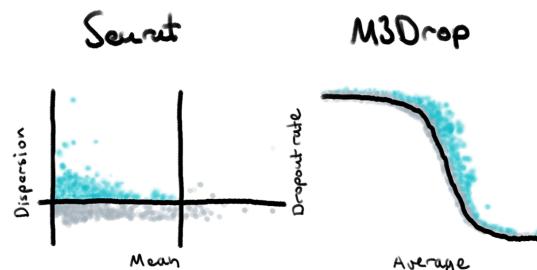
KIDNEY
OFF TARGET

What if we did things differently?

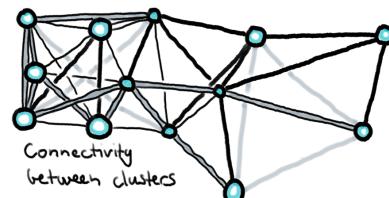
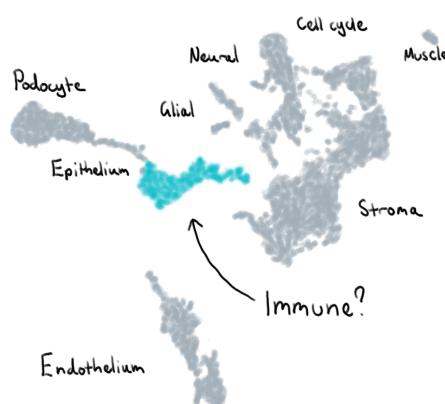
DROPLET SELECTION



GENE SELECTION



Partition-based graph abstraction



Cell velocity

Transcriptional direction of cells



5.1 Introduction

The kidney has a branching structure where incoming blood arteries are split into smaller and smaller vessels. At the end of the smallest capillaries are nephrons, the functional unit of the kidney. Blood is filtered in the glomerulus and the filtrate passes through the proximal tubule, loop of Henle, and distal tubule, then into the collecting duct. Each section of the nephron is made up of specialised cell types responsible for balancing concentrations of particular molecular species between the filtrate and blood. To do this they produce a range of specific transporter molecules that secrete or extract molecules including metal ions, small organic molecules and water. Disruptions to these important cell types can have a profound effect on the body as the delicate balance between concentrations of molecular species is lost.

Kidney disease can develop over a lifetime as the results of a range of factors. When the condition becomes chronic the only reliable treatments currently available are dialysis, which is time consuming and unpleasant, or a kidney transplant. Some kidney conditions are purely genetic and are caused by defects in kidney development or structure rather than damage from external factors. Understanding kidney development and structure is key to understanding these diseases and developing treatments for them.

One way to investigate kidney development is to grow kidney tissue from stem cells [182,188]. By providing the correct growth factors and conditions, stem cells which could become any cell type can be directed to differentiate into kidney tissue. Part of this process involves transferring the cells from growing in a flat, two-dimensional layer on a plate to a three-dimensional environment, where different types of cells can interact with each other and form structures that replicate a nephron. These tissues are known as kidney organoids and can be used to model various aspects of kidney development, but the extent to which organoids are useful depends on how well they replicate real kidney cell types.

Organoids have typically been characterised by immunofluorescence imaging using antibodies for known markers of nephron segments and cell types. This approach makes it easy to see what structures are present in an organoid and if their morphology resembles real tissues. However, using immunofluorescence relies on prior knowledge about which proteins are important, as well as antibodies for those proteins being available. Typically only four or five colour channels are available for immunofluorescence imaging, limiting the number of proteins that can be shown at once. Bulk RNA sequencing has also been used to characterise kidney organoids. While this approach is less biased and measures expression across the transcriptome, it only provides a global average of expression across an organoid and cannot show exactly which cell types are present or which genes are expressed in the same cells. Single-cell RNA-seq addresses some of these limitations by measuring expression in individual cells, allowing cells with similar expression profiles to be identified and investigated.

As in many other areas, scRNA-seq has been rapidly embraced as a useful tool in studying kidney development and there have already been several studies investigating the cell types present in adult kidney, developing kidney and kidney organoids. During my PhD I have contributed to this aspect of several studies [187,190,214,215]. The work I was most closely involved in profiled a set of kidney organoid scRNA-seq samples to identify the cell types that were present. The analysis we presented in this publication followed a standard scRNA-seq workflow including pre-processing, quality control, clustering and ordering of cells, and detection of marker genes. But there were still many decisions we had to make during the analysis. Which tools should be used for each step and which parameters should we use for these tools? Should we have used any alternative analysis approaches? In this chapter I revisit some of those decisions, demonstrate some alternative methods and discuss what difference they might have made to the results.

5.2 Summary of published work

The dataset I present here consists of scRNA-seq data from three kidney organoids. Cells from the CRL1502-C32 human iPSC cell line were cultured according to the Takasato kidney organoid protocol [186] to day 25. Three organoid samples were taken and dissociated before capturing cells in parallel on 10x Genomics Single Cell Chips and preparing sequencing libraries using the 10x Genomics Single Cell Library Kit V2. Sequencing was performed on an Illumina HiSeq using 100 base pair paired-end reads at the Australian Genome Research Facility. The organoid differentiation and cell isolation was performed by Pei Xuan Er and Alex Combes.

This dataset was first introduced in “Evaluation of variability in human kidney organoids” [187]. This study investigated the robustness and repeatability of the organoid differentiation protocol by comparing RNA-seq data from multiple batches, cell lines and time points. We found that organoids from the same batch had highly correlated expression profiles but that there could be significant differences between batches. The genes that were differentially expressed between batches were compared with a time course of organoids taken at different stages in the protocol. We saw that many of the differences between batches were in genes involved in organoid maturation, suggesting that some batches grow faster than others. This is an important consideration for experiments that want to make comparisons between organoids, for example between disease and gene-corrected cell lines. To confirm this result at the single-cell level we compared the organoid dataset described above with a second batch containing a single organoid. Integration and clustering with Seurat [119,129] produced 13 clusters that showed expression of markers for important kidney cell types as well as some off-target populations. These clusters contained cells from all four organoids but the proportions were different, particularly between the batches, with the fourth organoid being depleted for stromal populations and showing lower expression of mature podocyte genes. While this result reinforces the findings from the bulk RNA-seq data, it is difficult to say what might be the effect of different patterning, dissociation and cell capture rather than organoid maturation. The computational and statistical analysis for this study was performed by Belinda Phipson. I performed pre-processing of the single-cell data using Cell Ranger and some of the early analysis.

While this study made use of kidney organoid scRNA-seq data it didn’t fully profile the cell types present. To better understand what cell populations are present in these organoids and detectable using scRNA-seq, we performed a re-analysis of this data that was published as “Single-cell analysis reveals congruence between kidney organoids and human fetal kidney” [214]. This study was lead by Alex Combes who performed much of the lab work and interpretation of results while I performed all the computational analysis. Details of this analysis are presented online at <http://oshlacklab.com/combès-organoid-paper/> [216]. The full text of this publication is included in **Appendix E** but I will summarise the results here. Using a similar workflow based around the Seurat package we identified 13 clusters including clusters that were labelled as covering kidney stroma, podocytes, epithelium and endothelium, based on the expression of known kidney marker genes. Other clusters were associated with active phases of the cell cycle as well as off-target muscle progenitor, neural progenitor and glial cells types. Re-clustering of the podocyte and epithelial clusters revealed more detail of the nephron lineage with five clusters covering nephron progenitors, podocyte precursors, mature podocytes, proximal tubule precursors and distal tubule precursors. We used Monocle [77,138] to order the cells along a pseudotime trajectory which showed two branches, with nephron progenitors heading towards either a podocyte or tubular fate. Characterising the samples in this way allowed us to identify many of the cell types we would expect to be present in a developing human kidney, but this process cannot show exactly how similar they are to real tissue. For example we found podocytes in these organoids, but are organoid podocytes the same as normal kidney podocytes?

To answer this question we used the Seurat integration method based on canonical correlation

analysis [120,121] to combine our kidney organoid datasets with an scRNA-seq dataset from human foetal kidney previously published by Lindström et al. [195]. Clustering of the combined dataset produced 16 clusters, most of which contained cells from both datasets and covered similar cell types to what we saw in the organoid-only analysis. Some clusters were only present in one dataset, including the off-target muscle and neural cell types in the organoids and blood cells in the foetal kidney data. One of the new clusters represented a population of immune cells. Most of these came from the foetal kidney dataset but a few originated from the organoids. While there were insufficient cells to truly say that immune cells are present in these organoids it is possible they could arise somewhere during the differentiation process.

As with the organoid data we looked more closely at the nephron lineage in the combined dataset. We saw evidence of similar developmental trajectories with more mature podocytes in the organoid samples and fewer distal tubule cells. At this resolution we also found two clusters unique to the human foetal kidney data: a stromal population that may be more stem-like in nature and originally clustered with the nephron progenitors, and a separate podocyte cluster. Some of the differences between the datasets can be explained by the differences between the samples, for example the human foetal kidney dataset comes from a section of the outer cortex rather than a whole organ. For this reason it is important to look for differences between the datasets within cells that have been labelled as the same cell type. Doing this revealed an overall stress or heatshock signature in the foetal kidney data that is most likely the result of the tissue being exposed to suboptimal conditions and made it difficult to see smaller differences. However, we were able to identify some differences, including in signalling pathways, that may suggest further avenues for directing kidney organoid differentiation. Overall our analyses of this dataset support the presence of important kidney cell types in kidney organoids and builds confidence in their usefulness for disease modelling and drug screening studies.

5.3 Outline and motivation

While we followed a well-established workflow in our previous analysis, there were still many decisions we had to make about which analysis tools and parameters to use. As I discussed in **Chapter 2** methods for analysing scRNA-seq data have developed at a rapid rate and there are always new software packages to try. New methods should improve on what already exists or approach the data in a different. Even when a dataset is well understood applying new methods can reinforce what has already been seen or reveal interesting new information. For the remainder of this chapter I will explore some of the alternative analytical choices we could have made and approaches we could have used in our analysis of the kidney organoid dataset, and what difference they may have made to our interpretation of this data. I have selected these methods to look at because they either do something significantly different to what we used in our previous analysis or they have been shown to be effective in comparison studies. This chapter also provides an illustration of some of the thought processes involved in an analysis project. An outline of the analysis I have performed and where it will be discussed in this chapter is shown in Figure 5.1. I begin by looking at some of the pre-processing steps that are often overlooked, particularly methods for selecting cell-containing droplets. I also compare the alignment-based quantification produced by the standard pre-processing pipeline to a pseudoalignment quantification. The next section explores some of the choices that can be made when performing quality control to select high-quality cells. The main aim of this analysis is to identify the cell types present in kidney organoids which requires clustering together similar cells. Clustering methods often require a set of genes to be chosen and here I have trialled an alternative gene selection method and compared the resulting clusters to the previously published analysis. In the final section of this chapter I apply two newly developed analysis approaches to this dataset to see what additional information they can add. Details of the analysis including the code used and exploratory visualisations are displayed on a

website at <https://lazappi.github.io/phd-thesis-analysis/> [217] and the analysis methods used are provided in **Appendix F**.

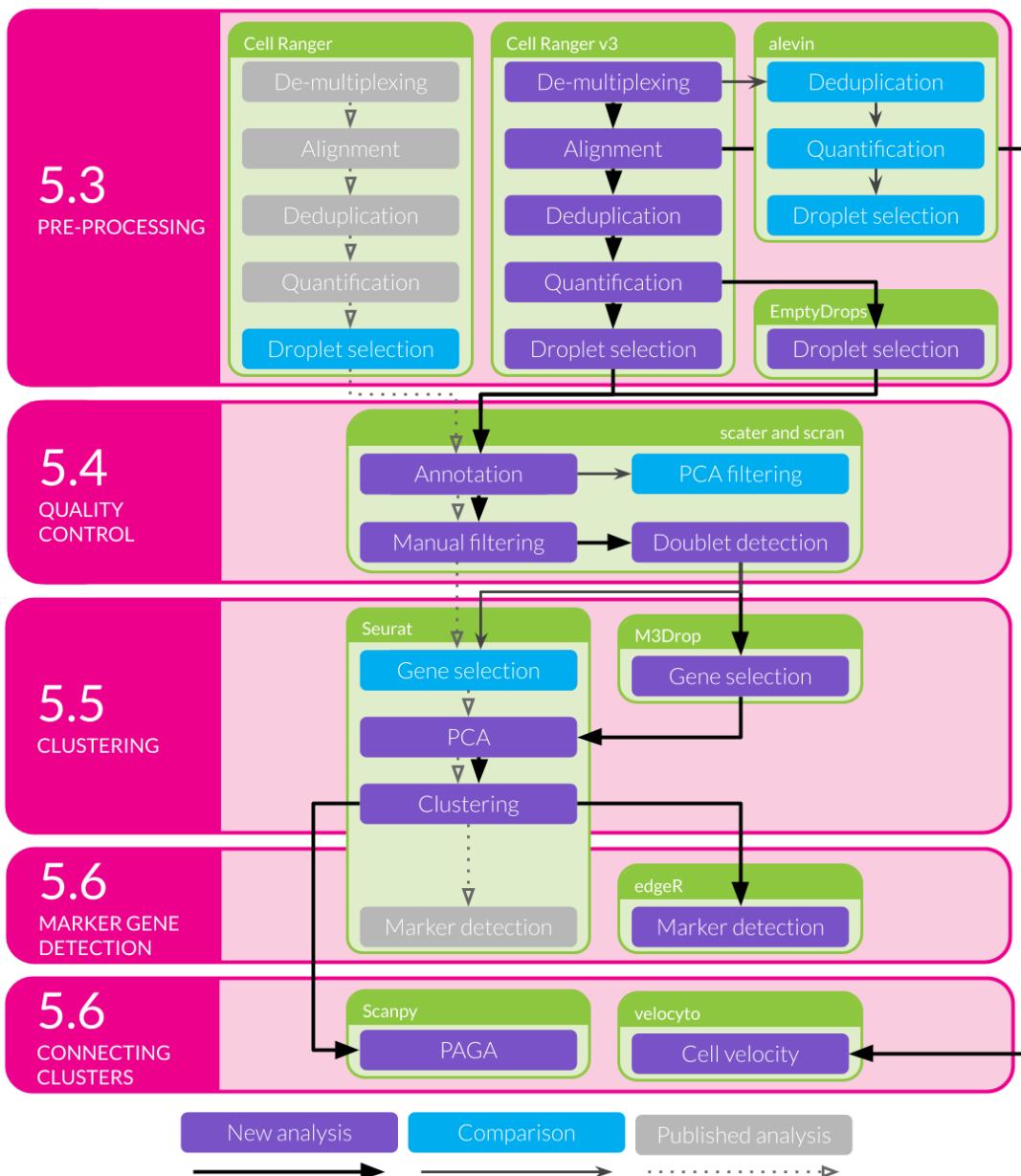


Figure 5.1: Outline of analysis of the kidney organoid dataset. Filled coloured boxes indicate stages of the analysis. Purple stages contribute to the final results shown here while blue stages were performed as a comparison. Grey stages were performed in the published analysis but are not shown here. The main analysis workflow is indicated with black arrows and comparison stages with pointed grey arrows. The published workflow is shown using smaller open grey arrows with dotted lines. Green shaded boxes show the software tools used and pink shaded boxes indicate the sections of this chapter that discuss the results of those stages.

5.4 Pre-processing

The first stage in any analysis of sequencing data is to convert the files returned by the sequencing machine to a format suitable for analysis. For scRNA-seq data produced using the 10x Genomics Chromium platform the standard tool for accomplishing this task is 10x's Cell Ranger software. This pipeline performs a number of tasks including demultiplexing the BCL base call files returned by Illumina sequencers and converting them to more common FASTQ sequence files, aligning reads to a reference genome, counting reads overlapping annotated genes, separating cells by barcode, and deduplicating UMIs. Some of these stages are performed by other tools such as Illumina's bcl2fastq software and the STAR splice-aware aligner [8] but with wrappers designed to handle specific features of 10x libraries. For the analysis presented here I have performed pre-processing of the first batch of three organoid samples using Cell Ranger (v3.0.1) with the GRCh38 reference genome provided by 10x Genomics (v3.0.0) based on the ENSEMBL gene annotation (v93). Each sample was processed separately to produce an expression matrix and then these were aggregated without any normalisation to form a single dataset.

5.4.1 Droplet selection

One of the final steps in pre-processing a dataset from a droplet-based cell capture method is to select droplets that contain cells. Many droplets are produced during the cell capture process but most of these will not contain cells and any reads associated with them will be the result of ambient RNA present in solution. The original Cell Ranger selection algorithm (used in our previous publications) required an expected number of cells for each sample (N). The 99th percentile of the total number of UMI counts in the top N cells was then taken as a robust estimate of the maximum number of counts in a cell and any droplet with at least 10 percent of this maximum was selected as a cell [34]. A similar approach is to select droplets that fall above the knee point (where curvature is minimised) of the cumulative fraction of counts with respect to increasing total count [31]. While these methods effectively select most real cells, they risk overlooking cells that have a lower RNA content than the majority of cells in the sample. The EmptyDrops method [99] in the DropletUtils package provides an alternative approach (Figure 5.2). A threshold on total counts is selected, below which droplets are assumed to be empty, and these empty droplets are used to produce a profile of the ambient RNA in the sample. A Monte Carlo sampling approach is then used to calculate a p-value indicating whether a particular droplet is likely to have come from the ambient distribution. A traditional threshold is also used to select cells with more total counts than the knee point when droplets are ordered by total counts. Cell Ranger v3 uses a modified version of the EmptyDrops procedure where the original selection method is substituted for the knee point threshold.

I compared the newer EmptyDrops and Cell Ranger v3 droplet selection methods with the older Cell Ranger thresholding approach. As we would expect, the original Cell Ranger algorithm is the most conservative, selecting only 7006 cells (Figure 5.3A). The EmptyDrops and Cell Ranger v3 methods, which use a testing procedure rather than a strict threshold, select many more cell-containing droplets. Many of these cells are identified by both methods but there are also a large number that are identified only by EmptyDrops. This is surprising as the two software packages provide different implementations of what is largely the same algorithm. Some of these differences may be due to the parameters I have chosen for EmptyDrops but it is difficult to say if these are different to those that are used by Cell Ranger. This highlights some of the trade-offs inherent in using a pre-designed pipeline for processing any dataset. While it is convenient to have a single tool that automates a series of tasks, this often comes at the cost of less control over each stage than would be possible if using a series of individual tools.

It is difficult to evaluate if the extra cells identified by EmptyDrops are real cells, or if the

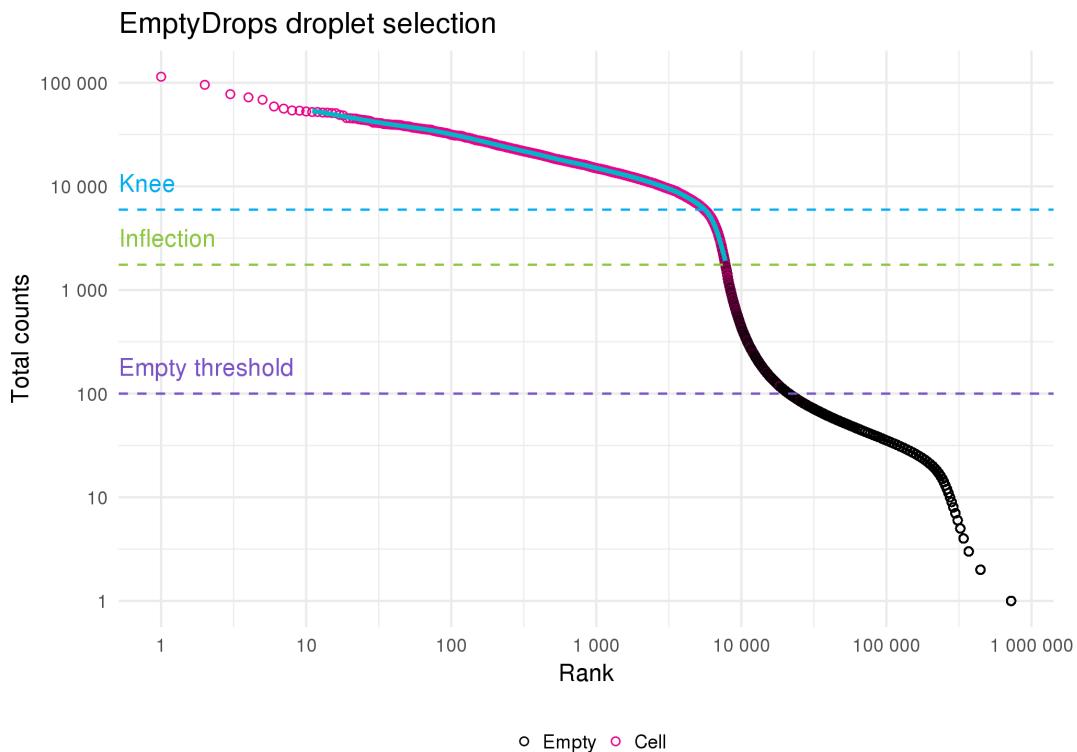


Figure 5.2: Droplet selection using EmptyDrops. Scatter plot of ordered total counts per droplet. Points are coloured by whether they were identified as cells (pink) or empty (black). Horizontal dotted lines show the knee point (blue), inflection point (green) and the threshold for droplets assumed to be empty (purple).

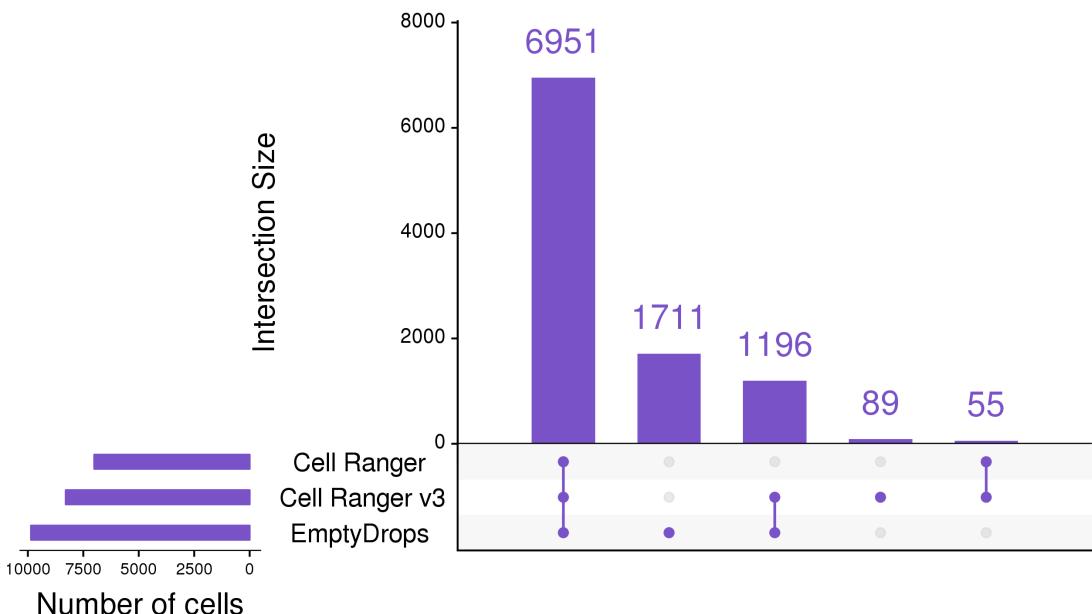


Figure 5.3: UpSet plot [218] comparing droplet selection methods. Cell Ranger v3 and EmptyDrops identify significantly more cells than the traditional Cell Ranger approach. My use of EmptyDrops also identifies a set of cells that are overlooked by the automated Cell Ranger v3 procedure.

algorithm has been too permissive and selected droplets that are actually empty. In the following section I will perform further quality control to remove low-quality cells so at this stage I chose to keep cells identified by either the EmptyDrops or Cell Ranger methods. Of the 945038 droplets with at least one count present in the raw dataset, 10002 were selected as cells by at least one of these methods.

5.4.2 Alevin comparison

Cell Ranger is based on the traditional workflow for processing RNA-seq reads, which is to align them to a reference genome then count how many reads overlap annotated genes. In recent years it has become common to use methods such as Salmon [12] and Kallisto [11] that directly estimate transcript expression without alignment. While these approaches do not produce an exact genomic location for each read they can be much faster, which is appealing for scRNA-seq data as there are many more cells to quantify. However, factors such as cell barcodes, UMIs and different model assumptions mean that these tools have had to be adapted for scRNA-seq data. Alevin [91] is the scRNA-seq mode available in Salmon.

Alevin has its own method for identifying which droplets contain cells. First, the knee point in the cumulative distribution of barcode frequencies is identified and any droplets above this are assumed to contain cells. The Levenshtein distance [219] between the barcodes of all droplets is calculated and if they are not sufficiently close to other barcodes they are considered to be noise and the associated reads discarded. Alevin performs a second round of cell whitelisting following quantification. Droplets are divided into three zones: high-quality (top half above the knee point), ambiguous (bottom half above the knee point) and low-quality (well below the knee point). A naïve Bayes classifier based on a set of features, including the fraction of reads mapped, the fraction of mitochondrial reads and the duplication rate, is then used to classify droplets as either high or low-quality. This is very different to the EmptyDrops approach, using a machine learning technique rather than a statistical test and technical features rather than the overall expression profile, and is more similar to approaches that have been suggested for quality control of cells [100].

Figure 5.4A shows a comparison of the alevin approach to the two methods I used to select cells in the previous section. Most of the cells are identified by all three methods but just like EmptyDrops, alevin finds a large number of cells that are overlooked by Cell Ranger v3. What is particularly striking is that there is very little overlap in the extra cells identified by alevin and EmptyDrops suggesting that the differences between the two approaches may be important. However, I should note that alevin uses its own quantification as input for classification while I provided the counts from Cell Ranger to EmptyDrops, so some of the differences may be due to the data they were given rather than the selection methods themselves. Alevin (and Cell Ranger v3) also processes the three samples separately while I performed EmptyDrops on the whole dataset.

Similar to Cell Ranger, the quantification stage of Alevin involves several steps including mapping reads and deduplicating UMIs. Alevin maps reads to the transcriptome and groups them by cell barcode and UMI. To deduplicate UMIs, alevin considers the mapping location and the UMI sequence (to account for sequencing errors). Because reads have been mapped to the transcriptome rather than the genome, there is an additional layer of information that can be useful for identifying if reads with the same UMI map to the same location. This should avoid over collapsing UMIs. Following these steps, the transcriptome level expression estimates are aggregated to the gene level for analysis. Figure 5.4B shows a comparison of the total counts per cell between alevin and the alignment-based approach using Cell Ranger. For cells that were identified by both pipelines (pink) the total counts are very similar. Cells identified just by the alignment-based approach (blue) tend to have fewer counts than those identified only by alevin (green). There are more differences if we look at the mean counts per gene (Figure

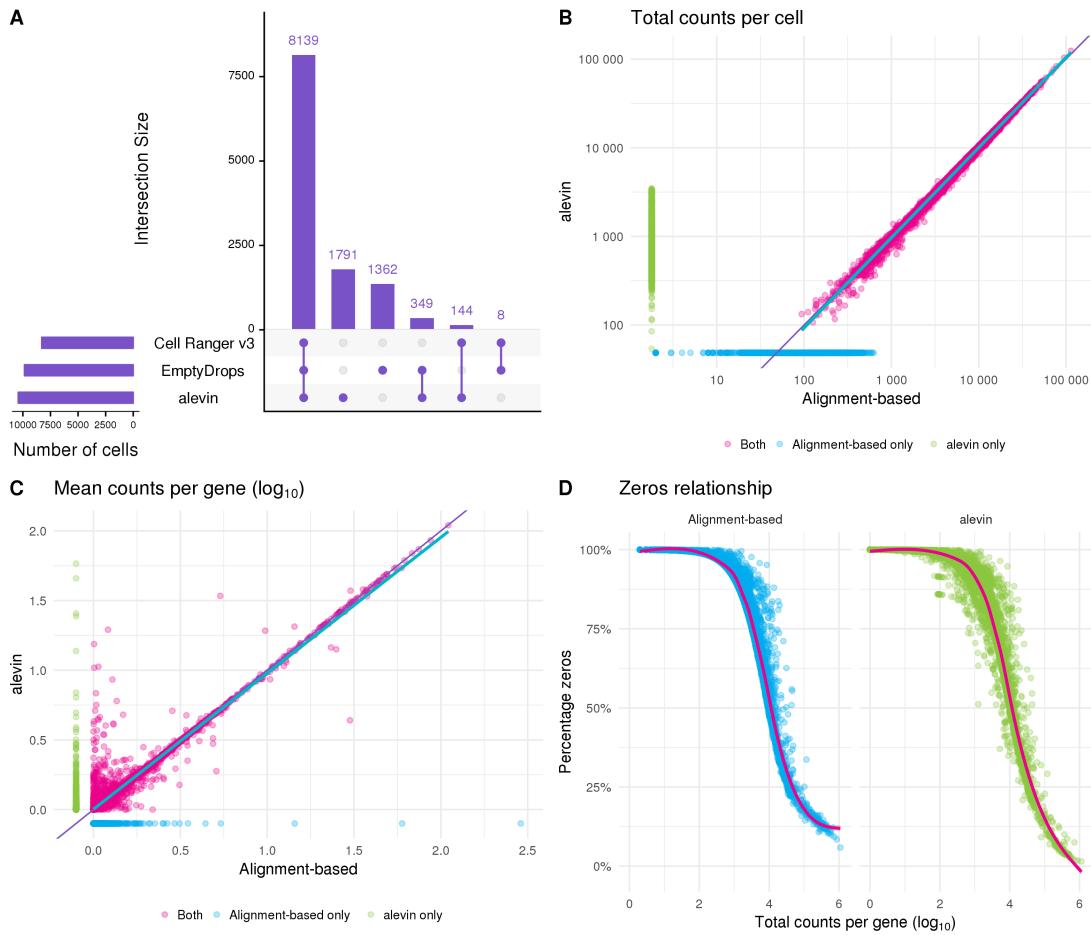


Figure 5.4: Comparison to quantification using alevin. (A) UpSet plot comparing the droplets identified as cells by the alevin, Cell Ranger v3 and EmptyDrops methods. (B) Scatter plot comparing total counts per cell as estimated by alevin and the alignment-based Cell Ranger pipeline. Points are coloured according to the method that identified them as cells (alevin only (green), alignment-based only (blue), both (pink)). Thin purple line shows $y = x$, thick blue line shows a linear fit for cells identified by both methods. (C) Scatter plot comparing mean counts per gene as estimated by the two pipelines, point colours and lines as in (B). (D) Relationship between the total counts per gene and the percentage of cells with a zero count for each pipeline.

5.4C). Most of the variability is at low expression levels, but there are some genes that are lowly expressed by one method but have relatively high expression in the other. One reason for this could be either method missing (or incorrectly assigning) counts altogether. Figure 5.4D shows the relationship between the total counts per gene and the percentage of cells with zero counts. This relationship is noisier for the alevin quantification, particularly below the trend line, and also shows more genes with a very low percentage of zeros.

While there are differences between the results from the two approaches there is nothing in this comparison to suggest that either is inaccurate. Genes that are found to be expressed by only one of the methods could possibly lead to differences in downstream interpretation but it is very difficult to say which is a better representation of true expression and it is unlikely that all genes providing similar evidence (for example markers of a cell type) would be affected in the same way. On this evidence alevin presents a viable alternative to alignment-based quantification for scRNA-seq data, particularly for large datasets where computational time is a concern. However, when alignment is required anyway, it is simpler and avoids repetition to use those alignments to produce an expression matrix for further analysis. For the remainder of the analysis I have used the alignment-based quantification of this dataset.

5.5 Quality control

In the previous section I discussed methods for selecting droplets that contain cells but there are further steps that are required in order to select high-quality cells. There are a variety of reasons that we may want to remove cells before further analysis. Cells can become damaged during processing, may not contain sufficient counts for analysis or may be the result of doublets where two cells are captured in the same droplet. Because these factors are not related just to the number of reads per cell they may not have been detected by the methods used previously and require different approaches.

Two tools that are useful for quality control of scRNA-seq data are the scater [101] and scran [109] packages. The scater package provides functions for calculating commonly used quality control metrics, adding annotations and visualisation, while scran can be used for various calculations such as assigning cell cycle phase and doublet scoring. After selecting cell-containing droplets I used these two packages to add extra information to the dataset. Additional feature annotation was retrieved from BioMart [220] using the biomaRt package [221] via functions in scater. Before calculating scater's quality control metrics I added mitochondrial genes as a feature control set, as the proportion of counts assigned to these genes can be an indicator of the health of a cell. I also used the cyclone function [222] in scran to score cell cycle activity and assign cell cycle phases to each cell. This method uses the relative expression of pairs of genes from previous studies of the cell cycle in order to score cell cycle activity. Thresholds can then be applied to these scores to assign a phase to each cell. Pre-computing a range of metrics is a valuable feature of scater as they are immediately available for exploration and filtering.

The simplest approach to quality control is to select thresholds for various metrics and to remove cells that fall outside these values. This is the approach I have taken here and in the previously published analysis. While previously these thresholds were selected by visually inspecting distributions, here I have used a more data-driven approach by selecting thresholds that are a specified number of median absolute deviations (MADs) from the median value. This is made easy by the `isOutlier` function in scater. The first filter I have applied is to the number of counts per cell, with thresholds on both the total number of counts per cell and the total number of features that are expressed (Figure 5.5A). These filters are important as cells that fall below these thresholds do not contain enough information to be useful for analysis. A similar threshold is applied to the percentage of counts assigned to mitochondrial genes (Figure 5.5B). A high proportion of mitochondrial expression is a sign of distress and can indicate that cells have become damaged during processing. Once most of the low-quality cells have been removed it becomes possible to check for doublets. Here I have used a simulation-based approach implemented in scran. This method is similar to the one suggested by DoubletFinder [98] and creates synthetic doublets by combining counts from randomly selected cells. The density of simulated doublets in the neighbourhood of each cell in PCA space is then used to calculate a doublet score to which a cutoff can be applied (Figure 5.5C). For this threshold I have chosen a value that gives a number of doublets roughly equal to the expected number of the cell capture technology (around one percent for the 10x Chromium).

An alternative approach is to rely purely on the quality control metrics themselves rather than manually selecting thresholds. PCA can be performed using the technical metrics such as total counts per cell rather than gene expression levels, and then outliers can be detected in PCA space (Figure 5.5D). This approach relies on the assumption that most cells will be high-quality and that the values that have been calculated will be sufficient to separate them from the low-quality cells. On this dataset the PCA-based approach identified a subset of the cells that were removed by my manual thresholds as low-quality. While I believe the idea of an automated quality control method is valuable, it will require more work to identify which features are most useful and it is likely that they will differ between technologies and tissue types. Additionally, any automated approach will always be more difficult to interpret than

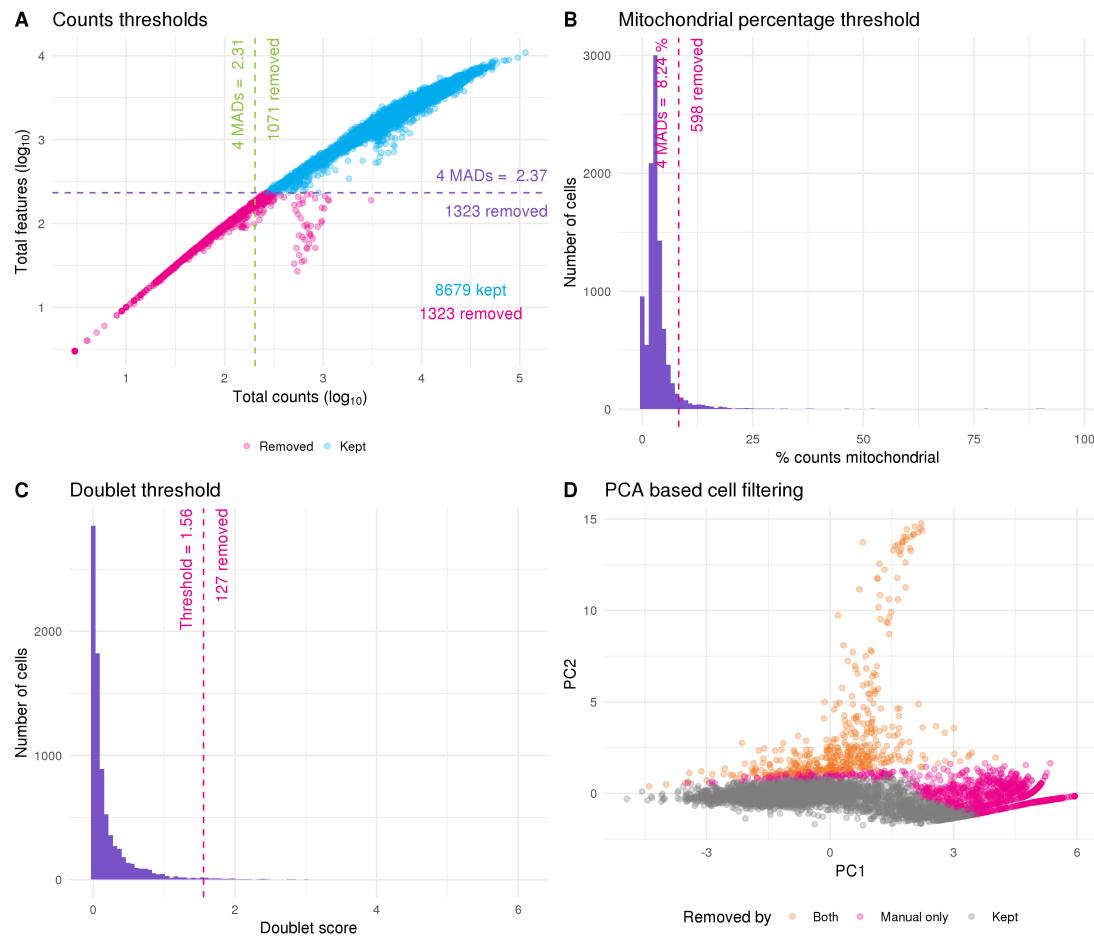


Figure 5.5: Thresholds used for quality control. (A) Scatter plot of total counts per cell against number of features expressed. Thresholds for selecting high-quality cells are shown by purple (features) and green (counts) dotted lines. Points are coloured according to whether cells were kept (blue) or removed (pink). (B) Histogram of percentage of counts assigned to mitochondrial genes with filtering thresholds shown in pink. (C) Histogram of doublet scores with threshold shown in pink. (D) PCA plot based on technical factors. Cells are coloured according to whether they were removed by both the PCA outlier method and manual filtering (orange) or just manual filtering (pink).

clear manual thresholds. For these reasons I have used the dataset produced by filtering cells using the manual thresholds for the rest of this analysis. I have also performed a minimal filtering of genes, selecting those that have at least one count in at least two high-quality cells. After quality control the dataset had 8059 cells and 22739 genes with a median of 7838 counts per cell and a median of 2506 genes expressed.

After removing low-quality cells it is important to inspect the data again to check that the quality control has been successful. If the selected thresholds are too severe it is possible to remove interesting populations of cells that differ from the majority of the dataset, such as a rare cell type that is smaller and therefore has less RNA. One way to do this is to look at differences in expression between the cells that have been removed and those that have been kept. Here I have calculated the predicted log fold change using the predFC function in the edgeR package and plotted it against the average expression for each gene (Figure 5.6A). Seeing a set of interesting genes (in this case known kidney marker genes) with positive fold changes in the removed cells would suggest that the selected thresholds are too stringent. That is not the case here as can be seen from the 15 genes with greatest fold change shown in Figure 5.6B.

On the other hand if the selected thresholds are not strict enough we will have retained a set of low-quality cells. One way to identify these is to look at the dataset in a reduced dimensional

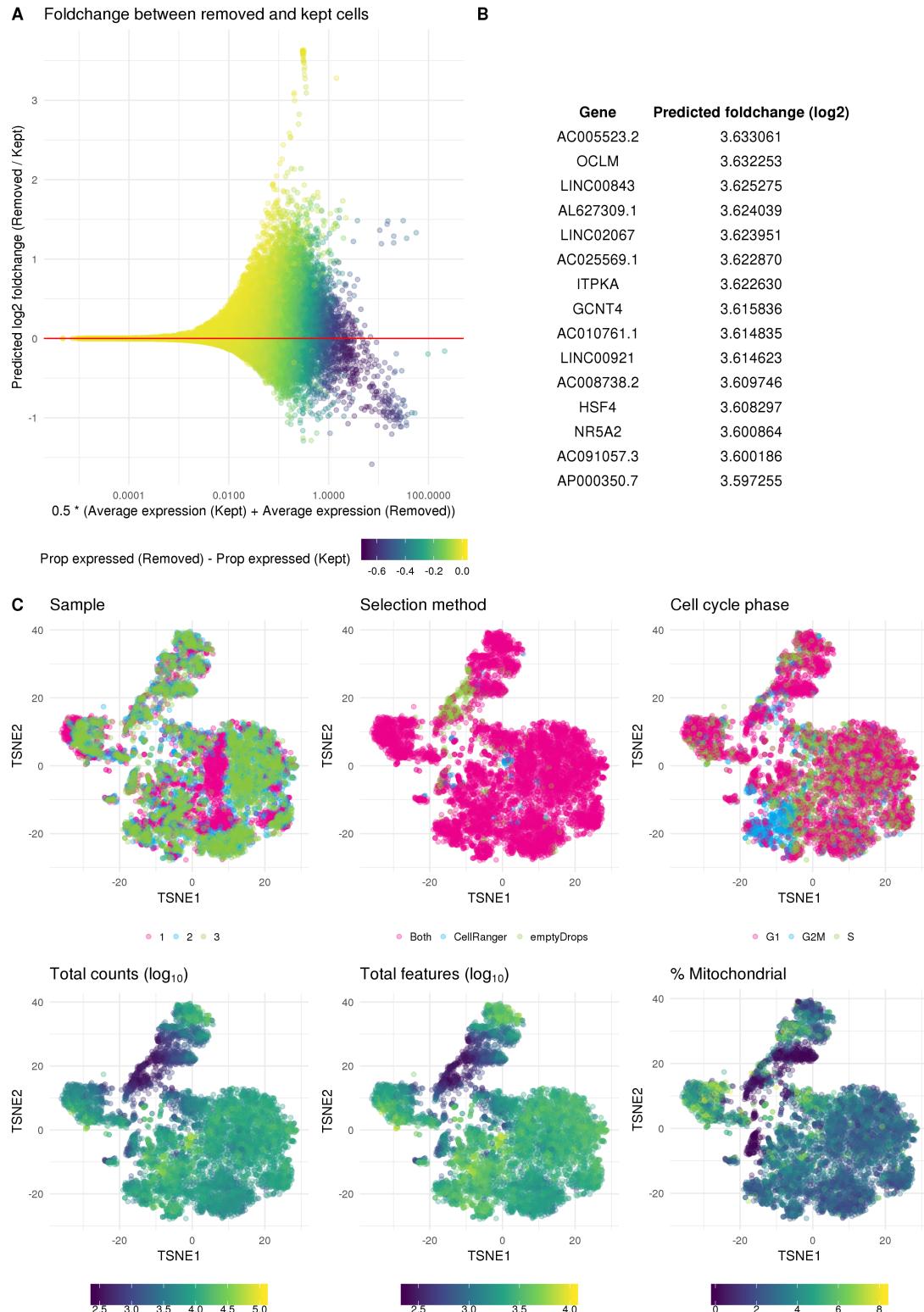


Figure 5.6: Validation of quality control. (A) Scatter plot of genes showing average expression against predicted fold change for removed cells compared to kept cells. Points are coloured according to the difference in proportion of cells that express each gene between removed and kept cells. (B) Top 15 genes when sorted by positive fold change in removed cells. (C) t-SNE plots of cells coloured by various technical factors including sample, droplet selection method, assigned cell cycle stage, total counts, total features and percentage mitochondrial counts.

space using techniques such as t-SNE or UMAP, highlighting technical factors that may be associated with low-quality cells. If a low-quality population is present we would expect to see it separate from the other cells. This approach is taken in Figure 5.6C where I have coloured a t-SNE plot by organoid sample, the droplet selection method, assigned cell cycle stage, total counts, total features and percentage of counts assigned to mitochondrial genes. These plots show a small population of cells that have fewer counts and have mostly been selected using the EmptyDrops method. However, this population is not clearly separated and may represent some intermediate cell type. At this stage I will retain those cells but following clustering it may be appropriate to revisit that decision if they do not appear to represent a logical biological population.

5.6 Clustering

Now that we have a dataset containing high-quality cells, the next step in my analysis is to perform clustering. It is necessary to use some technique to group and label cells. For this kind of dataset that contains an unknown set of cell types, an unsupervised approach is a good fit. Clustering allows us to group cells based purely on their expression profiles without making any assumptions about what those groups should be. Here I am going to use the graph-based clustering method available in the Seurat package. The first step in this approach is to select a subset of genes to use.

5.6.1 Gene selection

Most genes will either be expressed at a consistent level across all cells in a dataset or have too few counts to provide enough information about differences between cells. To reduce the noise in the dataset and the computational cost of analysis we want to select a set of genes that will be useful for separating cells. An intuitive approach is to filter based on the mean expression level to remove lowly expressed genes, and the gene variability to select those that explain differences between cells. This is the default approach used in Seurat (Figure 5.7A). The thresholds I have chosen here select 2457 highly variable genes.

The M3Drop package suggests an alternative approach that considers the proportion of zero counts per gene rather than the variability [104]. Here the idea is that if a gene is important for identifying a cell type then its expression will be concentrated in a subset of cells and there will be more zero counts than would be expected based on the overall mean expression. The original M3Drop method fits a Michaelis-Menten function to the relationships between mean expression and dropout rate (proportion of zeros) but for UMI datasets that have not been sequenced to saturation the authors suggest their NBDrop method instead, which fits a library size adjusted negative binomial model. A binomial test is then used to identify genes that have significantly more zeros than would be expected from the fitted model (Figure 5.7B). By using a statistical test this method removes the need to set multiple thresholds. Here M3Drop has selected 1952 genes with excess zeros.

If we compare the sets of genes selected by these two methods we see that a relatively small subset are identified by both of them (Figure 5.7D). These differences can be explained by the differences between variability and zero counts (Figure 5.7C). For example, it is possible for a gene to vary a lot between different cells but within an expression range that doesn't result in more zeros. Alternatively, a gene with relatively low dispersion may still have excess zeros if the underlying expression is low enough. For the previously published analysis we used the Seurat selection method, so here I am going to use the genes identified by M3Drop and see what effect that has on the clustering results.

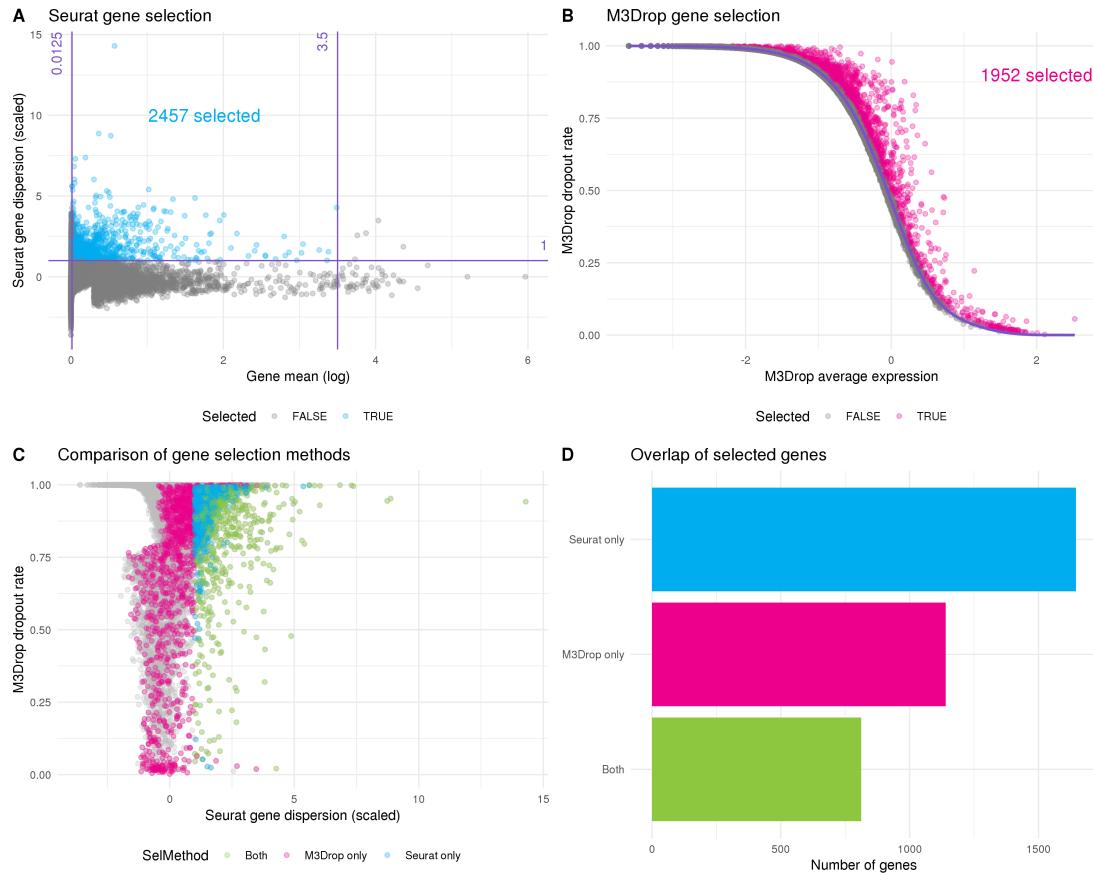


Figure 5.7: Selecting genes for clustering. (A) Scatter plot of mean gene expression level against dispersion. The default Seurat selection method sets thresholds on these axes. Purple lines indicate thresholds and selected genes are coloured blue. (B) The M3Drop method fits the relationship (purple line) between average expression (x-axis) and dropout rate (y-axis). Genes that have significantly more zeros than expected are selected (pink). (C) Comparison of gene selection methods. The dropout rate calculated by M3Drop is plotted against the dispersion calculated by Seurat. Points are coloured according to whether they were selected by Seurat (blue), M3Drop (pink) or both (green). (D) Bar plot of the number of genes selected by each method and the intersection. Most genes are identified by only one approach.

5.6.2 Resolution selection

There are several more steps in the Seurat clustering procedure. Once a set of genes is selected they are used to perform a PCA and a set of principal components is selected. Distances are then calculated between cells in this PCA space and the nearest neighbours for each cell are identified. By comparing the sets of neighbours for two cells and computing the Jaccard index, a shared nearest neighbour graph can be constructed where each node is a cell and the edges are the overlap in their neighbourhoods. A graph modularity optimisation algorithm such as Louvain modularity optimisation [133] can then be used to partition the graph and form clusters of cells. This graph-based clustering approach has been shown to be effective in comparisons of scRNA-seq clustering methods [130,131]. One of the decisions that needs to be made during this procedure is the value of the resolution parameter to use, which affects the number of clusters that are produced. As I have discussed in **Chapter 3**, clustering trees are one tool that can be helpful in making this decision. Figure 5.8A shows a clustering tree for this dataset with resolutions between zero and one.

The structure of this tree can tell us about how clusters are forming in this dataset. My general approach is to select a resolution where most of the branches have formed but before there are many low in-proportion edges crossing over between branches that can indicate that clusters are unstable. Displaying expression levels of genes that are expected to be important in the

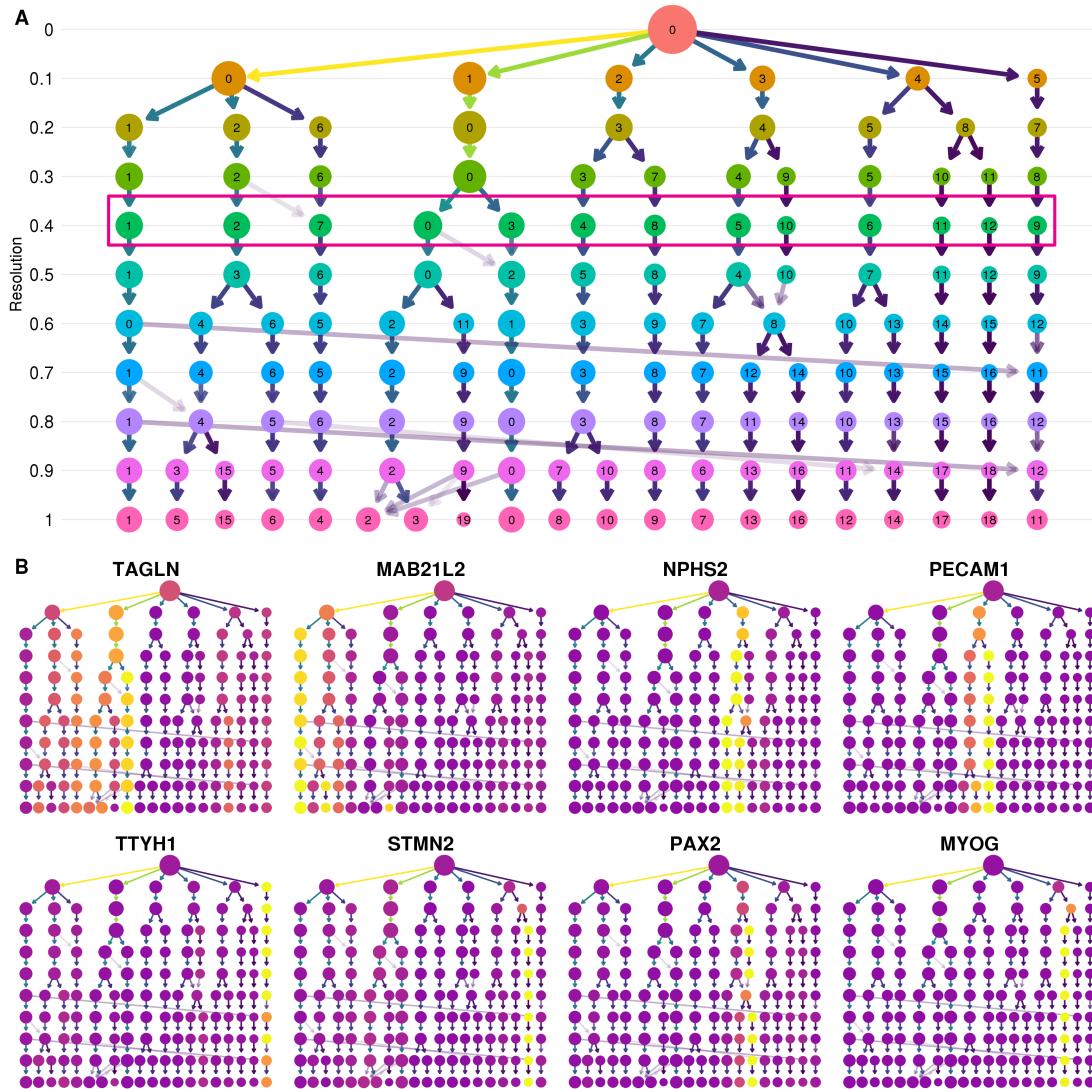


Figure 5.8: Selecting a clustering resolution. (A) Clustering tree showing resolutions between zero and one. Pink box shows the selected resolution. (B) Clustering trees with nodes coloured by a selection of published marker genes. Expression signals can help to select a clustering resolution.

dataset adds extra information to the structure of the tree (Figure 5.8B). Here I look to identify at which resolution we see clear, high-intensity signals for these genes. That is, a single or few clusters where they are highly expressed. Based on these trees I have selected a resolution of 0.4 which produces 13 clusters.

5.6.3 Cluster validation

The cluster assignments of the individual cells can be displayed in a reduced dimensional space such as that produced by UMAP [148,223] (Figure 5.9A). The dimensions shown in this visualisation and the cluster labels that are used to colour it are produced using different algorithms. By combining them we can confirm that the two approaches produce similar results. Clusters can have different densities in these plots so it is also useful to know how many cells they contain (Figure 5.9B). Now that we have a set of clusters we can check to see if they are associated with technical artifacts rather than biological factors.

Figure 5.9C shows the proportion of cells in each cluster that were identified by the two droplet selection methods I used, Cell Ranger v3 and EmptyDrops. Most of the additional cells identified by EmptyDrops have been placed in two of the clusters. The cells in these clusters

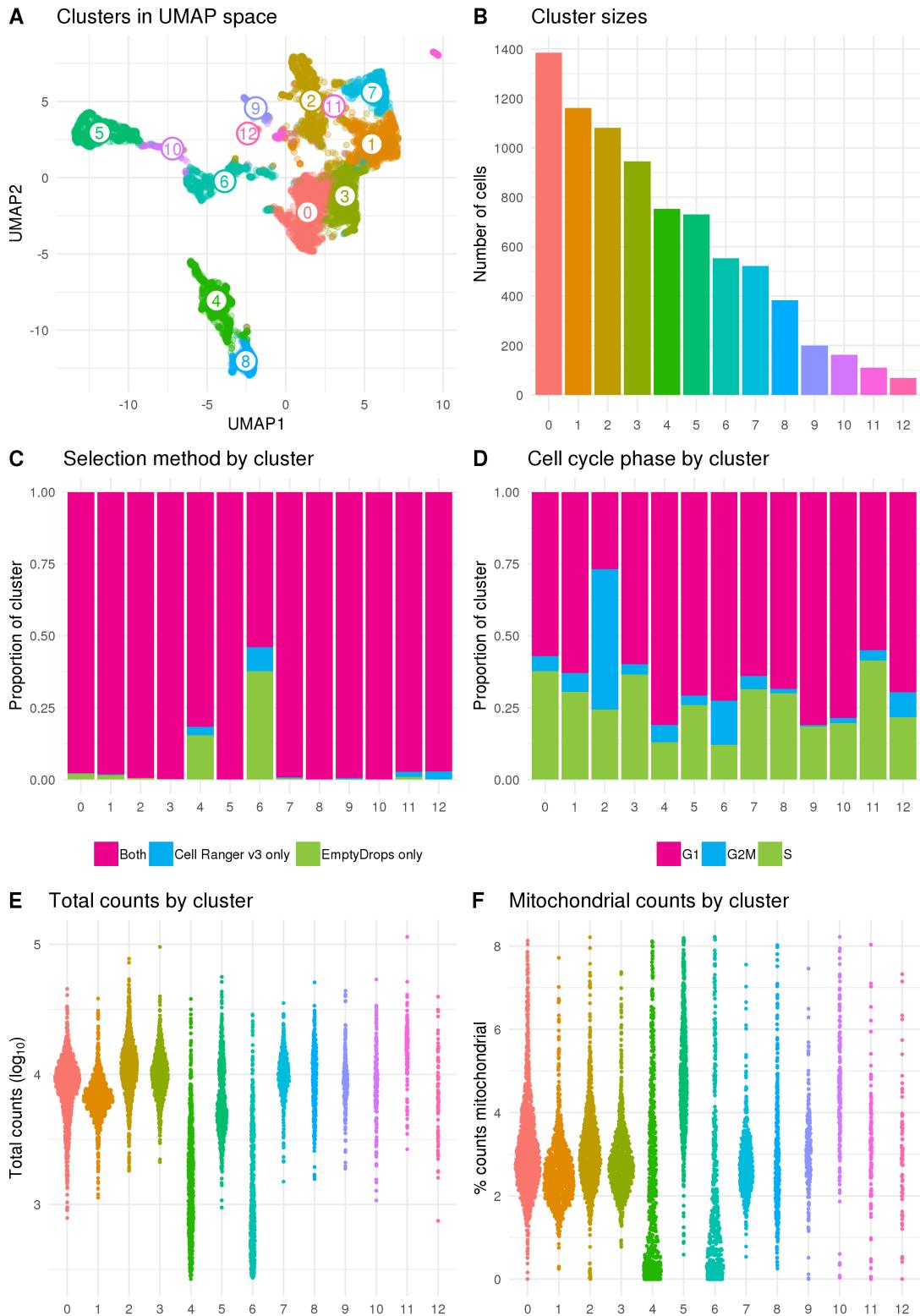


Figure 5.9: Validation of identified clusters. (A) Scatter plot of cells in UMAP space coloured by assigned cluster. (B) Bar chart of the number of cells in each cluster. (C) Stacked bar chart showing the proportion of cells in each cluster identified by each droplet selection method, Cell Ranger v3 only (blue), EmptyDrops only (green) or both (pink). (D) Stacked bar chart showing the proportion of cells in each cluster assigned to the G1 (pink), G2M (blue) or S (green) phases of the cell cycle. (E) SinaPlot [224] showing the total number of counts per cell in each cluster. (F) SinaPlot showing the percentage of counts assigned to mitochondrial genes per cell in each cluster.

also have fewer total counts (Figure 5.9E) and a lower percentage of mitochondrial counts (Figure 5.9F). This suggests that those clusters are at least in part a result of the lower count cells identified by EmptyDrops and it may be appropriate to remove them from the dataset. However, a large proportion of the cells in those clusters were also identified by Cell Ranger v3 and an alternative hypothesis is that these additional cells belong to a cell type that is smaller or contains less RNA for some other reason. Looking at the marker genes associated with those clusters can help to answer that question (**Section 5.7**).

5.6.4 Comparison to published clusters

Because we have previously analysed and interpreted this dataset I can compare these clusters to the previously published clusters. It is important to remember the differences between the two analyses. The published analysis included a fourth organoid from a separate batch that I have omitted here and was analysed using the standard Cell Ranger (v1.3.1) and Seurat workflows without EmptyDrops droplet selection or M3Drop gene selection. I also used different quality control thresholds and parameters at some stages. Figure 5.10A shows the proportion of cells in each cluster that were present in the published analysis. All of the clusters have some additional cells which is to be expected given that the use of the newer droplet selection methods produced a dataset with around 1000 more cells. The two clusters that contain the cells only identified by EmptyDrops (Cluster 4 and Cluster 6) stand out as having a very low proportion of cells from the previous analysis.

To compare the two sets of clusters I have calculated the Jaccard index between each cluster from the previously published analysis and each cluster from the analysis presented here (Figure 5.10B). Some of the clusters (such as Cluster 5) very clearly match up with a published cluster while others have overlaps with multiple similar clusters. For example, Cluster 2 which consists of cells from the two cell cycle clusters in the published analysis. There are four stromal clusters in both analyses but for two of them (Cluster 0 and Cluster 3) the cells are divided differently. The differences in clustering are the result of all the decisions made to this stage of the analysis including which cells and genes have been selected. Without a known truth it is difficult to determine which is better or more correct or if they simply provide different views of the same data with emphasis on different features and it is possible that both sets of clusters are biologically meaningful. The two EmptyDrops clusters do not show obvious signals but they do have some overlap with the published endothelial clusters which suggests that they may be a previously overlooked endothelial population. To confirm if this is the case we need to check whether they express known endothelial markers. Identifying marker genes for each cluster through differential expression testing can show us which genes are unique to each cluster.

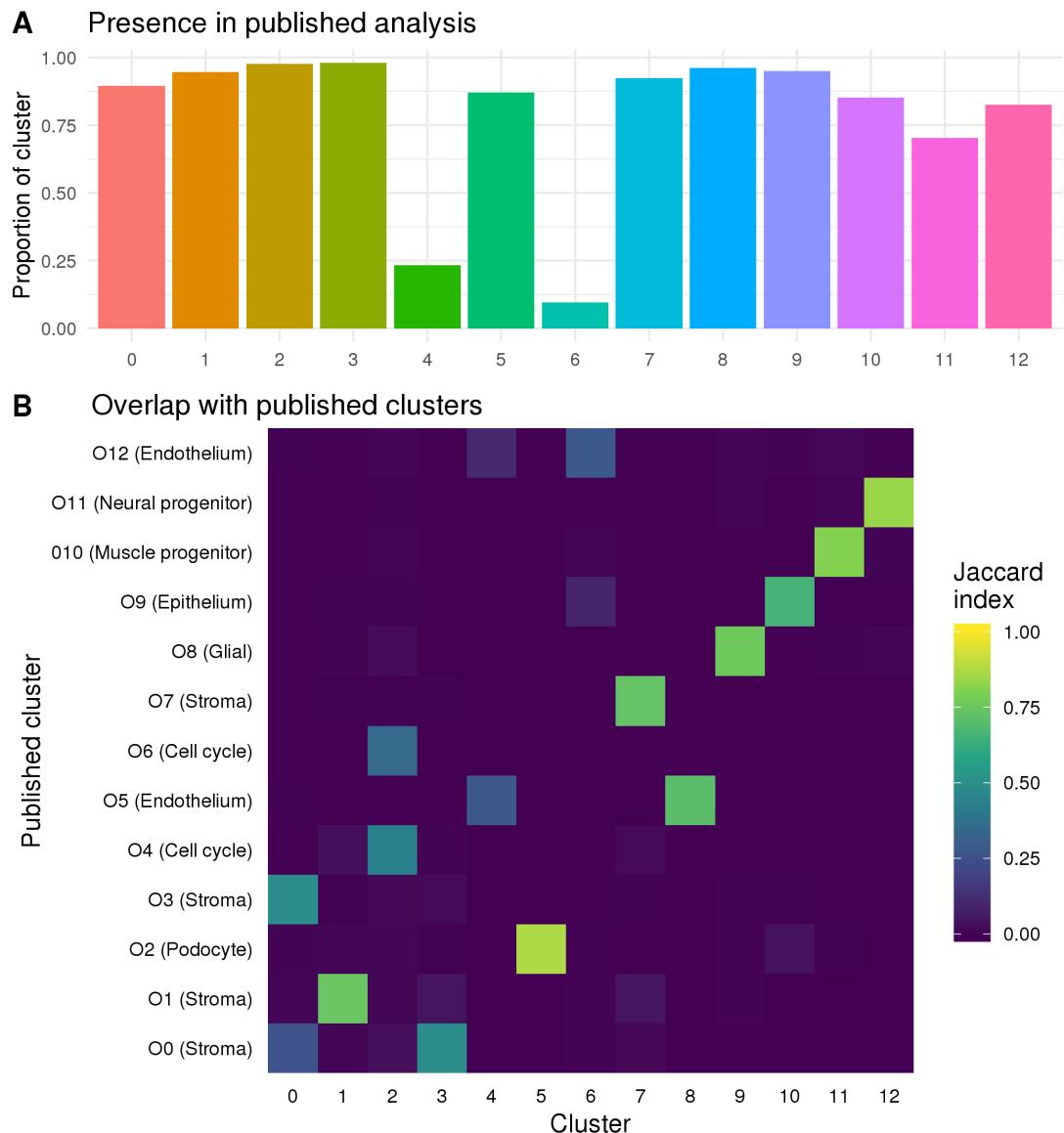


Figure 5.10: Comparison to previously published clusters. (A) Bar chart showing the proportion of cells in each cluster that were present in the previously published analysis. (B) Heatmap showing the Jaccard index between previously published clusters and the clusters from the current analysis. High values (yellow) indicate that many cells are shared between two clusters while low values (blue) indicate there is little overlap.

5.7 Marker gene detection

To try and identify which cell types a cluster represents we need to look more closely at what genes are expressed. Ideally we want to identify genes that are expressed in a single cluster at a relatively high level and whose function is well-known enough to interpret. These are often the same genes that are useful for techniques like immunofluorescence imaging. The most common way of identifying marker genes is to test for differential expression between the cells in a cluster and all other cells in the dataset, then inspect the significantly highly up-regulated genes. Many approaches have been suggested for this task, including a range of methods designed specifically for scRNA-seq data. Comparisons of these methods have found that the scRNA-seq specific methods do not show significantly better results than those designed for bulk RNA-seq or standard statistical tests [156,225]. For this analysis I have chosen to use the edgeR package as it performs well in these comparisons and allows for more complex designs such as including the proportion of genes that are expressed per cell (detection rate).

The edgeR method is based on the negative binomial distribution. After calculating normalisation factors to capture differences between cells and estimating common and gene-wise dispersions, a negative binomial is fitted to the dataset. We can evaluate this fit using some of the information calculated by the edgeR model such as the gene means (Figure 5.11A) and proportion of zeros per gene (Figure 5.11B) and comparing it to what we observe in the dataset. When looking at the comparison of means we see that at high expression levels the edgeR estimates become larger than what is observed. We also observed more zeros than expected when the proportion of zeros is low. Checking these fits can help us to decide if the dataset fits an ordinary negative binomial model or whether a zero-inflated negative model would be more appropriate, as has often been suggested for scRNA-seq data. For example, a method such as ZINB-WaVE can be used to calculate weights that can be added to the edgeR model to correct for zero inflation [64]. In this case I decided the edgeR fit was sufficiently good and these kinds of corrections were unnecessary.

Another important quantity that is calculated by the edgeR model is the biological coefficient of variation (BCV). BCV is the component of variability that captures the true underlying biological differences between replicates of the same RNA sample. Estimating BCV correctly can be particularly important for highly expressed genes. The BCV plot (Figure 5.11C) shows the common dispersion across all genes (pink line), the dispersion for each individual gene (points) and a trend across the dataset (blue line). The empirical Bayes approach of edgeR borrows information between genes in order to shrink the dispersion towards the trended value to achieve more robust estimates. The shape of the trend line can be used to check how well the BCV has been fit and we expect lowly expressed genes to be more variable than those that are highly expressed. Once we have a fitted negative binomial model we can test for differences between the groups we are interested in.

There are multiple testing procedures available in edgeR but the currently recommended approach is to fit a generalised linear model and test genes using a quasi-likelihood F-test. Figure 5.11D shows the number of significantly up-regulated genes for each cluster identified using this approach and the log fold change for individual genes. For most clusters we see a large number of positive marker genes but for the two clusters that contain most of the EmptyDrops cells many fewer markers have been identified. This could be a result of the lower counts in these clusters or a clue that these are not pure populations with clear markers.

It is important to look at the gene lists for each cluster in detail in order to interpret and assign cell types, but as we have previously analysed this dataset, and can be relatively confident of what it contains, I have plotted the edgeR results for a few of the genes that were previously identified to be important markers (Figure 5.11E). For most of the clusters we see that only markers for one particular cell type are significant with large fold changes and these are consistent with the overlaps between the published clusters presented in the previous section. For the clusters that had little overlap with the published analysis these results help to assign them. Cluster 4 clearly shows up-regulation of endothelial markers confirming this lineage as a possible identity for these cells. The other new cluster (Cluster 6) is more ambiguous with significant results for markers of several cell types but does show up-regulation of immune-related genes. Immune cells are not part of the lineage that these kidney organoids are differentiated towards, but we did see some organoid cells that clustered with immune cells from human foetal kidney in our previous analysis and there is a plausible pathway by which they could develop. Further lab-based experiments would be required to confirm whether these cells exist in kidney organoids. Based on the detected markers I have assigned cell types to clusters as shown in Table 5.1.

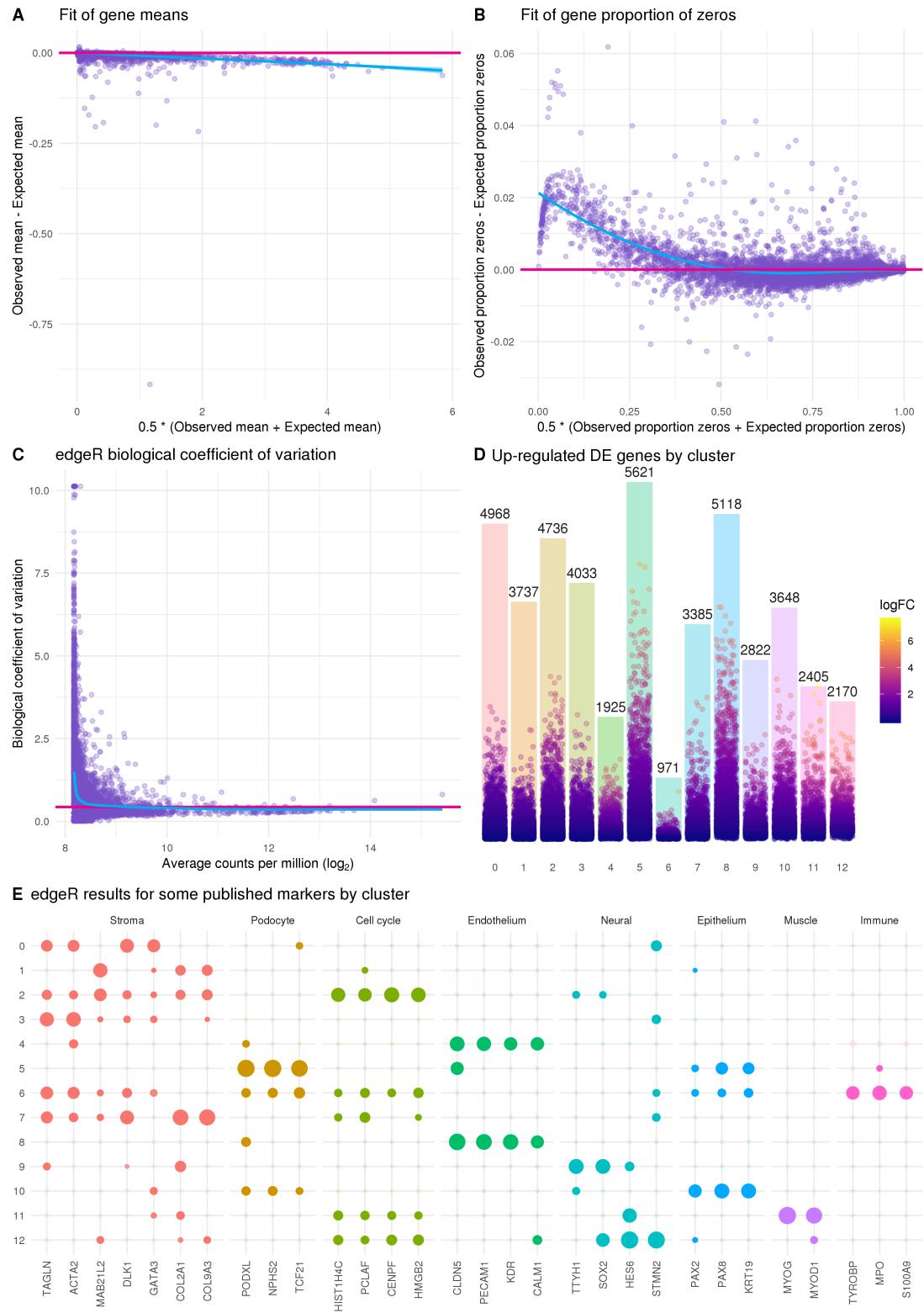


Figure 5.11: Results of edgeR differential expression analysis. (A) Scatter plot showing the edgeR fit of gene means. The x-axis shows the average of the observed gene mean and the expected gene mean from the edgeR fit and the y-axis shows the difference between the observed and expected means. The pink line lies at zero and the blue line is a LOESS fit. (B) Scatter plot showing the edgeR fit of gene proportion of zeros. Similar to (A) but showing the observed and expected proportion of zeros. (C) Plot of the biological coefficient as calculated by edgeR against the average log counts per million for each gene. (D) Bar chart of the number of significantly differentially expressed genes for each cluster with positive fold changes. Points show individual genes coloured by their log fold change. Position of each point on the y-axis also shows the log fold change but scaled to the height of each bar. (E) Differential expression results for some genes identified as important markers in the previously published analysis. Size of points indicates the positive log fold change and transparency indicates the negative false discovery rate.

