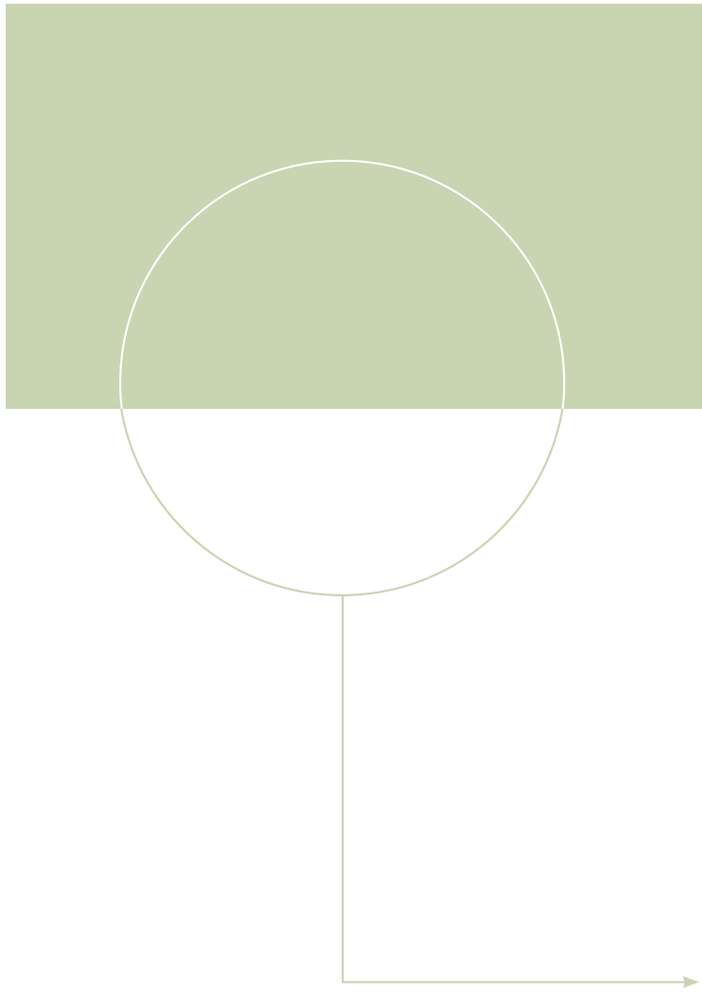


Master's thesis

2019

Geir Amund Svan Hasle

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Computer Science

Geir Amund Svan Hasle

Modelling Cell Cycle Phase Distribution in Cell Cultures

September 2019

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Modelling Cell Cycle Phase Distribution in Cell Cultures

Geir Amund Svan Hasle

Master of Technology in Computer Science
Submission date: September 2019
Supervisor: Pål Sætrom
Co-supervisor: Arnar Flatberg

Norwegian University of Science and Technology
Department of Computer Science

Acknowledgement

I would like to thank my supervisor Prof. Pål Sætrom for his insights and expert guidance. Special thanks must be given to my co-supervisor Arnar Flatberg for his many spontaneous lectures. Thanks to all my colleagues at the Genomics Core Facility for making this possible. To my family, thank you for your support.

Most of all, thank you Kristin.

G.A.S.H.

Abstract

We use single cell RNA gene expression data to study the cell cycle. We compare two methods for single cell RNA gene expression quantification and analyze both the resulting count matrices and their performance. A method is developed for ordering the cells with respect to their position in the cell cycle by using Principal Component Analysis. The gene expression profiles are modelled as function of cyclic time using smoothing cubic splines. We perform Partial Least Squares regression with the modelled gene expression profiles as input and determine which expression profiles that are cell cycle periodic within a significance threshold. The results are analyzed by functional enrichment for biological processes. The method is applied to three single cell RNA data sets.

A set of new candidate genes are found by performing a correlation analysis to find candidate genes that have consistent expression profiles in all three data sets. Enrichment analysis on the candidate genes show that our method is able to identify genes that have known cell cycle functions.

Sammendrag

Vi benytter oss av enkeltcelle RNA genekspresjonsdata for å studere cellesyklusen. To metoder for enkeltcelle RNA genekspresjonskvantifisering sammenlignes. Vi analyserer både tellematrisene og ytelsen på begge metodene. Ved å benytte prinsipialkomponentanalyse utvikler vi en metode for å bestemme en ordning av cellene med hensyn på deres posisjon i cellesyklusen. Genenes ekspresjonsprofiler modelleres så som funksjoner av syklisk tid ved hjelp av glattende splineinterpolasjon. En delvis minstekvadraters regresjon benyttes deretter ved å bruke splinemodellene som inndata for å finne de genutrykksprofilene som viser signifikant cellesyklisk-periodiske uttrykk innenfor et signifikansnivå. Vi bestemmer deretter om settet av de signifikante cellesykliske genprofilene har kjente biologiske funksjoner relatert til cellesyklusen. Vi analyserer tre enkeltcelle datasett med den utviklede metoden.

Vi bestemmer et sett med nye kandidatgener ved å se på overlappet av signifikant cellesyklusuttrykte gener mellom datasettene. En korrelasjonsanalyse utføres for å finne et sett med gener som viser konsistente uttrykksprofiler på tvers av datasettene. Deretter analyserer vi om settet av kandidatgener har kjente biologiske funksjoner tilknyttet cellesyklusen. Analysen viser at metoden er i stand til å identifisere gener med kjente biologiske cellesyklusfunksjoner.

Contents

.

List of Figures

List of Tables

Acronyms

DNA Deoxyribonucleic acid

RNA Ribonucleic acid

mRNA messenger RNA

tRNA transfer RNA

NGS Next Generation Sequencing

PCA Principal Component Analysis

PLS Partial Least Squares regression

Chapter 1

Introduction

The cell cycle is an important process in functioning cells. It regulates every step in the path towards mitosis, when the cell divides and become two cells. The defined steps in the cell cycle prevents cells from progressing through the cell cycle if irregularities are detected. This prevents dysfunctional cells with mutated DNA from spreading their mutations further. Cancer cells are cells that show irregular behaviour in the cell cycle. They carry mutations in their hereditary code such that they are not susceptible to the some of the regulatory mechanisms in the cell cycle. When cell divide without regulation they run a higher risk of developing new mutations and they diverge ever more from the functioning cells.

Common strategies for studying the cell cycle with RNA sequencing require that the cells to be studied must be synchronized. This is achieved by adding chemicals to a cell culture to block the cycling cells in a certain stage of the cell cycle. After a time, a new chemical is then added to release the cells from the block and the cycling cells will continue cycling as a synchronised culture. Samples are then extracted from the culture at fixed time intervals and each sample is sequenced using bulk-RNA technology. A critique of this approach is that we cannot know exactly how the observed gene expressions are affected by the synchronisation techniques employed. With the cost of single cell RNA technology dropping, more an more research facilities have the capability of acquiring genomic data from single cells in a biological sample. With this technology, we can analyze the gene expression of each cell in a sample individually. If we are able to apply a time ordering

to the cells observed in single cell RNA experiments, we can model the gene expressions without the need of synchronization.

1.1 Project goals

In this experiment we will study the cell cycle by analyzing single cell RNA gene expression profiles. We will compare the performance of single cell gene expression quantification from sequncing data. Two different technologies will be studied: CellRanger and the newly released STARsolo.

We will use the generated genomic data to derive a robust method for modelling and visualizing single cell gene expression data as function of cyclic time. The steps will involve pre-processing

- Pre-processing.
- Acquiring a ordering of the cell with respect to a cell cycle pseudotime.
- Modelling the gene expression profiles.
- Identification of cell cycle periodic gene expression profiles.
- Analysis of the cell cycle periodic genes.

We will use established bioinformatic software and develop utilities where we need them. Our method will be tested on multiple single cell gene expression data sets.

1.2 Report outline

Chapter 2 : We provides some background information on the relevant concepts and methods we need within cell biology, sequencing technology, computer science, mathematical methods and statistical methods.

Chapter 3 : Explains a strategy for identification of cell cycle periodic genes and gives the details involved in each step.

Chapter 4 : We present the results of our method applied to multiple data sets.

Chapter 5 : A discussion of the observations from our results are presented.

Chapter 6 : We give a conclusion of the experiment and summarize our observations.

Chapter 7 : A proposition of further improvements and analysis are made.

Chapter 2

Background

The purpose of this chapter is to provide some background theory that may be required or helpful to understand the material in this report. We will cover cell theory, where we describe the different kinds of cells, how they are built up and some basic functions of their biological machinery. This will include how cells store their hereditary information in the form of deoxyribonucleic acid (DNA), how this is used to transcribe ribonucleic acid (RNA) which in turn is translated to proteins or used for regulatory processes in the cell. We will also describe the process in which cells replicate themselves in the process called the cell cycle.

Further, we will give an introduction to the concept of DNA sequencing. Here, we will cover technologies used for RNA sequencing, the process in which the complementary DNA nucleotide sequence of actively expressed genes in a biological sample are obtained. We will give some historical context and show how the methods for sequencing have developed from Sanger sequencing to next generation sequencing (NGS) with the application of single cell sequencing.

We will give a description of the software and tools used to process the data obtained from the sequencing experiments in the laboratory. Genomics is a complex field at the intersection of several science disciplines. Often a bioinformatician analyzing data can choose from many different tools solving the same task using different algorithms. Even rerunning an experiment with slightly different parameters of the same software can yield

different results. In this field, both reusability of pipelines and reproducibility of their results is of great concern. In this regard, we will cover some tools and concepts commonly used to address this concern.

Lastly in this chapter, we will cover the mathematical and statistical methods used to analyze the data. We give definitions and concepts from linear algebra that we need to describe our methods. We cover cubic splines and introduce smoothing cubic splines. From statistics we will present the concept of covariance between correlated variables. This is a key concept in the methods Principal Component Analysis and Partial Least Squares. Both of which we will need in our method later.

2.1 Cell Biology

We give a short introduction to cell biology and cover some of the important

2.1.1 The Cell

Living organisms either consists of a single cell or of many cells working together in intricate systems. Each cell contains all the hereditary information particular to its species together with the biological machinery needed by the cell to replicate itself. The cell is often called the smallest unit of life. There are three known tenets of cell theory

1. All living things are composed of one or more cells.
2. The cell is the basic unit of life.
3. New cells arise from pre-existing cells.

All cells can be categorized as either *prokaryote* or *eukaryote*. Every cell will share some basic strucures, regardless of which category they belong to. A cell will always have an outer plasma membrane, an inner region of cytoplasm, DNA and ribosomes used to synthesize proteins. Cells are divided into two categories, prokaryotes and eukaryotes.

Prokaryotes are unicellular organisms that don't have a membrane-bound nucleus. Bacteria falls under this domain. Eukaryotes have a membrane-bound nucleus and many other components that we will cover next.

Eukaryotes

Eukaryotic cells are cells that have a membrane-bound nucleus. The nucleus is where eukaryotes store and replicate their DNA. The nucleus contains DNA that is wrapped around proteins in a structure called chromatin which in turn is organized into multiple chromosomes. This is suspended in the nucleoplasm. The packaging of DNA into chromosomes is very important to be able to fit all of the genetic code in the nuclues. The chromosomes in the nucleus contain most of the DNA found in eukaryote cells. The cell also contains ribosomes. The ribosomes are used to transcribe parts of the DNA to messenger RNA (mRNA), which in turn is used in protein production or to regulate other cell functions. This will be the topic of later sections in this chapter.

Eukaryotes also have other membrane-bound units, called *organelles*. An organelle is simply a membrane-bound unit inside a cell that performs special functions in the cell. All eukarytoic cells have the same basic set of membrane-bound organelles. Depending on the type of cell, the different organelles can come in varying size and numbers, advantageous to the main task of the cell type. An illustration of a cell can be seen in Figure 2.2.

The membrane of the nucleus consists of a double layer. Together, the two layers are called the nuclear envelope. As part of the outer layer of the envelope we find the *endoplasmic reticulum* (ER). This is the largest organelle of the cell. It is where proteins that will be incorporated into cellular membranes or proteins that will be secreted from the cell are synthesized. The proteins break of from the ER in small membrane-bound vesicles and travel through the cytoplasm until they reach their next destination.

The Golgi apparatus is an organelle that functions as a sort of postal office in the cell. Proteins that leave ER are destined for the Golgi apparatus. They travel to *cis* side of the organelle and are incorporated into the inside of the Golgi apparatus, called the *lumen*. The proteins then travel through the Golgi apparatus to the *trans* side. Underway, the

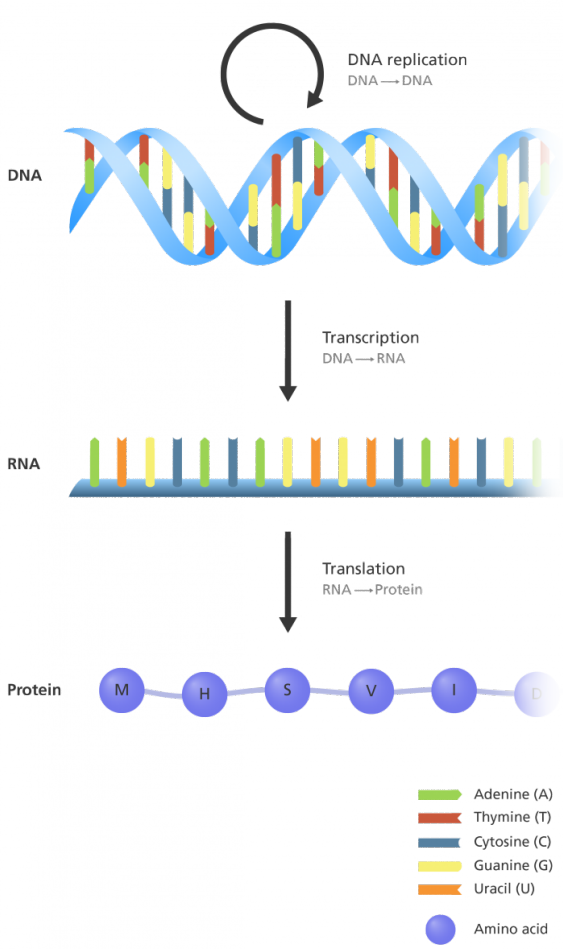


Figure 2.1: The central dogma of molecular biology.

proteins can be modified by adding or removing sugar molecules or by adding phosphate molecules. The proteins are then sorted based on the modifications before they finally break of in new vesicles, headed for their final destination.

The mitochondria are organelles that produce adeniose triphosphate (ATP). ATP is the main energy source of the cell. The mitocondria produces ATP through *cellular respiration* using chemical energy from nutrients such as sugars. Mitochondria are largely believed originate from bacteria that became engulfed by host eukaryotic cells and developed a symbiotic, mutual dependency that gave an evolutionary advantage [SD78; And+03]. A small part of the genetic material carried in a eukaryotic cell comes from mitochondria.

Lysosomes are organelles that contain digestive enzymes and function as a recycling center of the cell. It can break down macromolecules and particles floating around in the cytosol of the cell and reuse the molecules. It can also digest and neutralize particles brought

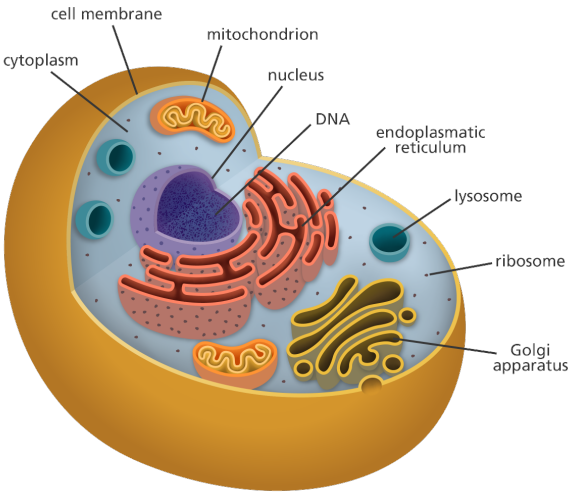


Figure 2.2: Organization of a Cell.
Source: www.yourgenome.org

in from the outside of the cell. The peroxisome is another organelle that is involved in detoxification and breaking down fatty acids and amino acids.

Another definition of an organelle is that any unit within a cell that has a function is counted as an organelle. That, is we don't require organelles to be enclosed by a membrane. Within this definition, *ribosomes* can be categorized as an organelle. Regardless of which of definition we operate under, the ribosomes serves a very important function within the cell. It is where the proteins are produced. Ribosomes can be found either attached to the ER in the *rough* part or freely suspended in the cytoplasm of the cell. As mentioned before, proteins that are destined for the membrane of the cell or proteins that will be secreted from the cell are synthesized in the ER. All other proteins, i.e. proteins that has a function within the cell, are synthesized at the ribosomes that are suspended freely in the cytoplasm of the cell. Exactly how a protein is synthesized within a ribosome is discussed further in section 2.1.3.

2.1.2 DNA - The genetic code

The DNA molecule is organized as two strands that coil around each other. The two strands form a double helix. Each strand of the DNA molecules consists of long, unbranched, paired polymers. The polymers again are built by multiple single sugar-phosphate molecules linked together, each with a nitrogen-containing base attached to

it. The sugar-phosphate molecule together with attached base is called a monomer nucleotide, or simply a nucleotide.

In the DNA, there are four possible bases for each monomer nucleotide. They are adenine, guanine, cytosine and thymine. Commonly, they are simply denoted by the one letter characters after their respective first letters: A, G, C and T. A single strand of DNA is then formed by binding the sugar-phosphate molecules of the nucleotide monomers together, resulting in a chain of A, G, C and T. The sugar-phosphate molecules are asymmetric such that a single strand of DNA is given a direction.

Each cell replicates its DNA by a process called *templated polymerization*. This is the process when a new strand of DNA is formed by linking together nucleotides together by matching with a pre-existing strand of DNA, called a template strand. In this process the nitrogen bonds are formed by linking A to T and G to C.

The A-T bonds are formed by two hydrogen bonds, while the G-C bonds are formed by three hydrogen bonds. Hence, the G-C bonds will be stronger than the A-T bonds. The pairing of A-Ts and G-Cs then controls the formation of a new strand of DNA onto the existing template strand. The result is double stranded DNA molecule where each strand consists of a complementary sequence of A, T, G and C nucleotides.

The bonds between the nitrogen bases are in turn much weaker than the sugar-phosphate bonds linking each nucleotide together in the single strands. This allows the double-stranded structure to be pulled apart from each other while maintaining the integrity of each of the strands. Each of the strands can then be used for continued replication in the templated polymerization process, passing on the hereditary code of the cell.

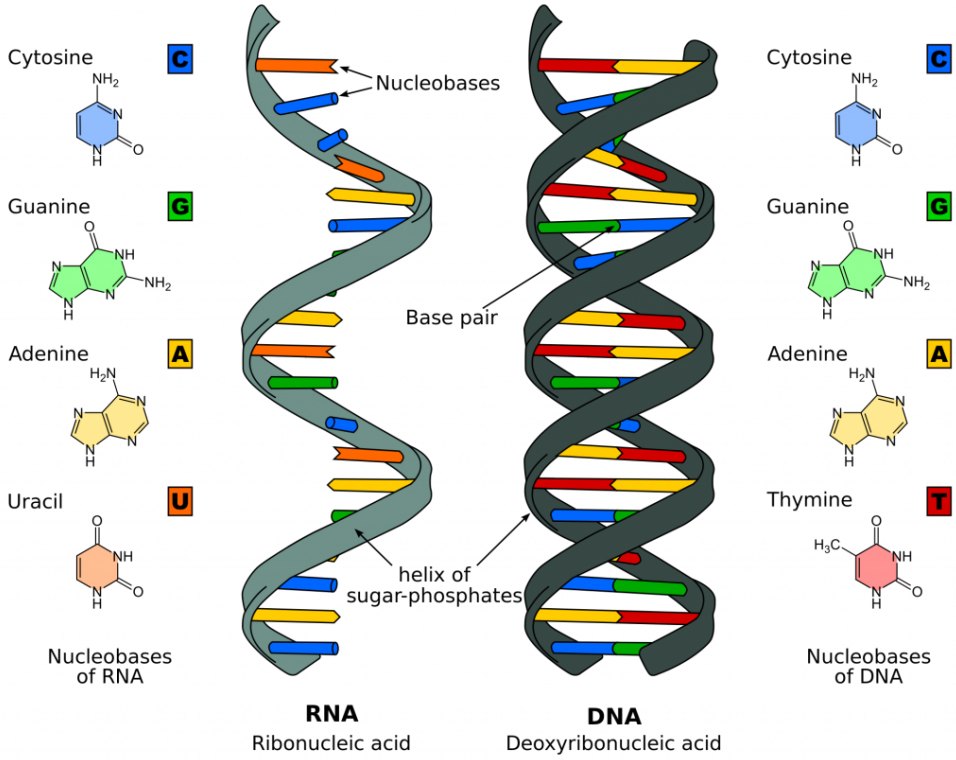


Figure 2.3: RNA and DNA molecules.
Source: Wikimedia Commons

2.1.3 RNA Transcription and Protein Translation

Different types of cells serve specific functions in an organism. Different segments of DNA encodes for different functions. A *gene* is a aprt of the DNA that carries the code used in production of protein. This part can be *expressed* through another templated polymerization process, called *transcription*. During transcription, certain parts of the DNA are used as templates to form *ribonucleic acid*, RNA. RNA is very similar to the DNA, but it differs in two important ways. Each component of the RNA is linked together by forming bonds between ribose molecules, as opposed to deoxribose in the DNA. The RNA nucleotides also have four bases, but the thymine base (T) is replaced with a uracil base (U). During transcription, U will match with A in the DNA and G and C will match, just as in the DNA polymerization. Only certain segments of a DNA strand is used to transcribe a RNA molecule and the RNA a is much shorter molecule than the DNA molecule. Even though one of the bases in the RNA is different than from the DNA, an RNA molecule perfectly translates to a segment of the DNA. The result after the templated polymerization process is one copy of a part of the DNA.

Each cell in an organism in general contains the same DNA. Since different parts of the DNA transcribes to different RNA molecules, different cells can use the same DNA to produce different RNA molecules suitable to the task or the state of the specific cell. Within a cell, the same segment of the DNA, a gene, is typically used repeatedly to produce many copies of an RNA molecule.

The RNA molecules that are used for protein synthesis are called messenger RNA (mRNA). This mRNA is headed for ribosomes located at the ER or ribosomes floating in the cytoplasm. The nucleotide sequence of the mRNA is read by passing through the ribosome between the small subunit and the large subunit. The nucleotide sequence from the mRNA is read three at a time in groups called *codons*. Each codon from the mRNA is matched by a three nucleotide long and complementary *anti-codon* from a transfer RNA (tRNA). tRNA is an RNA molecule that has an amino acid attached to it. Hence, the tRNA is responsible for the translation from the language of the nucleotide bases of the mRNA to amino acids, which are the building blocks of protein. As each codon from the mRNA is matched by the anti-codons from the tRNA, the amino acids carried by the tRNA are linked together in a chain of amino acids. The end product is a protein and the process of synthesizing proteins from mRNA is simply called translation.

2.1.4 Genes

Whenever information from a gene is in the process of producing a functional product, we say the gene is expressed. The result of gene expression is most often a protein, but functional RNA, like tRNA, can also be the end product of a gene expression. Only a small portion of the DNA, around 1.5%, consists of sequences that code for proteins. Even the parts of the DNA that are referred to as genes contain regions that are not used in protein translation. The gene is organized into segments called *introns*, the non-coding parts, and *exons*, the protein coding segments. When a part of the DNA has been transcribed, the resulting RNA molecule is called a pre mRNA. The introns must be cut away before the before it becomes a mRNA molecule that can be used for protein translation. This process is called RNA splicing (Figure 2.4).