Table 5.1: Cluster assignments based on detected marker genes

Cluster	Assignment
0	Stroma
1	Stroma
2	Cell cycle
3	Stroma
4	Endothelium
5	Podocyte
6	Possible immune
7	Stroma
8	Endothelium
9	Glial
10	Epithelium
11	Muscle progenitor
12	Neural progenitor

5.8 Connecting clusters

Clustering analysis is used to group cells and identify the cell types that are present in a dataset, but by itself clustering doesn't tell us anything about how those groups relate to each other. In the developmental context where there are multiple differentiating cell types these relationships are of interest. Some aspects of these relationships can be inferred from the proximity of clusters in reduced dimensional space or from the known biology of the assigned cell types, but this is largely based on intuition rather than information in the data. Computational methods to calculate developmental trajectories in scRNA-seq data have been a big focus of early scRNA-seq bioinformatics research. In our published analysis we used the Monocle package to create a pseudotime ordering for cells in the nephron lineage. However, methods like this do not perform as well on larger datasets with many cell types from possibly unconnected lineages, such as we have in the full organoid dataset. Here I will demonstrate two recently developed alternative approaches for connecting clusters and adding context to the current analysis.

5.8.1 Partition-based graph abstraction

Partition-based graph abstraction (PAGA) is a method for reducing the complexity of a dataset by estimating the connectivity between clusters [226]. The result is a reduced graph representation of the dataset where each node is a cluster and the edges show how connected they are (Figure 5.12A). PAGA is available as part of the scanpy package [227] and starts with the same kind of shared nearest neighbour cell graph that is used by Seurat's graph-based clustering. Connectivity between clusters is calculated by comparing the number of edges between cells in different clusters to the number that would be expected if edges were assigned randomly. This connectivity can be interpreted as the confidence in the presence of an actual connection between clusters.

The results for the organoid dataset show high connectivity between the various stromal clusters that make up the majority of cells. Connectivity between some of the other clusters also confirms what we would expect. For example the podocyte cluster is strongly connected to the epithelial cluster and the two off-target neural clusters (glial and neural progenitor) are strongly connected to each other. The position of the two clusters containing the additional EmptyDrops cells is less clear with connections seen between them, but not as strongly with the other clusters they would be expected to be related to, particularly between the two

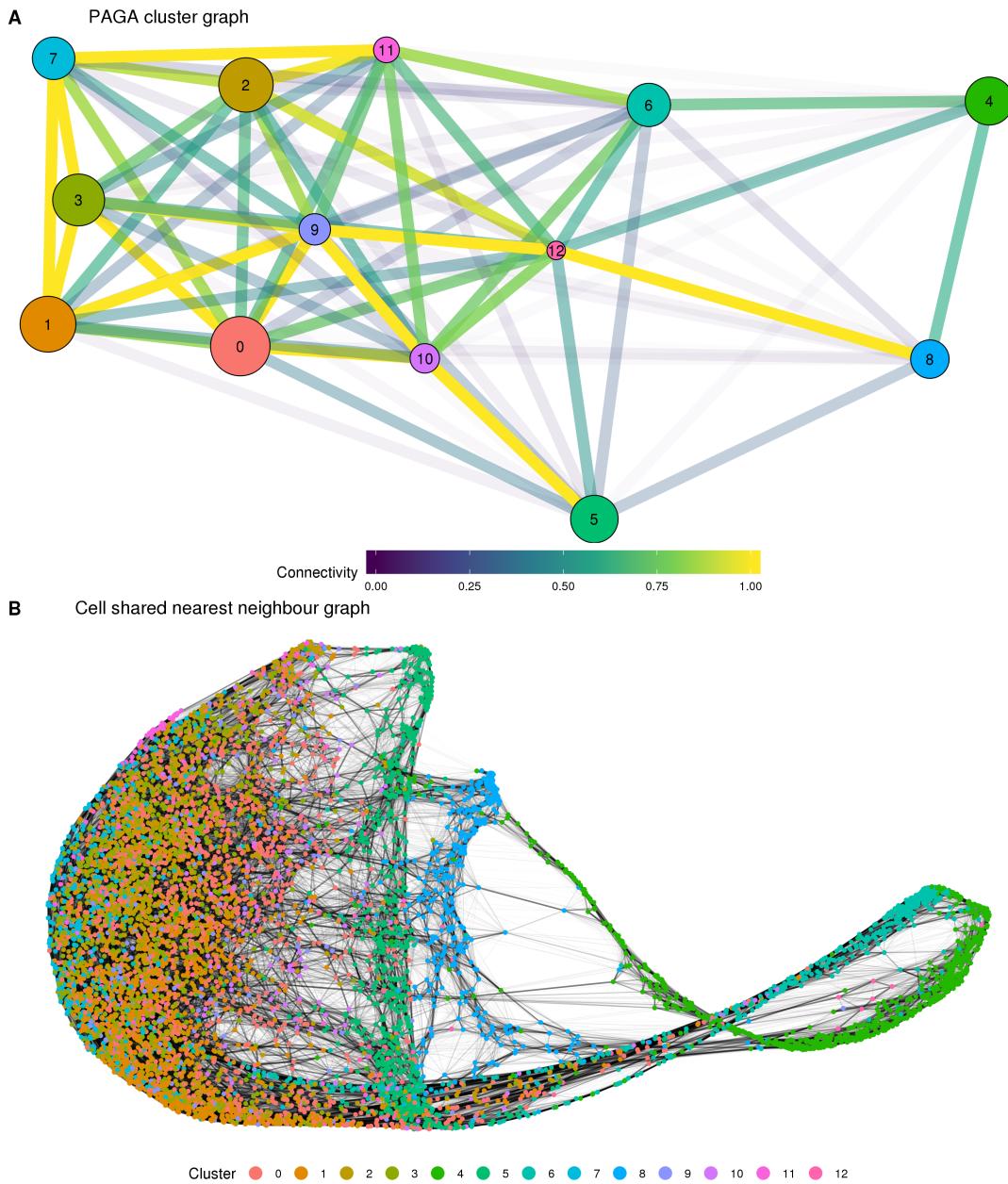


Figure 5.12: Results of partition-based graph abstraction (PAGA). (A) The cluster connectivity graph produced by PAGA. Each node represents a cluster and edges show the connectivity between clusters. The size of nodes indicates the number of cells in each cluster and the edge colour and transparency show the connectivity from low (blue) to high (yellow). (B) The underlying cell shared nearest neighbour graph. Nodes are cells coloured by cluster and edge weight indicates connectivity.

endothelial clusters. A possible explanation for this are technical features shared by these two clusters but not the clusters they are biologically similar to. The PAGA method does not give any estimate of the stability or variability of the connectivity between clusters but this would be a meaningful area for future research.

In some ways the graph produced by PAGA is similar but complementary to a clustering tree. The PAGA graph considers connectivity between clusters at the same resolution while clustering trees use the overlap between clusters from neighbouring resolutions to build a graph. Interestingly, the authors of PAGA emphasise the ability to explore a dataset at different resolutions by creating PAGA graphs from different sets of clusters produced using different clustering parameters. The development of two independent methods based on building graphs of clusters suggests that this is an approach that could have more general applications. The edges produced by PAGA can show which clusters are related but they don't provide any directionality to that relationship. One way to add some directionality to the cluster graph suggested by the PAGA authors is to perform a random walk on the cell graph to calculate a diffusion pseudotime across the dataset. However, this approach requires manually defining a root cell to use. An alternative approach for assigning directionality to cells involves looking at the splicing information available in the reads themselves.

5.8.2 Cell velocity

The majority of the reads in an scRNA-seq experiment come from mature mRNA where the introns have been removed through splicing, however some immature mRNA molecules that still retain their intronic sequences are also sequenced. The concept of cell velocity looks at these unspliced reads to determine which genes are being actively transcribed. By looking at which cellular processes are being switched on or off it is possible to infer an overall transcriptional direction for each cell. This approach was popularised for scRNA-seq data by the *velocyto* package [151]. Two steps are required for cell velocity analysis. First, *velocyto* inspects the aligned reads for each cell, but instead of only counting those that are inside an annotated exon it also generates a counts matrix of unspliced reads for each gene. These two matrices are then combined to estimate a velocity for each cell by solving a set of differential equations. The velocities can be projected onto a reduced dimensional space by looking at a correlation-based transition matrix on a nearest neighbour graph in that space. When there are more than a few hundred cells it is difficult to look at the velocity for each individual cell, so it can be useful to summarise them in some way. By forming a regular grid of points and using Gaussian smoothing to calculate an average of the nearby cells *velocyto* calculates a velocity field across the whole dataset, indicating the overall velocity of an area (Figure 5.13).

Looking at the velocity field for this dataset we see some of the expected developmental trajectories. For example, cells in the epithelial cluster are headed towards the more mature podocyte population. Despite having a limited connectivity in the PAGA results we see the endothelial-like cluster containing EmptyDrops cells (Cluster 4) trending towards the other endothelial cluster. Cluster 6 is again more ambiguous, showing velocity towards both endothelial and stromal clusters. This suggests that if there are immune cells present they may only be a subset of this population. We see little evidence of any clear directionality in the stromal clusters, likely because these cell types are more mature and are not actively changing their transcription profiles. There is an important caveat when looking at these *velocyto* results, which is that the velocity arrows depend on the space they are projected into and viewing them in different embeddings may give different views. Finding a more robust way to visualise cell velocity would be a useful area of further research that would improve the ability to interpret these results.

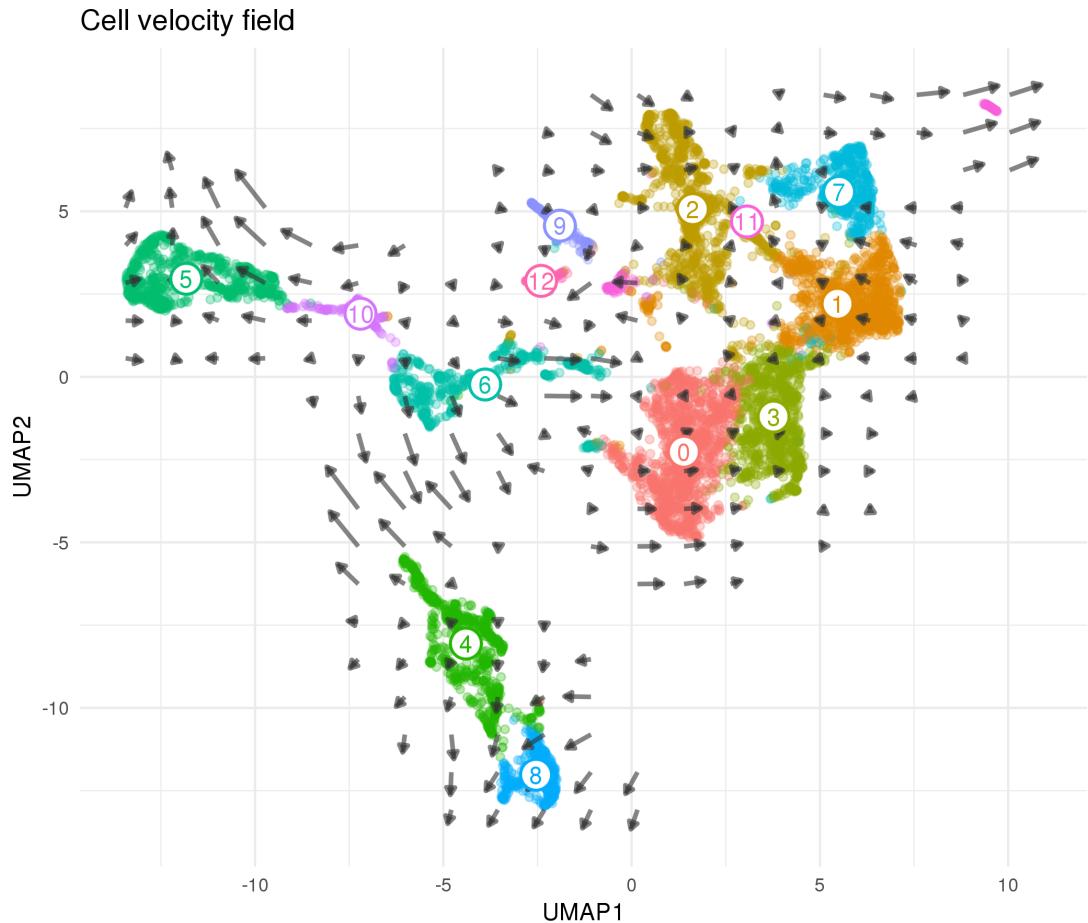


Figure 5.13: Cell velocity field calculated by *velocyto*. Points show cells in UMAP space coloured according to cluster. Arrows show the average velocity for cells in each region of a grid, with direction indicating the transcriptional profile cells are headed towards and length giving the rate of change.

5.9 Discussion

In this chapter I have presented an analysis of a kidney organoid scRNA-seq dataset, demonstrating a range of computational tools that can be used and focusing on decisions made during analysis and how these decisions affect the results. The dataset is a subset of the data presented in two previous publications where we investigated variability in the kidney organoid differentiation protocol and profiled the cell types that are present in these organoids. Our previous analysis included data from two batches of organoids as well as integration with a published human foetal kidney dataset but for this chapter I chose to use just the first batch of data which contains cells from three organoids. Data integration is an important area of research that has resulted in a range of methods produced recently, but it is still unclear which approaches are most effective. It can also be difficult to transfer the results of integration between analysis tools. As I wanted to demonstrate a range of tools here I have chosen to work with a single dataset, but how best to combine datasets is worthy of further consideration.

The most significant difference in my new analysis comes from one of the earliest steps. During the pre-processing stage of analysing droplet-based scRNA-seq data empty droplets must be separated from those that contain cells. While this step can have a significant impact on the results of an analysis it is often overlooked. I found that methods such as *EmptyDrops* that test for differences from the ambient RNA profile identified many more cells than the thresholding approach traditionally used in the *Cell Ranger* pipeline. Retaining cells that were identified by *EmptyDrops* or the more recent *Cell Ranger v3* which implements a similar approach produced

a dataset of 10002 cells. There are two possible explanations for the additional cells identified by these approaches. Either they are more sensitive at correctly selecting cell-containing droplets or they are more permissive and have incorrectly selected empty droplets. Quality control of the selected cells can help to determine which of these is true.

Typically, low-quality cells are removed from a dataset by setting thresholds on quantities that indicate that a cell is damaged or otherwise uninformative. I used a data-driven approach to select filtering thresholds that were a given number of MADs for the median for the total counts per cell, total expressed features per cell and the percentage of counts assigned to mitochondrial genes (which is a proxy for the health of a cell). These thresholds removed 1943 cells leaving 8059, including many of the additional cells identified by EmptyDrops. The fact that these cells were retained suggests that they are not obviously empty droplets and are likely to contain biologically relevant information. Using a statistical approach to select thresholds means that they can be automatically adapted to different datasets but it does assume that most cells are high-quality. For this reason it is still important to visualise distributions and where the thresholds lie. When I compared the thresholding approach to a fully-automated approach based on performing a PCA on a set of quality control metrics and detecting outliers I found that thresholding identified many more low-quality cells and was much easier to interpret.

The additional cells identified by EmptyDrops passed quality control but their biological significance does not become clear until later on in the analysis. Similar to the previously published analysis I used the graph-based clustering method in the Seurat package to form groups of cells with similar expression profiles. By using the clustering tree visualisation presented in **Chapter 4** I chose a resolution of 0.4 which produced 13 clusters. Most of the additional cells identified by EmptyDrops belong to two clusters (Cluster 4 and Cluster 6), but because these cells weren't present in the published analysis there is no clear overlap to associate them with a published cell type. To better interpret all the clusters I performed differential gene expression testing using edgeR, testing the cells in each cluster against the other cells in the dataset and looking for marker genes that are more highly expressed in a single cluster. Cluster 4 showed clear up-regulation of endothelial markers, while Cluster 6 had significant results for markers from various cell types including several genes associated with immune cells. Although not an expected part of the kidney organoid differentiation process, there is a plausible pathway by which immune cells could emerge, and when this dataset was combined with a human foetal kidney dataset we saw a small number of cells that clustered with immune cells and seemed to share similar expression. However, a significant amount of additional validation would be required to confirm that immune cells are present and that this apparent population isn't just the result of technical artifacts associated with the lower number of counts per cell. The other clusters were consistent with the previously published analysis and I assigned them to several kidney cell types, including several types of stroma, epithelium, mature podocytes and endothelium. I also observed several small off-target populations that were described previously, including neural progenitors, glial cells and muscle progenitors.

The Seurat procedure has been shown to perform well in comparison and consists of several steps: selecting a set of genes to use, performing PCA on those genes and selecting a set of principal components, building a shared nearest neighbour graph in PCA space, and using a modularity optimisation algorithm to partition the graph into clusters. For the published analysis we selected genes using the standard Seurat method which sets thresholds on gene expression level and variability. I compared genes selected in this way to those selected by the M3Drop method, which tests for genes with a greater number of zero counts than would be expected under a negative binomial model. Despite the significant differences between the two sets of genes the clusters identified by Seurat using the M3Drop genes were very similar to those in the published analysis. This suggests that graph-based clustering is fairly robust to exactly which genes are selected, as long as they capture enough of the variability in the dataset. The obvious difference between the two clusterings is the clusters containing the

EmptyDrops cells but it is reassuring that including these cells did not affect the other clusters and suggests that Seurat is also robust to the presence of some low-quality cells.

A standard analysis of an scRNA-seq dataset often ends here, but I also wanted to demonstrate how some newly developed methods can provide additional context to clusters. PAGA is a method that explores the connectivity between clusters, showing how they are related and reducing the dataset to a graph of clusters. When applied here I saw a high amount of connectivity amongst stromal clusters, and some of the expected developmental trajectories were also highly connected. Other clusters that we would expect to be similar, such as the two endothelial clusters, were not so related. Overall PAGA was not particularly effective when used on this dataset but I believe that is more likely to be a result of this specific dataset or the parameters selected rather than the approach itself. Another approach which gives additional context to a dataset is calculating cell velocity, which provides a transcriptional direction for each cell. Averaged across the dataset it can reveal where populations are headed or which populations are transcriptionally stable. The cell velocity field also highlighted some expected differentiation processes and showed cells in the possible immune cell population trending in opposite directions, suggesting that this cluster might contain sub-populations. Potentially this cluster may contain a mixture of immune cells, other cell types and low-quality cells.

Overall this new analysis is broadly consistent with the results we previously published, despite using different tools at several stages and making different decisions about how to set parameters and thresholds. The main differences between the two analyses are the result of the droplet selection step, with the test-based methods identifying many more cells. Including these cells in the analysis resulted in cluster containing a large number of cells only identified by EmptyDrops and showing a potential immune-like expression profile we hadn't observed before. A small change in parameters at this early stage could have hidden this population or introduced more low-quality cells, either of which would have had a large impact of the rest of the analysis. This emphasises the importance of carefully considering decisions during analysis, visualising the data to check and confirm them and iterating over an analysis if required. The overall consistency in results is reassuring and suggests that Seurat is robust to both the genes that are selected for clustering and the cells that are present. However, there is a large amount of heterogeneity in this dataset with many different cell types that can easily be separated. For an analysis where the goal was to identify more subtle differences, such as an experiment comparing patient-derived cells with cells from a corrected cell line, I would expect these decisions to be more important and to have a bigger impact on how the results are interpreted. Finally, new methods can extract additional information from an scRNA-seq dataset and provide additional evidence that helps with interpretation. I expect that methods like these will become a standard part of analyses, with multiple methods that approach a dataset in different ways being used to identify cell types and confirm hypotheses.

6

Conclusion

“Once you’ve decided that something’s absolutely true, you’ve closed your mind on it, and a closed mind doesn’t go anywhere. Question everything. That’s what education’s all about.”

— David and Leigh Eddings

Belgarath the Sorcerer, 1995

6. Conclusion

What did I do?

1. Build a database of tools
 - With a publicly available website
2. Develop a simulation package
 - Containing a flexible simulation model
3. Design a visualisation algorithm
 - Implemented in a software package
4. Analyse a kidney organoid dataset
 - Demonstrating different tools and decisions

What next?

Bigger data, more computation

Reference datasets

Benchmarking, robust methods, better software

Integration of data

More measurements, spatial data

My thesis explores the computational tools and techniques used to analyse single-cell RNA sequencing data. This type of data is the result of new technologies that have been developed over the last few years. It is now possible to capture individual cells, extract and isolate RNA from them and quantify the expression levels of genes in each cell. These new technologies allow us to inspect cells in a new way and open up a range of possibilities for biological studies, but usefulness of these technologies relies on our ability to interpret the data they produce. My PhD has coincided with a surge in the development of methods that attempt to make sense of this data and my thesis aims to chart this development and contribute to it. My contributions revolve around how we make decisions during an scRNA-seq analysis, which tools do we choose, how do we know that they work and what parameter values should we use?

My thesis contains four discrete but related projects that explore scRNA-seq analysis methods and decisions made during the analysis process.

In Chapter 2 - The scRNA-seq tools landscape I present my work developing and maintaining the scRNA-tools database, a database of software tools for analysing scRNA-seq data. I have recorded the details of over 450 tools including where they have been published, where the code is available, associated software licenses and the types of analysis they can perform. The scRNA-tools database allows researchers to identify focus areas of methods development, captures features of how tools are developed and shared, and displays trends in tool development over time. I have also designed a website interface to the database that makes this resource easily available to the research community. This website has been visited by hundreds of researchers from around the world. Over time new kinds of analysis have emerged and these could be integrated into the scRNA-tools database as categories to better organise the tools that are present. In the future the performance of the website could also be improved, particularly to better handle the large number of tools that it now displays.

Chapter 3 - Simulating scRNA-seq data describes Splatter, a software package for generating synthetic scRNA-seq datasets. Developers of new analysis tools should demonstrate their tools' effectiveness and simulations are an important way of evaluating computational methods. Splatter provides a consistent, easy-to-use interface to several published simulations models, allowing researchers to select the most appropriate model for their situation. This interface includes functions for estimating parameters from real datasets, objects for storing those parameters and functions for using them to produce synthetic datasets with associated intermediate values, in a standard format. One of these models is the Splat simulation we introduced in the Splatter publication. This model is based on a negative binomial distribution and includes many of the features considered important in scRNA-seq, data including variation in total counts per cell, relationships between the mean and variance of each gene, and the ability to add additional dropout based on mean expression. The Splat model has been designed to be flexible and can be used to generate datasets with different properties, including different groups of cells, batch effects and continuous trajectories. Splatter also contains functions for assessing the similarity of simulations to real scRNA-seq datasets and we used these to compare the performance of several simulation models. The results showed good performance for the Splat model overall, however we did see that performance varied with different capture technologies and tissue types. When comparing the models currently available in Splatter I saw that those with sophisticated estimation procedures, including BASiCS and ZINB-WaVE, performed extremely well. The performance of these newer models suggests that the Splat estimation procedure could be improved and doing so would improve its usefulness for developers. The simulation model could also be extended in various ways, for example by including an underlying gene network or by making it possible to use the same set of intermediate values for multiple simulations.

One of the analysis steps that has received a large amount of attention from scRNA-seq methods developers is using unsupervised clustering to form groups of cells with similar transcriptional protocols. These approaches could be assessed using simulations and each method will have some parameters that affect how many clusters are produced. The number of clusters used

in an analysis can affect how the results are interpreted but choosing these parameters is difficult. In **Chapter 4 - Visualising clustering across resolutions** I introduce clustering trees, an alternative, visualisation-based approach for selecting parameters that affect cluster number. For this approach a dataset is clustered multiple times at different resolutions and the overlap of samples between clusters at neighbouring resolutions is used to create a graph structure that is displayed as a tree. The structure of this tree can provide clues as to which resolution is most appropriate for an analysis, and more generally this visualisation can be used to display information across resolutions. How useful the clustering tree visualisation is depends somewhat on the clustering method used and it can become cluttered when there are many clusters present. Further work could look at improving these aspects of the algorithm and adapting it for fuzzy clustering methods. I also believe that the general idea of forming a graph of clusters may have other uses in a variety of applications. The clustering trees algorithm is implemented in a software package that is publicly available from the CRAN repository.

The final project of my PhD was to analyse a kidney organoid scRNA-seq dataset. This dataset was introduced in a publication investigating variability in the organoid differentiation protocol, where it was used to confirm that much of the variability is due to differences between batches rather than between organoids from the same batch. In a second publication, for which I performed the computational analysis, this dataset was profiled more thoroughly and the cell types present were identified. These included many of the cell types expected in the developing kidney as well as some off-target populations. To confirm these cell types we integrated the organoid data with a human foetal kidney dataset and found that many of the cell types were consistent. In **Chapter 5 - Analysis of kidney organoid scRNA-seq data** I re-analysed this dataset to demonstrate some alternative tools that we could have used in the analysis, focusing on how the decisions made at different stages affect the results. My main finding here was that using alternative droplet selection methods resulted in many more cells being included in the analysis. Continuing the analysis with these additional cells showed that many of them were from a population which showed some evidence of an immune-like signature. While immune cells are not expected in kidney organoids, and their presence would need to be validated experimentally, there is a plausible pathway by which they could emerge and we did see hints of their presence in the published analysis. My re-analysis also showed robustness of the clustering methods used to how genes were selected and the presence of the additional cells. For both the published analysis and the analysis in this thesis chapter I have followed a reproducible analysis workflow, including producing websites to display the analysis code and results and making the version controlled code publicly available.

The rise of single-cell technologies has been staggering and over the course of my PhD the field has moved from exploratory studies with a few hundred cells to detailed investigations of complex tissues and processes with tens or hundreds of thousands of cells. Although scRNA-seq technologies have only been available for a short time, they continue to improve and over the next few years we will continue to see significant changes. New platforms and protocols will be developed and used to generate even larger and more detailed datasets. Experiments that combine scRNA-seq with other single-cell measurements such as chromatin accessibility will become more common, as will spatial transcriptomics technologies that can retain a cell's position in a tissue while measuring its transcriptional profile.

The continued improvement in single-cell technologies will require further development of computational methods for analysing the data they produce, building on the significant work that has already been done. As bigger datasets become more common, tools will need to be carefully engineered to handle the computational workload. Bigger studies will also require methods to integrate multiple samples and different types of data. This has already become a significant area of research but it is a difficult problem and I expect it will take some time for the robustness and effectiveness of these methods to be established.

As I have discussed throughout my thesis, deciding which methods to use for various tasks is

a crucial part of scRNA-seq analysis. Currently these decisions are largely made based on the analyst's experience and intuition, what is common practice and how easy tools are to install and use. Benchmarking and comparison studies are vital for helping to understand which tools are effective, but they can be difficult and time consuming to perform. For some analysis tasks, such as differential expression testing, there are established evaluation metrics that can be used, but for others, such as ordering of cells or integration of datasets, it is less clear how to measure performance. Developing new metrics is vital for informative comparisons. As more benchmarking studies are performed, I hope the field crystallises around an established set of methods for common tasks and energy can be redirected towards alternative approaches, performing complex and novel analyses and improving the quality, usability and efficiency of existing tools.

Even when scRNA-seq analysis methods are reliable and effective, interpreting their results can be difficult and requires the input of a domain expert. Often there is a somewhat circular process where established knowledge is used to evaluate results from what is supposed to be an unsupervised process. To some degree this is the result of the novelty of the field and a reluctance to trust unfamiliar methods, but it is also true that we are looking at biology at a new resolution and this is challenging our current understanding. Large scale profiling projects such as the Human Cell Atlas will make interpretation easier by providing a reference to test against. When it is easier to assign identities to cells, it will become more practical to perform designed experiments such as studies that investigate the effect of a drug in a target cell type in a particular tissue. This has already begun to happen as exploratory analyses have been completed on many tissues. However, building these reference atlases is an extremely difficult task that will take many years and a well-coordinated global effort. Developing methods that best make use of these references will also require significant research but that is an effort that is already underway.

Single-cell RNA sequencing is an exciting technology that has opened up a world of possibilities, but we are still at the early stages of working out how best to use it. My contribution to this effort is in building a database that tracks the analysis tools that are available, and developing simulation software that can be used to evaluate methods. As well as deciding which tool to use, analysts need to choose parameters for those they select and I have developed a visualisation that can help choose parameters that change clustering resolution. I have also contributed to the biological literature by performing the computational analysis in a study that profiled the cell types present in kidney organoids and exploring how the decisions made during this analysis could affect the results and how they are interpreted. Overall I believe I have made a significant contribution to the emerging field of single-cell RNA sequencing data analysis during my PhD.

References

1. Watson JD, Crick FH. "Molecular structure of nucleic acids; a structure for deoxyribose nucleic acid." *Nature*. 1953;171:737–8. DOI: 10.1038/171737a0
2. Wilkins MHF, Stokes AR, Wilson HR. "Molecular structure of deoxypentose nucleic acids." *Nature*. 1953;171:738–40. DOI: 10.1038/171738a0
3. Franklin RE, Gosling RG. "Molecular configuration in sodium thymonucleate." *Nature*. 1953;171:740–1. DOI: 10.1038/171740a0
4. Raz T, Kapranov P, Lipson D, Letovsky S, Milos PM, Thompson JF. "Protocol dependence of sequencing-based gene expression measurements." *PLoS One*. 2011;6:e19287. DOI: 10.1371/journal.pone.0019287
5. Sultan M, Amstislavskiy V, Risch T, Schuette M, Dökel S, Ralser M, Balzereit D, Lehrach H, Yaspo M-L. "Influence of RNA extraction methods and library selection schemes on RNA-seq data." *BMC Genomics*. 2014;15:675. DOI: 10.1186/1471-2164-15-675
6. Voelker R, Small C, Bassham S, Catchen J, Sydes J, Cresko B. "RNA-seqlopedia." Available from: <https://web.archive.org/web/20181218043831/> <https://rnaseq.uoregon.edu/> Accessed: 2018-12-18
7. Mardis ER. "Next-generation DNA sequencing methods." *Annual Review of Genomics and Human Genetics*. 2008;9:387–402. DOI: 10.1146/annurev.genom.9.081307.164359
8. Dobin A, Davis CA, Schlesinger F, Drenkow J, Zaleski C, Jha S, Batut P, Chaisson M, Gingeras TR. "STAR: ultrafast universal RNA-seq aligner." *Bioinformatics*. 2013;29:15–21. DOI: 10.1093/bioinformatics/bts635
9. Kim D, Langmead B, Salzberg SL. "HISAT: a fast spliced aligner with low memory requirements." *Nature Methods*. 2015;12:357–60. DOI: 10.1038/nmeth.3317
10. Liao Y, Smyth GK, Shi W. "The Subread aligner: fast, accurate and scalable read mapping by seed-and-vote." *Nucleic Acids Research*. 2013;41:e108. DOI: 10.1093/nar/gkt214
11. Bray NL, Pimentel H, Melsted P, Pachter L. "Near-optimal probabilistic RNA-seq quantification." *Nature Biotechnology*. 2016;34:525–7. DOI: 10.1038/nbt.3519
12. Patro R, Duggal G, Love MI, Irizarry RA, Kingsford C. "Salmon provides fast and bias-aware quantification of transcript expression." *Nature Methods*. 2017;14:417–9. DOI: 10.1038/nmeth.4197
13. Robinson MD, Oshlack A. "A scaling normalization method for differential expression analysis of RNA-seq data." *Genome Biology*. 2010;11:R25. DOI: 10.1186/gb-2010-11-3-r25
14. Kim JK, Kolodziejczyk AA, Illicic T, Teichmann SA, Marioni JC. "Characterizing noise structure in single-cell RNA-seq distinguishes genuine from technical stochastic allelic expression." *Nature Communications*. 2015;6:8687. DOI: 10.1038/ncomms9687

15. Robinson MD, McCarthy DJ, Smyth GK. “edgeR: a Bioconductor package for differential expression analysis of digital gene expression data.” *Bioinformatics*. 2010;26:139–40. DOI: 10.1093/bioinformatics/btp616
16. McCarthy DJ, Chen Y, Smyth GK. “Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation.” *Nucleic Acids Research*. 2012;40:4288–97. DOI: 10.1093/nar/gks042
17. Anders S, Huber W. “Differential expression analysis for sequence count data.” *Genome Biology*. 2010;11:R106. DOI: 10.1186/gb-2010-11-10-r106
18. Love MI, Huber W, Anders S. “Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2.” *Genome Biology*. 2014;15:550. DOI: 10.1186/s13059-014-0550-8
19. Mortazavi A, Williams BA, McCue K, Schaeffer L, Wold B. “Mapping and quantifying mammalian transcriptomes by RNA-Seq.” *Nature Methods*. 2008;5:621–8. DOI: 10.1038/nmeth.1226
20. Wagner GP, Kin K, Lynch VJ. “Measurement of mRNA abundance using RNA-seq data: RPKM measure is inconsistent among samples.” *Theory in Biosciences*. 2012;131:281–5. DOI: 10.1007/s12064-012-0162-3
21. Ritchie ME, Phipson B, Wu D, Hu Y, Law CW, Shi W, Smyth GK. “limma powers differential expression analyses for RNA-sequencing and microarray studies.” *Nucleic Acids Research*. 2015;43:e47. DOI: 10.1093/nar/gkv007
22. Law CW, Chen Y, Shi W, Smyth GK. “voom: Precision weights unlock linear model analysis tools for RNA-seq read counts.” *Genome Biology*. 2014;15:R29. DOI: 10.1186/gb-2014-15-2-r29
23. Risso D, Ngai J, Speed TP, Dudoit S. “Normalization of RNA-seq data using factor analysis of control genes or samples.” *Nature Biotechnology*. 2014;32:896–902. DOI: 10.1038/nbt.2931
24. Tang F, Barbacioru C, Wang Y, Nordman E, Lee C, Xu N, Wang X, Bodeau J, Tuch BB, Siddiqui A, Lao K, Surani MA. “mRNA-Seq whole-transcriptome analysis of a single cell.” *Nature Methods*. 2009;6:377–82. DOI: 10.1038/nmeth.1315
25. Hashimshony T, Wagner F, Sher N, Yanai I. “CEL-Seq: single-cell RNA-Seq by multiplexed linear amplification.” *Cell Reports*. 2012;2:666–73. DOI: 10.1016/j.celrep.2012.08.003
26. Hashimshony T, Senderovich N, Avital G, Klochendler A, Leeuw Y de, Anavy L, Gennert D, Li S, Livak KJ, Rozenblatt-Rosen O, Dor Y, Regev A, Yanai I. “CEL-Seq2: sensitive highly-multiplexed single-cell RNA-Seq.” *Genome Biology*. 2016;17:77. DOI: 10.1186/s13059-016-0938-8
27. Sasagawa Y, Nikaido I, Hayashi T, Danno H, Uno KD, Imai T, Ueda HR. “Quartz-Seq: a highly reproducible and sensitive single-cell RNA sequencing method, reveals non-genetic gene-expression heterogeneity.” *Genome Biology*. 2013;14:R31. DOI: 10.1186/gb-2013-14-4-r31
28. Sasagawa Y, Danno H, Takada H, Ebisawa M, Tanaka K, Hayashi T, Kurisaki A, Nikaido I. “Quartz-Seq2: a high-throughput single-cell RNA-sequencing method that effectively uses limited sequence reads.” *Genome Biology*. 2018;19:29. DOI: 10.1186/s13059-018-1407-3
29. Picelli S, Björklund ÅK, Faridani OR, Sagasser S, Winberg G, Sandberg R. “Smart-seq2 for sensitive full-length transcriptome profiling in single cells.” *Nature Methods*. 2013;10:1096–8. DOI: 10.1038/nmeth.2639
30. Svensson V, Vento-Tormo R, Teichmann SA. “Exponential scaling of single-cell RNA-seq in the past decade.” *Nature Protocols*. 2018;13:599. DOI: 10.1038/nprot.2017.149
31. Macosko EZ, Basu A, Satija R, Nemesh J, Shekhar K, Goldman M, Tirosh I, Bialas AR, Kamitaki N, Martersteck EM, Trombetta JJ, Weitz DA, Sanes JR, Shalek AK, Regev A, McCarroll SA. “Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets.” *Cell*. 2015;161:1202–14. DOI: 10.1016/j.cell.2015.05.002

32. Klein AM, Mazutis L, Akartuna I, Tallapragada N, Veres A, Li V, Peshkin L, Weitz DA, Kirschner MW. "Droplet barcoding for single-cell transcriptomics applied to embryonic stem cells." *Cell.* 2015;161:1187–201. DOI: 10.1016/j.cell.2015.04.044
33. Zilionis R, Nainys J, Veres A, Savova V, Zemmour D, Klein AM, Mazutis L. "Single-cell barcoding and sequencing using droplet microfluidics." *Nature Protocols.* 2017;12:44–73. DOI: 10.1038/nprot.2016.154
34. Zheng GXY, Terry JM, Belgrader P, Ryvkin P, Bent ZW, Wilson R, Ziraldo SB, Wheeler TD, McDermott GP, Zhu J, Gregory MT, Shuga J, Montesclaros L, Underwood JG, Masquelier DA, Nishimura SY, Schnall-Levin M, Wyatt PW, et al. "Massively parallel digital transcriptional profiling of single cells." *Nature Communications.* 2017;8:14049. DOI: 10.1038/ncomms14049
35. Zhang X, Li T, Liu F, Chen Y, Yao J, Li Z, Huang Y, Wang J. "Comparative Analysis of Droplet-Based Ultra-High-Throughput Single-Cell RNA-Seq Systems." *Molecular Cell.* 2019;73:130–142.e5. DOI: 10.1016/j.molcel.2018.10.020
36. Kivioja T, Vähärautio A, Karlsson K, Bonke M, Enge M, Linnarsson S, Taipale J. "Counting absolute numbers of molecules using unique molecular identifiers." *Nature Methods.* 2012;9:72–4. DOI: 10.1038/nmeth.1778
37. Islam S, Zeisel A, Joost S, La Manno G, Zajac P, Kasper M, Lönnerberg P, Linnarsson S. "Quantitative single-cell RNA-seq with unique molecular identifiers." *Nature Methods.* 2014;11:163–6. DOI: 10.1038/nmeth.2772
38. Gierahn TM, Wadsworth MH 2nd, Hughes TK, Bryson BD, Butler A, Satija R, Fortune S, Love JC, Shalek AK. "Seq-Well: portable, low-cost RNA sequencing of single cells at high throughput." *Nature Methods.* 2017;14:395–8. DOI: 10.1038/nmeth.4179
39. Bose S, Wan Z, Carr A, Rizvi AH, Vieira G, Pe'er D, Sims PA. "Scalable microfluidics for single-cell RNA printing and sequencing." *Genome Biology.* 2015;16:120. DOI: 10.1186/s13059-015-0684-3
40. Hu P, Fabyanic E, Kwon DY, Tang S, Zhou Z, Wu H. "Dissecting Cell-Type Composition and Activity-Dependent Transcriptional State in Mammalian Brains by Massively Parallel Single-Nucleus RNA-Seq." *Molecular Cell.* 2017;68:1006–1015.e7. DOI: 10.1016/j.molcel.2017.11.017
41. Gao R, Kim C, Sei E, Foukakis T, Crosetto N, Chan L-K, Srinivasan M, Zhang H, Meric-Bernstam F, Navin N. "Nanogrid single-nucleus RNA sequencing reveals phenotypic diversity in breast cancer." *Nature Communications.* 2017;8:228. DOI: 10.1038/s41467-017-00244-w
42. Habib N, Avraham-Davidi I, Basu A, Burks T, Shekhar K, Hofree M, Choudhury SR, Aguet F, Gelfand E, Ardlie K, Weitz DA, Rozenblatt-Rosen O, Zhang F, Regev A. "Massively parallel single-nucleus RNA-seq with DroNc-seq." *Nature Methods.* 2017;14:955–8. DOI: 10.1038/nmeth.4407
43. Wu H, Kirita Y, Donnelly EL, Humphreys BD. "Advantages of Single-Nucleus over Single-Cell RNA Sequencing of Adult Kidney: Rare Cell Types and Novel Cell States Revealed in Fibrosis." *Journal of the American Society of Nephrology: JASN.* 2018;30:23–32. DOI: 10.1681/ASN.2018090912
44. Stoeckius M, Hafemeister C, Stephenson W, Houck-Loomis B, Chattopadhyay PK, Swerdlow H, Satija R, Smibert P. "Simultaneous epitope and transcriptome measurement in single cells." *Nature Methods.* 2017;14:865–8. DOI: 10.1038/nmeth.4380
45. Stoeckius M, Zheng S, Houck-Loomis B, Hao S, Yeung BZ, Mauck WM 3rd, Smibert P, Satija R. "Cell Hashing with barcoded antibodies enables multiplexing and doublet detection for single cell genomics." *Genome Biology.* 2018;19:224. DOI: 10.1186/s13059-018-1603-1
46. Moudgil A. "Multimodal scRNA-seq." 2019. DOI: 10.5281/zenodo.2628012

47. Technology Innovation Lab, New York Genome Center. “cite-seq.com.” Available from: <https://web.archive.org/web/20181218054100/https://cite-seq.com/> Accessed: 2018-12-18
48. Spanjaard B, Hu B, Mitic N, Olivares-Chauvet P, Janjuha S, Ninov N, Junker JP. “Simultaneous lineage tracing and cell-type identification using CRISPR-Cas9-induced genetic scars.” *Nature Biotechnology*. 2018;36:469–73. DOI: 10.1038/nbt.4124
49. Dixit A, Parnas O, Li B, Chen J, Fulco CP, Jerby-Arnon L, Marjanovic ND, Dionne D, Burks T, Raychowdhury R, Adamson B, Norman TM, Lander ES, Weissman JS, Friedman N, Regev A. “Perturb-Seq: Dissecting Molecular Circuits with Scalable Single-Cell RNA Profiling of Pooled Genetic Screens.” *Cell*. 2016;167:1853–1866.e17. DOI: 10.1016/j.cell.2016.11.038
50. Mimitou EP, Cheng A, Montalbano A, Hao S, Stoeckius M, Legut M, Roush T, Herrera A, Papalexi E, Ouyang Z, Satija R, Sanjana NE, Koralov SB, Smibert P. “Multiplexed detection of proteins, transcriptomes, clonotypes and CRISPR perturbations in single cells.” *Nature Methods*. 2019;16:409–12. DOI: 10.1038/s41592-019-0392-0
51. Biddy BA, Kong W, Kamimoto K, Guo C, Waye SE, Sun T, Morris SA. “Single-cell mapping of lineage and identity in direct reprogramming.” *Nature*. 2018;564:219–24. DOI: 10.1038/s41586-018-0744-4
52. Macaulay IC, Haerty W, Kumar P, Li YI, Hu TX, Teng MJ, Goolam M, Saurat N, Coupland P, Shirley LM, Smith M, Van der Aa N, Banerjee R, Ellis PD, Quail MA, Swerdlow HP, Zernicka-Goetz M, Livesey FJ, et al. “G&T-seq: parallel sequencing of single-cell genomes and transcriptomes.” *Nature Methods*. 2015;12:519–22. DOI: 10.1038/nmeth.3370
53. Han KY, Kim K-T, Joung J-G, Son D-S, Kim YJ, Jo A, Jeon H-J, Moon H-S, Yoo CE, Chung W, Eum HH, Kim S, Kim HK, Lee JE, Ahn M-J, Lee H-O, Park D, Park W-Y. “SIDR: simultaneous isolation and parallel sequencing of genomic DNA and total RNA from single cells.” *Genome Research*. 2017; DOI: 10.1101/gr.223263.117
54. Wang LY, Guo J, Cao W, Zhang M, He J, Li Z. “Integrated sequencing of exome and mRNA of large-sized single cells.” *Scientific Reports*. 2018;8:384. DOI: 10.1038/s41598-017-18730-y
55. Hu Y, Huang K, An Q, Du G, Hu G, Xue J, Zhu X, Wang C-Y, Xue Z, Fan G. “Simultaneous profiling of transcriptome and DNA methylome from a single cell.” *Genome Biology*. 2016;17:88. DOI: 10.1186/s13059-016-0950-z
56. Cadwell CR, Scala F, Li S, Livrizzi G, Shen S, Sandberg R, Jiang X, Tolias AS. “Multimodal profiling of single-cell morphology, electrophysiology, and gene expression using Patch-seq.” *Nature Protocols*. 2017;12:2531. DOI: 10.1038/nprot.2017.120
57. Bian S, Hou Y, Zhou X, Li X, Yong J, Wang Y, Wang W, Yan J, Hu B, Guo H, Wang J, Gao S, Mao Y, Dong J, Zhu P, Xiu D, Yan L, Wen L, et al. “Single-cell multiomics sequencing and analyses of human colorectal cancer.” *Science*. 2018;362:1060–3. DOI: 10.1126/science.aa03791
58. Grün D, Kester L, Oudenaarden A van. “Validation of noise models for single-cell transcriptomics.” *Nature Methods*. 2014;11:637–40. DOI: 10.1038/nmeth.2930
59. Liu S, Trapnell C. “Single-cell transcriptome sequencing: recent advances and remaining challenges.” *F1000Research*. 2016;5. DOI: 10.12688/f1000research.7223.1
60. Adam M, Potter AS, Potter SS. “Psychrophilic proteases dramatically reduce single cell RNA-seq artifacts: A molecular atlas of kidney development.” *Development*. 2017;144:3625–32. DOI: 10.1242/dev.151142
61. Hicks SC, Townes FW, Teng M, Irizarry RA. “Missing data and technical variability in single-cell RNA-sequencing experiments.” *Biostatistics*. 2017; DOI: 10.1093/biostatistics/kxx053
62. Pierson E, Yau C. “ZIFA: Dimensionality reduction for zero-inflated single-cell gene expression analysis.” *Genome Biology*. 2015;16:241. DOI: 10.1186/s13059-015-0805-z

63. Risso D, Perraudeau F, Gribkova S, Dudoit S, Vert J-P. "A general and flexible method for signal extraction from single-cell RNA-seq data." *Nature Communications*. 2018;9:284. DOI: 10.1038/s41467-017-02554-5
64. Van den Berge K, Perraudeau F, Soneson C, Love MI, Risso D, Vert J-P, Robinson MD, Dudoit S, Clement L. "Observation weights unlock bulk RNA-seq tools for zero inflation and single-cell applications." *Genome Biology*. 2018;19:24. DOI: 10.1186/s13059-018-1406-4
65. Miao Z, Deng K, Wang X, Zhang X. "DEsingle for detecting three types of differential expression in single-cell RNA-seq data." *Bioinformatics*. 2018; DOI: 10.1093/bioinformatics/bty332
66. Svensson V. "Droplet scRNA-seq is not zero-inflated." *bioRxiv*. 2019. p. 582064. DOI: 10.1101/582064
67. Dijk D van, Sharma R, Nainys J, Yim K, Kathail P, Carr AJ, Burdziak C, Moon KR, Chaffer CL, Pattabiraman D, Bierie B, Mazutis L, Wolf G, Krishnaswamy S, Pe'er D. "Recovering Gene Interactions from Single-Cell Data Using Data Diffusion." *Cell*. 2018;174:P716–729.E27. DOI: 10.1016/j.cell.2018.05.061
68. Huang M, Wang J, Torre E, Dueck H, Shaffer S, Bonasio R, Murray JI, Raj A, Li M, Zhang NR. "SAVER: gene expression recovery for single-cell RNA sequencing." *Nature Methods*. 2018;15:539–42. DOI: 10.1038/s41592-018-0033-z
69. Wang J, Agarwal D, Huang M, Hu G, Zhou Z, Conley VB, MacMullan H, Zhang NR. "Transfer learning in single-cell transcriptomics improves data denoising and pattern discovery." *bioRxiv*. 2018. p. 457879. DOI: 10.1101/457879
70. Li WV, Li JJ. "An accurate and robust imputation method scImpute for single-cell RNA-seq data." *Nature Communications*. 2018;9:997. DOI: 10.1038/s41467-018-03405-7
71. Andrews TS, Hemberg M. "False signals induced by single-cell imputation." *F1000Research*. 2018;7. DOI: 10.12688/f1000research.16613.1
72. Zeisel A, Muñoz-Manchado AB, Codeluppi S, Lönnberg P, La Manno G, Juréus A, Marques S, Munguba H, He L, Betsholtz C, Rolny C, Castelo-Branco G, Hjerling-Leffler J, Linnarsson S. "Brain structure. Cell types in the mouse cortex and hippocampus revealed by single-cell RNA-seq." *Science*. 2015;347:1138–42. DOI: 10.1126/science.aaa1934
73. Patel AP, Tirosh I, Trombetta JJ, Shalek AK, Gillespie SM, Wakimoto H, Cahill DP, Nahed BV, Curry WT, Martuza RL, Louis DN, Rozenblatt-Rosen O, Suvà ML, Regev A, Bernstein BE. "Single-cell RNA-seq highlights intratumoral heterogeneity in primary glioblastoma." *Science*. 2014;344:1396–401. DOI: 10.1126/science.1254257
74. Treutlein B, Brownfield DG, Wu AR, Neff NF, Mantalas GL, Espinoza FH, Desai TJ, Krasnow MA, Quake SR. "Reconstructing lineage hierarchies of the distal lung epithelium using single-cell RNA-seq." *Nature*. 2014;509:371–5. DOI: 10.1038/nature13173
75. Usoskin D, Furlan A, Islam S, Abdo H, Lönnberg P, Lou D, Hjerling-Leffler J, Haeggström J, Kharchenko O, Kharchenko PV, Linnarsson S, Ernfors P. "Unbiased classification of sensory neuron types by large-scale single-cell RNA sequencing." *Nature Neuroscience*. 2015;18:145–53. DOI: 10.1038/nn.3881
76. Buettner F, Natarajan KN, Casale FP, Proserpio V, Scialdone A, Theis FJ, Teichmann SA, Marioni JC, Stegle O. "Computational analysis of cell-to-cell heterogeneity in single-cell RNA-sequencing data reveals hidden subpopulations of cells." *Nature Biotechnology*. 2015;33:155–60. DOI: 10.1038/nbt.3102
77. Trapnell C, Cacchiarelli D, Grimsby J, Pokharel P, Li S, Morse M, Lennon NJ, Livak KJ, Mikkelsen TS, Rinn JL. "The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells." *Nature Biotechnology*. 2014;32:381–6. DOI: 10.1038/nbt.2859

78. Regev A, Teichmann SA, Lander ES, Amit I, Benoist C, Birney E, Bodenmiller B, Campbell P, Carninci P, Clatworthy M, Clevers H, Deplancke B, Dunham I, Eberwine J, Eils R, Enard W, Farmer A, Fugger L, et al. "The Human Cell Atlas." *eLife*. 2017;6. DOI: 10.7554/eLife.27041
79. Cao J, Packer JS, Ramani V, Cusanovich DA, Huynh C, Daza R, Qiu X, Lee C, Furlan SN, Steemers FJ, Adey A, Waterston RH, Trapnell C, Shendure J. "Comprehensive single-cell transcriptional profiling of a multicellular organism." *Science*. 2017;357:661–7. DOI: 10.1126/science.aam8940
80. Tabula Muris Consortium, Overall coordination, Logistical coordination, Organ collection and processing, Library preparation and sequencing, Computational data analysis, Cell type annotation, Writing group, Supplemental text writing group, Principal investigators. "Single-cell transcriptomics of 20 mouse organs creates a Tabula Muris." *Nature*. 2018;562:367–72. DOI: 10.1038/s41586-018-0590-4
81. Wu B, Li Y, Liu Y, Jin K, Zhao K, An C, Li Q, Gong L, Zhao W, Hu J, Qian J, Ouyang H, Zou X. "Cell atlas of human uterus." *bioRxiv*. 2018. p. 267849. DOI: 10.1101/267849
82. Han X, Wang R, Zhou Y, Fei L, Sun H, Lai S, Saadatpour A, Zhou Z, Chen H, Ye F, Huang D, Xu Y, Huang W, Jiang M, Jiang X, Mao J, Chen Y, Lu C, et al. "Mapping the Mouse Cell Atlas by Microwell-Seq." *Cell*. 2018;172:1091–1107.e17. DOI: 10.1016/j.cell.2018.02.001
83. Saunders A, Macosko E, Wysoker A, Goldman M, Krienen F, Rivera H de, Bien E, Baum M, Wang S, Goeva A, Nemesh J, Kamitaki N, Brumbaugh S, Kulp D, McCarroll SA. "A Single-Cell Atlas of Cell Types, States, and Other Transcriptional Patterns from Nine Regions of the Adult Mouse Brain." *bioRxiv*. 2018. p. 299081. DOI: 10.1101/299081
84. Plass M, Solana J, Wolf FA, Ayoub S, Misios A, Glažar P, Obermayer B, Theis FJ, Kocks C, Rajewsky N. "Cell type atlas and lineage tree of a whole complex animal by single-cell transcriptomics." *Science*. 2018;360. DOI: 10.1126/science.aaq1723
85. Bhaduri A, Nowakowski TJ, Pollen AA, Kriegstein AR. "Identification of cell types in a mouse brain single-cell atlas using low sampling coverage." *BMC Biology*. 2018;16:113. DOI: 10.1186/s12915-018-0580-x
86. Yuan H, Yan M, Zhang G, Liu W, Deng C, Liao G, Xu L, Luo T, Yan H, Long Z, Shi A, Zhao T, Xiao Y, Li X. "CancerSEA: a cancer single-cell state atlas." *Nucleic Acids Research*. 2018;47:D900–8. DOI: 10.1093/nar/gky939
87. Davie K, Janssens J, Koldere D, De Waegeneer M, Pech U, Kreft Ł, Aibar S, Makhzami S, Christiaens V, Bravo González-Blas C, Poovathingal S, Hulselmans G, Spanier KI, Moerman T, Vanspauwen B, Geurs S, Voet T, Lammertyn J, et al. "A Single-Cell Transcriptome Atlas of the Aging Drosophila Brain." *Cell*. 2018;174:982–998.e20. DOI: 10.1016/j.cell.2018.05.057
88. Taylor DM, Aronow BJ, Tan K, Bernt K, Salomonis N, Greene CS, Frolova A, Henrickson SE, Wells A, Pei L, Jaiswal JK, Whitsett J, Hamilton KE, MacParland SA, Kelsen J, Heuckerth RO, Potter SS, Vella LA, et al. "The Pediatric Cell Atlas: Defining the Growth Phase of Human Development at Single-Cell Resolution." *Developmental Cell*. 2019; DOI: 10.1016/j.devcel.2019.03.001
89. Lun ATL, McCarthy DJ, Marioni JC. "A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor." *F1000Research*. 2016;5:2122. DOI: 10.12688/f1000research.9501.2
90. Perraudeau F, Risso D, Street K, Purdom E, Dudoit S. "Bioconductor workflow for single-cell RNA sequencing: Normalization, dimensionality reduction, clustering, and lineage inference." *F1000Research*. 2017;6. DOI: 10.12688/f1000research.12122.1
91. Srivastava A, Malik L, Smith T, Sudbery I, Patro R. "Alevin efficiently estimates accurate gene abundances from dscRNA-seq data." *Genome Biology*. 2019;20:65. DOI: 10.1186/s13059-019-1670-y

92. Smith T, Heger A, Sudbery I. "UMI-tools: modeling sequencing errors in Unique Molecular Identifiers to improve quantification accuracy." *Genome Research*. 2017;27:491–9. DOI: 10.1101/gr.209601.116
93. Svensson V, Natarajan KN, Ly L-H, Miragaia RJ, Labalette C, Macaulay IC, Cvejic A, Teichmann SA. "Power analysis of single-cell RNA-sequencing experiments." *Nature Methods*. 2017;14:381–7. DOI: 10.1038/nmeth.4220
94. Parekh S, Ziegenhain C, Vieth B, Enard W, Hellmann I. "zUMIs - A fast and flexible pipeline to process RNA sequencing data with UMIs." *GigaScience*. 2018;7. DOI: 10.1093/gigascience/giy059
95. Tian L, Su S, Dong X, Amann-Zalcenstein D, Biben C, Seidi A, Hilton DJ, Naik SH, Ritchie ME. "scPipe: A flexible R/Bioconductor preprocessing pipeline for single-cell RNA-sequencing data." *PLoS Computational Biology*. 2018;14:e1006361. DOI: 10.1371/journal.pcbi.1006361
96. Yang A, Troup M, Lin P, Ho JWK. "Falco: a quick and flexible single-cell RNA-seq processing framework on the cloud." *Bioinformatics*. 2017;33:767–9. DOI: 10.1093/bioinformatics/btw732
97. DePasquale EAK, Schnell DJ, Valiente I, Blaxall BC, Grimes HL, Singh H, Salomonis N. "DoubletDecon: Cell-State Aware Removal of Single-Cell RNA-Seq Doublets." *bioRxiv*. 2018. p. 364810. DOI: 10.1101/364810
98. McGinnis CS, Murrow LM, Gartner ZJ. "DoubletFinder: Doublet Detection in Single-Cell RNA Sequencing Data Using Artificial Nearest Neighbors." *Cell Systems*. 2019;8:329–337.e4. DOI: 10.1016/j.cels.2019.03.003
99. Lun ATL, Riesenfeld S, Andrews T, Dao TP, Gomes T, participants in the 1st Human Cell Atlas Jamboree, Marioni JC. "EmptyDrops: distinguishing cells from empty droplets in droplet-based single-cell RNA sequencing data." *Genome Biology*. 2019;20:63. DOI: 10.1186/s13059-019-1662-y
100. Ilicic T, Kim JK, Kolodziejczyk AA, Bagger FO, McCarthy DJ, Marioni JC, Teichmann SA. "Classification of low quality cells from single-cell RNA-seq data." *Genome Biology*. 2016;17:29. DOI: 10.1186/s13059-016-0888-1
101. McCarthy DJ, Campbell KR, Lun ATL, Wills QF. "Scater: pre-processing, quality control, normalization and visualization of single-cell RNA-seq data in R." *Bioinformatics*. 2017;33:1179–86. DOI: 10.1093/bioinformatics/btw777
102. Leng N, Choi J, Chu L-F, Thomson JA, Kendziorski C, Stewart R. "OEFinder: a user interface to identify and visualize ordering effects in single-cell RNA-seq data." *Bioinformatics*. 2016; DOI: 10.1093/bioinformatics/btw004
103. Chen B, Herring CA, Lau KS. "pyNVR: Investigating factors affecting feature selection from scRNA-seq data for lineage reconstruction." *Bioinformatics*. 2018; DOI: 10.1093/bioinformatics/bty950
104. Andrews TS, Hemberg M. "M3Drop: Dropout-based feature selection for scRNASeq." *Bioinformatics*. 2018; DOI: 10.1093/bioinformatics/bty1044
105. Phipson B, Zappia L, Oshlack A. "Gene length and detection bias in single cell RNA sequencing protocols." *F1000Research*. 2017;6. DOI: 10.12688/f1000research.11290.1
106. Brennecke P, Anders S, Kim JK, Kołodziejczyk AA, Zhang X, Proserpio V, Baying B, Benes V, Teichmann SA, Marioni JC, Heisler MG. "Accounting for technical noise in single-cell RNA-seq experiments." *Nature Methods*. 2013;10:1093–5. DOI: 10.1038/nmeth.2645
107. Ding B, Zheng L, Zhu Y, Li N, Jia H, Ai R, Wildberg A, Wang W. "Normalization and noise reduction for single cell RNA-seq experiments." *Bioinformatics*. 2015;31:2225–7. DOI: 10.1093/bioinformatics/btv122

108. Vallejos CA, Marioni JC, Richardson S. "BASiCS: Bayesian Analysis of Single-Cell Sequencing data." *PLoS Computational Biology*. 2015;11:e1004333. DOI: 10.1371/journal.pcbi.1004333
109. Lun ATL, Bach K, Marioni JC. "Pooling across cells to normalize single-cell RNA sequencing data with many zero counts." *Genome Biology*. 2016;17:1–14. DOI: 10.1186/s13059-016-0947-7
110. Eling N, Richard AC, Richardson S, Marioni JC, Vallejos CA. "Correcting the Mean-Variance Dependency for Differential Variability Testing Using Single-Cell RNA Sequencing Data." *Cell Systems*. 2018;0. DOI: 10.1016/j.cels.2018.06.011
111. Stuart T, Satija R. "Integrative single-cell analysis." *Nature Reviews. Genetics*. 2019; DOI: 10.1038/s41576-019-0093-7
112. Park J-E, Polanski K, Meyer K, Teichmann SA. "Fast Batch Alignment of Single Cell Transcriptomes Unifies Multiple Mouse Cell Atlases into an Integrated Landscape." *bioRxiv*. 2018. p. 397042. DOI: 10.1101/397042
113. Gao X, Hu D, Gogol M, Li H. "ClusterMap: Compare multiple Single Cell RNA-Seq datasets across different experimental conditions." *Bioinformatics*. 2019; DOI: 10.1093/bioinformatics/btz024
114. Büttner M, Miao Z, Wolf FA, Teichmann SA, Theis FJ. "A test metric for assessing single-cell RNA-seq batch correction." *Nature Methods*. 2019;16:43–9. DOI: 10.1038/s41592-018-0254-1
115. Welch J, Kozareva V, Ferreira A, Vanderburg C, Martin C, Macosko E. "Integrative inference of brain cell similarities and differences from single-cell genomics." *bioRxiv*. 2018. p. 459891. DOI: 10.1101/459891
116. Mereu E, Iacono G, Guillaumet-Adkins A, Moutinho C, Lunazzi G, Santos C, Miguel-Escalada I, Ferrer J, Real FX, Gut I, Heyn H. "matchSCore: Matching Single-Cell Phenotypes Across Tools and Experiments." *bioRxiv*. 2018. p. 314831. DOI: 10.1101/314831
117. Hie B, Bryson B, Berger B. "Efficient integration of heterogeneous single-cell transcriptomes using Scanorama." *Nature Biotechnology*. 2019; DOI: 10.1038/s41587-019-0113-3
118. Lin Y, Ghazanfar S, Wang KYX, Gagnon-Bartsch JA, Lo KK, Su X, Han Z-G, Ormerod JT, Speed TP, Yang P, Yang JYH. "scMerge leverages factor analysis, stable expression, and pseudoreplication to merge multiple single-cell RNA-seq datasets." *Proceedings of the National Academy of Sciences of the United States of America*. 2019; DOI: 10.1073/pnas.1820006116
119. Butler A, Hoffman P, Smibert P, Papalexi E, Satija R. "Integrating single-cell transcriptomic data across different conditions, technologies, and species." *Nature Biotechnology*. 2018; DOI: 10.1038/nbt.4096
120. Hotelling H. "RELATIONS BETWEEN TWO SETS OF VARIATES." *Biometrika*. 1936;28:321–77. DOI: 10.1093/biomet/28.3-4.321
121. Hardoon DR, Szedmak S, Shawe-Taylor J. "Canonical correlation analysis: an overview with application to learning methods." *Neural Computation*. 2004;16:2639–64. DOI: 10.1162/0899766042321814
122. Berndt DJ, Clifford J. "Using Dynamic Time Warping to Find Patterns in Time Series." *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*. Seattle, WA; 1994. pp. 359–70. Available from: <http://dl.acm.org/citation.cfm?id=3000850.3000887>
123. Haghverdi L, Lun ATL, Morgan MD, Marioni JC. "Batch effects in single-cell RNA-sequencing data are corrected by matching mutual nearest neighbors." *Nature Biotechnology*. 2018; DOI: 10.1038/nbt.4091
124. Stuart T, Butler A, Hoffman P, Hafemeister C, Papalexi E, Mauck WM 3rd, Hao Y, Stoeckius M, Smibert P, Satija R. "Comprehensive Integration of Single-Cell Data." *Cell*. 2019;177:1888–1902.e21. DOI: 10.1016/j.cell.2019.05.031

125. Kiselev VY, Andrews TS, Hemberg M. "Challenges in unsupervised clustering of single-cell RNA-seq data." *Nature Reviews. Genetics.* 2019; DOI: 10.1038/s41576-018-0088-9
126. Guo M, Wang H, Potter SS, Whitsett JA, Xu Y. "SINCERA: a pipeline for single-cell RNA-seq profiling analysis." *PLoS Computational Biology.* 2015;11:e1004575. DOI: 10.1371/journal.pcbi.1004575
127. Kiselev VY, Kirschner K, Schaub MT, Andrews T, Yiu A, Chandra T, Natarajan KN, Reik W, Barahona M, Green AR, Hemberg M. "SC3: consensus clustering of single-cell RNA-seq data." *Nature Methods.* 2017;14:483–6. DOI: 10.1038/nmeth.4236
128. Anchang B, Hart TDP, Bendall SC, Qiu P, Bjornson Z, Linderman M, Nolan GP, Plevritis SK. "Visualization and cellular hierarchy inference of single-cell data using SPADE." *Nature Protocols.* 2016;11:1264–79. DOI: 10.1038/nprot.2016.066
129. Satija R, Farrell JA, Gennert D, Schier AF, Regev A. "Spatial reconstruction of single-cell gene expression data." *Nature Biotechnology.* 2015;33:495–502. DOI: 10.1038/nbt.3192
130. Duò A, Robinson MD, Soneson C. "A systematic performance evaluation of clustering methods for single-cell RNA-seq data." *F1000Research.* 2018;7. DOI: 10.12688/f1000research.15666.1
131. Freytag S, Tian L, Lönnstedt I, Ng M, Bahlo M. "Comparison of clustering tools in R for medium-sized 10x Genomics single-cell RNA-sequencing data." *F1000Research.* 2018;7. DOI: 10.12688/f1000research.15809.1
132. Kim T, Chen IR, Lin Y, Wang AY-Y, Yang JYH, Yang P. "Impact of similarity metrics on single-cell RNA-seq data clustering." *Briefings in Bioinformatics.* 2018; DOI: 10.1093/bib/bby076
133. Blondel VD, Guillaume J-L, Lambiotte R, Lefebvre E. "Fast unfolding of communities in large networks." *Journal of Statistical Mechanics.* 2008;2008:P10008. DOI: 10.1088/1742-5468/2008/10/P10008
134. Kiselev VY, Yiu A, Hemberg M. "scmap: projection of single-cell RNA-seq data across data sets." *Nature Methods.* 2018; DOI: 10.1038/nmeth.4644
135. Alquicira-Hernández J, Sathe A, Ji HP, Nguyen Q, Powell JE. "scPred: Cell type prediction at single-cell resolution." *bioRxiv.* 2018. p. 369538. DOI: 10.1101/369538
136. Lieberman Y, Rokach L, Shay T. "CaSTLe - Classification of single cells by transfer learning: Harnessing the power of publicly available single cell RNA sequencing experiments to annotate new experiments." *PloS One.* 2018;13:e0205499. DOI: 10.1371/journal.pone.0205499
137. Wagner F, Yanai I. "Moana: A robust and scalable cell type classification framework for single-cell RNA-Seq data." *bioRxiv.* 2018. p. 456129. DOI: 10.1101/456129
138. Qiu X, Mao Q, Tang Y, Wang L, Chawla R, Pliner HA, Trapnell C. "Reversed graph embedding resolves complex single-cell trajectories." *Nature Methods.* 2017;14:979–82. DOI: 10.1038/nmeth.4402
139. Ji Z, Ji H. "TSCAN: Pseudo-time reconstruction and evaluation in single-cell RNA-seq analysis." *Nucleic Acids Research.* 2016; DOI: 10.1093/nar/gkw430
140. Welch JD, Hartemink AJ, Prins JF. "SLICER: inferring branched, nonlinear cellular trajectories from single cell RNA-seq data." *Genome Biology.* 2016;17:106. DOI: 10.1186/s13059-016-0975-3
141. duVerle DA, Yotsukura S, Nomura S, Aburatani H, Tsuda K. "CellTree: an R/bioconductor package to infer the hierarchical structure of cell populations from single-cell RNA-seq data." *BMC Bioinformatics.* 2016;17:363. DOI: 10.1186/s12859-016-1175-6
142. Juliá M, Telenti A, Rausell A. "Sincell: an R/Bioconductor package for statistical assessment

- of cell-state hierarchies from single-cell RNA-seq.” *Bioinformatics*. 2015;31:3380–2. DOI: 10.1093/bioinformatics/btv368
143. Chen J, Schlitzer A, Chakarov S, Ginhoux F, Poidinger M. “Mpath maps multi-branching single-cell trajectories revealing progenitor cell progression during development.” *Nature Communications*. 2016;7:11988. DOI: 10.1038/ncomms11988
144. Street K, Risso D, Fletcher RB, Das D, Ngai J, Yosef N, Purdom E, Dudoit S. “Slingshot: cell lineage and pseudotime inference for single-cell transcriptomics.” *BMC Genomics*. 2018;19:477. DOI: 10.1186/s12864-018-4772-0
145. Cannoodt R, Saelens W, Yvan S. “Computational methods for trajectory inference from single-cell transcriptomics.” *European Journal of Immunology*. 2016;46:2496–506. DOI: 10.1002/eji.201646347
146. Pearson K. “On lines and planes of closest fit to systems of points in space.” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*. 1901;2:559–72. DOI: 10.1080/14786440109462720
147. Maaten L van der, Hinton G. “Visualizing Data using t-SNE.” *Journal of Machine Learning Research*. 2008;9:2579–605. Available from: <http://www.jmlr.org/papers/v9/vandermaaten08a.html>
148. McInnes L, Healy J. “UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction.” 2018; Available from: <http://arxiv.org/abs/1802.03426>
149. Saelens W, Cannoodt R, Todorov H, Saeys Y. “A comparison of single-cell trajectory inference methods.” *Nature Biotechnology*. 2019; DOI: 10.1038/s41587-019-0071-9
150. Svensson V, Pachter L. “RNA Velocity: Molecular Kinetics from Single-Cell RNA-Seq.” *Molecular Cell*. 2018;72:7–9. DOI: 10.1016/j.molcel.2018.09.026
151. La Manno G, Soldatov R, Zeisel A, Braun E, Hochgerner H, Petukhov V, Lidschreiber K, Kastriti ME, Lönnberg P, Furlan A, Fan J, Borm LE, Liu Z, Bruggen D van, Guo J, He X, Barker R, Sundström E, et al. “RNA velocity of single cells.” *Nature*. 2018;560:494–8. DOI: 10.1038/s41586-018-0414-6
152. Delmans M, Hemberg M. “Discrete distributional differential expression (D3E)—a tool for gene expression analysis of single-cell RNA-seq data.” *BMC Bioinformatics*. 2016;17:110. DOI: 10.1186/s12859-016-0944-6
153. Finak G, McDavid A, Yajima M, Deng J, Gersuk V, Shalek AK, Slichter CK, Miller HW, McElrath MJ, Prlic M, Linsley PS, Gottardo R. “MAST: a flexible statistical framework for assessing transcriptional changes and characterizing heterogeneity in single-cell RNA sequencing data.” *Genome Biology*. 2015;16:278. DOI: 10.1186/s13059-015-0844-5
154. Korthauer KD, Chu L-F, Newton MA, Li Y, Thomson J, Stewart R, Kendziora C. “A statistical approach for identifying differential distributions in single-cell RNA-seq experiments.” *Genome Biology*. 2016;17:222. DOI: 10.1186/s13059-016-1077-y
155. Kharchenko PV, Silberstein L, Scadden DT. “Bayesian approach to single-cell differential expression analysis.” *Nature Methods*. 2014;11:740–2. DOI: 10.1038/nmeth.2967
156. Soneson C, Robinson MD. “Bias, robustness and scalability in single-cell differential expression analysis.” *Nature Methods*. 2018;15:255–61. DOI: 10.1038/nmeth.4612
157. Qiu X, Hill A, Packer J, Lin D, Ma Y-A, Trapnell C. “Single-cell mRNA quantification and differential analysis with Census.” *Nature Methods*. 2017;14:309–15. DOI: 10.1038/nmeth.4150
158. Zhang JM, Kamath GM, Tse DN. “Valid post-clustering differential analysis for single-cell RNA-Seq.” *bioRxiv*. 2019. p. 463265. DOI: 10.1101/463265

159. Edsgård D, Reinius B, Sandberg R. "scphaser: haplotype inference using single-cell RNA-seq data." *Bioinformatics*. 2016; DOI: 10.1093/bioinformatics/btw484
160. Reinius B, Mold JE, Ramsköld D, Deng Q, Johnsson P, Michaëlsson J, Frisén J, Sandberg R. "Analysis of allelic expression patterns in clonal somatic cells by single-cell RNA-seq." *Nature Genetics*. 2016;48:1430–5. DOI: 10.1038/ng.3678
161. Jiang Y, Zhang NR, Li M. "SCALE: modeling allele-specific gene expression by single-cell RNA sequencing." *Genome Biology*. 2017;18:74. DOI: 10.1186/s13059-017-1200-8
162. Choi K, Raghupathy N, Churchill GA. "scBASE: A Bayesian mixture model for the analysis of allelic expression in single cells." *bioRxiv*. 2019. p. 383224. DOI: 10.1101/383224
163. Welch JD, Hu Y, Prins JF. "Robust detection of alternative splicing in a population of single cells." *Nucleic Acids Research*. 2016;44:e73. DOI: 10.1093/nar/gkv1525
164. Huang Y, Sanguinetti G. "BRIE: transcriptome-wide splicing quantification in single cells." *Genome Biology*. 2017;18:123. DOI: 10.1186/s13059-017-1248-5
165. Song Y, Botvinnik OB, Lovci MT, Kakaradov B, Liu P, Xu JL, Yeo GW. "Single-cell alternative splicing analysis with Expedition reveals splicing dynamics during neuron differentiation." *Molecular Cell*. 2017;67:148–161.e5. DOI: 10.1016/j.molcel.2017.06.003
166. Poirion O, Zhu X, Ching T, Garmire LX. "Using single nucleotide variations in single-cell RNA-seq to identify subpopulations and genotype-phenotype linkage." *Nature Communications*. 2018;9:4892. DOI: 10.1038/s41467-018-07170-5
167. Ding J, Lin C, Bar-Joseph Z. "Cell lineage inference from SNP and scRNA-Seq data." *Nucleic Acids Research*. 2019; DOI: 10.1093/nar/gkz146
168. Petti AA, Williams SR, Miller CA, Fiddes IT, Srivatsan SN, Chen DY, Fronick CC, Fulton RS, Church DM, Ley TJ. "Mutation detection in thousands of acute myeloid leukemia cells using single cell RNA-sequencing." *bioRxiv*. 2018. p. 434746. DOI: 10.1101/434746
169. Lindeman I, Emerton G, Mamanova L, Snir O, Polanski K, Qiao S-W, Sollid LM, Teichmann SA, Stubbington MJT. "BraCeR: B-cell-receptor reconstruction and clonality inference from single-cell RNA-seq." *Nature Methods*. 2018;15:563–5. DOI: 10.1038/s41592-018-0082-3
170. Afik S, Raulet G, Yosef N. "Reconstructing B cell receptor sequences from short-read single cell RNA-sequencir with BRAPeS." *Bioinformatics*. bioRxiv; 2018. p. e148.
171. Rizzetto S, Koppstein DNP, Samir J, Singh M, Reed JH, Cai CH, Lloyd AR, Eltahla AA, Goodnow CC, Luciani F. "B-cell receptor reconstruction from single-cell RNA-seq with VDJPuzzle." *Bioinformatics*. 2018;34:2846–7. DOI: 10.1093/bioinformatics/bty203
172. Stubbington MJT, Lönnberg T, Proserpio V, Clare S, Speak AO, Dougan G, Teichmann SA. "T cell fate and clonality inference from single-cell transcriptomes." *Nature Methods*. 2016;13:329–32. DOI: 10.1038/nmeth.3800
173. Afik S, Yates KB, Bi K, Darko S, Godec J, Gerdemann U, Swadling L, Douek DC, Klenerman P, Barnes EJ, Sharpe AH, Haining WN, Yosef N. "Targeted reconstruction of T cell receptor sequence from single cell RNA-seq links CDR3 length to T cell differentiation state." *Nucleic Acids Research*. 2017;45:e148. DOI: 10.1093/nar/gkx615
174. Fan J, Lee H-O, Lee S, Ryu D-E, Lee S, Xue C, Kim SJ, Kim K, Barkas N, Park PJ, Park W-Y, Kharchenko PV. "Linking transcriptional and genetic tumor heterogeneity through allele analysis of single-cell RNA-seq data." *Genome Research*. 2018;28:1217–27. DOI: 10.1101/gr.228080.117
175. Tian L, Dong X, Freytag S, Lê Cao K-A, Su S, JalalAbadi A, Amann-Zalcenstein D, Weber TS, Seidi A, Jabbari JS, Naik SH, Ritchie ME. "Benchmarking single cell RNA-sequencing analysis pipelines using mixture control experiments." *Nature Methods*. 2019; DOI: 10.1038/s41592-019-0425-8