The expression of a gene is regulated through complex processes and all stages of gene

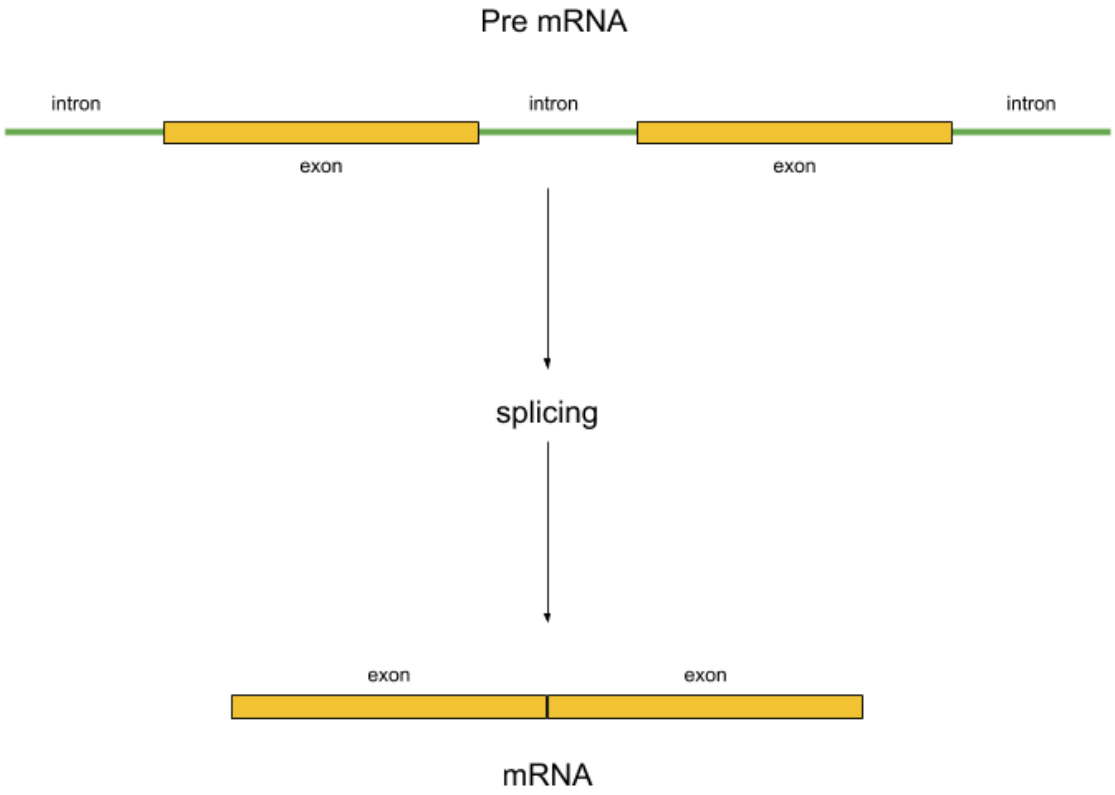


Figure 2.4: RNA splicing: from pre mRNA to mRNA.

expression can be affected by regulation. It allows cells to respond to stimuli from its surroundings and adapt to new circumstances. Gene regulation is also controls developmental paths a cell can take.

2.1.5 The Cell Cycle

The cell cycle is an important function in all cells. From the start of this chapter, we remember that all cells derive from pre-existing cells. All multi-cellular organisms are developed from a single cell, a fertilized egg. Multi-cellular organism does not grow because each of its cells grow. It grows because the cells recreate them selves and increase in numbers. The cell cycle is a series of phases and events that each cell must pass through if it is to replicate it self. The process in which cells grow and reproduce is also called *proliferation*. The cell cycle is carefully regulated to ensure that the end result, a new cell, is healthy and can perform its functions.

The cell cylcle consists of two stages. The *M-phase* (mitosis), where cell divides into

two daughter cells and the *interphase*, where the cell grows and ensures that necessary preparations are made before mitosis. The interphase is further divided into three phases G1, S and G2. There are also control stages in the cell cycle, called checkpoints. At the checkpoints, transition into the next phase can be blocked if the there are requirements within the cell that is not met or if the cell detects environmental factors that are not favourable for progression through the cell cycle. The cell cycle is illustrated in Figure **??**. The characteristics of each of the stages can be described by the following

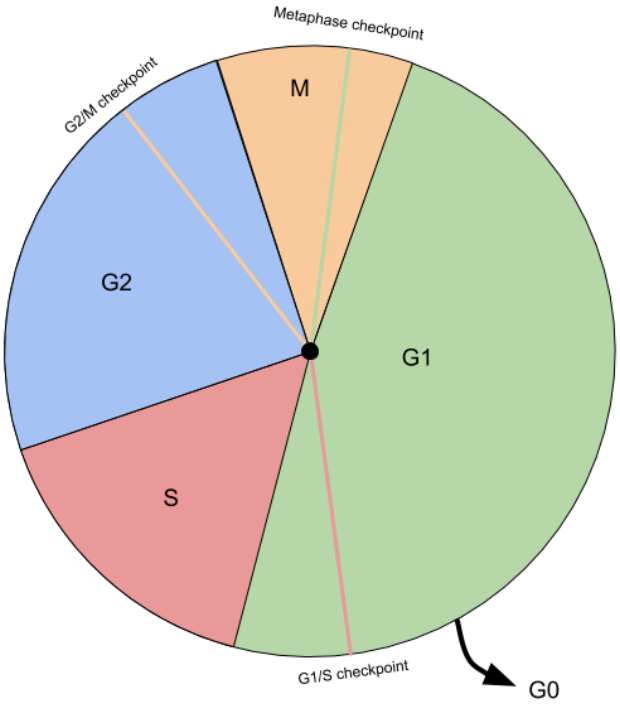


Figure 2.5: Illustration of the cell cycle with its phases and checkpoints.

- **G1 (Gap 1):** This is the first growing phase that proceeds immediately after a daughter cell is formed. The cell grows in size and it produces proteins and organelles.
- **G1/S checkpoint:** If the cell passes through this checkpoint, it is committed to the cell cycle. This checkpoint controls that all necessary supplies for synthesizing DNA are in place.

- **S (Synthesis)**: The cell replicates its DNA. All chromosomes are replicated and organized into *sister chromatids*.
- **G2 (Gap 2)**: This is the second growing phase. The cell increase in size and prepares for mitosis.
- **G2/M checkpoint**: Any damage to the DNA is controlled for and repaired. If the DNA is damaged, a process called *apoptosis* may be triggered. Apoptosis is a process of programmed cell death. This mechanism prevents dysfunctional DNA from being reproduced to new daughter cells.
- **M (Mitosis)**: The growth stops and the cell prepares for division. Within this phase, we also find the last checkpoint.
- **Metaphase checkpoint**: The sister chromatids are separated and organized for cell division.

Directly after the last checkpoint, *cytokinesis* follows. In this step, the components of the cell is organized and the cell finally divides.

There is also a last phase, the G0 phase. Here, the cell does not partake in proliferation. This can be intermediate phase for a cell, for example if it does not pass the G1/S checkpoint. In this case, the cell can re-enter the cell cycle later.It can also be a permanent phase. Neurons is one cell type that mostly don't proliferate and mostly spend their entire lifespan in G0. The duration of the cell cycle can vary both between organisms and different cell types within an organism.

Cancer cells are cells that typically carry mutations that make insusceptible to the regulatory mechanisms of the cell cycle. This mutations can affect genes that regulates cell growth, apoptosis or other functions that normally prevents erratic behaviour of the cell. Such mutations can make cancer cells proliferate at a higher rate compared to functional cells and they can also evade programmed cell death. When cancer cells grow unregulated, the differentiate increasingly from functioning cells. They can form tumors that prevent normal functions in an organism and they can spread to other sites in the organism. Studying the cell cycle is therefore an important endeavour to gain new knowledge on cancer cells and how they can be regulated.

2.1.6 Cell Cultures

Cell cultures are grown in an a controlled environment outside of the cell (in vitro). Cells of interest are isolated from tissue in a biological sample. They usually have a limited life span, but if a mutation occurs or the cells are manipulated, they can become an *immortalized cell line*. In an immortalized cell line, the cells can proliferate indefinitely as long as they have the resources required. These types of cell lines have many applications in research. It enables repeated experiments on biological material that has the same genetics. Because of the mutations that enables immortalization, they may show different behaviours that must be accounted for in experiments. This will depend on the type of experiment and the particular cell culture.

HaCat cells is an immortalized cell line that derived from human skin that shows high rates of proliferation in cell cultures. They have been used to model the metabolism of Vitamin D3 in the human skin. In [Pen+13], HaCat cells were used in a synchronization experiment to study cyclic gene expression profiles.

2.2 Sequencing Technologies

We will give a brief introduction to the field of DNA sequencing and its applications. The core technologies and how they have developed will be outlined. The field of DNA sequencing is concerned with determining the nucleotide sequences that form the DNA of a biological sample. By manual methods researchers where able to determine 12 base pair (bp) sequences from a bacteriophage in 1971[WT71]. Today, high throughput sequencing is able to determine up to 1000 billions of base pairs per experiment. Depending on tecnology, *reads* of base pairs are determined in sequence of varying length, called the *read length*. For example, a Illumina HiSeq X reports capability of 2x150 bp read length and $\tilde{5}$ billion reads per run. PacBio SequelSystems are capable of delivering up to 10 000 reads, each of length 30 000 bp. DNA sequencing is capable of more than just determining the nucleotide sequences in the genetic code of an organism. For example, when we prepare the biological samples for sequencing, we can construct complementary DNA (cDNA) molecules for RNA present in the sample. This can be used to determine the abundance

of mRNA in different types of tissues. Different sequencing technologies find appliances in a wide range of fields. From medicine and molecular biology to forensics.

2.2.1 Sanger Sequencing

The Sanger sequencing method was developed by Frederick Sanger in 1975[SC75]. In the original paper, the method is described as *"A simple and rapid method for determining nucleotide sequences in single-stranded DNA by primed synthesis with DNA polymerase..."*. DNA polymerase is an enzyme that is necessary for copying DNA. In DNA synthesis, DNA polymerase uses deoxribonucleotides to copy of existing DNA. *Oligonucletides* are synthetic DNA. In sanger sequencing, oligonucleotides are used to form a DNA template. The DNA template and a DNA sample of interest is then separated in four different reactions. In each reaction, the template is then extended through a *polymerase chain reaction*(PCR) where all DNA nucleotides (A,T,C,G) are available. To each reaction, a lower concentration of chain-terminating nucleotides of *one* of A,T,C or G is added. These chain-terminating nucleotides is missing a 3'-OH group such that DNA polymerase is not able to contiue DNA synthesis on a strand where it is added. Each reaction will then produce DNA molecules of random lengths, terminating at one of the four bases. In addition, these chain-terminators are labeled with a marker. In the early days of Sanger sequencing, radioactive marker where used.

The sequencing PCR product from each reaction is placed in separate lanes of acrylamide gel on a glass plate. By exposing the glass plate to an electric field, the DNA molecules will be organized by size within each lane. By exposing the gel to a X-ray film, the nucleotide sequence was then determined by manually inspecting marks from different lanes on a radioautograph. The marks signifies where the ends of the DNA molecules. Since each lane corresponds to a nucleotide, the sequence can be determined. Sanger sequencing is capable of performing reads up to 900 bp.

Refinements were progressively added to the Sanger method and it became increasingly automated. In the 80's, a method was developed that added fluorescent dyes. Today, fully automated solutions exist that performs one reaction and determines the sequence with sensitive cameras. Sanger sequencing has many applications still today and it is

known for its high accuracy. For example, it is used in verification experiments of new technologies.

2.2.2 Next Generation Sequencing

By the mid 1990's and early 2000, several new methods for sequencing started emerging and sequencing became increasingly comercially available. This wave of new technology in the field were given the name Next Generation Sequencing (NGS). Today, NGS is often synonymous with High Throughput Sequencing (HTS) because of its ability to produce large amounts of genomic data. With NGS, it is possible to determine millions of DNA sequences in parallell. Compared to Sanger sequencing, the reads are often shorter, 35-700 bp, and it is associated with higher error rates.

One of the methods from NGS is a method of short-read sequencing called *sequencing by synthesis* (SBS). Short read methods typically have read lengths that are less than 300 bp. Different SBS appliances have different strategies for determining a DNA sequence. We will follow the procedure for a Illumina sequencer.

During sequence library preparation, DNA is extracted from a biological sample *sheared*. This means that it is cut into smaller fragments of DNA. Then, sequence primers and adapters are added. The adapter binds the DNA molecule a fixed location in a sequencing *flow cell*. A flow cell is a disposable glass plate that chemicals can flow through. This enables automated chemical reactions inside the sequencer. After the DNA molecules bind to the flowcell, each location with a DNA attached to it is then amplified using polymerase so that clusters of DNA molecules are formed. The clustering is necessary to get a sufficient signal in the next step. Modified nucleotides are added to the flow cell. Each nucleotide is altered with a molecule that blocks polymerase from progressing after it is attached. In addition colour dye is attached. The left over nucleotides that has not been attached to a DNA strand is then washed away and a high sensitive camera can detect the colour at each location in the flow cell. The blocking molecule and the dye are then detached from the DNA strands by adding chemicals to the flow cell and washing out the residues. This process is referred to as a cycle. New cycles are performed until we have reached the target read length. After all the cycles are finished, the DNA strand

that has been synthesized is removed. *Index primers* are then added and an *index read* is performed. An index is a DNA sequence that is added to the DNA sequencing library. It uniquely identifies each biological sample in the experiment. This is necessary to be able to *demultiplex* all reads to the sample of their origin. The index read is typically 8 bp and is added to the DNA molecules during library preparation. After the index read one can optionally perform a *reverse read*. In this case, the index primer is first removed and then a complementary DNA (cDNA) strand is built from the original DNA. Index 2 primers are then added and we perform a new index read. After the primer is removed, cycles to determine the sequence of the cDNA initiates.

We call the first read the *forward read*, short R1, and the second read the *reverse read*, R2. On a Illumina NextSeq 500, the index of the first read is named index P5, while index 2 is named index P7. An illustration of the process can be seen in Figure 2.6.

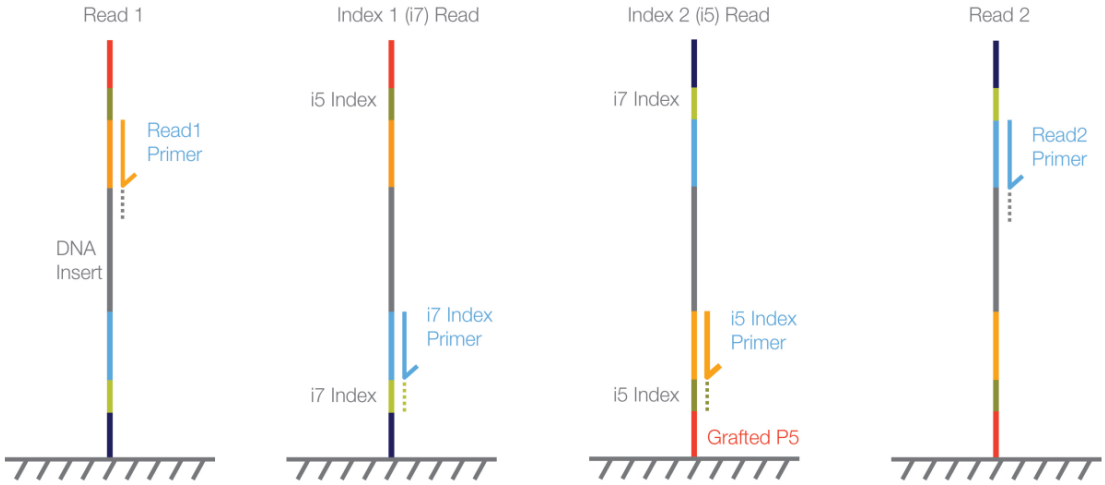


Figure 2.6: Illumina sequencing workflow.
Source: Illumina Indexed Sequencing Overview Guide

The strategy of using a forward and a reverse read enables us to reconstruct DNA that are longer than the cycle capabilities of a specific sequencer when we analyse the results . Take a sequencer that is capable of producing 150 bp DNA reads in one direction. If it is enabled for reverse reads as well, we can determine a longer sequence by matching the end of the first read with beginning of the reverse read. If we require an overlap of 50 bp, we can construct a DNA sequence that is 250 bp long.

NGS RNA sequencing, often called bulk-RNA, allows us to analyze the abundance of the genes that are present in a sample. A common application is to look for genes that are

differentially expressed between sample groups. If we sequence biopsies from two different types of tumors, we can examine which genes that differs significantly in expression levels between the two groups. By analyzing these, we may find new information on the characteristics of the tumors or the functions of the genes involved.

2.2.3 Singe Cell Sequencing

The previous methods retrieves gene expression information that is an average from all cells present in a sample. It is not able to distinguish between the origins of the sequences within a specific biological sample. Single cell sequencing exploits NGS technologies to examine sequences originating from individual cells in a sample. This allows for analyzing genomic data at higher resolution. This can be used to investigate genetic mutations carried by certain cell populations, like cancer cells. With single cell RNA sequencing (scRNA-seq) we can analyze the gene expression in a cell population, allowing us to better understand their functions.

Droplet single cell sequencing is a method where the sequence information can be traced back to its origin. Through a microfluidic channel system, each cell is captured in a oil droplet together with a *bead*. This bead is covered with oligos that each contain *cell barcode*, a *unique molecular identifier* (UMI) and the necessities to capture mRNA from the cell. Individual mRNA molecules from the cell attaches to the oligos on different locations on a bead. On a single bead, all cell barcodes consist of the same nucleotide sequence. These cell barcodes is what enables us to map each read back to a single cell. The UMI's are unique for each oligo on the bead. This enables us to quantify the number of copies originating from each mRNA molecule in the cell. The mRNA molecules attached to the oligos are reverse transcribed to cDNA and amplified with PCR. The cells are then pooled together and indexing sequences are attached. This allows to separate cell barcodes originating from different biological samples if we do sequencing on multiple samples simultaneously. An illustration of the preparation of droplet scRNA-seq using the 10X Chromium technology is given in Figure 2.7. In the case of the 10X technology for droplet, the prepared libraries are sequenced using Illumina technology. For example, the Illumina NextSeq 500 mentioned previously. One run is capable of profiling from 1000-10

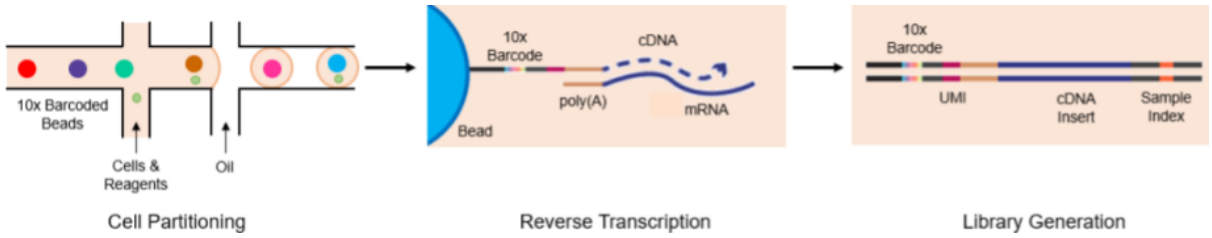


Figure 2.7: 10X library preparation workflow.
Source: genewiz.com

000 cells[1].

In scRNA-seq, each cell can be considered to be an individual sample. With thousands of cells in each experiment, it makes analysis more complicated. Analyzing single cell data often involves complex methods for clustering and visualizing cells that show similar features. The different clusters will often represent different types of cells. The clusters must be identified and we may have many different types of cell present. Deriving information from this is much more complicated than in bulk-RNA where this resolution is not attainable.

A draw back of scRNA-seq is that each single cell contain very little RNA. Therefore we get very sparse gene expression data. This means that many cells will show no expression in a significant amount of genes. This occurs when mRNA in a droplet does not attach to the oligos. The input mRNA is not captured by the beads at the same proportion in all droplets. This is referred to as a *dropout* and it is hard to separate it from a *missing value*, that the gene is not expressed in the cell.

2.3 Computer Science

Here we will introduce computational routines frequently involved with analyzing genomic data. We will cover the generation of raw sequences from the output of sequencers and how these are used to retrieve information on which genes that are expressed in a biological sample. In addition, we cover the concepts and technologies behind the tools needed to produce the workflows in our method. This covers package management, container

[1] https://kb.10xgenomics.com/hc/en-us/articles/360001378811-What-is-the-maximum-number-of-cells-that-can-be-profiled-

technologies and workflow management.

Demultiplexing Sequencer Output

As we discussed above, the raw sequencer output on modern sequencers are typically images. Each images contains colored pixels, depicting a nucleotide base in a positon on a flow cell. Bioinformatic software usually don't use these images as input. They operate on contiguous strings of letters representing the nucleotide sequence from a read. This pre-processing consists of two stages:

- **Base calling**: A specilized software reads the raw images from the sequencer output and for each position determines the color and the signal strength. Each position corresponds to a strand of DNA on the flow cell, the color will map to a base (A,T,C or G) and signal strength will determine the quality of the base. Images are read in a sequential manner to build a read. If the base could not be determined, a N is inserted.
- **Demultiplexing**: Each read is demultiplexed to a specific biological sample. This is done by mapping the index read to the corresponding sample. The information on which index sequence that belongs to which sample must be provided at runtime.

The reads are organized into fastq files. Typically, each sample will have one file for the forward read and one file for the reverse read. Figure 2.8 and the following describes the layout of a fastq file:

- **Header**: A header starting with @ containing the sequencer id, information on the position of the flow cell the read is read from and information of certain demultiplexer parameter, information on whether the read belongs to a pair of R1 and R2 reads, the index sequence.
- **The DNA read**: A string of A,T,C,G or N equal to the read length.
- **Separator**: A + sign.

- **The quality read**: A string of character encoding the quality of each of the bases in the DNA read. The string has the same length as the DNA read.

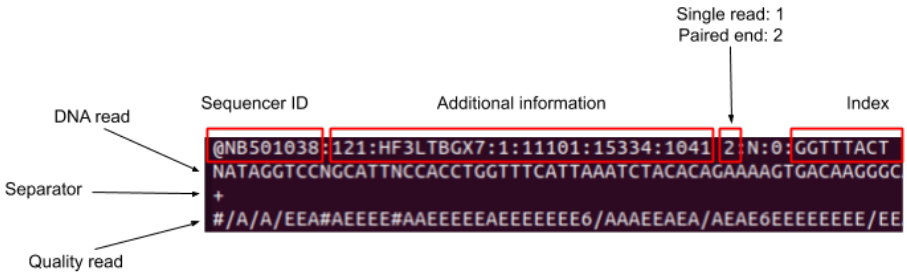


Figure 2.8: Fastq file format.

Sequence Alignment

In sequence alignment we map the sequence reads from the fastq files from a experiment to a *reference genome*. We want to find the locations in the DNA that the sequences belong to. These locations are in turn used to retrieve information of which genes the sequence belongs to by matching against a *gene annotation file*. Sequence alignment for RNA-seq data is challenging in two major ways. First, the sequences are very short compared to typical DNA reads. Second, mature mRNA molecules are spliced together from exons in the DNA. Because of this, a short read from a RNA molecule may map many different *loci* in. This means that a read may map to many different loci and that sequence from a RNA molecule may not to a contiguous part of the DNA. Because of the high volume output from NGS technology, the sequence alignment requires considerable computational efforts.

Many approaches exist to tackle these problems. Spliced Transcripts Alignment to a Reference (STAR)[Dob+13] is a highly efficient C++, standalone sequence aligner specifically aimed at solving these issues. The algorithm consists of two steps, seed searching and a scoring and stitching step.

In the seed searching, STAR search for a Maximal Mappable Prefix (MMP) sequentially. For a given sequence, S, and a reference genome sequence, G, MMP is the largest substring

in S that matches one or more locations in G. If S derives from mRNA that has one or more junctions, it cannot be mapped to a contiguous sequence in the reference genome. STAR first finds a MMP from the first base in S. Then, it finds a MMP for the remainder of S that was not part of the first match. This enables STAR to find matching locations with a single alignment pass over the reference genome. Furher, it can find all matching locations for multi mapping reads in the reference genome without any extra effort. When one alignment pass is done, STAR repeats the process from the other end of the reference. If the start of S is of low quality but the end has better quality, it is more likely that the reverse order pass will find MMPs. If the end of a read is not reached while searching for MMPs, STAR will look for alignments with a user specified amount of mismatches. If it still can not find good alignments, the tail will be clipped. To perform this searches STAR uses uncompressed *suffix arrays* (SA)[MM93]. SA is a string search algorithm that scales logarithmitcly in search time with reference genome length. Because SA data structure is uncompressed, it puts higher demand on memory. A human reference genome index uses roughly 30 GB of memory during a STAR run. Figure 2.9 shows an illustration of the pass over the reference genome index.

After the passes are finished, STAR chooses the good MMPs based on how many loci they align to. Those that are under a threshold for multiple alignments are used as anchor seeds. Seeds that has mapped within a user specified distance of these anchor seeds are then stitched together in attempt to create new reads. This allows STAR to detect unknown junctions. STAR uses a scoring scheme that can be specified for giving penalties to mismatches, insertions, deletions and splice junction gaps.

2.3.1 Package Management and Environments

Computer users often find themselves in need to install new software. Most tasks performed on a computers are realized through software implementations and their applications range from entertainment to research. Many applications can also be carried out different types software, each of them can require any amount of other software as a *dependency*. These dependencies can also be very specific and demand a exact version on series of software. Each of them in turn can have their own dependencies. This can be

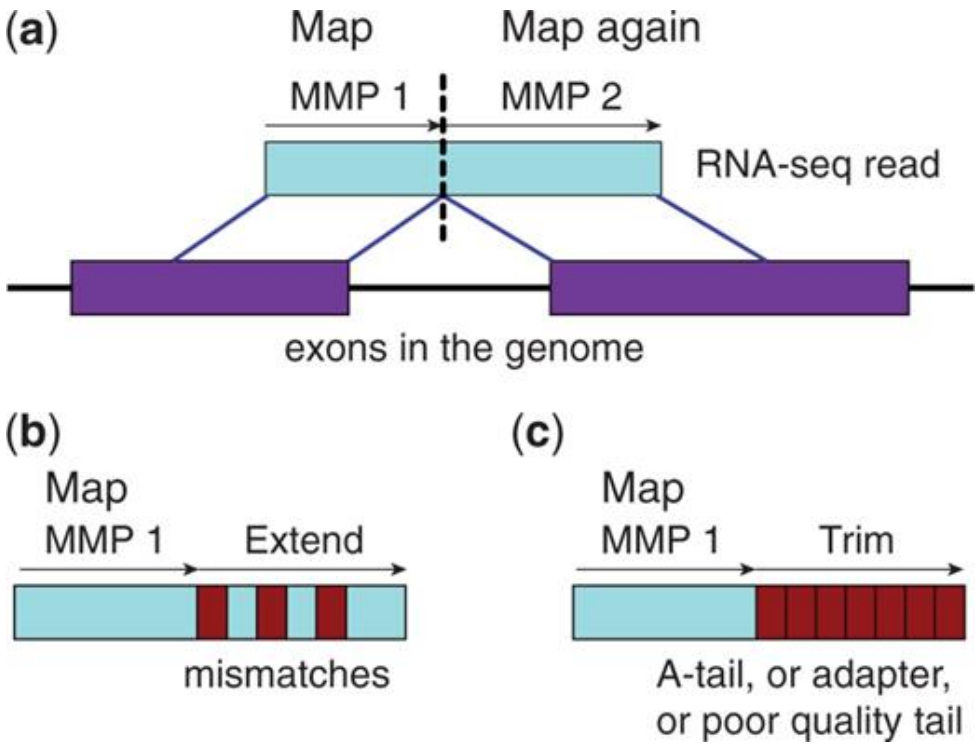


Figure 2.9: Illustration of pass over a reference genome index in star

depicted as a tree of dependencies, called a *dependency graph*. Nesting these graphs can be a tedious and error prone operation, causing a lot of frustration and even erroneous results. Even worse, the exact origin of such errors can be very hard to detect since they can happen in any layer of the software structure. They can happen far from the user interface. Another root to complexity is that two software may have overlapping software requirements, but the requirement differs in versions of the software. If a software is upgraded or downgraded while installing a new software, existing software stop working entirely or subtile errors may be introduced. The software must also be obtained somehow. This can be done through downloading or transfer files from other medium. Then manual installation or compiling follows. If you want a upgraded version, the process must be repeated. Another concern is reproducibility. This is especially important within research communities if results of one experiment is to be useful for other researchers. Software stacks that are manually built and maybe altered over time can be challenging to recreate perfectly.

Package managers aim to ease the burdens of these concerns. Package managers come in many different shapes and colors and are targeted towards solving different types of problems. Package managers make software, data or archive available as *packages*. Each

package is bundled with a manifest, describing versions, dependencies, build instructions and other necessities. Packages can be pre-built for different types of computer architectures or they may be a archive that must be compiled after the package is retrieved. A package manager keeps a record, often in the form of a database, of the packages installed together with their versions and dependencies. It can prevent new packages with conflicting dependencies of being installed or ask the user to specify the preferred outcome. The packages available through a package manager comes from *repositories*. One package manager may use several repositories and user may add or remove and even create own repositories by desire.

One of the earliest package managers where found in linux distributions in the early 1990's. One example of this is dpkg, which still finds its use today. These early package managers typically installs packages directly on the system and become available to all users. If multiple users have access to installing software on the system, one user can break the dependencies of other users on the system.

Another example of a package manager that has gained popularity in recent years is conda. It enables users to install software in an isolated environment without administrative priveliges on the system. By using isolated environments the packages are only available to the users with access to the directory of the installation. Each system user typically uses their home directories for installation. Each user can have several isolated envrionments, allowing them to quickly switch between software stacks or different versions of software. Whereas most package managers are specific to a operating system (OS), conda has support for a wide range of platforms.

Conda is a powerful tool in experimental development where new software in the frontiers of research have rapid version increments. It allows to quickly test new versions or new packages without breaking compatibility towards previous projects. Each project can have its own isolated environment. With conda it is also easy to export a manifest of the software stack of each envrionment. This allows for documentation and reproducibility at a high resolution. Community repositories are also easy to establish and include in conda. Within bioinformatics, a popular repository for conda is bioconda[Grü+18].

2.3.2 Container Technologies

Above we discussed concerns related to reproducibility in research fields and how isolated environments could ease this burden. Container technologies aims to solve this on another level within a computational environment, at the same enabling other useful features. Container achieves environment isolation through operating system virtualization. A running container attaches to operating system of the host computer. Within the container a isolated operating system is then available. Containers are created by building an *image*. An image becomes a container when it is executed and attaches to the host OS. When an image is built, a *base image* can be specified. Any other already existing can be used as a base. The base image can be a full scale linux distribution like Ubuntu or Debian or it can be specialized light images that are specifically designed to be minimal so it can downloaded and executed fast. When a image is built, we provide the instructions needed to install the software or applications we want available within the container. We can also specify run commands that will start an application when we execute the container.

Running containers can also communicate with other containers or any device connected to the host OS or machine. This is done by specifying channels or ports that should be open to the specific container. A container can connect to a wide range of operating systems as long as the image is built for the architecture of the machine. Containers achieve this by using a *conatiner engine*. This engine is the layer between the host OS and the virtualized OS. It translates instructions from the virtualized OS to instructions on the host OS and vice versa. This is visualized in Figure This makes applications bundled in containers extremely portable and easy to deploy. Applications that are distributed as containers will achieve highly reproducible results.

Two popular container technologies is Singularity and Docker. A Docker container requires elevated privileges to be executed. By default docker containers executes in a completely isolated namespace. Directories from the host can be mounted and ports can be opened for communication with other applications. Docker is specifically designed for deploying multiple micro applications and docker provides interfaces to orchestrate this. Because of the requirement of elevated privileges, Docker is more suited for system admin-

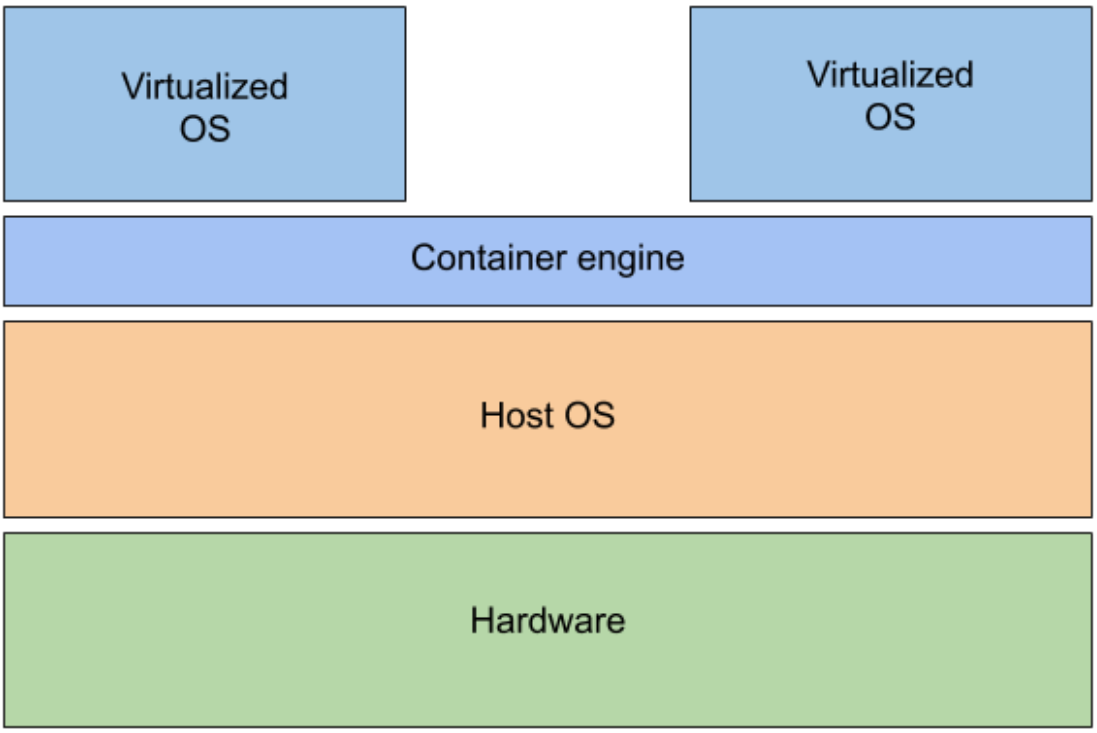


Figure 2.10: The conceptual layers involved in translating instructions from a virtualized OS to the hardware of the host machine.

istrators and developers that configures services. Singularity is designed specifically for isolating applications with complex software stacks and to execute scientific applications in reproducible and portable manner. Singularity runs with user privileges and share the same namespace as the user. Singularity can also access docker images, either directly from a repository or pre-built images existing on the host. Singularity is especially useful when performing computations on computational infrastructure. User typically do not have access to escalated privileges in such environments. Different projects may require using different infrastructures and platform. With applications bundled in Singularity containers, user need only download pre-built containers and deploy them.

2.3.3 Workflow Management

Other challenges developers of engineering or scientific applications often are faced with is to manage the data processing. From raw data to end product, there can be dozens of

steps involved. A series of steps that data flows through to reach a end product is called a *pipeline*. If a pipeline only needs one software stack, the pipeline can be implemented in bash script or python script in a straight-forward manner. But this is not always the case. Each step in a pipeline can potentially involve executing programs that each require complex software stacks. This is especially true in the field of bioinformatics where a jungle of highly specialized packages are available. These software stacks may have incompatible dependencies.

Snakemake[KR12] is a workflow manager that tackles this. It aims at enabling reproducible and scalable pipelines in human readable format. From the documentation[2], it is described by: *Snakemake workflows are essentially Python scripts extended by declarative code to define rules.* In each rule, a set of input files and a set of output files are specified. Each rule can specify a conda environment or a singularity file that the rule should be executed in. We then execute a pipeline by specifying a output file. Snakemake builds a directed acyclic graph (DAG) of the dependencies of the steps, or rules, involved in creating the output. An example DAG is given in Figure 2.11. Depending on the specified resource requirement of each rule, Snakemake can execute the rules in parallel. Snakemake also has the possibility to use wildcards in path names to generalize rules, we can write python inline and it has support for benchmarking the rules.

[2] https://snakemake.readthedocs.io/en/stable/

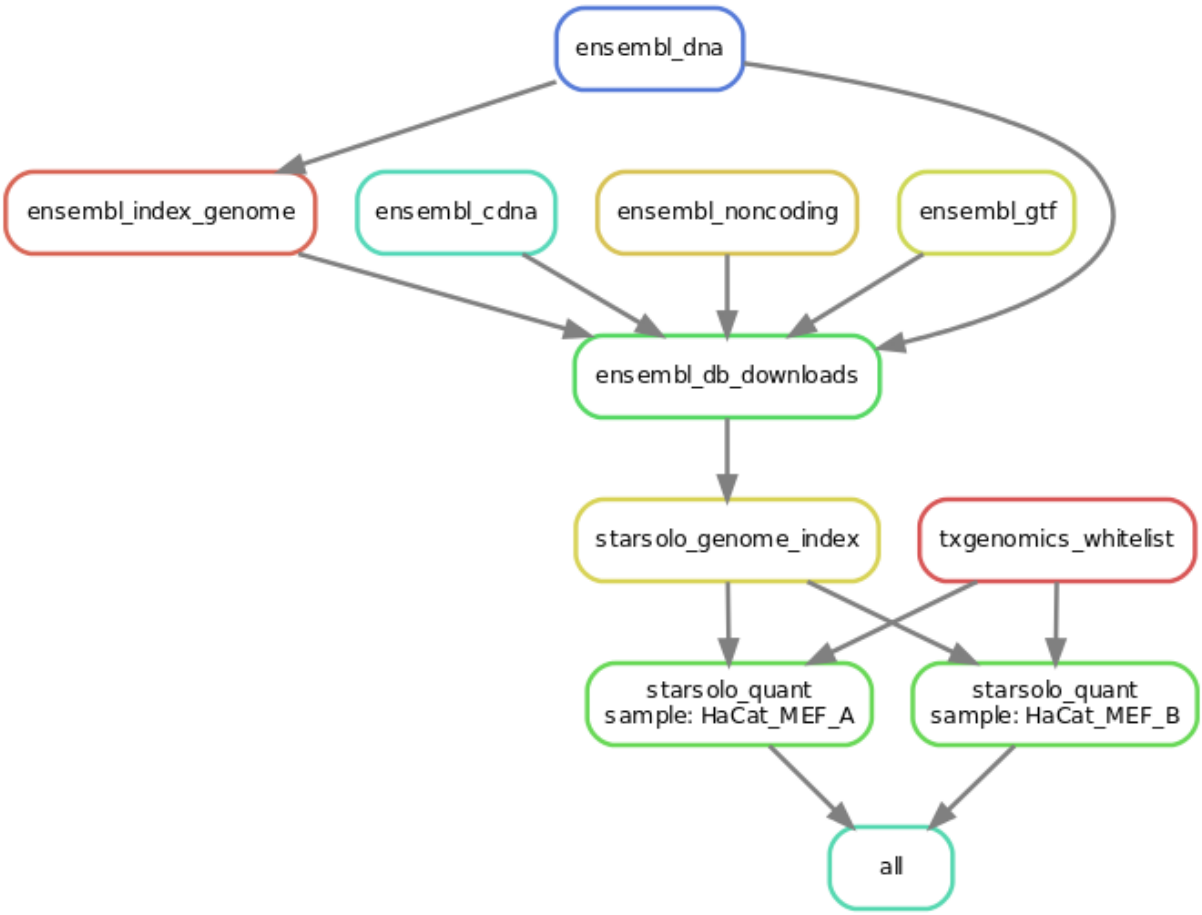


Figure 2.11: Snakemake builds a DAG of the dependencies between rules involved in creating the desired output. Here we build a count matrix from the fastq files of two single cell samples. A genome is downloaded and an index is built before quantification proceeds.

2.4 Mathematical Background

Here, we introduce the mathematical tools and concepts that we will need in our method. We cover periodic functions and trigonometry. Definitions from linear algebra is established and the eigenvalue decomposition of a matrix is presented. Finally, we introduce splines as a interpolation method. We show how a interpolating cubic spline can be constructed from solving a linear system of equations. Finally, the define the smoothing cubic spline as a optimization problem.

2.4.1 Periodic Functions

A function $f : \mathbb{X} \to \mathbb{Y}$ is said to be periodic if

$$f(x) = f(x + nT), \quad \text{for} \quad x, T \in \mathbb{X}, \quad n \in \mathbb{N}^+, \tag{2.1}$$

where $\mathbb{X}$ denotes the domain of f, $\mathbb{Y}$ denotes the range of f and $\mathbb{N}^+$ is the set of positive integers. We call T the *period* of f. Furher, if $f(x)$ is T-periodic, then

$$f(kx + \alpha), \quad \text{where} \quad kx, \alpha \in \mathbb{X}, \tag{2.2}$$

will have a period of $T/|k|$. We call α the *phase* of the periodic function. The phase will contribute to an offset in the values that f takes on as a function of x. Another useful term in describing periodic functions is the frequency, F. The frequency is defined as $F = \frac{1}{T}$. In engineering applications, the period will often be measured in seconds. The SI-unit for the frequency is Hz (Hertz) with dimension $[1/s]$.

We can manipulate any periodic function by stretching or shrinking the interval it takes to perform a full cycle by chosing an appropriate k. Let $g : \mathbb{R} \to [-1, 1]$ be a periodic function with $T = 2\pi$. If we want to model a periodic event in the shape of g over a 24-hour interval, we must chose a k that satisfies

$$\begin{aligned} 24 &= \frac{2\pi}{k} \\ k &= \frac{2\pi}{24}. \end{aligned}$$

The function $g(\frac{2\pi}{24}t)$ will be 24-hours periodic $\forall t \in \mathbb{R}$.

2.4.2 Trigonometry

In trigonometry, we study the relationships between the side lengths and the angles of triangles. In a right angle triangle, one angle will always be 90°. Given the right angle in Figure 2.12, defined by the corner points A, B and C, the side lengths a, b and c, and the angles α, β together with the right angle such that $\alpha + \beta + 90° = 180°$, we can define

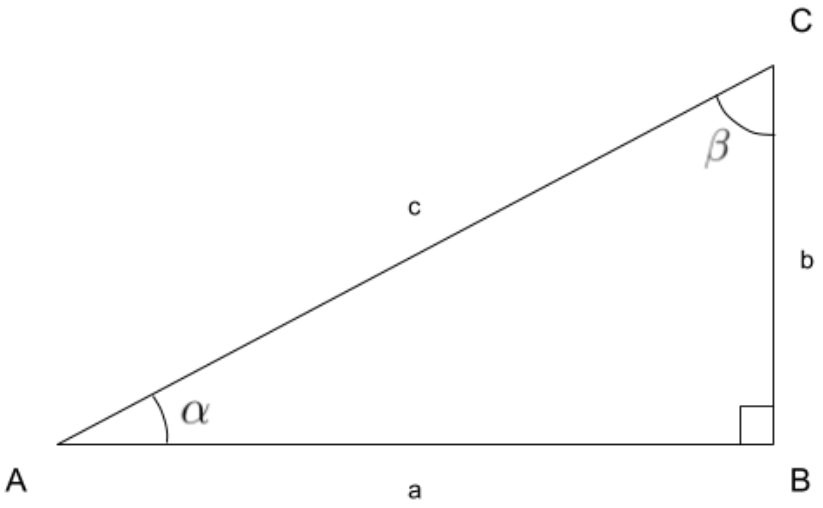


Figure 2.12: A right angled triangle defined by the three points, A, B and C, the side lengths a, b and c, and the angles α, β together with the right angle.

the relationships between the side lengths and the angles by the following trigonometric functions.

The *sine* function

$$\sin(\alpha) = \frac{b}{c}. \tag{2.3}$$

The *cosine* function

$$\cos(\alpha) = \frac{a}{c}. \tag{2.4}$$

And the *tangent* function

$$\tan(\alpha) = \frac{b}{a} = \frac{b/c}{a/c} = \frac{\sin(\alpha)}{\cos(\alpha)}. \tag{2.5}$$

Their inverse functions are known as the *arcsine*, the *arccosine* and the *arctangent*. They are useful if we know the relationship between the side lengths and want to find the angles of the triangle. They are defined by

$$\arcsin(\sin(\alpha)) = \arcsin(b/c) = \alpha, \tag{2.6}$$

$$\arccos(\cos(\alpha)) = \arccos(a/c) = \alpha, \tag{2.7}$$

$$\arctan(\tan(\alpha)) = \arctan(b/a) = \alpha. \tag{2.8}$$

Likewise, for the angle β, we have

$$\begin{aligned}
\sin(\beta) &= \frac{a}{c}, \\
\cos(\beta) &= \frac{b}{c}, \\
\tan(\beta) &= \frac{a}{b} = \frac{a/c}{b/c} = \frac{\sin(\beta)}{\cos(\beta)}.
\end{aligned}$$

Now, we see that

$$\begin{aligned}
\sin(\alpha) &= \cos(\beta) = \cos(90^\circ - \alpha), \\
\cos(\alpha) &= \sin(\beta) = \sin(90^\circ - \alpha).
\end{aligned}$$

Hence, for a given angle, the sine and the cosine have a *pahse offset* of 90°. These definitions are only valid for angles $\alpha \in [0^\circ, 90^\circ]$. We can extend the domain of these trigonometric functions to $\alpha \in \mathbb{R}$ by modelling them on a unit circle, that is, a circle with radius equal to 1. It is common to use *radians* as a unit of measure for the angles of periodic functions. One radian is the angle corresponding to an arc length equal to the radius of the circle. On the unit circle, a full circle will correspond to 2π. In Figure 2.13, we can see how the trigonometric functions are interpreted on the unit circle. The functions sin, cos and tan will be 2π-periodic functions with respect to the angle θ. Their inverse functions will only be one-to-one to parts of the domain because of the symmetry.

They are defined with the following domain and range

$$\cos : \mathbb{R} \to [-1, 1], \tag{2.9}$$

$$\sin : \mathbb{R} \to [-1, 1], \tag{2.10}$$

$$\tan : \mathbb{R} \to \mathbb{R}, \tag{2.11}$$

$$\arccos : [-1, 1] \to [0, \pi], \tag{2.12}$$

$$\arcsin : [-1, 1] \to [-\frac{\pi}{2}, \frac{\pi}{2}], \tag{2.13}$$

$$\arctan : \mathbb{R} \to (-\frac{\pi}{2}, \frac{\pi}{2}). \tag{2.14}$$

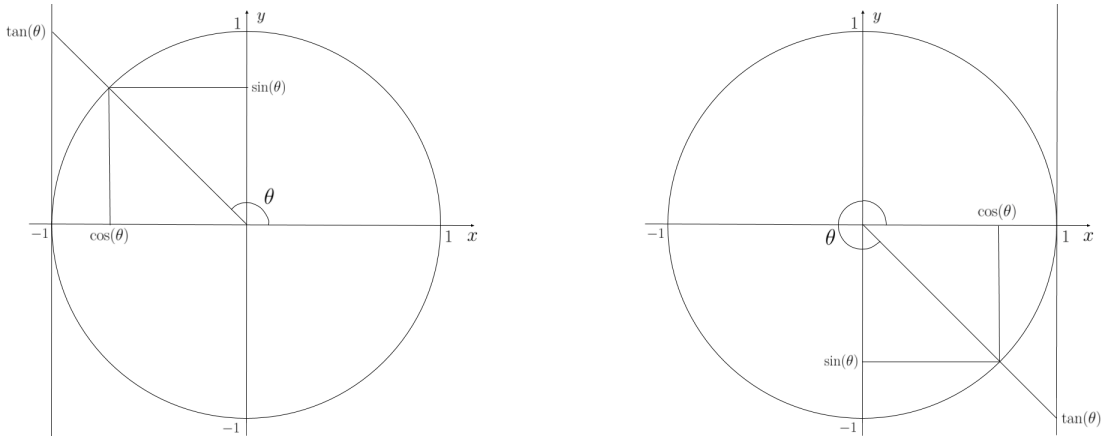


Figure 2.13: Two examples of a unit circle with an angle θ, $\cos(\theta)$ as the x-values, $\sin(\theta)$ as the y-values and $\tan(\theta)$ is interpreted as where an extension in the radial direction intersects the tangent of the unit circle at $\theta = 0$ or $\theta = \pi$.

2.4.3 Linear Algebra

Here, we will give a short introduction to some key concepts in the field of linear algebra. Some formal definitions that will enable the methods described in the following sections will be presented. Presentation will be based on the lectures from the course TMA4145 Linear Methods[3] given at NTNU, autumn 2013.

Linear algebra is a field of mathematics in which we study the governing rules and con-

[3] https://www.ntnu.edu/studies/courses/TMA4145

ventions used to describe linear equations, linear functions and their representations as vectors and matrices. We will use uppercase openfaced symbols ($\mathbb{X}$) to denote vector spaces. Boldfaced uppercase symbols ($\mathbf{X}$) denotes matrices. Lowercase boldfaced symbols ($\mathbf{x}$) denotes vectors, while lowercase unboldfaced symbols (x) denotes scalars.

Vector Spaces

A *real vector space* is a set $\mathbb{X}$ equipped with an operation called addition,

$$\mathbb{X} \times \mathbb{X} \to \mathbb{X}, \quad (\mathbf{x}, \mathbf{y}) \mapsto \mathbf{x} + \mathbf{y},$$

an operation called scalar multiplication,

$$\mathbb{R} \times \mathbb{X} \to \mathbb{X} \quad (\lambda, \mathbf{x}) \mapsto \lambda\mathbf{x},$$

an element $0 \in \mathbb{X}$ called the *zero vector*, and for each $x \in \mathbb{X}$ an additive inverse $-\mathbf{x} \in \mathbb{X}$, such that $\forall \mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{X}$ and scalars $\lambda, \mu \in \mathbb{R}$ the following properties hold:

$$\begin{aligned}
\text{Additive identity:} \quad & \mathbf{x} + 0 = \mathbf{x}, \\
\text{Additive inverse:} \quad & \mathbf{x} + (-\mathbf{x}) = 0, \\
\text{Symmetry:} \quad & \mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}, \\
\text{Associativity:} \quad & \mathbf{x} + (\mathbf{y} + \mathbf{z}) = (\mathbf{x} + \mathbf{y}) + \mathbf{z}, \\
\text{Multiplicative identity:} \quad & 1\mathbf{x} = \mathbf{x}, \\
\text{Compatibility:} \quad & \mu(\lambda\mathbf{x}) = (\mu\lambda)\mathbf{x}, \\
\text{Distributivity:} \quad & \lambda(\mathbf{x} + \mathbf{y}) = \lambda\mathbf{x} + \lambda\mathbf{y}, \\
\text{Distributivity:} \quad & (\lambda + \mu)\mathbf{x} = \lambda\mathbf{x} + \mu\mathbf{x}.
\end{aligned}$$

The elements in $\mathbb{X}$ are called vectors. A vector space is called a real vector space if the field of scalars is $\mathbb{R}$. A vector space over the field of complex numbers $\mathbb{C}$ is called a complex vector space. We will only be concerned with real vector spaces in the following sections.