176. Bertram JF, Douglas-Denton RN, Diouf B, Hughson MD, Hoy WE. "Human nephron number: implications for health and disease." *Pediatric Nephrology*. 2011;26:1529–33. DOI: 10.1007/s00467-011-1843-8
177. Australian Bureau of Statistics. "National Health Survey: First Results, 2017-18." Commonwealth of Australia; Commonwealth of Australia; 2018 Dec. Available from: <http://www.abs.gov.au/ausstats/abs@.nsf/Lookup/by%20Subject/4364.0.55.001~2017-18~Main%20Features~Kidney%20disease~65>
178. Australian Bureau of Statistics. "Causes of Death, Australia, 2017." Commonwealth of Australia; Commonwealth of Australia; 2018 Sep. Available from: <http://www.abs.gov.au/AUSSTATS/abs@.nsf/Lookup/3303.0Main+Features12017?OpenDocument>
179. Patel SR, Dressler GR. "The genetics and epigenetics of kidney development." *Seminars in Nephrology*. 2013;33:314–26. DOI: 10.1016/j.semephrol.2013.05.004
180. Little MH. "Improving our resolution of kidney morphogenesis across time and space." *Current Opinion in Genetics & Development*. 2015;32:135–43. DOI: 10.1016/j.gde.2015.03.001
181. Lindström NO, McMahon JA, Guo J, Tran T, Guo Q, Rutledge E, Parvez RK, Saribekyan G, Schuler RE, Liao C, Kim AD, Abdelhalim A, Ruffins SW, Thornton ME, Baskin L, Grubbs B, Kesselman C, McMahon AP. "Conserved and Divergent Features of Human and Mouse Kidney Organogenesis." *Journal of the American Society of Nephrology: JASN*. 2018;29:785–805. DOI: 10.1681/ASN.2017080887
182. Tian P, Lennon R. "The myriad possibility of kidney organoids." *Current Opinion in Nephrology and Hypertension*. 2019;28:211–8. DOI: 10.1097/MNH.0000000000000498
183. Takahashi K, Yamanaka S. "Induction of pluripotent stem cells from mouse embryonic and adult fibroblast cultures by defined factors." *Cell*. 2006;126:663–76. DOI: 10.1016/j.cell.2006.07.024
184. Nakagawa M, Koyanagi M, Tanabe K, Takahashi K, Ichisaka T, Aoi T, Okita K, Mochiduki Y, Takizawa N, Yamanaka S. "Generation of induced pluripotent stem cells without Myc from mouse and human fibroblasts." *Nature Biotechnology*. 2008;26:101–6. DOI: 10.1038/nbt1374
185. Fatehullah A, Tan SH, Barker N. "Organoids as an in vitro model of human development and disease." *Nature Cell Biology*. 2016;18:246–54. DOI: 10.1038/ncb3312
186. Takasato M, Er PX, Chiu HS, Maier B, Baillie GJ, Ferguson C, Parton RG, Wolvetang EJ, Roost MS, Chuva de Sousa Lopes SM, Little MH. "Kidney organoids from human iPS cells contain multiple lineages and model human nephrogenesis." *Nature*. 2015;526:564–8. DOI: 10.1038/nature15695
187. Phipson B, Er PX, Combes AN, Forbes TA, Howden SE, Zappia L, Yen H-J, Lawlor KT, Hale LJ, Sun J, Wolvetang E, Takasato M, Oshlack A, Little MH. "Evaluation of variability in human kidney organoids." *Nature Methods*. 2019;16:79–87. DOI: 10.1038/s41592-018-0253-2
188. Little MH, Hale LJ, Howden SE, Kumar SV. "Generating Kidney from Stem Cells." *Annual Review of Physiology*. 2019;81:335–57. DOI: 10.1146/annurev-physiol-020518-114331
189. Bartfeld S, Clevers H. "Stem cell-derived organoids and their application for medical research and patient treatment." *Journal of Molecular Medicine*. 2017;95:729–38. DOI: 10.1007/s00109-017-1531-7
190. Kumar SV, Er PX, Lawlor KT, Motazedian A, Scurr M, Ghobrial I, Combes AN, Zappia L, Oshlack A, Stanley EG, Little MH. "Kidney micro-organoids in suspension culture as a scalable source of human pluripotent stem cell-derived kidney cells." *Development*. 2019;146. DOI: 10.1242/dev.172361
191. Wilson PC, Humphreys BD. "Kidney and organoid single-cell transcriptomics: the end of the beginning." *Pediatric Nephrology*. 2019;1–7. DOI: 10.1007/s00467-018-4177-y

192. Brazovskaja A, Treutlein B, Camp JG. "High-throughput single-cell transcriptomics on organoids." *Current Opinion in Biotechnology*. 2018;55:167–71. DOI: 10.1016/j.copbio.2018.11.002
193. Park J, Shrestha R, Qiu C, Kondo A, Huang S, Werth M, Li M, Barasch J, Suszták K. "Single-cell transcriptomics of the mouse kidney reveals potential cellular targets of kidney disease." *Science*. 2018;360:758–63. DOI: 10.1126/science.aar2131
194. Young MD, Mitchell TJ, Vieira Braga FA, Tran MGB, Stewart BJ, Ferdinand JR, Collard G, Botting RA, Popescu D-M, Loudon KW, Vento-Tormo R, Stephenson E, Cagan A, Farndon SJ, Del Castillo Velasco-Herrera M, Guzzo C, Richoz N, Mamanova L, et al. "Single-cell transcriptomes from human kidneys reveal the cellular identity of renal tumors." *Science*. 2018;361:594–9. DOI: 10.1126/science.aat1699
195. Lindström NO, Guo J, Kim AD, Tran T, Guo Q, De Sena Brandine G, Ransick A, Parvez RK, Thornton ME, Basking L, Grubbs B, McMahon JA, Smith AD, McMahon AP. "Conserved and Divergent Features of Mesenchymal Progenitor Cell Types within the Cortical Nephrogenic Niche of the Human and Mouse Kidney." *Journal of the American Society of Nephrology: JASN*. 2018;29:806–24. DOI: 10.1681/ASN.2017080890
196. Wu H, Uchimura K, Donnelly EL, Kirita Y, Morris SA, Humphreys BD. "Comparative Analysis and Refinement of Human PSC-Derived Kidney Organoid Differentiation with Single-Cell Transcriptomics." *Cell Stem Cell*. 2018;23:869–881.e8. DOI: 10.1016/j.stem.2018.10.010
197. Davis S, Kutum R, Zappia L, Sorenson J, Kiselev V, Olivier P, Botvinnik O, Korthauer K, Gitter A, Campbell KR, Hickey P, Tuncel MA, MikeDMorgan, markrobinsonuzh, Vallejos C, Wang Z, Salomonis N, dyl4nm4rsh4ll, et al. "Awesome Single Cell." DOI: 10.5281/zenodo.1294021
198. Zappia L, Phipson B, Oshlack A. "Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database." *PLoS Computational Biology*. 2018;14:e1006245. DOI: 10.1371/journal.pcbi.1006245
199. Huber W, Carey VJ, Gentleman R, Anders S, Carlson M, Carvalho BS, Bravo HC, Davis S, Gatto L, Girke T, Gottardo R, Hahne F, Hansen KD, Irizarry RA, Lawrence M, Love MI, MacDonald J, Obenchain V, et al. "Orchestrating high-throughput genomic analysis with Bioconductor." *Nature Methods*. 2015;12:115–21. DOI: 10.1038/nmeth.3252
200. Amezquita RA, Carey VJ, Carpp LN, Geistlinger L, Lun ATL, Marini F, Rue-Albrecht K, Risso D, Soneson C, Waldron L, Pagès H, Smith M, Huber W, Morgan M, Gottardo R, Hicks SC. "Orchestrating Single-Cell Analysis with Bioconductor." *bioRxiv*. 2019. p. 590562. DOI: 10.1101/590562
201. Zappia L, Phipson B, Oshlack A. "Splatter: simulation of single-cell RNA sequencing data." *Genome Biology*. 2017;18:174. DOI: 10.1186/s13059-017-1305-0
202. Lun A, Risso D. "SingleCellExperiment: S4 Classes for Single Cell Data." 2017. Available from: <http://bioconductor.org/packages/SingleCellExperiment/>
203. Campbell KR, Yau C. "Probabilistic modeling of bifurcations in single-cell gene expression data using a Bayesian mixture of factor analyzers." *Wellcome Open Research*. 2017;2:19. DOI: 10.12688/wellcomeopenres.11087.1
204. Campbell KR, Yau C. "Uncovering pseudotemporal trajectories with covariates from single cell and bulk expression data." *Nature Communications*. 2018;9:2442. DOI: 10.1038/s41467-018-04696-6
205. Barron M, Zhang S, Li J. "A sparse differential clustering algorithm for tracing cell type changes via single-cell RNA-sequencing data." *Nucleic Acids Research*. 2017;46:e14. DOI: 10.1093/nar/gkx1113
206. Tung P-Y, Blischak JD, Hsiao CJ, Knowles DA, Burnett JE, Pritchard JK, Gilad Y. "Batch

- effects and the effective design of single-cell gene expression studies.” *Scientific Reports*. 2017;7:39921. DOI: 10.1038/srep39921
207. Soneson C, Robinson MD. “Towards unified quality verification of synthetic count data with countsimQC.” *Bioinformatics*. 2017; DOI: 10.1093/bioinformatics/btx631
208. Zappia L, Oshlack A. “Clustering trees: a visualization for evaluating clusterings at multiple resolutions.” *GigaScience*. 2018;7. DOI: 10.1093/gigascience/giy083
209. Pedersen TL. “tidygraph: A Tidy API for Graph Manipulation.” 2018. Available from: <https://CRAN.R-project.org/package=tidygraph>
210. Csardi G, Nepusz T. “The igraph software package for complex network research.” *Inter-Journal, Complex Systems*. 2006;1695:1–9.
211. Pedersen TL. “ggraph: An Implementation of Grammar of Graphics for Graphs and Networks.” 2018. Available from: <https://CRAN.R-project.org/package=ggraph>
212. Anderson E. “The Irises of the Gaspe Peninsula.” *Bulletin of the American Iris Society*. 1935;59:2–5.
213. Fisher RA. “The use of multiple measurements in taxonomic problems.” *Annals of Eugenics*. 1936;7:179–88. DOI: 10.1111/j.1469-1809.1936.tb02137.x
214. Combes AN, Zappia L, Er PX, Oshlack A, Little MH. “Single-cell analysis reveals congruence between kidney organoids and human fetal kidney.” *Genome Medicine*. 2019;11:3. DOI: 10.1186/s13073-019-0615-0
215. Lawlor KT, Zappia L, Lefevre J, Park J-S, Hamilton NA, Oshlack A, Little MH, Combes AN. “Nephron progenitor commitment is a stochastic process influenced by cell migration.” *eLife*. 2019;8:e41156. DOI: 10.7554/eLife.41156
216. Zappia L, Combes AN, Er PX, Oshlack A, Little MH. “Combes organoid paper analysis code.” 2019. DOI: 10.5281/zenodo.2548990
217. Zappia L. “PhD thesis analysis.” 2019. DOI: 10.5281/zenodo.2622384
218. Lex A, Gehlenborg N, Strobelt H, Vuillemot R, Pfister H. “UpSet: Visualization of Intersecting Sets.” *IEEE Transactions on Visualization and Computer Graphics*. 2014;20:1983–92. DOI: 10.1109/TVCG.2014.2346248
219. Levenshtein VI. “Binary codes capable of correcting deletions, insertions, and reversals.” *Soviet physics doklady*. 1966. pp. 707–10.
220. Smedley D, Haider S, Durinck S, Pandini L, Provero P, Allen J, Arnaiz O, Awedh MH, Baldock R, Barbiera G, Bardou P, Beck T, Blake A, Bonierbale M, Brookes AJ, Bucci G, Buetti I, Burge S, et al. “The BioMart community portal: an innovative alternative to large, centralized data repositories.” *Nucleic Acids Research*. 2015;43:W589–98. DOI: 10.1093/nar/gkv350
221. Durinck S, Moreau Y, Kasprzyk A, Davis S, De Moor B, Brazma A, Huber W. “BioMart and Bioconductor: a powerful link between biological databases and microarray data analysis.” *Bioinformatics*. 2005;21:3439–40. DOI: 10.1093/bioinformatics/bti525
222. Scialdone A, Natarajan KN, Saraiva LR, Proserpio V, Teichmann SA, Stegle O, Marioni JC, Buettner F. “Computational assignment of cell-cycle stage from single-cell transcriptome data.” *Methods*. 2015;85:54–61. DOI: 10.1016/j.ymeth.2015.06.021
223. Becht E, McInnes L, Healy J, Dutertre C-A, Kwok IWH, Ng LG, Ginhoux F, Newell EW. “Dimensionality reduction for visualizing single-cell data using UMAP.” *Nature Biotechnology*. 2018; DOI: 10.1038/nbt.4314
224. Sidiropoulos N, Sohi SH, Pedersen TL, Porse BT, Winther O, Rapin N, Bagger FO. “SinaPlot: An Enhanced Chart for Simple and Truthful Representation of Single Observations Over

- Multiple Classes.” *Journal of Computational and Graphical Statistics: A Joint Publication of American Statistical Association, Institute of Mathematical Statistics, Interface Foundation of North America.* 2018;27:673–6. DOI: 10.1080/10618600.2017.1366914
225. Jaakkola MK, Seyednasrollah F, Mehmoor A, Elo LL. “Comparison of methods to detect differentially expressed genes between single-cell populations.” *Briefings in Bioinformatics.* 2016; DOI: 10.1093/bib/bbw057
226. Wolf FA, Hamey FK, Plass M, Solana J, Dahlin JS, Göttgens B, Rajewsky N, Simon L, Theis FJ. “PAGA: graph abstraction reconciles clustering with trajectory inference through a topology preserving map of single cells.” *Genome Biology.* 2019;20:59. DOI: 10.1186/s13059-019-1663-x
227. Wolf FA, Angerer P, Theis FJ. “SCANPY: large-scale single-cell gene expression data analysis.” *Genome Biology.* 2018;19:15. DOI: 10.1186/s13059-017-1382-0
228. R Core Team. “R: A language and environment for statistical computing.” Vienna, Austria: R Foundation for Statistical Computing; 2018.
229. Lun A, Risso D. “SingleCellExperiment: S4 classes for single cell data.” 2019. R package version 1.4.1
230. Jacomy M, Venturini T, Heymann S, Bastian M. “ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software.” *PloS One.* 2014;9:e98679. DOI: 10.1371/journal.pone.0098679
231. Wickham H. “ggplot2: Elegant Graphics for Data Analysis.” Springer New York; 2010. ISBN: 9780387981406
232. Wilke CO. “cowplot: Streamlined plot theme and plot annotations for ‘ggplot2’.” 2018. R package version 0.9.3
233. Gehlenborg N. “UpSetR: A more scalable alternative to Venn and Euler diagrams for visualizing intersecting sets.” 2017. R package version 1.3.3
234. Auguie B. “GridExtra: Miscellaneous functions for “grid” graphics.” 2017. R package version 2.3
235. Pedersen TL. “ggforce: Accelerating ‘ggplot2’.” 2018. R package version 0.1.3
236. Wickham H. “tidyverse: Easily install and load the ‘tidyverse’.” 2017. R package version 1.2.1
237. Wickham H, François R, Henry L, Müller K. “dplyr: A grammar of data manipulation.” 2018. R package version 0.7.8
238. Wickham H, Henry L. “tidyr: Easily tidy data with ‘spread()’ and ‘gather()’ functions.” 2018. R package version 0.8.2
239. Henry L, Wickham H. “purrr: Functional programming tools.” 2018. R package version 0.2.5
240. Morgan M, Van Twisk D. “LoomExperiment: LoomExperiment container.” 2019. R package version 1.0.2
241. Blischak J, Carbonetto P, Stephens M. “workflowr: A framework for reproducible and collaborative data science.” 2018. R package version 1.1.1
242. Xie Y. “knitr: A Comprehensive Tool for Reproducible Research in R.” In: Stodden V, Leisch F, Peng RD, editors. *Implementing Reproducible Research.* CRC Press; 2014. ISBN: 9781466561595
243. Xie Y. “Dynamic Documents with R and knitr.” CRC Press; 2016. DOI: 10.1201/b15166 ISBN: 9781482203547

244. Xie Y. “knitr: A general-purpose package for dynamic report generation in R.” 2018. R package version 1.20
245. Xie Y, Allaire JJ, Grolemund G. “R Markdown: The Definitive Guide.” CRC Press LLC; 2018. ISBN: 9781138359338
246. Allaire J, Xie Y, McPherson J, Luraschi J, Ushey K, Atkins A, Wickham H, Cheng J, Chang W. “rmarkdown: Dynamic documents for R.” 2018. R package version 1.10
247. Xie Y. “bookdown: Authoring books and technical documents with R Markdown.” 2018. R package version 0.7
248. Xie Y. “Bookdown: Authoring Books and Technical Documents with R Markdown.” CRC Press; 2016. ISBN: 9781138700109
249. Zappia L, Oshlack A. “clustree: Visualise clusterings at different resolutions.” 2019. R package version 0.3.0
250. Wickham H. “forcats: Tools for working with categorical variables (factors).” 2018. R package version 0.3.0
251. Wickham H, Chang W, Henry L, Pedersen TL, Takahashi K, Wilke C, Woo K. “ggplot2: Create elegant data visualisations using the grammar of graphics.” 2018. R package version 3.1.0
252. Hester J. “glue: Interpreted string literals.” 2018. R package version 1.3.0
253. Edmondson M. “googleAnalyticsR: Google Analytics API into R.” 2018. R package version 0.6.0
254. Edmondson M. “googleAuthR: Authenticate and create Google APIs.” 2018. R package version 0.7.0
255. Müller K. “here: A simpler way to find your files.” 2017. R package version 0.1
256. Ooms J, Temple Lang D, Hilaiel L. “jsonlite: A robust, high performance JSON parser and generator for R.” 2018. R package version 1.6
257. Ooms J. “The jsonlite Package: A Practical and Consistent Mapping Between JSON Data and R Objects.” 2014; Available from: <http://arxiv.org/abs/1403.2805>
258. Spinu V, Grolemund G, Wickham H. “lubridate: Make dealing with dates a little easier.” 2018. R package version 1.7.4
259. Grolemund G, Wickham H. “Dates and Times Made Easy with lubridate.” *Journal of Statistical Software, Articles*. 2011;40:1–25. DOI: 10.18637/jss.v040.i03
260. Bache SM, Wickham H. “magrittr: A forward-pipe operator for R.” 2014. R package version 1.5
261. Ushey K, McPherson J, Cheng J, Atkins A, Allaire J. “packrat: A dependency management system for projects and their R package dependencies.” 2018. R package version 0.4.9-3
262. Neuwirth E. “RColorBrewer: ColorBrewer palettes.” 2014. R package version 1.1-2
263. Wickham H, Hester J, Francois R. “readr: Read rectangular text data.” 2018. R package version 1.3.1
264. Csárdi G, Wickham H, Chang W, Hester J, Morgan M, Tenenbaum D. “remotes: R package installation from remote repositories, including ‘GitHub’.” 2018. R package version 2.0.0
265. Wickham H. “stringr: Simple, consistent wrappers for common string operations.” 2018. R package version 1.3.1
266. Garnier S. “viridis: Default color maps from ‘matplotlib’.” 2018. R package version 0.5.1

A

Splatter additional files

A.1 Additional figures

Additional figures

List of additional figures

- Additional Figure 1** - Simple simulation model
- Additional Figure 2** - Lun simulation model
- Additional Figure 3** - Lun 2 simulation model
- Additional Figure 4** - Benchmarking (genes)
- Additional Figure 5** - Benchmarking (cells)
- Additional Figure 6** - Camp dataset comparison
- Additional Figure 7** - Klein dataset comparison
- Additional Figure 8** - Tung dataset comparison
- Additional Figure 9** - Zeisel dataset comparison
- Additional Figure 10** - Engel dataset comparison
- Additional Figure 11** - Dataset processing times
- Additional Figure 12** - MADs heatmap
- Additional Figure 13** - Filtered Tung dataset comparison
- Additional Figure 14** - Gene goodness-of-fit
- Additional Figure 15** - Example non-linear gene
- Additional Figure 16** - Dispersion estimation
- Additional Figure 17** - Mean-zeros relationship

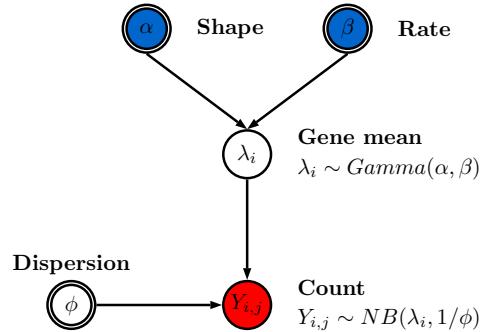


Figure 1: Diagram of the Simple simulation model. Input parameters are indicated with double borders, blue shading shows those that can be estimated from real data. Red shading indicates the final output.

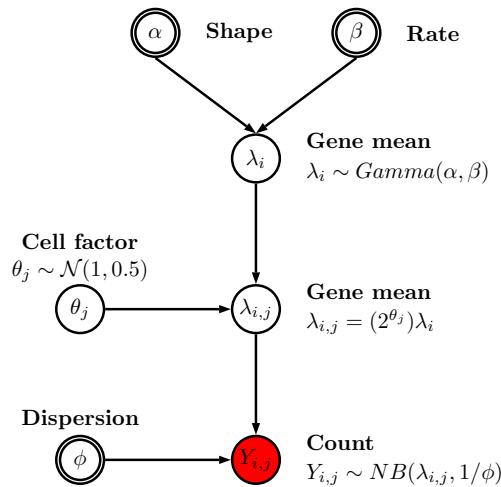


Figure 2: Diagram of the core Lun simulation model. Input parameters are indicated with double borders, blue shading shows those that can be estimated from real data. Red shading indicates the final output.

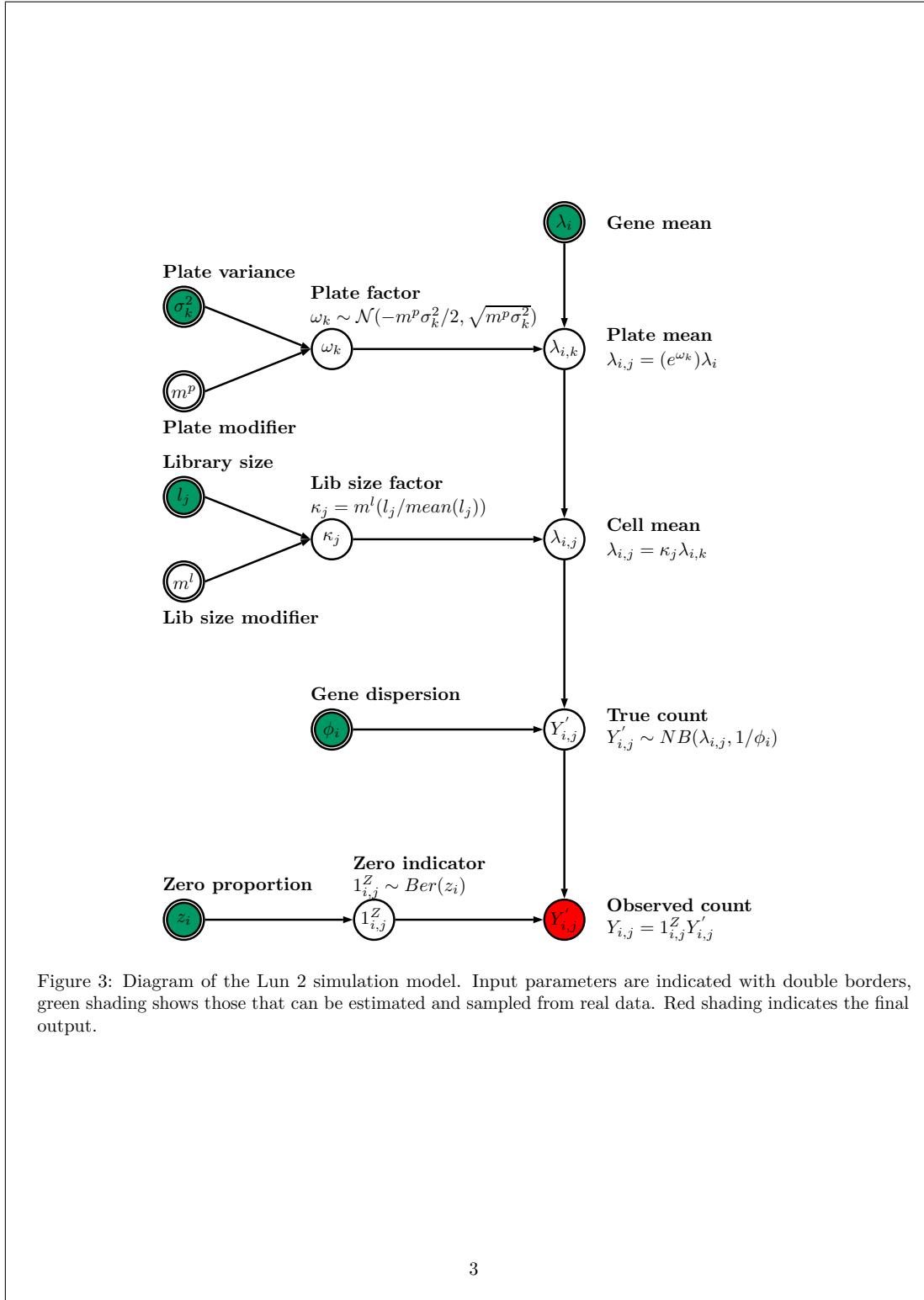
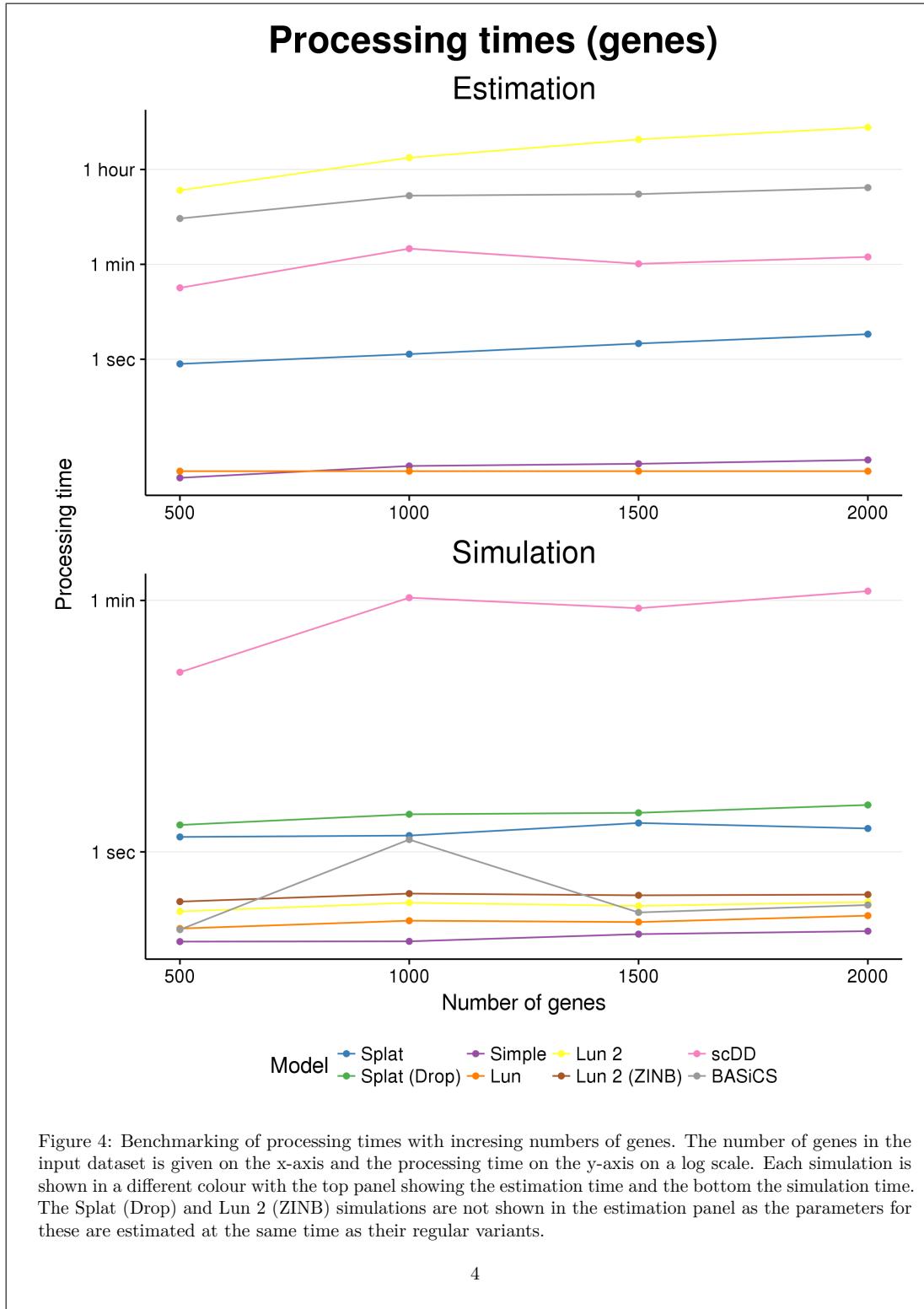
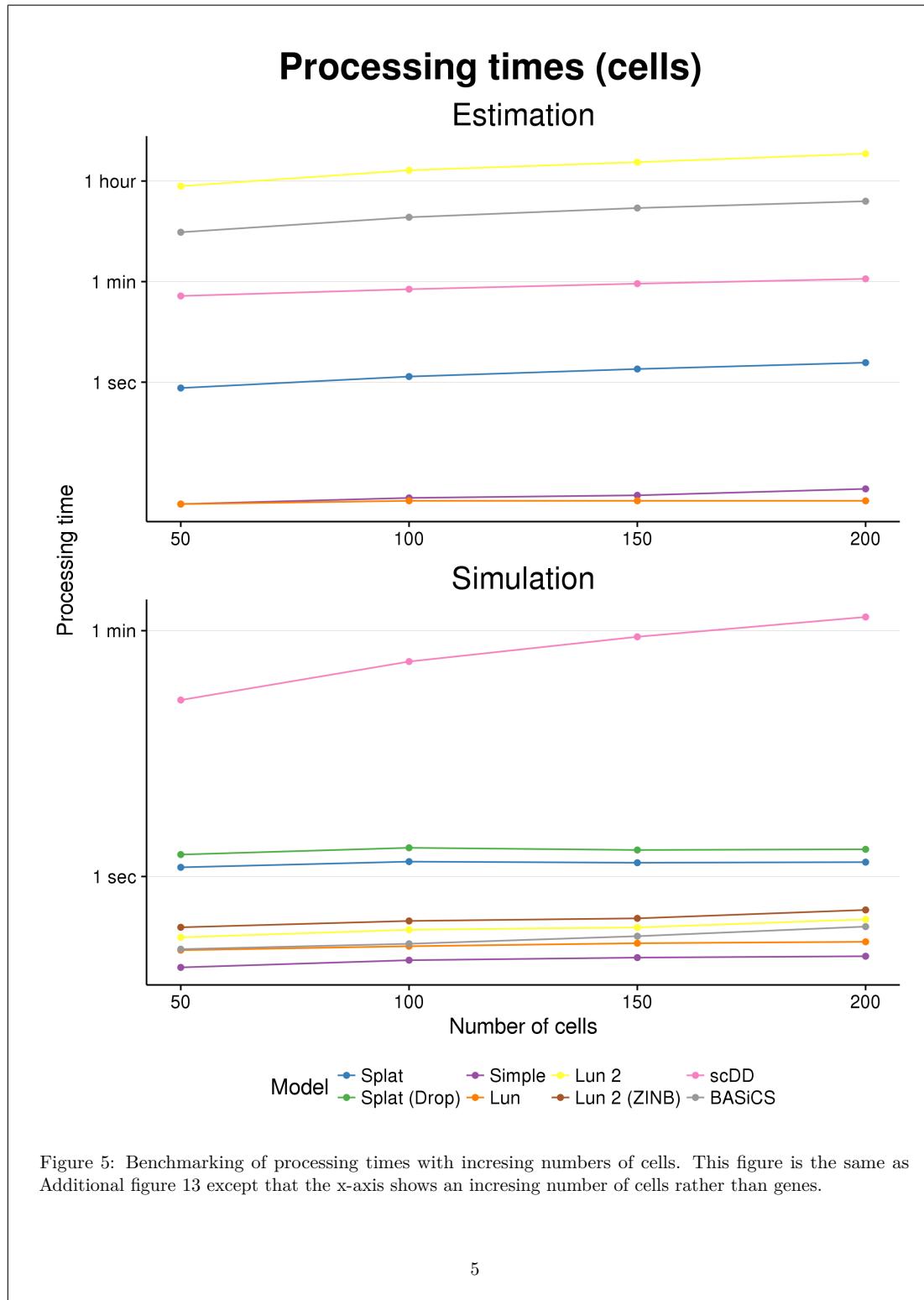
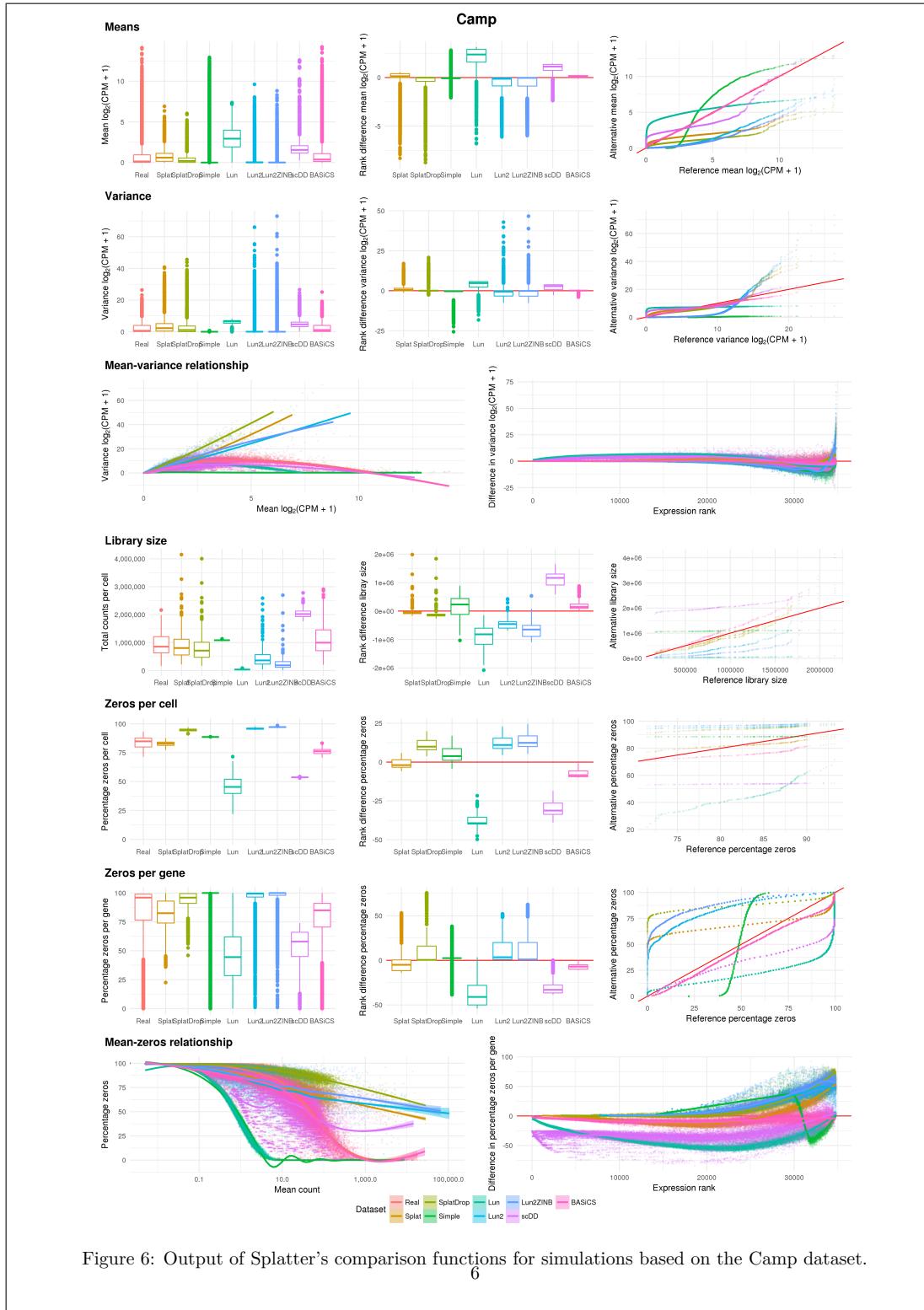


Figure 3: Diagram of the Lun 2 simulation model. Input parameters are indicated with double borders, green shading shows those that can be estimated and sampled from real data. Red shading indicates the final output.







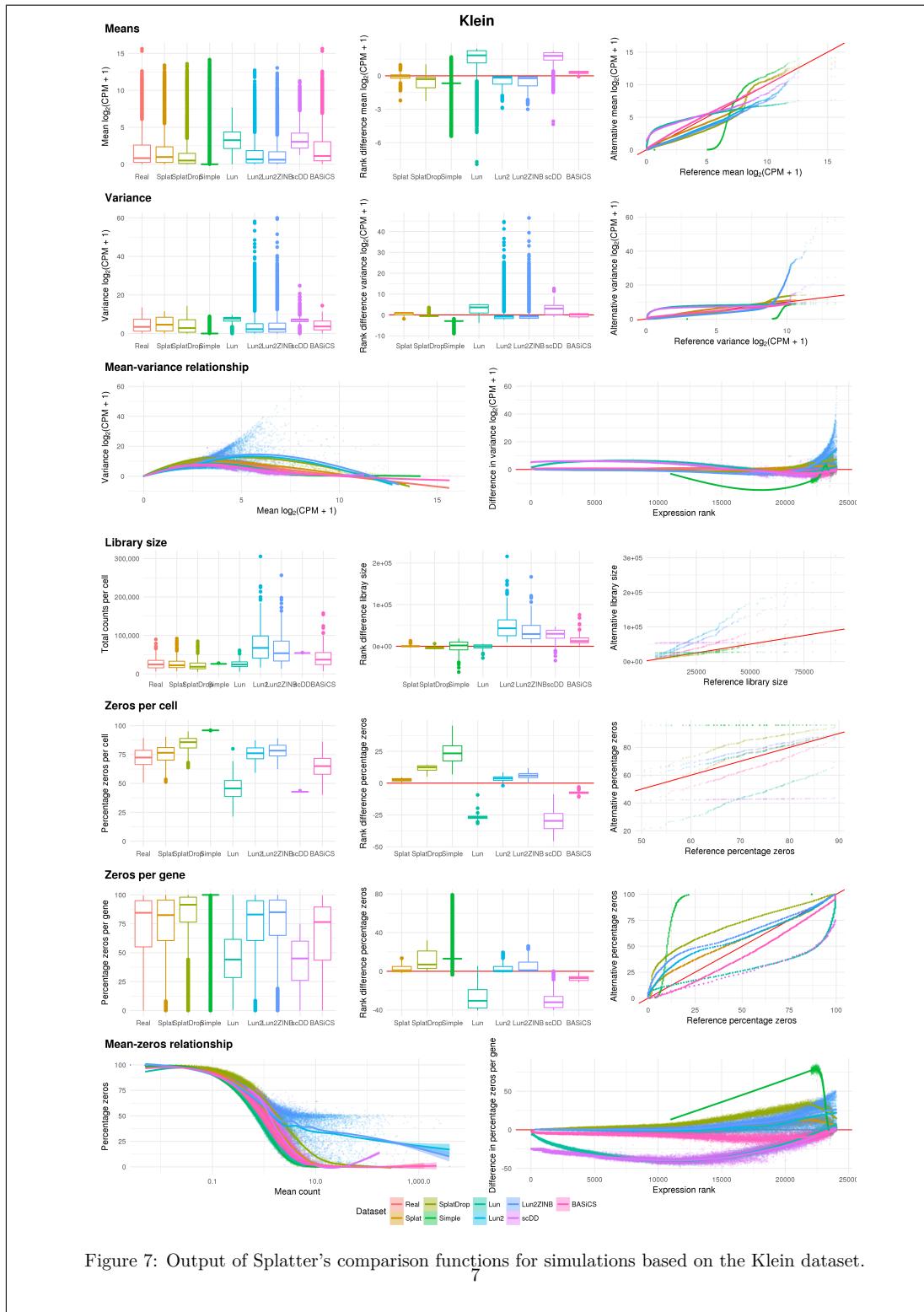


Figure 7: Output of Splatter's comparison functions for simulations based on the Klein dataset.

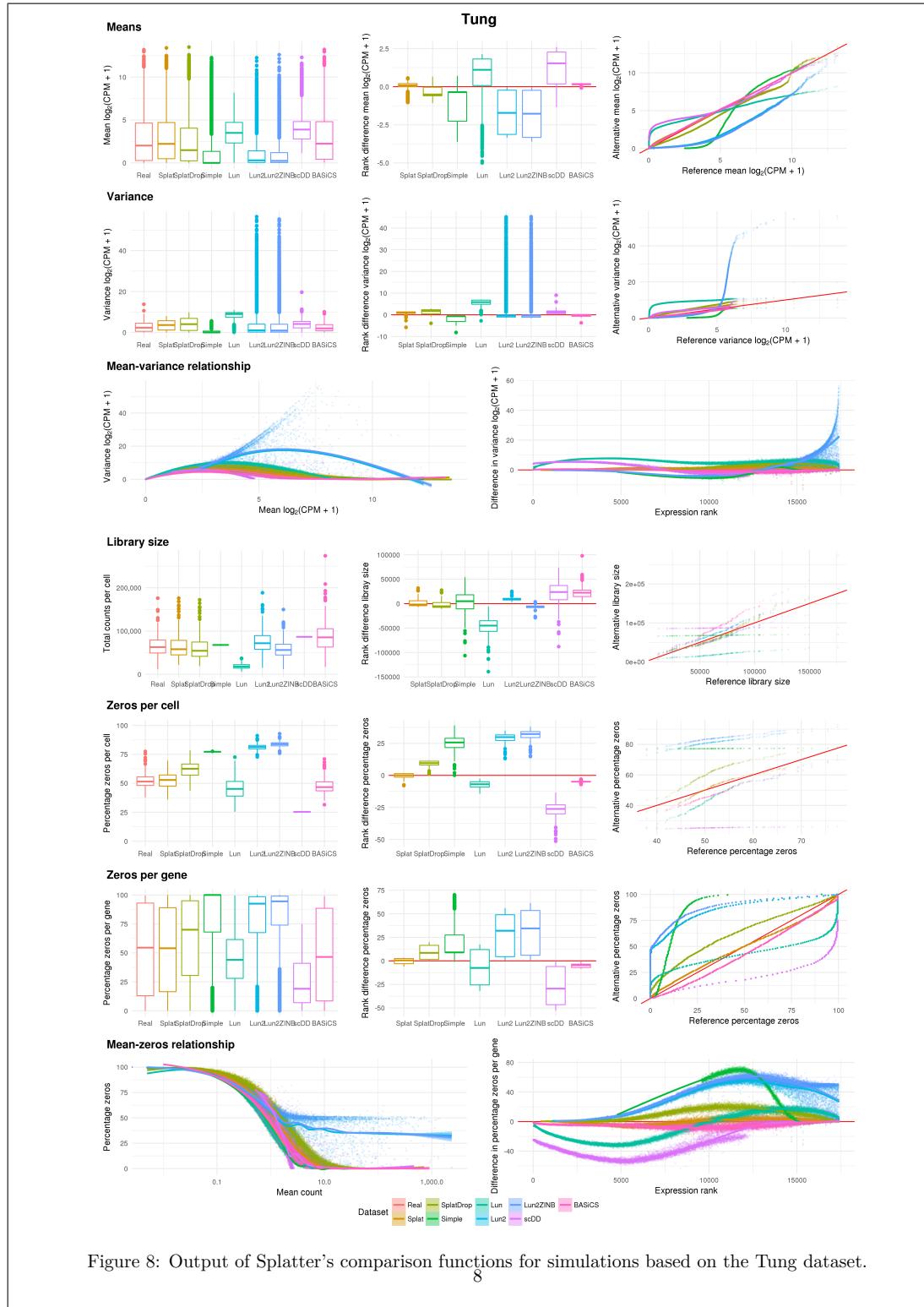


Figure 8: Output of Splatter's comparison functions for simulations based on the Tung dataset.

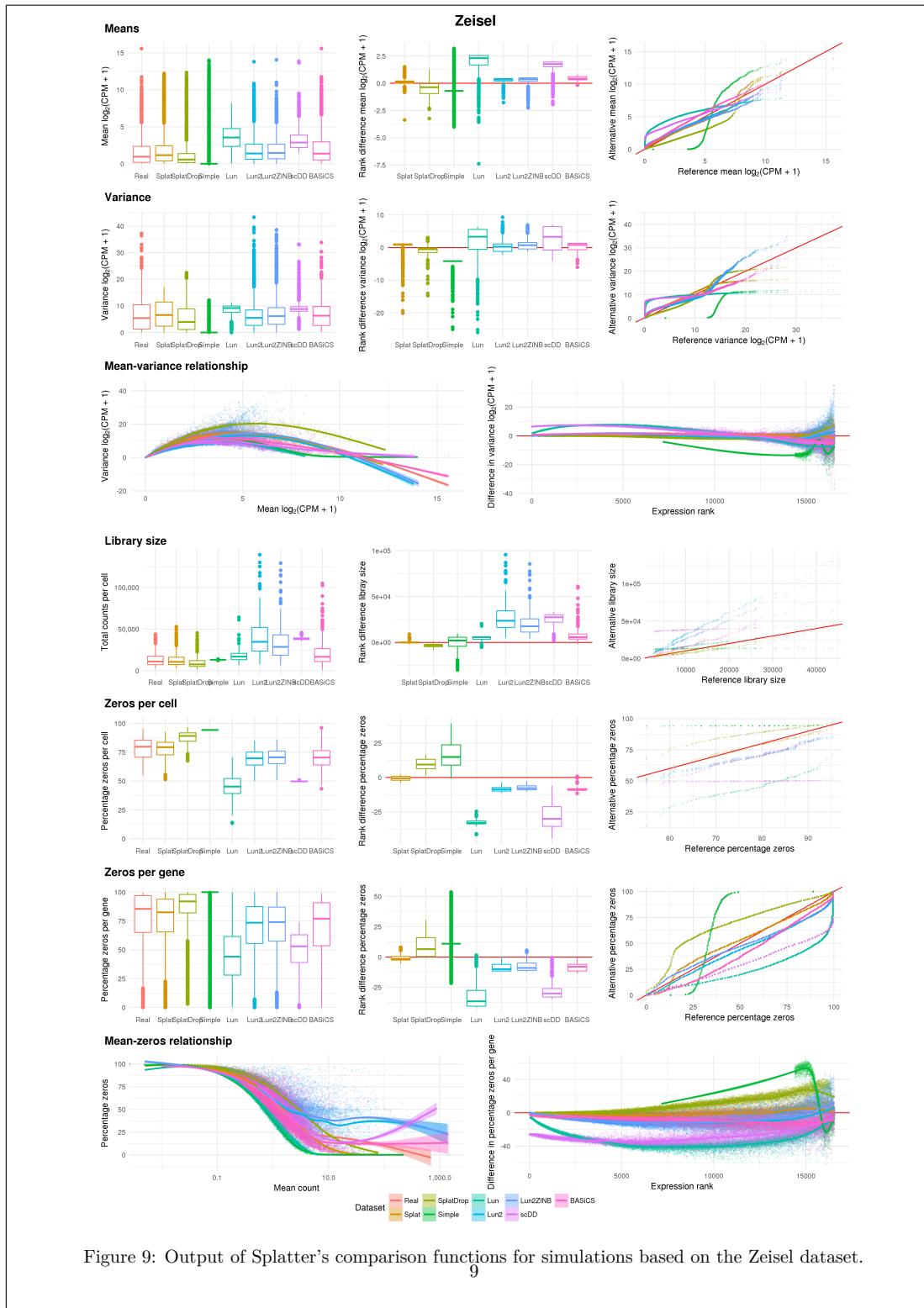
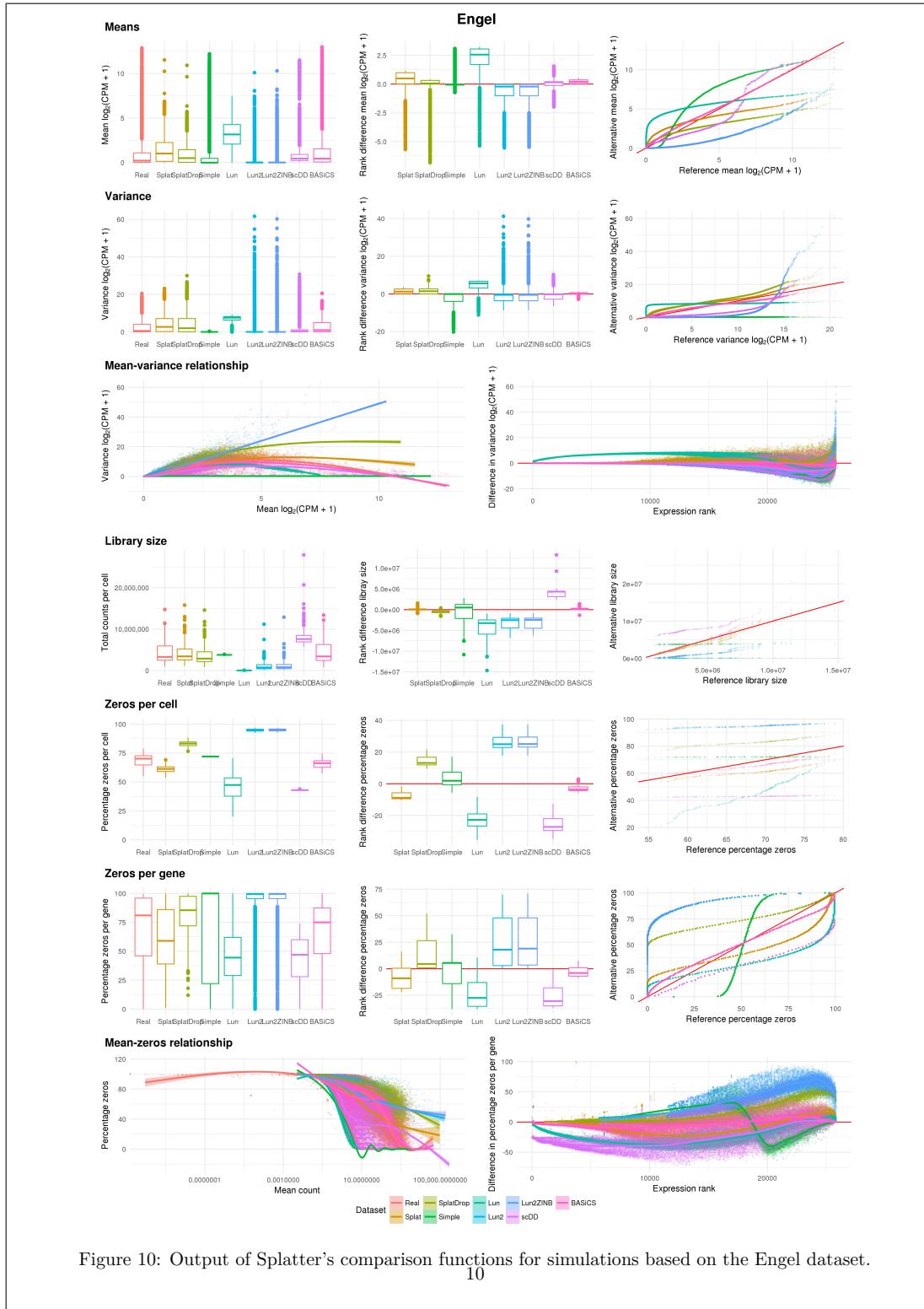
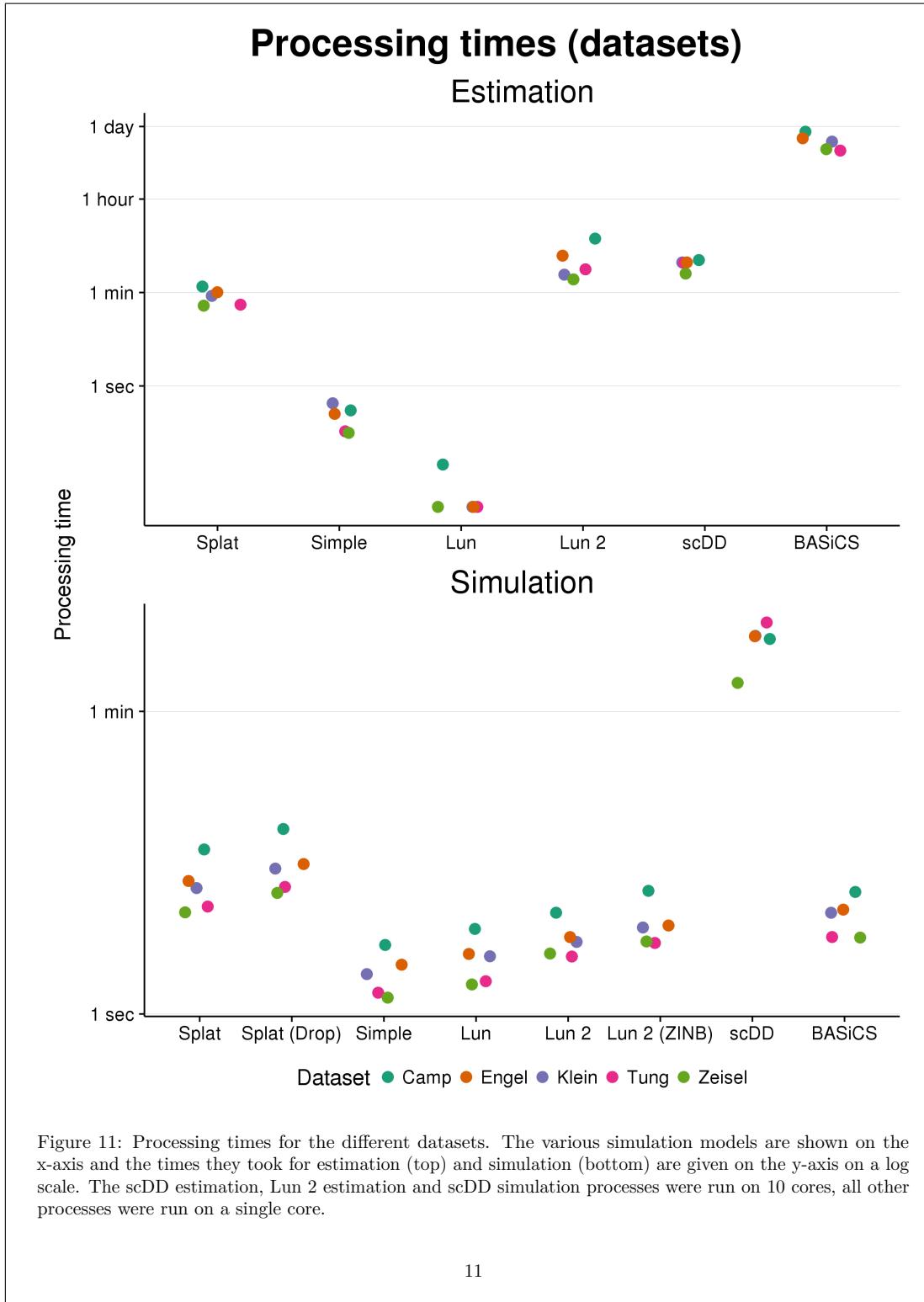
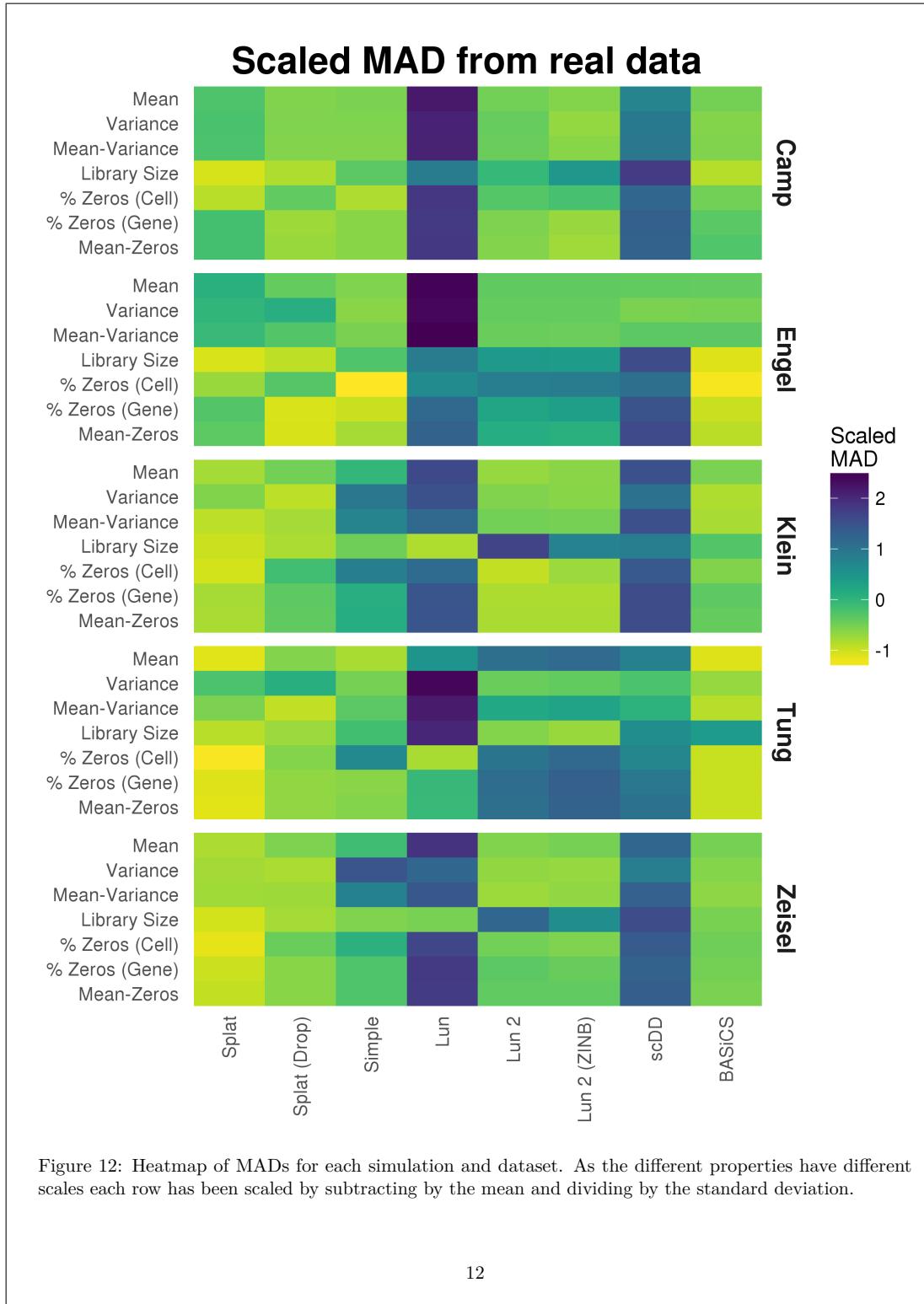


Figure 9: Output of Splatter's comparison functions for simulations based on the Zeisel dataset.







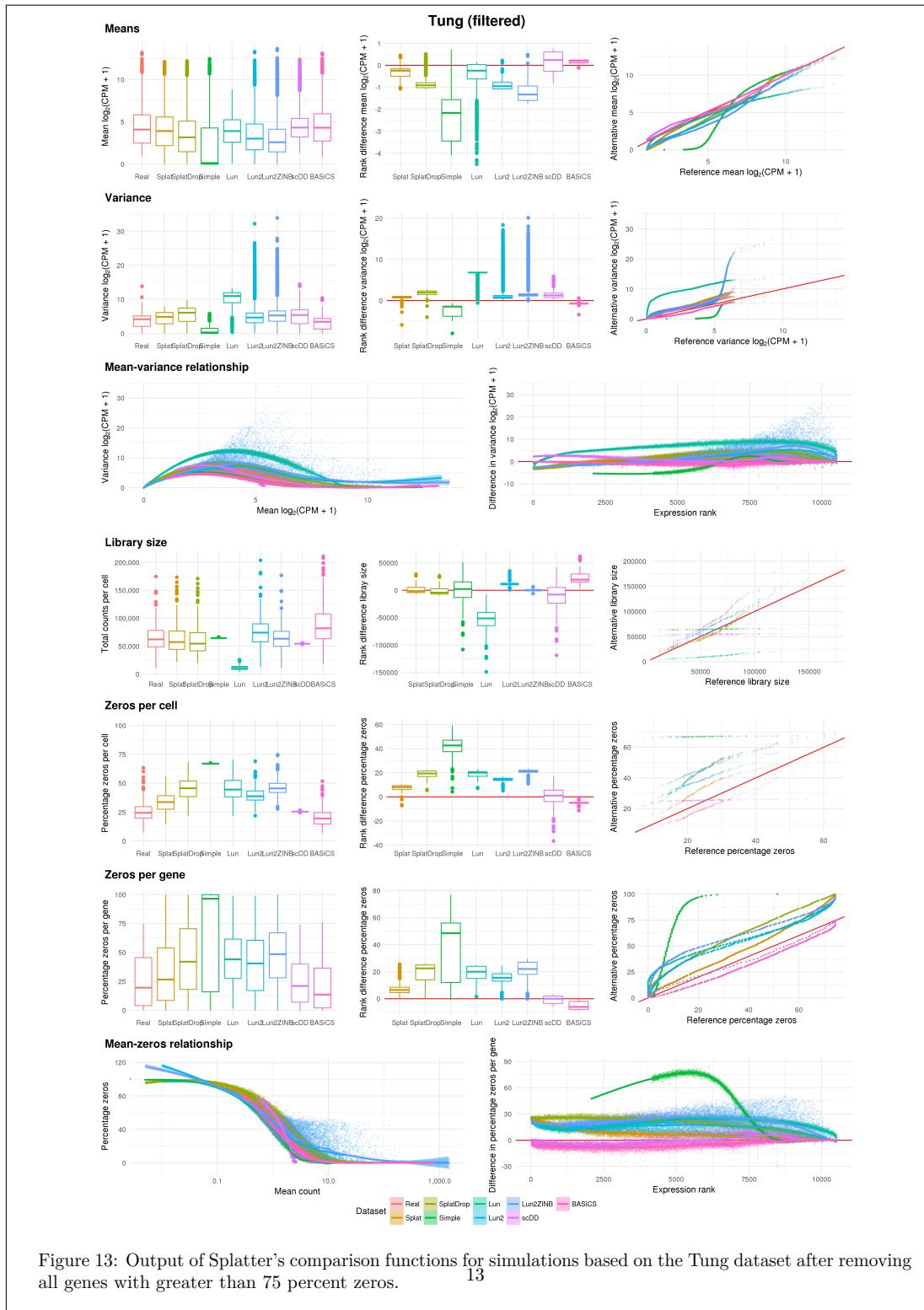


Figure 13: Output of Splatter's comparison functions for simulations based on the Tung dataset after removing all genes with greater than 75 percent zeros. 13

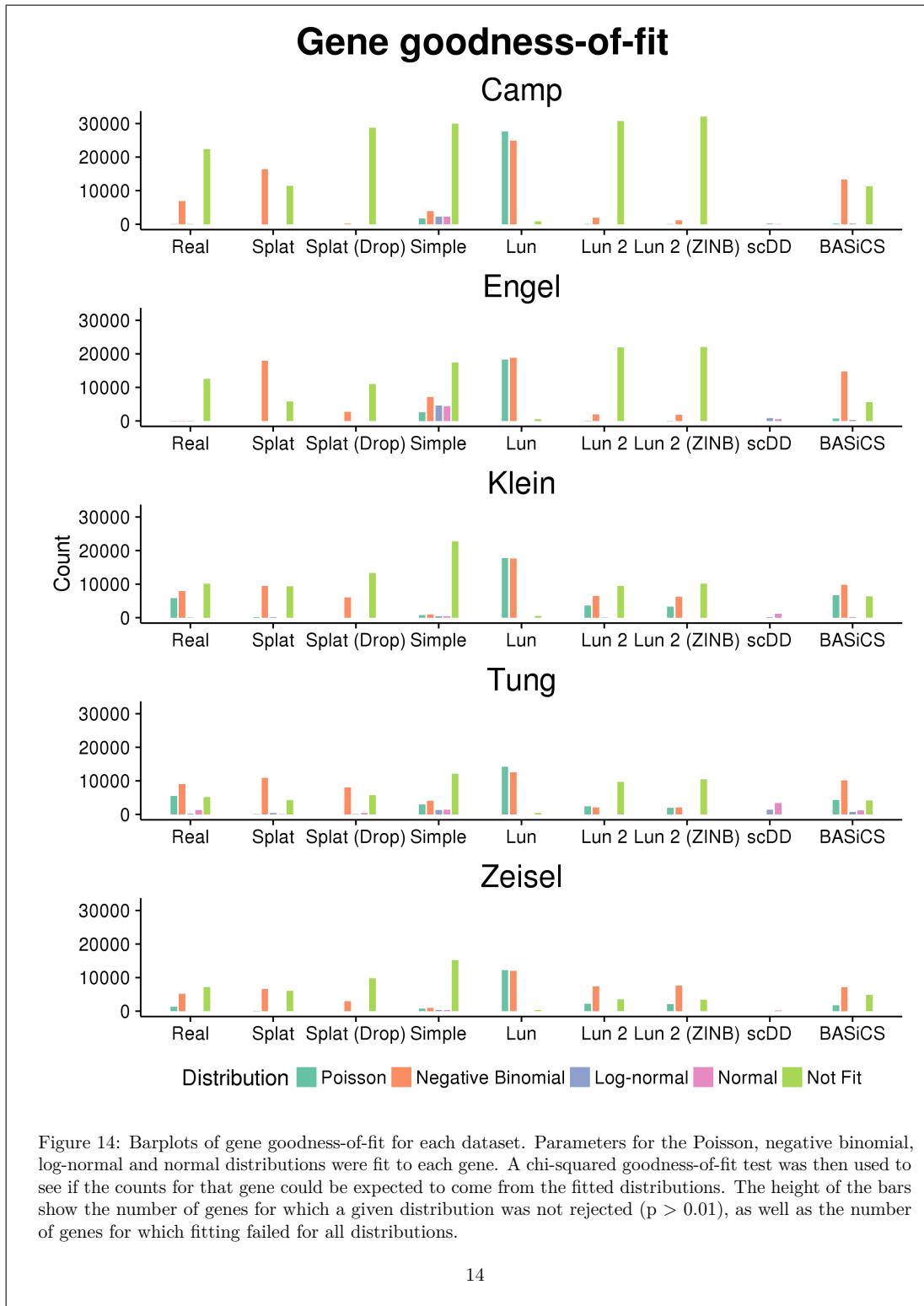
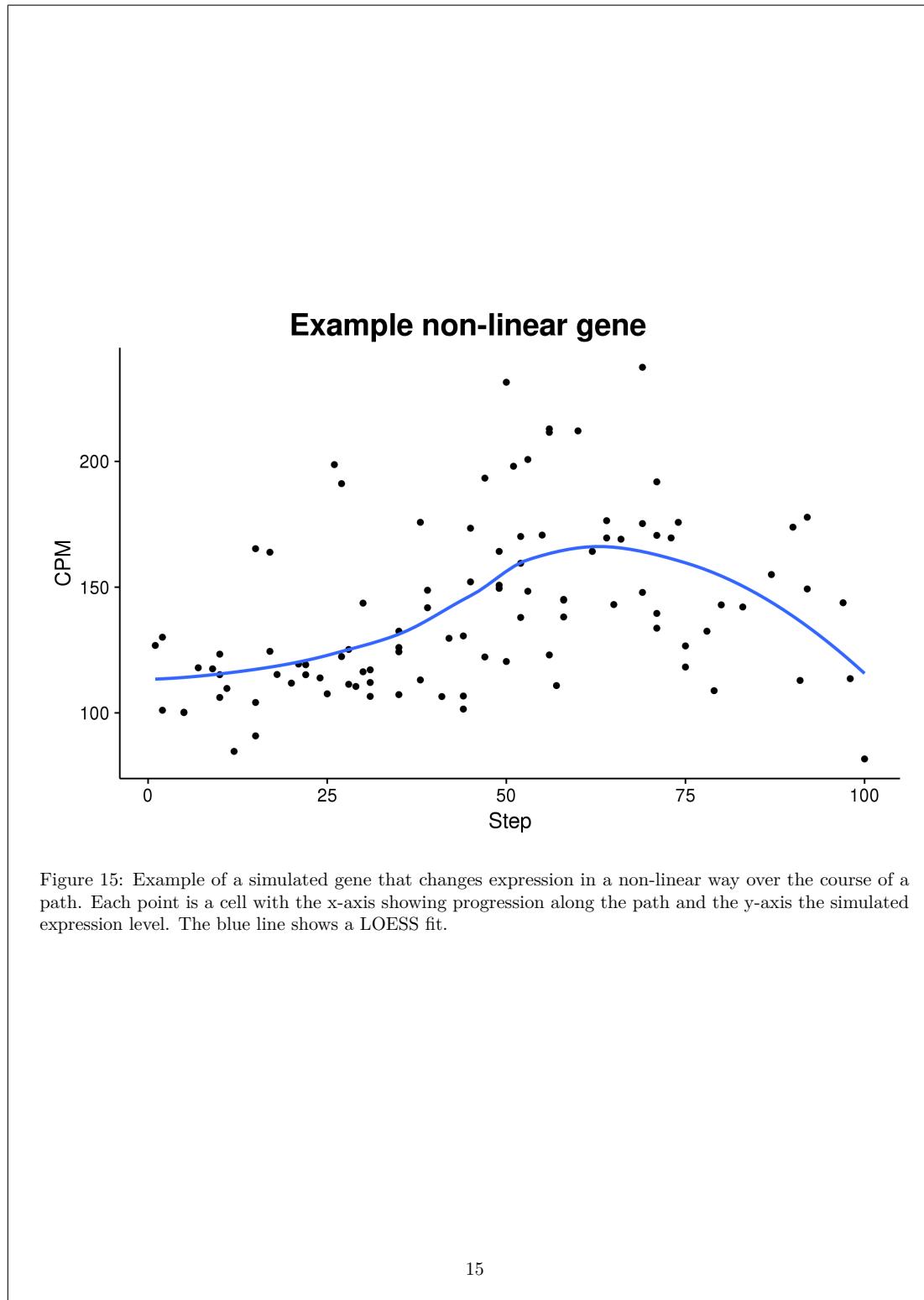


Figure 14: Barplots of gene goodness-of-fit for each dataset. Parameters for the Poisson, negative binomial, log-normal and normal distributions were fit to each gene. A chi-squared goodness-of-fit test was then used to see if the counts for that gene could be expected to come from the fitted distributions. The height of the bars show the number of genes for which a given distribution was not rejected ($p > 0.01$), as well as the number of genes for which fitting failed for all distributions.



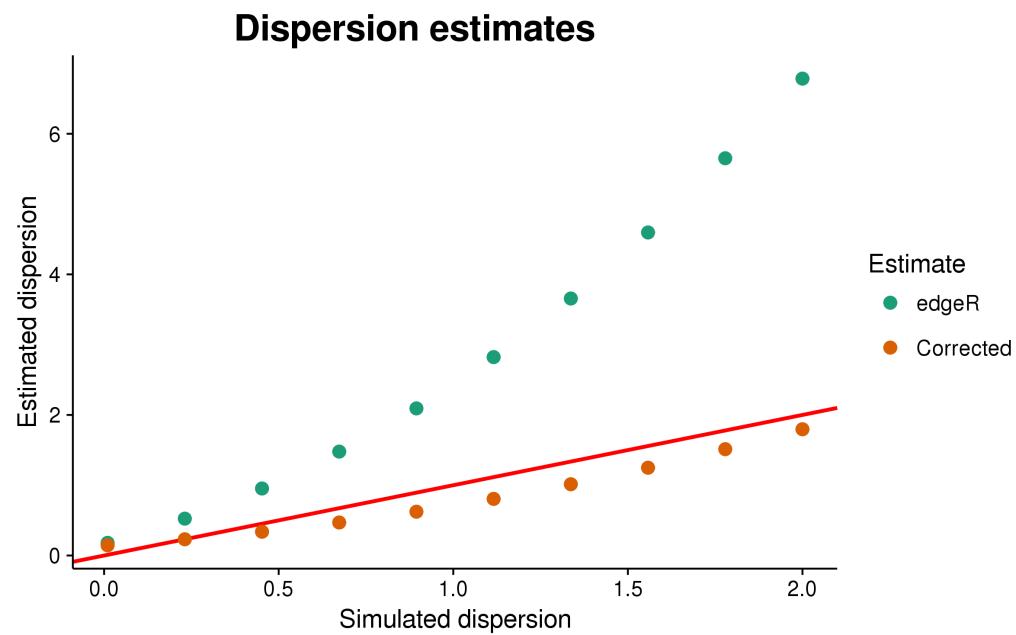


Figure 16: Correction of edgeR dispersion estimates. Scatter plot of estimated dispersions against the true simulated values. Estimates of common dispersion obtained from edgeR (green) can be inflated for single-cell data. The Splat simulation uses a linearly corrected value (orange) in its estimation procedure. The red line shows the true values.