Linear Subspaces

Let $\mathbb{X}$ be a vector space. A subset $S \subset \mathbb{X}$ is a *subspace* of $\mathbb{X}$ if it is closed under linear operations. That is,

$$S \text{ is a subspace of } \mathbb{X} \iff \lambda\mathbf{x} + \mu\mathbf{y} \in S \text{ whenever } \mathbf{x}, \mathbf{y} \in S \text{ and } \lambda, \mu \in \mathbb{R}.$$

In particular, the zero-set $\{0\}$ is a subspace in any vector space since

$$\lambda 0 + \mu 0 = 0 \in \{0\}, \quad \forall \lambda, \mu \in \mathbb{R}.$$

Linear Dependence

A *linear combination* of vectors $\mathbf{x}_1, ..., \mathbf{x}_n$ is a finite sum

$$\sum_j^n a_j \mathbf{x}_j,$$

where $a_1, ..., a_n$ are scalars.

A set of vectors $\{\mathbf{v}_1, \mathbf{v}_2, ...\}$ is called *linearly dependent* if any of the vectors is a linear combination of some of the other vectors.

$$\{\mathbf{v}_1, \mathbf{v}_2, ...\} \text{ linearly dependent} \iff \sum_{j=1}^n a_j \mathbf{v}_j = 0 \quad n \in \mathbb{N}, \text{ at least one } a_j \neq 0.$$

Otherwise, the set is *linearly independent*.

$$\{\mathbf{v}_1, \mathbf{v}_2, ...\} \text{ linearly independent} \iff \left[\forall n \in \mathbb{N} : \sum_{j=1}^n a_j \mathbf{v}_j = 0 \Rightarrow a_j = 0 \quad \forall j \right].$$

That is, a linear combination of a linearly independent set of vectors will amount to zero if, and only if, all the scalars are exactly zero.

Linear Transformations

Let $\mathbb{X}$ and $\mathbb{Y}$ be real vector spaces, and let $T : \mathbb{X} \to \mathbb{Y}$ be a mapping between them. We say that T is a *linear transformation* if it preserves the linear structure of a vector space. That is

$$T \text{ linear} \iff T(\lambda\mathbf{x} + \mu\mathbf{y}) = \lambda T\mathbf{x} + \mu T\mathbf{y}, \quad \mathbf{x}, \mathbf{y} \in \mathbb{X}, \lambda, \mu \in \mathbb{R}.$$

Any real valued *matrix* $A \in \mathbb{M}_{m\times n}(\mathbb{R})$ defines a linear transformation $\mathbb{R}^n \to \mathbb{R}^m$:

$$\underset{\mathbf{x}}{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}} \mapsto \underset{\mathbf{A}}{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}} \underset{\mathbf{x}}{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}} = \underset{\mathbf{Ax}}{\begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \end{bmatrix}}$$

Inner Product Spaces

Let $\mathbb{X}$ be a real vector space and let $\mathbf{x}, \mathbf{y} \in \mathbb{X}$. An *inner product* on $\mathbb{X}$, $\langle \dot{;} \dot{\rangle}$, is a map $\mathbb{X} \times \mathbb{X} \to \mathbb{R}$ that satisfies

$$\begin{aligned} \text{Symmetric:} \quad & \langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle, \\ \text{Linear in its first argument:} \quad & \langle \lambda\mathbf{x}, \mathbf{y} \rangle = \lambda \langle \mathbf{x}, \mathbf{y} \rangle, \\ & \langle \mathbf{x} + \mathbf{y}, \mathbf{z} \rangle = \langle \mathbf{x}, \mathbf{z} \rangle + \langle \mathbf{y}, \mathbf{z} \rangle, \\ \text{Positive definite:} \quad & \langle \mathbf{x}, \mathbf{x} \rangle > 0 \quad \text{for } \mathbf{x} \neq 0, \end{aligned}$$

for $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{X}$ and $\lambda \in \mathbb{R}$. Any map that satisfies this can be chosen. Usually, the *dot product* in $\mathbb{R}^n$ is chosen. This is defined by

$$\langle \mathbf{x}, \mathbf{y} \rangle := \sum_{i=1}^{n} x_i y_i.$$

Further, two vectors, $(\mathbf{x}, \mathbf{y})$ is said to be *orthogonal* if they satisfy

$$\langle \mathbf{x}, \mathbf{y} \rangle = 0.$$

Any vector space equipped with a norm is called an *inner product space*.

Normed Spaces

It can be proven that any inner product space $(\mathbb{X}, \quad \langle \cdot, \cdot \rangle)$ yields a natural norm given by

$$||\mathbf{x}|| := (\langle \mathbf{x}, \mathbf{x} \rangle)^{1/2}.$$

If we choose the dot product as our inner product, this gives

$$||\mathbf{x}|| := \langle \mathbf{x}, \mathbf{x} \rangle^{1/2} = \sqrt{\sum_{i=1}^{n} x_i^2}.$$

This is also known as the *Euclidian norm*, often simply denoted by $|| \cdot ||_2$. For any real vector $\mathbf{x}$, this is interpreted as its geometrical distance from the origin.

2.4.4 Eigenvalue Decomposition

Let $\mathbf{A} \in \mathbb{M}_{n \times n}(\mathbb{R})$ be a real valued and square matrix with dimensions $n \times n$. We can define the concepts of *eigenvalues* and *eigenvectors* in terms of the equation

$$\mathbf{A}\mathbf{v} = \lambda \mathbf{v}. \tag{2.15}$$

An eigenvalue, λ, of the matrix $\mathbf{A}$, is a number λ such that Equation (2.15) has a solution $\mathbf{v} \neq 0$. The vector v is called an eigenvector of $\mathbf{A}$, corresponding to the eigenvalue λ. The eigenvectors corresponding to an eigenvalue λ and the zero-vector, form a vector subspace of $\mathbb{X}$. This vector subspace is called the *eigenspace* of $\mathbf{A}$ corresponding to the eigenvalue λ. The set of all eigenvalues of $\mathbf{A}$, $\sigma(\mathbf{A})$, is called the *spectrum* of $\mathbf{A}$.

From this definition, we see that the eigenvectors of $\mathbf{A}$ are those vectors that are only scaled by a constant λ (the eigenvalue corresponding to that eigenvector) when we apply the linear transformation $\mathbf{A}$ on them. An illustration of the application of $\mathbf{A}$ on an eigenvector $\mathbf{v}$ can be seen in Figure 2.14.

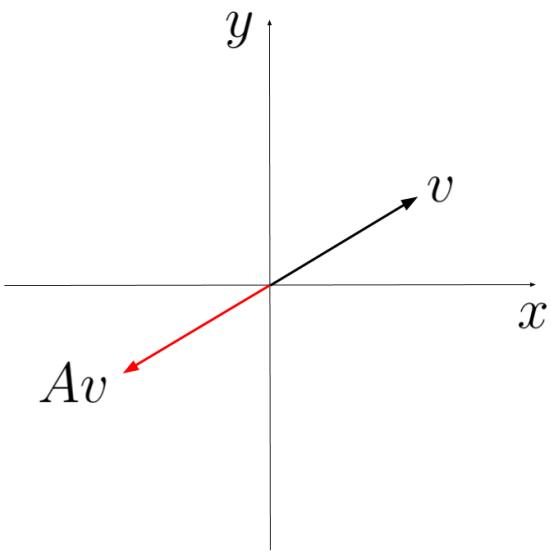


Figure 2.14: Illustration of an application of the linear transformation $\mathbf{A}$ on an eigenvector, $\mathbf{v}$, of $\mathbf{A}$. In this example, $\mathbf{Av} = \lambda\mathbf{v}$ with $\lambda = -1$.

The eigenvalues of the matrix $\mathbf{A}$ can be found by obtaining the roots of the *characteristic polynomial* of $\mathbf{A}$. We define this in terms of Equation (2.15) by rewriting it to

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{v} = 0,$$

where $\mathbf{I}$ is a $n \times n$ matrix that is 1 on its diagonal elements and zeros everywhere else, also called the identity matrix of dimension $n \times n$. For any vector $\mathbf{v} \neq 0$, the determinant $\det(\mathbf{A} - \lambda\mathbf{I})$ must then be zero. By developing this determinant, we will obtain a polynomial of degree n in terms of λ, called the characteristic polynomial of A. Hence, A will have at least one eigenvalue and at most n eigenvalues. In practice, it is inconvenient to explicitly solve this polynomial by direct calculation for n larger than about 4. It is more common to employ numerical methods to approximate the eigenvalues in such cases.

Let $\mathbf{A} \in \mathbb{M}_{n\times n}(\mathbb{R})$ and let $\mathbf{v}_i$ denote the eigenvectors of $\mathbf{A}$. If $\mathbf{A}$ has n linearly independent eigenvectors corresponding to eigenvalues λ_i (not necessarily distinct). Let

$$\mathbf{Q} = \begin{bmatrix} \mathbf{v_1} & \mathbf{v_2} & \cdots & \mathbf{v_n} \end{bmatrix}$$

be the $n \times n$ matrix with the eigenvectors as its column vectors. Then, we can write

$$\begin{aligned}\mathbf{AQ} &= \mathbf{A}\begin{bmatrix}\mathbf{v_1} & \mathbf{v_2} & \cdots & \mathbf{v_n}\end{bmatrix} \\ &= \begin{bmatrix}\mathbf{Av_1} & \mathbf{Av_2} & \cdots & \mathbf{Av_n}\end{bmatrix} \\ &= \begin{bmatrix}\lambda_1\mathbf{v_1} & \lambda_2\mathbf{v_2} & \cdots & \lambda_n\mathbf{v_n}\end{bmatrix}.\end{aligned}$$

Further, let

$$\mathbf{\Lambda} = \begin{bmatrix}\lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n\end{bmatrix}.$$

Since $\mathbf{Q}$ is $n \times n$ and consists of n linearly independent column vectors, it can be shown that it is invertible. Then we can write

$$\begin{aligned}\mathbf{AQ} &= \mathbf{Q\Lambda} \\ \mathbf{AQQ}^{-1} &= \mathbf{Q\Lambda Q}^{-1} \\ \mathbf{A} &= \mathbf{Q\Lambda Q}^{-1}.\end{aligned} \tag{2.16}$$

This form is called the *eigendecomposition* of $\mathbf{A}$.

Eigenvalue decomposition also has a useful property for symmetric matrices. Let $\mathbf{A}$ be a real, symmetric matrix such that $\mathbf{A} = \mathbf{A}^T$ and let $(\lambda_i, \mathbf{v}_i)$, $(\lambda_j, \mathbf{v}_j)$ be eigenvalue-eigenvector pairs with $\lambda_i \neq \lambda_j$. Then, we have

$$\mathbf{v}_i^T\mathbf{A}\mathbf{v}_j = (\mathbf{v}_i^T\mathbf{A}\mathbf{v}_j)^T = \mathbf{v}_j^T\mathbf{A}\mathbf{v}_i,$$

by using the symmetry of $\mathbf{A}$. Then, we get

$$\begin{aligned}\mathbf{v}_i^T\mathbf{A}\mathbf{v}_j = \mathbf{v}_i^T\lambda_j\mathbf{v}_j = \lambda_j\mathbf{v}_i^T\mathbf{v}_j \quad &\text{and} \\ \mathbf{v}_j^T\mathbf{A}\mathbf{v}_i = \mathbf{v}_j^T\lambda_i\mathbf{v}_i = \lambda_i\mathbf{v}_j^T\mathbf{v}_i.&\end{aligned}$$

Since we have that $\lambda_i \neq \lambda_j$ and $\lambda_i\mathbf{v}_j^T\mathbf{v}_i = \lambda_j\mathbf{v}_i^T\mathbf{v}_j$, then we must have that $\mathbf{v}_i^T\mathbf{v}_j = 0$. Hence, the eigenvectors of a symmetric matrix form an orthogonal basis. If $A \in \mathbb{M}_{n\times n}(\mathbb{R})$, they form an orthogonal basis of $\mathbb{R}^n$. If we divide each eigenvector by its own length, the

set of all eigenvectors of $\mathbf{A}$ is said to form an *orthonormal* basis of $\mathbb{R}^n$.

2.4.5 Smoothing Cubic Splines

Splines refer to an interpolation technique that has many applications in Computer Aided Design and computer graphics. Using spline interpolation is preferred to polynomial interpolation because it often yields smoother results. By using piecewise polynomials, splines can achieve this with a lower polynomial degree.

Assume we are given a set of data points $(x_0, f_0), (x_1, f_1), \ldots, (x_n, f_n)$, called our *control points*, from some function $f : \mathbb{R} \mapsto \mathbb{R}$ such that $f(x_i) = f_i$. In many applications, one seeks to approximate f on the intervals connecting the data points. This is called interpolation. We can construct a polynomial $P_k(x)$ of degree $k \leq n$ that intersects our data points. If k is large, these approximations tends to oscillate drastically and their approximation of f tends to be poor. With splines, rather than constructing a single polynimal of degree k over the entire domain $[x_0, x_n]$, we construct piecewise polynomials of a much lower degree that are defined on intervals in between our control points. This lower degree polynomials are then joined together so that they form a continuous curve on $[x_0, x_n]$. This is called a *piecewise polynomial interpolation*.

When performing spline interpolation, we must choose a degree of our piecewise polynomials. A typical choice for degree in many applications is $k = 3$. This is because polynomials of degree 3 have both continuous first and second derivatives. This enables smooth curves that are easy to join together without any visible effects. Let $q_i(x)$ be polynomials of degree 3 and let

$$g(x) = q_i(x) \quad \text{when } x \in [x_i, x_{i+1}]$$

and assume $x_i < x_{i+1}$ for $i \in [0, n-1]$. Further, we require that $g(x_i) = f(x_i) = f_i$ together with $g'(x_0) = k_0$ and $g'(x_n) = k_n$ for any two given k_0, k_n. Then, the *Existence and Uniqueness Theorem of Cubic Splines* states that there exists *one and only one* cubic spline $g(x)$ satisfying the above conditions. For $i \in [0, n-1]$, we define $c_i = 1/h_i =$

$1/(x_{i+1} - x_i)$ and require

$$\begin{aligned} q_i(x_i) &= f(x_i), \\ q_i(x_{i+1}) &= f(x_{i+1}), \\ q_i'(x_i) &= k_i, \\ q_i'(x_{i+1}) &= k_{i+1}, \end{aligned}$$

where q_i' is the first derivative of q_i and k_i is to be determined for $i \in [1, n-1]$. For the second derivative, we have the following conditon

$$q_{i-1}''(x_i) = q_i''(x_i) \quad \text{for } i \in [1, n-1].$$

Then, it can be shown that the system determined by the following system of linear equations has a unique solution for $\mathbf{k} = [\, k_1 \quad \dots \quad k_{n-1} \,]$:

$$c_{i-1}k_{i-1} + 2(c_{i-1} + c_i)k_i + c_i k_{i+1} = 3\left[c_{i-1}^2(f_i - f_{i-1}) + c_i(f_{i+1-f_i})\right] \tag{2.17}$$

for $i \in [1, n-1]$. Now, with given k_0, k_n and $\mathbf{k}$ determined by the above, we define the piecewise polynomials by

$$q_i(x) = a_{i0} + a_{i1}(x - x_i) + a_{i2}(x - x_i)^2 + a_{i3}(x - x_i)^3,$$

for $i \in [0, n-1]$. It can be shown that the following coefficients satisfy our conditions:

$$\begin{aligned} a_{i0} &= q_i(x_i) = f_i, \\ a_{i1} &= q_i'(x_i) = k_i, \\ a_{i2} &= \frac{q_i''(x_i)}{2} = \frac{3}{h_i^2}(f_{i+1} - f_i) - \frac{1}{h_i}(k_{i+1} + 2k_i), \\ a_{i3} &= \frac{q_i'''(x_i)}{6} = \frac{2}{h_i^3}(f_i - f_{i+1}) + \frac{1}{h_i^2}(k_{i+1} + k_i). \end{aligned}$$

One way to alter the behaviour of an interpolating cubic spline is to alter its end conditions. If we require that $g''(x_0) = g''(x_n)$, this is called the *natural conditions*.

As we saw, this way of constructing a cubic spline will interpolate the values of $f(x)$ exactly at all control points. Another form of the cubic spline that has many applications in data analysis is the *smoothing cubic spline*. In smoothing cubic splines, we free the spline of the requirement that it must interpolate the values of $f(x)$ at the control points. Rather, we introduce an objective function that it should minimize over the domain. Let $\hat{f}$ be a cubic spline freed of its interpolation requirements and let our control points be as previously defined. Our smoothing cubic spline will then be the suitable function satisfying

$$g = \arg\min_{\hat{f}} \left[pR(\hat{f}) + (1-p)C(\hat{f}) \right], \tag{2.18}$$

where R and C are defined by

$$R(\hat{f}) = \sum_{i=0}^{n} (f_i - \hat{f}(x_i))^2,$$
$$C(\hat{f}) = \int_{x_0}^{x_n} ||\hat{f}(t)||^2 dt,$$

and p is the *smoothing paramater*, to be determined somehow. The term R only accounts the values that $\hat{f}$ takes on at the control points. It favors candidate functions that have values close to values specified by the control points. On the other hand, C accounts for the characteristics of $\hat{f}$ over the whole domain determined by our control points. It will give penalty to candidate functions with large second derivatives. This means that the more oscillatory the candidate function is, the larger the contribution will be from this factor. We see that if we choose $p = 1$, then R determines our optimization problem completely and our solution becomes the interpolating spline from above. If we choose $p = 0$, then C becomes paramount. In this case, our solution will be the least squares line through the control points.

Since an interpolating spline still may yield oscillations, smoothing splines can be used when require smoother approximations of $f(x)$. This will come with a cost of loosing accuracy, but not all applications require a perfect interpolation at the control points. For example, if we wish to to represent a trend for a stock share that has many local extrema, fitting a smoothing cubic spline would be good candidate for a strategy. As we will see later in this thesis, it also find application in modelling gene expression profiles.

2.5 Statistical Methods

In this section, we will introduce some statistical principles and methods that we will need in this experiment. Unboldfaced upper case symbols will represent random variables while boldfaced upper case symbols represent matrices. Boldfaced lower case symbols represent vectors and unboldfaced lower case symbols represent scalars.

2.5.1 Covariance and Correlation

Let $X = [\, x_1 \quad x_2 \quad \dots \quad x_n \,]^T$ and $Y = [\, y_1 \quad y_2 \quad \dots \quad y_n \,]^T$ be two discrete and random variables, represented as vectors. The mean, μ_X, and variance, σ_X^2, is defined by

$$\mu_X = \frac{1}{n}\sum_{i=1}^{n} x_i$$
$$\sigma_X^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_x)^2,$$

where n is called the sample size. The *covariance* of two random vectors will be a measure of how much they vary together. The covariance between X and Y, $\sigma(X, Y)$, is given by

$$\sigma(X, Y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y).$$

Let $\mathbf{X} \in \mathbb{M}_{n\times p}(\mathbb{R})$ represent the set of p random variables of n observations, $\mathbf{X} = [\, X_1 \quad X_2 \quad \dots \quad X_p \,]$. The random variables, X_i, will be the column vectors of $\mathbf{X}$. Then, we can build the *variance-covariance* matrix, $\mathrm{Cov}(\mathbf{X}) = \mathbf{C}$, with elements given by $c_{ij} = \sigma(X_i, X_j)$. $\mathbf{C}$ will be both square and symmetric and the variance of the random variables will be the diagonal elements in $\mathbf{C}$. Usually, $\mathbf{C}$ is just called the covariance matrix. Suppose the random variables of $\mathbf{X}$ has zero mean. This can be achieved by subtracting the mean, μ_i, from each of the corresponding random variables, X_i of $\mathbf{X}$. Then, we can calculate the covariance matrix by

$$\mathbf{C} = \frac{1}{n-1}\mathbf{X}\mathbf{X}^T. \tag{2.19}$$

From this, it follows that for a linear combination of random variables, $\mathbf{a}^T\mathbf{X} = [a_1X_1 + a_2X_2 + \cdots + a_pX_p]$, its covariance matrix is given by

$$\begin{aligned} \text{Cov}(\mathbf{a}^T\mathbf{X}) &= \frac{1}{n-1}\mathbf{a}^T\mathbf{X}(\mathbf{a}^T\mathbf{X})^T \\ &= \frac{1}{n-1}\mathbf{a}^T\mathbf{X}(\mathbf{X}^T\mathbf{a}) \\ &= \mathbf{a}^T\mathbf{C}\mathbf{a}. \end{aligned} \tag{2.20}$$

From the covariance matrix, we can define the *correlation matrix*, $\mathbf{R}$. The elements of $\mathbf{R}$ is given by

$$\rho_{ij} = \frac{\sigma_{ij}}{\sqrt{\sigma_{ii}}\sqrt{\sigma_{jj}}}. \tag{2.21}$$

Each element, ρ_{ij}, of $\mathbf{R}$ measures the *linear association* between X_i and X_j. The elements will take values, $\rho_{ij} \in [-1, 1]$, where 1 is total positive linear correlation, -1 is total negative linear correlation and 0 means no linear correlation.

2.5.2 Principal Component Analysis

Principal Component Anlyis (PCA) is a *unsupervised dimensional reduction technique*. It is often employed as an exploratory technique in data sets that have independent observations from many correlated variables. PCA explains the variance-covariance structure of a data set through a reduced set of linear combinations of the variables, called the principal components. PCA can often reveal complex global structures from a higher dimension space by observing patterns in the lower dimension space of the principal components. This is achieved by finding the directions with maximum variability in the space determined by the original variables as its axes. The k directions in this system that explains the directions of the k highest variabilities are called the k-first principal components of the data set.

The components are chosen so the first principal component explains the highest variability. The second component is chosen so that it explains the direction of second highest variability under the condition that it is orthogonal to the first principal component, and so on. By rotating the data set into the new system of the k-first principal components, we

can *often* explain *almost* as much of the variability as in the original covariance structure.

Let $\mathbf{X} = [\,\mathbf{x}_1 \quad \mathbf{x}_2 \quad \dots \quad \mathbf{x}_p\,]$ be a system of p variables, each consisting of n observations such that $\mathbf{X} \in \mathbb{M}_{n\times p}(\mathbb{R})$. Assume that $\mathbf{X}$ is mean centered. Further, let $\mathbf{C} \in \mathbb{M}_{p\times p}(\mathbb{R})$ be the covariance matrix of $\mathbf{X}$ and let $(\lambda_i, \mathbf{v}_i)$ be the pairs of eigenvalues and eigenvectors of $\mathbf{C}$ such that Equation 2.16 holds with A as $\mathbf{C}$. Let us choose the ith principal component of $\mathbf{X}$, $\mathbf{t}_i$, as follows

$$\mathbf{t}_i = \mathbf{v}_i^T\mathbf{X} = v_{i1}\mathbf{x}_1 + v_{i2}\mathbf{x}_2 + \cdots + v_{ip}\mathbf{x}_p.$$

Then, since $\mathbf{X}$ is mean centered, we have that

$$\begin{aligned}\sigma^2_{\mathbf{t}_i} &= \sum_{j=1}^{p}(v_{ij}X_j)^2 = (\mathbf{v}_i^T\mathbf{X})(\mathbf{v}_i\mathbf{X})^T \\ &= \mathbf{v}_i^T\mathbf{X}\mathbf{X}^T\mathbf{v}_i \\ &= \mathbf{v}_i^T\mathbf{C}\mathbf{v}_i \\ &= \lambda_i.\end{aligned}$$

Here, the last step follows by Equation 2.16. Note that the scaling by sample size $1/(n-1)$ has been omitted. As will become clear later, it will not matter in this application. Now, by choosing two arbitrary principal components, $\mathbf{t}_i$ and $\mathbf{t}_k$ with $i \neq k$, we find their covariance by using the same elaboration as above to be

$$\begin{aligned}\mathrm{Cov}(\mathbf{t}_i, \mathbf{t}_k) &= (\mathbf{v}_i^T\mathbf{X})(\mathbf{v}_k\mathbf{X})^T = \mathbf{v}_i^T\mathbf{X}\mathbf{X}^T\mathbf{v}_k \\ &= \mathbf{v}_i^T\mathbf{C}\mathbf{v}_k = \lambda_k\mathbf{v}_i^T\mathbf{v}_k \\ &= 0.\end{aligned}$$

The last step follows from the orthogonality of eigenvectors of symmetric matrices.

From this, we can choose the first principal component of $\mathbf{X}$ by first finding the largest eigenvalue of its covariance matrix. Then, we construct the first principal component by selecting the corresponding eigenvector and then for each observation of $\mathbf{X}$ finding the projection in this direction. It is also common to require that the eigenvectors of $\mathbf{C}$ are

unit vectors. Then this can be formulated as

$$\mathbf{v}_1 = \arg\max_{||\mathbf{v}||=1} \mathbf{v}^T \mathbf{X}\mathbf{X}^T \mathbf{v}. \tag{2.22}$$

In PCA, we call $\mathbf{v}_i$ the *loading vectors* of $\mathbf{X}$. Each component of $\mathbf{v}_i$ is called the *loadings*. The loadings corresponds to the importance a particular variable has when transforming the observations of $\mathbf{X}$. The greater absolute value the loading has, the more it contributes to the explained variance in the direction of the loading vector.

Let $\mathbf{T} \in \mathbb{M}_{n\times k}(\mathbb{R})$ consist of the k-first principal components as its column vectors. Further, let $\mathbf{P} \in \mathbb{M}_{p\times k}(\mathbb{R})$ be the matrix consisting of the corresponding k-first loading vectors as its column vectors. Now, we can write

$$\mathbf{X} = \mathbf{T}\mathbf{P}^T + \mathbf{E}, \tag{2.23}$$

where $\mathbf{E}$ is the residuals, or the error, of the approximation.

It is common to investigate the proportion of total variance explained by each principal component. This can be done by first finding the eigenvalue decomposition of $\mathbf{C}$, and then for each principal component i calculate

$$\frac{\lambda_i}{\sum_{j=1}^{p} \lambda_j}.$$

In data sets with many correlated variables, only a few principal components are needed to explain a large proportion of the variance in $\mathbf{X}$. How many will depend on the particular data set and the accuracy needed for the application. By increasing the number of principal components, we can explain a larger proportion of the covariance structure, but interpretation and visualization gets increasingly difficult.

As briefly mentioned in Chapter 2.4.4, determining exact eigenvalues for large systems are infeasible. In practice, we use methods like NIPALS[Wol75] or matrix factorization methods like *Singular Value Decomposition*(SVD)[Jol11] or even approximations of it, like randomized SVD[Mar+10].

2.5.3 Partial Least Squares Regression

In PCA, we saw that if we had one data set $\mathbf{X}$, we could use its covariance matrix to reduce the dimensionality of $\mathbf{X}$ while conserving as much of the original variation as possible. In a Partial Least Squares regression (PLS) we want to relate a system of variables and observations to a *response*, $\mathbf{y}$. The response can also be multivariate, in which case we relate it to the multidimensional $\mathbf{Y}$. This is typically called PLS2.

Let $\mathbf{X}$ be $n \times p$, consisting of n independent observations of p variables. Let $\mathbf{Y}$ be $n \times q$ consist of q response variables, also with n observations. PLS is based on the decomposition formulated in Equation 2.23, but PLS differs in that we consider the nature of the response variable when we construct the principal components. Hence, PLS is called a *supervised* method.

By Equation 2.23, we transform $\mathbf{X}$ and $\mathbf{Y}$ to

$$\mathbf{Y} = \mathbf{T}\mathbf{Q}^T + \mathbf{F}, \tag{2.24}$$

$$\mathbf{X} = \mathbf{T}\mathbf{P}^T + \mathbf{E}, \tag{2.25}$$

where $\mathbf{T}$ is $n \times k$ gives the k-first principal components of $\mathbf{X}$, $\mathbf{P}$ is $p \times k$ and $\mathbf{Q}$ is $p \times q$. The matrices of error $\mathbf{F}$ is $n \times q$ and $\mathbf{E}$ is $n \times p$. It can be shown that, for a non-singular $p \times k$ matrix $\mathbf{W}$,

$$\mathbf{T} = \mathbf{X}\mathbf{W},$$

will also satisfy the above. After $\mathbf{T}$ is constructed, we find $\mathbf{Q}^T$ by the least squares solution of Equation 2.24, such that

$$\mathbf{Q}^T = (\mathbf{T}^T\mathbf{T})^{-1}\mathbf{T}^T\mathbf{Y}.$$

Then, the matrix $\mathbf{B}$ of the regression coefficients of $\mathbf{Y} = \mathbf{X}\mathbf{B} + \mathbf{F}$ can be constructed by

$$\mathbf{B} = \mathbf{W}\mathbf{Q}^T = \mathbf{W}(\mathbf{T}^T\mathbf{T})^{-1}\mathbf{T}^T\mathbf{Y},$$

together with the fitted response matrix $\hat{\mathbf{Y}}$

$$\hat{\mathbf{Y}} = \mathbf{T}\mathbf{Q}^T = \mathbf{T}(\mathbf{T}^T\mathbf{T})^{-1}\mathbf{T}^T\mathbf{Y}.$$

In PLS the components of $\mathbf{T}$ are denoted as the *scores*. The component of $\mathbf{P}$ are called the *X-loadings* while components of $\mathbf{Q}$ are called the *Y-loadings*. Above we shoved how the components involved in a PLS can be constructed from $\mathbf{W}$, but not how we could find $\mathbf{W}$. There are many approaches for this. For PLS2, a common approach is to solve the following optimization problem for $i \in [1, \ldots, k]$:

$$\mathbf{w}_i = \arg\max_{\mathbf{w}} \mathbf{w}^T \mathbf{X}^T \mathbf{Y} \mathbf{Y}^T \mathbf{w}$$

under the constraints

$$\mathbf{w}_i^T (\mathbf{I}_p - \mathbf{W}\mathbf{W}^+)\mathbf{w}_i = 1 \quad \text{and}$$
$$\mathbf{t}_i^T \mathbf{t}_j = \mathbf{w}_i^T \mathbf{X}^T \mathbf{X} \mathbf{w}_j \quad \text{for } j \in [1, \ldots, i-1].$$

Here, $\mathbf{I}_p$ is the $p \times p$ identity matrix and $\mathbf{W}^+$ is *pseudoinverse* of $\mathbf{W}$. It is necessary to specify the pseudoinverse because typically $p >> k$ and an algebraic inverse is not defined for such systems. A pseudoinverse can be found by performing a SVD. The matrix $\mathbf{W}$ is called the *loading weights*. Its elements will tell us how much each of the original variables in $\mathbf{X}$ contributes to the model.

PLS is a versatile method with many applications. By building a PLS model, one can predict a set of response variables when we have new observation. This is very useful if we have a set of variables that are difficult or costly to measure. If we can find a set of related variables that er easier or cheaper to measure, we can perform the costly measurements on a set of observations and bulid a model using PLS. In the future, we can use the model to predict the response. By inspecting the components involved in building the model, one can also find information on in which degree the different variables contributes to the model.

Chapter 3

Methods and tools

Here we will present the data sets used in this experiment and the software used for the analysis. We will present a strategy for obtaining a cell cycle pseudotime for the single cell data and model the gene expressions in the data sets. A method for determining significant cell cycle periodic genes will be outlined and we give a strategy for analysing the joined results from all the data sets used.

3.1 Data Origin

3.1.1 HaCat-MEF Cell Cultivation

Both HaCaT and MEF cell lines were obtained from the American Type Culture Collection (ATCC) and cultivated in a humidified incubator at 37°C in 5% CO2. HaCaT is a human keratinocyte cell line and was cultured in Dulbecco's modified Eagle's medium (DMEM) (Sigma-Aldrich, D6419) supplemented with 10% fetal bovine serum (FBS) (Sigma-Aldrich, F7524), 0.1mg/ml gentamicin (Gibco, 15710049), 2mM L-Glutamine (Sigma-Aldrich, G7513) and 1.25µg/ml fungizone (Sigma-Aldrich, A2942). MEF is a mouse embryonic cell line and was cultured in DMEM supplemented with 10% FBS, 1% Penicillin-Streptomycin (Sigma-Aldrich, P0781), 2 mM L-Glutamine and 1% MEM non-essential amino acid solution ((Sigma-Aldrich, M7145). The cells were harvested at

about 70% confluency and counted using the Moxi Z cell counter (ORFLO Technologies). We made a suspension of 900 000 HaCaT cells and 100 000 MEF cells and centrifuged at 150 relative centrifugal force (rcf) for 5 minutes at room temperature (RT). After washing the cells by carefully resuspending them in 1ml phosphate buffered saline (PBS) (Sigma-Aldrich, BR0014G) supplemented with 400 µg/ml Bovine Serum Albumin (BSA) (Sigma-Aldrich, A7030), the cells were counted and centrifuged again at 150 rcf for 5 minutes at RT. Then PBS-BSA solution was added to achieve a cell concentration of about 1200 cells per µL, followed by gentle resuspension of the cells before they were transferred through a 40 µM cell strainer. Cell suspension was kept on ice until proceeding with the 10x Genomics Single Cell Protocol.

3.1.2 Library Preparation and Sequencing for HaCat-MEF cells

Cells were processed using the 10x Genomics Chromium Controller and the Chromium Single Cell 3' Gene Expression Library & Gel Bead Kit (PN 1000006) following the standard manufacturer's protocols[1]. In brief, between 6000 and 8000 live cells were loaded onto the Chromium controller in an effort to recover between 3000 and 4000 cells for library preparation and sequencing. Gel beads were prepared according to standard manufacturer's protocols. Oil partitions of single-cell oligo coated gel beads (GEMs) were captured and reverse transcription was performed, resulting in cDNA tagged with a cell barcode and unique molecular identifier (UMI). Next, GEMs were broken and cDNA was amplified and quantified using an Agilent Bioanalyzer High Sensitivity chip (Agilent Technologies). To prepare the final libraries, amplified cDNA was enzymatically fragmented, end-repaired, and polyA tagged. Fragments were then size selected using SPRIselect magnetic beads (Omega Bio-tek). Next, Illumina sequencing adapters were ligated to the size-selected fragments and cleaned up using SPRIselect magnetic beads (Omega Bio-tek). Finally, sample indices were selected and amplified, followed by a double sided size selection using SPRIselect magnetic beads (Omega Bio-tek). Final library quality was assessed using an Agilent Bioanalyzer High Sensitivity chip. Samples were then sequenced on the Illumina NextSeq 500 with 26 cycles for Read 1, 8 cycles index read and 134 cycles for Read 2.

[1] http://tiny.cc/nv32bz

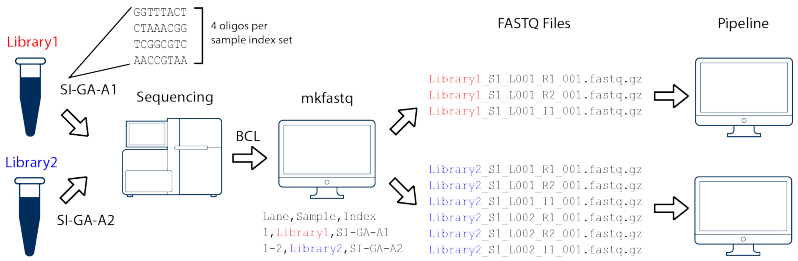


Figure 3.1: CellRanger mkfastq workflow.
Source: https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/using/mkfastq

3.1.3 Demultiplexing Sequencer Output for HaCat-MEF cells

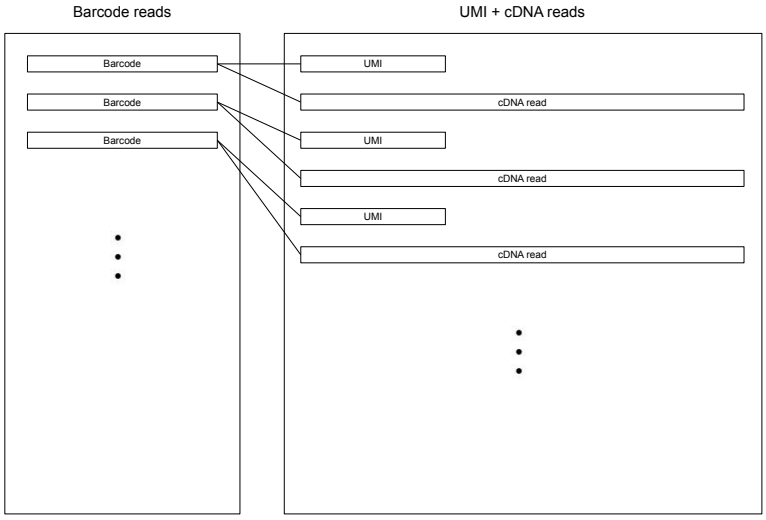
Demultiplexing was performed by using CellRanger 3.0.0 with the mkfastq workflow. The mkfastq workflow reads the Illumina sequencer output files and generate fastq files for each of the samples if there are multiple samples being processed. The fastq files will be organized as two files, one for read one (R1) and for read 2 (R2). The R1 files will contain the barcode reads and the UMI reads, while R2 consists of the corresponding cDNA reads. An illustration of the format can be seen in Figure 3.2b. Two technical replicates of the HaCat-MEF culture will be sequenced. Hence, our output fastq files will consist of two sets of R1 and R2 files. One named HaCat_MEF_A and one named HaCat_MEF_B.

3.1.4 Additional Data Sets

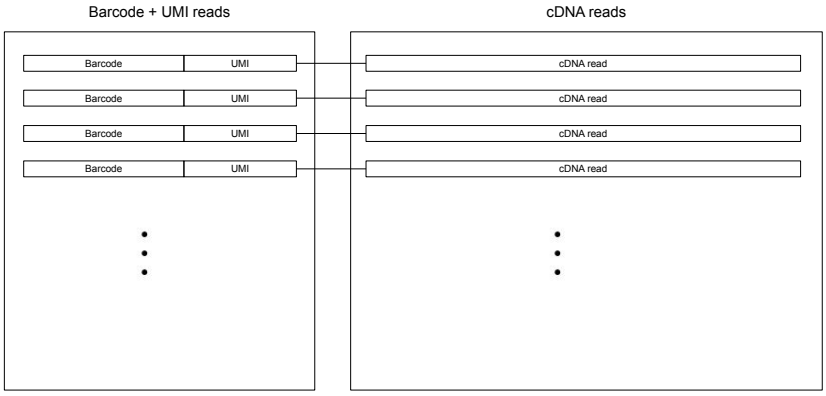
One data set consisting of 293t cells and a data set from Jurkat cells are downloaded from the 10X resources page[2][3]. Their origins are discussed in [Zhe+17].

Fastq files were downloaded from the resource page and reformatted using the python script provided in Appendix D.1. Reformatting was necessary because these data sets were demultiplexed using CellRanger 1.0.0, which is incompatible with software used for sequence alignment in this experiment. An illustration of the difference between the formats can be seen in Figure 3.2.

[2] https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.1.0/293t
[3] https://support.10xgenomics.com/single-cell-gene-expression/datasets/1.1.0/jurkat



(a) CellRanger 1.0.0



(b) CellRanger 3.0.0

Figure 3.2: Formatting of fastq files for CellRanger 1.0.0 and CellRanger 3.0.0.

3.2 Computational Systems

Our method will be developed on an Infrastructure as a Service (IaaS) node in an OpenStack[SAE12] environment provided at HUNT Cloud, NTNU[4]. The configuration is pre-

[4] https://www.ntnu.edu/mh/huntcloud

sented in Table 3.1.

Table 3.1: Configuration of IaaS node.

CPU	Intel Xeon E5-2680 v3
CPU clock	2.4 GHz
vCPUS	48
Memory	96GB
OS	Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-145-generic x86_64)

3.3 Software and Tools

The key software used in this experiment together with its version is listed in Table 3.2. Extensive descriptions of the conda environments used in this experiment can be found at Appendix E.2.

Table 3.2: Software and versions used in the experiments.

Software	Version
conda	4.6.7
snakemake	5.4.3
singularity	2.6.1-dist
python	3.6.8
scanpy	1.3.7
numpy	1.15.4
scipy	1.2.0
csaps	0.5.0
pandas	0.23.4
jupyterlab	0.34.5
R	3.5.1
gprofiler2	0.15
STAR	2.7.0e
CellRanger	3.0.0

3.4 Comparison of Sequence Alignment Methods

We will compare the performance of the CellRanger workflow with the recent addition of single cell support to STAR through the STARsolo algorithm. The comparison will be performed on the HaCat_MEF_A sample. We are interested in both execution times and

investigating if there are any differences in the resulting count matrices. Specifically, we want to investigate if there are any differences in the number of total reads or detected genes for the different types of genes, called biotypes, present in our data set. For both CellRanger and STARsolo, the results will be generated using the single cell pipeline developed at the Genomics Core Facility, NTNU. The pipeline is implemented in Snakemake. At this stage of the experiment, STARsolo has recently been released. As part of this experiment, we implement a Snakemake workflow and integrate it with the existing pipeline at GCF. See Appendix E.1 for a reference to the pipeline and the version used in this experiment. The pipeline takes the fastq files as input and produce count matrices using CellRanger or STARsolo. These count matrices are in turn used to generate a .h5ad AnnData object. The resulting data structure will be loaded into a scanpy environment for analysis.

3.4.1 CellRanger

For CellRanger, we will use the `cellranger count` workflow with a reference index built by following the recipe from the 10X resources web page[5], with the exception that we will use GRCh38 release 94 as our human reference genome. The build scripts can also be found in Appendix D.2. After we have built the reference index, we run CellRangers count workflow with the following:

```
cellranger count \
            --localcores 48 \
            --fastqs data/raw/fastq \
            --id HaCat_MEF_A \
            --sample HaCat_MEF_A \
            --transcriptome data/ext/ensembl/homo_sapiens__mus_musculus/GRCh38/cellranger \
            --expect-cells 3500 \
            --chemistry SC3Pv2 \
            --nopreflight \
            --disable-ui
```

This will generate the count matrix for the sample HaCat_MEF_A. The CellRanger count

[5] https://support.10xgenomics.com/single-cell-gene-expression/software/release-notes/build#hg19mm10_3.0.0

workflow gives two count matrices as output in two different folders, `outs/raw_feature_bc_matrix` and `outs/filtered_feature_bc_matrix`. Since STARsolo does not provide similar filtering when it generates the count matrix, we must use the `outs/raw_feature_bc_matrix` to give a realistic comparison. To convert this to a .h5ad AnnData object, we use the script provided in Chapter D.3. To generate AnnData object, we execute

```
python scripts/cellranger_scanpy.py \
            HaCat_MEF_A/outs/raw_feature_bc_matrix/matrix.mtx \
            -o HaCat_MEF_A/scanpy/adata.h5ad  \
            -f cellranger \
            --genome homo_sapiens__mus_musculus
```

The resulting AnnData object will be loaded into a scanpy environment. Cell barcodes that have zero reads will be filtered. Then, all genes with zero reads in all cell barcodes are filtered. We call the remaining genes our *detected genes*. To simplify our analysis, we will only use the cells originating from the human cells in our experiment. The variable name for each gene in our count matrix will be prefixed with `GRCh38_` if they are mapped to the human genome or `GRCm38_` if they are mapped to mouse. For each cell, we simply count the number of reads mapped to human genes and the number of reads mapped to mouse genes. From this, we calculate the percent mapped to mouse for every cell in our count matrix. We set a threshold of 15% of reads mapped to mouse. All cell barcodes above this threshold are classified as mouse cells and excluded from our analysis. These barcodes are written to a file and will in turn be used to filter out barcodes that originates from the mouse cells in the count matrices generated by STARsolo.

3.4.2 STARsolo

For STAR, we build the reference index by running `STAR --runMode genomeGenerate`. We will use the same genome as we have used for CellRanger above. The index is built with the following

```
STAR --runThreadN 48 \
    --runMode genomeGenerate \
    --genomeDir data/ext/ensembl/homo_sapiens/GRCh38/star \
```

```
    --genomeFastaFiles data/ext/ensembl/homo_sapiens/GRCh38/fasta/genome.fa \
    --sjdbGTFfile data/ext/ensembl/homo_sapiens/GRCh38/genes/genes.gtf \
    --sjdbOverhang 98
```

To generate the count matrix, we use the recently added `STAR --soloType Droplet` algorithm, which enables a drop-in replacement for 10X CellRanger gene quantification output. For a STARsolo run, we must specify the single cell barcode length and the UMI length used in the library preparation kit. We must also specify where in the read STARsolo should start reading the UMI sequence. In addition, we must provide a *barcode whitelist*. The whitelist should contain all valid barcodes specific to chemistry used in the library preparation of the samples. The whitelists for the different 10X chemistry sets are bundled within the CellRanger software and can be fetched directly from its folder structure. A table listing which of the official 10X whitelists that should be used together with the different 10X chemistries can be found in Table C.1 in Appendix C.1. The barcodes read by STARsolo during the count algorithm are validated against the specified whitelist. Barcodes with one mismatch against the whitelist will be corrected and used in the count matrix. Barcodes with more than one mismatch will be discarded. We generate the count matrices by the following

```
STAR --soloType Droplet \
    --readFilesCommand zcat \
    --soloCBwhitelist data/ext/10xgenomics/homo_sapiens/GRCh38/737K-august-2016.txt \
    --readFilesIn
        data/raw/fastq/HaCat_MEF_A_S1_L001_R2_001.fastq.gz,\
        data/raw/fastq/HaCat_MEF_A_S1_L002_R2_001.fastq.gz,\
        data/raw/fastq/HaCat_MEF_A_S1_L003_R2_001.fastq.gz,\
        data/raw/fastq/HaCat_MEF_A_S1_L004_R2_001.fastq.gz\
         \
        data/raw/fastq/HaCat_MEF_A_S1_L001_R1_001.fastq.gz,\
        data/raw/fastq/HaCat_MEF_A_S1_L002_R1_001.fastq.gz,\
        data/raw/fastq/HaCat_MEF_A_S1_L003_R1_001.fastq.gz,\
        data/raw/fastq/HaCat_MEF_A_S1_L004_R1_001.fastq.gz \
    --genomeDir data/ext/ensembl/homo_sapiens/GRCh38/star \
    --outFileNamePrefix data/tmp/singlecell/quant/star/HaCat_MEF_A/ \
    --soloCBlen 16 \
    --soloUMIlen 10 \
```

Table 3.3: The different parameters used to generate count matrices from STARsolo and the resulting name of the respective datasets.

Name	Extra Parameters
star_default	No extra parameters
star_tweaked	–outFilterMismatchNoverLmax 1.0 –outFilterMismatchNoverReadLmax 0.05
star_mm_tweaked	–outFilterMismatchNoverLmax 1.0 –outFilterMismatchNoverReadLmax 0.05 –outFilterMultimapNmax 1

```
--soloUMIstart 17 \
--runThreadN 48 \
--genomeLoad LoadAndKeep
```

In addition, we will perform two extra alignments using STARsolo, with the additional parameters listed in Table 3.3. The parameter `--outFilterMismatchNoverLmax` determines a cutoff for the ratio of the allowed number of mismatches compared to the length of mapped reads. Setting this value to 1.0 is equivalent to disabling the check. Instead, we use the parameter `--outFilterMismatchNoverReadLmax` to determine the ratio by comparing the number of mismatches to length of the read itself. For a read length of 100, a value of 0.05 will correspond to allowing a mismatch of less than or equal to 5 nucleotides to be counted as a hit for a particular location in the genome. The last parameter, `--outFilterMultimapNmax` determines the maximum allowed number of mapped locations in the genome. If the number of mapped locations is above this threshold, the read will be categorized as "mapped to too many loci" in output logs of STAR and it will not be included in the count matrix. The default parameters for STAR if no thresholds are manually specified are `--outFilterMultimapNmax 10`, `--outFilterMismatchNoverLmax 0.3` and `--outFilterMismatchNoverReadLmax 1.0` (equivalent to disabled).

Each of the resulting count matrices will be converted to .h5ad AnnData objects, by the same script as for CellRanger, to be analyzed in a scanpy environment.

3.4.3 Benchmarking Alignment Methods

The metrics are obtained by using Snakemakes benchmark features. In Snakemake, we can specify an output file for benchmark metrics with the benchmark tag. Snakemake will then use functionality provided by the python package psutil[6] to track and report the metrics in a .tsv format. We will report the accumulated metrics obtained by running both the HaCat_MEF_A and HaCat_MEF_B sample through each of the workflows.

3.5 Pre-processing of Single Cell data

Here, will explain the steps involved in pre-processing single cell data and investigating basic Quality Control (QC) metrics for our experiment.

The count matrices generated by STARsolo must be aggregated into one data set containing all the data from both samples. We do this by executing the script listed in Appendix D.3 with the following parameters:

```
python src/single-cell/rules/quant/scripts/cellranger_scanpy.py \
    data/tmp/singlecell/quant/star/HaCat_MEF_A/Solo.out/matrix.mtx \
    data/tmp/singlecell/quant/star/HaCat_MEF_B/Solo.out/matrix.mtx \
    -o data/tmp/singlecell/quant/aggregate/star/scanpy/scanpy_aggr.h5ad \
    -f star \
    --normalize mapped
```

The samples are normalized by down sampling reads such that both data sets have the same number of total reads. The aggregated AnnData object is loaded into a scanpy environment. The cells that belong to mouse are filtered out by using the list of barcodes assigned to mouse in Chapter 3.4.1. We apply a basic pre-filter to remove low quality cells and genes that are represented in too few cells. We use a threshold of 500 detected genes per cell. For genes, each gene must be represented in at least 5 cells to pass the filter. These filters are only applied as an initial step to reduce the set of our viable cell candidates drastically.