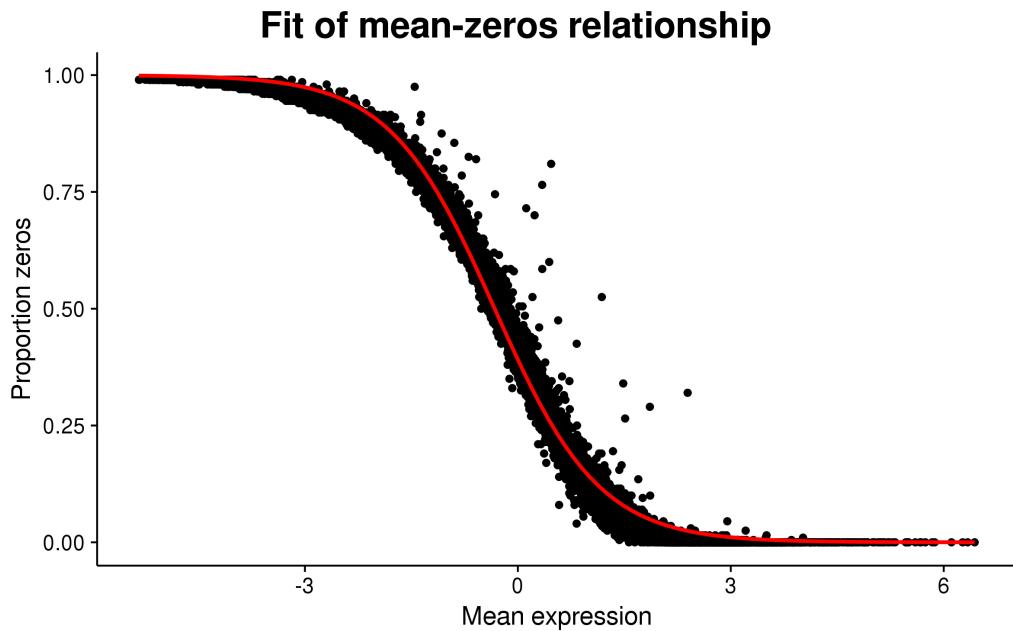


Figure 17: Fitting the mean-zeros relationship. Points show genes in the Tung dataset and the red line is a logistic function fitted to the data using Splat's estimation procedure.

A.2 Session information

R Session Information

2017-07-20

Code version: 905b291f81259313ee06d326524c31541d622055

Session information

Details of the R environment and packages used for analysis.

```
## setting value
## version R version 3.4.0 (2017-04-21)
## system x86_64, linux-gnu
## ui X11
## language (EN)
## collate en_US.UTF-8
## tz <NA>
## date 2017-07-20
##
## package * version date
## AnnotationDbi 1.38.1 2017-06-05
## assertthat 0.2.0 2017-04-11
## backports 1.1.0 2017-05-22
## base * 3.4.0 2017-05-30
## beeswarm 0.2.3 2016-04-25
## bindr 0.1 2016-11-13
## bindrcpp * 0.2 2017-06-17
## Biobase * 2.36.2 2017-05-31
## BiocGenerics * 0.22.0 2017-05-31
## BiocParallel * 1.10.1 2017-05-31
## biomaRt 2.32.1 2017-06-13
## bit 1.1-12 2014-04-09
## bit64 0.9-7 2017-05-08
## bitops 1.0-6 2013-08-17
## blob 1.1.0 2017-06-17
## broom 0.4.2 2017-02-13
## caTools 1.17.1 2014-09-10
## cellranger 1.1.0 2016-07-27
## checkmate 1.8.3 2017-07-03
## class 7.3-14 2015-08-30
## clues * 0.5.9 2016-10-14
## cluster 2.0.6 2017-03-10
## codetools 0.2-15 2016-10-05
## colorspace 1.3-2 2016-12-14
## compiler 3.4.0 2017-05-30
## cowplot * 0.7.0 2016-10-28
## data.table 1.10.4 2017-02-01
## datasets * 3.4.0 2017-05-30
## DBI 0.7 2017-06-18
## DelayedArray 0.2.7 2017-06-05
## DEoptimR 1.0-8 2016-11-19
## devtools * 1.13.2 2017-06-02
```

```
## digest          0.6.12  2017-01-27
## doParallel     1.0.10  2015-10-14
## doRNG          1.6.6   2017-04-10
## dplyr          * 0.7.1   2017-06-22
## e1071          1.6-8   2017-02-02
## edgeR          3.18.1  2017-06-14
## evaluate        0.10    2016-10-11
## forcats        0.2.0   2017-01-23
## foreach         1.4.3   2015-10-13
## foreign         0.8-68  2017-04-24
## gdata           2.18.0  2017-06-06
## GenomeInfoDb    1.12.2  2017-06-13
## GenomeInfoDbData 0.99.0  2017-05-31
## GenomicRanges   1.28.3  2017-05-31
## ggbeeswarm       0.5.3   2016-12-01
## ggplot2         * 2.2.1   2016-12-30
## glue             1.1.1   2017-06-21
## gplots           3.0.1   2016-03-30
## graphics         * 3.4.0   2017-05-30
## grDevices        * 3.4.0   2017-05-30
## grid              3.4.0   2017-05-30
## gridExtra        2.2.1   2016-02-29
## gtable            0.2.0   2016-02-26
## gtools            3.5.0   2015-05-29
## haven             1.0.0   2016-09-23
## hms               0.3     2016-11-22
## htmltools         0.3.6   2017-04-28
## httpuv            1.3.3   2015-08-04
## httr              1.2.1   2016-07-03
## IRanges           2.10.2  2017-05-31
## iterators         1.0.8   2015-10-13
## jsonlite          1.5     2017-06-01
## KernSmooth        2.23-15 2015-06-29
## knitr             * 1.16   2017-05-18
## lattice           0.20-35 2017-03-25
## lazyeval          0.2.0   2016-06-12
## limma              3.32.2  2017-06-14
## locfit            1.5-9.1 2013-04-20
## lubridate          1.6.0   2016-09-13
## magrittr          * 1.5     2014-11-22
## Matrix             1.2-10  2017-04-28
## matrixStats        0.52.2  2017-04-14
## memoise            1.1.0   2017-04-21
## methods           * 3.4.0   2017-05-30
## mime               0.5     2016-07-07
## mnormt            1.5-5   2016-10-15
## modelr             0.1.0   2016-08-31
## munsell            0.4.3   2016-02-13
## mvtnorm            1.0-6   2017-03-02
## nlme              3.1-131 2017-02-06
## parallel          * 3.4.0   2017-05-30
## pcaPP              1.9-72  2017-06-27
## pheatmap           1.0.8   2015-12-11
## pkgconfig          2.0.1   2017-03-21
```

```
##  pkgmaker          0.22   2014-05-14
##  plyr              1.8.4   2016-06-08
##  psych             1.7.5   2017-05-03
##  purrr            * 0.2.2.2 2017-05-11
##  R6                2.2.2   2017-06-17
##  RColorBrewer      1.1-2   2014-12-07
##  Rcpp              0.12.12 2017-07-15
##  RCurl              1.95-4.8 2016-03-01
##  readr              * 1.1.1   2017-05-16
##  readxl             1.0.0   2017-04-18
##  registry           0.3     2015-07-08
##  reshape2            1.4.2   2016-10-22
##  rhdf5              2.20.0   2017-05-31
##  rjson              0.2.15   2014-11-03
##  rlang              0.1.1   2017-05-18
##  rmarkdown            * 1.6     2017-06-15
##  rngtools            1.2.4   2014-03-06
##  robustbase          0.92-7   2016-12-09
##  ROCR               1.0-7   2015-03-26
##  rprojroot            1.2     2017-01-16
##  rrcov              1.4-3   2016-09-06
##  RSQLite             2.0     2017-06-19
##  rstudioapi          0.6     2016-06-27
##  rvest               0.3.2   2016-06-17
##  S4Vectors            0.14.3  2017-06-05
##  SC3                * 1.4.2   2017-07-13
##  scales              0.4.1   2016-11-09
##  scater              * 1.4.0   2017-06-14
##  shiny               1.0.3   2017-04-26
##  shinydashboard       0.6.1   2017-06-14
##  splatter             * 1.1.2   2017-07-16
##  stats               * 3.4.0   2017-05-30
##  stats4              3.4.0   2017-05-30
##  stringi              1.1.5   2017-04-07
##  stringr              1.2.0   2017-02-18
##  SummarizedExperiment 1.6.3   2017-05-31
##  tibble              * 1.3.3   2017-05-28
##  tidyr              * 0.6.3   2017-05-15
##  tidyverse            * 1.1.1   2017-01-27
##  tools               3.4.0   2017-05-30
##  tximport             1.4.0   2017-05-31
##  utils               * 3.4.0   2017-05-30
##  vipor               0.4.5   2017-03-22
##  viridis              0.4.0   2017-03-27
##  viridisLite          0.2.0   2017-03-24
##  withr               1.0.2   2016-06-20
##  WriteXLS             4.0.0   2015-12-07
##  XML                 3.98-1.9 2017-06-19
##  xml2                1.1.1   2017-01-24
##  xtable              1.8-2   2016-02-05
##  XVector              0.16.0   2017-05-31
##  yaml                2.1.14   2016-11-12
##  zlibbioc            1.22.0   2017-05-31
##  source
```

```
## Bioconductor
## CRAN (R 3.4.0)
## CRAN (R 3.4.0)
## local
## CRAN (R 3.4.0)
## CRAN (R 3.4.0)
## CRAN (R 3.4.0)
## CRAN (R 3.4.0)
## Bioconductor
## Bioconductor
## Bioconductor
## Bioconductor
## Bioconductor
## CRAN (R 3.4.0)
## local
## CRAN (R 3.4.0)
## CRAN (R 3.4.0)
## local
## CRAN (R 3.4.0)
## Bioconductor
## CRAN (R 3.4.0)
## Bioconductor
## CRAN (R 3.4.0)
## Bioconductor
## Bioconductor
## Bioconductor
## CRAN (R 3.4.0)
## CRAN (R 3.4.0)
## CRAN (R 3.4.0)
## CRAN (R 3.4.0)
## local
## local
## local
## CRAN (R 3.4.0)
```



```
## CRAN (R 3.4.0)
## CRAN (R 3.4.0)
## Bioconductor
## Bioconductor
## CRAN (R 3.4.0)
## Bioconductor
## CRAN (R 3.4.0)
## CRAN (R 3.4.0)
## Github (Oshlack/splatter@5df5a7c)
## local
## local
## CRAN (R 3.4.0)
## CRAN (R 3.4.0)
## Bioconductor
## CRAN (R 3.4.0)
## CRAN (R 3.4.0)
## CRAN (R 3.4.0)
## local
## Bioconductor
## local
## CRAN (R 3.4.0)
## Bioconductor
## CRAN (R 3.4.0)
## Bioconductor
```

B

Splatter
documentation

B.1 Splatter vignette

Introduction to Splatter

Luke Zappia

2019-05-02

Contents

- 1 Installation
 - 2 Quickstart
 - 3 The Splat simulation
 - 4 The `SplatParams` object
 - 4.1 Getting and setting
 - 5 Estimating parameters
 - 6 Simulating counts
 - 6.1 Simulating groups
 - 6.2 Simulating paths
 - 6.3 Batch effects
 - 6.4 Convenience functions
 - 7 Other simulations
 - 8 Other expression values
 - 9 Comparing simulations and real data
 - 9.1 Comparing differences
 - 9.2 Making panels
 - 10 Citing Splatter
- Session information



Splatter logo

Welcome to Splatter! Splatter is an R package for the simple simulation of single-cell RNA sequencing data. This vignette gives an overview and introduction to Splatter's functionality.

1 Installation

Splatter can be installed from Bioconductor:

```
if (!requireNamespace("BiocManager", quietly=TRUE))
  install.packages("BiocManager")
BiocManager::install("splatter")
```

To install the most recent development version from Github use:

```
BiocManager::install("oshlack/splatter", dependencies = TRUE,
  build_vignettes = TRUE)
```

2 Quickstart

Assuming you already have a matrix of count data similar to that you wish to simulate there are two simple steps to creating a simulated data set with Splatter. Here is an example using the example dataset in the scater package:

```
# Load package
library(splatter)

# Load example data
library(scater)
data("sc_example_counts")
# Estimate parameters from example data
params <- splatEstimate(sc_example_counts)
# Simulate data using estimated parameters
sim <- splatSimulate(params)

## Getting parameters...

## Creating simulation object...

## Simulating library sizes...

## Simulating gene means...

## Simulating BCV...

## Simulating counts...

## Simulating dropout (if needed)...

## Done!
```

These steps will be explained in detail in the following sections but briefly the first step takes a dataset and estimates simulation parameters from it and the second step takes those parameters and simulates a new dataset.

3 The Splat simulation

Before we look at how we estimate parameters let's first look at how Splatter simulates data and what those parameters are. We use the term 'Splat' to refer to the Splatter's own simulation and differentiate it from the package itself. The core of the Splat model is a gamma-Poisson distribution used to generate a gene by cell matrix of counts. Mean expression levels for each gene are simulated from a gamma distribution (https://en.wikipedia.org/wiki/Gamma_distribution) and the Biological Coefficient of Variation is used to enforce a mean-variance trend before counts are simulated from a Poisson distribution (https://en.wikipedia.org/wiki/Poisson_distribution). Splat also allows you to simulate expression outlier genes (genes with mean expression outside the gamma distribution) and dropout (random knock out of counts based on mean expression). Each cell is given an expected library size (simulated from a log-normal distribution) that makes it easier to match to a given dataset.

Splat can also simulate differential expression between groups of different types of cells or differentiation paths between different cells types where expression changes in a continuous way. These are described further in the simulating counts section.

4 The SplatParams object

All the parameters for the Splat simulation are stored in a `SplatParams` object. Let's create a new one and see what it looks like.

```
params <- newSplatParams()  
params
```

```

## A Params object of class SplatParams
## Parameters can be (estimable) or [not estimable], 'Default' or
## 'NOT DEFAULT'
##
## Global:
## (Genes) (cells) [Seed]
## 10000 100 138572
##
## 28 additional parameters
##
## Batches:
## [Batches] [Batch cells] [Location] [Scale]
## 1 100 0.1 0.1
##
## Mean:
## (Rate) (Shape)
## 0.3 0.6
##
## Library size:
## (Location) (scale) (Norm)
## 11 0.2 FALSE
##
## Exprs outliers:
## (Probability) (Location) (scale)
## 0.05 4 0.5
##
## Groups:
## [Groups] [Group Probs]
## 1 1
##
## Diff expr:
## [Probability] [Down Prob] [Location] [Scale]
## 0.1 0.5 0.1 0.4
##
## BCV:
## (Common Disp) (DoF)
## 0.1 60
##
## Dropout:
## [Type] (Midpoint) (Shape)
## none 0 -1
##
## Paths:
## [From] [Steps] [Skew] [Non-linea
r]
## 0 100 0.5 0.
1
## [Sigma Factor]
## 0.8

```

As well as telling us what type of object we have (“A `Params` object of class `SplatParams`”) and showing us the values of the parameter this output gives us some extra information. We can see which parameters can be estimated by the `splatEstimate` function (those in parentheses), which can’t be estimated (those in brackets) and which have been changed from their default values (those in ALL CAPS). For more details about the parameters of the Splat simulation refer to the Splat parameters vignette (`splat_params.html`).

4.1 Getting and setting

If we want to look at a particular parameter, for example the number of genes to simulate, we can extract it using the `getParam` function:

```
getParam(params, "nGenes")
```

```
## [1] 10000
```

Alternatively, to give a parameter a new value we can use the `setParam` function:

```
params <- setParam(params, "nGenes", 5000)
getParam(params, "nGenes")
```

```
## [1] 5000
```

If we want to extract multiple parameters (as a list) or set multiple parameters we can use the `getParams` or `setParams` functions:

```
# Set multiple parameters at once (using a list)
params <- setParams(params, update = list(nGenes = 8000, mean.ra
te = 0.5))
# Extract multiple parameters as a list
getParams(params, c("nGenes", "mean.rate", "mean.shape"))

## $nGenes
## [1] 8000
##
## $mean.rate
## [1] 0.5
##
## $mean.shape
## [1] 0.6

# Set multiple parameters at once (using additional arguments)
params <- setParams(params, mean.shape = 0.5, de.prob = 0.2)
params
```

```

## A Params object of class SplatParams
## Parameters can be (estimable) or [not estimable], 'Default' or
## 'NOT DEFAULT'
##
## Global:
## (GENES) (cells) [Seed]
## 8000 100 138572
##
## 28 additional parameters
##
## Batches:
## [Batches] [Batch cells] [Location] [Scale]
## 1 100 0.1 0.1
##
## Mean:
## (RATE) (SHAPE)
## 0.5 0.5
##
## Library size:
## (Location) (scale) (Norm)
## 11 0.2 FALSE
##
## Exprs outliers:
## (Probability) (Location) (scale)
## 0.05 4 0.5
##
## Groups:
## [Groups] [Group Probs]
## 1 1
##
## Diff expr:
## [PROBABILITY] [Down Prob] [Location] [Scale]
## 0.2 0.5 0.1 0.4
##
## BCV:
## (Common Disp) (DoF)
## 0.1 60
##
## Dropout:
## [Type] (Midpoint) (shape)
## none 0 -1
##
## Paths:
## [From] [Steps] [Skew] [Non-linea
r]
## 0 100 0.5 0.
1
## [Sigma Factor]
## 0.8

```

The parameters with have changed are now shown in ALL CAPS to indicate that they been changed form the default.

We can also set parameters directly when we call `newSplatParams` :

```

params <- newSplatParams(lib.loc = 12, lib.scale = 0.6)
getParams(params, c("lib.loc", "lib.scale"))

```

```
## $lib.loc
## [1] 12
##
## $lib.scale
## [1] 0.6
```

5 Estimating parameters

Splat allows you to estimate many of its parameters from a data set containing counts using the `splatEstimate` function.

```
# Check that sc_example_counts is an integer matrix
class(sc_example_counts)

## [1] "matrix"

typeof(sc_example_counts)

## [1] "integer"

# Check the dimensions, each row is a gene, each column is a cell
7
dim(sc_example_counts)

## [1] 2000    40

# Show the first few entries
sc_example_counts[1:5, 1:5]

##           cell_001 cell_002 cell_003 cell_004 cell_005
## Gene_0001      0     123      2      0      0
## Gene_0002    575      65      3    1561    2311
## Gene_0003      0      0      0      0   1213
## Gene_0004      0      1      0      0      0
## Gene_0005      0      0     11      0      0

params <- splatEstimate(sc_example_counts)
```

Here we estimated parameters from a counts matrix but `splatEstimate` can also take a `singleCellExperiment` object. The estimation process has the following steps:

1. Mean parameters are estimated by fitting a gamma distribution to the mean expression levels.
2. Library size parameters are estimated by fitting a log-normal distribution to the library sizes.
3. Expression outlier parameters are estimated by determining the number of outliers and fitting a log-normal distribution to their difference from the median.
4. BCV parameters are estimated using the `estimatedDisp` function from the `edgeR` package.

5. Dropout parameters are estimated by checking if dropout is present and fitting a logistic function to the relationship between mean expression and proportion of zeros.

For more details of the estimation procedures see `?splatEstimate`.

6 Simulating counts

Once we have a set of parameters we are happy with we can use `splatSimulate` to simulate counts. If we want to make small adjustments to the parameters we can provide them as additional arguments, alternatively if we don't supply any parameters the defaults will be used:

```
sim <- splatSimulate(params, nGenes = 1000)

## Getting parameters...

## Creating simulation object...

## Simulating library sizes...

## Simulating gene means...

## Simulating BCV...

## Simulating counts...

## Simulating dropout (if needed)...

## Done!

sim

## class: SingleCellExperiment
## dim: 1000 40
## metadata(1): Params
## assays(6): BatchCellMeans BaseCellMeans ... TrueCounts counts
## rownames(1000): Gene1 Gene2 ... Gene999 Gene1000
## rowData names(4): Gene BaseGeneMean OutlierFactor GeneMean
## colnames(40): cell1 cell2 ... cell39 cell40
## colData names(3): Cell Batch Explibsize
## reducedDimNames(0):
## spikeNames(0):
```

Looking at the output of `splatSimulate` we can see that `sim` is `SingleCellExperiment` object with 1000 features (genes) and 40 samples (cells). The main part of this object is a features by samples matrix containing the simulated counts (accessed using `counts`), although it can also hold other expression measures such as FPKM or TPM. Additionally a `SingleCellExperiment` contains phenotype information about each cell

(accessed using `colData`) and feature information about each gene (accessed using `rowData`). Splatter uses these slots, as well as `assays`, to store information about the intermediate values of the simulation.

```
# Access the counts
counts(sim)[1:5, 1:5]

##      Cell1 Cell2 Cell3 Cell4 Cell5
## Gene1    11     0     0     0 4814
## Gene2     0     0     0     0     0
## Gene3 1848   210     6     1 1096
## Gene4     0     0     0     0     0
## Gene5     0     0    27    29    80

# Information about genes
head(rowData(sim))

## DataFrame with 6 rows and 4 columns
##           Gene      BaseGeneMean OutlierFactor      G
## GeneMean
##             <factor>      <numeric>      <numeric>      <n
## numeric>
## Gene1    Gene1  162.27961350832          1 162.2796
## Gene2    Gene2  0.00394586265413722          1 0.0039458626
## Gene3    Gene3  27.0488594070323          1 27.048859
## Gene4    Gene4  0.00893408360914568          1 0.0089340836
## Gene5    Gene5  43.218226825995          1 43.21822
## Gene6    Gene6  0.436957397662496          1 0.43695739
## Gene7    Gene7  0.436957397662496          1 0.43695739
## Gene8    Gene8  0.436957397662496          1 0.43695739
## Gene9    Gene9  0.436957397662496          1 0.43695739
## Gene10   Gene10 0.436957397662496          1 0.43695739
## Gene11   Gene11 0.436957397662496          1 0.43695739
## Gene12   Gene12 0.436957397662496          1 0.43695739
## Gene13   Gene13 0.436957397662496          1 0.43695739
## Gene14   Gene14 0.436957397662496          1 0.43695739
## Gene15   Gene15 0.436957397662496          1 0.43695739
## Gene16   Gene16 0.436957397662496          1 0.43695739

# Information about cells
head(colData(sim))

## DataFrame with 6 rows and 3 columns
##           Cell      Batch      ExpLibsize
##             <factor> <character>      <numeric>
## Cell1    Cell1    Batch1  633156.33789341
## Cell2    Cell2    Batch1  543355.825769202
## Cell3    Cell3    Batch1 361636.014794991
## Cell4    Cell4    Batch1 273146.845309571
## Cell5    Cell5    Batch1 762131.522140643
## Cell6    Cell6    Batch1 1111265.22540117

# Gene by cell matrices
names(assays(sim))

## [1] "BatchCellMeans" "BaseCellMeans"  "BCV"          "CellM
## eans"
## [5] "TrueCounts"     "counts"
```

```
# Example of cell means matrix
assays(sim)$cellMeans[1:5, 1:5]

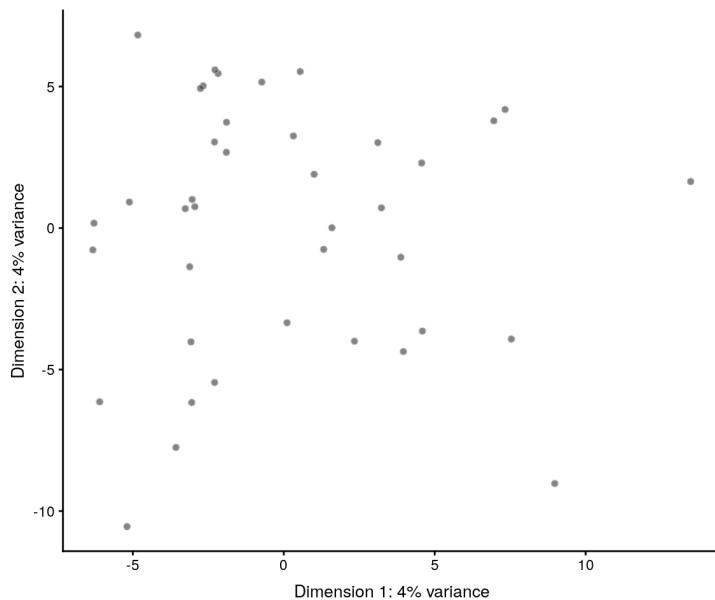
##          cell11      cell12      cell13      cell14
cell15
## Gene1 1.656039e+01 9.782325e-05 7.783061e-19 1.550924e-15 4.
785735e+03
## Gene2 4.884689e-23 1.469951e-58 3.569772e-10 1.902942e-58 6.
634616e-32
## Gene3 1.850411e+03 2.161880e+02 6.841197e+00 1.690751e+00 1.
095851e+03
## Gene4 2.167214e-43 1.058928e-27 3.464428e-109 1.379729e-56 1.
953030e-03
## Gene5 5.842502e-03 1.104395e-03 2.863923e+01 3.535881e+01 7.
466149e+01
```

An additional (big) advantage of outputting a `singleCellExperiment` is that we get immediate access to other analysis packages, such as the plotting functions in `scater`. For example we can make a PCA plot:

```
# Use scater to calculate logcounts
sim <- normalize(sim)

## Warning in .local(object, ...): using library sizes as size factors

# Plot PCA
plotPCA(sim)
```



(**NOTE:** Your values and plots may look different as the simulation is random and produces different results each time it is run.)

For more details about the `singleCellExperiment` object refer to the
[\[vignette\]](#) [SCE-vignette](#)
[\(<https://bioconductor.org/packages-devel/bioc/vignettes/SingleCellExperiment/inst/doc/intro.html>\)](https://bioconductor.org/packages-devel/bioc/vignettes/SingleCellExperiment/inst/doc/intro.html)

For information about what you can do with scater refer to the scater documentation and vignette (<https://bioconductor.org/packages/release/bioc/vignettes/scater/inst/doc/vignette.html>).

The splatsimulate function outputs the following additional information about the simulation:

- **Cell information (colData)**
 - cell - Unique cell identifier.
 - Group - The group or path the cell belongs to.
 - ExpLibsize - The expected library size for that cell.
 - Step (paths only) - How far along the path each cell is.
- **Gene information (rowData)**
 - Gene - Unique gene identifier.
 - BaseGeneMean - The base expression level for that gene.
 - outlierFactor - Expression outlier factor for that gene (1 is not an outlier).
 - GeneMean - Expression level after applying outlier factors.
 - DEFac[Group] - The differential expression factor for each gene in a particular group (1 is not differentially expressed).
 - GeneMean[Group] - Expression level of a gene in a particular group after applying differential expression factors.
- **Gene by cell information (assays)**
 - BaseCellMeans - The expression of genes in each cell adjusted for expected library size.
 - BCV - The Biological Coefficient of Variation for each gene in each cell.
 - cellMeans - The expression level of genes in each cell adjusted for BCV.
 - TrueCounts - The simulated counts before dropout.
 - Dropout - Logical matrix showing which counts have been dropped in which cells.

Values that have been added by Splatter are named using `UpperCamelCase` to separate them from the `underscore_naming` used by scater and other packages. For more information on the simulation see `?splatsimulate`.

6.1 Simulating groups

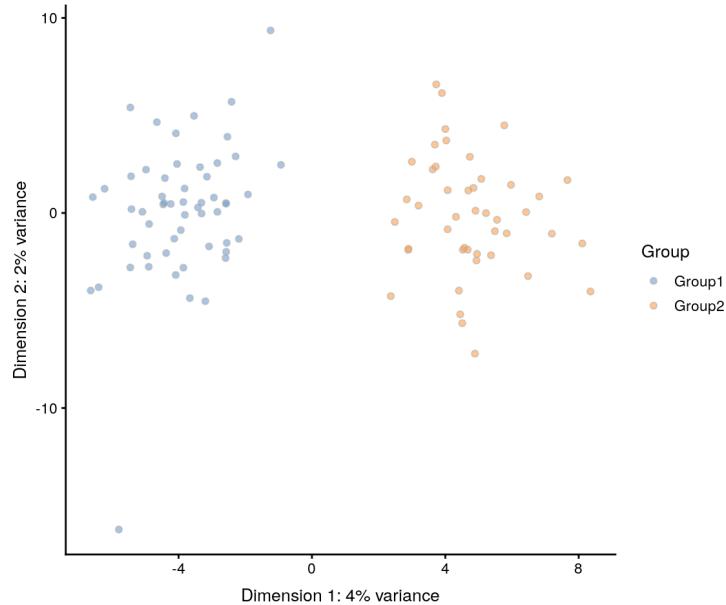
So far we have only simulated a single population of cells but often we are interested in investigating a mixed population of cells and looking to see what cell types are present or what differences there are between them. Splatter is able to simulate these situations by changing the `method` argument. Here we are going to simulate two groups, by specifying the `group.prob` parameter and setting the `method` parameter to "groups":

(**NOTE:** We have also set the `verbose` argument to `FALSE` to stop Splatter printing progress messages.)

```
sim.groups <- splatsimulate(group.prob = c(0.5, 0.5), method =
"groups",
                               verbose = FALSE)
sim.groups <- normalize(sim.groups)
```

```
## Warning in .local(object, ...): using library sizes as size for
actors
```

```
plotPCA(sim.groups, colour_by = "Group")
```



As we have set both the group probabilities to 0.5 we should get approximately equal numbers of cells in each group (around 50 in this case). If we wanted uneven groups we could set `group.prob` to any set of probabilities that sum to 1.

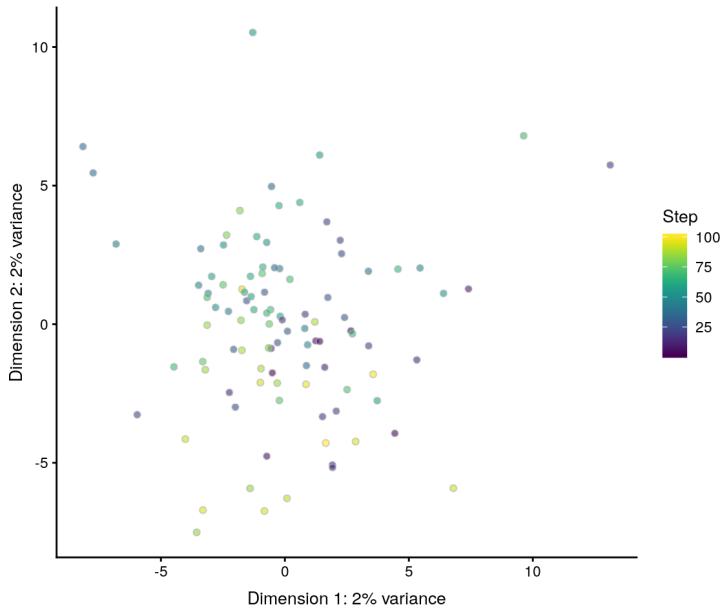
6.2 Simulating paths

The other situation that is often of interest is a differentiation process where one cell type is changing into another. Splatter approximates this process by simulating a series of steps between two groups and randomly assigning each cell to a step. We can create this kind of simulation using the "paths" method.

```
sim.paths <- splatsimulate(method = "paths", verbose = FALSE)
sim.paths <- normalize(sim.paths)
```

```
## Warning in .local(object, ...): using library sizes as size for
actors
```

```
plotPCA(sim.paths, colour_by = "Step")
```



Here the colours represent the “step” of each cell or how far along the differentiation path it is. We can see that the cells with dark colours are more similar to the originating cell type and the light coloured cells are closer to the final, differentiated, cell type. By setting additional parameters it is possible to simulate more complex process (for example multiple mature cell types from a single progenitor).

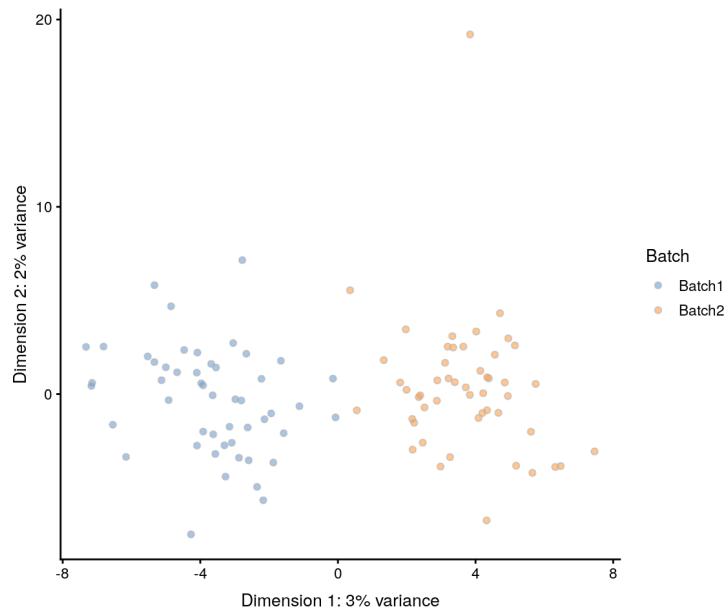
6.3 Batch effects

Another factor that is important in the analysis of any sequencing experiment are batch effects, technical variation that is common to a set of samples processed at the same time. We apply batch effects by telling Splatter how many cells are in each batch:

```
sim.batches <- splatsimulate(batchCells = c(50, 50), verbose = F
ALSE)
sim.batches <- normalize(sim.batches)

## Warning in .local(object, ...): using library sizes as size f
actors

plotPCA(sim.batches, colour_by = "Batch")
```

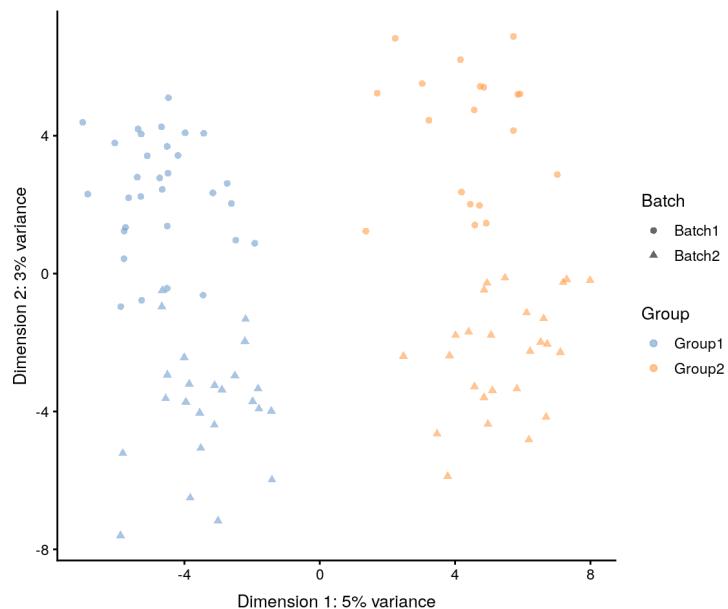


This looks a lot like what we simulated groups and that is because the process is very similar. The difference is that batch effects are applied to all genes, not just those that are differentially expressed, and the effects are usually smaller. By combining groups and batches we can simulate both unwanted variation that we aren't interested in (batch) and the wanted variation we are looking for (group):

```
sim.groups <- splatSimulate(batchCells = c(50, 50), group.prob =
c(0.5, 0.5),
                               method = "groups", verbose = FALSE)
sim.groups <- normalize(sim.groups)

## Warning in .local(object, ...): using library sizes as size factors

plotPCA(sim.groups, shape_by = "Batch", colour_by = "Group")
```



Here we see that the effects of the group (first component) are stronger than the batch effects (second component) but by adjusting the parameters we could made the batch effects dominate.

6.4 Convenience functions

Each of the Splatter simulation methods has it's own convenience function. To simulate a single population use `splatsimulateSingle()` (equivalent to `splatsimulate(method = "single")`), to simulate groups use `splatsimulateGroups()` (equivalent to `splatsimulate(method = "groups")`) or to simulate paths use `splatsimulatePaths()` (equivalent to `splatsimulate(method = "paths")`).

7 Other simulations

As well as it's own Splat simulation method the Splatter package contains implementations of other single-cell RNA-seq simulations that have been published or wrappers around simulations included in other packages. To see all the available simulations run the `listsims()` function:

```
listsims()
```

```
## Splatter currently contains 13 simulations
##
## Splat (splat)
## DOI: 10.1186/s13059-017-1305-0 GitHub: Oshlack/splatter
## The Splat simulation generates means from a gamma distribution, adjusts them for BCV and generates counts from a gamma-poisson. Dropout and batch effects can be optionally added.
##
## Splat single (splatsingle)
## DOI: 10.1186/s13059-017-1305-0 GitHub: Oshlack/splatter
## The Splat simulation with a single population.
##
## Splat Groups (splatGroups)
## DOI: 10.1186/s13059-017-1305-0 GitHub: Oshlack/splatter
## The Splat simulation with multiple groups. Each group can have its own differential expression probability and fold change distribution.
##
## Splat Paths (splatPaths)
## DOI: 10.1186/s13059-017-1305-0 GitHub: Oshlack/splatter
## The Splat simulation with differentiation paths. Each path can have its own length, skew and probability. Genes can change in non-linear ways.
##
## Simple (simple)
## DOI: 10.1186/s13059-017-1305-0 GitHub: Oshlack/splatter
## A simple simulation with gamma means and negative binomial counts.
##
## Lun (lun)
## DOI: 10.1186/s13059-016-0947-7 GitHub: MarioniLab/Deconvolution2016
## Gamma distributed means and negative binomial counts. Cells are given a size factor and differential expression can be simulated with fixed fold changes.
##
## Lun 2 (lun2)
## DOI: 10.1093/biostatistics/kxw055 GitHub: MarioniLab/PlateEffects2016
## Negative binomial counts where the means and dispersions have been sampled from a real dataset. The core feature of the Lun 2 simulation is the addition of plate effects. Differential expression can be added between two groups of plates and optionally a zero-inflated negative-binomial can be used.
##
## scDD (scDD)
## DOI: 10.1186/s13059-016-1077-y GitHub: kdkorthauer/scDD
## The scDD simulation samples a given dataset and can simulate differentially expressed and differentially distributed genes between two conditions.
##
## BASICS (BASICS)
## DOI: 10.1371/journal.pcbi.1004333 GitHub: catavallejos/BASICS
## The BASICS simulation is based on a bayesian model used to deconvolve biological and technical variation and includes spike-ins and batch effects.
##
## mfa (mfa)
## DOI: 10.12688/wellcomeopenres.11087.1 GitHub: kieranrcampbell/mfa
```

```

## The mfa simulation produces a bifurcating pseudotime trajectory. This can optionally include genes with transient changes in expression and added dropout.
##
## PhenoPath (pheno)
## DOI: 10.1101/159913 GitHub: kieranrcampbell/phenopath
## The PhenoPath simulation produces a pseudotime trajectory with different types of genes.
##
## ZINB-WaVE (zinb)
## DOI: 10.1101/125112 GitHub: drisso/zinbwave
## The ZINB-WaVE simulation simulates counts from a sophisticated zero-inflated negative-binomial distribution including cell and gene-level covariates.
##
## SparseDC (sparseDC)
## DOI: 10.1093/nar/gkx1113 GitHub: cran/SparseDC
## The SparseDC simulation simulates a set of clusters across two conditions, where some clusters may be present in only one condition.

```

Each simulation has it's own prefix which gives the name of the functions associated with that simulation. For example the prefix for the simple simulation is `simple` so it would store it's parameters in a `SimpleParams` object that can be created using `newSimpleParams()` or estimated from real data using `simpleEstimate()`. To simulate data using that simulation you would use `simpleSimulate()`. Each simulation returns a `SingleCellExperiment` object with intermediate values similar to that returned by `splatSimulate()`. For more detailed information on each simulation see the appropriate help page (eg. `?simpleSimulate` for information on how the simple simulation works or `?lun2Estimate` for details of how the Lun 2 simulation estimates parameters) or refer to the appropriate paper or package.

8 Other expression values

Splatter is designed to simulate count data but some analysis methods expect other expression values, particularly length-normalised values such as TPM or FPKM. The `scater` package has functions for adding these values to a `SingleCellExperiment` object but they require a length for each gene. The `addGeneLengths` function can be used to simulate these lengths:

```

sim <- simpleSimulate(verbose = FALSE)
sim <- addGeneLengths(sim)
head(rowData(sim))

## DataFrame with 6 rows and 3 columns
##           Gene      GeneMean     Length
##           <factor>    <numeric> <numeric>
## Gene1   Gene1  3.28104748809824    5210
## Gene2   Gene2  2.70699225514288    2618
## Gene3   Gene3  5.02153797531669    4864
## Gene4   Gene4  0.932123671863265   1269
## Gene5   Gene5  1.54033300781193    1746
## Gene6   Gene6  1.99567718581344    5139

```

We can then use `scater` to calculate TPM:

```

tpm(sim) <- calculateTPM(sim, rowData(sim)$Length)
tpm(sim)[1:5, 1:5]

##          cell1    cell2    cell3    cell4    cell5
## Gene1    0.00000 92.24158 30.00788 273.11117 30.16607
## Gene2   120.80264 122.37806 238.87101 301.95000 300.16277
## Gene3    0.00000 131.73757 192.85495 97.51296 193.87158
## Gene4    0.00000 126.23552  0.00000 249.17418 123.84967
## Gene5    90.56738 91.74850  89.54243  0.00000 270.04335

```

The default method used by `addGeneLengths` to simulate lengths is to generate values from a log-normal distribution which are then rounded to give an integer length. The parameters for this distribution are based on human protein coding genes but can be adjusted if needed (for example for other species). Alternatively lengths can be sampled from a provided vector (see `?addGeneLengths` for details and an example).

9 Comparing simulations and real data

One thing you might like to do after simulating data is to compare it to a real dataset, or compare simulations with different parameters or models. Splatter provides a function `compareSCEs` that aims to make these comparisons easier. As the name suggests this function takes a list of `SingleCellExperiment` objects, combines the datasets and produces some plots comparing them. Let's make two small simulations and see how they compare.

```

sim1 <- splatSimulate(nGenes = 1000, batchCells = 20, verbose =
FALSE)
sim2 <- simpleSimulate(nGenes = 1000, nCells = 20, verbose = FALSE)
comparison <- compareSCEs(list(Splat = sim1, Simple = sim2))

names(comparison)

## [1] "RowData" "ColData" "Plots"

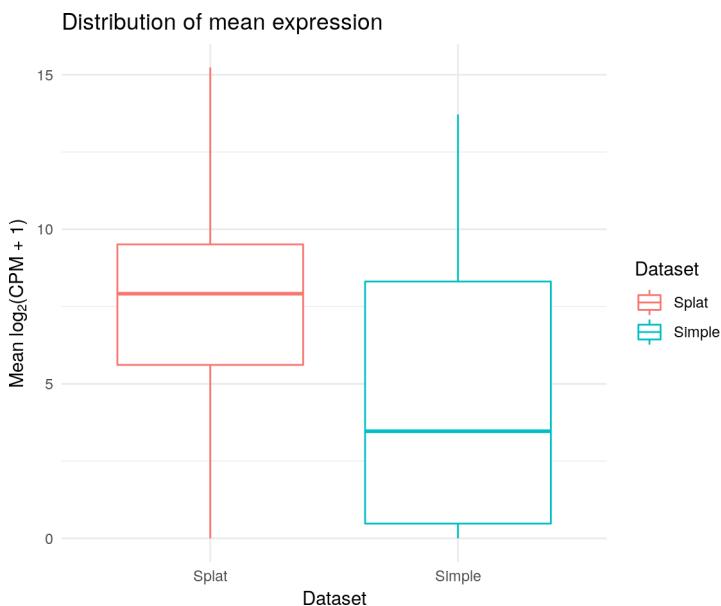
names(comparison$Plots)

## [1] "Means"      "Variances"   "MeanVar"     "LibrarySize
s"
## [5] "ZerosGene"   "ZerosCell"   "MeanZeros"

```

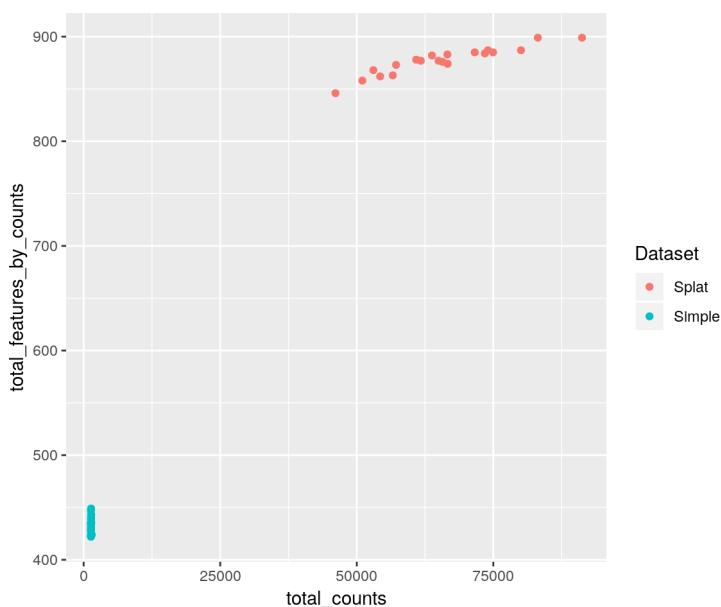
The returned list has three items. The first two are the combined datasets by gene (`RowData`) and by cell (`ColData`) and the third contains some comparison plots (produced using `ggplot2`), for example a plot of the distribution of means:

```
comparison$Plots$Means
```



These are only a few of the plots you might want to consider but it should be easy to make more using the returned data. For example, we could plot the number of expressed genes against the library size:

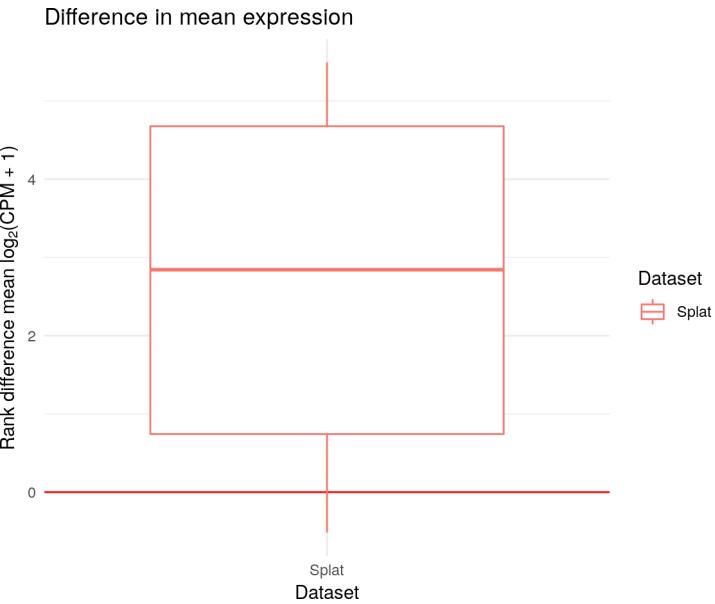
```
library("ggplot2")
ggplot(comparison$colData,
       aes(x = total_counts, y = total_features_by_counts, colour = Dataset)) +
  geom_point()
```



9.1 Comparing differences

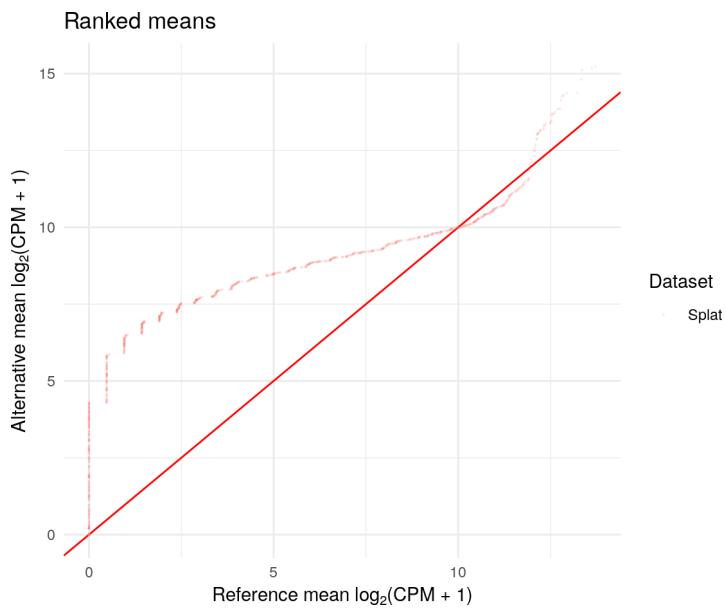
Sometimes instead of visually comparing datasets it may be more interesting to look at the differences between them. We can do this using the `diffsCES` function. Similar to `compareCES` this function takes a list of `SingleCellExperiment` objects but now we also specify one to be a reference. A series of similar plots are returned but instead of showing the overall distributions they demonstrate differences from the reference.

```
difference <- diffSCEs(list(splat = sim1, simple = sim2), ref =
  "Simple")
difference$Plots$Means
```



We also get a series of Quantile-Quantile plot that can be used to compare distributions.

```
difference$QQPlots$Means
```



9.2 Making panels

Each of these comparisons makes several plots which can be a lot to look at. To make this easier, or to produce figures for publications, you can make use of the functions `makeCompPanel`, `makeDiffPanel` and `makeOverallPanel`.

These functions combine the plots into a single panel using the `cowplot` package. The panels can be quite large and hard to view (for example in RStudio's plot viewer) so it can be better to output the panels and view them separately. Luckily `cowplot` provides a convenient function for saving the images. Here are some suggested parameters for outputting each of the panels:

```
# This code is just an example and is not run
panel <- makeCompPanel(comparison)
cowplot::save_plot("comp_panel.png", panel, nrow = 4, ncol = 3)

panel <- makeDiffPanel(difference)
cowplot::save_plot("diff_panel.png", panel, nrow = 3, ncol = 5)

panel <- makeOverallPanel(comparison, difference)
cowplot::save_plot("overall_panel.png", panel, ncol = 4, nrow = 7)
```

10 Citing Splatter

If you use Splatter in your work please cite our paper:

```
citation("splatter")

##
##  Zappia L, Phipson B, Oshlack A. Splatter: simulation of
##  single-cell RNA sequencing data. Genome Biology. 2017;
##  doi:10.1186/s13059-017-1305-0
##
## A BibTeX entry for LaTeX users is
##
##  @Article{,
##    author = {Luke Zappia and Belinda Phipson and Alicia Oshl
##              ack},
##    title = {Splatter: simulation of single-cell RNA sequenci
##              ng data},
##    journal = {Genome Biology},
##    year = {2017},
##    url = {http://dx.doi.org/10.1186/s13059-017-1305-0},
##    doi = {10.1186/s13059-017-1305-0},
##  }
```

Session information

```
sessionInfo()
```

```
## R version 3.6.0 (2019-04-26)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.2 LTS
##
## Matrix products: default
## BLAS:    /home/biocbuild/bbs-3.9-bioc/R/lib/libRblas.so
## LAPACK:  /home/biocbuild/bbs-3.9-bioc/R/lib/libRlapack.so
##
## locale:
## [1] LC_CTYPE=en_US.UTF-8      LC_NUMERIC=C
## [3] LC_TIME=en_US.UTF-8       LC_COLLATE=C
## [5] LC_MONETARY=en_US.UTF-8   LC_MESSAGES=en_US.UTF-8
## [7] LC_PAPER=en_US.UTF-8     LC_NAME=C
## [9] LC_ADDRESS=C              LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel stats4   stats     graphics grDevices utils
datasets
## [8] methods   base
##
## other attached packages:
## [1] scater_1.12.0           ggplot2_3.1.1
## [3] splatter_1.8.0          SingleCellExperiment_1.6.0
## [5] SummarizedExperiment_1.14.0 DelayedArray_0.10.0
## [7] BiocParallel_1.18.0      matrixStats_0.54.0
## [9] Biobase_2.44.0          GenomicRanges_1.36.0
## [11] GenomeInfoDb_1.20.0     IRanges_2.18.0
## [13] S4Vectors_0.22.0        BiocGenerics_0.30.0
## [15] BiocStyle_2.12.0
##
## loaded via a namespace (and not attached):
## [1] viridis_0.5.1            edgeR_3.26.0
## [3] BiocSingular_1.0.0       viridisLite_0.3.0
## [5] splines_3.6.0            lsei_1.2-0
## [7] DelayedMatrixStats_1.6.0 assertthat_0.2.1
## [9] BiocManager_1.30.4       sp_1.3-1
## [11] GenomeInfoDbData_1.2.1  viper_0.4.5
## [13] yaml_2.2.0               pillar_1.3.1
## [15] backports_1.1.4          lattice_0.20-38
## [17] limma_3.40.0             glue_1.3.1
## [19] digest_0.6.18            XVector_0.24.0
## [21] checkmate_1.9.1          colorspace_1.4-1
## [23] cowplot_0.9.4            htmltools_0.3.6
## [25] Matrix_1.2-17            plyr_1.8.4
## [27] pkgconfig_2.0.2          bookdown_0.9
## [29] zlibbioc_1.30.0          purrr_0.3.2
## [31] scales_1.0.0              tibble_2.1.1
## [33] withr_2.1.2              lazyeval_0.2.2
## [35] survival_2.44-1.1        magrittr_1.5
## [37] crayon_1.3.4              evaluate_0.13
## [39] MASS_7.3-51.4             beeswarm_0.2.3
## [41] tools_3.6.0               fitdistrplus_1.0-14
## [43] stringr_1.4.0             munsell_0.5.0
## [45] locfit_1.5-9.1            irlba_2.3.3
## [47] akima_0.6-2              compiler_3.6.0
## [49] rsvd_1.0.0                rlang_0.3.4
## [51] grid_3.6.0                 RCurl_1.95-4.12
## [53] BiocNeighbors_1.2.0        bitops_1.0-6
## [55] labeling_0.3               rmarkdown_1.12
## [57] npsurv_0.4-0              gtable_0.3.0
```

```
## [59] R6_2.4.0                  gridExtra_2.3
## [61] knitr_1.22                 dplyr_0.8.0.1
## [63] stringi_1.4.3              ggbeeswarm_0.6.0
## [65] Rcpp_1.0.1                 tidyselect_0.2.5
## [67] xfun_0.6
```


B.2 Splat parameters vignette

- 1 The base Splat model
- 2 Splat simulation parameters
 - 2.1 Global parameters
 - 2.2 Batch parameters
 - 2.3 Mean parameters
 - 2.4 Library size parameters
 - 2.5 Expression outlier parameters
 - 2.6 Biological Coefficient of Variation (BCV) parameters
 - 2.7 Dropout parameters
 - 2.8 Path parameters

Splat simulation parameters

Last updated: 18 April 2019

This vignette describes the Splat simulation model and the parameters it uses in more detail.

```
library("splatter")
#> Loading required package: SingleCellExperiment
#> Loading required package: SummarizedExperiment
#> Loading required package: GenomicRanges
#> Loading required package: stats4
#> Loading required package: BiocGenerics
#> Loading required package: parallel
#>
#> Attaching package: 'BiocGenerics'
#> The following objects are masked from 'package:parallel':
#>
#>     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
#>     clusterExport, clusterMap, parApply, parCapply, parLapply,
#>     parLapplyLB, parRapply, parSapply, parSapplyLB
#> The following objects are masked from 'package:stats':
#>
#>     IQR, mad, sd, var, xtabs
#> The following objects are masked from 'package:base':
#>
#>     Filter, Find, Map, Position, Reduce, anyDuplicated, append,
#>     as.data.frame, basename, cbind, colnames, dirname, do.call,
#>     duplicated, eval, evalq, get, grep, grepL, intersect,
#>     is.unsorted, lapply, mapply, match, mget, order, paste0,
#>     pmax,
#>     pmax.int, pmin, pmin.int, rank, rbind, rownames, sapply,
#>     setdiff, sort, table, tapply, union, unique, unsplit,
#>     which,
#>     which.max, which.min
#> Loading required package: S4Vectors
#>
#> Attaching package: 'S4Vectors'
#> The following object is masked from 'package:base':
#>
#>     expand.grid
#> Loading required package: IRanges
#> Loading required package: GenomeInfoDb
#> Loading required package: Biobase
#> Welcome to Bioconductor
#>
#>     Vignettes contain introductory material; view with
#>     'browseVignettes()'. To cite Bioconductor, see
#>     'citation("Biobase")', and for packages 'citation("pkgname")'.
#> Loading required package: DelayedArray
#> Loading required package: matrixStats
#>
#> Attaching package: 'matrixStats'
#> The following objects are masked from 'package:Biobase':
#>
#>     anyMissing, rowMedians
#> Loading required package: BiocParallel
#>
#> Attaching package: 'DelayedArray'
#> The following objects are masked from 'package:matrixStats':
#>
```

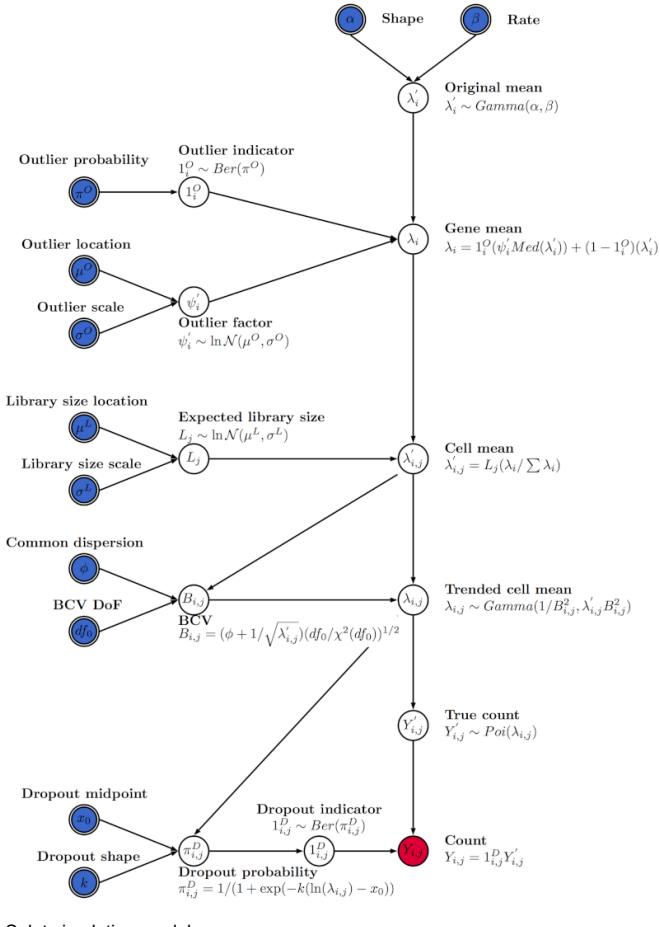
```

#>      colMaxs, colMins, colRanges, rowMaxs, rowMins, rowRanges
#> The following objects are masked from 'package:base':
#>
#>      aperm, apply, rowsum
#> Registered S3 methods overwritten by 'ggplot2':
#>      method      from
#>      [.quosures    rlang
#>      c.quosures    rlang
#>      print.quosures rlang
library("scater")
#> Loading required package: ggplot2
#>
#> Attaching package: 'scater'
#> The following object is masked from 'package:S4Vectors':
#>
#>      rename
#> The following object is masked from 'package:stats':
#>
#>      filter
library("ggplot2")

```

1 The base Splat model

This figure, taken from the Splatter publication, describes the core of the Splat simulation model.



The Splat simulation uses a hierarchical probabilistic where different aspects of a dataset are generated from appropriate statistical distributions. The first stage generates a mean expression level for each gene. These are originally chosen from a Gamma distribution. For some genes that are selected to be outliers with high expression a factor is generated from a log-normal distribution. These factors are then multiplied by the median gene mean to create new means for those genes.

The next stage incorporates variation in the counts per cell. An expected library size (total counts) is chosen for each cell from a log-normal distribution. The library sizes are then used to scale the gene means for each cell, resulting in a range a counts per cell in the simulated dataset. The gene means are then further adjusted to enforce a relationship between the mean expression level and the variability.

The final cell by gene matrix of gene means is then used to generate a count matrix using a Poisson distribution. The result is a synthetic dataset consisting of counts from a Gamma-Poisson (or negative-binomial) distribution. An additional optional step can be used to replicate a “dropout” effect. A probability of dropout is generated using a logistic function based on the underlying mean expression level. A Bernoulli distribution is then used to create a dropout matrix which sets some of the generated counts to zero.

The model described here will generate a single population of cells but the Splat simulation has been designed to be as flexible as possible and can create scenarios including multiple groups of cells (cell types), continuous paths between cell types and multiple experimental batches. The parameters used to create these types of simulations and how they interact with the model are described below.

2 Splat simulation parameters

Within Splatter the parameters for the Splat simulation model are held in the `SplatParams` object. Let's create one of these objects and see what it looks like.

```

params <- newsplatParams()
params
#> A Params object of class SplatParams
#> Parameters can be (estimable) or [not estimable], 'default' or 'NOT DEFAULT'
#>
#> Global:
#> (Genes) (Cells) [Seed]
#> 10000     100   138572
#>
#> 28 additional parameters
#>
#> Batches:
#> [Batches] [Batch Cells] [Location] [Scale]
e]
#>           1          100        0.1       0.
1
#>
#> Mean:
#> (Rate) (shape)
#> 0.3      0.6
#>
#> Library size:
#> (Location) (Scale) (Norm)
#>      11      0.2    FALSE
#>
#> Exprs outliers:
#> (Probability) (Location) (Scale)
#>      0.05      4        0.5
#>
#> Groups:
#> [Groups] [Group Probs]
#>           1            1
#>
#> Diff expr:
#> [Probability] [Down Prob] [Location] [scale]
e]
#>           0.1        0.5        0.1       0.
4
#>
#> BCV:
#> (Common Disp) (DoF)
#>      0.1        60
#>
#> Dropout:
#> [Type] (Midpoint) (Shape)
#> none      0        -1
#>
#> Paths:
#> [From] [Steps] [Skew] [Non-1i
near]
#>           0        100        0.5
0.1
#> [Sigma Factor]
#>           0.8

```

Like all the parameter objects in Splatter printing this object displays all the parameters required for this simulation. As we haven't set any of the parameters the default values are shown but if we were to change any of them they would be highlighted. We can also see which parameters can be

estimated by the Splat estimation procedure and which can't. The default values have been chosen to be fairly realistic but it is recommended that estimation is used to get a simulation that is more like the data you are interested in. Parameters can be modified by setting them in the `SplatParams` object or by providing them directly to the simulation function.

The rest of this section provides details of all this parameters and explains how they can be used with examples.

2.1 Global parameters

These parameters are used in every simulation model and control global features of the dataset produced.

2.1.1 nGenes - Number of genes

The number of genes to simulate.

```
# Set the number of genes to 1000
params <- setParam(params, "nGenes", 1000)

sim <- splatsimulate(params, verbose = FALSE)
dim(sim)
#> [1] 1000 100
```

2.1.2 nCells - Number of cells

The number of genes to simulate. In the Splat simulation this cannot be set directly but must be controlled using the `batchCells` parameter.

2.1.3 seed - Random seed

Seed to use for generating random numbers including selecting values from distributions. By changing this value multiple simulated datasets with the same parameters can be produced. Simulations produced using the same set of parameters and random seed should be identical but there may be differences between operating systems, software versions etc.

2.2 Batch parameters

These parameters control experimental batches in the simulated dataset. The overall effect of how batch effects are included in the model is similar to technical replicates (i.e. the same biological sample sequenced multiple times). This means that the underlying structure is consistent between batches but a global technical signature is added may separate them.

2.2.1 nBatches - Number of batches

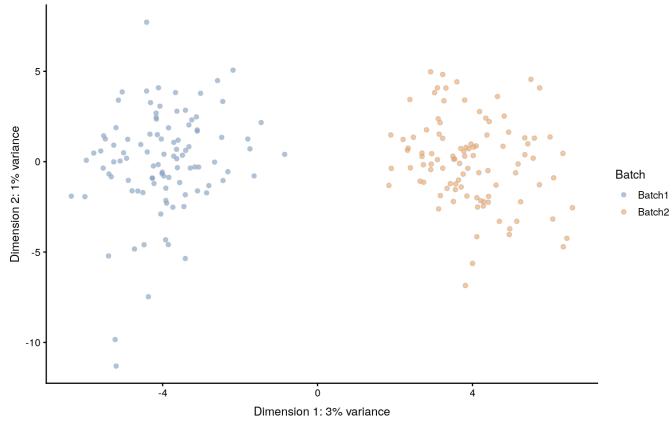
The number of batches in the simulation. This cannot be set directly but is controlled by setting `batchCells`.

2.2.2 batchCells - Cells per batch

A vector specifying the number of cells in each batch. The number of batches (`nBatches`) is equal to the length of the vector and the number of cells (`nCells`) is equal to the sum.

```
# Simulation with two batches of 100 cells
sim <- splatsimulate(params, batchCells = c(100, 100), verbose = FALSE)

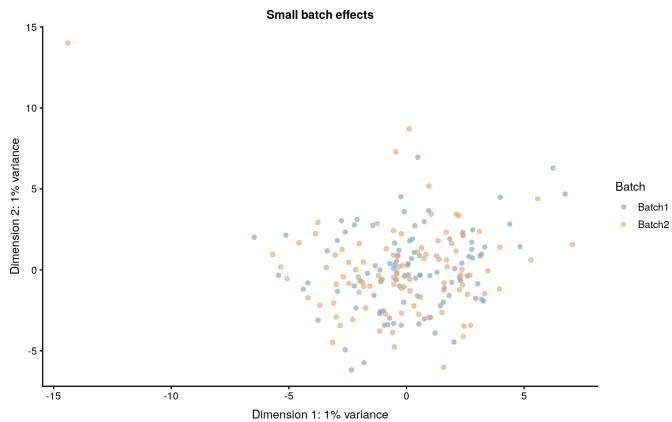
# PCA plot using scatter
sim <- normalize(sim)
#> Warning in .local(object, ...): using library sizes as size factors
plotPCA(sim, colour_by = "Batch")
```



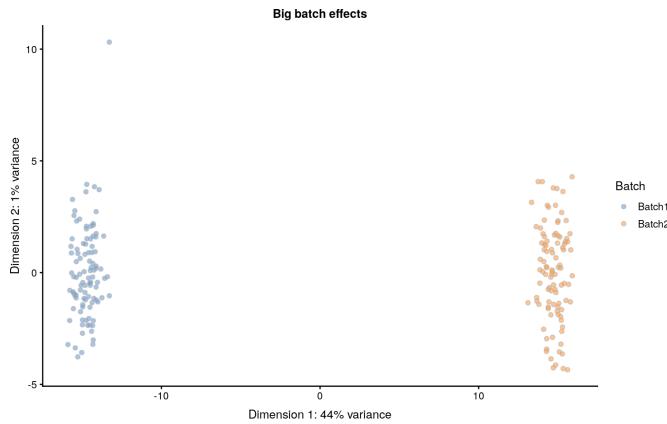
2.2.3 batch.facLoc - Batch factor location and batch.facScale - Batch factor scale

Batches are specified by generating a small scaling factor for each gene in each batch from a log-normal distribution. These factors are then applied to the underlying gene means in each batch. Modifying these parameters affects how different the batches are from each other by generating bigger or smaller factors.

```
# Simulation with small batch effects
sim1 <- splatsimulate(params, batchCells = c(100, 100),
batch.facLoc = 0.001, batch.facScale =
0.001,
verbose = FALSE)
sim1 <- normalize(sim1)
#> Warning in .local(object, ...): using library sizes as size factors
plotPCA(sim1, colour_by = "Batch") + ggtitle("Small batch effects")
```



```
# Simulation with big batch effects
sim2 <- splatsimulate(params, batchCells = c(100, 100),
                       batch.facLoc = 0.5, batch.facScale =
                       0.5,
                       verbose = FALSE)
sim2 <- normalize(sim2)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim2, colour_by = "Batch") + ggtitle("Big batch effe
cts")
```



2.3 Mean parameters

These parameters control the distribution that is used to generate the underlying original gene means.

2.3.1 mean.shape - Mean shape and mean.rate - Mean rate

These parameters control the Gamma distribution that gene means are drawn from. The relationship between shape and rate can be complex and it is often better to use values estimated from a real dataset than to try and manually set them. Although these parameters control the base gene means the means in the final simulation will depend upon other parts of the model, particularly the simulated total counts per cell (library sizes).

2.4 Library size parameters

These parameters control the expected number of counts for each cell. Note that because of sampling the actual counts per cell in the final simulation may be different. Turning on the dropout effect will also effect this. We use the term "library size" here for consistency but expected total counts would be more appropriate.

2.4.1 lib.loc - Library size location and lib.scale - Library size scale

These parameters control the shape of the distribution that is used to generate library sizes for each cell. Increasing `lib.loc` will lead to more counts per cell and increasing `lib.scale` will result in more variability in the counts per cell.

2.4.2 lib.norm - Library size distribution

The default (and recommended) distribution used for library sizes in the Splat simulation is a log-normal. However, in rare cases a normal distribution might be more appropriate. Setting `lib.norm` to `TRUE` will use a normal distribution instead of a log-normal.

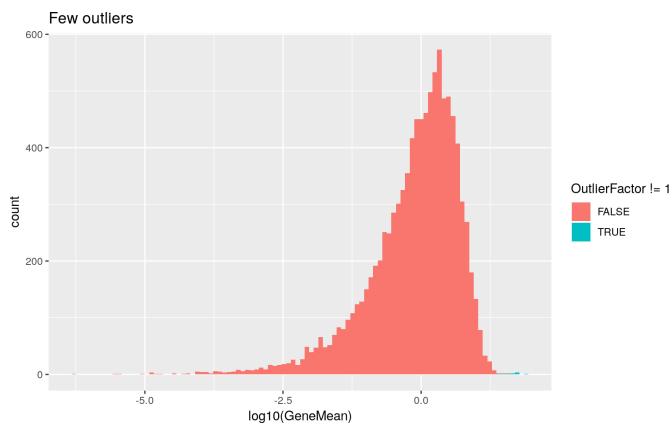
2.5 Expression outlier parameters

When developing the Splat simulation we found that while the Gamma distribution was generally a good match for gene means for some datasets it did not properly capture highly expressed genes. For this reason we added expression outliers to the Splat model.

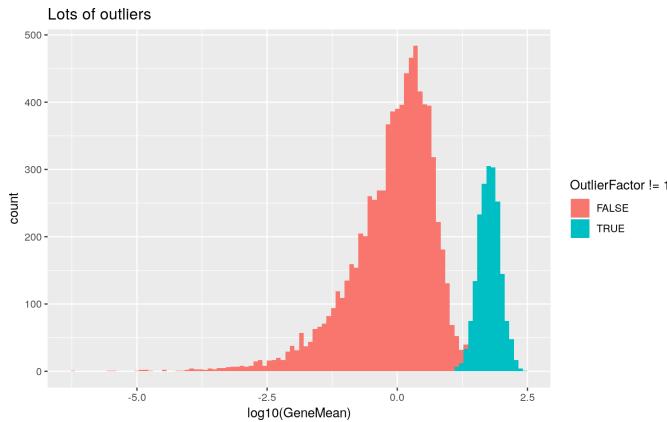
2.5.1 `out.prob` - Expression outlier probability

This parameter controls the probability that genes will be selected to be expression outliers. Higher values will result in more outlier genes.

```
# Few outliers
sim1 <- splatSimulate(out.prob = 0.001, verbose = FALSE)
ggplot(as.data.frame(rowData(sim1)),
       aes(x = log10(GeneMean), fill = OutlierFactor != 1))
+
  geom_histogram(bins = 100) +
  ggtitle("Few outliers")
```



```
# Lots of outliers
sim2 <- splatSimulate(out.prob = 0.2, verbose = FALSE)
ggplot(as.data.frame(rowData(sim2)),
       aes(x = log10(GeneMean), fill = OutlierFactor != 1))
+
  geom_histogram(bins = 100) +
  ggtitle("Lots of outliers")
```



2.5.2 out.facLoc - Expression outlier factor location and out.facScale - Expression outlier factor scale

The expression outlier factors are drawn from a log-normal distribution controlled by these parameters. The generated factors are applied to the median mean expression rather than the existing mean for the selected genes. This is to be consistent with the estimation procedure for these factors and to make sure that the final means are outliers. For example to avoid the situation where a factor is applied to a lowly expressed gene, making it just moderately expressed rather than an expression outlier.

2.5.3 Group parameters

Up until this stage of the simulation only a single population of cells is considered but we often want to simulate datasets with multiple kinds of cells. We do this by assigning cells to groups.

2.5.4 nGroups - Number of groups

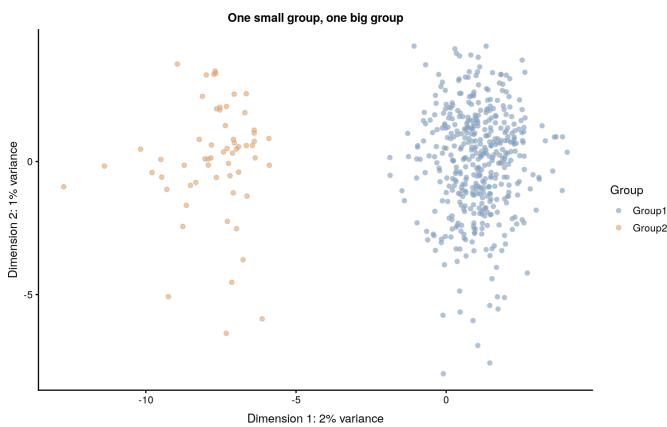
The number of groups to simulate. This parameter cannot be set directly and is controlled using `group.prob`.