[6] https://psutil.readthedocs.io/en/latest/

The following three parameters can usually be calculated from all single cell data sets. The number of detected genes per cell, the number of reads per cell and the fraction of reads belonging to mitochondrial RNA in each cell. Previous studies show that a high fraction of mitochondrial reads are related to low quality of the cell[GKK12; Ili+16] as it may indicate cell death or membrane damage. The QC parameters for number of detected genes and total reads per cell barcode can be related to either cell size or technical factors from library preparation or sequecng. A common procedure for single cell QC is to leave out the cells that show unusual values in these metrics in further analysis[GSM18]. Here, unusual values must be interpreted isolated for each data set. These metrics can vary greatly between experiments and it is hard to set universal thresholds that will be valid for all single cell data sets.

When we filter the data further, we must consider all these three metrics and find a balance between the thresholds. The metrics may be related to each other and should therefore not be considered in isolation. The thresholds should also be as permissive as possible to avoid filtering viable cells[LT19] and the definition of a viable cells will be specific for different types of analysis. The probability density functions of the QC parameters will be estimated by using *kernel density estimates* viewed in *violin plots*. From this, we set thresholds to remove any *bimodal distributions*. A distribution is said to be bimodal if multiple peaks can be observed in its probability density function.

As a part of our method for pseudotime modelling, we will use a method of unsupervised dimensional reduction technique called Principal Component Analyis (PCA). The centered data set will be projected into the two first principal components of the PCA. We want the variability explained by the two first components to be highly related to the biology of the data set as opposed to nuisance factors, such the total number of reads. As a quality control, we will perform a PCA analysis and investigate if additional pre-processing steps are necessary.

To reduce the impact of noisy, low variable genes in the PCA of our data set, we will identify the genes that are most variable in our data set and use only these as input in the dimensional reduction. Here, we use the scanpy method `scanpy.pp.highly_variable_genes()`, which is a python re-implementation of the method described in [Sat+15]. Also here, we want to be as permissive as possible. Hence, we use very low thresholds for the minimum

mean and minimum dispersions (`min_mean=0.0001` and `min_disp=0.0001`, respectively). The method expects log-transformed, normalized data. The data is first normalized to Reads Per Million (RPM) with the scanpy method `scanpy.tl.normalize_per_cell()` with `reads_per_cell_after=1e6`. The data is then log-transformed by `scanpy.pp.log1p()`. This normalization uses $\log_{10}(\text{RPM} + 1)$ and by this, it preserves the sparsity of the data matrix.

The PCA of the data set is calculated by `scanpy.tl.pca()`. This scanpy method is an interface to the Scikit-learn PCA implementation[Ped+11]. We investigate our QC metrics in the two first components of the PCA and remove any visible factors using scanpys `scanpy.tl.regress_out()`. After these steps, our data set will be ready for further analysis.

3.6 Deriving a Pseudotime Model

We wish to derive a method for modelling the gene expression levels in our data set as a function of cyclic time. We will identify cells that have a high expression of marker genes that are previously shown to be strongly related to the cell cycle. These cells will be scored according to their relative expression of marker genes within each of the cell cycle phases G1, S and G2/M. According to this score, they will be classified to one of these three phases. A threshold for this score is calculated within each of the phases and cells that falls below the threshold are discarded. We perform a PCA analysis with two components of the data set with the set of marker genes as input. For each cell, we will compute its angle in this plane, relative to the PC1 axis. This angle will be used as an ordering of the cells within each phase before the cells are concatenated to represent the order of the cell cycle.

3.6.1 Identification of Proliferating Cells

We will assign each cell in the data set to one of the three phases, G1, S or G2/M. We use a modified version of the scanpy method `scanpy.tl.score_genes_cell_cycle()`

together with a set of marker genes that are highly correlated to each of the three phases. The method will give each cell a score for all three phases. The score will be based on a comparison of expression levels of the marker genes of the respective phases compared to a robust mean expression of a random selection of other genes in the dataset. A cell is assigned a phase according to its highest score. The original scanpy method only takes a set of genes for the S phase and G2/M phase as input. In the original method, a cell is classified to G1 if both the score for the S phase and the G2/M phase are negative. This approach is unsatisfactory for our goals since it is not able to separate G0 from G1. We want to classify a cell to G1 only if we believe that the cell is under active proliferation. The modified version of the scanpy method is presented in Appendix D.4.

As our marker genes we use the genes defined in [Whi+02] and [Tir+16] and compile them to one list presented in Table C.2 in Appendix C.2. The full list consists of 128 marker genes. We select the genes associated with G1 and M/G1 to be our G1 marker genes, the genes associated with S to be our S marker genes and the genes associated with G2 and G2/M to be our G2/M marker genes. Only the cells that have a sufficient score will be kept for further analysis. As a threshold for the score we use 60% of the mean of the top 10 scores within each phase. The rest of the cells are filtered out.

3.6.2 Pseudotime Ordering of Cells

For each cell, we calculate its angle with the PC1 axis in the PC1,PC2 plane of the filtered data set. The angle is found by calculating the arctangent with the PC1 component as the adjacent and the PC2 component as the opposite. This angle will then be used as an estimate of the cells location in the cell cycle. The cells are sorted by their corresponding angles and given an order that progress from zero to the length of the data set.

The direction and the starting point of the ordering from the angles in the PC1,PC2 plane is arbitrary and may not represent the progression of the cell cycle as we wish to model it. In our model, we want the G1 phase to start at exactly time zero. We must find the boundaries between the phases in our pseudotime ordering and shift it so that the phase boundary between G2/M and G1 is located at exactly zero. We also want the phases to progress from G1 to S and then to G2/M along the ordering. The ordering

is divided into bins of 20 cells and for each bin we calculate the fraction of cells in each of the phases. Linear interpolation is used on each of these fractions to generate linear curves that represents the fraction of cells in each phase as a function of the cell order. We take the phase boundaries to be where these lines intersect. That is, the boundary of G2/M and G1 is where the fraction of cells classified as G1 equals the fraction of cells classified as G2/M.

As a quality control of our ordering, we calculate the mean expression of a selection marker genes within each phase and compare them. We expect that cells in the region between the boundaries G2/M-G1 and G1-S will have a higher mean expression of the G1 marker genes compared to the rest of the cells in the ordering. Cells between G1-S and S-G2/M should have a higher expression of the S marker genes and cells between S-G2/M and G2/M-G1 should have a higher expression of G2 and G2/M marker genes. We now use the full set of genes with RPM normalized values for the cells in our pseudotime ordering. The expressions are calculated for the set of marker genes given in [Whi+02].

In the human cell cycle, the cells divide approximately every 24 hours[GM00]. In these 24 hours, approximately 11 hours is spent in G1, 8 hours in S, 4 hours in G2 and 1 hour in M. In our model, we will assign each cell a time in the interval $[0, 24]$. This is achieved by dividing the ordering of the cells by the previously found phase boundaries in the pseudotime order. The cells have ordering between the boundaries for G2/M-G1 and G1-S are spread out uniformly in the half-open interval $[0, 11)$, the cells between G1-S and S-G2/M will be in $[11, 19)$ and the cells between S-G2/M and G2/M-G1 are given values in $[19, 24)$. The pseudotime order has now been transformed to a pseudotime time series, from now on referred to only as the pseudotime.

3.7 Gene Expression Modelling using Smoothing Cubic Splines

We want to model the gene expression levels along the pseudotime as periodically, continuous and sufficiently smooth functions. To achieve this, smoothing cubic splines will be used. The spline will fit a curve to a set of data points, called the *control points*. The

resulting function will be continuous and twice differentiable everywhere.

3.7.1 Control Points

To make the cubic spline periodic, we must make the control points periodic. This is achieved by dividing the cells in the pseudotime into groups based on uniform time intervals. For a gene expression, we calculate the mean expression within each of the groups. These mean values are given time points corresponding to mid-point in each group. These control points will lie in the open interval $(0, 24)$. We close this interval by adding the mean of the first and the last interval to the time points 0 and 24. We now have periodic control points in the closed interval $[0, 24]$. In addition, we want our resulting splines to have periodic derivatives at the boundaries. That is, if we let $\hat{f}(t)$ be our cubic smoothing spline, we want that $\hat{f}(0) = \hat{f}(24)$ and $\hat{f}'(0) = \hat{f}'(24)$. To achieve this, we add the interior of our control points once to the left by subtracting 24 from their time values and once to the right by adding 24. Figure 3.3 shows how this is accomplished for a sample gene.

3.7.2 Optimal Smoothing Parameter

When we generate the smoothing splines for a set of data points, we must decide upon a *smoothing parameter*, $p \in [0, 1]$. This smoothing parameter gives the weighting of the constraints given in Equation 2.18. A smoothing parameter of 0 will give the least squares line through the control points and a smoothing parameter of 1 will yield an interpolating spline. This is illustrated in Figure 3.4.

We want our spline model to be as smooth as possible without sacrificing the representation of the data. To find an optimal smoothing parameter for our data set, we calculate the residuals and the curvature at each control point on a set of marker genes for a range of smoothing parameter values. Let T be the set of time points of our control points and let Y_g be the set of the mean gene expression related to each of the time points by the description above for a marker gene $g \in G$. For a smoothing parameter value, p, let $\hat{f}_{pg}$ denote the resulting cubic smoothing spline for the gene g. We calculate the following for

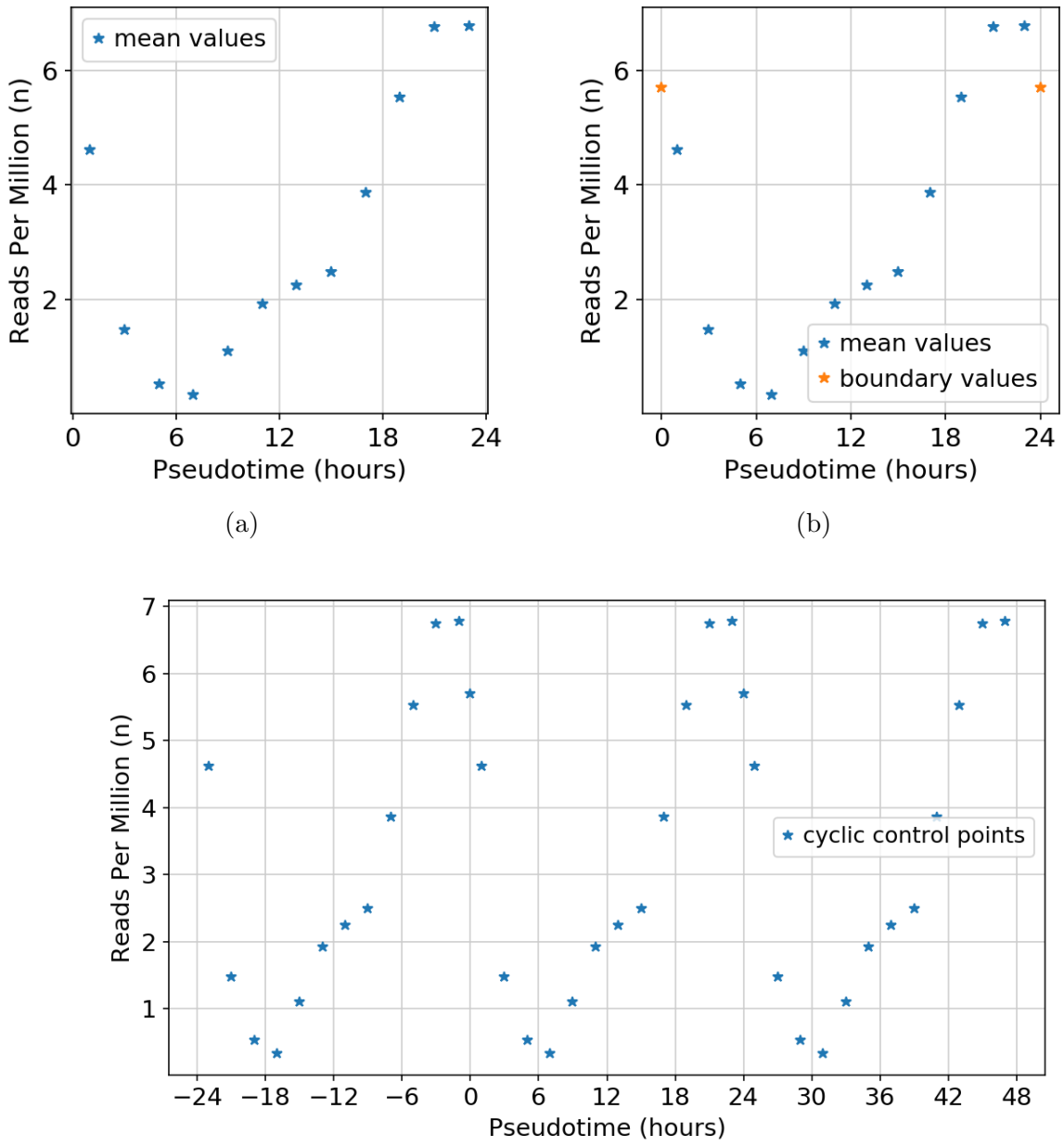


Figure 3.3: **Periodic control points**. 3.3a) Control points generated by centered mean values in fixed time intervals of 2 hours along the pseudotime. 3.3b) Control points with the added boundary values at 0 and 24. 3.3c) Periodic control points by adding the interior of the control points once to the left and once to the right.

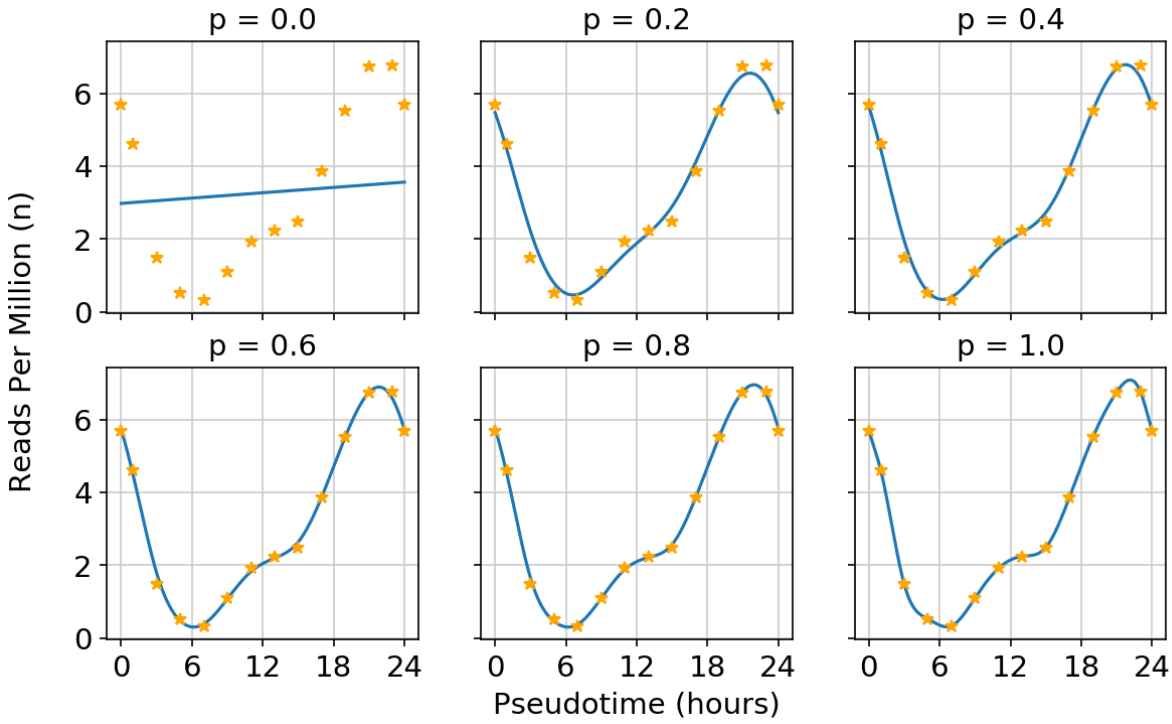


Figure 3.4: **Smoothing parameter**. The smoothing cubic splines for an example gene expression as a function of the smoothing parameter p. A smoothing parameter of $p = 0$ gives the least square line through the control points and $p = 1$ yields the interpolating cubic spline.

each marker gene, g.

$$r_g(p) = ||\hat{f}_{pg}(T) - Y_g||_2,$$

and for $h = 24/N$ with $N = 10^4$

$$c_g(p) = \sum_j^N \frac{|\hat{f}''_{pg}(jh)|}{(1 + \hat{f}'_{pg}(jh)^2)^{\frac{3}{2}}}.$$

From this, we construct the mean values over all genes for each of the two functions such that for each p, we have

$$\overline{r}(p) = \frac{\sum_{g \in G} r_g(p)}{N_g},$$
$$\overline{c}(p) = \frac{\sum_{g \in G} c_g(p)}{N_g},$$

where N_g is the length of G. The result is two functions, $\overline{r}(p)$, the mean 2-norm of the residuals as a function of the smoothing parameter p and $\overline{c}(p)$, the mean curvature at the location of each control point as a function of p. Each of these functions are scaled so

that $\overline{r} : [0,1] \to [0,1]$ and $\overline{c} : [0,1] \to [0,1]$. We construct an objective function $g(p)$ and choose p_{opt} such that

$$g(p) = \overline{r}(p) + \overline{c}(p), \quad \text{and}$$
$$p_{\text{opt}} = \arg\min_{p} g(p).$$

In this objective function, we expect $\overline{r}$ to be strictly decreasing and $\overline{c}$ to be strictly increasing. If the control points follow a close to smooth trajectory, $\overline{r}$ will decrease rapidly and $\overline{c}$ will grow first rapidly, then slowly as functions of p. If our pseudotime of the cells is able to capture cyclic expressions, we expect this objective function to yield lower values of p.

As the genes used to generate values for the objective function, we use the full set of marker genes given in Appendix C.2. We only need our spline model to be a good fit for expression patterns of genes that show cyclic expression patterns. Hence, it is unnecessary to use a large set of genes that may not be descriptive to our ends. We use the code in Appendix D.6 to find the optimal smoothing parameter. To generate our cubic smoothing spline models, we use the code provided in Appendix D.5. The code uses the python library csaps[7] to generate the cubic smoothing splines.

3.8 Identification of Cell Cycle Periodic Genes

Here we will define a strategy used to identify cyclically expressed genes in our data. We will analyze the spline model of the full set of detected genes in our data set. We construct a time series for all genes by subsampling the splines generated for each of the genes. The expression data generated from the splines will be regressed onto a sine and a cosine curve with period equal to the estimated human cell cycle of 24 hours using Partial Least Squares regression (PLS). Each gene will be projected into the plane consisting of the loading weights to each of the PLS components. Both the significance level and the phase in the cell cycle will be determined in this plane.

[7] https://github.com/espdev/csaps

3.8.1 Partial Least Squares Regression

We will use the spline model generated by the above as a smoother for our gene expression data. We subsample the smoothing spline expression models on fixed time intervals. Each gene is subsampled on 25 equidistant time points in the range 0 to 24. This data matrix will be regressed onto two response variables consisting of a sine and a cosine curve of period 24 hours evaluated on the same time points as for the gene expression obersvations.

For our PLS regression, we will use the implementation miirPLS given in Appendix D.7. This PLS imlpementation was developed for the synchronization experiment described in [Pen+13].

3.8.2 Significance Threshold

To determine a significance threshold for the result of the PLS regression, we use the same approach as in [JLB03]. We will give a brief explanation of the concept. A detailed description is given in the Method of the referred study.

The distance from the origin to each genes location in the plane of the scaled loading weights, for simplicity called the loading plane, will be a measure of how strongly it is coupled to the cell cycle. To determine a significance cut-off value for the genes distance in the plane, the expression patterns will be regressed on to randomly permuted response variables. This permutation is performed 1000 times. For each permutation, the length of each gene in the loading plane is calculated and sorted. We then choose a cut off value for the length corresponding to a significance level p, so that $(1-p)100\%$ of the genes have a length in the loading plane below this value. In other words, there is an estimate $p100\%$ probability that a gene has a length above this value by chance. To get a more robust estimate for the cut-off value, we use the mean cut-off value over all 1000 resampling runs. In our experiment, we use $p = 0.05$ to determine the cut-off values.

3.8.3 Phase Assignment

We expect genes with a significant magnitude along both axes to show cyclic behavior with period equal to the cell cycle, but with a phase off-set to that is explained by some linear combination of the two response variables. Therefore, the angle in the loading plane will describe a genes phase in the cell cycle. Each gene is assigned an angle by calculating the arctangent in the loading plane with its first scaled loading weight as the adjacent and the second scaled loading weight as its opposite.

The method of phase assignment follows the procedure outlined in [Pen+13]. To classify the cell cycle phase for the genes in our data set that show significant cell cycle periodic behavior, we will use marker genes. Again, we use the list compiled in C.2. For each phase in this list, we take the genes that have significant cyclic behavior in our data set and calculate their phase angle probability density function by finding the mean angle and the variance and use this in a normal distribution. Some marker genes may show expression patterns that are not consistent with their known phase. These marker genes will contribute to a larger variance in its corresponding distributions. To account for this, we will use random resampling with leave-10%-out when we calculate the mean and the variance for each of the distributions. We perform 500 resamples and sort based on the (mean,variance) pairs based on the variance values. We then select the median variance with its corresponding mean phase angle. These values will be used for phase classification for the rest of the genes. Each gene is classified to a phase by assigning the most probable phase by the probability density functions.

If the cell cycle phase classification in the loading plane shows out-of-order assignments, we increase the variance of the distribution with the lowest variance incrementally by 25%. Then, phases are re-assigned and the process is repeated until there are no out-of-order assignments. To find the phase angle distributions and to assign the cell cycle phases in the loading plane, we use the code listed in Appendix D.8.

3.9 Functional Enrichment of Genes

To evaluate the results of our method, we perform a functional enrichment of the significant cell cycle periodic genes against Gene Ontology Resource[Ash+00; Con18] (GO). GO is a comprehensive resource for annotation of genes and gene products with their properties. The ontology is divided into three groups, Molecular Function (MF), Cellular Component (CC) and Biological Process (BP). Each of these groups consists of *terms* that are related to each other in a directed acyclic graph structure. Children terms usually define more specific properties than their parent and a term may have more than one parent. A gene may also be related to more than one term. When we perform functional enrichment on a set of genes, we retrieve the terms that are over-represented within a significance threshold. The retrieved terms will describe the properties of our input gene list.

We perform the enrichment on the genes in each of the assigned phases using the R package gProfiler2[Rau+19] against GO:BP with the g:SCS algorithm described in [Rei+07] for significance threshold. We use a significance threshold of 0.05 in our queries. This is equivalent to a false discovery rate of 5%

We will also filter the retrieved terms by their term size. If a term has a small term size, even an overlap of one gene will yield significant result and term itself may be too specific for our purpose. If a term size is very large, its definition may be too broad for any meaningful interpretation. We use 10 as a lower threshold and 2500 for the upper. The code used to retrieve the results from the GO enrichment is provided in Appendix D.9.

3.10 Evaluation of Aggregate Results

We find the significant cell cycle periodic genes that are present in all three data sets. This set will be further divided into known cell cycle periodic genes (our marker genes) and a set of new candidate genes. These sets will be further divided into two cell cycle phases. We must take into account that some genes may have been assigned different cell cycle

phases in our data sets. To tackle this, we will use a different approach to determine their phases. The idea is that if a gene is present in all three data sets and its gene expression model is sufficiently similar between them, we can use the mean peaktime from the three expression models to implicitly assign a phase.

Here, sufficiently similar is quite an open definition. We will calculate the correlation matrix between the three expression models by using their spline model in each of the data sets. We subsample the three spline models and calculate their correlation matrix given in Chapter **??**. As a measure of similarity, we will use the mean of the upper triangular part of the correlation matrix. The upper triangular correlation matrix will be the sum of all pairwise correlations. This mean correlation will be inspected and we will apply a suitable threshold.

Last, we will perform functional enrichment on the candidates in each of the estimated phases.