2.5.5 group.prob - Group probabilities

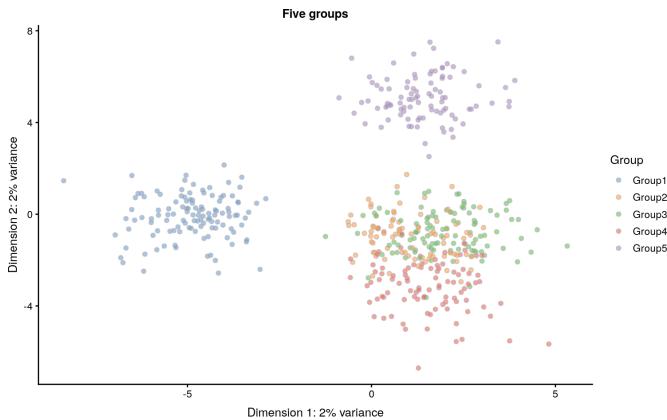
A vector giving the probability that cells will be assigned to groups. The length of the vector gives the number of groups (`nGroups`) and the probabilities must sum to 1. Adjusting the number and relative values of the probabilities changes the number and relative sizes of the groups. To simulate groups we also need to use the `splatSimulateGroups` function or set `method = "groups"`.

```
params.groups <- newSplatParams(batchCells = 500, nGenes = 1000)

# One small group, one big group
sim1 <- splatSimulateGroups(params.groups, group.prob = c(0.9, 0.1),
                             verbose = FALSE)
sim1 <- normalize(sim1)
#> Warning in .local(object, ...): using library sizes as size factors
plotPCA(sim1, colour_by = "Group") + ggtitle("One small group, one big group")
```



```
# Five groups
sim2 <- splatsimulateGroups(params.groups,
                             group.prob = c(0.2, 0.2, 0.2, 0.
                             2, 0.2),
                             verbose = FALSE)
sim2 <- normalize(sim2)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim2, colour_by = "Group") + ggtitle("Five groups")
```



Note: Once there are more than three or four groups it becomes difficult to properly view them in PCA space. We use PCA here for simplicity but generally a non-linear dimensionality reduction such as t-SNE or UMAP is a more useful way to visualise the groups.

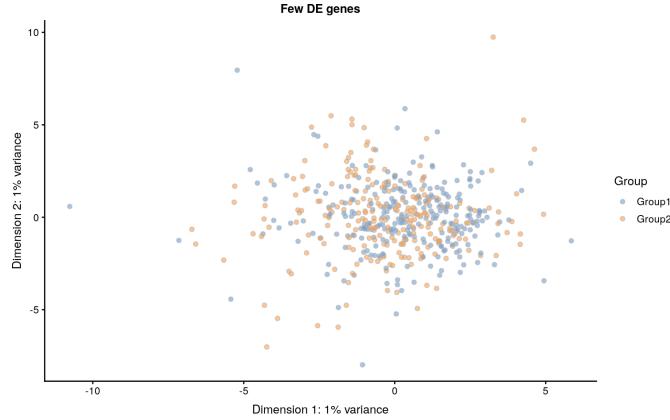
2.5.6 Differential expression parameters

Different groups are created by modifying the base expression levels of selected genes. The process for doing this is to simulate differential expression (DE) between each group and a fictional base cell. Altering the differential expression parameters controls how similar groups are to each other.

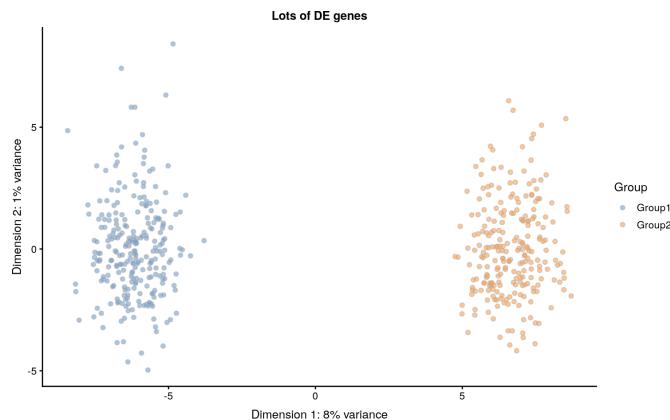
2.5.7 de.prob - DE probability

This parameter controls the probability that a gene will be selected to be differentially expressed.

```
# Few DE genes
sim1 <- splat$simulateGroups(params.groups, group.prob = c(0.5, 0.5),
                               de.prob = 0.01, verbose = FALSE)
sim1 <- normalize(sim1)
#> Warning in .local(object, ...): using library sizes as size factors
plotPCA(sim1, colour_by = "Group") + ggtitle("Few DE genes")
```



```
# Lots of DE genes
sim2 <- splat$simulateGroups(params.groups, group.prob = c(0.5, 0.5),
                               de.prob = 0.3, verbose = FALSE)
sim2 <- normalize(sim2)
#> Warning in .local(object, ...): using library sizes as size factors
plotPCA(sim2, colour_by = "Group") + ggtitle("Lots of DE genes")
```



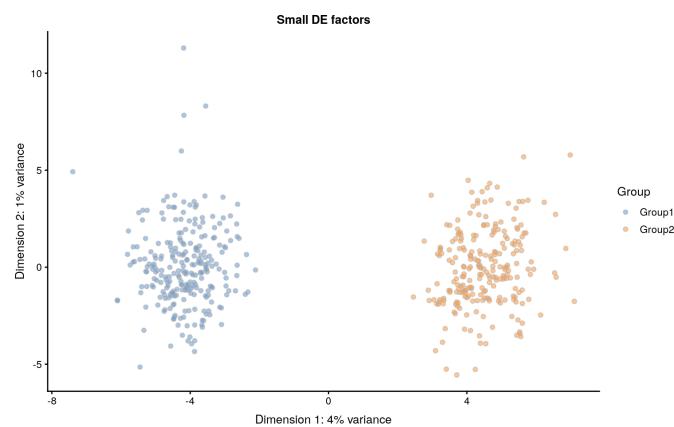
2.5.8 de.downProb - Down-regulation probability

A selected DE gene can be either down-regulated (has a factor less than one) or up-regulated (has a factor greater than one). This parameter controls the probability that a selected gene will be down-regulated.

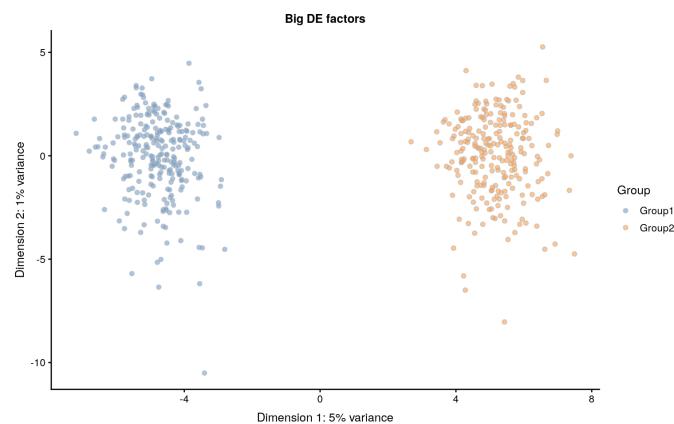
2.5.9 de.facLoc - DE factor location and de.facScale - DE factor scale

Differential expression factors are produced from a log-normal distribution in a similar way to batch effect factors and expression outlier factors. Changing these parameters can result in more or less extreme differences between groups.

```
# Small DE factors
sim1 <- splatSimulateGroups(params.groups, group.prob = c(0.5, 0.5),
                             de.facLoc = 0.01, verbose = FALSE)
sim1 <- normalize(sim1)
#> Warning in .local(object, ...): using library sizes as size factors
plotPCA(sim1, colour_by = "Group") + ggtitle("Small DE factors")
```



```
# Big DE factors
sim2 <- splatSimulateGroups(params.groups, group.prob = c(0.5, 0.5),
                             de.facLoc = 0.3, verbose = FALSE)
sim2 <- normalize(sim2)
#> Warning in .local(object, ...): using library sizes as size factors
plotPCA(sim2, colour_by = "Group") + ggtitle("Big DE factors")
```

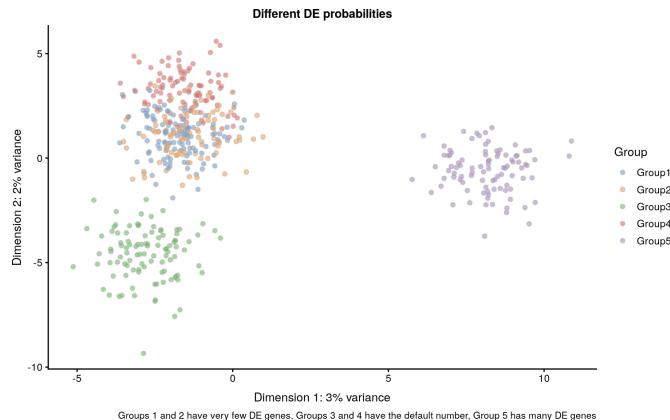


Just looking at the PCA plots this effect seems similar to adjusting `de.prob` but the effect is achieved in a different way. A higher `de.prob` means that more genes are differentially expressed but changing the DE factors changes the level of DE for the same number of genes.

2.5.10 Complex differential expression

Each of the differential expression parameters can be specified for each group by providing a vector of values. These vectors must be the same length as `group.prob`. Specifying parameters as vectors allows more complex simulations where groups are more or less different to each other rather than being equally distinct. Here are some examples of different DE scenarios.

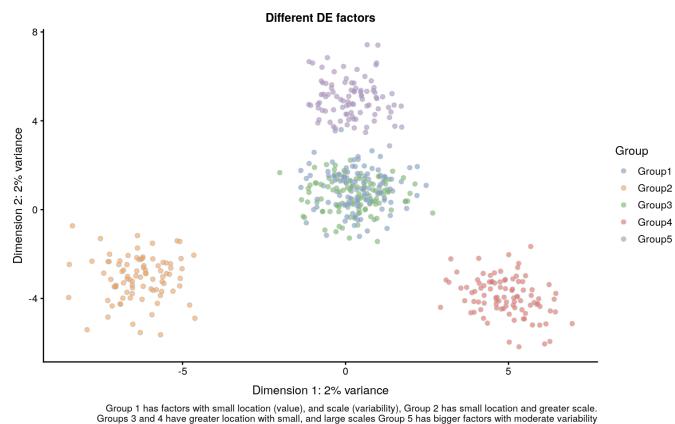
```
# Different DE probs
sim1 <- splat$simulateGroups(params.groups,
                               group.prob = c(0.2, 0.2, 0.2,
                               0.2, 0.2),
                               de.prob = c(0.01, 0.01, 0.1, 0.1
                               , 0.3),
                               verbose = FALSE)
sim1 <- normalize(sim1)
#> Warning in .local(object, ...): using library sizes as size factors
plotPCA(sim1, colour_by = "Group") +
  labs(title = "Different DE probabilities",
       caption = paste("Groups 1 and 2 have very few DE genes",
                      "Groups 3 and 4 have the default number",
                      "Group 5 has many DE genes"))
```



```

# Different DE factors
sim2 <- splatSimulateGroups(params.groups,
                             group.prob = c(0.2, 0.2, 0.2, 0.
                             2, 0.2),
                             de.facLoc = c(0.01, 0.01, 0.1,
                             0.1, 0.2),
                             de.facScale = c(0.2, 0.5, 0.2,
                             0.5, 0.4),
                             verbose = FALSE)
sim2 <- normalize(sim2)
#> Warning in .local(object, ...): using library sizes as si
ze factors
plotPCA(sim2, colour_by = "Group") +
  labs(title = "Different DE factors",
       caption = paste("Group 1 has factors with small loc
ation (value),",
                      "and scale (variability),",
                      "Group 2 has small location and gre
ater scale.\n",
                      "Groups 3 and 4 have greater locati
on with small,",
                      "and large scales",
                      "Group 5 has bigger factors with mo
derate variability"))

```

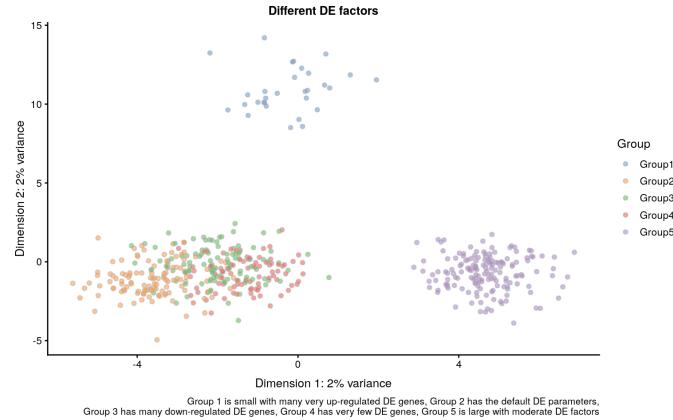


```
# Combination of everything
sim3 <- splat$simulateGroups(params.groups,
                                group.prob = c(0.05, 0.2, 0.2,
                                0.2, 0.35),
                                de.prob = c(0.3, 0.1, 0.2, 0.01,
                                0.1),
                                de.downProb = c(0.1, 0.4, 0.9,
                                0.6, 0.5),
                                de.facLoc = c(0.6, 0.1, 0.1, 0.0
                                1, 0.2),
                                de.facscale = c(0.1, 0.4, 0.2,
                                0.5, 0.4),
                                verbose = FALSE)
sim3 <- normalize(sim3)
#> Warning in .local(object, ...): using library sizes as size factors
plotPCA(sim3, colour_by = "Group") +
  labs(title = "Different DE factors",
       caption = paste(
         "Group 1 is small with many very up-regulated D
E genes,",

         "Group 2 has the default DE parameters,\n",
         "Group 3 has many down-regulated DE genes,",

         "Group 4 has very few DE genes,",

         "Group 5 is large with moderate DE factors"))
)
```



2.6 Biological Coefficient of Variation (BCV) parameters

The BCV parameters control the variability of the genes in the simulated dataset.

2.6.1 bcv.common - Common BCV

The `bcv.common` parameter controls the underlying common variability across all genes in the dataset.

2.6.2 bcv.df - BCV Degrees of Freedom

This parameter sets the degrees of freedom used in the BCV inverse chi-squared distribution. Changing this changes the effect of mean expression on the variability of a gene.

2.7 Dropout parameters

These parameters control whether additional dropout is added to increase the number of zeros in the simulated dataset and if it is how that is applied.

2.7.1 `dropout.type` - Dropout type

This parameter determines the kind of dropout effect to simulate. Setting it to "none" means no dropout, "experiment" is global dropout using the same set of parameters for every cell, "batch" uses the same parameters for every cell in the same batch, "group" uses the same parameters for every cell in the same group and "cell" uses a different set of parameters for every cell.

2.7.2 `dropout.mid` - Dropout mid point and `dropout.shape` - Dropout shape

The probability that a particular count in a particular cell is set to zero is related to the mean expression of that gene in that cell. This relationship is represented using a logistic function with these parameters. The `dropout.mid` parameter controls the point at which the probability is equal to 0.5 and the `dropout.shape` controls how it changes with increasing expression. These parameters must be vectors of the appropriate length depending on the selected dropout type.

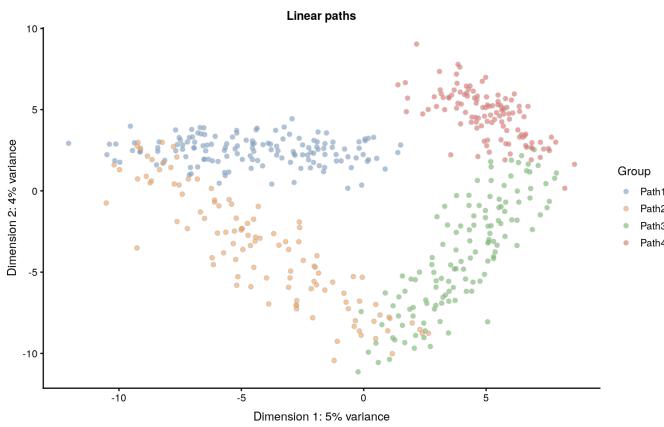
2.8 Path parameters

For many uses simulating groups is sufficient but in some cases it is more appropriate to simulate continuous changes between cell types. The number of paths and the probability of assigning cells to them is still controlled by the `group.prob` parameter and the amount of change along a path is controlled by the DE parameters but other aspects of the `splatSimulatePaths` model are controlled by these parameters.

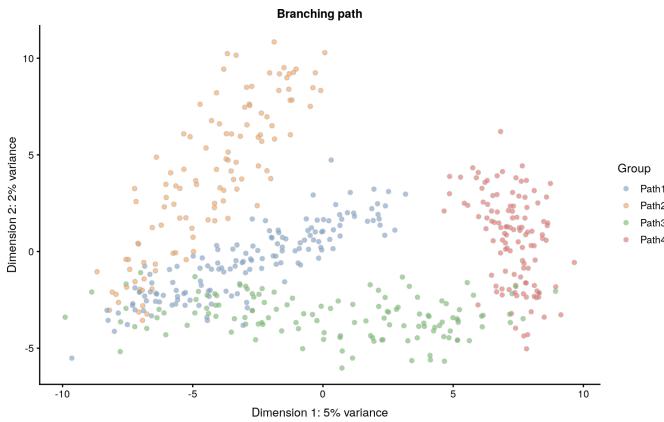
2.8.1 `path.from` - Path origin

This parameter controls the order of differentiation paths. It is a vector the same length as `group.prob` giving the starting position of each path. For example a `path.from` of `c(0, 1, 1, 3)` would indicate the Path 1 starts at the origin (0), Path 2 starts at the end of Path 1, Path 3 also starts at the end of Path 1 (a branch point) and Path 4 starts at the end of Path 3.

```
# Linear paths
sim1 <- splatSimulatePaths(params.groups,
                           group.prob = c(0.25, 0.25, 0.25,
                           0.25),
                           de.prob = 0.5, de.facLoc = 0.2,
                           path.from = c(0, 1, 2, 3),
                           verbose = FALSE)
sim1 <- normalize(sim1)
#> Warning in .local(object, ...): using library sizes as size factors
plotPCA(sim1, colour_by = "Group") + ggtitle("Linear paths")
```



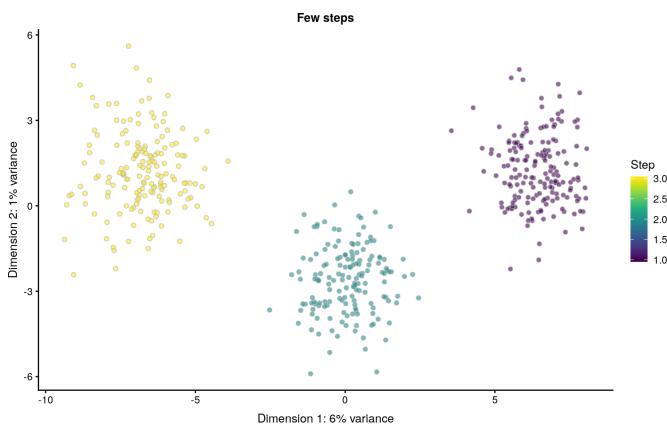
```
# Branching path
sim2 <- splatsimulatePaths(params.groups,
                           group.prob = c(0.25, 0.25, 0.25,
                           0.25),
                           de.prob = 0.5, de.facLoc = 0.2,
                           path.from = c(0, 1, 1, 3),
                           verbose = FALSE)
sim2 <- normalize(sim2)
#> Warning in .local(object, ...): using library sizes as size factors
plotPCA(sim2, colour_by = "Group") + ggtitle("Branching path")
```



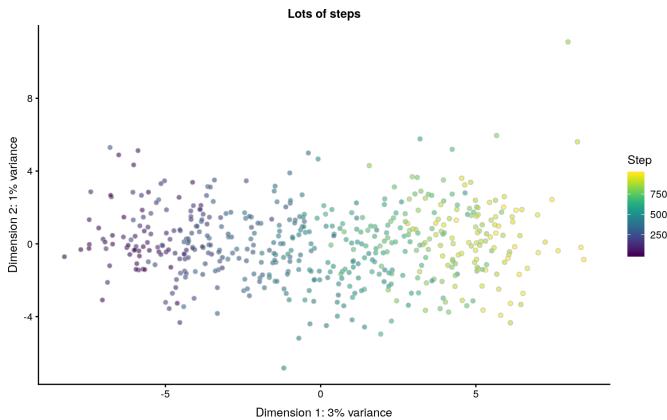
2.8.2 path.nsteps - Number of steps

A path is created by using the same differential expression procedure as used for groups to generate an end point. Interpolation is then used to create a series of steps between the start and end points. This parameter controls the number of steps along a path and therefore how discrete or smooth it is.

```
# Few steps
sim1 <- splatsimulatePaths(params.groups, path.nsteps = 3,
                           de.prob = 0.5, de.facLoc = 0.2, v
                           erbose = FALSE)
sim1 <- normalize(sim1)
#> Warning in .local(object, ...): using library sizes as size factors
plotPCA(sim1, colour_by = "Step") + ggtitle("Few steps")
```



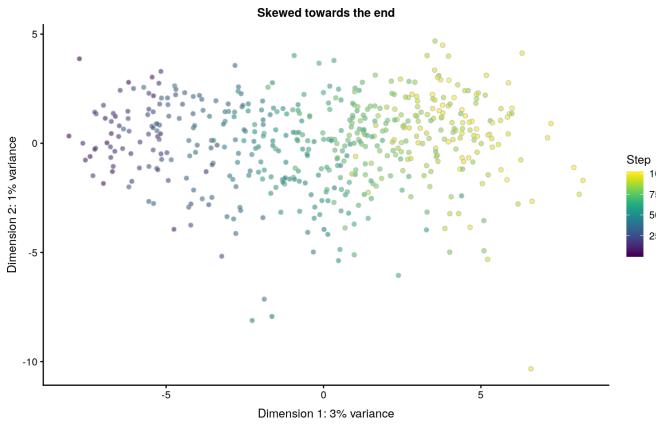
```
# Lots of steps
sim2 <- splatsimulatePaths(params.groups, path.nSteps = 1000,
                           de.prob = 0.5, de.facLoc = 0.2, verbose = FALSE)
sim2 <- normalize(sim2)
#> Warning in .local(object, ...): using library sizes as size factors
plotPCA(sim2, colour_by = "Step") + ggtitle("Lots of steps")
```



2.8.3 path.skew - Path skew

By default cells are evenly distributed along a path but it sometimes be useful to introduce a skew in the distribution. For example you may want to simulate a scenario with few stem-like cells and many differentiated cells. Setting `path.skew` to 0 will mean that all cells come from the end point while higher values up to 1 will skew them towards the start point.

```
# Skew towards the end
sim1 <- splatsimulatePaths(params.groups, path.skew = 0.1,
                           de.prob = 0.5, de.facLoc = 0.2, verbose = FALSE)
sim1 <- normalize(sim1)
#> Warning in .local(object, ...): using library sizes as size factors
plotPCA(sim1, colour_by = "Step") + ggtitle("Skewed towards the end")
```



2.8.4 `path.nonlinearProb` - Non-linear probability

Most genes are interpolated in a linear way along a path but in reality this may not always be the case. For example it is easy to imagine a gene that is lowly-expressed at the start of a process, highly-expressed in the middle and lowly-expressed again at the end. The `path.nonlinearProb` parameter controls the probability that a gene will change in a non-linear way along a path.

2.8.5 `path.sigmaFac` - Path skew

Non-linear changes along a path are achieved by building a Brownian bridge between the two end points. A Brownian bridge is Brownian motion controlled in such a way that the end points are fixed. The `path.sigmaFac` parameter controls how extreme each step in the Brownian motion is and therefore how much the interpolation differs from a linear path.

B.3 Splatter manual

Package ‘splatter’

July 31, 2019

Type Package
Title Simple Simulation of Single-cell RNA Sequencing Data
Version 1.8.0
Date 2019-04-18
Author Luke Zappia
Maintainer Luke Zappia <luke.zappia@mcri.edu.au>
Description Splatter is a package for the simulation of single-cell RNA sequencing count data. It provides a simple interface for creating complex simulations that are reproducible and well-documented. Parameters can be estimated from real data and functions are provided for comparing real and simulated datasets.
License GPL-3 + file LICENSE
LazyData TRUE
Depends R (>= 3.4), SingleCellExperiment
Imports akima, BiocGenerics, BiocParallel, checkmate, edgeR, fitdistrplus, ggplot2, locfit, matrixStats, methods, scales, scater (>= 1.7.4), stats, SummarizedExperiment, utils, crayon
Suggests BiocStyle, covr, cowplot, knitr, limSolve, lme4, progress, pscl, testthat, rmarkdown, S4Vectors, scDD, scran, mfa, phenopath, BASiCS, zinbwave, SparseDC, BiocManager, spelling
biocViews SingleCell, RNASeq, Transcriptomics, GeneExpression, Sequencing, Software, ImmunoOncology
URL <https://github.com/Oshlack/splatter>
BugReports <https://github.com/Oshlack/splatter/issues>
RoxygenNote 6.1.1
Encoding UTF-8
VignetteBuilder knitr
Language en-GB
git_url https://git.bioconductor.org/packages/splatter
git_branch RELEASE_3_9
git_last_commit 0195bb5
git_last_commit_date 2019-05-02
Date/Publication 2019-07-30

R topics documented:

addFeatureStats	3
addGeneLengths	4
BASiCSEstimate	5
BASiCSPParams	6
BASiCSSimulate	7
bridge	8
bringItemsForward	8
compareSCEs	9
diffSCEs	10
expandParams	11
getLNormFactors	12
getParam	12
getParams	13
getPathOrder	13
listSims	14
logistic	14
lun2Estimate	15
Lun2Params	16
lun2Simulate	17
lunEstimate	18
LunParams	19
lunSimulate	19
makeCompPanel	20
makeDiffPanel	21
makeOverallPanel	22
mfaEstimate	22
MFAPParams	23
mfaSimulate	24
newParams	25
Params	26
phenoEstimate	26
PhenoParams	27
phenoSimulate	27
rbindMatched	28
scDDEstimate	29
SCDDPParams	30
scDDSimulate	31
setParam	32
setParams	33
setParamsUnchecked	34
setParamUnchecked	34
showDFs	35
showPP	35
showValues	36
simpleEstimate	36
SimpleParams	37
simpleSimulate	37
sparseDCEstimate	38
SparseDCPParams	39
sparseDCSimulate	40

<i>addFeatureStats</i>	3
splatEstBCV	41
splatEstDropout	41
splatEstimate	42
splatEstLib	43
splatEstMean	43
splatEstOutlier	44
SplatParams	44
splatSimBatchCellMeans	46
splatSimBatchEffects	46
splatSimBCVMeans	47
splatSimCellMeans	47
splatSimDE	48
splatSimDropout	48
splatSimGeneMeans	49
splatSimLibSizes	49
splatSimTrueCounts	50
splatSimulate	50
splatter	52
summariseDiff	53
winsorize	53
zinbEstimate	54
ZINBParams	55
zinbSimulate	55

Index	57
--------------	-----------

addFeatureStats	<i>Add feature statistics</i>
------------------------	-------------------------------

Description

Add additional feature statistics to a SingleCellExperiment object

Usage

```
addFeatureStats(sce, value = c("counts", "cpm", "tpm", "fpkm"),
log = FALSE, offset = 1, no.zeros = FALSE)
```

Arguments

sce	SingleCellExperiment to add feature statistics to.
value	the expression value to calculate statistics for. Options are "counts", "cpm", "tpm" or "fpkm". The values need to exist in the given SingleCellExperiment.
log	logical. Whether to take log2 before calculating statistics.
offset	offset to add to avoid taking log of zero.
no.zeros	logical. Whether to remove all zeros from each feature before calculating statistics.

4

*addGeneLengths***Details**

Currently adds the following statistics: mean, variance, coefficient of variation, median and median absolute deviation. Statistics are added to the `rowData` slot and are named `Stat[Log]Value[No0]` where Log and No0 are added if those arguments are true. UpperCamelCase is used to differentiate these columns from those added by analysis packages.

Value

SingleCellExperiment with additional feature statistics

*addGeneLengths**Add gene lengths***Description**

Add gene lengths to an SingleCellExperiment object

Usage

```
addGeneLengths(sce, method = c("generate", "sample"), loc = 7.9,  
               scale = 0.7, lengths = NULL)
```

Arguments

sce	SingleCellExperiment to add gene lengths to.
method	Method to use for creating lengths.
loc	Location parameter for the generate method.
scale	Scale parameter for the generate method.
lengths	Vector of lengths for the sample method.

Details

This function adds simulated gene lengths to the `rowData` slot of a `SingleCellExperiment` object that can be used for calculating length normalised expression values such as TPM or FPKM. The `generate` method simulates lengths using a (rounded) log-normal distribution, with the default `loc` and `scale` parameters based on human protein-coding genes. Alternatively the `sample` method can be used which randomly samples lengths (with replacement) from a supplied vector.

Value

SingleCellExperiment with added gene lengths

Examples

```
# Default generate method  
sce <- simpleSimulate()  
sce <- addGeneLengths(sce)  
head(rowData(sce))  
# Sample method (human coding genes)  
## Not run:  
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
```

5

```

library(GenomicFeatures)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
tx.lens <- transcriptLengths(txdb, with.cds_len = TRUE)
tx.lens <- tx.lens[tx.lens$cds_len > 0, ]
gene.lens <- max(splitAsList(tx.lens$tx_len, tx.lens$gene_id))
sce <- addGeneLengths(sce, method = "sample", lengths = gene.lens)

## End(Not run)

```

BASiCSEstimate*Estimate BASiCS simulation parameters***Description**

Estimate simulation parameters for the BASiCS simulation from a real dataset.

Usage

```

BASiCSEstimate(counts, spike.info = NULL, batch = NULL, n = 20000,
                thin = 10, burn = 5000, regression = TRUE,
                params = newBASiCSParams(), verbose = TRUE, progress = TRUE, ...)

## S3 method for class 'SingleCellExperiment'
BASiCSEstimate(counts, spike.info = NULL,
                batch = NULL, n = 20000, thin = 10, burn = 5000,
                regression = TRUE, params = newBASiCSParams(), verbose = TRUE,
                progress = TRUE, ...)

## S3 method for class 'matrix'
BASiCSEstimate(counts, spike.info = NULL,
                batch = NULL, n = 20000, thin = 10, burn = 5000,
                regression = TRUE, params = newBASiCSParams(), verbose = TRUE,
                progress = TRUE, ...)

```

Arguments

counts	either a counts matrix or a SingleCellExperiment object containing count data to estimate parameters from.
spike.info	data.frame describing spike-ins with two columns: "Name" giving the names of the spike-in features (must match rownames(counts)) and "Input" giving the number of input molecules.
batch	vector giving the batch that each cell belongs to.
n	total number of MCMC iterations. Must be $\geq \max(4, \text{thin})$ and a multiple of thin.
thin	thinning period for the MCMC sampler. Must be ≥ 2 .
burn	burn-in period for the MCMC sampler. Must be in the range $1 \leq \text{burn} < \text{n}$ and a multiple of thin.
regression	logical. Whether to use regression to identify over-dispersion. See BASiCS_MCMC for details.
params	BASiCSParams object to store estimated values in.

6

BASiCSPParams

<code>verbose</code>	logical. Whether to print progress messages.
<code>progress</code>	logical. Whether to print additional BASiCS progress messages.
<code>...</code>	Optional parameters passed to BASiCS_MCMC .

Details

This function is just a wrapper around [BASiCS_MCMC](#) that takes the output and converts it to a *BASiCSPParams* object. Either a set of spike-ins or batch information (or both) must be supplied. If only batch information is provided there must be at least two batches. See [BASiCS_MCMC](#) for details.

Value

BASiCSPParams object containing the estimated parameters.

Examples

```
## Not run:
# Load example data
library(scater)
data("sc_example_counts")

spike.info <- data.frame(Name = rownames(sc_example_counts)[1:10],
                         Input = rnorm(10, 500, 200),
                         stringsAsFactors = FALSE)
params <- BASiCSEstimate(sc_example_counts[1:100, 1:30],
                        spike.info)
params

## End(Not run)
```

BASiCSPParams

*The BASiCSPParams class***Description**

S4 class that holds parameters for the BASiCS simulation.

Parameters

The BASiCS simulation uses the following parameters:

`nGenes` The number of genes to simulate.

`nCells` The number of cells to simulate.

`[seed]` Seed to use for generating random numbers.

Batch parameters `nBatches` Number of batches to simulate.

`batchCells` Number of cells in each batch.

Gene parameters `gene.params` A `data.frame` containing gene parameters with two columns: Mean (mean expression for each biological gene) and Delta (cell-to-cell heterogeneity for each biological gene).

Spike-in parameters `nSpikes` The number of spike-ins to simulate.

`spike.means` Input molecules for each spike-in.

BASiCSSimulate

7

Cell parameters `cell.params` A `data.frame` containing gene parameters with two columns: Phi (mRNA content factor for each cell, scaled to sum to the number of cells in each batch) and S (capture efficient for each cell).

Variability parameters `theta` Technical variability parameter for each batch.

The parameters not shown in brackets can be estimated from real data using [BASiCSEstimate](#). For details of the BASiCS simulation see [BASiCSSimulate](#).

BASiCSSimulate*BASiCS simulation*

Description

Simulate counts using the BASiCS method.

Usage

```
BASiCSSimulate(params = newBASiCSParams(), verbose = TRUE, ...)
```

Arguments

<code>params</code>	<code>BASiCSParams</code> object containing simulation parameters.
<code>verbose</code>	logical. Whether to print progress messages
<code>...</code>	any additional parameter settings to override what is provided in <code>params</code> .

Details

This function is just a wrapper around [BASiCS_Sim](#) that takes a `BASiCSParams`, runs the simulation then converts the output to a `SingleCellExperiment` object. See [BASiCS_Sim](#) for more details of how the simulation works.

Value

`SingleCellExperiment` containing simulated counts

References

Vallejos CA, Marioni JC, Richardson S. BASiCS: Bayesian Analysis of Single-Cell Sequencing data. PLoS Computational Biology (2015).

Paper: [10.1371/journal.pcbi.1004333](https://doi.org/10.1371/journal.pcbi.1004333)

Code: <https://github.com/catavallejos/BASiCS>

Examples

```
sim <- BASiCSSimulate()
```

8

bringItemsForward

bridge*Brownian bridge*

Description

Calculate a smoothed Brownian bridge between two points. A Brownian bridge is a random walk with fixed end points.

Usage

```
bridge(x = 0, y = 0, N = 5, n = 100, sigma.fac = 0.8)
```

Arguments

x	starting value.
y	end value.
N	number of steps in random walk.
n	number of points in smoothed bridge.
sigma.fac	multiplier specifying how extreme each step can be.

Value

Vector of length n following a path from x to y.

bringItemsForward*Bring items forward*

Description

Move selected items to the start of a list.

Usage

```
bringItemsForward(l1, items)
```

Arguments

l1	list to adjust item order.
items	vector of items to bring to the front. Any not in the list will be ignored.

Value

list with selected items first

`compareSCEs`

9

`compareSCEs` *Compare SingleCellExperiment objects*

Description

Combine the data from several SingleCellExperiment objects and produce some basic plots comparing them.

Usage

```
compareSCEs(sces, point.size = 0.1, point.alpha = 0.1, fits = TRUE,  
            colours = NULL)
```

Arguments

<code>sces</code>	named list of SingleCellExperiment objects to combine and compare.
<code>point.size</code>	size of points in scatter plots.
<code>point.alpha</code>	opacity of points in scatter plots.
<code>fits</code>	whether to include fits in scatter plots.
<code>colours</code>	vector of colours to use for each dataset.

Details

The returned list has three items:

`RowData` Combined row data from the provided SingleCellExperiments.

`ColData` Combined column data from the provided SingleCellExperiments.

`Plots` Comparison plots

- `Means` Boxplot of mean distribution.
- `Variances` Boxplot of variance distribution.
- `MeanVar` Scatter plot with fitted lines showing the mean-variance relationship.
- `LibrarySizes` Boxplot of the library size distribution.
- `ZerosGene` Boxplot of the percentage of each gene that is zero.
- `ZerosCell` Boxplot of the percentage of each cell that is zero.
- `MeanZeros` Scatter plot with fitted lines showing the mean-zeros relationship.

The plots returned by this function are created using `ggplot` and are only a sample of the kind of plots you might like to consider. The data used to create these plots is also returned and should be in the correct format to allow you to create further plots using `ggplot`.

Value

List containing the combined datasets and plots.

Examples

```
sim1 <- splatSimulate(nGenes = 1000, batchCells = 20)  
sim2 <- simpleSimulate(nGenes = 1000, nCells = 20)  
comparison <- compareSCEs(list(Splat = sim1, Simple = sim2))  
names(comparison)  
names(comparison$Plots)
```

10

diffSCEs

*diffSCEs**Diff SingleCellExperiment objects*

Description

Combine the data from several SingleCellExperiment objects and produce some basic plots comparing them to a reference.

Usage

```
diffSCEs(sces, ref, point.size = 0.1, point.alpha = 0.1, fits = TRUE,  
colours = NULL)
```

Arguments

<code>sces</code>	named list of SingleCellExperiment objects to combine and compare.
<code>ref</code>	string giving the name of the SingleCellExperiment to use as the reference
<code>point.size</code>	size of points in scatter plots.
<code>point.alpha</code>	opacity of points in scatter plots.
<code>fits</code>	whether to include fits in scatter plots.
<code>colours</code>	vector of colours to use for each dataset.

Details

This function aims to look at the differences between a reference SingleCellExperiment and one or more others. It requires each SingleCellExperiment to have the same dimensions. Properties are compared by ranks, for example when comparing the means the values are ordered and the differences between the reference and another dataset plotted. A series of Q-Q plots are also returned.

The returned list has five items:

`Reference` The SingleCellExperiment used as the reference.

`RowData` Combined feature data from the provided SingleCellExperiments.

`ColData` Combined column data from the provided SingleCellExperiments.

`Plots` Difference plots

`Means` Boxplot of mean differences.

`Variances` Boxplot of variance differences.

`MeanVar` Scatter plot showing the difference from the reference variance across expression ranks.

`LibraeySizes` Boxplot of the library size differences.

`ZerosGene` Boxplot of the differences in the percentage of each gene that is zero.

`ZerosCell` Boxplot of the differences in the percentage of each cell that is zero.

`MeanZeros` Scatter plot showing the difference from the reference percentage of zeros across expression ranks.

`QQPlots` Quantile-Quantile plots

`Means` Q-Q plot of the means.

`Variances` Q-Q plot of the variances.

`expandParams`

11

`LibrarySizes` Q-Q plot of the library sizes.
`ZerosGene` Q-Q plot of the percentage of zeros per gene.
`ZerosCell` Q-Q plot of the percentage of zeros per cell.

The plots returned by this function are created using `ggplot` and are only a sample of the kind of plots you might like to consider. The data used to create these plots is also returned and should be in the correct format to allow you to create further plots using `ggplot`.

Value

List containing the combined datasets and plots.

Examples

```
sim1 <- splatSimulate(nGenes = 1000, batchCells = 20)
sim2 <- simpleSimulate(nGenes = 1000, nCells = 20)
difference <- diffSCEs(list(Splat = sim1, Simple = sim2), ref = "Simple")
names(difference)
names(difference$Plots)
```

expandParams

Expand parameters

Description

Expand the parameters that can be vectors so that they are the same length as the number of groups.

Usage

```
expandParams(object, ...)
## S4 method for signature 'BASiCSPParams'
expandParams(object)

## S4 method for signature 'LunParams'
expandParams(object)

## S4 method for signature 'SplatParams'
expandParams(object)
```

Arguments

<code>object</code>	object to expand.
<code>...</code>	additional arguments.

Value

Expanded object.

12

getParam

getLNormFactors *Get log-normal factors*

Description

Randomly generate multiplication factors from a log-normal distribution.

Usage

```
getLNormFactors(n.facs, sel.prob, neg.prob, fac.loc, fac.scale)
```

Arguments

n.facs	Number of factors to generate.
sel.prob	Probability that a factor will be selected to be different from 1.
neg.prob	Probability that a selected factor is less than one.
fac.loc	Location parameter for the log-normal distribution.
fac.scale	Scale factor for the log-normal distribution.

Value

Vector containing generated factors.

getParam *Get a parameter*

Description

Accessor function for getting parameter values.

Usage

```
getParam(object, name)

## S4 method for signature 'Params'
getParam(object, name)
```

Arguments

object	object to get parameter from.
name	name of the parameter to get.

Value

The extracted parameter value

Examples

```
params <- newSimpleParams()
getParam(params, "nGenes")
```

`getParams`

13

`getParams`

Get parameters

Description

Get multiple parameter values from a Params object.

Usage

```
getParams(params, names)
```

Arguments

<code>params</code>	Params object to get values from.
<code>names</code>	vector of names of the parameters to get.

Value

List with the values of the selected parameters.

Examples

```
params <- newSimpleParams()  
getParams(params, c("nGenes", "nCells", "mean.rate"))
```

`getPathOrder`

Get path order

Description

Identify the correct order to process paths so that preceding paths have already been simulated.

Usage

```
getPathOrder(path.from)
```

Arguments

<code>path.from</code>	vector giving the path endpoints that each path originates from.
------------------------	--

Value

Vector giving the order to process paths in.

14

logistic

listSims*List simulations*

Description

List all the simulations that are currently available in Splatter with a brief description.

Usage

```
listSims(print = TRUE)
```

Arguments

print logical. Whether to print to the console.

Value

Invisibly returns a data.frame containing the information that is displayed.

Examples

```
listSims()
```

logistic*Logistic function*

Description

Implementation of the logistic function

Usage

```
logistic(x, x0, k)
```

Arguments

x value to apply the function to.
x0 midpoint parameter. Gives the centre of the function.
k shape parameter. Gives the slope of the function.

Value

Value of logistic function with given parameters

lun2Estimate

15

lun2Estimate	<i>Estimate Lun2 simulation parameters</i>
--------------	--

Description

Estimate simulation parameters for the Lun2 simulation from a real dataset.

Usage

```

lun2Estimate(counts, plates, params = newLun2Params(), min.size = 200,
             verbose = TRUE, BPPARAM = SerialParam())

## S3 method for class 'SingleCellExperiment'
lun2Estimate(counts, plates,
              params = newLun2Params(), min.size = 200, verbose = TRUE,
              BPPARAM = SerialParam())

## S3 method for class 'matrix'
lun2Estimate(counts, plates, params = newLun2Params(),
              min.size = 200, verbose = TRUE, BPPARAM = SerialParam())

```

Arguments

counts	either a counts matrix or a SingleCellExperiment object containing count data to estimate parameters from.
plates	integer vector giving the plate that each cell originated from.
params	Lun2Params object to store estimated values in.
min.size	minimum size of clusters when identifying group of cells in the data.
verbose	logical. Whether to show progress messages.
BPPARAM	A BiocParallelParam instance giving the parallel back-end to be used. Default is SerialParam which uses a single core.

Details

See [Lun2Params](#) for more details on the parameters.

Value

LunParams object containing the estimated parameters.

Examples

```

## Not run:
# Load example data
library(scater)
data("sc_example_counts")
data("sc_example_cell_info")

plates <- factor(sc_example_cell_info$Mutation_Status)
params <- lun2Estimate(sc_example_counts, plates, min.size = 20)
params

```

16

Lun2Params

```
## End(Not run)
```

*Lun2Params**The Lun2Params class*

Description

S4 class that holds parameters for the Lun2 simulation.

Parameters

The Lun2 simulation uses the following parameters:

nGenes The number of genes to simulate.

nCells The number of cells to simulate.

[seed] Seed to use for generating random numbers.

Gene parameters **gene.params** A `data.frame` containing gene parameters with two columns: Mean (mean expression for each gene) and Disp (dispersion for each gene).

zi.params A `data.frame` containing zero-inflated gene parameters with three columns: Mean (mean expression for each gene), Disp (dispersion for each, gene), and Prop (zero proportion for each gene).

[nPlates] The number of plates to simulate.

Plate parameters **plate.ingroup** Character vector giving the plates considered to be part of the "ingroup".

plate.mod Plate effect modifier factor. The plate effect variance is divided by this value.

plate.var Plate effect variance.

Cell parameters **cell.plates** Factor giving the plate that each cell comes from.

cell.libSizes Library size for each cell.

cell.libMod Modifier factor for library sizes. The library sizes are multiplied by this value.

Differential expression parameters **de.nGenes** Number of differentially expressed genes.

de.fc Fold change for differentially expressed genes.

The parameters not shown in brackets can be estimated from real data using [lun2Estimate](#). For details of the Lun2 simulation see [lun2Simulate](#).

lun2Simulate

17

lun2Simulate *Lun2 simulation*

Description

Simulate single-cell RNA-seq count data using the method described in Lun and Marioni "Overcoming confounding plate effects in differential expression analyses of single-cell RNA-seq data".

Usage

```
lun2Simulate(params = newLun2Params(), zinb = FALSE, verbose = TRUE,  
...)
```

Arguments

params	Lun2Params object containing simulation parameters.
zinb	logical. Whether to use a zero-inflated model.
verbose	logical. Whether to print progress messages
...	any additional parameter settings to override what is provided in params.

Details

The Lun2 simulation uses a negative-binomial distribution where the means and dispersions have been sampled from a real dataset (using [lun2Estimate](#)). The other core feature of the Lun2 simulation is the addition of plate effects. Differential expression can be added between two groups of plates (an "ingroup" and all other plates). Library size factors are also applied and optionally a zero-inflated negative-binomial can be used.

If the number of genes to simulate differs from the number of provided gene parameters or the number of cells to simulate differs from the number of library sizes the relevant parameters will be sampled with a warning. This allows any number of genes or cells to be simulated regardless of the number in the dataset used in the estimation step but has the downside that some genes or cells may be simulated multiple times.

Value

SingleCellExperiment containing simulated counts.

References

Lun ATL, Marioni JC. Overcoming confounding plate effects in differential expression analyses of single-cell RNA-seq data. *Biostatistics* (2017).

Paper: [dx.doi.org/10.1093/biostatistics/kxw055](https://doi.org/10.1093/biostatistics/kxw055)

Code: <https://github.com/MarioniLab/PlateEffects2016>

Examples

```
sim <- lun2Simulate()
```

18

lunEstimate

lunEstimate *Estimate Lun simulation parameters*

Description

Estimate simulation parameters for the Lun simulation from a real dataset.

Usage

```
lunEstimate(counts, params = newLunParams())

## S3 method for class 'SingleCellExperiment'
lunEstimate(counts,
            params = newLunParams())

## S3 method for class 'matrix'
lunEstimate(counts, params = newLunParams())
```

Arguments

counts	either a counts matrix or an SingleCellExperiment object containing count data to estimate parameters from.
params	LunParams object to store estimated values in.

Details

The nGenes and nCells parameters are taken from the size of the input data. No other parameters are estimated. See [LunParams](#) for more details on the parameters.

Value

LunParams object containing the estimated parameters.

Examples

```
# Load example data
library(scater)
data("sc_example_counts")

params <- lunEstimate(sc_example_counts)
params
```

LunParams

19

LunParams*The LunParams class***Description**

S4 class that holds parameters for the Lun simulation.

Parameters

The Lun simulation uses the following parameters:

nGenes The number of genes to simulate.

nCells The number of cells to simulate.

[nGroups] The number of groups to simulate.

[groupCells] Vector giving the number of cells in each simulation group/path.

[seed] Seed to use for generating random numbers.

Mean parameters **[mean.shape]** Shape parameter for the mean gamma distribution.

[mean.rate] Rate parameter for the mean gamma distribution.

Counts parameters **[count.disp]** The dispersion parameter for the counts negative binomial distribution.

Differential expression parameters **[de.nGenes]** The number of genes that are differentially expressed in each group

[de.upProp] The proportion of differentially expressed genes that are up-regulated in each group

[de.upFC] The fold change for up-regulated genes

[de.downFC] The fold change for down-regulated genes

The parameters not shown in brackets can be estimated from real data using [lunEstimate](#). For details of the Lun simulation see [lunSimulate](#).

lunSimulate*Lun simulation***Description**

Simulate single-cell RNA-seq count data using the method described in Lun, Bach and Marioni "Pooling across cells to normalize single-cell RNA sequencing data with many zero counts".

Usage

```
lunSimulate(params = newLunParams(), verbose = TRUE, ...)
```

Arguments

params LunParams object containing Lun simulation parameters.

verbose logical. Whether to print progress messages.

... any additional parameter settings to override what is provided in **params**.

20

*makeCompPanel***Details**

The Lun simulation generates gene mean expression levels from a gamma distribution with shape = mean.shape and rate = mean.rate. Counts are then simulated from a negative binomial distribution with mu = means and size = 1 / bcv.common. In addition each cell is given a size factor (2 ^ rnorm(nCells, mean = 0, sd = 0.5)) and differential expression can be simulated with fixed fold changes.

See [LunParams](#) for details of the parameters.

Value

SingleCellExperiment object containing the simulated counts and intermediate values.

References

Lun ATL, Bach K, Marioni JC. Pooling across cells to normalize single-cell RNA sequencing data with many zero counts. *Genome Biology* (2016).

Paper: [dx.doi.org/10.1186/s13059-016-0947-7](https://doi.org/10.1186/s13059-016-0947-7)

Code: <https://github.com/MarioniLab/Deconvolution2016>

Examples

```
sim <- lunSimulate()
```

makeCompPanel*Make comparison panel***Description**

Combine the plots from `compareSCEs` into a single panel.

Usage

```
makeCompPanel(comp, title = "Comparison", labels = c("Means",
  "Variance", "Mean-variance relationship", "Library size",
  "Zeros per gene", "Zeros per cell", "Mean-zeros relationship"))
```

Arguments

<code>comp</code>	list returned by <code>compareSCEs</code> .
<code>title</code>	title for the panel.
<code>labels</code>	vector of labels for each of the seven plots.

Value

Combined panel plot

`makeDiffPanel`

21

Examples

```
## Not run:  
sim1 <- splatSimulate(nGenes = 1000, batchCells = 20)  
sim2 <- simpleSimulate(nGenes = 1000, nCells = 20)  
comparison <- compareSCEs(list(Splat = sim1, Simple = sim2))  
panel <- makeCompPanel(comparison)  
  
## End(Not run)
```

`makeDiffPanel`

Make difference panel

Description

Combine the plots from `diffSCEs` into a single panel.

Usage

```
makeDiffPanel(diff, title = "Difference comparison",  
             labels = c("Means", "Variance", "Library size", "Zeros per cell",  
                      "Zeros per gene", "Mean-variance relationship",  
                      "Mean-zeros relationship"))
```

Arguments

<code>diff</code>	list returned by <code>diffSCEs</code> .
<code>title</code>	title for the panel.
<code>labels</code>	vector of labels for each of the seven sections.

Value

Combined panel plot

Examples

```
## Not run:  
sim1 <- splatSimulate(nGenes = 1000, batchCells = 20)  
sim2 <- simpleSimulate(nGenes = 1000, nCells = 20)  
difference <- diffSCEs(list(Splat = sim1, Simple = sim2), ref = "Simple")  
panel <- makeDiffPanel(difference)  
  
## End(Not run)
```

22

mfaEstimate

 makeOverallPanel *Make overall panel*

Description

Combine the plots from `compSCEs` and `diffSCEs` into a single panel.

Usage

```
makeOverallPanel(comp, diff, title = "Overall comparison",
  row.labels = c("Means", "Variance", "Mean-variance relationship",
  "Library size", "Zeros per cell", "Zeros per gene",
  "Mean-zeros relationship"))
```

Arguments

<code>comp</code>	list returned by <code>compareSCEs</code> .
<code>diff</code>	list returned by <code>diffSCEs</code> .
<code>title</code>	title for the panel.
<code>row.labels</code>	vector of labels for each of the seven rows.

Value

Combined panel plot

Examples

```
## Not run:
sim1 <- splatSimulate(nGenes = 1000, batchCells = 20)
sim2 <- simpleSimulate(nGenes = 1000, nCells = 20)
comparison <- compSCEs(list(Splat = sim1, Simple = sim2))
difference <- diffSCEs(list(Splat = sim1, Simple = sim2), ref = "Simple")
panel <- makeOverallPanel(comparison, difference)

## End(Not run)
```

 mfaEstimate *Estimate mfa simulation parameters*

Description

Estimate simulation parameters for the mfa simulation from a real dataset.

MFAParams

23

Usage

```
mfaEstimate(counts, params = newMFAParams())

## S3 method for class 'SingleCellExperiment'
mfaEstimate(counts,
            params = newMFAParams()

## S3 method for class 'matrix'
mfaEstimate(counts, params = newMFAParams())
```

Arguments

counts either a counts matrix or a SingleCellExperiment object containing count data to estimate parameters from.

params MFAParams object to store estimated values in.

Details

The nGenes and nCells parameters are taken from the size of the input data. The dropout lambda parameter is estimate using `empirical_lambda`. See **MFAParams** for more details on the parameters.

Value

MFAParams object containing the estimated parameters.

Examples

```
# Load example data
library(scater)
data("sc_example_counts")

params <- mfaEstimate(sc_example_counts)
params
```

MFAParams*The MFAParams class***Description**

S4 class that holds parameters for the mfa simulation.

Parameters

The mfa simulation uses the following parameters:

nGenes The number of genes to simulate.

nCells The number of cells to simulate.

[seed] Seed to use for generating random numbers.

[trans.prop] Proportion of genes that show transient expression. These genes are briefly up or down-regulated before returning to their initial state

24

mfaSimulate

[zero.neg] Logical. Whether to set negative expression values to zero. This will zero-inflate the data.

[dropout.present] Logical. Whether to simulate dropout.

dropout.lambda Lambda parameter for the exponential dropout function.

The parameters not shown in brackets can be estimated from real data using [mfaEstimate](#). See [create_synthetic](#) for more details about the parameters. For details of the Splatter implementation of the mfa simulation see [mfaSimulate](#).

*mfaSimulate**MFA simulation*

Description

Simulate a bifurcating pseudotime path using the mfa method.

Usage

```
mfaSimulate(params = newMFAParams(), verbose = TRUE, ...)
```

Arguments

params	MFAParams object containing simulation parameters.
verbose	Logical. Whether to print progress messages.
...	any additional parameter settings to override what is provided in params.

Details

This function is just a wrapper around [create_synthetic](#) that takes a [MFAParams](#), runs the simulation then converts the output from log-expression to counts and returns a [SingleCellExperiment](#) object. See [create_synthetic](#) and the mfa paper for more details about how the simulation works.

Value

SingleCellExperiment containing simulated counts

References

Campbell KR, Yau C. Probabilistic modeling of bifurcations in single-cell gene expression data using a Bayesian mixture of factor analyzers. Wellcome Open Research (2017).

Paper: [10.12688/wellcomeopenres.11087.1](https://doi.org/10.12688/wellcomeopenres.11087.1)

Code: <https://github.com/kieranrcampbell/mfa>

Examples

```
sim <- mfaSimulate()
```

`newParams`

25

`newParams`

New Params

Description

Create a new Params object. Functions exist for each of the different Params subtypes.

Usage

```
newBASICSParams(...)  
newLun2Params(...)  
newLunParams(...)  
newMFAParams(...)  
newPhenoParams(...)  
newSCDDParams(...)  
newSimpleParams(...)  
newSparseDCParams(...)  
newSplatParams(...)  
newZINBParams(...)
```

Arguments

... additional parameters passed to [setParams](#).

Value

New Params object.

Examples

```
params <- newSimpleParams()  
params <- newSimpleParams(nGenes = 200, nCells = 10)
```

26

phenoEstimate

Params*The Params virtual class*

Description

Virtual S4 class that all other Params classes inherit from.

Parameters

The Params class defines the following parameters:

`nGenes` The number of genes to simulate.

`nCells` The number of cells to simulate.

`[seed]` Seed to use for generating random numbers.

The parameters not shown in brackets can be estimated from real data.

phenoEstimate*Estimate PhenoPath simulation parameters*

Description

Estimate simulation parameters for the PhenoPath simulation from a real dataset.

Usage

```
phenoEstimate(counts, params = newPhenoParams())

## S3 method for class 'SingleCellExperiment'
phenoEstimate(counts,
              params = newPhenoParams()

## S3 method for class 'matrix'
phenoEstimate(counts, params = newPhenoParams())
```

Arguments

`counts` either a counts matrix or an SingleCellExperiment object containing count data to estimate parameters from.
`params` PhenoParams object to store estimated values in.

Details

The `nGenes` and `nCells` parameters are taken from the size of the input data. The total number of genes is evenly divided into the four types. See [PhenoParams](#) for more details on the parameters.

Value

PhenoParams object containing the estimated parameters.

PhenoParams

27

Examples

```
# Load example data
library(scater)
data("sc_example_counts")

params <- phenoEstimate(sc_example_counts)
params
```

*PhenoParams**The PhenoParams class***Description**

S4 class that holds parameters for the PhenoPath simulation.

Parameters

The PhenoPath simulation uses the following parameters:

`nGenes` The number of genes to simulate.

`nCells` The number of cells to simulate.

`[seed]` Seed to use for generating random numbers.

`[n.de]` Number of genes to simulate from the differential expression regime

`[n.pst]` Number of genes to simulate from the pseudotime regime

`[n.pst.beta]` Number of genes to simulate from the pseudotime + beta interactions regime

`[n.de.pst.beta]` Number of genes to simulate from the differential expression + pseudotime + interactions regime

The parameters not shown in brackets can be estimated from real data using [phenoEstimate](#). For details of the PhenoPath simulation see [phenoSimulate](#).

*phenoSimulate**PhenoPath simulation***Description**

Simulate counts from a pseudotime trajectory using the PhenoPath method.

Usage

```
phenoSimulate(params = newPhenoParams(), verbose = TRUE, ...)
```

Arguments

`params` PhenoParams object containing simulation parameters.

`verbose` logical. Whether to print progress messages

`...` any additional parameter settings to override what is provided in `params`.

28

*rbindMatched***Details**

This function is just a wrapper around [simulate_phenopath](#) that takes a `PhenoParams`, runs the simulation then converts the output from log-expression to counts and returns a `SingleCellExperiment` object. The original simulated log-expression values are returned in the LogExprs assay. See [simulate_phenopath](#) and the PhenoPath paper for more details about how the simulation works.

Value

SingleCellExperiment containing simulated counts

References

Campbell K, Yau C. Uncovering genomic trajectories with heterogeneous genetic and environmental backgrounds across single-cells and populations. bioRxiv (2017).

Paper: [10.1101/159913](https://doi.org/10.1101/159913)

Code: <https://github.com/kieranrcampbell/phenopath>

Examples

```
sim <- phenoSimulate()
```

rbindMatched*Bind rows (matched)*

Description

Bind the rows of two data frames, keeping only the columns that are common to both.

Usage

```
rbindMatched(df1, df2)
```

Arguments

df1	first data.frame to bind.
df2	second data.frame to bind.

Value

data.frame containing rows from df1 and df2 but only common columns.

scDDEstimate

29

scDDEstimate	<i>Estimate scDD simulation parameters</i>
--------------	--

Description

Estimate simulation parameters for the scDD simulation from a real dataset.

Usage

```
scDDEstimate(counts, params = newSCDDParams(), verbose = TRUE,
             BPPARAM = SerialParam(), ...)

## S3 method for class 'matrix'
scDDEstimate(counts, params = newSCDDParams(),
             verbose = TRUE, BPPARAM = SerialParam(), conditions, ...)

## S3 method for class 'SingleCellExperiment'
scDDEstimate(counts,
             params = newSCDDParams(), verbose = TRUE, BPPARAM = SerialParam(),
             condition = "condition", ...)

## Default S3 method:
scDDEstimate(counts, params = newSCDDParams(),
             verbose = TRUE, BPPARAM = SerialParam(), condition, ...)
```

Arguments

counts	either a counts matrix or a SingleCellExperiment object containing count data to estimate parameters from.
params	SCDDParams object to store estimated values in.
verbose	logical. Whether to show progress messages.
BPPARAM	A BiocParallelParam instance giving the parallel back-end to be used. Default is SerialParam which uses a single core.
...	further arguments passed to or from other methods.
conditions	Vector giving the condition that each cell belongs to. Conditions can be 1 or 2.
condition	String giving the column that represents biological group of interest.

Details

This function applies [preprocess](#) to the counts then uses [scDD](#) to estimate the numbers of each gene type to simulate. The output is then converted to a SCDDParams object. See [preprocess](#) and [scDD](#) for details.

Value

SCDDParams object containing the estimated parameters.

30

*SCDDParams***Examples**

```
## Not run:
# Load example data
library(scater)
data("sc_example_counts")

conditions <- sample(1:2, ncol(sc_example_counts), replace = TRUE)
params <- scDDEstimate(sc_example_counts, conditions = conditions)
params

## End(Not run)
```

SCDDParams*The SCDDParams class***Description**

S4 class that holds parameters for the scDD simulation.

Parameters

The SCDD simulation uses the following parameters:

nGenes The number of genes to simulate (not used).
nCells The number of cells to simulate in each condition.
[seed] Seed to use for generating random numbers.
SCdat [SingleCellExperiment](#) containing real data.
nDE Number of DE genes to simulate.
nDP Number of DP genes to simulate.
nDM Number of DM genes to simulate.
nDB Number of DB genes to simulate.
nEE Number of EE genes to simulate.
nEP Number of EP genes to simulate.
[sd.range] Interval for fold change standard deviations.
[modeFC] Values for DP, DM and DB mode fold changes.
[varInflation] Variance inflation factors for each condition. If all equal to 1 will be set to NULL (default).
[condition] String giving the column that represents biological group of interest.

The parameters not shown in brackets can be estimated from real data using [scDDEstimate](#). See [simulateSet](#) for more details about the parameters. For details of the Splatter implementation of the scDD simulation see [scDDSimulate](#).

`scDDSimulate`

31

`scDDSimulate` *scDD simulation*

Description

Simulate counts using the scDD method.

Usage

```
scDDSimulate(params = newSCDDParams(), plots = FALSE,  
plot.file = NULL, verbose = TRUE, BPPARAM = SerialParam(), ...)
```

Arguments

<code>params</code>	SCDDParams object containing simulation parameters.
<code>plots</code>	logical. whether to generate scDD fold change and validation plots.
<code>plot.file</code>	File path to save plots as PDF.
<code>verbose</code>	logical. Whether to print progress messages
<code>BPPARAM</code>	A <code>BiocParallelParam</code> instance giving the parallel back-end to be used. Default is <code>SerialParam</code> which uses a single core.
...	any additional parameter settings to override what is provided in <code>params</code> .

Details

This function is just a wrapper around `simulateSet` that takes a `SCDDParams`, runs the simulation then converts the output to a `SingleCellExperiment` object. See `simulateSet` for more details about how the simulation works.

Value

`SingleCellExperiment` containing simulated counts

References

Korthauer KD, Chu L-F, Newton MA, Li Y, Thomson J, Stewart R, et al. A statistical approach for identifying differential distributions in single-cell RNA-seq experiments. *Genome Biology* (2016).

Paper: [10.1186/s13059-016-1077-y](https://doi.org/10.1186/s13059-016-1077-y)

Code: <https://github.com/kdkorthauer/scDD>

Examples

```
## Not run:  
sim <- scDDSimulate()  
  
## End(Not run)
```

32

setParam

setParam*Set a parameter*

Description

Function for setting parameter values.

Usage

```
setParam(object, name, value)

## S4 method for signature 'BASiCSPParams'
setParam(object, name, value)

## S4 method for signature 'Lun2Params'
setParam(object, name, value)

## S4 method for signature 'LunParams'
setParam(object, name, value)

## S4 method for signature 'Params'
setParam(object, name, value)

## S4 method for signature 'PhenoParams'
setParam(object, name, value)

## S4 method for signature 'SCDDParams'
setParam(object, name, value)

## S4 method for signature 'SplatParams'
setParam(object, name, value)

## S4 method for signature 'ZINBParams'
setParam(object, name, value)
```

Arguments

object	object to set parameter in.
name	name of the parameter to set.
value	value to set the parameter to.

Value

Object with new parameter value.

Examples

```
params <- newSimpleParams()
setParam(params, "nGenes", 100)
```

`setParams`

33

`setParams`

Set parameters

Description

Set multiple parameters in a Params object.

Usage

```
setParams(object, update = NULL, ...)

## S4 method for signature 'Params'
setParams(object, update = NULL, ...)

## S4 method for signature 'SplatParams'
setParams(object, update = NULL, ...)
```

Arguments

<code>object</code>	Params object to set parameters in.
<code>update</code>	list of parameters to set where <code>names(update)</code> are the names of the parameters to set and the items in the list are values.
<code>...</code>	additional parameters to set. These are combined with any parameters specified in <code>update</code> .

Details

Each parameter is set by a call to `setParam`. If the same parameter is specified multiple times it will be set multiple times. Parameters can be specified using a list via `update` (useful when collecting parameter values in some way) or individually (useful when setting them manually), see examples.

Value

Params object with updated values.

Examples

```
params <- newSimpleParams()
params
# Set individually
params <- setParams(params, nGenes = 1000, nCells = 50)
params
# Set via update list
params <- setParams(params, list(mean.rate = 0.2, mean.shape = 0.8))
params
```

34

`setParamUnchecked`

`setParamsUnchecked` *Set parameters UNCHECKED*

Description

Set multiple parameters in a Params object.

Usage

```
setParamsUnchecked(params, update = NULL, ...)
```

Arguments

<code>params</code>	Params object to set parameters in.
<code>update</code>	list of parameters to set where <code>names(update)</code> are the names of the parameters to set and the items in the list are values.
<code>...</code>	additional parameters to set. These are combined with any parameters specified in <code>update</code> .

Details

Each parameter is set by a call to [setParam](#). If the same parameter is specified multiple times it will be set multiple times. Parameters can be specified using a list via `update` (useful when collecting parameter values in some way) or individually (useful when setting them manually), see examples.
THE FINAL OBJECT IS NOT CHECKED FOR VALIDITY!

Value

Params object with updated values.

`setParamUnchecked` *Set a parameter UNCHECKED*

Description

Function for setting parameter values. THE OUTPUT IS NOT CHECKED FOR VALIDITY!

Usage

```
setParamUnchecked(object, name, value)  
  
## S4 method for signature 'Params'  
setParamUnchecked(object, name, value)
```

Arguments

<code>object</code>	object to set parameter in.
<code>name</code>	name of the parameter to set.
<code>value</code>	value to set the parameter to.

`showDFs`

35

Value

Object with new parameter value.

`showDFs`

Show data.frame

Description

Function used for pretty printing data.frame parameters.

Usage

`showDFs(dfs, not.default)`

Arguments

`dfs` list of data.frames to show.

`not.default` logical vector giving which have changed from the default.

`showPP`

Show pretty print

Description

Function used for pretty printing params object.

Usage

`showPP(params, pp)`

Arguments

`params` object to show.

`pp` list specifying how the object should be displayed.

Value

Print params object to console

36

simpleEstimate

showValues

*Show values***Description**

Function used for pretty printing scale or vector parameters.

Usage

```
showValues(values, not.default)
```

Arguments

values	list of values to show.
not.default	logical vector giving which have changed from the default.

*simpleEstimate**Estimate simple simulation parameters***Description**

Estimate simulation parameters for the simple simulation from a real dataset.

Usage

```
simpleEstimate(counts, params = newSimpleParams())

## S3 method for class 'SingleCellExperiment'
simpleEstimate(counts,
               params = newSimpleParams()

## S3 method for class 'matrix'
simpleEstimate(counts, params = newSimpleParams())
```

Arguments

counts	either a counts matrix or a SingleCellExperiment object containing count data to estimate parameters from.
params	SimpleParams object to store estimated values in.

Details

The nGenes and nCells parameters are taken from the size of the input data. The mean parameters are estimated by fitting a gamma distribution to the library size normalised mean expression level using `fitdist`. See [SimpleParams](#) for more details on the parameters.

Value

SimpleParams object containing the estimated parameters.

SimpleParams

37

Examples

```
# Load example data
library(scater)
data("sc_example_counts")

params <- simpleEstimate(sc_example_counts)
params
```

SimpleParams*The SimpleParams class***Description**

S4 class that holds parameters for the simple simulation.

Parameters

The simple simulation uses the following parameters:

nGenes The number of genes to simulate.
nCells The number of cells to simulate.
[seed] Seed to use for generating random numbers.
mean.shape The shape parameter for the mean gamma distribution.
mean.rate The rate parameter for the mean gamma distribution.
[count.disp] The dispersion parameter for the counts negative binomial distribution.

The parameters not shown in brackets can be estimated from real data using [simpleEstimate](#). For details of the simple simulation see [simpleSimulate](#).

simpleSimulate*Simple simulation***Description**

Simulate counts from a simple negative binomial distribution without simulated library sizes, differential expression etc.

Usage

```
simpleSimulate(params = newSimpleParams(), verbose = TRUE, ...)
```

Arguments

params	SimpleParams object containing simulation parameters.
verbose	logical. Whether to print progress messages
...	any additional parameter settings to override what is provided in params.

38

*sparseDCEstimate***Details**

Gene means are simulated from a gamma distribution with `shape = mean.shape` and `rate = mean.rate`. Counts are then simulated from a negative binomial distribution with `mu = means` and `size = 1 / counts.disp`. See [SimpleParams](#) for more details of the parameters.

Value

`SingleCellExperiment` containing simulated counts

Examples

```
sim <- simpleSimulate()
# Override default parameters
sim <- simpleSimulate(nGenes = 1000, nCells = 50)
```

sparseDCEstimate *Estimate SparseDC simulation parameters*

Description

Estimate simulation parameters for the SparseDC simulation from a real dataset.

Usage

```
sparseDCEstimate(counts, conditions, nclusters, norm = TRUE,
                  params = newSparseDCParams())

## S3 method for class 'SingleCellExperiment'
sparseDCEstimate(counts, conditions,
                  nclusters, norm = TRUE, params = newSparseDCParams())

## S3 method for class 'matrix'
sparseDCEstimate(counts, conditions, nclusters,
                  norm = TRUE, params = newSparseDCParams())
```

Arguments

<code>counts</code>	either a counts matrix or an <code>SingleCellExperiment</code> object containing count data to estimate parameters from.
<code>conditions</code>	numeric vector giving the condition each cell belongs to.
<code>nclusters</code>	number of cluster present in the dataset.
<code>norm</code>	logical, whether to library size normalise counts before estimation. Set this to FALSE if counts is already normalised.
<code>params</code>	<code>PhenoParams</code> object to store estimated values in.

Details

The `nGenes` and `nCells` parameters are taken from the size of the input data. The counts are preprocessed using `pre_proc_data` and then parameters are estimated using `sparsedc_cluster` using lambda values calculated using `lambda1_calculator` and `lambda2_calculator`.

See [SparseDCParams](#) for more details on the parameters.

SparseDCParams

39

Value

SparseParams object containing the estimated parameters.

Examples

```
# Load example data
library(scater)
data("sc_example_counts")

set.seed(1)
conditions <- sample(1:2, ncol(sc_example_counts), replace = TRUE)

params <- sparseDCEstimate(sc_example_counts[1:500, ], conditions,
                           nclusters = 3)
params
```

*SparseDCParams**The SparseDCParams class***Description**

S4 class that holds parameters for the SparseDC simulation.

Parameters

The SparseDC simulation uses the following parameters:

nGenes The number of genes to simulate in each condition.

nCells The number of cells to simulate.

[seed] Seed to use for generating random numbers.

markers.n Number of marker genes to simulate for each cluster.

markers.shared Number of marker genes for each cluster shared between conditions. Must be less than or equal to **markers.n**.

[markers.same] Logical. Whether each cluster should have the same set of marker genes.

clusts.c1 Numeric vector of clusters present in condition 1. The number of times a cluster is repeated controls the proportion of cells from that cluster.

clusts.c2 Numeric vector of clusters present in condition 2. The number of times a cluster is repeated controls the proportion of cells from that cluster.

[mean.lower] Lower bound for cluster gene means.

[mean.upper] Upper bound for cluster gene means.

The parameters not shown in brackets can be estimated from real data using [sparseDCEstimate](#). For details of the SparseDC simulation see [sparseDCSimulate](#).

40

sparseDCSimulate

sparseDCSimulate *SparseDC simulation*

Description

Simulate counts from cluster in two conditions using the SparseDC method.

Usage

```
sparseDCSimulate(params = newSparseDCParams(), verbose = TRUE, ...)
```

Arguments

params	SparseDCParams object containing simulation parameters.
verbose	logical. Whether to print progress messages
...	any additional parameter settings to override what is provided in params.

Details

This function is just a wrapper around `sim_data` that takes a `SparseDCParams`, runs the simulation then converts the output from log-expression to counts and returns a `SingleCellExperiment` object. The original simulated log-expression values are returned in the LogExprs assay. See `sim_data` and the SparseDC paper for more details about how the simulation works.

Value

SingleCellExperiment containing simulated counts

References

Campbell K, Yau C. Uncovering genomic trajectories with heterogeneous genetic and environmental backgrounds across single-cells and populations. bioRxiv (2017).

Barron M, Zhang S, Li J. A sparse differential clustering algorithm for tracing cell type changes via single-cell RNA-sequencing data. Nucleic Acids Research (2017).

Paper: [10.1093/nar/gkx1113](https://doi.org/10.1093/nar/gkx1113)

Examples

```
sim <- sparseDCSimulate()
```

`splatEstBCV`

41

`splatEstBCV`*Estimate Splat Biological Coefficient of Variation parameters*

Description

Parameters are estimated using the `estimateDisp` function in the `edgeR` package.

Usage

```
splatEstBCV(counts, params)
```

Arguments

<code>counts</code>	counts matrix to estimate parameters from.
<code>params</code>	SplatParams object to store estimated values in.

Details

The `estimateDisp` function is used to estimate the common dispersion and prior degrees of freedom. See `estimateDisp` for details. When estimating parameters on simulated data we found a broadly linear relationship between the true underlying common dispersion and the `edgR` estimate, therefore we apply a small correction, $\text{disp} = 0.1 + 0.25 * \text{edgeR}.disp$.

Value

SplatParams object with estimated values.

`splatEstDropout`*Estimate Splat dropout parameters*

Description

Estimate the midpoint and shape parameters for the logistic function used when simulating dropout.

Usage

```
splatEstDropout(norm.counts, params)
```

Arguments

<code>norm.counts</code>	library size normalised counts matrix.
<code>params</code>	SplatParams object to store estimated values in.

Details

Logistic function parameters are estimated by fitting a logistic function to the relationship between log2 mean gene expression and the proportion of zeros in each gene. See `nls` for details of fitting. Note this is done on the experiment level, more granular (eg. group or cell) level dropout is not estimated.

42

*splatEstimate***Value**

SplatParams object with estimated values.

<i>splatEstimate</i>	<i>Estimate Splat simulation parameters</i>
----------------------	---

Description

Estimate simulation parameters for the Splat simulation from a real dataset. See the individual estimation functions for more details on how this is done.

Usage

```
splatEstimate(counts, params = newSplatParams())

## S3 method for class 'SingleCellExperiment'
splatEstimate(counts,
              params = newSplatParams())

## S3 method for class 'matrix'
splatEstimate(counts, params = newSplatParams())
```

Arguments

counts either a counts matrix or a SingleCellExperiment object containing count data to estimate parameters from.
params SplatParams object to store estimated values in.

Value

SplatParams object containing the estimated parameters.

See Also

[splatEstMean](#), [splatEstLib](#), [splatEstOutlier](#), [splatEstBCV](#), [splatEstDropout](#)

Examples

```
# Load example data
library(scater)
data("sc_example_counts")

params <- splatEstimate(sc_example_counts)
params
```

splatEstLib

43

splatEstLib *Estimate Splat library size parameters*

Description

The Shapiro-Wilks test is used to determine if the library sizes are normally distributed. If so a normal distribution is fitted to the library sizes, if not (most cases) a log-normal distribution is fitted and the estimated parameters are added to the params object. See [fitdist](#) for details on the fitting.

Usage

```
splatEstLib(counts, params)
```

Arguments

counts	counts matrix to estimate parameters from.
params	splatParams object to store estimated values in.

Value

splatParams object with estimated values.

splatEstMean *Estimate Splat mean parameters*

Description

Estimate rate and shape parameters for the gamma distribution used to simulate gene expression means.

Usage

```
splatEstMean(norm.counts, params)
```

Arguments

norm.counts	library size normalised counts matrix.
params	SplatParams object to store estimated values in.

Details

Parameter for the gamma distribution are estimated by fitting the mean normalised counts using [fitdist](#). The 'maximum goodness-of-fit estimation' method is used to minimise the Cramer-von Mises distance. This can fail in some situations, in which case the 'method of moments estimation' method is used instead. Prior to fitting the means are winsorized by setting the top and bottom 10 percent of values to the 10th and 90th percentiles.

Value

SplatParams object with estimated values.

44

SplatParams

splatEstOutlier	<i>Estimate Splat expression outlier parameters</i>
-----------------	---

Description

Parameters are estimated by comparing means of individual genes to the median mean expression level.

Usage

```
splatEstOutlier(norm.counts, params)
```

Arguments

norm.counts	library size normalised counts matrix.
params	SplatParams object to store estimated values in.

Details

Expression outlier genes are detected using the Median Absolute Deviation (MAD) from median method. If the log2 mean expression of a gene is greater than two MADs above the median log2 mean expression it is designated as an outlier. The proportion of outlier genes is used to estimate the outlier probability. Factors for each outlier gene are calculated by dividing mean expression by the median mean expression. A log-normal distribution is then fitted to these factors in order to estimate the outlier factor location and scale parameters using [fitdist](#).

Value

SplatParams object with estimated values.

SplatParams	<i>The SplatParams class</i>
-------------	------------------------------

Description

S4 class that holds parameters for the Splatter simulation.

Parameters

The Splatter simulation requires the following parameters:

`nGenes` The number of genes to simulate.

`nCells` The number of cells to simulate.

`[seed]` Seed to use for generating random numbers.

Batch parameters `[nBatches]` The number of batches to simulate.

`[batchCells]` Vector giving the number of cells in each batch.

`[batch.facLoc]` Location (meanlog) parameter for the batch effect factor log-normal distribution. Can be a vector.

SplatParams

45

[batch.facScale] Scale (sdlog) parameter for the batch effect factor log-normal distribution. Can be a vector.

Mean parameters mean.shape Shape parameter for the mean gamma distribution.

mean.rate Rate parameter for the mean gamma distribution.

Library size parameters lib.loc Location (meanlog) parameter for the library size log-normal distribution, or mean parameter if a normal distribution is used.

lib.scale Scale (sdlog) parameter for the library size log-normal distribution, or sd parameter if a normal distribution is used.

lib.norm Logical. Whether to use a normal distribution for library sizes instead of a log-normal.

Expression outlier parameters out.prob Probability that a gene is an expression outlier.

out.facLoc Location (meanlog) parameter for the expression outlier factor log-normal distribution.

out.facScale Scale (sdlog) parameter for the expression outlier factor log-normal distribution.

Group parameters [nGroups] The number of groups or paths to simulate.

[group.prob] Probability that a cell comes from a group.

Differential expression parameters [de.prob] Probability that a gene is differentially expressed in a group. Can be a vector.

[de.downProb] Probability that a differentially expressed gene is down-regulated. Can be a vector.

[de.facLoc] Location (meanlog) parameter for the differential expression factor log-normal distribution. Can be a vector.

[de.facScale] Scale (sdlog) parameter for the differential expression factor log-normal distribution. Can be a vector.

Biological Coefficient of Variation parameters bcv.common Underlying common dispersion across all genes.

bcv.df Degrees of Freedom for the BCV inverse chi-squared distribution.

Dropout parameters dropout.type The type of dropout to simulate. "none" indicates no dropout, "experiment" is global dropout using the same parameters for every cell, "batch" uses the same parameters for every cell in each batch, "group" uses the same parameters for every cell in each group and "cell" uses a different set of parameters for each cell.

dropout.mid Midpoint parameter for the dropout logistic function.

dropout.shape Shape parameter for the dropout logistic function.

Differentiation path parameters [path.from] Vector giving the originating point of each path.

This allows path structure such as a cell type which differentiates into an intermediate cell type that then differentiates into two mature cell types. A path structure of this form would have a "from" parameter of c(0, 1, 1) (where 0 is the origin). If no vector is given all paths will start at the origin.

[path.nSteps] Vector giving the number of steps to simulate along each path. If a single value is given it will be applied to all paths. This parameter was previously called path.length.

[path.skew] Vector giving the skew of each path. Values closer to 1 will give more cells towards the starting population, values closer to 0 will give more cells towards the final population. If a single value is given it will be applied to all paths.

[path.nonlinearProb] Probability that a gene follows a non-linear path along the differentiation path. This allows more complex gene patterns such as a gene being equally expressed at the beginning and end of a path but lowly expressed in the middle.

46

splatSimBatchEffects

[`path.sigmaFac`] Sigma factor for non-linear gene paths. A higher value will result in more extreme non-linear variations along a path.

The parameters not shown in brackets can be estimated from real data using [splatEstimate](#). For details of the Splatter simulation see [splatSimulate](#).

*splatSimBatchCellMeans**Simulate batch means***Description**

Simulate a mean for each gene in each cell incorporating batch effect factors.

Usage

```
splatSimBatchCellMeans(sim, params)
```

Arguments

`sim` SingleCellExperiment to add batch means to.
`params` SplatParams object with simulation parameters.

Value

SingleCellExperiment with simulated batch means.

splatSimBatchEffects *Simulate batch effects***Description**

Simulate batch effects. Batch effect factors for each batch are produced using [getLNormFactors](#) and these are added along with updated means for each batch.

Usage

```
splatSimBatchEffects(sim, params)
```

Arguments

`sim` SingleCellExperiment to add batch effects to.
`params` SplatParams object with simulation parameters.

Value

SingleCellExperiment with simulated batch effects.

`splatSimBCVMeans`

47

`splatSimBCVMeans` *Simulate BCV means*

Description

Simulate means for each gene in each cell that are adjusted to follow a mean-variance trend using Biological Coefficient of Variation taken from and inverse gamma distribution.

Usage

```
splatSimBCVMeans(sim, params)
```

Arguments

<code>sim</code>	SingleCellExperiment to add BCV means to.
<code>params</code>	SplatParams object with simulation parameters.

Value

SingleCellExperiment with simulated BCV means.

`splatSimCellMeans` *Simulate cell means*

Description

Simulate a gene by cell matrix giving the mean expression for each gene in each cell. Cells start with the mean expression for the group they belong to (when simulating groups) or cells are assigned the mean expression from a random position on the appropriate path (when simulating paths). The selected means are adjusted for each cell's expected library size.

Usage

```
splatSimSingleCellMeans(sim, params)
```

```
splatSimGroupCellMeans(sim, params)
```

```
splatSimPathCellMeans(sim, params)
```

Arguments

<code>sim</code>	SingleCellExperiment to add cell means to.
<code>params</code>	SplatParams object with simulation parameters.

Value

SingleCellExperiment with added cell means.

48

splatSimDropout

splatSimDE*Simulate group differential expression*

Description

Simulate differential expression. Differential expression factors for each group are produced using `getLNormFactors` and these are added along with updated means for each group. For paths care is taken to make sure they are simulated in the correct order.

Usage

```
splatSimGroupDE(sim, params)  
splatSimPathDE(sim, params)
```

Arguments

`sim` SingleCellExperiment to add differential expression to.
`params` splatParams object with simulation parameters.

Value

SingleCellExperiment with simulated differential expression.

splatSimDropout*Simulate dropout*

Description

A logistic function is used to form a relationship between the expression level of a gene and the probability of dropout, giving a probability for each gene in each cell. These probabilities are used in a Bernoulli distribution to decide which counts should be dropped.

Usage

```
splatSimDropout(sim, params)
```

Arguments

`sim` SingleCellExperiment to add dropout to.
`params` SplatParams object with simulation parameters.

Value

SingleCellExperiment with simulated dropout and observed counts.

`splatSimGeneMeans`

49

`splatSimGeneMeans` *Simulate gene means*

Description

Simulate gene means from a gamma distribution. Also simulates outlier expression factors. Genes with an outlier factor not equal to 1 are replaced with the median mean expression multiplied by the outlier factor.

Usage

```
splatSimGeneMeans(sim, params)
```

Arguments

<code>sim</code>	SingleCellExperiment to add gene means to.
<code>params</code>	SplatParams object with simulation parameters.

Value

SingleCellExperiment with simulated gene means.

`splatSimLibSizes` *Simulate library sizes*

Description

Simulate expected library sizes. Typically a log-normal distribution is used but there is also the option to use a normal distribution. In this case any negative values are set to half the minimum non-zero value.

Usage

```
splatSimLibSizes(sim, params)
```

Arguments

<code>sim</code>	SingleCellExperiment to add library size to.
<code>params</code>	SplatParams object with simulation parameters.

Value

SingleCellExperiment with simulated library sizes.

50

splatSimulate

splatSimTrueCounts	<i>Simulate true counts</i>
--------------------	-----------------------------

Description

Simulate a true counts matrix. Counts are simulated from a poisson distribution where Each gene in each cell has it's own mean based on the group (or path position), expected library size and BCV.

Usage

```
splatSimTrueCounts(sim, params)
```

Arguments

sim	SingleCellExperiment to add true counts to.
params	SplatParams object with simulation parameters.

Value

SingleCellExperiment with simulated true counts.

splatSimulate	<i>Splat simulation</i>
---------------	-------------------------

Description

Simulate count data from a fictional single-cell RNA-seq experiment using the Splat method.

Usage

```
splatSimulate(params = newSplatParams(), method = c("single", "groups",
  "paths"), verbose = TRUE, ...)

splatSimulateSingle(params = newSplatParams(), verbose = TRUE, ...)

splatSimulateGroups(params = newSplatParams(), verbose = TRUE, ...)

splatSimulatePaths(params = newSplatParams(), verbose = TRUE, ...)
```

Arguments

params	SplatParams object containing parameters for the simulation. See SplatParams for details.
method	which simulation method to use. Options are "single" which produces a single population, "groups" which produces distinct groups (eg. cell types) or "paths" which selects cells from continuous trajectories (eg. differentiation processes).
verbose	logical. Whether to print progress messages.
...	any additional parameter settings to override what is provided in params.

splatSimulate

51

Details

Parameters can be set in a variety of ways. If no parameters are provided the default parameters are used. Any parameters in `params` can be overridden by supplying additional arguments through a call to `setParams`. This design allows the user flexibility in how they supply parameters and allows small adjustments without creating a new `SplatParams` object. See examples for a demonstration of how this can be used.

The simulation involves the following steps:

1. Set up simulation object
2. Simulate library sizes
3. Simulate gene means
4. Simulate groups/paths
5. Simulate BCV adjusted cell means
6. Simulate true counts
7. Simulate dropout
8. Create final dataset

The final output is a `SingleCellExperiment` object that contains the simulated counts but also the values for various intermediate steps. These are stored in the `colData` (for cell specific information), `rowData` (for gene specific information) or `assays` (for gene by cell matrices) slots. This additional information includes:

`colData Cell` Unique cell identifier.

Group The group or path the cell belongs to.

ExpLibSize The expected library size for that cell.

Step (paths only) how far along the path each cell is.

`rowData Gene` Unique gene identifier.

BaseGeneMean The base expression level for that gene.

OutlierFactor Expression outlier factor for that gene. Values of 1 indicate the gene is not an expression outlier.

GeneMean Expression level after applying outlier factors.

BatchFac[Batch] The batch effects factor for each gene for a particular batch.

DEFac[Group] The differential expression factor for each gene in a particular group. Values of 1 indicate the gene is not differentially expressed.

SigmaFac[Path] Factor applied to genes that have non-linear changes in expression along a path.

`assays BatchCellMeans` The mean expression of genes in each cell after adding batch effects.

BaseCellMeans The mean expression of genes in each cell after any differential expression and adjusted for expected library size.

BCV The Biological Coefficient of Variation for each gene in each cell.

CellMeans The mean expression level of genes in each cell adjusted for BCV.

TrueCounts The simulated counts before dropout.

Dropout Logical matrix showing which values have been dropped in which cells.

Values that have been added by Splatter are named using UpperCamelCase in order to differentiate them from the values added by analysis packages which typically use underscore_naming.

52

*splatter***Value**

SingleCellExperiment object containing the simulated counts and intermediate values.

References

Zappia L, Phipson B, Oshlack A. Splatter: simulation of single-cell RNA sequencing data. *Genome Biology* (2017).

Paper: [10.1186/s13059-017-1305-0](https://doi.org/10.1186/s13059-017-1305-0)

Code: <https://github.com/Oshlack/splatter>

See Also

[splatSimLibSizes](#), [splatSimGeneMeans](#), [splatSimBatchEffects](#), [splatSimBatchCellMeans](#),
[splatSimDE](#), [splatSimCellMeans](#), [splatSimBCVMeans](#), [splatSimTrueCounts](#), [splatSimDropout](#)

Examples

```
# Simulation with default parameters
sim <- splatSimulate()
## Not run:
# Simulation with different number of genes
sim <- splatSimulate(nGenes = 1000)
# Simulation with custom parameters
params <- newSplatParams(nGenes = 100, mean.rate = 0.5)
sim <- splatSimulate(params)
# Simulation with adjusted custom parameters
sim <- splatSimulate(params, mean.rate = 0.6, out.prob = 0.2)
# Simulate groups
sim <- splatSimulate(method = "groups")
# Simulate paths
sim <- splatSimulate(method = "paths")

## End(Not run)
```

splatter

splatter.**Description**

splatter is a package for the well-documented and reproducible simulation of single-cell RNA-seq count data.

Details

As well as it's own simulation model **splatter** provides functions for the estimation of model parameters.

See Also

Zappia L, Phipson B, Oshlack A. Splatter: Simulation Of Single-Cell RNA Sequencing Data. bioRxiv. 2017; doi:10.1101/133173

<i>summariseDiff</i>	53
----------------------	----

summariseDiff	<i>Summarise diffSCEs</i>
----------------------	---------------------------

Description

Summarise the results of [diffSCEs](#). Calculates the Median Absolute Deviation (MAD), Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) for the various properties and ranks them.

Usage

```
summariseDiff(diff)
```

Arguments

diff	Output from diffSCEs
-------------	--------------------------------------

Value

data.frame with MADs, MAEs, RMSEs, scaled statistics and ranks

Examples

```
sim1 <- splatSimulate(nGenes = 1000, batchCells = 20)
sim2 <- simpleSimulate(nGenes = 1000, nCells = 20)
difference <- diffSCEs(list(Splat = sim1, Simple = sim2), ref = "Simple")
summary <- summariseDiff(difference)
head(summary)
```

winsorize	<i>Winsorize vector</i>
------------------	-------------------------

Description

Set outliers in a numeric vector to a specified percentile.

Usage

```
winsorize(x, q)
```

Arguments

x	Numeric vector to winsorize
q	Percentile to set from each end

Value

Winsorized numeric vector

54

`zinbEstimate`

<code>zinbEstimate</code>	<i>Estimate ZINB-WaVE simulation parameters</i>
---------------------------	---

Description

Estimate simulation parameters for the ZINB-WaVE simulation from a real dataset.

Usage

```
zinbEstimate(counts, design.samples = NULL, design.genes = NULL,
  common.disp = TRUE, iter.init = 2, iter.opt = 25,
  stop.opt = 1e-04, params = newZINBParams(), verbose = TRUE,
  BPPARAM = SerialParam(), ...)

## S3 method for class 'SingleCellExperiment'
zinbEstimate(counts,
  design.samples = NULL, design.genes = NULL, common.disp = TRUE,
  iter.init = 2, iter.opt = 25, stop.opt = 1e-04,
  params = newZINBParams(), verbose = TRUE, BPPARAM = SerialParam(),
  ...)

## S3 method for class 'matrix'
zinbEstimate(counts, design.samples = NULL,
  design.genes = NULL, common.disp = TRUE, iter.init = 2,
  iter.opt = 25, stop.opt = 1e-04, params = newZINBParams(),
  verbose = TRUE, BPPARAM = SerialParam(), ...)
```

Arguments

<code>counts</code>	either a counts matrix or a SingleCellExperiment object containing count data to estimate parameters from.
<code>design.samples</code>	design matrix of sample-level covariates.
<code>design.genes</code>	design matrix of gene-level covariates.
<code>common.disp</code>	logical. Whether or not a single dispersion for all features is estimated.
<code>iter.init</code>	number of iterations to use for initialization.
<code>iter.opt</code>	number of iterations to use for optimization.
<code>stop.opt</code>	stopping criterion for optimization.
<code>params</code>	ZINBParams object to store estimated values in.
<code>verbose</code>	logical. Whether to print progress messages.
<code>BPPARAM</code>	A BiocParallelParam instance giving the parallel back-end to be used. Default is SerialParam which uses a single core.
<code>...</code>	additional arguments passes to zinbFit .

Details

The function is a wrapper around [zinbFit](#) that takes the fitted model and inserts it into a [ZINBParams](#) object. See [ZINBParams](#) for more details on the parameters and [zinbFit](#) for details of the estimation procedure.

ZINBParams

55

Value

ZINBParams object containing the estimated parameters.

Examples

```
## Not run:  
# Load example data  
library(scater)  
data("sc_example_counts")  
  
params <- zinbEstimate(sc_example_counts)  
params  
  
## End(Not run)
```

ZINBParams*The ZINBParams class*

Description

S4 class that holds parameters for the ZINB-WaVE simulation.

Parameters

The ZINB-WaVE simulation uses the following parameters:

nGenes The number of genes to simulate.
nCells The number of cells to simulate.
[seed] Seed to use for generating random numbers.
model Object describing a ZINB model.

The majority of the parameters for this simulation are stored in a [ZinbModel](#) object. Please refer to the documentation for this class and its constructor([zinbModel](#)) for details about all the parameters.

The parameters not shown in brackets can be estimated from real data using [zinbEstimate](#). For details of the ZINB-WaVE simulation see [zinbSimulate](#).

zinbSimulate*ZINB-WaVE simulation*

Description

Simulate counts using the ZINB-WaVE method.

Usage

```
zinbSimulate(params = newZINBParams(), verbose = TRUE, ...)
```

56

*zinbSimulate***Arguments**

params	ZINBParams object containing simulation parameters.
verbose	logical. Whether to print progress messages
...	any additional parameter settings to override what is provided in params.

Details

This function is just a wrapper around `zinbSim` that takes a `ZINBParams`, runs the simulation then converts the output to a `SingleCellExperiment` object. See `zinbSim` and the ZINB-WaVE paper for more details about how the simulation works.

Value

`SingleCellExperiment` containing simulated counts

References

Campbell K, Yau C. Uncovering genomic trajectories with heterogeneous genetic and environmental backgrounds across single-cells and populations. bioRxiv (2017).

Risso D, Perraudeau F, Gribkova S, Dudoit S, Vert J-P. ZINB-WaVE: A general and flexible method for signal extraction from single-cell RNA-seq data bioRxiv (2017).

Paper: [10.1101/125112](https://doi.org/10.1101/125112)

Code: <https://github.com/drisso/zinbwave>

Examples

```
sim <- zinbSimulate()
```

Index

addFeatureStats, 3
 addGeneLengths, 4
 assays, 51

 BASiCS_MCMC, 5, 6
 BASiCS_Sim, 7
 BASiCSEstimate, 5, 7
 BASiCSParams, 6, 7
 BASiCSParams-class (BASiCSParams), 6
 BASiCSSimulate, 7, 7
 BiocParallelParam, 15, 29, 31, 54
 bridge, 8
 bringItemsForward, 8

 colData, 51
 compareSCEs, 9, 20, 22
 create_synthetic, 24

 diffSCEs, 10, 21, 22, 53

 empirical_lambda, 23
 estimateDisp, 41
 expandParams, 11
 expandParams, BASiCSParams-method
 (expandParams), 11
 expandParams, LunParams-method
 (expandParams), 11
 expandParams, SplatParams-method
 (expandParams), 11

 fitdist, 36, 43, 44

 getLNormFactors, 12, 46, 48
 getParam, 12
 getParam, Params-method (getParam), 12
 getParams, 13
 getPathOrder, 13
 ggplot, 9, 11

 lambda1_calculator, 38
 lambda2_calculator, 38
 listSims, 14
 logistic, 14
 lun2Estimate, 15, 16, 17
 Lun2Params, 15, 16

 Lun2Params-class (Lun2Params), 16
 lun2Simulate, 16, 17
 lunEstimate, 18, 19
 LunParams, 18, 19, 20
 LunParams-class (LunParams), 19
 lunSimulate, 19, 19

 makeCompPanel, 20
 makeDiffPanel, 21
 makeOverallPanel, 22
 mfaEstimate, 22, 24
 MFAParams, 23, 23, 24
 MFAParams-class (MFAParams), 23
 mfaSimulate, 24, 24

 newBASiCSParams (newParams), 25
 newLun2Params (newParams), 25
 newLunParams (newParams), 25
 newMFAParams (newParams), 25
 newParams, 25
 newPhenoParams (newParams), 25
 newSCDDParams (newParams), 25
 newSimpleParams (newParams), 25
 newSparseDCParams (newParams), 25
 newSplatParams (newParams), 25
 newZINBParams (newParams), 25
 nls, 41

 Params, 26
 Params-class (Params), 26
 phenoEstimate, 26, 27
 PhenoParams, 26, 27, 28
 PhenoParams-class (PhenoParams), 27
 phenoSimulate, 27, 27
 pre_proc_data, 38
 preprocess, 29

 rbindMatched, 28
 rowData, 4, 51

 scDD, 29
 scDDEstimate, 29, 30
 SCDDParams, 30, 31
 SCDDParams-class (SCDDParams), 30
 scDDSimulate, 30, 31

58

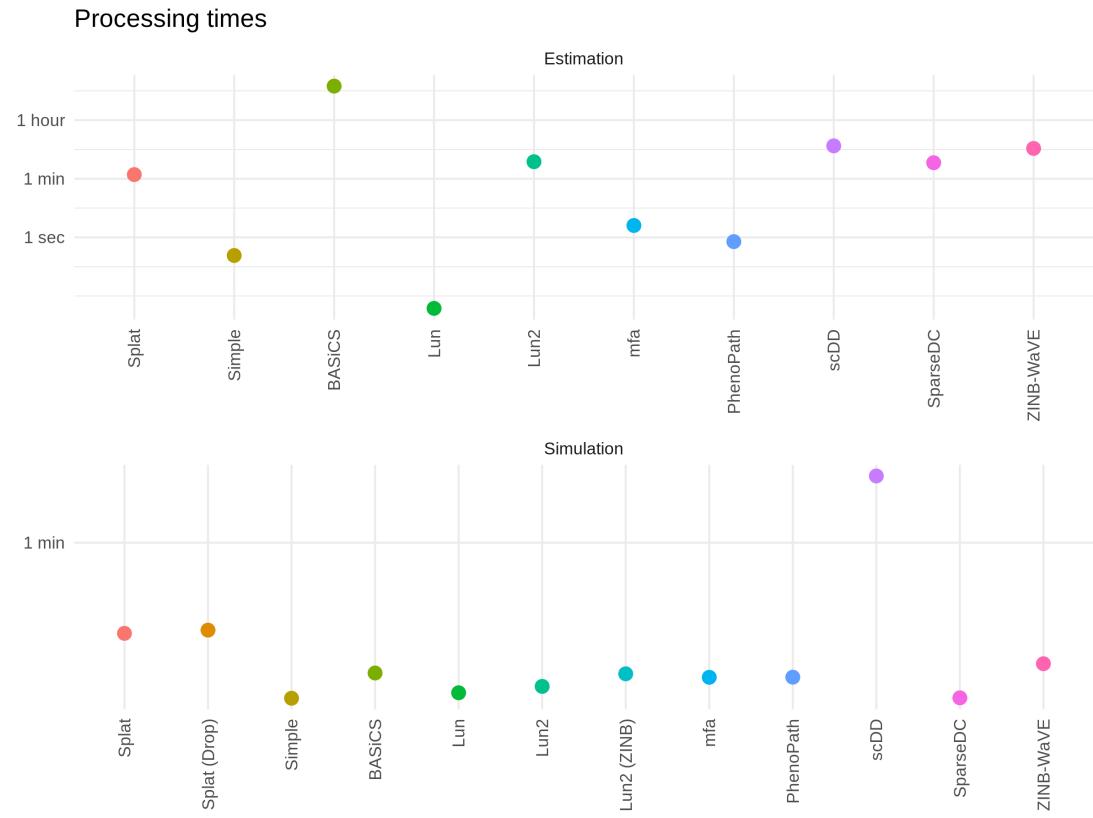
INDEX

SerialParam, [15, 29, 31, 54](#)
 setParam, [32, 33, 34](#)
 setParam, BASiCSParams-method
 (setParam), [32](#)
 setParam, Lun2PParams-method (setParam),
 [32](#)
 setParam, LunParams-method (setParam), [32](#)
 setParam, Params-method (setParam), [32](#)
 setParam, PhenoParams-method (setParam),
 [32](#)
 setParam, SCDDParams-method (setParam),
 [32](#)
 setParam, SplatParams-method (setParam),
 [32](#)
 setParam, ZINBParams-method (setParam),
 [32](#)
 setParams, [25, 33, 51](#)
 setParams, Params-method (setParams), [33](#)
 setParams, SplatParams-method
 (setParams), [33](#)
 setParamsUnchecked, [34](#)
 setParamUnchecked, [34](#)
 setParamUnchecked, Params-method
 (setParamUnchecked), [34](#)
 showDFs, [35](#)
 showPP, [35](#)
 showValues, [36](#)
 sim_data, [40](#)
 simpleEstimate, [36, 37](#)
 SimpleParams, [36, 37, 38](#)
 SimpleParams-class (SimpleParams), [37](#)
 simpleSimulate, [37, 37](#)
 simulate_phenopath, [28](#)
 simulateSet, [30, 31](#)
 SingleCellExperiment, [4, 7, 24, 28, 30, 31,](#)
 [40, 51, 56](#)
 sparsedc_cluster, [38](#)
 sparseDCEstimate, [38, 39](#)
 SparseDCParams, [38, 39, 40](#)
 SparseDCParams-class (SparseDCParams),
 [39](#)
 sparseDCSimulate, [39, 40](#)
 splatEstBCV, [41, 42](#)
 splatEstDropout, [41, 42](#)
 splatEstimate, [42, 46](#)
 splatEstLib, [42, 43](#)
 splatEstMean, [42, 43](#)
 splatEstOutlier, [42, 44](#)
 SplatParams, [44, 50](#)
 SplatParams-class (SplatParams), [44](#)
 splatSimBatchCellMeans, [46, 52](#)
 splatSimBatchEffects, [46, 52](#)
 splatSimBCVMeans, [47, 52](#)
 splatSimCellMeans, [47, 52](#)
 splatSimDE, [48, 52](#)
 splatSimDropout, [48, 52](#)
 splatSimGeneMeans, [49, 52](#)
 splatSimGroupCellMeans
 (splatSimCellMeans), [47](#)
 splatSimGroupDE (splatSimDE), [48](#)
 splatSimLibSizes, [49, 52](#)
 splatSimPathCellMeans
 (splatSimCellMeans), [47](#)
 splatSimPathDE (splatSimDE), [48](#)
 splatSimSingleCellMeans
 (splatSimCellMeans), [47](#)
 splatSimTrueCounts, [50, 52](#)
 splatSimulate, [46, 50](#)
 splatSimulateGroups (splatSimulate), [50](#)
 splatSimulatePaths (splatSimulate), [50](#)
 splatSimulateSingle (splatSimulate), [50](#)
 splatter, [52](#)
 splatter-package (splatter), [52](#)
 summariseDiff, [53](#)
 winsorize, [53](#)
 zinbEstimate, [54, 55](#)
 zinbFit, [54](#)
 ZinbModel, [55](#)
 zinbModel, [55](#)
 ZINBParams, [54, 55, 56](#)
 ZINBParams-class (ZINBParams), [55](#)
 zinbSim, [56](#)
 zinbSimulate, [55, 55](#)

C

Simulation
comparison

C.1 Timings



C.2 Package versions

Versions of simulation packages used for the comparison.

Package	Version
BASiCS	1.4.1
DESeq2	1.22.2
edgeR	3.22.3
ggplot2	3.0.0
limma	3.36.2
mfa	1.4.1
phenopath	1.6.0
scater	1.10.1
scDD	1.6.1
scran	1.10.2
SingleCellExperiment	1.4.1
SparseDC	0.1.17
splatter	1.6.1
zinbwave	1.4.1

D

clustree
documentation

D.1 clustree vignette

Plotting clustering trees

Luke Zappia

Last updated: 24 February 2019

- 1 What is a clustering tree?
- 2 A simple example
 - 2.1 The data
 - 2.2 Plotting a tree
 - 2.3 Controlling aesthetics
 - 2.3.1 SC3 stability index
 - 2.4 Layout
 - 2.5 Adding labels
- 3 Clustering trees for scRNA-seq data
 - 3.1 SingleCellExperiment objects
 - 3.2 Seurat objects
 - 3.3 Using genes as aesthetics
- 4 Overlaying clustering trees
 - 4.1 Choosing what to colour
 - 4.2 Labelling nodes
 - 4.3 Showing the side view
- 5 Modifying appearance
 - 5.1 Legends
- 6 Citing clustree
- References

1 What is a clustering tree?

Clustering analysis is used in many contexts to group similar samples. One problem when conducting this kind of analysis is how many clusters to use. This is usually controlled by a parameter provided to the clustering algorithm, such as k for k -means clustering.

Statistics designed to help you make this choice typically either compare two clusterings or score a single clustering. A clustering tree is different in that it visualises the relationships between at a range of resolutions.

To build a clustering tree we need to look at how cells move as the clustering resolution is increased. Each cluster forms a node in the tree and edges are constructed by considering the cells in a cluster at a lower resolution (say $k = 2$) that end up in a cluster at the next highest resolution (say $k = 3$). By connecting clusters in this way we can see how clusters are related to each other, which are clearly distinct and which are unstable. Extra information about the cells in each node can also be overlaid in order to help make the decision about which resolution to use. For more information about clustering trees please refer to our associated publication (Zappia and Oshlack 2018).

2 A simple example

To demonstrate what a clustering tree looks like we will work through a short example using the well known `iris` dataset.

2.1 The data

The `iris` dataset consists of measurements (sepal length, sepal width, petal length and petal width) of 150 iris flowers, 50 from each of three species (**Iris setosa**, **Iris versicolor** and **Iris virginica**). For more information see `?iris`. We are going to use a version of this dataset that has already been clustered. Let's load the data and take a look:

```
library(clustree)
3> Loading required package: ggraph
3> Loading required package: ggplot2
3> Registered S9 methods overwritten by `ggplot2`:
3>   method      from
3>   [.quosures    rlang
3>   c.quosures    rlang
3>   print.quosures rlang
data("iris_clusts")

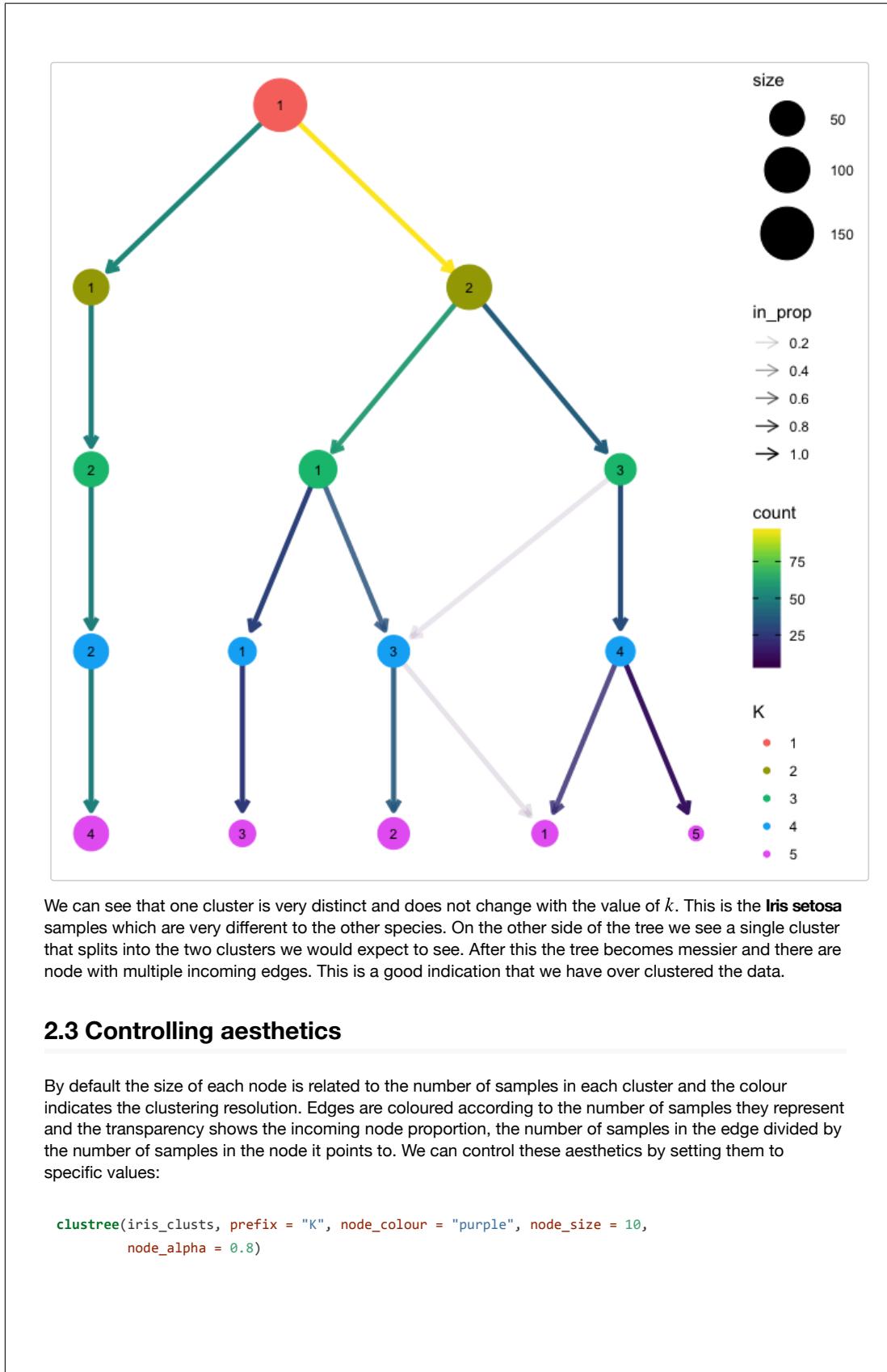
head(iris_clusts)
3>   Sepal.Length Sepal.Width Petal.Length Petal.Width Species K7 K8 K9 K0 K0
3> 7       0.7      9.0       7.0       4.8  setosa  7  7  8  8  0
3> 8       0.3      9.4       7.0       4.8  setosa  7  7  8  8  0
3> 9       0.1      9.8       7.9       4.8  setosa  7  7  8  8  0
3> 0       0.0      9.7       7.0       4.8  setosa  7  7  8  8  0
3> 0       0.4      9.0       7.0       4.8  setosa  7  7  8  8  0
3> 0       0.0      9.3       7.1       4.0  setosa  7  7  8  8  0
3>   PC7      PC8
3> 7 -8.020780 -4.9739318
3> 8 -8.170708  4.7114478
3> 9 -8.222337  4.7003030
3> 0 -8.100909  4.9728334
3> 0 -8.182171 -4.9801000
3> 0 -8.824204 -4.1079940
```

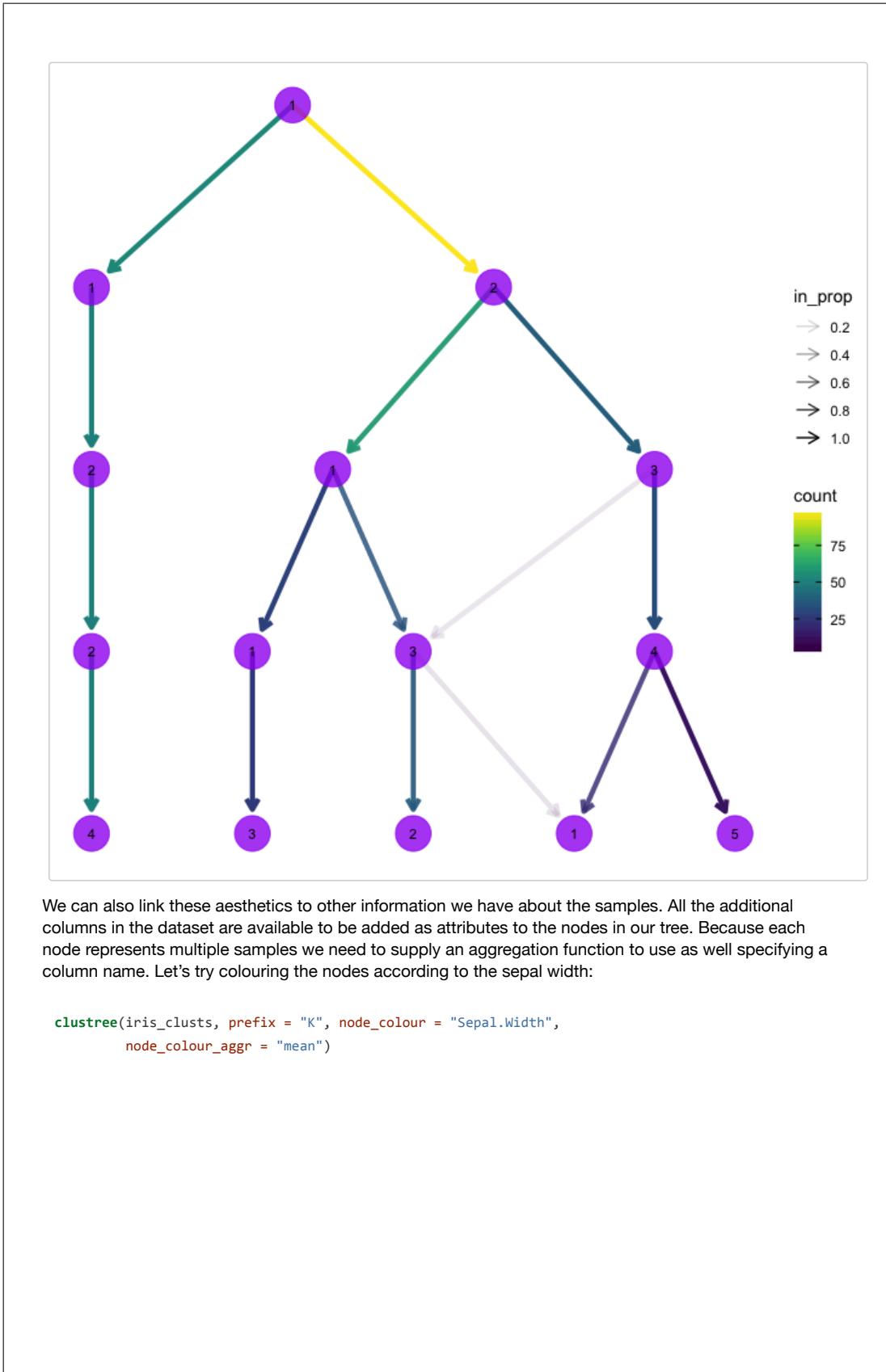
Here we have a `data.frame` with the normal `iris` datasets, the measurements and species, plus some additional columns. These columns contain the cluster assignments from clustering this data using k -means with values ok k from $k = 1$ to $k = 5$.

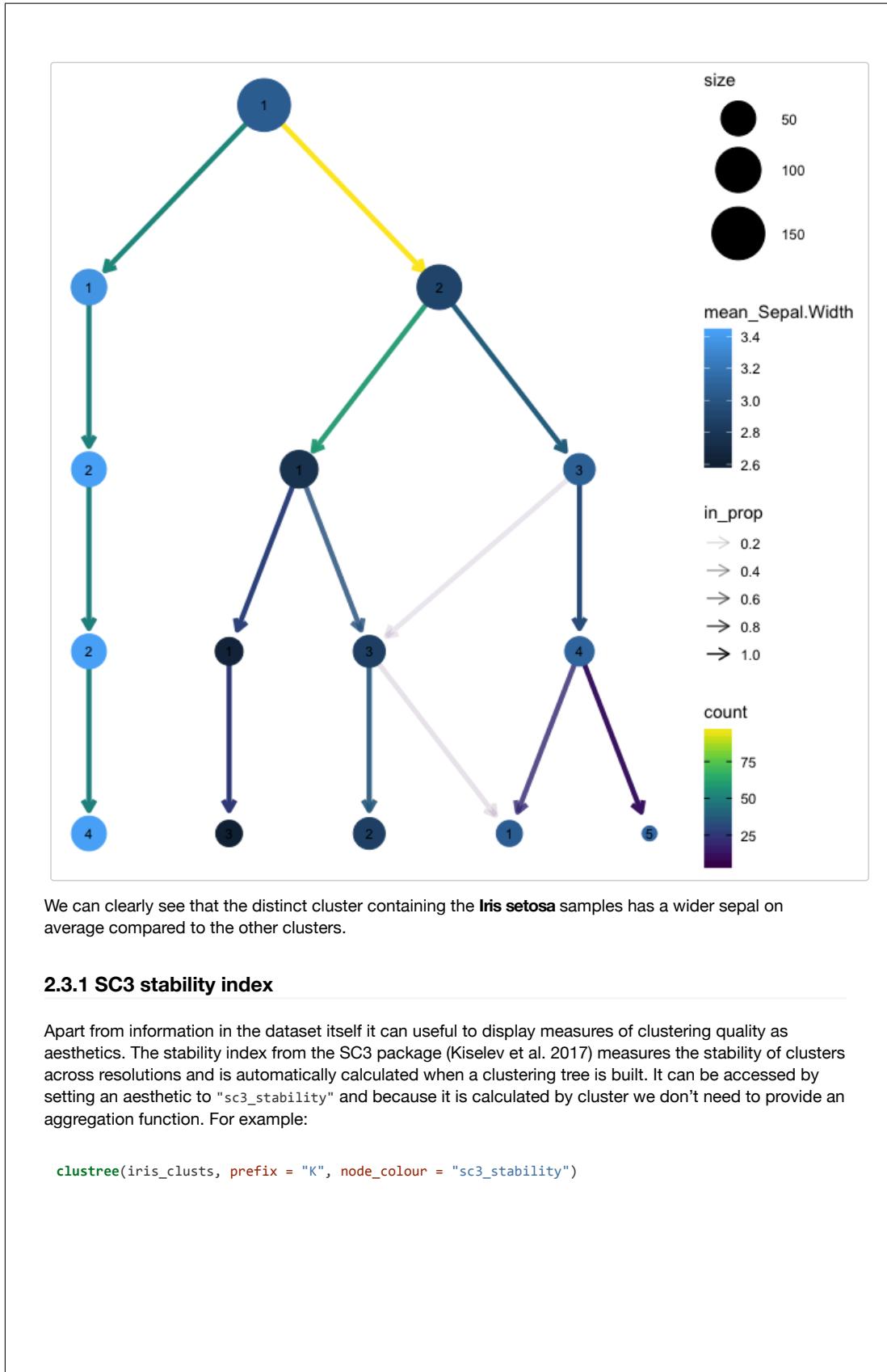
2.2 Plotting a tree

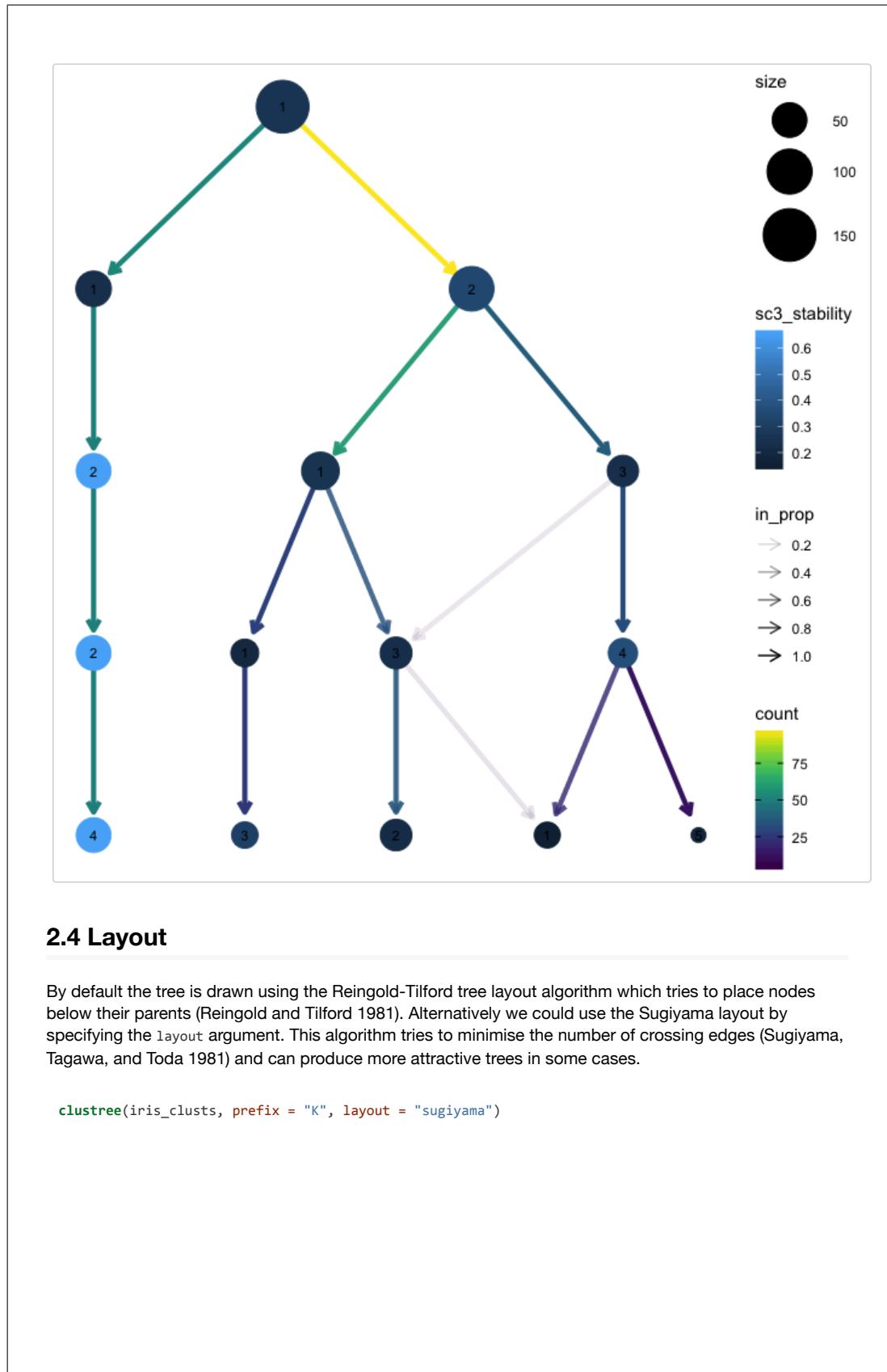
This clustering information is all we need to build a clustering tree. Each column must consist of numeric values indicating which cluster each sample has been assigned to. To plot the tree we just pass this information to the `clustree` function. We also need to specify a `prefix` string to indicate which columns contain the clusterings.

```
clustree(iris_clusts, prefix = "K")
```





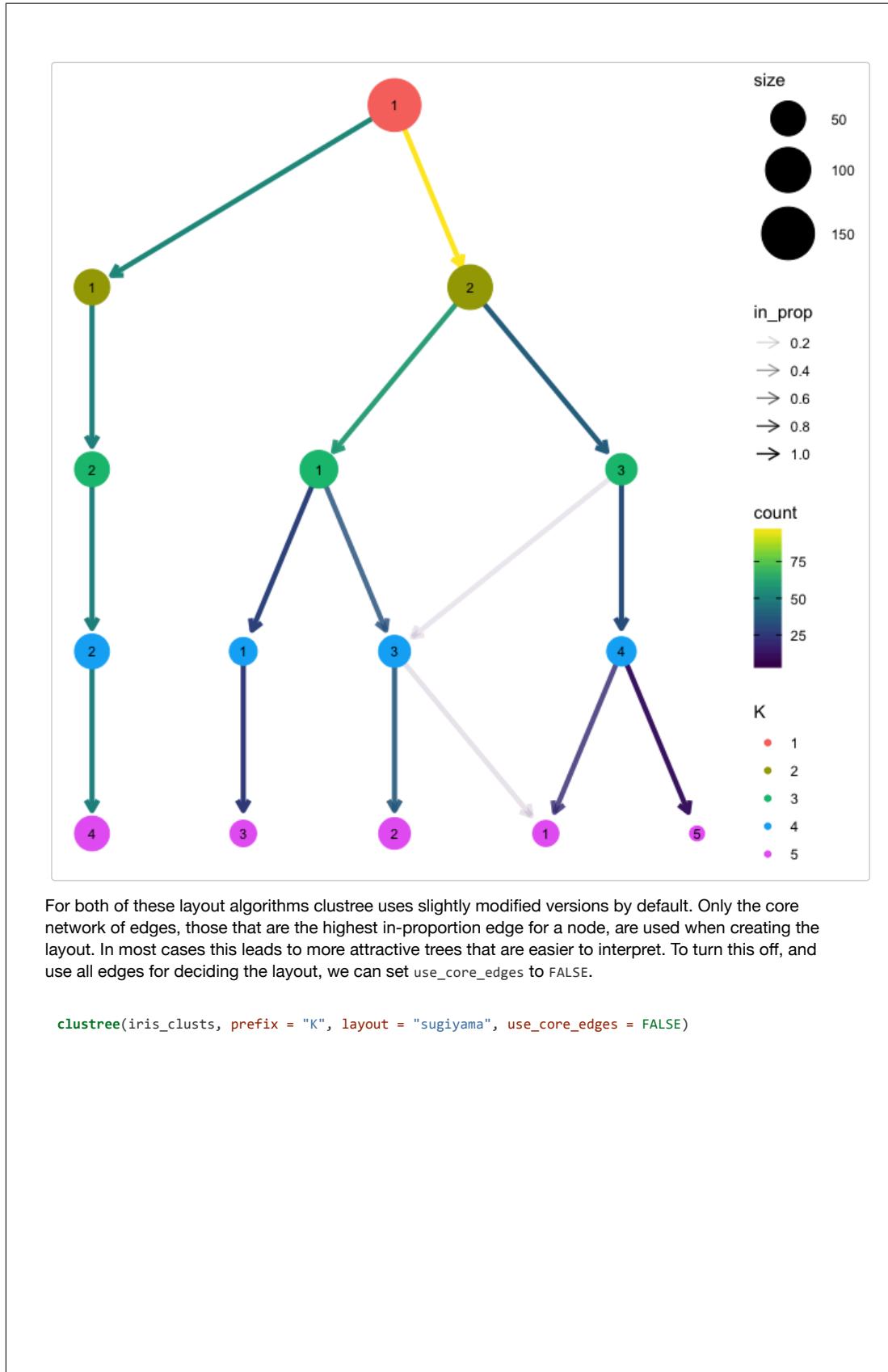


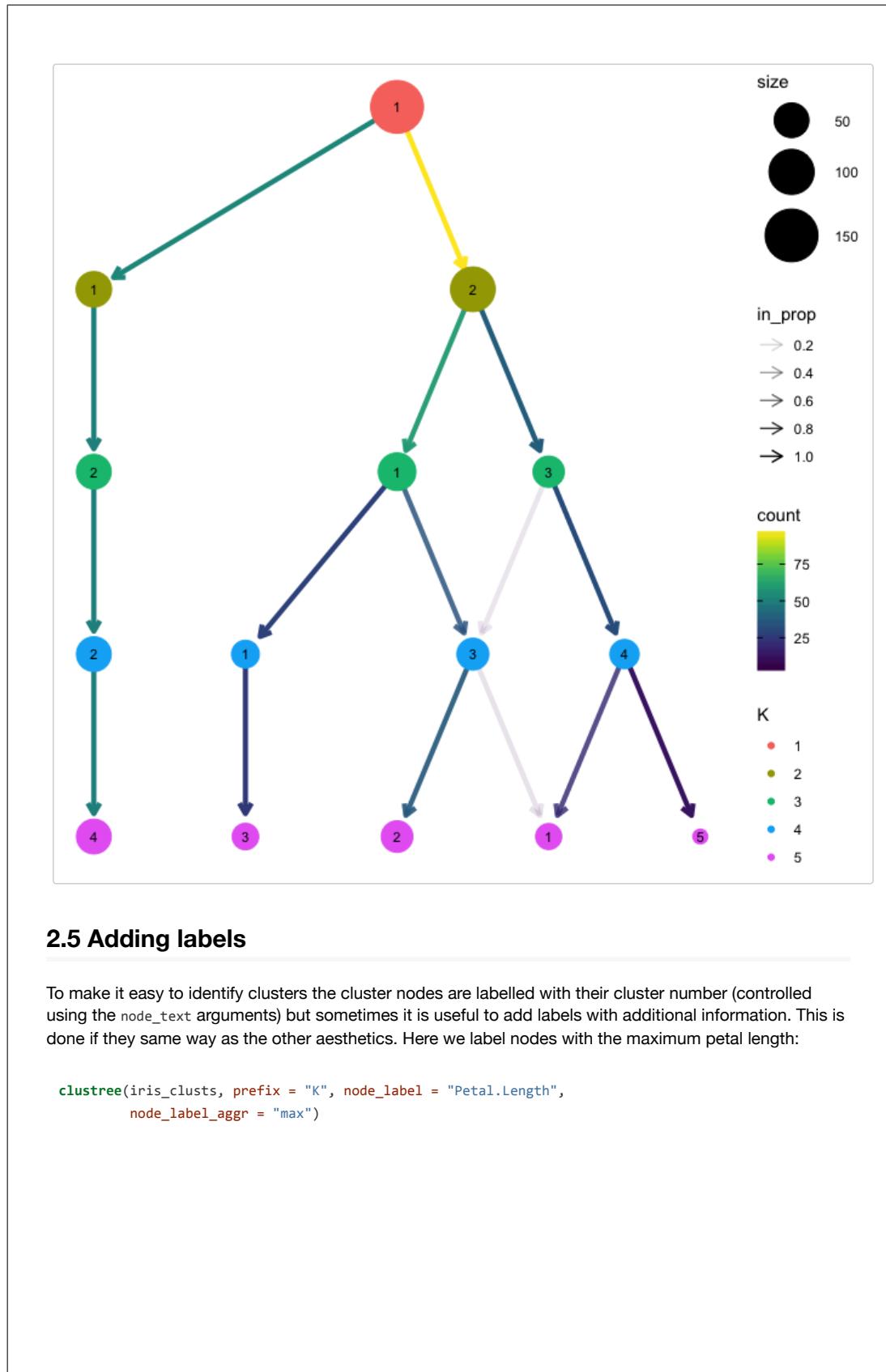


2.4 Layout

By default the tree is drawn using the Reingold-Tilford tree layout algorithm which tries to place nodes below their parents (Reingold and Tilford 1981). Alternatively we could use the Sugiyama layout by specifying the `layout` argument. This algorithm tries to minimise the number of crossing edges (Sugiyama, Tagawa, and Toda 1981) and can produce more attractive trees in some cases.

```
clustree(iris_clusts, prefix = "K", layout = "sugiyama")
```

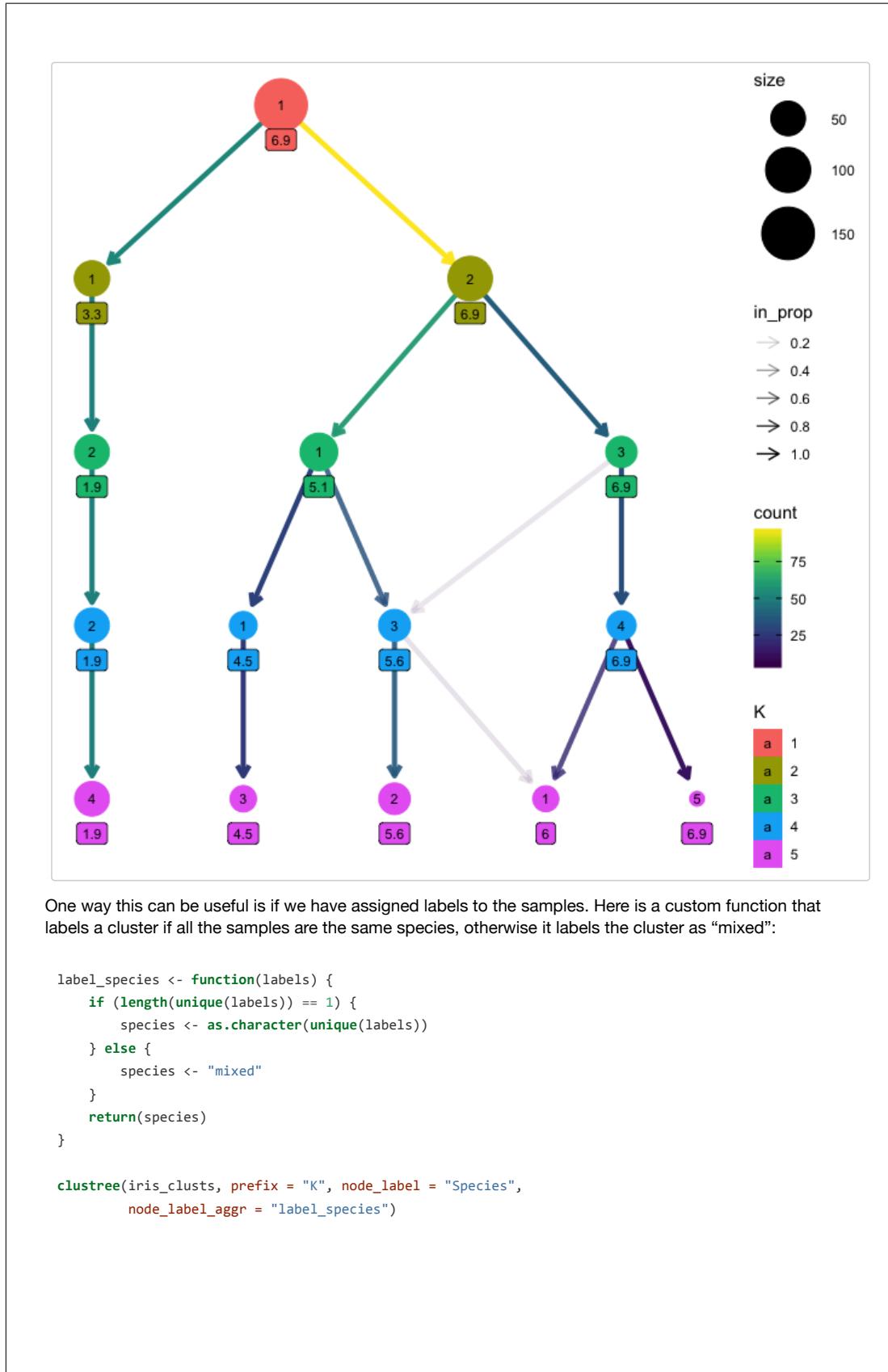


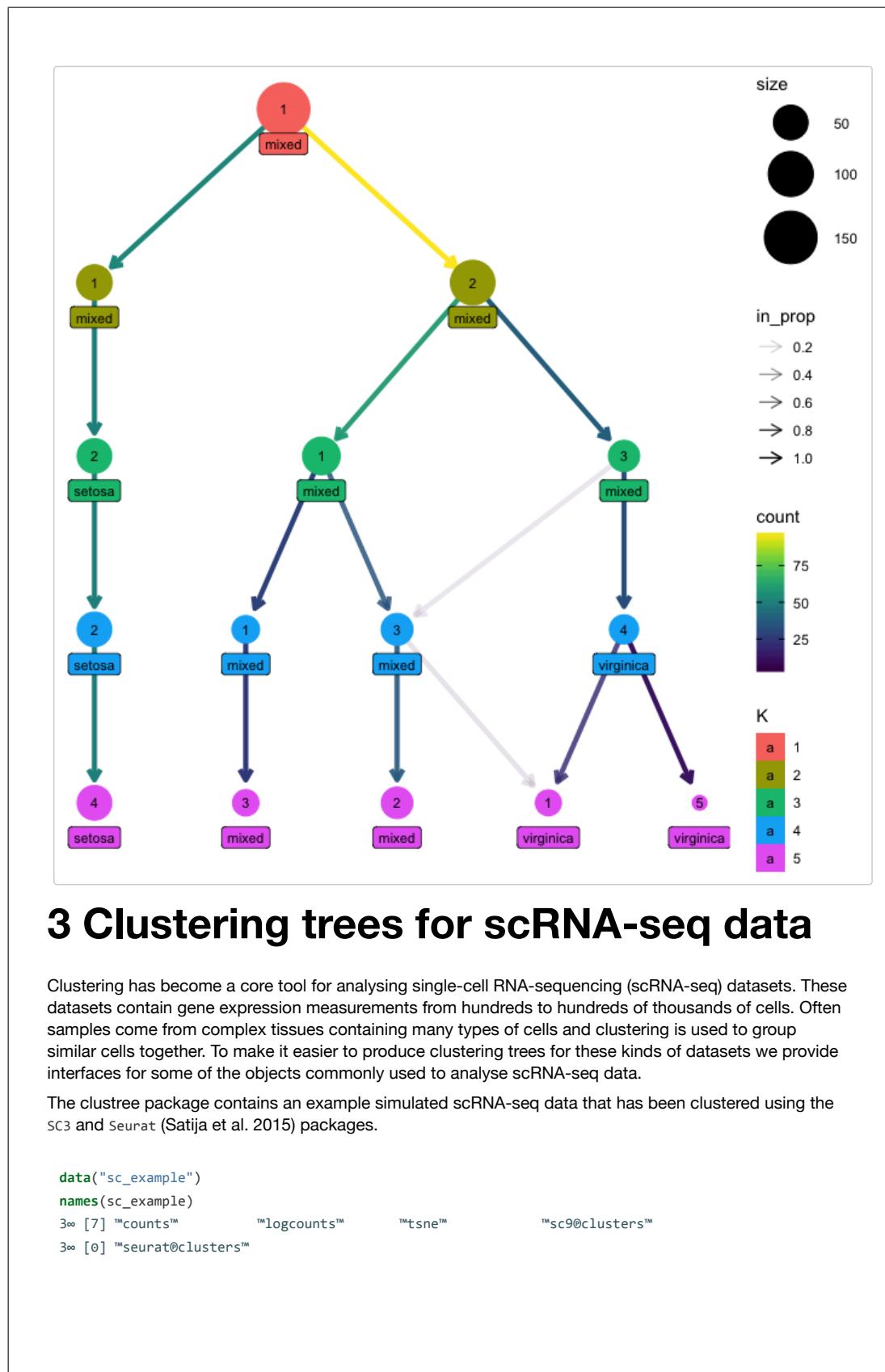


2.5 Adding labels

To make it easy to identify clusters the cluster nodes are labelled with their cluster number (controlled using the `node_text` argument) but sometimes it is useful to add labels with additional information. This is done if they same way as the other aesthetics. Here we label nodes with the maximum petal length:

```
clustree(iris_clusts, prefix = "K", node_label = "Petal.Length",
          node_label_aggr = "max")
```





3.1 SingleCellExperiment objects

The `SingleCellExperiment` is one of these common objects, used across a range of Bioconductor packages. Let's have a look at an example, but first we need to convert the example dataset to a `SingleCellExperiment` object:

```
suppressPackageStartupMessages(library("SingleCellExperiment"))

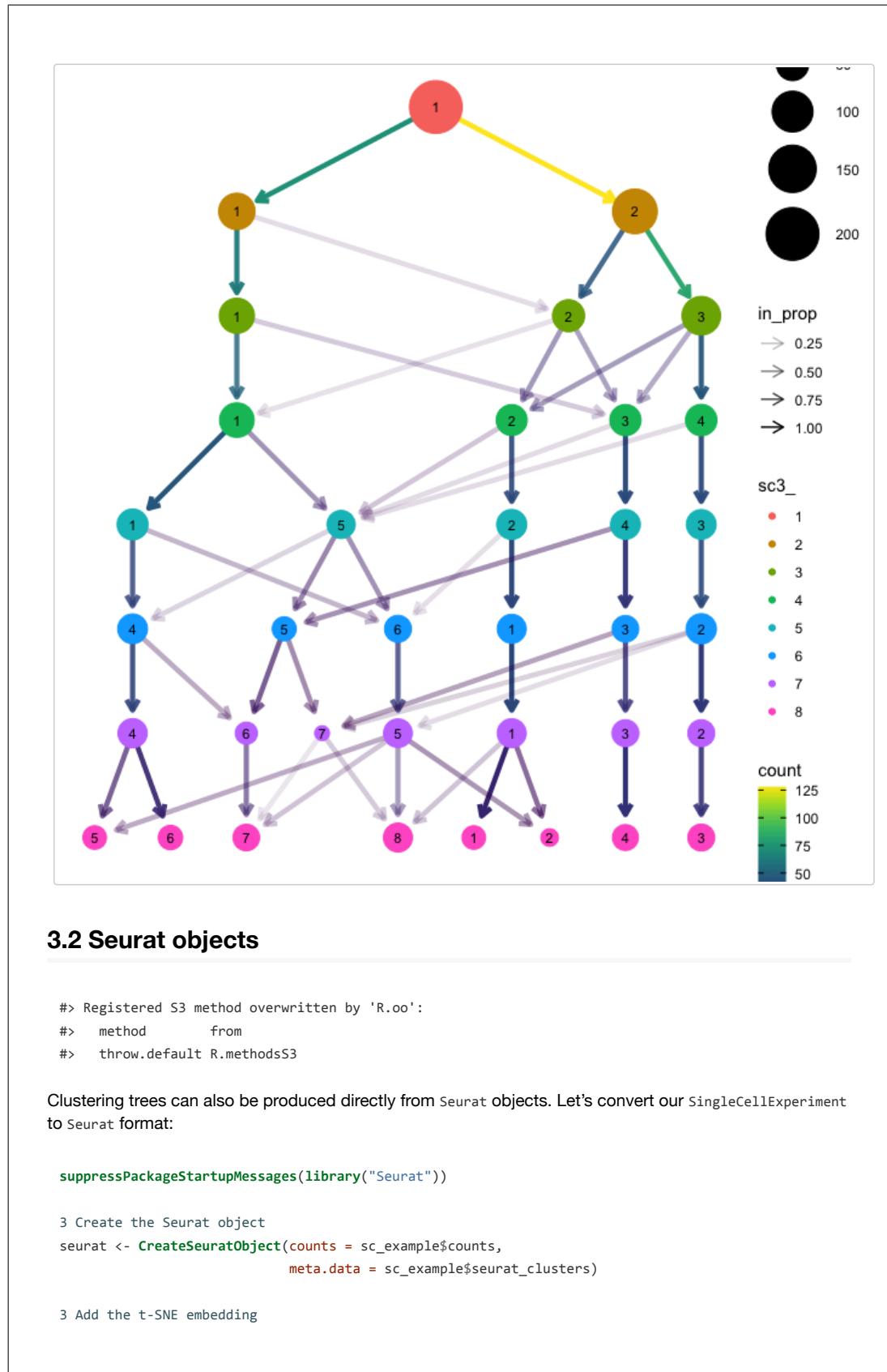
sce <- SingleCellExperiment(assays = list(counts = sc_example$counts,
                                           logcounts = sc_example$logcounts),
                           colData = sc_example$sc3_clusters,
                           reducedDims = SimpleList(TSNE = sc_example$tsne))
```

The clustering information is held in the `colData` slot.

```
head(colData(sce))
#> DataFrame with 0 rows and 2 columns
#>   sc9@7@clusters sc9@8@clusters sc9@9@clusters sc9@0@clusters
#>   <factors>     <factors>     <factors>     <factors>
#> 1 Cell7         7             8             9             0
#> 2 Cell8         7             8             9             8
#> 3 Cell9         7             7             7             7
#> 4 Cell0         7             7             7             9
#> 5 Cell0         7             8             9             0
#> 6 Cell0         7             8             8             8
#> 
#>   sc9@0@clusters sc9@0@clusters sc9@1@clusters sc9@2@clusters
#>   <factors>     <factors>     <factors>     <factors>
#> 1 Cell7         9             8             8             9
#> 2 Cell8         8             7             7             7
#> 3 Cell9         0             0             0             0
#> 4 Cell0         0             9             9             0
#> 5 Cell0         9             8             0             2
#> 6 Cell0         8             7             7             7
```

We can plot a clustering tree in the same way we did with a `data.frame`. In this case the clustering column names contain a suffix that needs to be stripped away, so we will pass that along as well.

```
clustree(sce, prefix = "sc3_", suffix = "_clusters")
```



3.2 Seurat objects

```
#> Registered S3 method overwritten by 'R.oo':
#>   method      from
#>   throw.default R.methodsS3
```

Clustering trees can also be produced directly from Seurat objects. Let's convert our SingleCellExperiment to Seurat format:

```
suppressPackageStartupMessages(library("Seurat"))

3 Create the Seurat object
seurat <- CreateSeuratObject(counts = sc_example$counts,
                             meta.data = sc_example$seurat_clusters)

3 Add the t-SNE embedding
```

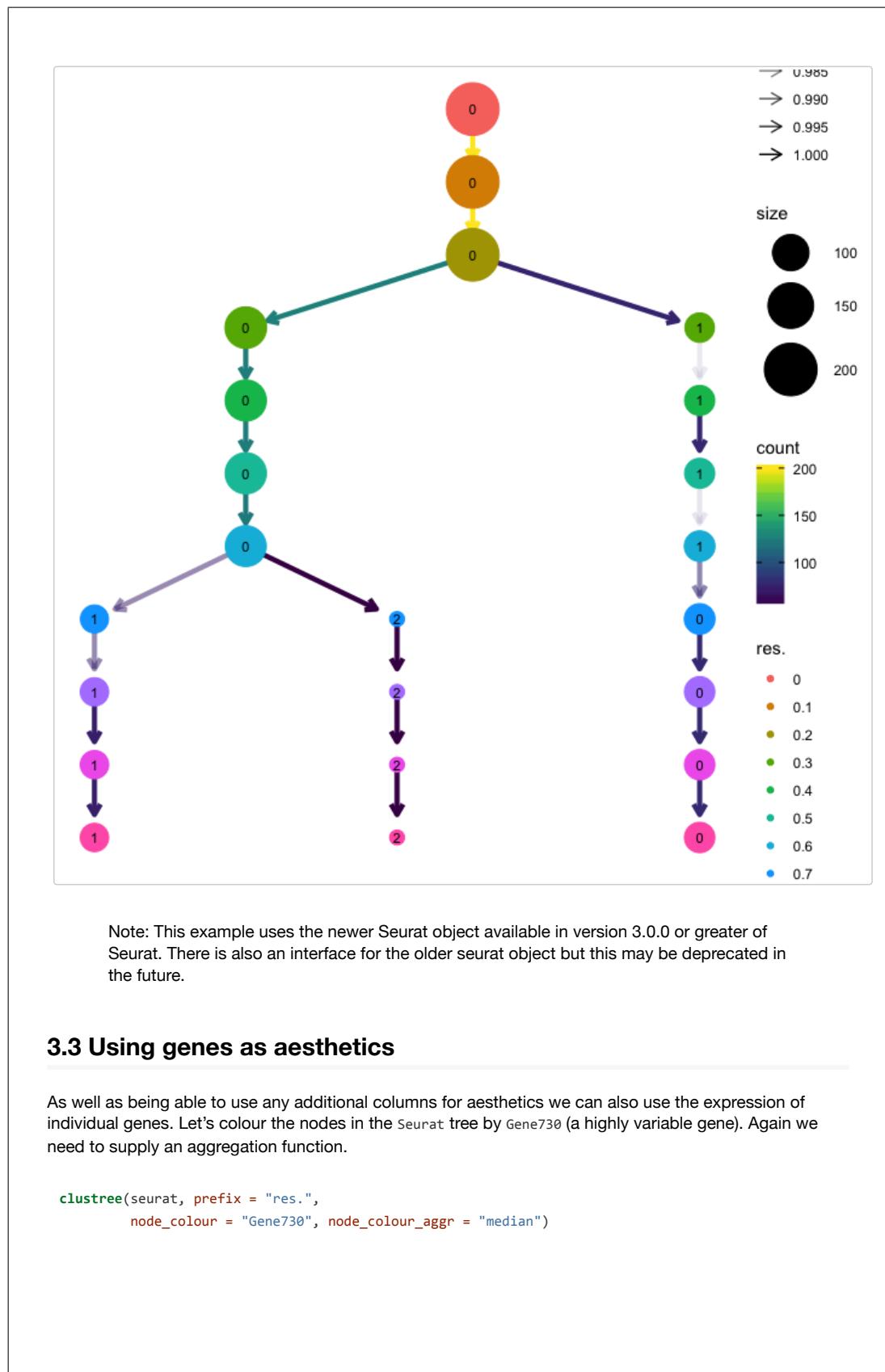
```
seurat[['TSNE']] <- CreateDimReducObject(embeddings = sc_example$tsne,
                                         key = "tSNE_")
3oo Warning: No assay specified, setting assay as RNA by default.
3oo Warning: No columnnames present in cell embeddings, setting to @tSNE@7:8@
```