Chapter 4

Results

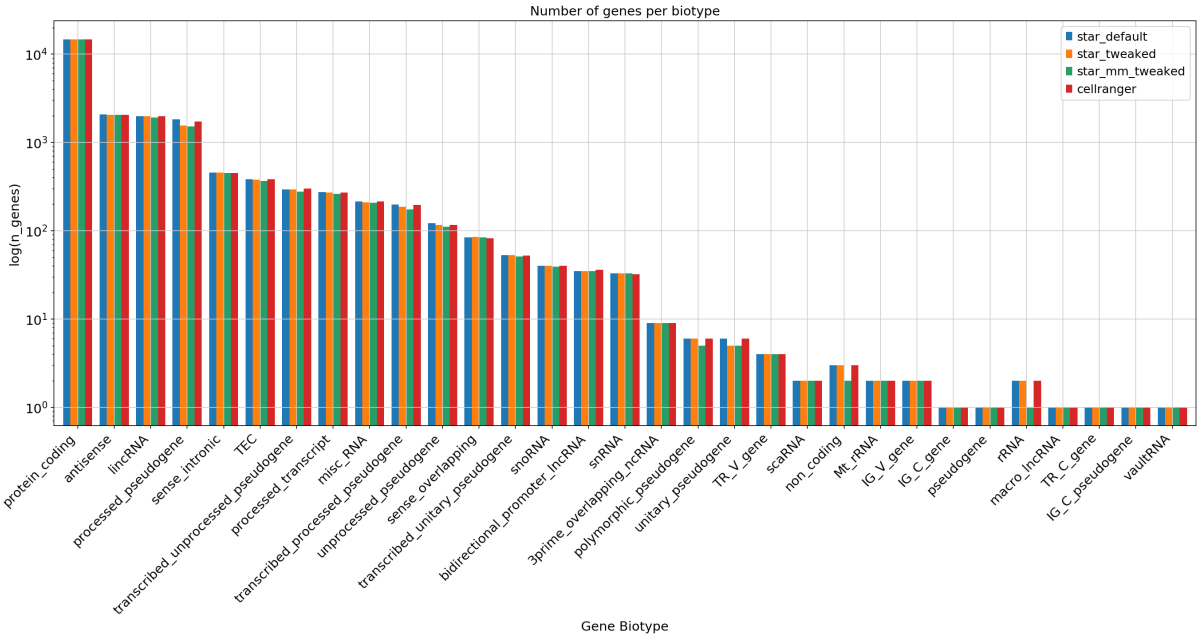
4.1 Sequence Alignment

Here, we will present the comparison of the methods for sequence alignments described in Chapter 3.4. The comparison is performed by analyzing the A sample of the HaCat-MEF cell culture. Cell barcodes that are classified originating from mouse cells are excluded. Besides this, the count matrices are only filtered for cell barcodes and genes that have zero reads.

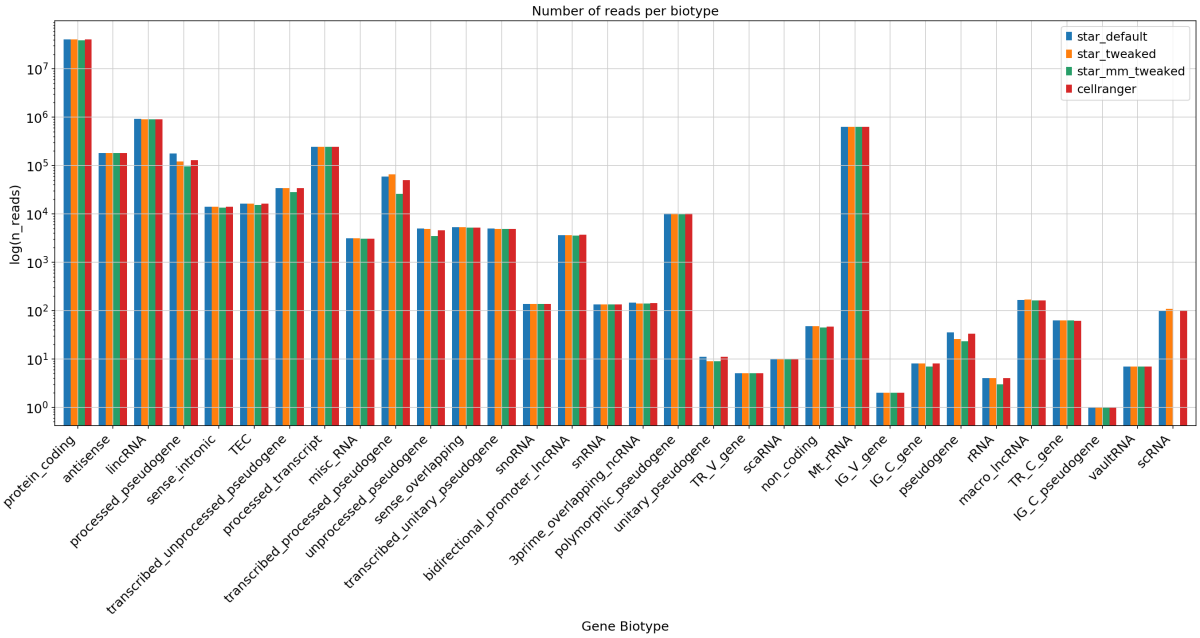
4.1.1 Comparison of Alignment Methods

From Figure 4.1a, we see that CellRanger and STARsolo yields almost identical results in the number of genes per biotype. This is not surprising as CellRangers count workflow uses and underlying STAR aligner with some extra processing of the data. In additon, the same reference genome and the same gene annotation file is used on all methods. The effects of the additional parameters for STARsolo are also evident from Table 4.1. As we become more stringent with the number of mismatches allowed, we can see a decrease in the number of genes mapped to the pseudogene biotypes while the results for protein encoding does not change more than -0.78%. Pseudogenes are defined as sequences that are similar to functioning proteins but does not encode for protein because of present deficiancies

or mutations in their sequences. Hence, it is expected that a more strict threshold for multimappers are likely to impact the pseudogenes. This becomes more evident if we look at the relative change of reads per biotype between star_default and star_mm_tweaked. From Table 4.2, we can clearly see that the reads mapped to pseudogenes decreases drastically as we become more stringent with the mismatch and multimappers policy. Here, number of reads mapped to protein encoding only changes by -4.52% The number of total reads per alignment method are presented in Table 4.3. Except from the difference of $\sim 16\dot{1}0^6$ total reads between star_mm_tweaked and CellRanger, the differences are very subtile between the methods. The largest impact comes from introducing the stricter multimapper policy for STARsolo.



(a) Number of genes within each biotype compared by alignment method.



(b) Number of reads within each biotype compared by alignment method.

Figure 4.1: Comparison of alignment using CellRanger and STARsolo.

Table 4.1: Top 8 relative change in detected genes per biotype from method star_default to star_mm_twaked, ranked by absolute relative change.

Biotype	Relative Change[%]
processed_pseudogene	-17.57
transcribed_processed_pseudogene	-11.68
unprocessed_pseudogene	-9.02
transcribed_unprocessed_pseudogene	-6.12
processed_transcript	-4.40
TEC	-4.19
transcribed_unitary_pseudogene	-3.77
lincRNA	-3.56
(12th) protein_coding	-0.78

Table 4.2: Top 8 relative change in total reads per biotype from method star_default to star_mm_twaked, ranked by absolute relative change.

Biotype	Relative Change[%]
transcribed_processed_pseudogene	-56.47
processed_pseudogene	-45.99
pseudogene	-34.29
unprocessed_pseudogene	-30.95
unitary_pseudogene	-18.18
transcribed_unprocessed_pseudogene	-16.61
TEC	-6.75
non_coding	-6.25
(12th) protein_coding	-4.52

Table 4.3: Total number of reads per alignment method.

Alignment method	Total reads [n]
star_default	43359354.0
star_tweaked	43079081.0
star_mm_tweaked	41351617.0
cellranger	42962068.0

4.1.2 Benchmarking

The benchmark metrics from STARsolo and CellRanger are presented in Table 4.4. The experiments are run on the system described in Chapter 3.2 and the results are obtained by employing the benchmark tag in the snakemake rules defining the count workflows.

We see that measured runtimes for STARsolo outperforms the CellRanger default count workflow by a factor of 11.56. CellRanger count has 1.93 times more I/O in compared to STARsolo and CellRanger count writes 8.00 times more data compared to STARsolo.

Because CellRanger by default performs secondary analysis on the count data, we suspected this might be the source of the large I/O out observed. Therefore CellRanger was run one more time with the parameter –nosecondary. This disables the secondary analysis completely. STARsolo also has the advantage the advantage that it can load the reference genome index into shared memory. In this case, STARsolo only can load the reference genome into RAM almost instantly when it aligns the second sample. We also performed benchmarks for STARsolo with this feature disabled.

STARsolo without shared memory results in a 13% increase in measured runtime. This makes out 237 seconds. By investigating the output logs of STARsolo, we can see that this matches the reported time for loading the genome. The increase in 29 GB for I/O in also matches the size of the reference genome index.

CellRanger count with no secondary analysis offers a speedup in runtime of 20.7% compared to default parameters. Otherwise, the only significant difference is the I/O in with a difference of 10 GB less and I/O out with 0.45 GB less. The secondary analysis uses information from the alignment files to derive statistics, and the difference in I/O in can be attributed to omitting this step. We also that the I/O out has not changed significantly. This implies the large I/O out observed in CellRanger must attributed to intermediate operations it performs during the count workflow.

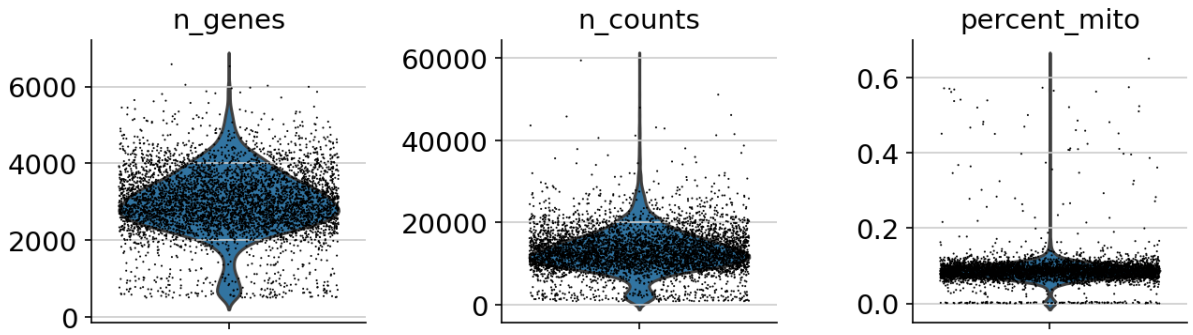
Table 4.4: Benchmark metrics comparing STARsolo to cellranger count. The reported metrics for runtime and I/O are accumulated metrics across the two samples, HaCat_MEF_A and HaCat_MEF_B. The metric for max memory usage reported is the peak Resident Set Size across the two samples.

Workflow	Runtime[s]	Max memory[GB]	I/O in[GB]	I/O out[GB]	Output size[GB]
STARsolo	1793.51	37.61	49.65	54.78	57.42
STARsolo no shmem	2030.02	37.79	76.97	54.21	57.41
CellRanger nosecondary	17176.03	54.21	85.99	437.32	20.87
CellRanger default	20733.09	53.28	96.04	438.32	21.32

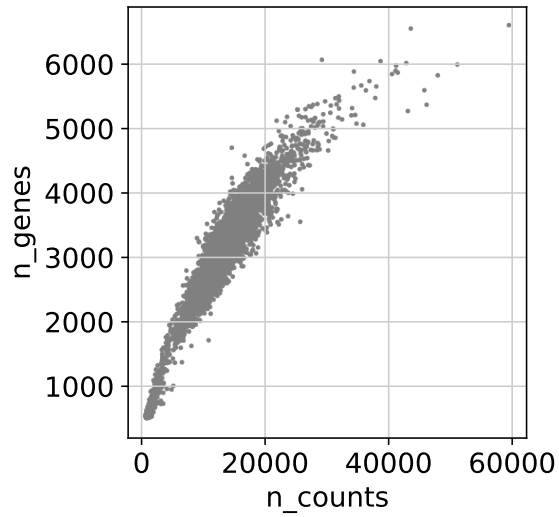
4.2 Pre-Processing and Quality Control

From here on, the aggregate HaCat_MEF aligned with the method star_mm_tweaked will be used. After intial filtering, we are left with 5056 cells and 18331 genes in our data set. From Figure 4.2a, we observe bi-modal distributions for both the number of detected genes per cell and the number of reads per cell. From Figure 4.2b, we see that a threshold of 8000 reads will remove bi-modality in both number of genes and reads. This will also remove most of the outliers in fraction of mitochondrial reads in Figure 4.2c. In Figure 4.3 the data set has been filtered a threshold of 8000 reads and a fraction of 0.2 for mitochondrial reads. No hard filter for number of detected genes were applied.

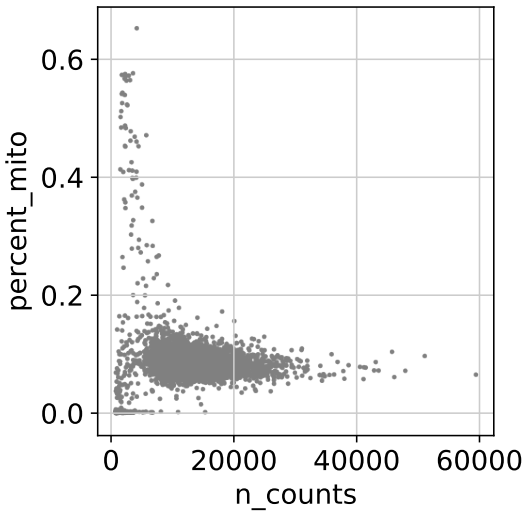
The data is RPM normalized and log(RPM)+1 transformed. We find 6538 genes classified as highly variable in our data set. These are used as input for the PCA analysis in Figure 4.4. We observe that there are no apparent patterns related to the fraction of mitochondrial reads or to the batch id. For PC1, we can see that both the number of detected genes and the number of reads per cell are positively correlated to PC1. We regress out the number of total reads per cell and perform a new PCA projection seen in 4.5. As expected the variation explained by PC1 no longer correlates to the number of reads.



(a)

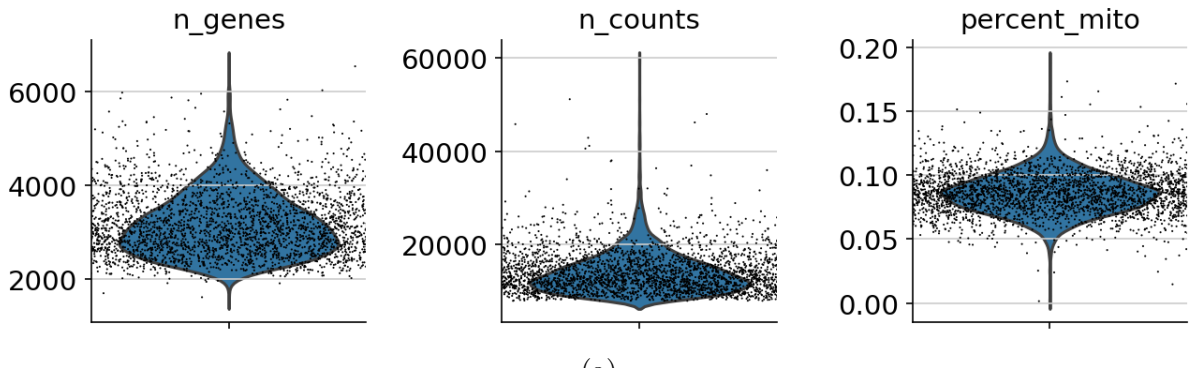


(b)

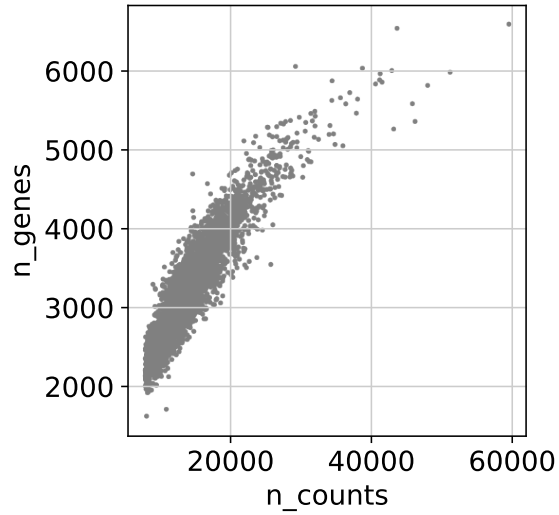


(c)

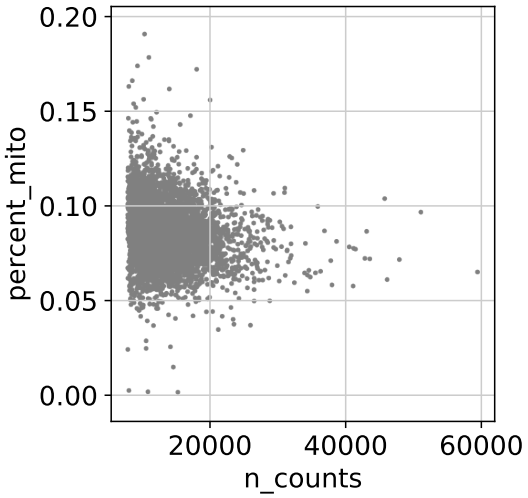
Figure 4.2: 4.2a) Kernel density estimates for number of genes detected per cell (n_genes), number of reads per cell (n_counts) and percent mitochondrial reads per cell (percent_mito). 4.2b) Number of genes detected versus number of reads per cell. 4.2c) Percent mitochondrial reads versus number of reads per cell.



(a)



(b)



(c)

Figure 4.3: 4.2a) Kernel density estimates after filtering for number of genes detected per cell (n_genes), number of reads per cell (n_counts) and percent mitochondrial reads per cell (percent_mito). 4.2b) Number of genes detected versus number of reads per cell after filtering. 4.2c) Percent mitochondrial reads versus number of reads per cell after filtering.

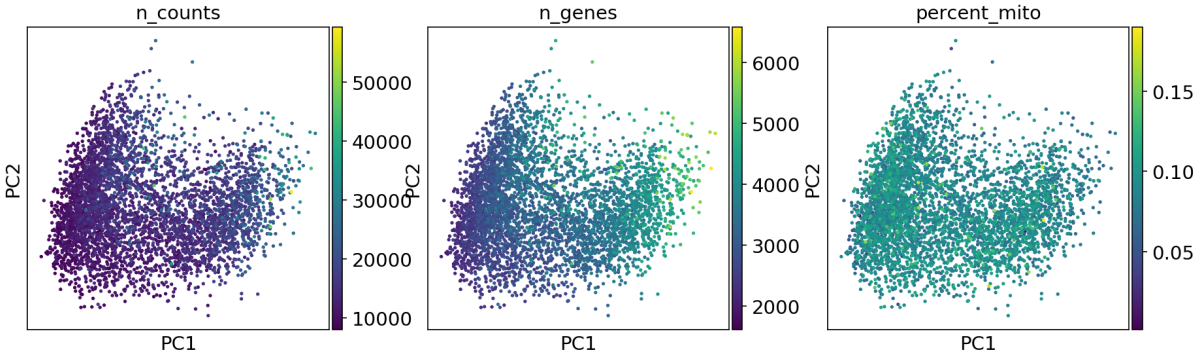


Figure 4.4: Projection of the data set into the PC1,PC2 plane. The cells are colored by number of reads (n_counts), the number of detected genes (n_genes), fraction of mitochondrial reads (percent_mito) and batch id (library_id).

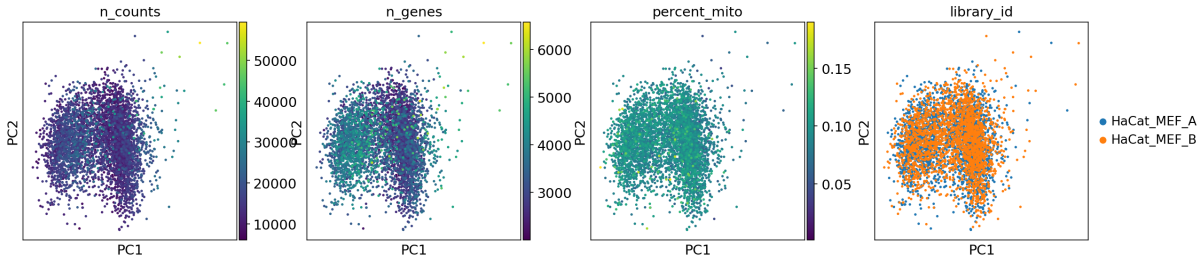


Figure 4.5: Projection of the data set into the PC1,PC2 plane after linear regression on the number of reads. The cells are colored by number of reads (n_counts), the number of detected genes (n_genes), fraction of mitochondrial reads (percent_mito).

4.3 Pseudotime Modelling

Here, We will model the gene expression levels in our data set as a function of cyclic time. Proliferating cells will be identified by scoring them against marker genes. The proliferating cells will then be sorted to represent their order with respect to the cell cycle.

4.3.1 Pseudotime Ordering

Here, 105 of the marker genes are present as highly variable genes (14 G1, 47 S, 74 G2/M). The score for each phase is calculated and we perform the PCA projection. In Figure 4.6 we see the resulting projection into the PC1,PC2 plane colored by the scores to each of the phases and the assigned phase for each cell. From the PCA projection, we can see three distinct clusters and that each cluster is correlated to high score in one of the three phases. In the PC1 component, the clusters with high scores in each of the phases are separated. A high score in G2/M is related to low values in PC1 and high G1 scores to a higher value. S lies between. Most of the variation related to the cell cycle seems to be explained by the PC2 component, where a low PC2 component is related to a high score for S and a high PC2 component is related to a high G2/M score. Both components captures variation related to the cell cycle and the three clusters with a high phase score form a triangular pattern in the PC1,PC2 plane.

The data set consists of 577 cells after filtering, of which 157 cells are classified as G1, 215

as S and 205 as G2/M. Figure 4.7 shows a new projection of the data set after filtering by the phase scores. The three clusters are now more distinct. We calculate the angle of the remaining cells in the PC1,PC2 plane. The cells are sorted accordingly and assigned an order in the pseudotime.

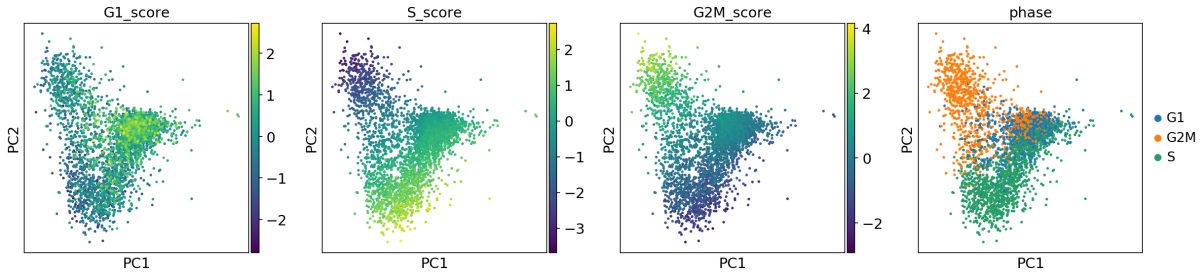


Figure 4.6: Projection into the PC1,PC2 plane of the data set with the highly variable marker genes as input. The plots are colored by G1 score, S score, G2/M score and assigned phase.

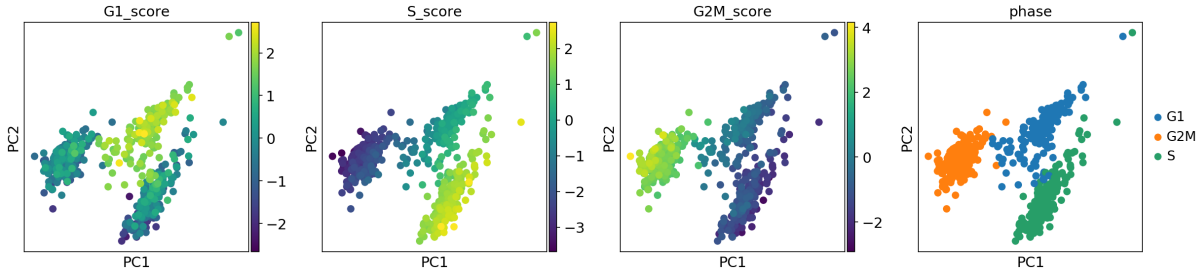


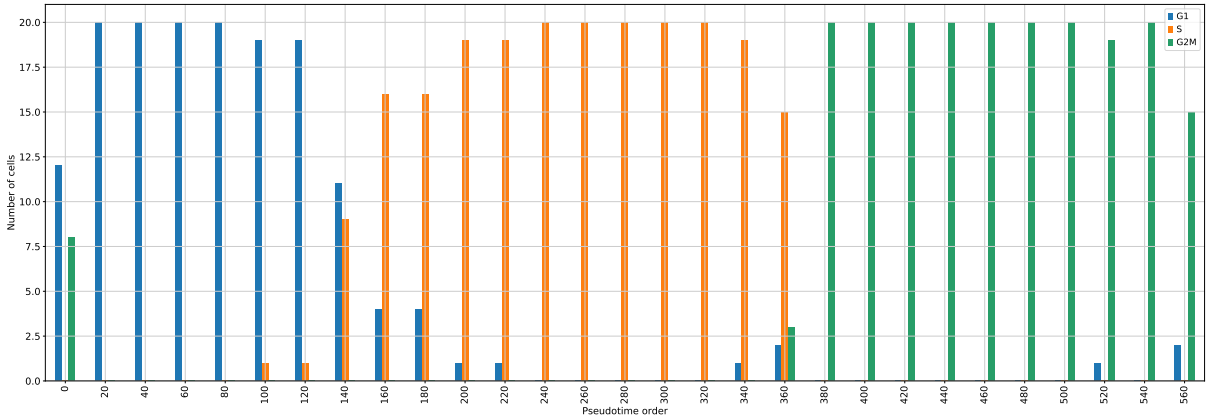
Figure 4.7: Projection into the PC1,PC2 plane of the data set after filtering by the phase scores. The plots are colored by G1 score, S score, G2/M score and assigned phase.