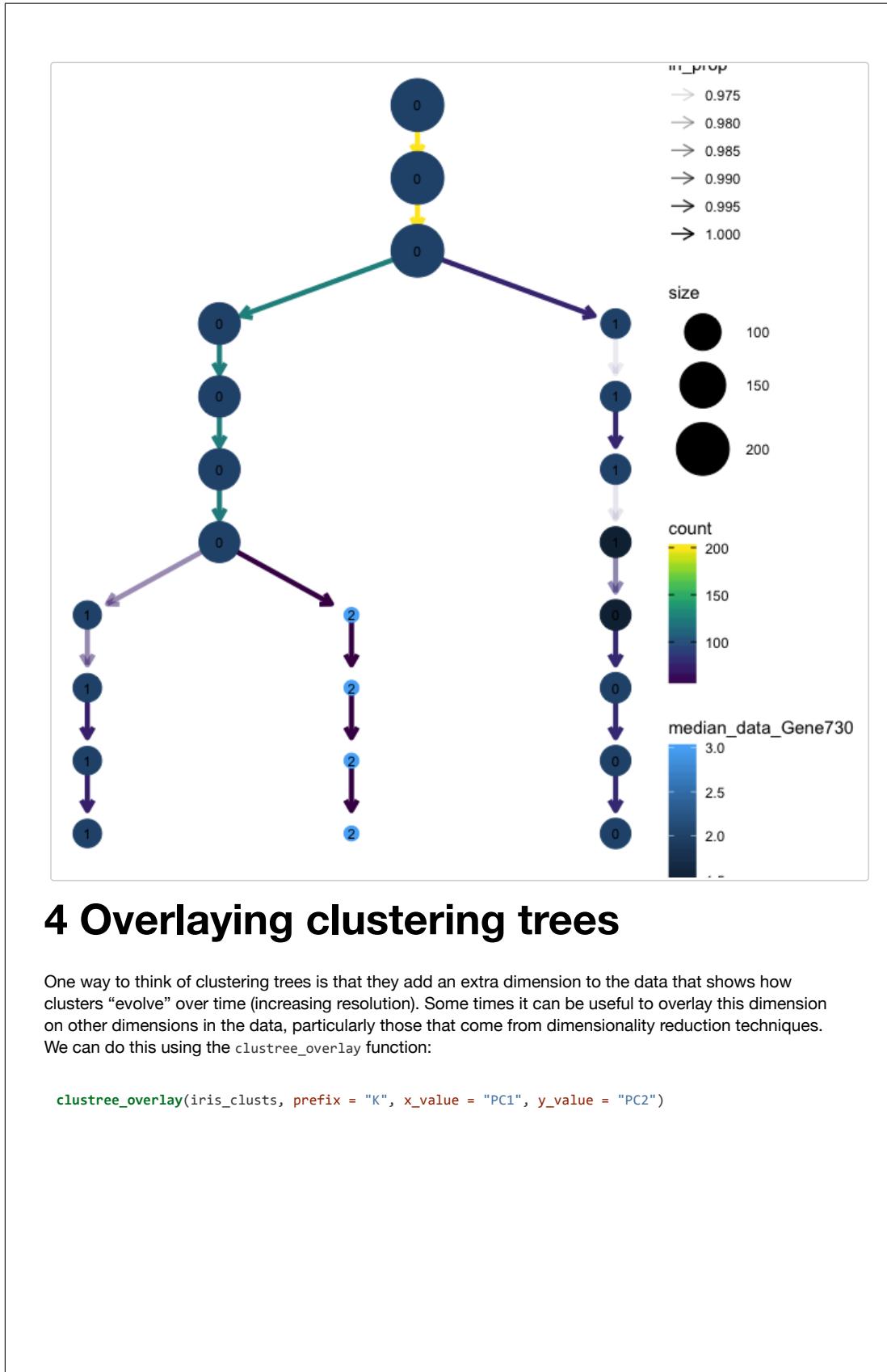
In this case the clustering information is held in the `meta.data` slot which can be accessed using the `[[` operator:

```
head(seurat[[[]])
      orig.ident nCount@RNA nFeature@RNA nGene nUMI res.4 res.4.7
3oo Cell17 SeuratProject     0002      049  049 0002    4    4
3oo Cell18 SeuratProject     0188      003  003 0188    4    4
3oo Cell19 SeuratProject     9304      040  040 9304    4    4
3oo Cell0 SeuratProject     0288      080  080 0288    4    4
3oo Cell0 SeuratProject     0402      038  038 0402    4    4
3oo Cell0 SeuratProject     0124      074  074 0124    4    4
3oo             res.4.8 res.4.9 res.4.0 res.4.0 res.4.0 res.4.1 res.4.2 res.4.3
3oo Cell17     4     7     7     7     7     4     4    4
3oo Cell18     4     7     7     7     7     4     4    4
3oo Cell19     4     4     4     4     4     7     7    7
3oo Cell0     4     4     4     4     4     7     7    7
3oo Cell0     4     7     7     7     7     4     4    4
3oo Cell0     4     4     4     4     4     7     7    7
3oo             res.7
3oo Cell17     4
3oo Cell18     4
3oo Cell19     7
3oo Cell0     7
3oo Cell0     4
3oo Cell0     7
```

We can now produce a clustering tree using this object. In this example the prefix for clustering columns is `res.`, but in most cases the default prefix from `Seurat` will be automatically used.

```
clustree(seurat, prefix = "res.")
```

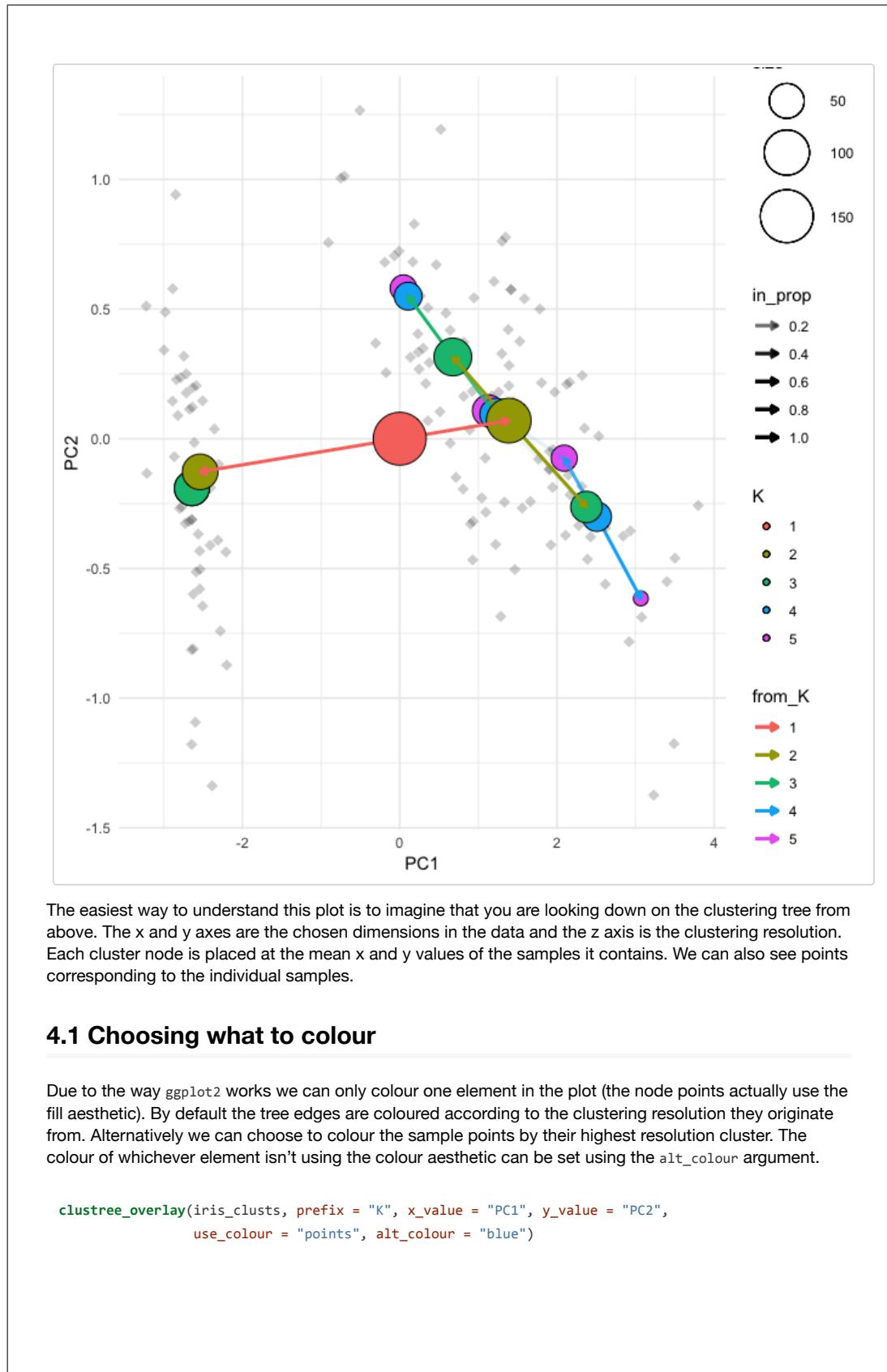


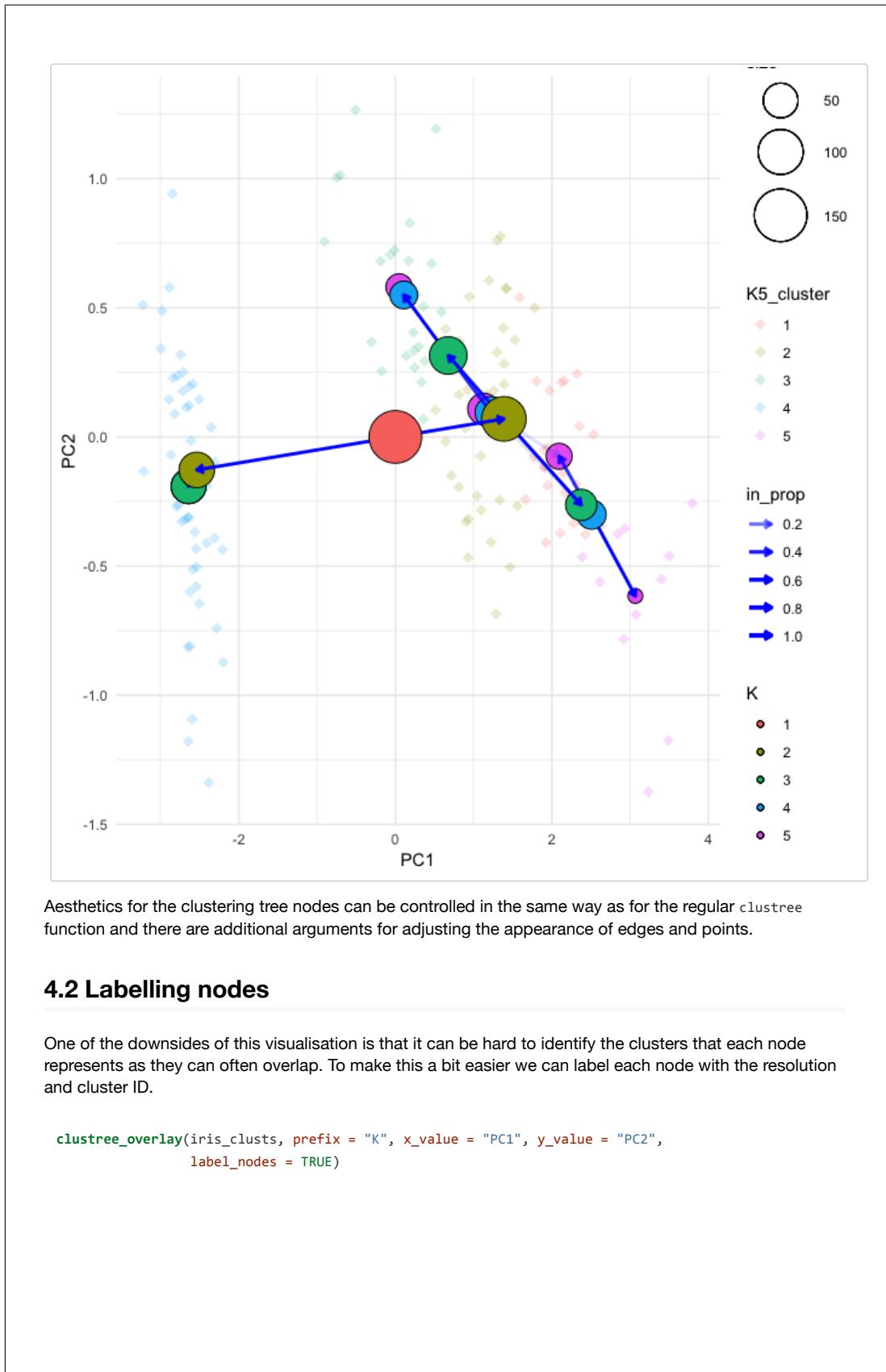


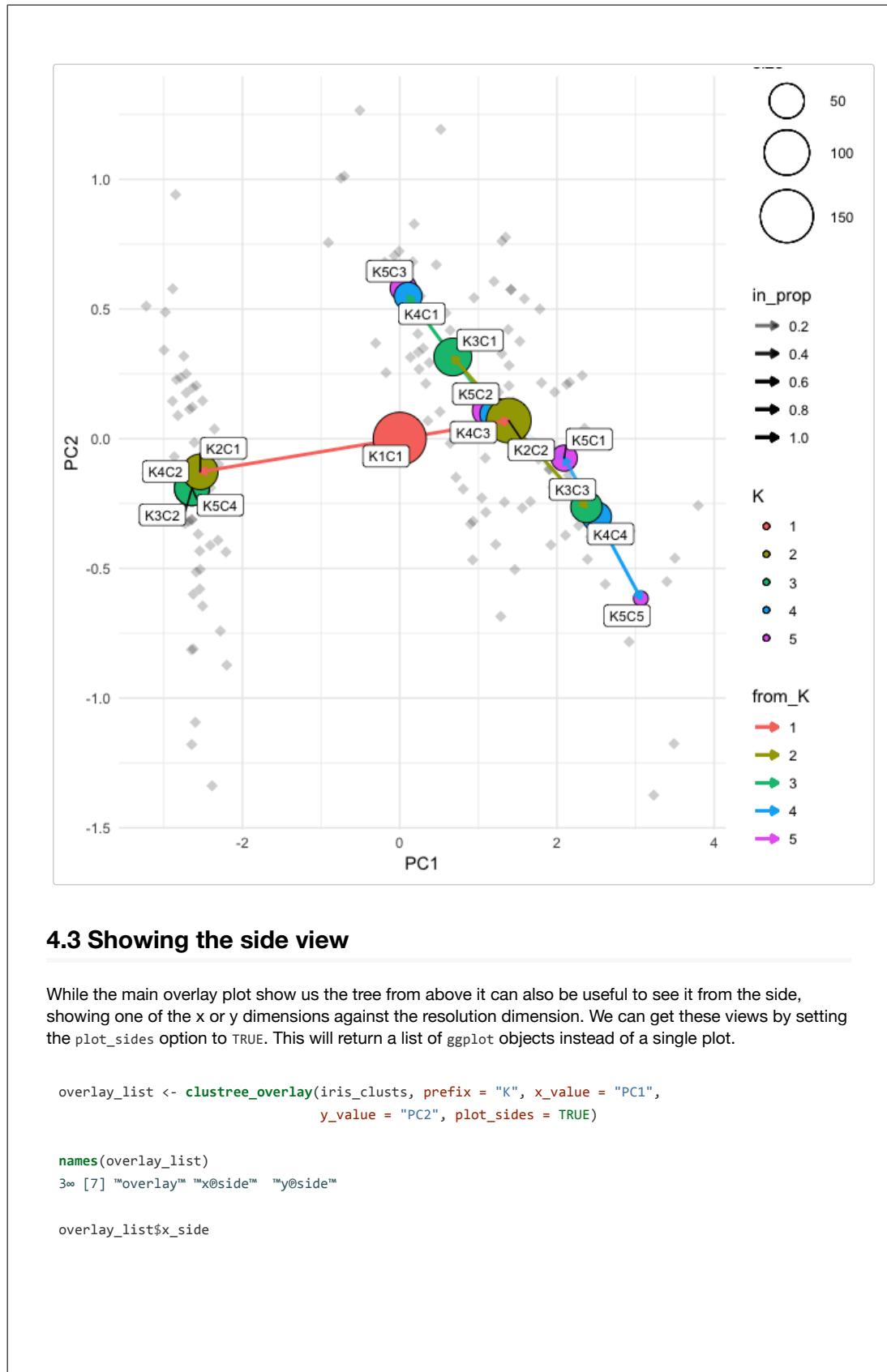
4 Overlaying clustering trees

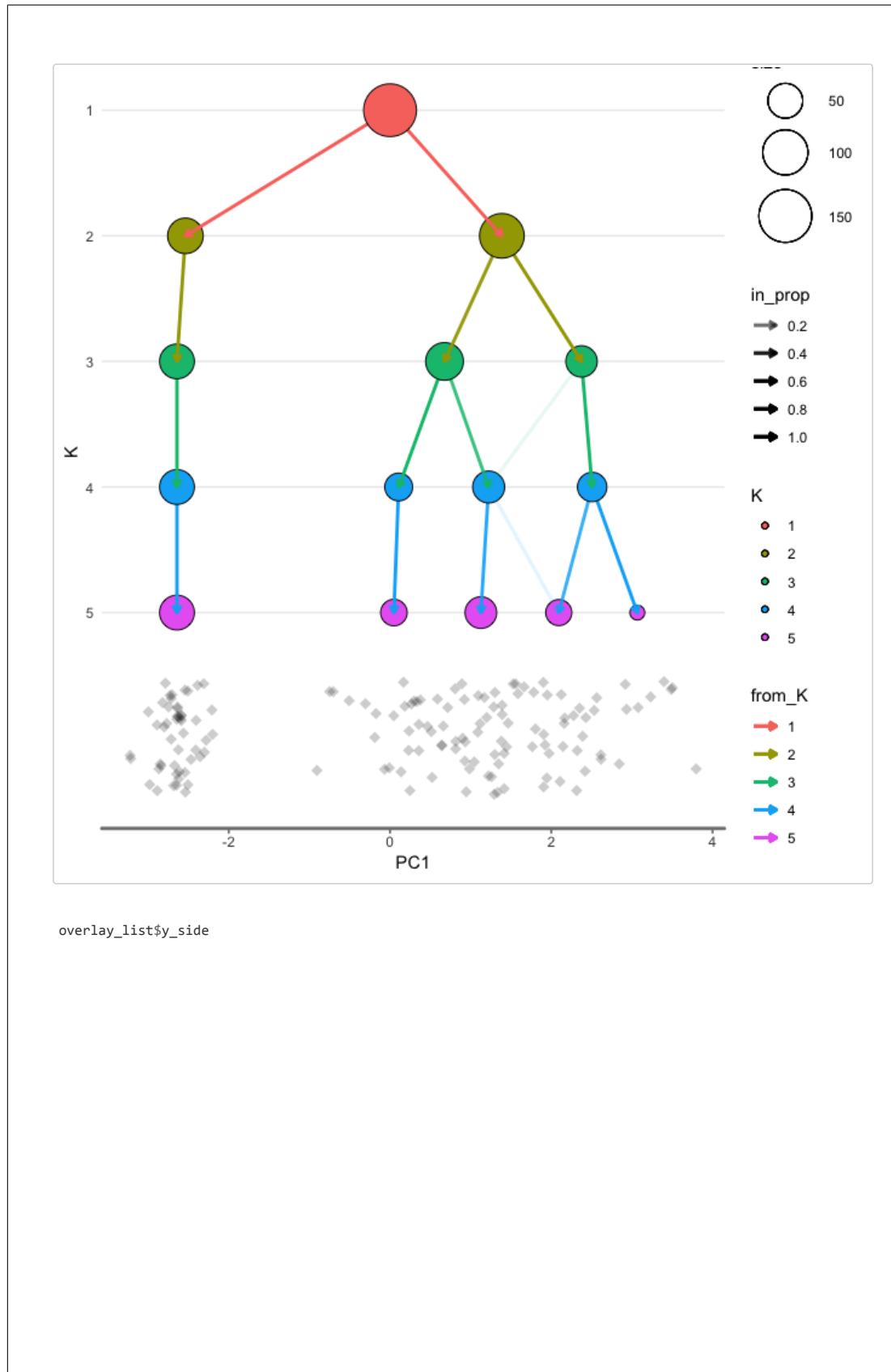
One way to think of clustering trees is that they add an extra dimension to the data that shows how clusters “evolve” over time (increasing resolution). Some times it can be useful to overlay this dimension on other dimensions in the data, particularly those that come from dimensionality reduction techniques. We can do this using the `clustree_overlay` function:

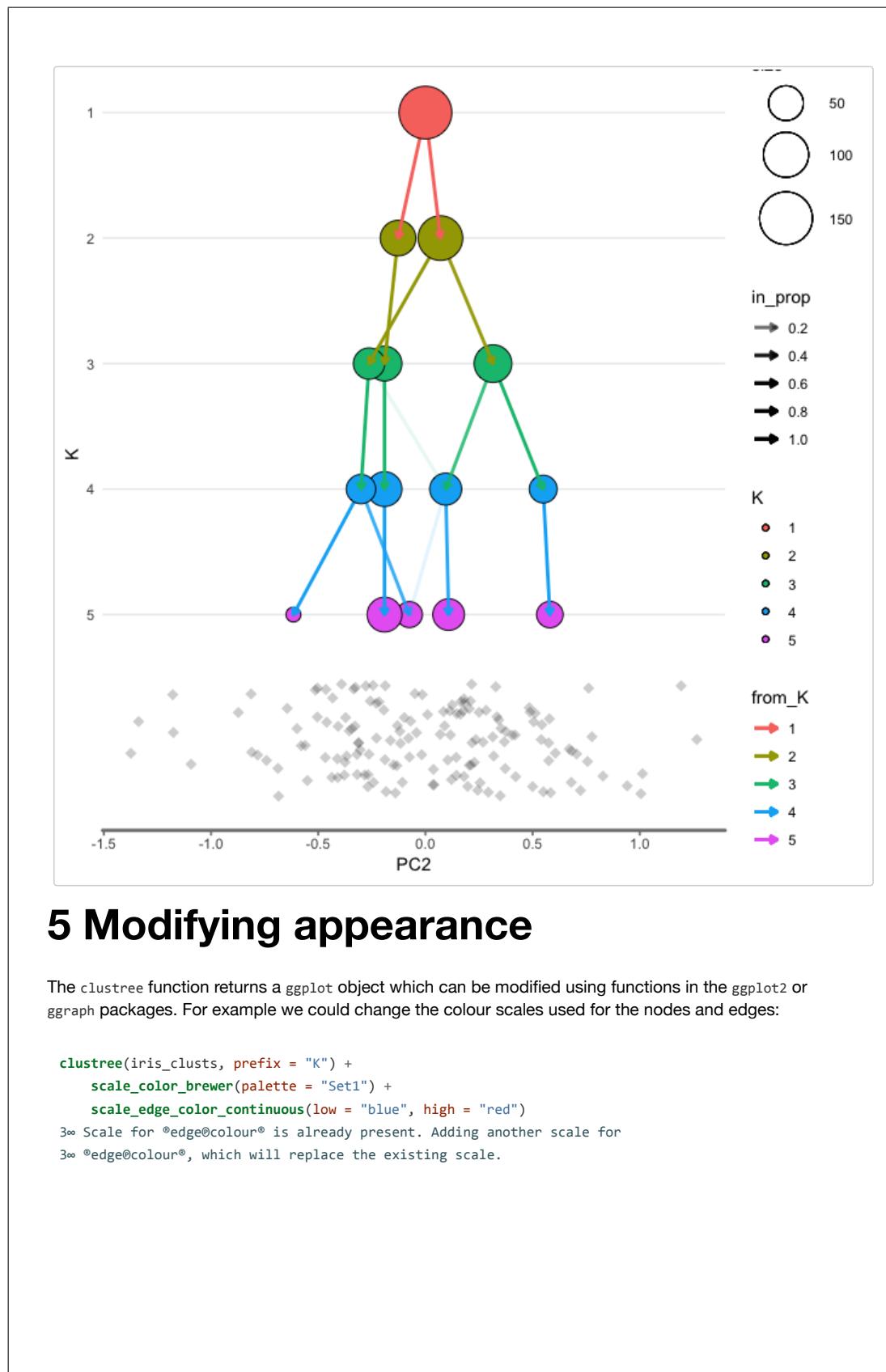
```
clustree_overlay(iris_clusts, prefix = "K", x_value = "PC1", y_value = "PC2")
```







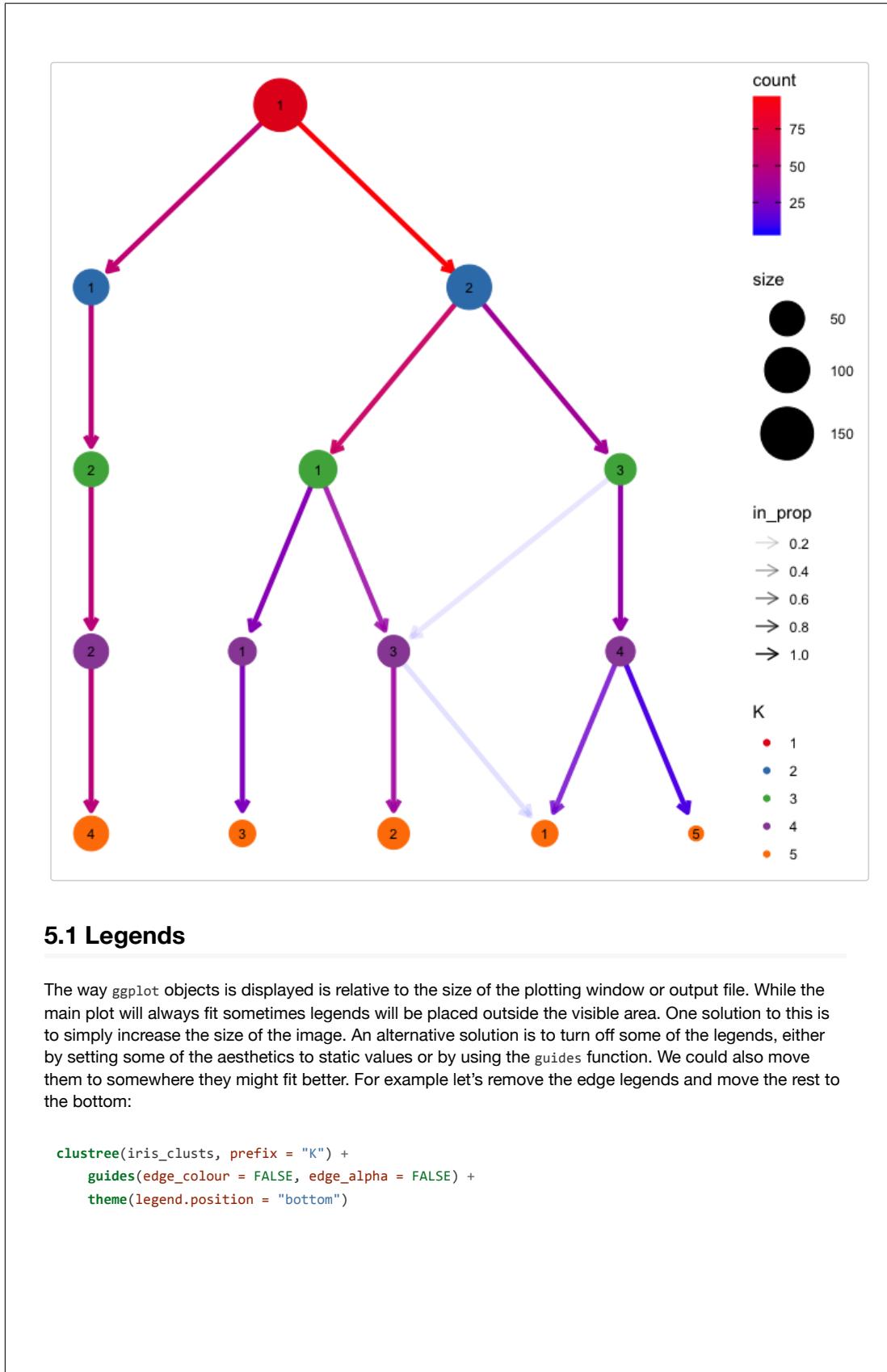


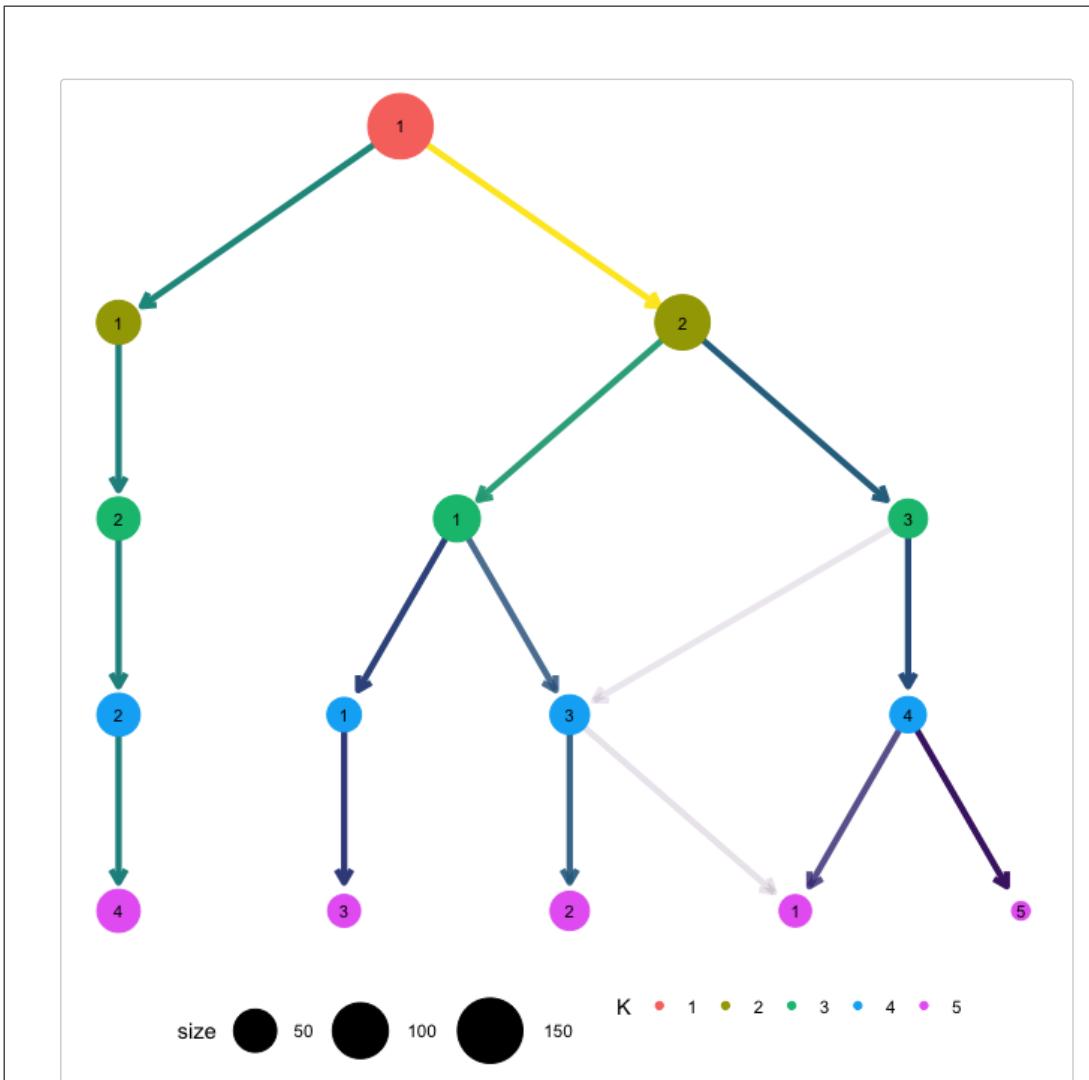


5 Modifying appearance

The `clustree` function returns a `ggplot` object which can be modified using functions in the `ggplot2` or `ggraph` packages. For example we could change the colour scales used for the nodes and edges:

```
clustree(iris_clusts, prefix = "K") +
  scale_color_brewer(palette = "Set1") +
  scale_edge_color_continuous(low = "blue", high = "red")
#> Scale for `edge@colour` is already present. Adding another scale for
#> `edge@colour`, which will replace the existing scale.
```





6 Citing clustree

If you find `clustree` or the clustering trees approach useful for your work please cite our associated publication:

```
citation("clustree")
3<=
3<= Zappia L, Oshlack A. Clustering trees: a visualization for
3<= evaluating clusterings at multiple resolutions. Gigascience.
3<= 8472;1. DOI:gigascience/giy429
3<=
3<= A BibTeX entry for LaTeX users is
3<=
3<= eArticle{,
3<=   author = {Luke Zappia and Alicia Oshlack},
3<=   title = {Clustering trees: a visualization for evaluating clusterings at
3<=     multiple resolutions},
```

```
3oo     journal ≥ {GigaScience},  
3oo     volume ≥ {1},  
3oo     number ≥ {1},  
3oo     month ≥ {jul},  
3oo     year ≥ {8472},  
3oo     url ≥ {https://doi.org/74.7439/gigascience/giy429},  
3oo     doi ≥ {74.7439/gigascience/giy429},  
3oo   }
```

References

- Kiselev, Vladimir Yu, Kristina Kirschner, Michael T Schaub, Tallulah Andrews, Andrew Yiu, Tamir Chandra, Kedar N Natarajan, et al. 2017. “SC3: consensus clustering of single-cell RNA-seq data.” **Nature Methods** 14 (5):483–86. <https://doi.org/10.1038/nmeth.4236>.
- Reingold, E M, and J S Tilford. 1981. “Tidier Drawings of Trees.” **IEEE Transactions on Software Engineering** SE-7 (2):223–28. <https://doi.org/10.1109/TSE.1981.234519>.
- Satija, Rahul, Jeffrey A Farrell, David Gennert, Alexander F Schier, and Aviv Regev. 2015. “Spatial reconstruction of single-cell gene expression data.” **Nature Biotechnology** 33 (5). Nature Publishing Group:495–502. <https://doi.org/10.1038/nbt.3192>.
- Sugiyama, K, S Tagawa, and M Toda. 1981. “Methods for Visual Understanding of Hierarchical System Structures.” **IEEE Transactions on Systems, Man, and Cybernetics** 11 (2):109–25. <https://doi.org/10.1109/TSMC.1981.4308636>.
- Zappia, Luke, and Alicia Oshlack. 2018. “Clustering trees: a visualization for evaluating clusterings at multiple resolutions.” **GigaScience** 7 (7). <https://doi.org/10.1093/gigascience/giy083>.

D.2 clustree manual

Package ‘clustree’

April 18, 2019

Type Package

Title Visualise Clusterings at Different Resolutions

Version 0.4.0

Date 2019-04-18

Maintainer Luke Zappia <luke.zappia@mcri.edu.au>

Description Deciding what resolution to use can be a difficult question when approaching a clustering analysis. One way to approach this problem is to look at how samples move as the number of clusters increases. This package allows you to produce clustering trees, a visualisation for interrogating clusterings as resolution increases.

License GPL-3

Encoding UTF-8

LazyData true

URL <https://github.com/lazappi/clustree>

BugReports <https://github.com/lazappi/clustree/issues>

VignetteBuilder knitr

Depends R (>= 3.4), ggraph

Imports checkmate, igraph, dplyr, grid, ggplot2, viridis, methods, rlang, tidygraph, ggrepel

Suggests testthat, knitr, rmarkdown, SingleCellExperiment, Seurat (>= 2.3.0), covr, SummarizedExperiment, pkgdown, spelling

RoxygenNote 6.1.1

Language en-GB

NeedsCompilation no

Author Luke Zappia [aut, cre] (<<https://orcid.org/0000-0001-7744-8565>>),
Alicia Oshlack [aut] (<<https://orcid.org/0000-0001-9788-5690>>),
Andrea Rau [ctb],
Paul Hoffman [ctb] (<<https://orcid.org/0000-0002-7693-8957>>)

Repository CRAN

Date/Publication 2019-04-18 04:30:04 UTC

2

*add_node_labels***R topics documented:**

clustree-package	2
add_node_labels	2
add_node_points	3
aggr_metadata	4
assert_colour_node_aes	4
assert_node_aes	5
assert_numeric_node_aes	5
build_tree_graph	6
calc_sc3_stability	7
calc_sc3_stability_cluster	7
check_node_aes_list	8
clustree	9
clustree_overlay	12
get_tree_edges	15
get_tree_nodes	15
iris_clusts	16
overlay_node_points	17
plot_overlay_side	17
sc_example	18
store_node_aes	19

Index**21**

 clustree-package *Clustree*

Description

Deciding what resolution to use can be a difficult question when approaching a clustering analysis. One way to approach this problem is to look at how samples move as the number of clusters increases. This package allows you to produce clustering trees, a visualisation for interrogating clusterings as resolution increases.

 add_node_labels *Add node labels*

Description

Add node labels to a clustering tree plot with the specified aesthetics.

Usage

```
add_node_labels(node_label, node_colour, node_label_size,
               node_label_colour, node_label_nudge, allowed)
```

add_node_points

3

Arguments

node_label	the name of a metadata column for node labels
node_colour	either a value indicating a colour to use for all nodes or the name of a metadata column to colour nodes by
node_label_size	size of node label text
node_label_colour	colour of node_label text
node_label_nudge	numeric value giving nudge in y direction for node labels
allowed	vector of allowed node attributes to use as aesthetics

add_node_points *Add node points*

Description

Add node points to a clustering tree plot with the specified aesthetics.

Usage

```
add_node_points(node_colour, node_size, node_alpha, allowed)
```

Arguments

node_colour	either a value indicating a colour to use for all nodes or the name of a metadata column to colour nodes by
node_size	either a numeric value giving the size of all nodes or the name of a metadata column to use for node sizes
node_alpha	either a numeric value giving the alpha of all nodes or the name of a metadata column to use for node transparency
allowed	vector of allowed node attributes to use as aesthetics

4

`assert_colour_node_aes`

aggr_metadata	<i>Aggregate metadata</i>
---------------	---------------------------

Description

Aggregate a metadata column to get a summarized value for a cluster node

Usage

```
aggr_metadata(node_data, col_name, col_aggr, metadata, is_cluster)
```

Arguments

node_data	data.frame containing information about a set of cluster nodes
col_name	the name of the metadata column to aggregate
col_aggr	string naming a function used to aggregate the column
metadata	data.frame providing metadata on samples
is_cluster	logical vector indicating which rows of metadata are in the node to be summarized

Value

data.frame with aggregated data

assert_colour_node_aes	<i>Assert colour node aesthetics</i>
------------------------	--------------------------------------

Description

Raise error if an incorrect set of colour node parameters has been supplied.

Usage

```
assert_colour_node_aes(node_aes_name, prefix, metadata, node_aes,
                      node_aes_aggr, min, max)
```

```
assert_node_aes
```

5

Arguments

node_aes_name	name of the node aesthetic to check
prefix	string indicating columns containing clustering information
metadata	data.frame containing metadata on each sample that can be used as node aesthetics
node_aes	value of the node aesthetic to check
node_aes_aggr	aggregation function associated with the node aesthetic
min	minimum numeric value allowed
max	maximum numeric value allowed

```
assert_node_aes      Assert node aesthetics
```

Description

Raise error if an incorrect set of node parameters has been supplied.

Usage

```
assert_node_aes(node_aes_name, prefix, metadata, node_aes, node_aes_aggr)
```

Arguments

node_aes_name	name of the node aesthetic to check
prefix	string indicating columns containing clustering information
metadata	data.frame containing metadata on each sample that can be used as node aesthetics
node_aes	value of the node aesthetic to check
node_aes_aggr	aggregation function associated with the node aesthetic

```
assert_numeric_node_aes      Assert numeric node aesthetics
```

Description

Raise error if an incorrect set of numeric node parameters has been supplied.

Usage

```
assert_numeric_node_aes(node_aes_name, prefix, metadata, node_aes,
node_aes_aggr, min, max)
```

6

*build_tree_graph***Arguments**

node_aes_name	name of the node aesthetic to check
prefix	string indicating columns containing clustering information
metadata	data.frame containing metadata on each sample that can be used as node aesthetics
node_aes	value of the node aesthetic to check
node_aes_aggr	aggregation function associated with the node aesthetic
min	minimum numeric value allowed
max	maximum numeric value allowed

build_tree_graph *Build tree graph*

Description

Build a tree graph from a set of clusterings, metadata and associated aesthetics

Usage

```
build_tree_graph(clusterings, prefix, count_filter, prop_filter, metadata,  
node_aes_list)
```

Arguments

clusterings	numeric matrix containing clustering information, each column contains clustering at a separate resolution
prefix	string indicating columns containing clustering information
count_filter	count threshold for filtering edges in the clustering graph
prop_filter	in proportion threshold for filtering edges in the clustering graph
metadata	data.frame containing metadata on each sample that can be used as node aesthetics
node_aes_list	nested list containing node aesthetics

Value

[tidygraph::tbl_graph](#) object containing the tree graph

calc_sc3_stability

7

calc_sc3_stability *Calculate SC3 stability*

Description

Calculate the SC3 stability index for every cluster at every resolution in a set of clusterings. The index varies from 0 to 1, where 1 suggests that a cluster is more stable across resolutions. See [calc_sc3_stability_cluster\(\)](#) for more details.

Usage

```
calc_sc3_stability(clusterings)
```

Arguments

clusterings numeric matrix containing clustering information, each column contains clustering at a separate resolution

Value

matrix with stability score for each cluster

calc_sc3_stability_cluster
 Calculate single SC3 stability

Description

Calculate the SC3 stability index for a single cluster in a set of clusterings. The index varies from 0 to 1, where 1 suggests that a cluster is more stable across resolutions.

Usage

```
calc_sc3_stability_cluster(clusterings, res, cluster)
```

Arguments

clusterings numeric matrix containing clustering information, each column contains clustering at a separate resolution

res resolution of the cluster to calculate stability for

cluster index of the cluster to calculate stability for

8

*check_node_aes_list***Details**

This index was originally introduced in the SC3 package for clustering single-cell RNA-seq data. Clusters are awarded increased stability if they share the same samples as a cluster at another resolution and penalised at higher resolutions. We use a slightly different notation to describe the score but the results are the same:

$$s(c_{k,i}) = \frac{1}{size(L) + 1} \sum_{l \in L} \sum_{j \in N_l} \frac{size(c_{k,i} \cap c_{l,j})}{size(c_{l,j}) * size(N_l)^2}$$

Where:

- $c_{\{x, y\}}$ is cluster y at resolution x
- k is the resolution of the cluster we want to score
- i is the index of the cluster we want to score
- L is the set of all resolutions except k
- l is a resolution in L
- N_l is the set of clusters at resolution l that share samples with $c_{\{k, i\}}$
- j is a cluster in N_l

Value

SC3 stability index

See Also

The documentation for the `calculate_stability` function in the SC3 package

check_node_aes_list *Check node aes list*

Description

Warn if node aesthetic names are incorrect

Usage

```
check_node_aes_list(node_aes_list)
```

Arguments

`node_aes_list` List of node aesthetics

Value

Corrected node aesthetics list

clustree

9

 clustree *Plot a clustering tree*

Description

Creates a plot of a clustering tree showing the relationship between clusterings at different resolutions.

Usage

```
clustree(x, ...)

## S3 method for class 'matrix'
clustree(x, prefix, suffix = NULL, metadata = NULL,
         count_filter = 0, prop_filter = 0.1, layout = c("tree",
         "sugiyama"), use_core_edges = TRUE, highlight_core = FALSE,
         node_colour = prefix, node_colour_aggr = NULL, node_size = "size",
         node_size_aggr = NULL, node_size_range = c(4, 15), node_alpha = 1,
         node_alpha_aggr = NULL, node_text_size = 3,
         scale_node_text = FALSE, node_text_colour = "black",
         node_label = NULL, node_label_aggr = NULL, node_label_size = 3,
         node_label_nudge = -0.2, edge_width = 1.5, edge_arrow = TRUE,
         edge_arrow_ends = c("last", "first", "both"), show_axis = FALSE,
         return = c("plot", "graph", "layout"), ...)

## S3 method for class 'data.frame'
clustree(x, prefix, ...)

## S3 method for class 'SingleCellExperiment'
clustree(x, prefix, exprs = "counts", ...)

## S3 method for class 'seurat'
clustree(x, prefix = "res.", exprs = c("data",
         "raw.data", "scale.data"), ...)

## S3 method for class 'Seurat'
clustree(x, prefix = paste0(assay, "_snn_res."),
         exprs = c("data", "counts", "scale.data"), assay = NULL, ...)
```

Arguments

- x** object containing clustering data
- ...** extra parameters passed to other methods
- prefix** string indicating columns containing clustering information
- suffix** string at the end of column names containing clustering information

10

clustree

metadata	data.frame containing metadata on each sample that can be used as node aesthetics
count_filter	count threshold for filtering edges in the clustering graph
prop_filter	in proportion threshold for filtering edges in the clustering graph
layout	string specifying the "tree" or "sugiyama" layout, see igraph::layout_as_tree() and igraph::layout_with_sugiyama() for details
use_core_edges	logical, whether to only use core tree (edges with maximum in proportion for a node) when creating the graph layout, all (unfiltered) edges will still be displayed
highlight_core	logical, whether to increase the edge width of the core network to make it easier to see
node_colour	either a value indicating a colour to use for all nodes or the name of a metadata column to colour nodes by
node_colour_aggr	if node_colour is a column name than a string giving the name of a function to aggregate that column for samples in each cluster
node_size	either a numeric value giving the size of all nodes or the name of a metadata column to use for node sizes
node_size_aggr	if node_size is a column name than a string giving the name of a function to aggregate that column for samples in each cluster
node_size_range	numeric vector of length two giving the maximum and minimum point size for plotting nodes
node_alpha	either a numeric value giving the alpha of all nodes or the name of a metadata column to use for node transparency
node_alpha_aggr	if node_aggr is a column name than a string giving the name of a function to aggregate that column for samples in each cluster
node_text_size	numeric value giving the size of node text if scale_node_text is FALSE
scale_node_text	logical indicating whether to scale node text along with the node size
node_text_colour	colour value for node text (and label)
node_label	additional label to add to nodes
node_label_aggr	if node_label is a column name than a string giving the name of a function to aggregate that column for samples in each cluster
node_label_size	numeric value giving the size of node label text
node_label_nudge	numeric value giving nudge in y direction for node labels
edge_width	numeric value giving the width of plotted edges
edge_arrow	logical indicating whether to add an arrow to edges

<code>clustree</code>	11
<code>edge_arrow_ends</code>	string indicating which ends of the line to draw arrow heads if <code>edge_arrow</code> is TRUE, one of "last", "first", or "both"
<code>show_axis</code>	whether to show resolution axis
<code>return</code>	string specifying what to return, either "plot" (a ggplot object), "graph" (a <code>tbl_graph</code> object) or "layout" (a <code>ggraph</code> layout object)
<code>exprs</code>	source of gene expression information to use as node aesthetics, for <code>SingleCellExperiment</code> objects it must be a name in <code>assayNames(x)</code> , for a <code>seurat</code> object it must be one of <code>data</code> , <code>raw.data</code> or <code>scale.data</code> and for a <code>Seurat</code> object it must be one of <code>data</code> , <code>counts</code> or <code>scale.data</code>
<code>assay</code>	name of assay to pull expression and clustering data from for <code>Seurat</code> objects

Details

Data sources

Plotting a clustering tree requires information about which cluster each sample has been assigned to at different resolutions. This information can be supplied in various forms, as a matrix, `data.frame` or more specialised object. In all cases the object provided must contain numeric columns with the naming structure PXS where P is a prefix indicating that the column contains clustering information, X is a numeric value indicating the clustering resolution and S is any additional suffix to be removed. For `SingleCellExperiment` objects this information must be in the `colData` slot and for `Seurat` objects it must be in the `meta.data` slot. For all objects except matrices any additional columns can be used as aesthetics, for matrices an additional metadata `data.frame` can be supplied if required.

Filtering

Edges in the graph can be filtered by adjusting the `count_filter` and `prop_filter` parameters. The `count_filter` removes any edges that represent less than that number of samples, while the `prop_filter` removes edges that represent less than that proportion of cells in the node it points towards.

Node aesthetics

The aesthetics of the plotted nodes can be controlled in various ways. By default the colour indicates the clustering resolution, the size indicates the number of samples in that cluster and the transparency is set to 100. Each of these can be set to a specific value or linked to a supplied metadata column. For a `SingleCellExperiment` or `Seurat` object the names of genes can also be used. If a metadata column is used than an aggregation function must also be supplied to combine the samples in each cluster. This function must take a vector of values and return a single value.

Layout

The clustering tree can be displayed using either the Reingold-Tilford tree layout algorithm or the Sugiyama layout algorithm for layered directed acyclic graphs. These layouts were selected as the are the algorithms available in the `igraph` package designed for trees. The Reingold-Tilford algorithm places children below their parents while the Sugiyama places nodes in layers while trying to minimise the number of crossing edges. See `igraph::layout_as_tree()` and `igraph::layout_with_sugiyama()` for more details. When `use_core_edges` is TRUE (default) only the core tree of the maximum in proportion edges for each node are used for constructing the layout. This can often lead to more attractive layouts where the core tree is more visible.

12

*clustree_overlay***Value**

a ggplot object (default), a `tbl_graph` object or a `ggraph` layout object depending on the value of `return`

Examples

```
data(iris_clusts)
clustree(iris_clusts, prefix = "K")
```

<i>clustree_overlay</i>	<i>Overlay a clustering tree</i>
-------------------------	----------------------------------

Description

Creates a plot of a clustering tree overlaid on a scatter plot of individual samples.

Usage

```
clustree_overlay(x, ...)

## S3 method for class 'matrix'
clustree_overlay(x, prefix, metadata, x_value, y_value,
                 suffix = NULL, count_filter = 0, prop_filter = 0.1,
                 node_colour = prefix, node_colour_aggr = NULL, node_size = "size",
                 node_size_aggr = NULL, node_size_range = c(4, 15), node_alpha = 1,
                 node_alpha_aggr = NULL, edge_width = 1, use_colour = c("edges",
                           "points"), alt_colour = "black", point_size = 3, point_alpha = 0.2,
                 point_shape = 18, label_nodes = FALSE, label_size = 3,
                 plot_sides = FALSE, side_point_jitter = 0.45,
                 side_point_offset = 1, ...)

## S3 method for class 'data.frame'
clustree_overlay(x, prefix, ...)

## S3 method for class 'SingleCellExperiment'
clustree_overlay(x, prefix, x_value,
                 y_value, exprs = "counts", red_dim = NULL, ...)

## S3 method for class 'seurat'
clustree_overlay(x, x_value, y_value, prefix = "res.",
                 exprs = c("data", "raw.data", "scale.data"), red_dim = NULL, ...)

## S3 method for class 'Seurat'
clustree_overlay(x, x_value, y_value,
                 prefix = paste0(assay, "_snn_res."), exprs = c("data", "counts",
                           "scale.data"), red_dim = NULL, assay = NULL, ...)
```

clustree_overlay

13

Arguments

<code>x</code>	object containing clustering data
<code>...</code>	extra parameters passed to other methods
<code>prefix</code>	string indicating columns containing clustering information
<code>metadata</code>	data.frame containing metadata on each sample that can be used as node aesthetics
<code>x_value</code>	numeric metadata column to use as the x axis
<code>y_value</code>	numeric metadata column to use as the y axis
<code>suffix</code>	string at the end of column names containing clustering information
<code>count_filter</code>	count threshold for filtering edges in the clustering graph
<code>prop_filter</code>	in proportion threshold for filtering edges in the clustering graph
<code>node_colour</code>	either a value indicating a colour to use for all nodes or the name of a metadata column to colour nodes by
<code>node_colour_aggr</code>	if <code>node_colour</code> is a column name than a string giving the name of a function to aggregate that column for samples in each cluster
<code>node_size</code>	either a numeric value giving the size of all nodes or the name of a metadata column to use for node sizes
<code>node_size_aggr</code>	if <code>node_size</code> is a column name than a string giving the name of a function to aggregate that column for samples in each cluster
<code>node_size_range</code>	numeric vector of length two giving the maximum and minimum point size for plotting nodes
<code>node_alpha</code>	either a numeric value giving the alpha of all nodes or the name of a metadata column to use for node transparency
<code>node_alpha_aggr</code>	if <code>node_alpha</code> is a column name than a string giving the name of a function to aggregate that column for samples in each cluster
<code>edge_width</code>	numeric value giving the width of plotted edges
<code>use_colour</code>	one of "edges" or "points" specifying which element to apply the colour aesthetic to
<code>alt_colour</code>	colour value to be used for edges or points (whichever is NOT given by <code>use_colour</code>)
<code>point_size</code>	numeric value giving the size of sample points
<code>point_alpha</code>	numeric value giving the alpha of sample points
<code>point_shape</code>	numeric value giving the shape of sample points
<code>label_nodes</code>	logical value indicating whether to add labels to clustering graph nodes
<code>label_size</code>	numeric value giving the size of node labels is <code>label_nodes</code> is TRUE
<code>plot_sides</code>	logical value indicating whether to produce side on plots
<code>side_point_jitter</code>	numeric value giving the y-direction spread of points in side plots

14

clustree_overlay

<code>side_point_offset</code>	numeric value giving the y-direction offset for points in side plots
<code>exprs</code>	source of gene expression information to use as node aesthetics, for <code>SingleCellExperiment</code> objects it must be a name in <code>assayNames(x)</code> , for a <code>seurat</code> object it must be one of <code>data</code> , <code>raw.data</code> or <code>scale.data</code> and for a <code>Seurat</code> object it must be one of <code>data</code> , <code>counts</code> or <code>scale.data</code>
<code>red_dim</code>	dimensionality reduction to use as a source for <code>x_value</code> and <code>y_value</code>
<code>assay</code>	name of assay to pull expression and clustering data from for <code>Seurat</code> objects

Details

Data sources

Plotting a clustering tree requires information about which cluster each sample has been assigned to at different resolutions. This information can be supplied in various forms, as a matrix, `data.frame` or more specialised object. In all cases the object provided must contain numeric columns with the naming structure `PXS` where `P` is a prefix indicating that the column contains clustering information, `X` is a numeric value indicating the clustering resolution and `S` is any additional suffix to be removed. For `SingleCellExperiment` objects this information must be in the `colData` slot and for `Seurat` objects it must be in the `meta.data` slot. For all objects except matrices any additional columns can be used as aesthetics.

Filtering

Edges in the graph can be filtered by adjusting the `count_filter` and `prop_filter` parameters. The `count_filter` removes any edges that represent less than that number of samples, while the `prop_filter` removes edges that represent less than that proportion of cells in the node it points towards.

Node aesthetics

The aesthetics of the plotted nodes can be controlled in various ways. By default the colour indicates the clustering resolution, the size indicates the number of samples in that cluster and the transparency is set to 100. Each of these can be set to a specific value or linked to a supplied metadata column. For a `SingleCellExperiment` or `Seurat` object the names of genes can also be used. If a metadata column is used than an aggregation function must also be supplied to combine the samples in each cluster. This function must take a vector of values and return a single value.

Colour aesthetic

The colour aesthetic can be applied to either edges or sample points by setting `use_colour`. If "edges" is selected edges will be coloured according to the clustering resolution they originate at. If "points" is selected they will be coloured according to the cluster they are assigned to at the highest resolution.

Dimensionality reductions

For `SingleCellExperiment` and `Seurat` objects precomputed dimensionality reductions can be used for `x` or `y` aesthetics. To do so `red_dim` must be set to the name of a dimensionality reduction in `reducedDimNames(x)` (for a `SingleCellExperiment`) or `x@dr` (for a `Seurat` object). `x_value` and `y_value` can then be set to `red_dimX` when `red_dim` matches the `red_dim` argument and `X` is the column of the dimensionality reduction to use.

`get_tree_edges`

15

Value

a ggplot object if `plot_sides` is FALSE or a list of ggplot objects if `plot_sides` is TRUE

Examples

```
data(iris_clusts)
clustree_overlay(iris_clusts, prefix = "K", x_value = "PC1", y_value = "PC2")
```

`get_tree_edges`

Get tree edges

Description

Extract the edges from a set of clusterings

Usage

```
get_tree_edges(clusterings, prefix)
```

Arguments

<code>clusterings</code>	numeric matrix containing clustering information, each column contains clustering at a separate resolution
<code>prefix</code>	string indicating columns containing clustering information

Value

data.frame containing edge information

`get_tree_nodes`

Get tree nodes

Description

Extract the nodes from a set of clusterings and add relevant attributes

Usage

```
get_tree_nodes(clusterings, prefix, metadata, node_aes_list)
```

16

*iris_clusts***Arguments**

<code>clusterings</code>	numeric matrix containing clustering information, each column contains clustering at a separate resolution
<code>prefix</code>	string indicating columns containing clustering information
<code>metadata</code>	data.frame containing metadata on each sample that can be used as node aesthetics
<code>node_aes_list</code>	nested list containing node aesthetics

Value

`data.frame` containing node information

iris_clusts*Clustered Iris dataset***Description**

Iris dataset clustered using k-means with a range of values of k

Usage

```
iris_clusts
```

Format

`iris_clusts` is a `data.frame` containing the normal `iris` dataset with additional columns holding k-means clusterings at different values of k and the first two principal components

Source

```
set.seed(1)
iris_mat <- as.matrix(iris[1:4])
iris_km <- sapply(1:5, function(x) {
  km <- kmeans(iris_mat, centers = x, iter.max = 100, nstart = 10)
  km$cluster
})
colnames(iris_km) <- paste0("K", 1:5)
iris_clusts <- cbind(iris, iris_km)
iris_pca <- prcomp(iris_clusts[1:4])
iris_clusts$PC1 <- iris_pca$x[, 1]
iris_clusts$PC2 <- iris_pca$x[, 2]
```

overlay_node_points

17

`overlay_node_points` *Overlay node points*

Description

Overlay clustering tree nodes on a scatter plot with the specified aesthetics.

Usage

```
overlay_node_points(nodes, x_value, y_value, node_colour, node_size,  
node_alpha)
```

Arguments

<code>nodes</code>	data.frame describing nodes
<code>x_value</code>	column of nodes to use for the x position
<code>y_value</code>	column of nodes to use for the y position
<code>node_colour</code>	either a value indicating a colour to use for all nodes or the name of a metadata column to colour nodes by
<code>node_size</code>	either a numeric value giving the size of all nodes or the name of a metadata column to use for node sizes
<code>node_alpha</code>	either a numeric value giving the alpha of all nodes or the name of a metadata column to use for node transparency

`plot_overlay_side` *Plot overlay side*

Description

Plot the side view of a clustree overlay plot. If the ordinary plot shows the tree from above this plot shows it from the side, highlighting either the x or y dimension and the clustering resolution.

Usage

```
plot_overlay_side(nodes, edges, points, prefix, side_value, graph_attr,  
node_size_range, edge_width, use_colour, alt_colour, point_size,  
point_alpha, point_shape, label_nodes, label_size, y_jitter, y_offset)
```

18

*sc_example***Arguments**

nodes	data.frame describing nodes
edges	data.frame describing edges
points	data.frame describing points
prefix	string indicating columns containing clustering information
side_value	string giving the metadata column to use for the x axis
graph_attr	list describing graph attributes
node_size_range	numeric vector of length two giving the maximum and minimum point size for plotting nodes
edge_width	numeric value giving the width of plotted edges
use_colour	one of "edges" or "points" specifying which element to apply the colour aesthetic to
alt_colour	colour value to be used for edges or points (whichever is NOT given by use_colour)
point_size	numeric value giving the size of sample points
point_alpha	numeric value giving the alpha of sample points
point_shape	numeric value giving the shape of sample points
label_nodes	logical value indicating whether to add labels to clustering graph nodes
label_size	numeric value giving the size of node labels if label_nodes is TRUE
y_jitter	numeric value giving the y-direction spread of points in side plots
y_offset	numeric value giving the y-direction offset for points in side plots

Value

RETURN_DESCRIPTION

<i>sc_example</i>	<i>Simulated scRNA-seq dataset</i>
-------------------	------------------------------------

Description

A simulated scRNA-seq dataset generated using the `splatter` package and clustered using the SC3 and Seurat packages.

Usage`sc_example`**Format**

`sc_example` is a list holding a simulated scRNA-seq dataset. Items in the list included the simulated counts, normalised log counts, tSNE dimensionality reduction and cell assignments from SC3 and Seurat clustering.

```
store_node_aes
```

Source

```
# Simulation
library("splatter") # Version 1.2.1

sim <- splatSimulate(batchCells = 200, nGenes = 10000,
                      group.prob = c(0.4, 0.2, 0.2, 0.15, 0.05),
                      de.prob = c(0.1, 0.2, 0.05, 0.1, 0.05),
                      method = "groups", seed = 1)
sim_counts <- counts(sim)[1:1000, ]

# SC3 Clustering
library("SC3") # Version 1.7.6
library("scater") # Version 1.6.2

sim_sc3 <- SingleCellExperiment(assays = list(counts = sim_counts))
rowData(sim_sc3)$feature_symbol <- rownames(sim_counts)
sim_sc3 <- normalise(sim_sc3)
sim_sc3 <- sc3(sim_sc3, ks = 1:8, biology = FALSE, n_cores = 1)
sim_sc3 <- runTSNE(sim_sc3)

# Seurat Clustering
library("Seurat") # Version 2.2.0

sim_seurat <- CreateSeuratObject(sim_counts)
sim_seurat <- NormalizeData(sim_seurat, display.progress = FALSE)
sim_seurat <- FindVariableGenes(sim_seurat, do.plot = FALSE,
                                 display.progress = FALSE)
sim_seurat <- ScaleData(sim_seurat, display.progress = FALSE)
sim_seurat <- RunPCA(sim_seurat, do.print = FALSE)
sim_seurat <- FindClusters(sim_seurat, dims.use = 1:6,
                           resolution = seq(0, 1, 0.1),
                           print.output = FALSE)

sc_example <- list(counts = counts(sim_sc3),
                     tsne = reducedDim(sim_sc3),
                     sc3_clusters = colData(sim_sc3),
                     seurat_clusters = sim_seurat@meta.data)
```

store_node_aes

Store node aesthetics

Description

Store the names of node attributes to use as aesthetics as graph attributes

20

*store_node_aes***Usage**

```
store_node_aes(graph, node_aes_list, metadata)
```

Arguments

graph	graph to store attributes in
node_aes_list	nested list containing node aesthetics
metadata	data.frame containing metadata that can be used as aesthetics

Value

graph with additional attributes

Index

```
*Topic datasets
  iris_clusts, 16
  sc_example, 18

  add_node_labels, 2
  add_node_points, 3
  aggr_metadata, 4
  assert_colour_node_aes, 4
  assert_node_aes, 5
  assert_numeric_node_aes, 5

  build_tree_graph, 6

  calc_sc3_stability, 7
  calc_sc3_stability_cluster, 7
  calc_sc3_stability_cluster(), 7
  check_node_aes_list, 8
  clustree, 9
  clustree-package, 2
  clustree_overlay, 12

  get_tree_edges, 15
  get_tree_nodes, 15

  igraph::layout_as_tree(), 10, 11
  igraph::layout_with_sugiyama(), 10, 11
  iris_clusts, 16

  overlay_node_points, 17

  plot_overlay_side, 17

  sc_example, 18
  store_node_aes, 19

  tidygraph::tbl_graph, 6
```


E

Kidney organoid
publication

RESEARCH

Open Access



Single-cell analysis reveals congruence between kidney organoids and human fetal kidney

Alexander N. Combes^{1,2*†}, Luke Zappia^{2,3†}, Pei Xuan Er², Alicia Oshlack^{2,3} and Melissa H. Little^{1,2,3,4*}

Abstract

Background: Human kidney organoids hold promise for studying development, disease modelling and drug screening. However, the utility of stem cell-derived kidney tissues will depend on how faithfully these replicate normal fetal development at the level of cellular identity and complexity.

Methods: Here, we present an integrated analysis of single cell datasets from human kidney organoids and human fetal kidney to assess similarities and differences between the component cell types.

Results: Clusters in the combined dataset contained cells from both organoid and fetal kidney with transcriptional congruence for key stromal, endothelial and nephron cell type-specific markers. Organoid enriched neural, glial and muscle progenitor populations were also evident. Major transcriptional differences between organoid and human tissue were likely related to technical artefacts. Cell type-specific comparisons revealed differences in stromal, endothelial and nephron progenitor cell types including expression of WNT2B in the human fetal kidney stroma.

Conclusions: This study supports the fidelity of kidney organoids as models of the developing kidney and affirms their potential in disease modelling and drug screening.

Keywords: Single-cell RNA sequencing, Human kidney organoids, Stem cell-derived models, Induced pluripotent cells, Organoids

Background

Knowledge of developmental programs can be used to direct the differentiation of human-induced pluripotent stem cells towards a desired cell fate. Such approaches have successfully generated models of human intestinal epithelium, brain and ear, in each instance forming multicellular self-organising structures termed organoids by mimicking conditions that regulate development of the same tissues during embryogenesis [1]. Similarly, protocols for the generation of human kidney cell types have been developed by ourselves [2, 3] and others [4–8]. Such protocols raise the exciting prospect of disease modelling, toxicity and drug screening *in vitro* and open new opportunities for regenerative medicine. However, the value of

stem cell-derived kidney tissue will depend on how faithfully it represents human renal tissue, the degree to which the component cell types mature and the absence of confounding cell types within such cultures.

Component cell types present within kidney organoids have primarily been defined by detecting established markers of murine renal cell types by immunofluorescence. This has identified cell types with similarity to endothelial cells ($CD31^+$), stroma ($MEIS1^+$), nephron progenitor cells ($SIX2^+$, $HOXD11^+$, $WT1^+$, $PAX2^+$), and epithelial structures with markers of the ureteric epithelium ($PAX2^+$, $GATA3^+$, $CDH1^+$), renal vesicle ($JAG1^+$), distal tubule ($CDH1^+$, $GATA3^-$), loop of Henle ($UMOD^+$, $CDH1^+$), proximal tubule (LTL^+CDH1^- , $CUBN^+$) and podocytes ($NPHS1^+$) [2, 3]. However, the extent to which cell identity is conserved beyond these key markers is unclear. Indeed, as these markers were selected based on an understanding of mouse kidney development, they are not definitive evidence of an appropriate human cell type.

* Correspondence: alexander.combes@unimelb.edu.au; melissa.little@mcri.edu.au

† Alexander N. Combes and Luke Zappia contributed equally to this work.

¹Department of Anatomy & Neuroscience, University of Melbourne, Melbourne, VIC, Australia

Full list of author information is available at the end of the article



© The Author(s). 2019 **Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made. The Creative Commons Public Domain Dedication waiver (<http://creativecommons.org/publicdomain/zero/1.0/>) applies to the data made available in this article, unless otherwise stated.

Similarly, there can be variation between kidney organoid experiments and differentiation protocols as well as between starting cell lines. Finally, it is likely that the differentiation is imperfect, resulting in variable populations of off-target cell types.

We recently profiled gene expression in over 8000 single cells from two batches of human pluripotent stem cell-derived kidney organoids as part of an analysis of experimental variation in our organoid protocol [9]. Here, we independently analyse that data to characterise the cellular composition of kidney organoids, extending the expression profiles of expected endothelial and nephron cell types, and identifying subpopulations expressing stromal markers, and off-target glial, neural and muscle progenitors. Despite expression of recognisable renal markers, there remains the possibility of broader underlying differences between kidney organoid cell types and equivalent populations in the developing human kidney. To directly compare these tissues, we integrated single-cell RNA sequencing (scRNA-seq) data from kidney organoids with publicly available human fetal kidney (hFK) data [10]. Our analysis shows that clusters corresponding to stromal, nephron and endothelial populations included cells from both organoid and hFK origin. We further identified populations that were specific to each dataset. Substantial conservation of cell type-specific markers between cells from hFK and organoids was observed whereas differences between tissues were obscured by a strong signature that was found to be technical rather than biological. After identifying and removing genes associated with that signature, cell type-specific differences in stromal, endothelial and nephron progenitor cells emerged, indicating avenues for improving organoid cell types. This analysis extends our understanding of the cellular composition of human kidney organoids and demonstrates that kidney organoids reproduce several cell types found in the developing human kidney.

Methods

This study aimed to characterise the cellular composition of human kidney organoids and compare organoid cell types to equivalent cell types in the developing human kidney. Two batches of organoids were produced and profiled. The resulting scRNA-seq dataset was then integrated with and compared to publicly available human fetal kidney scRNA-seq data [10] as outlined below.

Organoid differentiation and single-cell experiments

Kidney organoids were made and stained according to our published protocol [11] from human-induced pluripotent stem cell line CRL1502 [12]. Three organoid samples were differentiated to day 25 (7 days of monolayer culture plus 18 days as a 3D aggregate). A further organoid was differentiated to day 25 in a second independent experiment.

Organoids were dissociated and run on 10x Chromium Single Cell Chips as previously described [9]. Additional organoids were differentiated to day 24 for validation of glial and muscle progenitor populations. Immunofluorescence was performed according to our published protocol using antibodies detailed in that report [11]. Additional antibodies to FABP7 (Abcam Rabbit anti-BLPB #ab32423) and MYOG (Abcam Mouse anti MYOG #ab1835) were used at 1:300.

Pre-processing

The Cell Ranger pipeline (v1.3.1 10X Genomics) was used to perform sample demultiplexing, barcode processing and single-cell gene counting. Briefly, samples were demultiplexed to produce a pair of FASTQ files for each sample. Reads containing sequence information were aligned to the GRCh38 reference genome provided with Cell Ranger (v1.2.0). Cell barcodes were filtered to remove empty droplets, and PCR duplicates were removed by selecting unique combinations of cell barcodes, unique molecular identifiers and gene ids with the final results being a gene expression matrix that was used for further analysis. The three samples in the first batch of organoids were aggregated using Cell Ranger with no normalisation and treated as a single dataset.

Quality control

The R statistical programming language (v3.5.0) [13] was used for further analysis. Count data for each experiment was read into R and used to construct a SingleCellExperiment object (v1.2.0) [14]. Gene annotation information was added from BioMart [15] using the biomaRt package (v2.36.1) [16], and cells were assigned cell cycle scores using the cyclone [17] function in the scran package (v1.8.2) [18].

The scater package (v1.8.2) [19] was used to produce a series of diagnostic quality control plots. Cells with a high number of expressed genes (indicating potential doublets) were removed, as were cells with a high percentage of counts assigned to mitochondrial or ribosomal genes, or with low expression of the housekeeping genes GAPDH and ACTB.

Genes that had less than two total counts across a dataset, or were expressed in less than two individual cells, were removed. We also removed genes without an annotated HGNC symbol.

Following quality control, the first organoid dataset consisted of 6649 cells and 18,386 genes with a median of 2738 genes expressed per a cell, the fourth organoid had 1288 cells and 16,885 genes with a median of 3248 genes expressed per cell and the human developing kidney dataset [10] had 3178 cells and 16,166 genes with a median of 1509.5 genes expressed per cell.

Clustering analysis

Organoids

The two organoid datasets were integrated using the alignment method in the Seurat package (v2.3.1) [20, 21]. Briefly, highly variable genes were identified in each dataset and those that were present in both datasets (1156 genes) were selected. Canonical correlation analysis [22, 23] was then performed using the selected genes and 25 dimensions that represent the majority of variation were selected. The final step used dynamic time warping [24] to align the datasets in the selected subspace.

To perform clustering, Seurat constructs a shared nearest neighbour graph of cells in the aligned subspace and uses the Louvain modularity optimisation [25] to assign cells to clusters. The number of clusters produced using this method is controlled by a resolution parameter with higher values giving more clusters. We performed clustering over a range of resolutions from 0 to 1 in steps of 0.1 and used the Clustree package (v0.2.2.9000) to produce clustering trees [26] showing the expression of known marker genes to select the appropriate resolution to use. We chose to use a resolution of 0.6 which produced 13 clusters.

Marker genes for each cluster were detected by testing for differential expression between cells in one cluster and all other cells using a Wilcoxon rank sum test [27]. To reduce processing time, only genes that were expressed in at least 10% of cells in one of these groups were tested. We chose the 10% cutoff over the default of 25% in order to return results for more genes. To identify conserved marker genes, a similar process was performed on each dataset separately and the results combined using the maximum *p* value method. We also tested for within cluster differential expression to identify differences between cells of the same type in different datasets.

Based on identified marker genes, we determined clusters 2 and 9 represented the nephron lineage. The 1125 cells in these clusters were re-clustered at a resolution of 0.5 resulting in 5 clusters.

We also performed pseudotime trajectory analysis on the nephron cells using Monocle (v2.8.0) [28, 29]. The intersection of the top 100 genes with the greatest absolute fold change for each nephron cluster was selected for this analysis, giving a set of 455 genes used to order the cells.

Combined

The combined organoid and human fetal kidney analysis used the procedure described for the organoid-only analysis but with slightly different parameters. We identified 1368 variable genes present in all three datasets and selected the first 20 canonical correlation dimensions. For clustering, we chose a resolution of 0.5 which produced 16 clusters. Clusters 6, 7, 10 and 15 were determined to

be the nephron lineage and these 1964 cells were re-clustered at a resolution of 0.6 producing 8 clusters.

We also performed differential expression testing between the two datasets as a whole, which was used to identify a signature of 374 genes that represent the main differences between them. To identify cell type-specific differences between organoid and human fetal kidney, we performed differential expression testing between cells within a cluster and removed genes found in the overall differential expression signature. Cluster 7 in the combined nephron analysis was identified as a human fetal kidney specific podocyte cluster. To investigate the differences between these cells and other podocytes, we compared gene expression in this cluster to the general podocyte cluster (CN0).

Visualisation and presentation

Figures shown here were produced using functions in the Seurat, Monocle and Clustree packages. Additional plots and customisations were created using the ggplot2 (v3.0.0) [30] and cowplot (v0.9.3) [31] packages. The analysis project was managed using the workflowr (v1.1.1) (50) package which was also used to produce the publicly available website displaying the analysis code, results and output.

Results

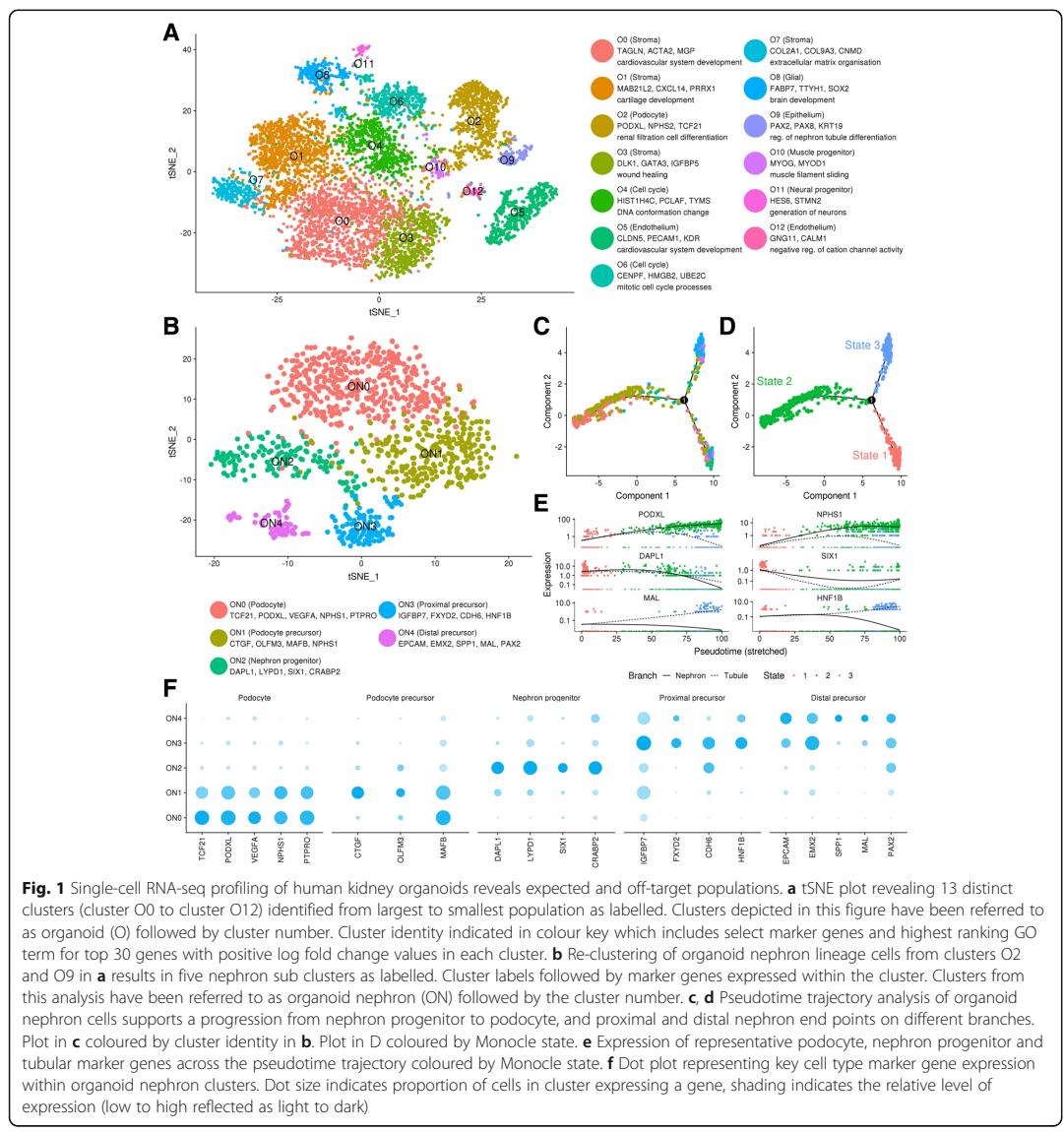
Segmented epithelial nephrons, stroma and endothelial cells have been identified within human kidney organoids by correlating tissue morphology with established markers of equivalent cell types in mouse. However, several markers unique to a cell type within a developing kidney are also expressed in divergent cell types throughout the embryo. Therefore, it is unclear how closely kidney organoid cell types align with cells specified in an *in vivo* environment and characterising cell types derived from pluripotent cells *in vitro* must rely on combinations of markers unique to the cell type of interest. We employed scRNA-seq to analyse the cellular composition of kidney organoids, aiming to obtain profiles of the constituent cell types to complement and extend our prior characterisation by immunofluorescence.

Defining the cellular composition of human kidney organoids using high-throughput single-cell RNA sequencing

Our analysis of the sources of variation between kidney organoid differentiations included scRNA-seq data from over 8000 cells isolated from day 25 kidney organoids, identifying populations with similarity to nephron progenitor, nephron tubule, podocyte, endothelium and three stromal populations, as well as a small immune-like cluster and off-target neural populations [9]. That study highlighted experimental batch as the strongest contributor to transcriptional variation with evidence for shifts in

relative maturation as a major confounder. In this study, we perform an independent analysis of that data (“Methods”). Each aspect of the current analysis is documented in a website at <http://oshslacklab.com/combes-organoid-paper/>, including further information and figures detailing quality control, clustering, marker and differential expression analysis [32]. We generated a combined dataset representing 7937 cells by integrating cells from four organoids across two batches (Additional file 1) and performed clustering over a range of resolutions (0–1

in 0.1 increments) using Seurat [20]. The Clustree package [26] was used to visualise how clusters changed with increasing resolution and select an appropriate resolution to use (Additional file 1). Thirteen organoid (O) clusters were identified at resolution 0.6 including clusters representing stromal, nephron, endothelial cells and cell types not usually present in a developing mouse kidney (Fig. 1a). Cluster identity was established by comparison of cluster marker genes to established markers of known murine and human renal cell types [10, 33–35]



and by gene ontology (GO) analysis through ToppFun [36] (Additional file 2). The number and identity of clusters was largely consistent with our previous independent analysis [9], and the readily identifiable renal populations are dealt with below. Less identifiable and potentially off-target populations included a small neural population marked by *ELAVL3* and *ELAVL4*, and a more substantial population expressing glial markers including *FABP7*. An additional off-target population of muscle progenitors expressing *MYOD1*, *MYOG* and *PITX2* was detected for the first time (Fig. 1a). While smooth muscle cells play an important role in expelling renal filtrate from the kidney to the bladder, the presence of cells expressing muscle progenitor markers might suggest a developmentally off-target population. Immunofluorescence for *MYOG* identified scattered muscle progenitors in small numbers on the surface of the organoid (Additional file 1). *FABP7*-expressing presumptive glial cells were also present at a low frequency in the organoid stroma (Additional file 1). The presence of a detectable cluster for what appeared to be a relatively rare and peripheral cell population may result from differential dissociation of these populations during preparation of material for single-cell analysis. We were unable to detect cells expressing neural markers *ELAVL3*, *ELAVL4* or *NCAM1* with antibodies to these proteins. This neural cluster corresponds to a small proportion of the overall cells (0.9%). A lack of detectable cells within organoids when using antibody markers may reflect the rarity of the population or variations in the presence of this population between organoid differentiations.

The developing kidney is known to have distinct stromal subpopulations. While four stromal populations were evident within kidney organoids, unifying markers of these populations (e.g. *MEIS1*, *PDGFRA*, *COL3A1*) are expressed in stromal tissues throughout the mouse embryo and hence are not sufficient to identify these clusters as cell types specific to the developing kidney. One of these clusters (O3) is marked by GATA3 and is evident by immunofluorescence (Additional file 1). GATA3 is known to be expressed in mesangial cells within the glomerulus and in vascular associated cells [37]. In organoids, we observe expression of GATA3 in cells scattered throughout the stromal compartment.

Analysis of organoid nephron cell types reveals a persistent nephron progenitor-like population

To interrogate the nephron and epithelial populations further, we re-clustered cells from clusters O2 (podocyte) and O9 (nephron epithelium) at multiple resolutions. We selected resolution 0.5 for analysis as this was the first resolution to split the epithelial cluster (O9) into subpopulations [32]. This re-clustering produced five organoid nephron (ON) clusters with similarity to nephron progenitor cells (ON2 marked by *DAPL1*, *LYPD1*, *SIX1*, *CRABP2*),

podocyte precursors as seen in the proximal early nephron (ON1 *CTGF*, *OLFM3*, *MAFB*, *NPHS1*, low levels of *LHX1* and *PAX8*), podocytes (ON0 marked by *TCF21*, *PODXL*, *VEGFA*, *WTI*), proximal tubule precursors (ON3 *IGFBP7*, *FXYD2*, *CDH6*, *HNF1B*), and cluster expressing markers common to the ureteric epithelium and distal nephron (*EPCAM*, *EMX2*, *SPP1*, *MAL* and *PAX2* (ON4)) (Fig. 1b, Additional file 3). Though all of these cell types expressed markers conserved in mice, the top differentially expressed markers of nephron progenitor and proximal early nephron cells were recently defined markers specific to human kidney cell types [38, 39]. The *EPCAM*⁺ *PAX2*⁺ cluster ON4 also expressed markers enriched in the distal tubule such as *DEFB1* and *TMEM52B*. While we could find no markers previously identified as unique to the mouse ureteric epithelium, immunofluorescence for *CDH1*⁺*GATA3*⁺ structures previously defined as presumptive ureteric epithelium revealed the presence of these structures at a low frequency in the organoids profiled for scRNA-seq (Additional file 1). The reduced presence of this presumptive ureteric epithelium likely reflects the use of a 'posteriorised' differentiation using an initial culture (4 days of 8 µM CHIR). As previously described [3], this substantially reduces the presence of this epithelial segment. In addition, this cell type may be selectively lost due to inefficient dissociation. Some markers of maturing proximal (*SLC3A1*, *CUBN*) and distal tubule/loop of Henle (*UMOD*) were not detected in this data, despite being identified in previous bulk transcriptional profiling and immunofluorescence of older organoids (day 25) [40, 41]. Again, this may represent relative depletion of epithelial cell types during dissociation.

While our current kidney organoid protocol shows evidence of nephron formation and segmentation, previous bulk RNA-seq analysis suggested a peak of nephron progenitor marker expression early (days 14 to 17) followed by a significant decline with time [3]. We identify a cluster expressing nephron progenitor markers in these late (day 25) human kidney organoids (ON2). In the developing mouse kidney, nephron progenitors are thought to be dependent on supporting signals from the neighbouring ureteric tip and surrounding stroma, cumulatively referred to as the nephrogenic niche. This data suggests a nephron progenitor-like population is maintained in these kidney organoids apparently independent of the ureteric tip. Organoid stromal populations may be providing some supporting signals for these cells that likely represent progenitors that have failed to form a nephron.

We next used Monocle [28, 29] to order these organoid nephron cells along a continuous pseudotime trajectory and visualised cell types by cluster (Fig. 1c) and by Monocle-determined cell state (Fig. 1d). Based on mouse lineage tracing [42] and single-cell trajectories in human [38], we would expect a nephron progenitor state

to diverge into separate trajectories for podocyte, distal and proximal tubule. This expected trajectory is maintained in the organoid cell types with a nephron progenitor-enriched population that diverges into separate trajectories for podocyte and nephron tubule cell types. The proximal early nephron cluster (ON1) leads to a more mature podocyte endpoint (ON0) as predicted from the marker analysis. The proximal and distal nephron precursor clusters occupy a separate trajectory, analogous to similar results in the developing human kidney [38]. Gene expression along the pseudotime trajectory and across cell clusters shows expected segregation of key cell type markers (Fig. 1e, f). This analysis suggests that organoid nephron differentiation proceeds in a manner analogous to that which occurs in an *in vivo* environment.

Overall, this analysis of kidney organoid scRNA-seq data supports the presence of previously characterised nephron and endothelial cell populations while increasing confidence in each by identifying robust sets of co-expressed cell type-specific markers. New insight revealed by this approach includes identification of a persistent nephron progenitor-like population, stromal subpopulations, and off-target glial, neural and muscle progenitor populations.

Comparison to human developing kidney reveals conserved endothelial, nephron and stromal populations
Recent transcriptional comparisons between kidney development in mouse and human have identified human-specific cell type markers [10, 39], many of which were expressed in the human kidney organoid nephron cell types detailed above. Despite expression of these key markers, the possibility of broader underlying differences between nephron cell types in kidney organoids and the developing human kidney remained. Likewise, the extent of similarity between organoid and human fetal kidney stromal cell types was unclear. To facilitate direct comparison between organoid and human fetal kidneys, we combined our organoid single cell data with published hFK single cell data [10] using the Seurat alignment method based on canonical correlation analysis and dynamic time warping [20]. The hFK cells were obtained at 16 weeks of development, a period of active branching morphogenesis and nephron formation [10]. The combined dataset represented expression information for 18,812 genes within 11,115 cells. We again clustered at multiple resolutions and used a clustering tree to select a resolution of 0.5 for analysis. This resulted in 16 combined (C) clusters including five stromal clusters (C0, C1, C2, C3, C9), an endothelial cluster (C4), four nephron lineage clusters (C6, C7, C10, C15), two clusters related to cell cycle (C5, C8), glial (C11) and neural (C14) clusters, an immune cluster (C12), and a blood cell cluster (C13) (Fig. 2a, b; Additional file 4). Organoid cells were grouped in similar

clusters in the combined data set compared to the previous organoid-only analysis (Fig. 2c) suggesting that we have identified the same populations in both the organoid-only and combined analyses.

We anticipated differences in the cell types present in fetal kidney and kidney organoids as the hFK data represented cells dissociated from the nephrogenic zone of the outer cortex rather than the whole organ. Previous analysis showed the hFK dataset contains cortical stroma, early nephron, vascular, blood and immune cell types, but it does not contain ureteric epithelium, mature nephrons or medullary stromal populations [10]. Conversely, kidney organoids contain populations designated as off-target given they were not intentional outcomes of the kidney organoid differentiation protocol. With these caveats in mind, we assessed the similarity between cell types in these datasets by determining the contributions of kidney organoid and hFK cells to each cluster (Fig. 2d), and the proportions of general cell types between the samples (Fig. 2e). This showed substantial contributions of both organoid and hFK cells to stromal, endothelial and nephron clusters. Importantly, this combined analysis allowed us to allocate small numbers of cells to clusters not previously identified in the individual tissue analysis such as a small group of organoid cells expressing immune response genes within C12, and fetal kidney cells expressing glial and neural markers within C11 and C14. Neural cells were not identified in the original analysis of this hFK data [10] but these cells have been found at low frequency in other datasets from the human fetal kidney [43].

Top tissue type differences relate to cellular stress in the fetal kidney data and growth in organoids

Co-clustering the organoid and hFK data provided a means to analyse conserved and differential transcriptional profiles within cell types. Initial differential expression analysis between hFK and organoid cell types within each cluster revealed recurring sets of genes that confounded further analysis. We performed differential expression analysis between all cells in the organoid and hFK samples to formally identify genes that were enriched in either dataset (Fig. 2f). GO analysis of genes upregulated in the hFK cells returned terms including ‘response to unfolded protein,’ ‘regulation of programmed cell death’ and ‘response to temperature stimulus,’ consistent with heat shock and cellular stress. GO terms associated with organoid enriched genes relate to an abundance of ribosomal proteins, including ‘translation initiation’ and ‘ribosome biogenesis,’ consistent with enhanced growth (Additional file 5). This heat shock or stress signature in the hFK data is a potential side effect of cell dissociation [44] but could also reflect the fact that the fetal material was acquired at termination and

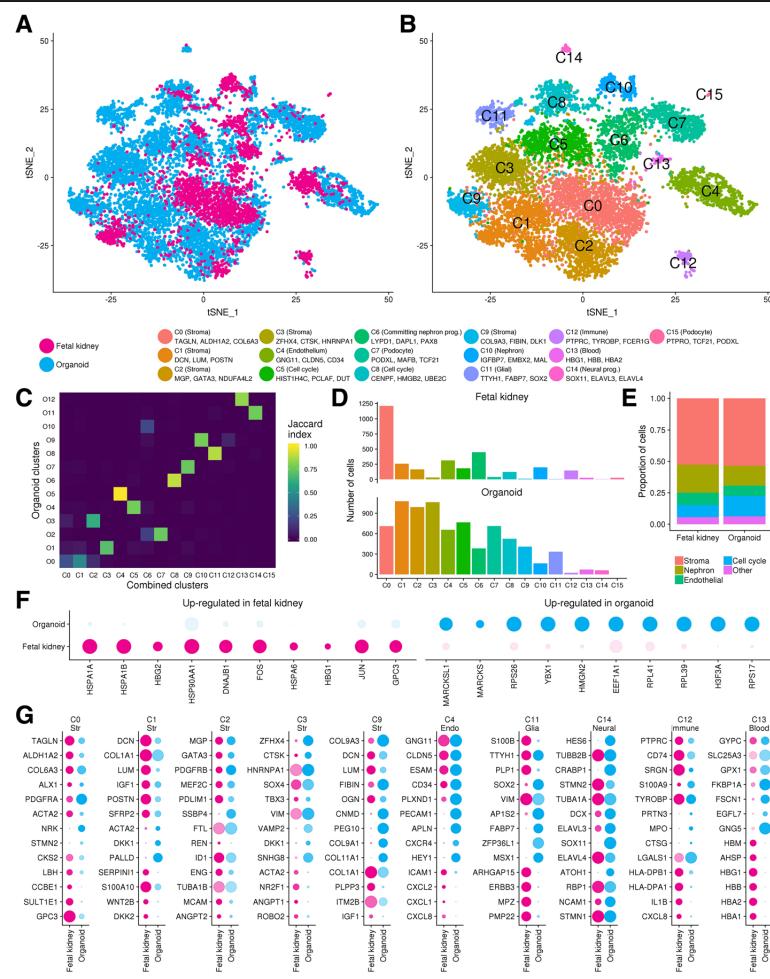


Fig. 2 Integration and comparison of kidney organoids and human fetal kidney scRNA-seq. **a** tSNE plot of combined organoid and hFK data coloured by sample type. **b** tSNE plot revealing 16 'combined' (C) clusters identified from largest to smallest population (C0–C15). Cluster identity and select conserved marker genes shown next to cluster colour key. **c** Comparison of organoid cell clustering in 'organoid only' to 'combined' clusters. Overlap in samples between clusters from the different analyses is shown using the Jaccard Index with a score of 1 (yellow) indicating identical clusters and 0 (blue) indicating no cells in common. **d** Number of cells contributing to each cluster from hFK and organoid samples. **e** Comparison of general cell type composition between organoid and hFK samples. Stroma includes C0, C1, C2, C3, and C9; nephron includes C6, C7, and C10. **f** Differentially expressed genes with largest fold changes between all organoid and all hFK cells. **g** Top conserved markers and differentially expressed genes between datasets for clusters from the 'combined' analysis. Cell cycle clusters not displayed. Similar analysis for nephron clusters is presented in Fig. 3

may have been subjected to suboptimal conditions prior to processing. We note this signature is only revealed after comparing to another dataset and is a potential drawback with using cells derived from primary tissue, which is otherwise an ideal reference. The ribosomal RNA signature enriched in kidney organoids may reflect a higher metabolic rate within kidney organoid cells in culture [45]. Most tissue culture media, including that

used for kidney organoid culture, is high in glucose with culture conditions also physiologically hyperoxic compared to the fetus.

Conserved and differential expression analysis between fetal and organoid cell types

The major differences between the hFK and organoid datasets are likely to be technical artefacts due to sample

isolation and culture conditions. To focus on biological differences between cell types, we removed the genes identified as sample-specific from subsequent differential expression analyses between the datasets. We then compared hFK and organoid cells within each cluster to assess conserved and differentially expressed genes (Additional files 4 and 6), summarised in Table 1 and Fig. 1g.

Conserved stromal and endothelial cell types between organoid and fetal kidney

Our understanding of stromal subpopulations in developing human kidneys is relatively poor. However, Lindstrom

and colleagues sampled the cortex of the developing human kidney and classified the stromal cells within this sample into five populations representing three major groups: Lindstrom cluster (L) 9, marked by *REN*, *MGP* and *GATA3*; L2 marked by *LUM*, *SFRP2* and *DCN*; and three grouped subpopulations L10–12 with markers including *TCF21*, *ALDH1A2*, *ANGPT1*, *MEIS1*, and *TAGLN* [10]. These populations appear to be conserved in kidney organoids with the combined cluster C0 correlating to the L10–12 clusters (conserved markers include *TCF21*, *ALDH1A2*, *ANGPT1* and *TAGLN*); C1 representing L2 (conserved markers include *LUM*, *SFRP2*, and *DCN*); and C2 representing L9 (conserved markers include *MGP* and

Table 1 Summary of gene expression analyses for combined clusters excluding cell cycle and nephron lineage clusters

Cluster and sample origin		DE	Top enriched genes	GO summary for ≥ 10 DE genes	Top cluster markers	Top conserved markers
C0 Stroma	hFK	3	CKS2, LBH, CCBE1		SULT1E1, GPC3, MEG3, SERPINH1, ALDH1A2, TAGLN	TAGLN, ALDH1A2, PDGFRα, ZEB2, ACTA2, ALX1, SNAI2, COL6A3
	Org	2	NRK, STMN2			
C1 Stroma	hFK	87	SERPINI1, COL1A1, S100A10, SFRP2, ANXA1, ASPN	ECM organisation, signalling receptor binding	DCN, COL1A1, LUM, IGF1, POSTN, SFRP2, COL1A2, OGN	IGF1, SFRP2, COL1A1, DCN, LUM, COL1A2, COL3A1, POSTN
	Org	5	RPL27A, PALLD, ACTA2, DKK1			
C2 Stroma	hFK	11	REN, ID1, PDLM1, ITGA8, ENG, CPM	Endothelial and smooth muscle development	DLK1, MGP, NDUFA4L2, GATA3, APOE, PDGFRB, MEF2C	MGP, GATA3, NDUFA4L2, MEF2C, ACTA2, PDGFRB, PDLM1
	Org	2	SSBP4, FTL			
C3 Stroma	hFK	23	ACTA2, NR2F1, ANGPT1, ROBO2	No significant results	MAB21L2, CXCL14, PRRX1, ZFHX4, MAB21L1, CD24, COL9A2	ZFHX4, CTSK, HNRNPA1, DNMT3OS, SOX4, LIMA1, TBX3
	Org	12	VIM, VAMP2, SNHG8, DKK1			
C4 Endothelium	hFK	45	FN1, RBP5, PLPP3, CCL21, LGALS1, CXCL1, CXCL2, CXCL8, ICAM1	Signalling receptor binding, chemokine activity	GNG11, EGFL7, CLDN5, ESAM, PVLP1, CD34, CAV1, ARHGAP29, APLNR, CAV1, CD34, KDR, TIE1	GNG11, EGFL7, CLDN5, ESAM, PVLP1, S100A16, ARHGAP29, APLNR, CAV1, CD34, KDR, TIE1
	Org	44	APLN, PECAM1, MMP1, CAV1, CXCR4, HEY1			
C9 Stroma	hFK	67	COL1A1, PLPP3, ITM2B, IGF1, SPON2	Cell adhesion, ECM organisation	COL2A1, COL9A3, CNMD, MIA, COL9A2, COL9A1, FIBIN	COL9A3, DCN, LUM, FIBIN, COL1A2, OGN, IGFBP6, COL1A1, SOX9, SFRP2, MGP
	Org	41	CNMD, PEG10, COL9A1, COL9A3, GNG5, COL11A1			
C11 Glia	hFK	203	S100B, PLP1, MPZ, PMP22, ARHGAP15	Axon development	AP1S2, TTHY1, FABP7, SOX2, MSX1, PCSK1N	S100B, GPM6B, TTYH1, PLP1, SOX2, NKAIN3, PMP22, CNP, VIM
	Org	136	AP1S2, FABP7, ZFP36L1, MSX1			
C12 Immune	hFK	88	HLA-DPB1, HLA-DPA1, HLA-DRB1, CXCR4, CD83	Response to IFN gamma, antigen binding	HLA-DRA, CD74, SRGN, S100A9, TYROBP, S100A8, HLA-DPB1, LYZ	SRGN, LYZ, S100A9, TYROBP, S100A8, FCER1G, SPP1, FTL, CD74
	Org	24	PRTN3, MPO, CTSG, AZU1			
C13 Blood	hFK	24	HBM, AHSP, ALAS2, HEMGN, SLC25A37	Erythrocyte development, oxygen transport	HBG1, HBB, HBA2, HBA1, HMB, AHSP, ALAS2, SCNA, GNG11, HEMGN	CYPc, SLC25A39, SLC25A37, GPX1, HEY1, COPZ1, PRDX2, ACTB, SELENBP1, PFN1
	Org	168	FSCN1, EGFL7, FKBP1A, EIF4G2, GNG5, TPM4, BAX			
C14 Neural	hFK	N/A			HES6, CRABP1, TUBB2B, STMN2, TAGLN3, SSTR2	N/A
	Org	N/A				

Abbreviations: DE differentially expressed (adjusted *p* value < 0.05, absolute log fold change greater than 0.8), ECM extracellular matrix. Full lists available in Additional files and on website [32]

GATA3). Thus, human kidney organoids contain stromal populations similar to hFK cortical stromal populations, which form part of the nephrogenic niche and have been shown to influence nephron formation in mice [46, 47]. Few genes are differentially expressed between hFK and organoid cells in C0 but these include *NRK* and *STMN2*, upregulated in organoid cells, and *CCBE1* and *LBH* upregulated in hFK. More substantial differences are apparent in C1 with 87 genes upregulated in hFK compared to organoid cells including several signalling molecules such as *SFRP2*, *IGF1*, *WNT2B*, *DKK2* and *SEMA3A*. The expression of *WNT2B* is notable as this ligand is not expressed in the developing mouse kidney, and WNT signalling has several critical roles in kidney development. hFK cells within Cluster C3 express higher levels of markers associated with vascular smooth muscle and mesangial development including *RENIN* [48]. However, several highly upregulated markers are conserved within this cluster and the conserved signature also features several smooth muscle-associated genes such as *GATA3*, *MEF2C*, *ACTA2*, *HOPX*, *ANGPT2* and *PDGFRB*. Stromal clusters C3 (marked by *MAB2L2*, *CXCL14*, *PRRX1*) and C9 (*COL2A1*, *COL9A3*, *CNMD*) had smaller contributions of hFK cells (less than 5% of total cells), which could indicate that these organoid stromal clusters are less similar to native cell types, or that equivalent stromal populations are not adequately represented in the hFK sample. In situ hybridisation results from the mouse embryo indicate that some of the most upregulated markers of organoid stromal cell types C3 and C9 are expressed in the medullary and ureteric stroma of the developing kidney (Additional file 1), which was not sampled in the hFK data.

The endothelial cluster C4 featured extensive conservation of established markers such as *CLDN5*, *CDH5*, *CD34*, *KDR*, *TIE1*, *SOX17*, *SOX7* and *FLT1*. GO analysis of conserved markers resulted in terms related to vascular development indicating congruence between organoid endothelial cells and those from the hFK data. Despite this conservation, several genes were enriched in hFK or organoid endothelial cells. hFK upregulated genes were associated with cell signalling including chemokines *CXCL1*, *CXCL2* and *CXCL8* whereas organoid cells expressed higher levels of endothelial markers such as *PLXND1*, *APLN* and *PECAM1*. As such organoid endothelial cells appear to be appropriately specified but differ in the complement of signalling molecules they express. Whether this is the result of cell intrinsic factors or a response to being embedded in a different stromal environment is unclear.

Glial, neural, immune and blood clusters

Some clusters in the combined analysis consisted mainly of cells from one dataset. Organoid-enriched clusters include a glial cluster C11 (marked by *TTYH1*, *FABP7*,

SOX2), which included less than ten hFK cells, and C14 (marked by *SOX11*, *ELAVL3*, *ELAVL4*) that mostly contained organoid cells. GO analysis of the top markers of these clusters identifies terms associated with glial cell differentiation (C11) and generation of neurons (C14). There is some evidence of neural precursors being present during mouse [49] and human [43] kidney development, and neural populations play an important role in adult renal physiology. However, neural and renal progenitors have distinct embryonic origins, the former from the ectoderm and the latter from the intermediate mesoderm [6, 50, 51]. As our organoid protocol directs the bulk of cells towards an intermediate mesoderm-like fate [3], these glial and neural cell types are considered off-target and may reflect cells that adopted an alternative identity during the early stages of differentiation and persisted in culture.

An immune cell cluster (C12, marked by *PTPRC*, *TYROBP*, *FCER1G*) was mostly derived from hFK cells but surprisingly, 20 organoid cells contributed to this cluster and shared expression of immune response genes. During development, haematopoietic progenitors, cells can arise from specialised endothelial cells termed hemogenic endothelium [52]. This type of endothelium occurs within the aorta-gonad-mesonephros region [53], which is adjacent to the site of kidney development. Markers of hemogenic endothelial cells such as *PECAM1*, *KDR*, *KIT* and *CDH5* [52] are expressed in the organoid vasculature and, as such, it is possible that kidney organoids have some capacity to generate cells involved in the immune response.

Top markers of cluster C13 included genes highly expressed in blood (*HBG1*, *HBB*, *HBA2*), a signature primarily driven by the hFK-derived cells within the cluster as organoid-derived cells within this cluster did not share expression of these cell type-specific markers. GO analysis of upregulated organoid genes, and genes that were conserved markers between fetal and organoid cells in this cluster led to terms that were not related to blood.

Conserved nephron progenitor and early nephron cell types
hFK and organoid cells were present in clusters representing nephron progenitor cells (C6) and nephron epithelium (C10), but other hFK cells were split between a cluster containing most organoid podocytes (C7) and a hFK-specific podocyte cluster (C15).

Congruence and differences between hFK and kidney organoid nephron cell types

To compare nephron subpopulations in detail, we re-clustered cells from the combined nephron lineage clusters (C6, C7, C10, C15) in isolation. This generated seven combined nephron (CN) clusters with cells from hFK and organoid co-clustering within populations representing nephron progenitors (CN2), differentiating

nephron progenitors (CN1), distal (CN4) and proximal (CN5) nephron segments, podocyte precursor (CN3) and podocyte cells (CN0). The hFK-specific podocyte cluster (CN7) noted previously was maintained in this analysis, and a new cluster of hFK stromal cells, marked by *COL3A1*, *POSTN* and *MEG3* (CN6) was resolved. CN6 does not express any known nephron markers aside from

TMEM100, which was reported to be specific to nephron progenitors in the human fetal kidney [10]. In the absence of other nephron progenitor markers, this cluster does not appear to be part of the nephron lineage (Fig. 3a, b). Cells within this stromal CN6 cluster may have been associated with the nephron lineage based on the broad expression of stromal markers within human nephron progenitor

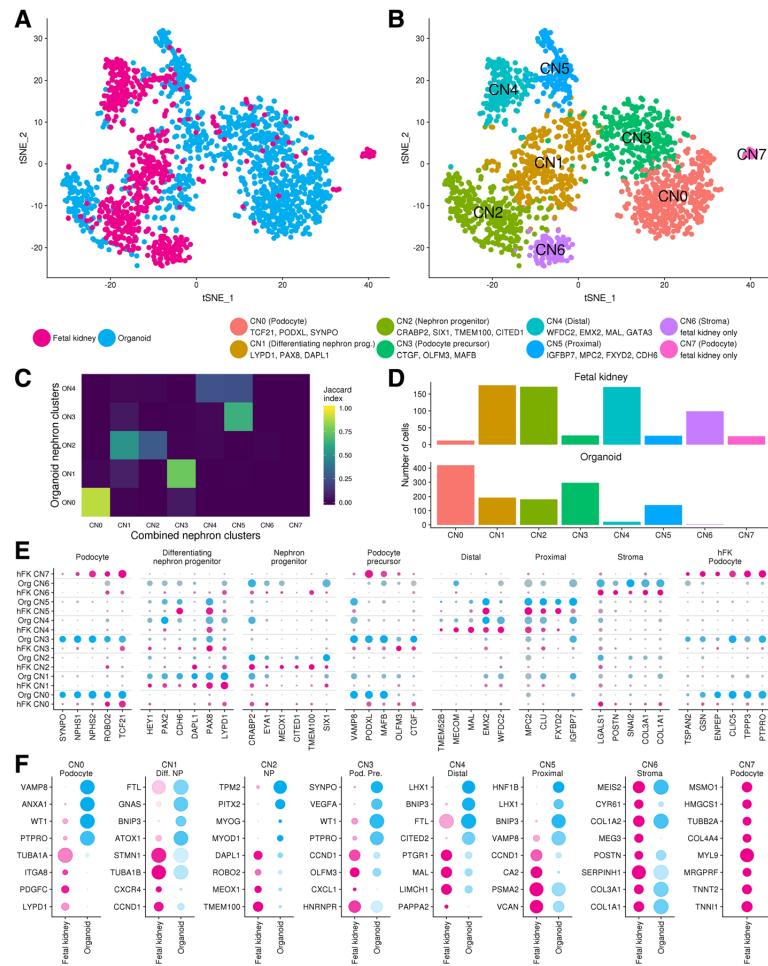


Fig. 3 Comparison of nephron cell types within kidney organoids and human fetal kidney. **a, b** Sample of origin and re-clustering of combined nephron (CN) lineage cells results in eight clusters. Cluster identity and select conserved marker genes shown next to cluster colour key. Cells for this analysis were selected from combined clusters C6, C7, C10 and C15. **c** Comparison of organoid cells between organoid nephron (ON) and combined nephron (CN) clusters. Colours show overlap in cells between clusters according to the Jaccard Index. **d** Number of cells in each combined nephron cluster by dataset. **e** Split dot plot showing relative expression for select marker genes within organoid and hFK cells in the combined nephron clusters. hFK data in pink, organoid in blue. Circle size represents the proportion of cells in the cluster expressing that gene, shading indicates expression level (low to high reflected as light to dark). **f** Top differentially expressed genes between datasets within combined nephron clusters. Chart colouring and shading as per **e**. Results for CN6 and CN7 are not differential expression results as few (CN6) or no (CN7) organoid cells are present within these clusters. These instead reflect top cluster markers (CN6) or markers enriched in CN7 but not CN0 or CN3. Organoid expression values for CN6 are derived from three organoid cells within this cluster

cells [10]. Again, organoid nephron cells formed similar groups when clustered with the hFK data as when clustered alone (Fig. 3c). hFK and organoid cells contributed to most clusters with the largest contributions in the nephron progenitor and differentiating progenitor cluster (Fig. 3d). hFK cells were scattered through the two organoid podocyte clusters but again a group of hFK podocyte cells remained separate. In the clusters representing the nephron epithelium, organoid cells were located in the proximal nephron cluster with few cells contributing to the distal while the hFK cells displayed the opposite pattern, consistent with the previous hFK analysis [10] and the under representation of distal nephron in these organoid samples.

Key cell-type markers were conserved between the datasets but differential expression testing revealed underlying differences in transcriptional profiles between hFK and organoid nephron cell types (Table 2, Fig. 3e, f, Additional files 7 and 8). hFK cells within nephron progenitor clusters showed enhanced expression of progenitor genes *TMEM100*, *MEOX1*, *ROBO2* and *DAPL1*, and a small portion of organoid nephron

progenitors expressed low levels of muscle progenitor genes *MYOD1* and *MYOG* (Additional file 1). Thus, despite the conservation of key markers, this population may be adopting a muscle progenitor fate in the absence of appropriate signals to reinforce nephron progenitor identity. Alternatively, a small number of muscle progenitors may have clustered with the nephron progenitors due to similarities in their expression profiles. Likewise, underlying a conserved profile, hFK cells within the differentiating nephron progenitor cluster expressed elevated levels of proliferation associated genes such as *CCND1* and *PCLAF* likely reflecting higher proliferation rates as seen during commitment to nephron formation in vivo [54].

Most of the top markers of the nephron tubule clusters represented established markers of distal and proximal tubule and were conserved between hFK and organoid cells. hFK cells expressed higher levels of distal markers *MAL*, *LIMCH1* and *PAPPA2* in distal cluster CN4 and organoid cells in proximal cluster CN5 had increased expression of genes associated with ureteric bud and pronephros development.

Table 2 Summary of gene expression analyses for combined nephron clusters excluding stromal cluster CN6

Cluster and sample origin		DE	Top enriched genes	GO summary for ≥ 10 DE genes	Top cluster markers	Top conserved markers
CN0 Pod	hFK	79	TUB1A1, NR2F1, SOX4, LGALS1, NUSAP1, LYPD1	Central nervous system development	S100A6, TCF21, PODXL, TPPP3, SBSPON, MMP5, CPXM1, DUSP23, THSD7A, MME	TCF21, MME, PODXL, THSD7A, TPPP3, SBSPON, MAFB, ENPEP, ROBO2, NPHS2
	Org	175	VAMP8, ANXA1, AIF1, S100A4, TGFBR3, WT1	ATP synthesis, oxidative phosphorylation	LYPD1, PAX8, HIST1H4C, PCLAF, CDH6, DAPL1, CCND1, RBP1, HMGB2	LYPD1, PAX8, HIST1H4C, PCLAF, DAPL1, PCP4, HEY1, CDH6, RBP1, PAX2
CN1 Diff. NP	hFK	37	CCND1, PCNA, CXCR4, TUBA1A, TBUA1B, STMN1	mRNA splicing	LYPD1, PAX8, HIST1H4C, PCLAF, CDH6, DAPL1, CCND1, RBP1, HMGB2	LYPD1, PAX8, HIST1H4C, PCLAF, DAPL1, PCP4, HEY1, CDH6, RBP1, PAX2
	Org	4	FTL, GNAS, BNIP, ATOX1			
CN2 NP	hFK	24	TMEM100, ITM2C, MEOX1, EPCAM, ROBO2, DAPL1	Amyloid precursor biosynthesis	ACTC1, NNAT, MYLPF, MYL1, TMEM100, TPM2, CRABP2, TUB1A1	NNAT, CRABP2, TUB1A1, IGF2, SIX1, TMEM100, CITED1, SOX4, MEOX1, MEIS2
	Org	25	ACT1C1, MYLPF, TPM2, PITX2, MYOG, MYOD	Muscle filament sliding		
CN3 Pod. Pre.	hFK	37	CCND1, OLFM3, CXCL1, HNRNPR, STMN1	No significant BP terms	CTGF, GPX3, TSPAN8, PAPPA, PAPPA, ITIH5, SERINC5, HES4, NPHS2, NPHS1, MAFB, PTPRO	CTGF, OLFM3, BCAM, NPHS1, ARHGAP29, CLDN1, LEPROT, TMP1, STON2, CLDN5, MAFB
	Org	78	GPX3, ANXA1, S100A4, AIF1, PTPRO, SYNPO, VEGFA, WT1	Epithelial cell diff. involved in kidney development		
CN4 Distal	hFK	26	PTGR1, MAL, LIMCH1, ELF3, ALDH1A1, PAPPA2	Epithelium development	HBG2, ATF3, LIMCH1, MAL, WFDC2, BTG2, HES1, KLF6, MECOM, ELF3, TUBB2B, GATA3	WFDC2, EMX2, LIMCH1, MAL, TUBB2B, SAT1, MECOM, HMGA1, HES1, GATA3, ATP1B1, GNG11
	Org	11	LHX1, BNIP3, FTL, CKB, BASP1, CITED2, HNRNPAB	Kidney morphogenesis		
CN5 Proximal	hFK	34	CCND1, CA2, VCAN, ELF3, DCDC2, FLRT3	No significant BP terms	IGFBP7, CD24, PCP4, PCSK1N, FXYD2, EMX2, MPC2, APOE, CLU, CFAP126, FTL, ATP1B1	IGFBP7, MPC2, SMIM24, FLRT3, EMX2, FXYD2, GNG11, TSPAN12, CLU, PCP4, ATP1B1, PDZK1
	Org	14	PCSK1N, CITED2, BNIP3, S100A13, MLLT1, PRDX5, VAMP8, LHX1	Ureteric bud morphogenesis, pronephros development		
CN7 hFK Pod	hFK				CXCL12, TNNI1, TNNT2, MME, MYL9, MRGPRF, TPPP3, ANXA2, COL4A4, ADM, PTPRO, MSMO1	N/A
	Org					

Abbreviations: *DE* differentially expressed (adjusted *p* value < 0.05, absolute log fold change greater than 0.8), *hFK* human fetal kidney, *Org* organoid. GO summary reporting top significant Gene Ontology (GO) Biological process (BP) results when fetal kidney or organoid DE gene lists were greater than or equal to ten genes. Full lists of cluster markers, conserved and differentially expressed genes and corresponding GO analyses available in Additional files or on website [32]

Of the three podocyte clusters, CN3 featured conserved expression of recently defined human podocyte precursor markers *MAFB*, *CTGF* and *OLFM3* [38] though *OLFM3* was expressed at higher levels in hFK cells in this cluster and organoid cells had higher levels of several podocyte markers such as *PTPRO*, *SYNPO*, *VEGFA* and *WT1*. hFK and organoid cells in CN0 expressed podocyte markers *TCF21*, *POXDL*, *ROBO2* and *NPHS2* but hFK cells within this cluster maintained expression of human nephron progenitor markers *LGALS1* and *LYPD1* [38] suggesting these cells may represent a more progenitor-like state than the organoid cells in this cluster. A final hFK-specific podocyte cluster formed distinct from the other two clusters which still included podocyte markers such as *PTPRO* and *TCF21* as marker genes but also included genes such as *TNNI1*, *TNNT2* and *MYL9*, which are expressed in cardiac muscle and podocytes. As markers of CN0 and CN7 largely overlap and represent a maturing podocyte state (*POXDL*, *PTPRO*, *TCF21* *AIF1*), we investigated disparities between these clusters by performing differential expression analysis (Additional file 9).

Podocyte genes *COL4A4* and *ANXA2* as well as *TNNI1*, *TUBA1A*, *COL9A1*, *STMN1* and *CA2* were upregulated in hFK-specific CN7. Genes upregulated in CN0 included podocyte enriched genes *ANXA1*, *GPX3*, *VAMP8*, *DACHI* and *WT1* as well as additional genes related to ATP synthesis that likely relate to the increased growth rate in culture. While CN0 expresses established podocyte markers, this cluster may also contain podocyte precursor states or podocytes that have not yet matured into glomeruli. Such a distinction may underlie the separation between CN0 and CN7.

This comparative analysis of nephron cell types within kidney organoids and human fetal kidney shows strong conservation of key cell type-specific markers while uncovering differences in the expression levels of key nephron progenitor markers, and a separation of some hFK podocytes from others, potentially reflecting in vivo maturation. We did not observe differences in transporter expression or markers of tubule maturation between organoid and hFK samples but that may be due to insufficient depth of profiling in these scRNA-seq datasets.

Discussion

We performed an in-depth analysis of nephron subpopulations in kidney organoids and found co-expression of a robust suite of established cell type-specific markers. Pseudotime analysis of these cell types suggests organoid nephron formation replicates an expected developmental trajectory from nephron progenitor to podocyte and tubular end points. We then asked whether there were underlying differences between kidney organoid cell types and equivalent populations in the developing human kidney. Organoid and hFK single-cell RNA-seq datasets were

integrated and clustered, with cells from both datasets contributing to most clusters. Conserved gene expression between organoid and hFK cells within endothelial, stromal and nephron cell types revealed congruence between these cell types demonstrating the capacity of organoids to represent many aspects of the developing human kidney. Where immunofluorescence had identified the presence of stromal markers in organoids, our single-cell analysis identified five stromal subpopulations, at least three of which are conserved to some level in the developing human kidney. The remaining two may represent renal stromal populations that are simply not represented in the hFK data set due to limited tissue collection, or off-target stromal cell types. Further comparisons with more complete human data sets will be required to discern between these options.

A recent study from Wu et al. also used single-cell analysis to analyse our kidney organoid protocol across time and compare it to another organoid protocol and human kidney cell types [55]. While the renal populations and congruence with human kidney cell types we identify are consistent with those results, the relative proportion and types of cells captured by that analysis differs somewhat from our findings. For example, we detect a muscle progenitor population, where they detected a cluster of melanocytes. We detect a glial and a neural cluster and they detected a neural progenitor population and four neuron clusters. The proportion of off-target populations in those samples is higher than in our analysis (~20% Wu et al., 6% this study), as is the proportion of tubule to podocyte cells. As such, the proportions of cell types generated by the same differentiation protocol are likely to vary between laboratories.

We were unable to resolve a distinct population representing ureteric epithelium in either dataset; however, markers previously used to define this population were under represented in the organoids analysed in this study. We previously reported simultaneous generation of presumptive ureteric epithelium and nephron lineages, with the proportions of cell types generated dependent on the timing of exposure to signals that pattern the anterior-posterior axis of the intermediate mesoderm [3]. The organoids generated for the present study were distinctly posterior and hence contained a lower frequency of epithelial GATA3⁺ structures. Recent studies argue that ureteric epithelium and nephron lineages cannot be generated simultaneously because they arise from distinct regions during embryonic development and instead must be generated using distinct protocols [7]. Analysis of this epithelial GATA3⁺ cluster from other kidney organoids is required to further explore the identity of this cell type. However, the absence of any ureteric epithelial cluster in the hFK data suggests a resistance of this tubular epithelium to dissociate into single cells. This same population

may be resistant to single-cell isolation from organoids. Hence, this will need to be overcome, perhaps using alternative technologies such as nuclear RNA preparations.

Differential expression within combined organoid and fetal kidney clusters identified an upregulation of ribosomal genes in kidney organoids and a heat shock/unfolded protein response in the fetal kidney data, both likely the result of technical artefacts rather than fundamental differences in cell identity. Examining gene expression after excluding these sample-enriched genes revealed additional differences between hFK and organoid cell types including notable changes in the levels of expression of nephron progenitor marker genes and in the levels and repertoire of growth factors expressed by stromal and endothelial cells. What was not evident in either dataset was the expression of several cell type-specific ligands and receptors that are apparent in analogous single cell datasets from the developing mouse kidney [33, 56]. For example, genes such as *GDNF* and *RET*, which encode a key ligand and receptor pair are known to operate in human kidney development as both genes cause renal birth defects when mutated in humans [57]. Being unable to detect the expression of such important genes in the reference hFK dataset leaves the possibility of important differences between organoid and hFK cell types that may only be revealed with deeper profiling.

Conclusions

This analysis supports a conservation of cell types between organoids and human fetal kidney. Overall, the data presented here builds confidence in the fidelity of organoid nephron, stromal and endothelial cell types, which will encourage disease modelling and drug screening efforts in human kidney organoids.

Additional files

Additional file 1: Supporting analysis for organoid data set. Figure with quality control, integration and supporting analysis for organoid dataset. (PNG 7123 kb)

Additional file 2: Organoid cluster markers. Organoid cluster markers and associated GO terms. (XLSX 3062 kb)

Additional file 3: Organoid nephron cluster markers. Organoid nephron cluster markers and associated GO terms. (XLSX 1089 kb)

Additional file 4: Combined conserved markers. Markers conserved between organoid and hFK cells within combined clusters and associated GO terms. (XLSX 77 kb)

Additional file 5: Differential expression analysis between organoid and hFK samples. Results from differential expression analysis between organoid and hFK samples used to identify a signature representing the main differences between the organoid and hFK datasets. (XLSX 2050 kb)

Additional file 6: Differential expression analysis for combined clusters. Results from differential expression analysis for organoid and hFK cells within combined clusters. (XLSX 2893 kb)

Additional file 7: Combined nephron conserved markers. Combined nephron cluster markers and associated GO terms. (XLSX 673 kb)

Additional file 8: Differential expression analysis for combined nephron clusters. Results from differential expression testing between organoid and hFK cells within each combined nephron cluster after removal of the sample-enriched signature. (XLSX 2040 kb)

Additional file 9: Differential expression analysis between podocytes in CN0 vs CN7. Results for differential gene expression testing between hFK-specific podocyte cluster CN7 and mixed organoid and hFK podocyte cluster CN0. (XLSX 77 kb)

Acknowledgements

Single cell sequencing was performed at the Australian Genome Research Facility Genomics Innovation Hub with the assistance of J. Jabbari and A. Seidi. Microscopy was performed at the Murdoch Children's Research Institute. We would like to thank D. Newgreen, D. Zhang, K.N. North, P.J. Houweling, S.M. Wilson, J.M. Vanslambrouck, S.E. Howden, and K.S. Tan from the Murdoch Children's Research Institute for assisting in generation of organoids or providing antibody reagents.

Funding

This work was supported by the Australian Research Council (DE150100652), the National Health and Medical Research Council (NHMRC) of Australia (GNT1156567), National Institutes of Health Rebuilding a Kidney consortium (DK107344) and seed funding from the Murdoch Children's Research Institute and the University of Melbourne. ANC was supported by a Discovery Early Career Researcher Award from the Australian Research Council. M.H.L. is a Senior Principal Research Fellow of the NHMRC (GNT1136085). A.O. is a Career Development Fellow of the NHMRC. L.Z. is supported by an Australian Government Research Training Program (RTP) Scholarship. MCRI is supported by the Victorian Government's Operational Infrastructure Support Program. These funding bodies had no role in any aspect of the study design, execution, or writing.

Availability of data and materials

Both organoid datasets are available from GEO accession number GSE114802 [58] and the Lindstrom fetal kidney dataset is available from GEO accession GSE102596 [59]. A website showing reports produced during analysis, including the exact software versions and parameters used, can be accessed at <http://oshslacklab.com/combes-organoid-paper/> and the analysis code is available at <https://github.com/Oshslack/combes-organoid-paper> [32].

Authors' contributions

PE performed the organoid differentiations, single-cell isolation and immunofluorescence. LZ designed and performed the single-cell analysis and generated figures under the supervision of AO. ANC designed and performed the single-cell experiments and immunofluorescence and was primarily responsible for data interpretation and writing, under the supervision of MHL. All authors contributed intellectually to the project and manuscript revisions. All authors read and approved the final manuscript.

Ethics approval and consent to participate

This study includes published, publicly available data from consented, anonymized, human fetal tissue, obtained from elective terminations following review of the study by Keck School of Medicine of the University of Southern California's Institutional Review Board and in accordance with the Declaration of Helsinki [10].

Consent for publication

Not applicable.

Competing interests

M.H.L. has consulted for and received funding from Organovo Holdings. The remaining authors declare that they have no competing interests.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Author details

¹Department of Anatomy & Neuroscience, University of Melbourne, Melbourne, VIC, Australia. ²Murdoch Children's Research Institute, Melbourne, VIC, Australia. ³School of Biosciences, The University of Melbourne, Melbourne, VIC, Australia. ⁴Department of Paediatrics, The University of Melbourne, Melbourne, VIC, Australia.

Received: 24 September 2018 Accepted: 14 January 2019
Published online: 23 January 2019

References

- McCauley HA, Wells JM. Pluripotent stem cell-derived organoids: using principles of developmental biology to grow human tissues in a dish. *Development*. 2017;144(6):958–62.
- Takasato M, Er PX, Becroft M, Vanslambrouck JM, Stanley EG, Elefanty AG, et al. Directing human embryonic stem cell differentiation towards a renal lineage generates a self-organizing kidney. *Nat Cell Biol*. 2014;16(1):118–26.
- Takasato M, Er PX, Chiu HS, Maier B, Baillie GJ, Ferguson C, et al. Kidney organoids from human iPS cells contain multiple lineages and model human nephrogenesis. *Nature*. 2015;526(7574):564–8.
- Freedman BS, Brooks CR, Lam AQ, Fu H, Morizane R, Agrawal V, et al. Modelling kidney disease with CRISPR-mutant kidney organoids derived from human pluripotent epiblast spheroids. *Nat Commun*. 2015;6:8715.
- Morizane R, Lam AQ, Freedman BS, Kishi S, Valerius MT, Bonventre JV. Nephron organoids derived from human pluripotent stem cells model kidney development and injury. *Nat Biotechnol*. 2015;33(1):1193–200.
- Taguchi A, Kaku Y, Ohmori T, Sharmin S, Ogawa M, Sasaki H, et al. Redefining the *in vivo* origin of metanephric nephron progenitors enables generation of complex kidney structures from pluripotent stem cells. *Cell Stem Cell*. 2014;14(1):53–67.
- Taguchi A, Nishinakamura R. Higher-order kidney organogenesis from pluripotent stem cells. *Cell Stem Cell*. 2017;21(6):730–46 e6.
- Przepiorksi A, Sander V, Tran T, Hollywood JA, Sorrenson B, Shih JH, et al. A simple bioreactor-based method to generate kidney organoids from pluripotent stem cells. *Stem Cell Reports*. 2018;11(2):470–84.
- Phipson B, Er PX, Combes AN, Forbes TA, Howden SE, Zappia L, et al. Evaluation of variability in human kidney organoids. *Nat Methods*. 2019;16(1):79–87.
- Lindstrom NO, Guo J, Kim AD, Tran T, Guo Q, De Sena BG, et al. Conserved and divergent features of mesenchymal progenitor cell types within the cortical nephrogenic niche of the human and mouse kidney. *J Am Soc Nephrol*. 2018;29(3):806–24.
- Takasato M, Er PX, Chiu HS, Little MH. Generation of kidney organoids from human pluripotent stem cells. *Nat Protoc*. 2016;11(9):1681–92.
- Briggs JA, Sun J, Shepherd J, Ovchinnikov DA, Chung T, Nayler SP, et al. Integration-free induced pluripotent stem cells model genetic and neural developmental features of Down syndrome etiology. *Stem Cells*. 2013;31(3):467–78.
- R Core Team: R: A language and environment for statistical computing: R Foundation for Statistical Computing, Vienna, Austria; 2013. Available from: URL: <http://www.R-project.org/>.
- Lun A, Risso D. SingleCellExperiment: S4 Classes for Single Cell Data. bioconductor.org2017.
- Smedley D, Haider S, Durinck S, Pandini L, Provero P, Allen J, et al. The BioMart community portal: an innovative alternative to large, centralized data repositories. *Nucleic Acids Res*. 2015;43(W1):W589–W98.
- Durinck S, Moreau Y, Kasprzyk A, Davis S, De Moor B, Brazma A, et al. BioMart and Bioconductor: a powerful link between biological databases and microarray data analysis. *Bioinformatics*. 2005;21(16):3439–40.
- Scialdone A, Natarajan KN, Saraiva LR, Proserpio V, Teichmann SA, Stegle O, et al. Computational assignment of cell-cycle stage from single-cell transcriptome data. *Methods*. 2015;85:54–61.
- Lun AT, McCarthy DJ, Marioni JC. A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor. *F1000Research*. 2016;5:2122.
- McCarthy DJ, Campbell KR, Lun AT, Wills QF. Scater: pre-processing, quality control, normalization and visualization of single-cell RNA-seq data in R. *Bioinformatics*. 2017;33(8):1179–86.
- Butler A, Hoffman P, Smibert P, Papalexi E, Satija R. Integrating single-cell transcriptomic data across different conditions, technologies, and species. *Nat Biotechnol*. 2018;36(5):411–20.
- Satija R, Farrell JA, Gennert D, Schier AF, Regev A. Spatial reconstruction of single-cell gene expression data. *Nat Biotechnol*. 2015;33(5):495–502.
- Hardoon DR, Szedmak S, Shawe-Taylor J. Canonical correlation analysis: an overview with application to learning methods. *Neural Comput*. 2004;16(12):2639–64.
- Hotelling H. Relations between two sets of variates. *Biometrika*. 1936;28:321–77.
- Berndt DJ, Clifford J. Using dynamic time warping to find patterns in time series. *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*; 1994.
- Blondel VD, Guillaume JL, Lambiotte R, Lefebvre E. Fast unfolding of communities in large networks. *J Stat Mech-Theory E*. 2008;200810008. <http://iopscience.iop.org/article/10.1088/1742-5468/2008/10/P10008/meta>.
- Zappia L, Oshlack A. Clustering trees: a visualization for evaluating clusterings at multiple resolutions. *GigaScience*. 2018;7(7):giy083.
- Bauer DF. Constructing confidence sets using rank statistics. *J Am Stat Assoc*. 1972;67(339):687–90.
- Qiu X, Mao Q, Tang Y, Wang L, Chawla R, Pliner HA, et al. Reversed graph embedding resolves complex single-cell trajectories. *Nat Methods*. 2017;14(10):979–82.
- Trapnell C, Cacchiarelli D, Grimsby J, Pokharel P, Li S, Morse M, et al. The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells. *Nat Biotechnol*. 2014;32(4):381–6.
- Wickham H. *ggplot2: Elegant Graphics for Data Analysis*. 2 ed. New York: Springer International Publishing; 2016. p. 260.
- Wilke CO. *cowplot: Streamlined Plot Theme and Plot Annotations for 'ggplot2'*. 2018 [May 21, 2018]. Available from: <https://CRAN.R-project.org/package=cowplot>.
- Zappia L, Combes AN, Er PX, Little M, Oshlack A. Combes organoid paper analysis code. GitHub. 2018. Available from: <http://oshlacklab.com/combes-organoid-paper/>. Accessed 12 Jan 2018.
- Combes AN, Phipson B, Zappia L, Lawlor KT, Er PX, Oshlack A, et al. High throughput single cell RNA-seq of developing mouse kidney and human kidney organoids reveals a roadmap for recreating the kidney. *bioRxiv*. 2017;332:273–86.
- Georgas K, Rumballe B, Valerius MT, Chiu HS, Thiagarajan RD, Lesieur E, et al. Analysis of early nephron patterning reveals a role for distal RV proliferation in fusion to the ureteric tip via a cap mesenchyme-derived connecting segment. *Dev Biol*. 2009;332(2):273–86.
- Thiagarajan RD, Georgas KM, Rumballe BA, Lesieur E, Chiu HS, Taylor D, et al. Identification of anchor genes during kidney development defines ontological relationships, molecular subcompartments and regulatory pathways. *PLoS One*. 2011;6(2):e17286.
- Chen J, Bardes EE, Aronow BJ, Jegga AG. ToppGene Suite for gene list enrichment analysis and candidate gene prioritization. *Nucleic Acids Res*. 2009;37(Web Server issue):W305–11.
- Bruski EW, Potter SS. Changes in the gene expression programs of renal mesangial cells during diabetic nephropathy. *BMC Nephrol*. 2012;13:70.
- Lindstrom NO, De Sena BG, Tran T, Ransick A, Suh G, Guo J, et al. Progressive recruitment of mesenchymal progenitors reveals a time-dependent process of cell fate acquisition in mouse and human nephrogenesis. *Dev Cell*. 2018;45(5):651–60 e4.
- O'Brien LL, Guo Q, Lee Y, Tran T, Benazet JD, Whitney PH, et al. Differential regulation of mouse and human nephron progenitors by the six family of transcriptional regulators. *Development*. 2016;143(4):595–608.
- Phipson B, Er PX, Combes AN, Forbes TA, Howden SE, Zappia L, et al. Transcriptional evaluation of the developmental accuracy, reproducibility and robustness of kidney organoids derived from human pluripotent stem cells. *Nature Methods*. 2018.
- Takasato M, Er PX, Chiu HS, Maier B, Baillie GJ, Ferguson C, et al. Kidney organoids from human iPS cells contain multiple lineages and model human nephrogenesis. *Nature*. 2016;536(7615):238.
- Kobayashi A, Mugford JW, Krautberger AM, Naiman N, Liao J, McMahon AP. Identification of a multipotent self-renewing stromal progenitor population during mammalian kidney organogenesis. *Stem Cell Reports*. 2014;3(4):650–62.
- Young MD, Mitchell TJ, Vieira Braga FA, Tran MGB, Stewart BJ, Ferdinand JR, et al. Single-cell transcriptomes from human kidneys reveal the cellular identity of renal tumors. *Science*. 2018;361(6402):594–9.
- Adam M, Potter AS, Potter SS. Psychrophilic proteases dramatically reduce single-cell RNA-seq artifacts: a molecular atlas of kidney development. *Development*. 2017;144(19):3625–32.

45. Rudra D, Warner JR. What better measure than ribosome synthesis? *Genes Dev.* 2004;18(20):2431–6.
46. Das A, Tanigawa S, Karner CM, Xin M, Lum L, Chen C, et al. Stromal-epithelial crosstalk regulates kidney progenitor cell differentiation. *Nat Cell Biol.* 2013;15(9):1035–44.
47. Fetting JL, Guay JA, Karolak MJ, Iozzo RV, Adams DC, Maridas DE, et al. FOXD1 promotes nephron progenitor differentiation by repressing decorin in the embryonic kidney. *Development.* 2014;141(1):17–27.
48. Sequeira Lopez ML, Penty ES, Nomasa T, Smithies O, Gomez RA. Renin cells are precursors for multiple cell types that switch to the renin phenotype when homeostasis is threatened. *Dev Cell.* 2004;6(5):719–28.
49. Sariola H, Holm K, Henke-Fahle S. Early innervation of the metanephric kidney. *Development.* 1988;104(4):589–99.
50. James RG, Schultheiss TM. Patterning of the avian intermediate mesoderm by lateral plate and axial tissues. *Dev Biol.* 2003;253(1):109–24.
51. Lancaster MA, Knoblich JA. Generation of cerebral organoids from human pluripotent stem cells. *Nat Protoc.* 2014;9(10):2329–40.
52. Hirschi KK. Hemogenic endothelium during development and beyond. *Blood.* 2012;119(21):4823–7.
53. Medvinsky A, Dzierzak E. Definitive hematopoiesis is autonomously initiated by the AGM region. *Cell.* 1996;86(6):897–906.
54. Short KM, Combes AN, Lefevre J, Ju AL, Georgas KM, Lamberton T, et al. Global quantification of tissue dynamics in the developing mouse kidney. *Dev Cell.* 2014;29(2):188–202.
55. Wu H, Uchimura K, Donnelly EL, Kirita Y, Morris SA, Humphreys BD. Comparative analysis and refinement of human PSC-derived kidney organoid differentiation with single-cell transcriptomics. *Cell Stem Cell.* 2018;23(6):869–81.
56. Magella B, Adam M, Potter AS, Venkatasubramanian M, Chetal K, Hay SB, et al. Cross-platform single cell analysis of kidney development shows stromal cells express Gdnf. *Developmental biology.* 2017.
57. Vivante A, Kohl S, Hwang DY, Dworschak GC, Hildebrandt F. Single-gene causes of congenital anomalies of the kidney and urinary tract (CAKUT) in humans. *Pediatr Nephrol.* 2014;29(4):695–704.
58. Phipson B, Zappia L, Combes AN. Single cell RNA-Seq of four human kidney organoids. Accession number GSE114802. 2018.
59. Ransick A, Kim AD, De Sena Brandine G, Lindstrom NO, McMahon A. Single Cell RNA-Seq profiling human embryonic kidney cortex cells. *Gene Expression Omnibus.* Accession number GSE102596. 2018.

Ready to submit your research? Choose BMC and benefit from:

- fast, convenient online submission
- thorough peer review by experienced researchers in your field
- rapid publication on acceptance
- support for research data, including large and complex data types
- gold Open Access which fosters wider collaboration and increased citations
- maximum visibility for your research: over 100M website views per year

At BMC, research is always in progress.

Learn more biomedcentral.com/submissions



F

Analysis methods

F.1 Pre-processing

The Cell Ranger pipeline (v3.0.1) [34] was used to perform sample demultiplexing, barcode processing and single-cell gene counting. Briefly, samples were demultiplexed to produce a pair of FASTQ files for each sample. Reads containing sequence information were aligned using the reference provided with Cell Ranger (v3.0.0) based on the GRCh38 reference genome and ENSEMBL gene annotation (v93). PCR duplicates were removed by selecting unique combinations of cell barcodes, unique molecular identifiers and gene ids with the final results being a gene expression matrix that was used for further analysis. The three samples were aggregated using Cell Ranger with no normalisation and treated as a single dataset containing 945038 droplets. The R statistical programming language (v3.5.0) [228] was used for further analysis. Count data was read into R and used to construct a SingleCellExperiment object (v1.4.1) [229].

F.1.1 Droplet selection

The droplet selection method in this version of Cell Ranger identified 8291 cell containing droplets. This is more than the traditional Cell Ranger approach based on a simple threshold on total counts which identified 7006 cells. The EmptyDrops method [99] available in the DropletUtils package (v1.2.2) was also applied. Droplets with less than 100 total counts were used to construct the ambient RNA profile and an FDR threshold of less than 0.01 was used to select 9858 droplets. The droplets selected by either the Cell Ranger v3 method or EmptyDrops were kept giving a dataset with 10002 cells. Gene annotation information was added from BioMart [220] using the biomaRt package (v2.38.0) [221] and cells were assigned cell cycle scores using the cyclone [222] function in the scran package (v1.10.2) [109].

F.1.2 Alevin

An alternative quantification was performed using the alevin method [91] available as part of the salmon software package (v0.12.0) [12]. This produced a dataset with 10423 cells and 37247 genes.

F.2 Quality control

The scater package (v1.10.1) [101] was used to produce a series of diagnostic quality control plots and select thresholds for filtering. Cells were removed from the dataset if they had \log_{10} total counts less than 2.309 (4 MADs from the median), \log_{10} total features expressed less than 2.3659 (4 MADs) or more than 8.2411 percent of counts (4 MADs) assigned to mitochondrial genes. An automated filtering method available in scater that detects outliers in a PCA of cells based on technical factors was also performed for comparison but not used for filtering. After filtering doublet scores for each cell were calculated using the simulation-based method available in scran and cells with a score greater than 1.5572 were removed. A minimal filter was used to remove genes that did not have at least 1 count in at least 2 cells. After quality control the dataset had 8059 cells and 22739 genes with a median of 7838 counts per cell and a median of 2506 genes expressed.

F.3 Clustering

F.3.1 Gene selection

Two methods were used to select genes to use for clustering. The default selection method in the Seurat package (v2.3.4) [129] uses thresholds on mean expression (between 0.0125 and 3.5) and dispersion (greater than 1), which selected 2457 genes. The M3Drop (v3.10.3) [104] method was also used to test for genes with significantly more zero counts than expected and 1952 genes with an adjusted p-value less than 0.01 were selected. Following comparisons between these two methods the genes identified by M3Drop were used for clustering.

F.3.2 Graph-based clustering

Seurat was used to perform graph-based clustering. PCA was performed using the selected genes and the first 15 principal components were used to construct a shared nearest neighbour graph using the overlap between the 30 nearest neighbours of each cell. Louvain modularity optimisation [133] was used to partition this graph with a resolution parameter between 0 and 1. Clustering tree visualisations [208] were produced using the clustree package (v0.3.0) showing the expression of previously identified marker genes. By inspecting these trees a resolution of 0.4 which chosen which produced 13 clusters. To compare these cluster to those that had been previously published the Jaccard index was calculated between pairs of clusters from each analysis and visualised as a heatmap.

F.4 Marker genes

Marker genes for each cluster were identified using edgeR (v3.24.3) [15,16]. Additional filtering was performed to remove genes with a \log_{10} maximum average counts in any cluster less than -1.3505. The edgeR negative binomial model was then fitted to the dataset using a design matrix that included the detection rate (scaled proportion of genes expressed in each cell). The quasi-likelihood F-test was used to test cells in one cluster against all other cells. Genes were considered significant markers for a cluster if they had an FDR less than 0.05 and a log fold change greater than 1. Identities were assigned to each cluster by comparing the detected genes to previously published lists of cell type markers.

F.5 Partition-based graph abstraction (PAGA)

Partition-based graph abstraction (PAGA) [226] was performed using the scanpy library (v1.4) [227] for the Python programming language (v3.6.8). A shared nearest neighbour graph was built using the same parameters as were used by Seurat and connectivity between the Seurat clusters was calculated using the PAGA algorithm. A ForceAtlas2 [230] layout of the cell graph was calculated based on the PAGA cluster graph. The results of the PAGA analysis were read into R for visualisation.

F.6 Cell velocity

The velocito (v0.17.16) [151] Python program was used to calculate spliced and unspliced count matrices from the aligned reads. The resulting Loom files for each sample were then read into R and merged. The remaining analysis was performed using the velocito R package

(v0.6). Genes were removed from the spliced matrix if they had did not have a mean expression greater than 0.2 in at least one cluster or from the unspliced matrix if they did not have a mean expression greater than 0.05 in at least one cluster. Relative velocity estimates were then calculated for each cell and projected onto reduced dimensional spaces for visualisation.

F.7 Other packages

Visualisations and figures were primarily created using the ggplot2 (v3.1.0) [231] and cowplot (v0.9.4) [232] packages using the viridis colour palette (v0.5.1) for continuous data. UpSet plots [218] were produced using the UpSetR package (v1.3.3) [233] with help from the gridExtra package (v2.3) [234] and SinaPlots [224] using the ggforce package (v0.1.3) [235]. Data manipulation was performed using other packages in the tidyverse (v1.2.1) [236] particularly dplyr (v0.7.8) [237], tidyr (v0.7.8) [238] and purrr (v0.3.0) [239]. Loom files were exported from R using the LoomExperiment package (v1.0.2) [240]. The analysis project was managed using the workflowr (v1.1.1) [241] package which was also used to produce the publicly available website displaying the analysis code, results and output. Reproducible reports were produced using knitr (v1.21) [242–244] and R Markdown (v1.11) [245,246] and converted to HTML using Pandoc (v1.17.2).

G

Session information

G.1 Important packages

Details of the main packages used to build this thesis.

- bookdown (v0.7) – writing complex documents using R Markdown [247,248]
- clustree (v0.3.0) – producing clustering tree visualisations [208,249]
- cowplot (v0.9.3) – constructing complex panel figures [232]
- dplyr (v0.7.8) – manipulation of data frames [237]
- forcats (v0.3.0) – manipulation of factors [250]
- ggplot2 (v3.1.0) – data visualisation [231,251]
- glue (v1.3.0) – complex joining of strings [252]
- googleAnalyticsR (v0.6.0) – downloading Google Analytics data [253]
- googleAuthR (v0.7.0) – Google API authentication [254]
- here (v0.1) – locating files [255]
- jsonlite (v1.6) – reading JSON files [256,257]
- knitr (v1.20) – generating dynamic reports using R [242–244]
- lubridate (v1.7.4) – manipulation of dates [258,259]
- magrittr (v1.5) – piping between functions [260]
- packrat (v0.4.9.3) – management of package dependencies [261]
- purrr (v0.2.5) – functional programming for lists [239]
- RColorBrewer (v1.1.2) – ColorBrewer colour palettes [262]
- readr (v1.3.1) – input of data files [263]
- remotes (v2.0.0) – installation of packages from GitHub [264]
- rmarkdown (v1.10) – dynamic documents for R [246]
- stringr (v1.3.1) – manipulation of strings [265]
- tidyverse (v0.8.2) – tidying of data [238]
- viridis (v0.5.1) – viridis colour palettes [266]

G.2 Full session information

```
## Session info -----
## setting  value
## version  R version 3.6.1 (2017-01-27)
## system   x86_64, linux-gnu
## ui        X11
## language en_US.UTF-8
## collate  en_US.UTF-8
## tz       UTC
## date    2019-08-12

## Packages -----
##  package     * version  date      source
## assertthat    0.2.0    2017-04-11 CRAN (R 3.6.1)
## backports     1.1.2    2017-12-13 CRAN (R 3.6.1)
## base         * 3.6.1    2019-07-22 local
## bindr         0.1.1    2018-03-13 CRAN (R 3.6.1)
## bindrcpp     * 0.2.2    2018-03-29 CRAN (R 3.6.1)
## bookdown      0.7     2018-02-18 CRAN (R 3.6.1)
## checkmate     1.9.1    2019-01-15 CRAN (R 3.6.1)
## clustree      * 0.3.0    2019-02-24 CRAN (R 3.6.1)
## colorspace    1.3-2    2016-12-14 CRAN (R 3.6.1)
## compiler      3.6.1    2019-07-22 local
## cowplot       * 0.9.3    2018-07-15 CRAN (R 3.6.1)
## crayon        1.3.4    2017-09-16 CRAN (R 3.6.1)
## datasets      * 3.6.1    2019-07-22 local
## devtools      1.13.6   2018-06-27 CRAN (R 3.6.1)
## digest        0.6.18   2018-10-10 CRAN (R 3.6.1)
## dplyr         * 0.7.8    2018-11-10 CRAN (R 3.6.1)
## evaluate      0.12     2018-10-09 CRAN (R 3.6.1)
## farver        1.1.0    2018-11-20 CRAN (R 3.6.1)
##forcats       0.3.0    2018-02-19 CRAN (R 3.6.1)
```

```
## ggforce      0.2.2   2019-04-23 CRAN (R 3.6.1)
## ggplot2       * 3.1.0   2018-10-25 CRAN (R 3.6.1)
## ggraph        * 1.0.2   2018-07-07 CRAN (R 3.6.1)
## ggrepel       0.8.0   2018-05-09 CRAN (R 3.6.1)
## glue          1.3.0   2018-07-17 CRAN (R 3.6.1)
## graphics     * 3.6.1   2019-07-22 local
## grDevices    * 3.6.1   2019-07-22 local
## grid          3.6.1   2019-07-22 local
## gridExtra     2.3     2017-09-09 CRAN (R 3.6.1)
## gtable        0.2.0   2016-02-26 CRAN (R 3.6.1)
## here          * 0.1     2017-05-28 CRAN (R 3.6.1)
## hms           0.4.2   2018-03-10 CRAN (R 3.6.1)
## htmltools     0.3.6   2017-04-28 CRAN (R 3.6.1)
## igraph         1.2.2   2018-07-27 CRAN (R 3.6.1)
## jsonlite      * 1.6     2018-12-07 CRAN (R 3.6.1)
## knitr         * 1.20    2018-02-20 CRAN (R 3.6.1)
## labeling       0.3     2014-08-23 CRAN (R 3.6.1)
## lazyeval       0.2.1   2017-10-29 CRAN (R 3.6.1)
## magrittr      1.5     2014-11-22 CRAN (R 3.6.1)
## maps          * 3.3.0   2018-04-03 CRAN (R 3.6.1)
## MASS          7.3-51.4 2019-03-31 CRAN (R 3.6.1)
## memoise       1.1.0   2017-04-21 CRAN (R 3.6.1)
## methods       * 3.6.1   2019-07-22 local
## munsell       0.5.0   2018-06-12 CRAN (R 3.6.1)
## packrat       0.4.9-3  2018-06-01 CRAN (R 3.6.1)
## pillar         1.3.0   2018-07-14 CRAN (R 3.6.1)
## pkgconfig     2.0.2   2018-08-16 CRAN (R 3.6.1)
## plyr          1.8.4   2016-06-08 CRAN (R 3.6.1)
## polyclip      1.9-1   2018-07-27 CRAN (R 3.6.1)
## purrr         0.2.5   2018-05-29 CRAN (R 3.6.1)
## R6             2.3.0   2018-10-04 CRAN (R 3.6.1)
## RColorBrewer  * 1.1-2  2014-12-07 CRAN (R 3.6.1)
## Rcpp           1.0.0   2018-11-07 CRAN (R 3.6.1)
## readr          * 1.3.1   2018-12-21 CRAN (R 3.6.1)
## rlang          0.3.0.1  2018-10-25 CRAN (R 3.6.1)
## rmarkdown      1.10    2018-06-11 CRAN (R 3.6.1)
## rprojroot     1.3-2   2018-01-03 CRAN (R 3.6.1)
## rstudioapi    0.8     2018-10-02 CRAN (R 3.6.1)
## scales         1.0.0   2018-08-09 CRAN (R 3.6.1)
## stats          * 3.6.1   2019-07-22 local
## stringi        1.2.4   2018-07-20 CRAN (R 3.6.1)
## stringr       * 1.3.1   2018-05-10 CRAN (R 3.6.1)
## tibble         1.4.2   2018-01-22 CRAN (R 3.6.1)
## tidygraph      1.1.1   2018-11-20 CRAN (R 3.6.1)
## tidyrr         * 0.8.2   2018-10-28 CRAN (R 3.6.1)
## tidyselect     0.2.5   2018-10-11 CRAN (R 3.6.1)
## tools          3.6.1   2019-07-22 local
## tweenr         1.0.1   2018-12-14 CRAN (R 3.6.1)
## utils          * 3.6.1   2019-07-22 local
## viridis        0.5.1   2018-03-29 CRAN (R 3.6.1)
## viridisLite   0.3.0   2018-02-01 CRAN (R 3.6.1)
## withr          2.1.2   2018-03-15 CRAN (R 3.6.1)
## xfun            0.3     2018-07-06 CRAN (R 3.6.1)
## yaml           2.2.0   2018-07-25 CRAN (R 3.6.1)
```