4.3.2 Phase Distribution Model

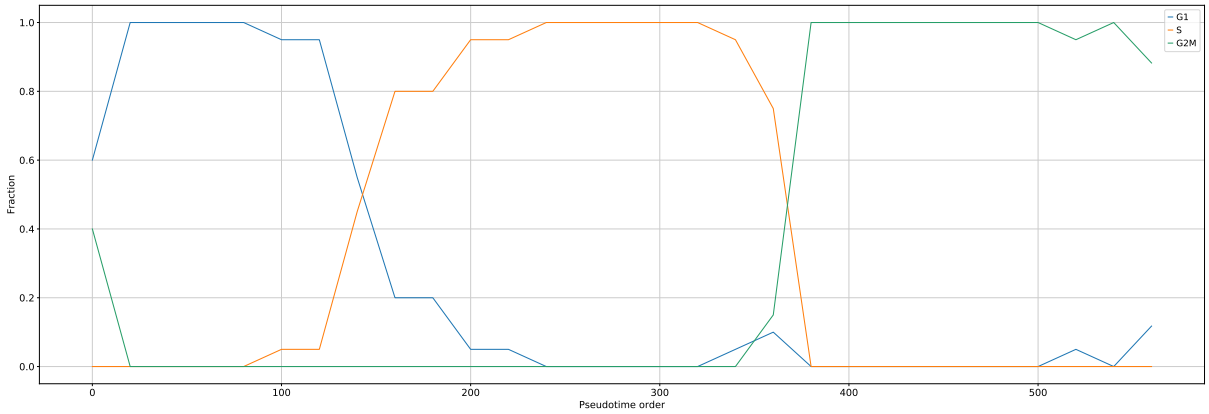
Figure 4.8 shows the number of cells per phase within each bin along the pseudotime ordering together with the linear curves representing the fraction of cells within each phase. The ordering has been shifted to align the G2/M-G1 boundary at zero. By intersecting the lines, the boundary for G1-S is found to be located at order 143 and the boundary for S-G2/M is located at order 368.

Figure 4.9 shows the comparison of the mean expressions of a set of marker genes within each of the modelled phases. Figure 4.9a shows the mean expression of the G1 marker genes. Six out of the ten present marker genes have their highest mean expression in the modelled G1 phase, but two of these show low overall expression. In Figure 4.9b, we see

that the marker genes for S does slightly better. Eight of the marker genes have their highest mean expression in the modelled S phase and two have their highest expression in the modelled G1 and G2/M phase. For the G2 and G2/M genes in Figure 4.9c and 4.9d, all 21 genes have their highest expression in G2/M. In total 35 of 41, or $\sim$ 85%, of the selected marker genes have their highest mean expression in their corresponding modelled phase in the pseudotime ordering.



(a)



(b)

Figure 4.8: **Pseudotime ordering for HaCat cells.** 4.8a) Number of cells per phase for each bin of size 20 through the pseudotime ordering of the cells. 4.8b) Linear curves representing the fraction of cells in each phase along the pseudotime ordering.

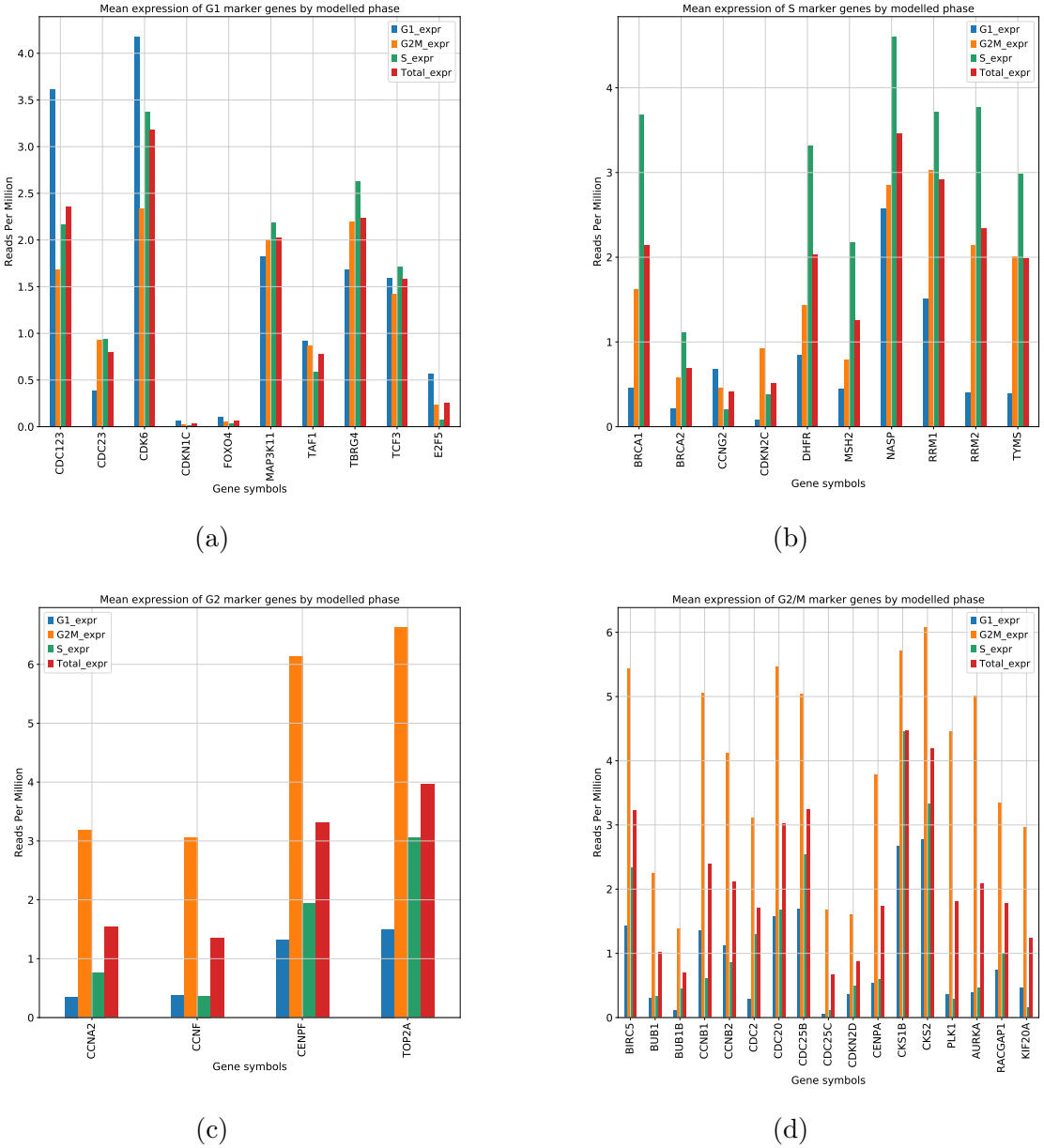


Figure 4.9: **Mean expressions for HaCat cells**. Mean expression of marker genes in the modelled phases along the pseudotime ordering. The following shows the mean expressions for 4.9a) G1 marker genes. 4.9b) S marker genes. 4.9c) G2 marker genes. 4.9d) G2/M marker genes.

4.4 Modelling Gene Expression with Cubic Smoothing Splines

From the acquired ordering of the cells, we want to model their gene expressions. We apply the strategy outlined in Chapter 3.7 to obtain the spline model for the genes in our

data set. First, we must determine a optimal smoothing parameter for our data set. We find the optimal smoothing parameter to be $p = 0.15$. The objective function is presented in Figure 4.10. The resulting model is presented in Figure 4.11 together with a scatter plot for the expression levels in each cell along the pseudotime ordering for each of the presented genes.

We see that some of the marker genes have cells that show similar expression levels throughout the ordering. Our spline model is still able produce cyclic expression because distribution of zero expressions varies. There are also examples of marker genes that have very distinct activity in their modelled phase. For example PLK1 and AURKA in G2.

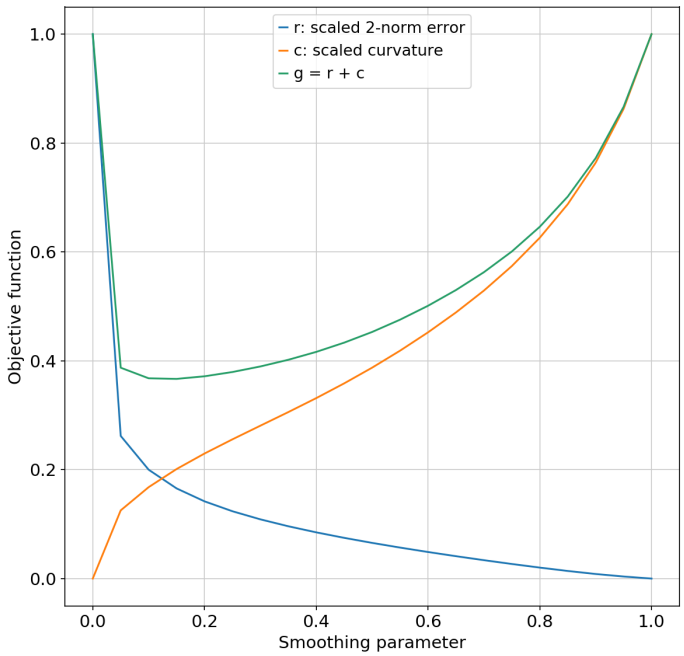
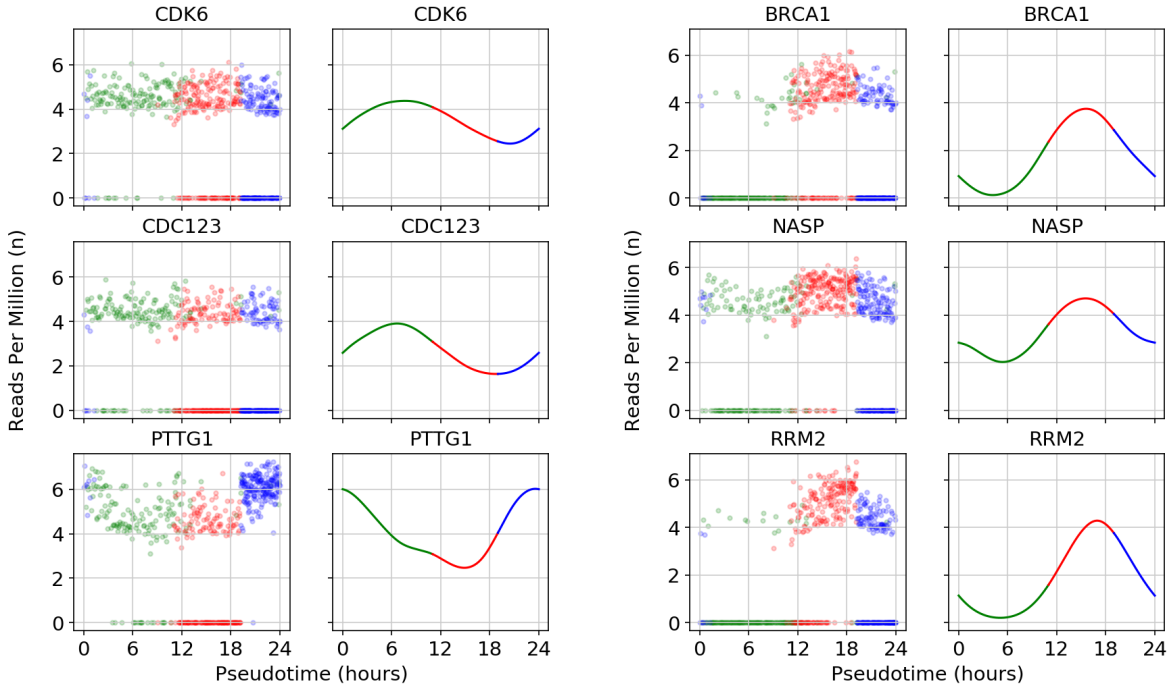
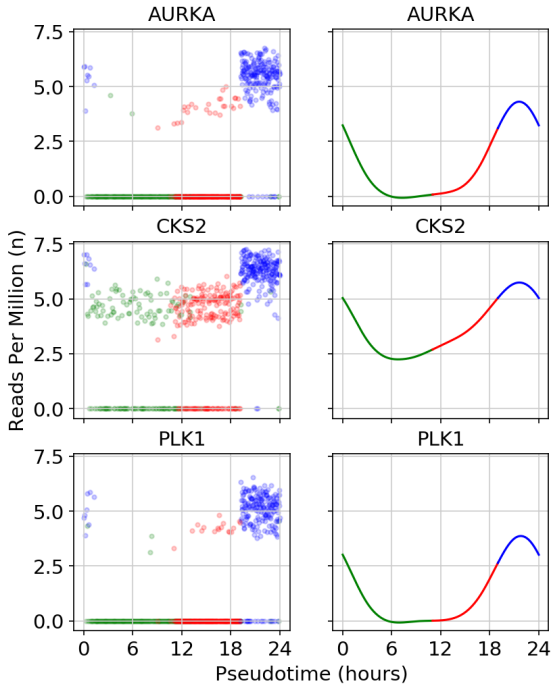


Figure 4.10: **Smoothing parameter for HaCat cells**. Scaled 2-norm of the residuals, $r(p)$, together with the curvature $c(p)$ and the objective function $g(p)$.

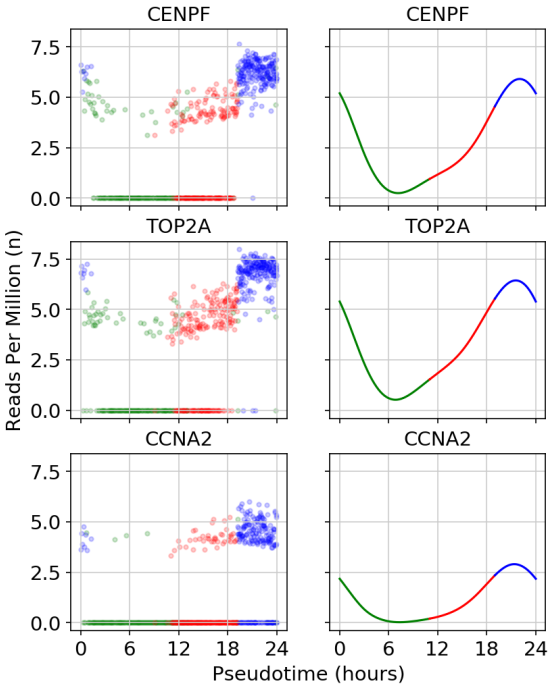


(a)

(b)



(c)



(d)

Figure 4.11: **Gene expression model for HaCat cells.** Scatter plots of the gene expression through the pseudotime ordering compared to the resulting spline model for each of the genes. The scatter plot is colored by the cell classification in Chapter 4.3.1 while the spline model is colored by modelled phase after the time. For the scatter plot, cells classified as G1 are green, red for S and blue for G2/M. In the spline model, the color is green between 0 and 11 hours (G1), red for 11 to 19 hours (S) and blue for 19 to 24 hours (G2/M). 4.11a) Shows the comparison for a selection of M/G1 (PTTG1) and G1 marker genes. 4.11b) Shows the comparison for a selection of S marker genes. 4.11d) Shows the comparison for a selection of G2 marker genes. 4.11c) Shows the comparison for a selection of G2/M marker genes.

4.5 Identification of Cell Cycle Periodic Genes

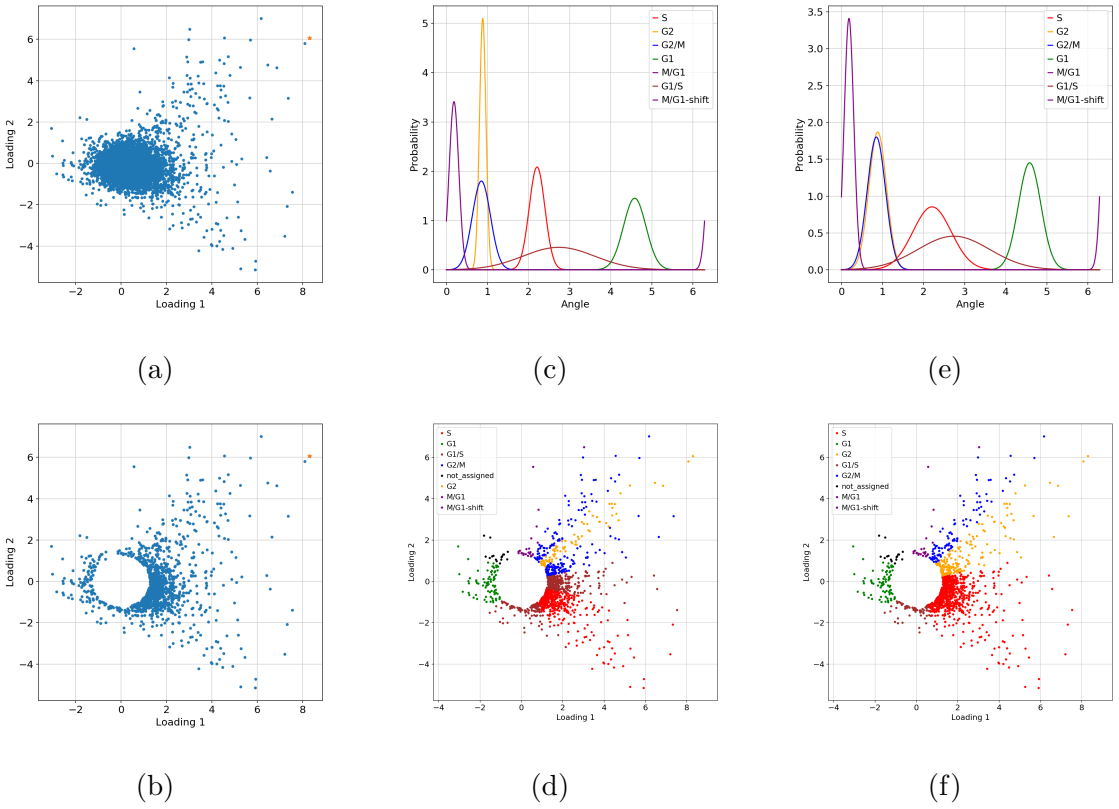
We now wish to identify the genes in our data set that show significant cyclic expression profiles. We use the spline model obtained for our data set and subsample all 18331 genes in our data set on hourly time points. Our input to the PLS will then consist of 25 observations and 18331 variables. Our response variables in the PLS will be one sine and one cosine function with a period of one cell cycle, 24 hours. To generate the observations, the sine and cosine are evaluated at the modelled timepoints corresponding to the generated observations for the genes.

We find a cut-off for the length in the loading plane, corresponding to a p-value 5% to be $r = 1.27$. There are 1239 genes in our dataset that show significant cell cycle periodicity. In Figure 4.12a and 4.12b the position of gene in the loading are presented before and after filtering. Figure 4.12c shows the initial phase angle distribution of the marker genes and Figure 4.12d shows the corresponding cell cycle phase assignments. Because of out-of-order phase assignments, the variances in the distribution are adjusted using the method in Chapter 3.8.3. The variance for G2 is increased in 9 iterations. For S, we need 8 iterations. The result is shown in Figure 4.12e and 4.12f. We see that the distributions for G2 and G2/M are almost identical. This indicates that their marker genes show similar expression patterns. There are 16 genes that are not captured by any of the marker gene distributions. By inspecting the spline model of these 16 genes, we find them to be G1 and assign the phase manually. The end result can be seen in Figure 4.13.

The number of assigned genes to each of the cell cycle phases can be seen in Table 4.5. Over half of the genes are assigned to the S phase. Of the 128 marker genes used, 111 are present as significant cell cycle periodic genes in this data set. Of these 111 marker genes, 69 has been assigned their expected phase.

Figure 4.14 shows the top eight genes within each classified cell cycle phase. For M/G1 we see that many of expression profiles among the top eight resembles those we would expect from G2 or G2/M. CCNB1 is known to be G2/M but here it is assigned to M/G1. The other phases presents expression profiles that clearly resembles their phases, but in G2 and G2/M we can see that both have assigned genes that are known to be in a neighboring phase. The expression profiles from the top eight G2 and G2/M are hard to distinguish

visually. From the phase assignment, we know that their marker genes showed similar expression profiles. It is therefore not surprising that we have boundary issues in the assignments between the phases. In S we also have one marker gene from a neighboring phase, but its expression profile is clearly S.



(a) (c) (e)

(b) (d) (f)

Figure 4.12: **PLS regression on the HaCat cells**. The phases appended with `-shift` is where we have accounted for the periodicity of the phases. 4.12a) Position of all 18331 in the loading plane. The most significant gene, TOP2A, is marked in yellow. 4.12b) The position in the loading plane of the 1239 genes that show significant cell cycle periodicity. The most significant gene, TOP2A, is marked in yellow. 4.12c) The initial phase angle distribution of the marker genes. 4.12d) The initial assignment of cell cycle phase. 4.12e) The phase angle distribution of the marker genes after adjusting variances according to out-of-order assignments. 4.12f) Assignment after adjusting variances according to out-of-order assignments.

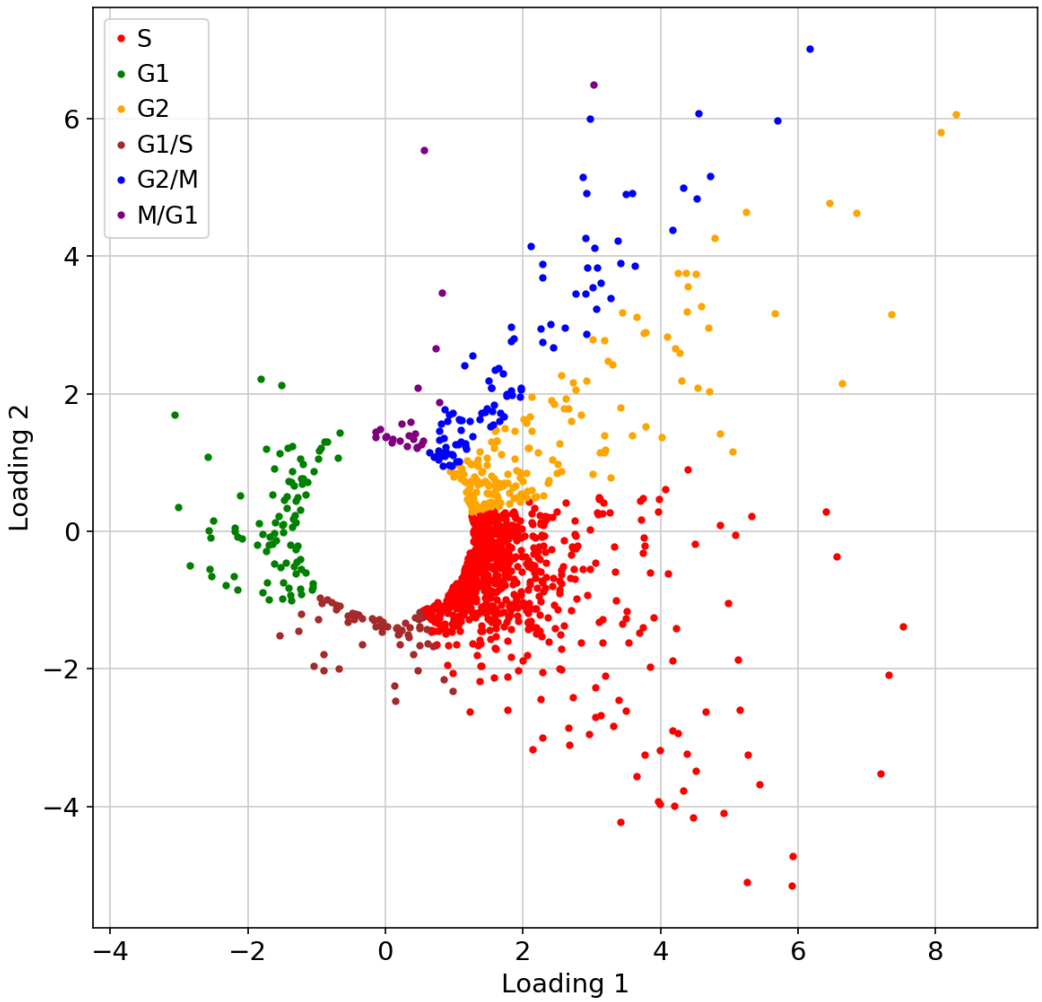
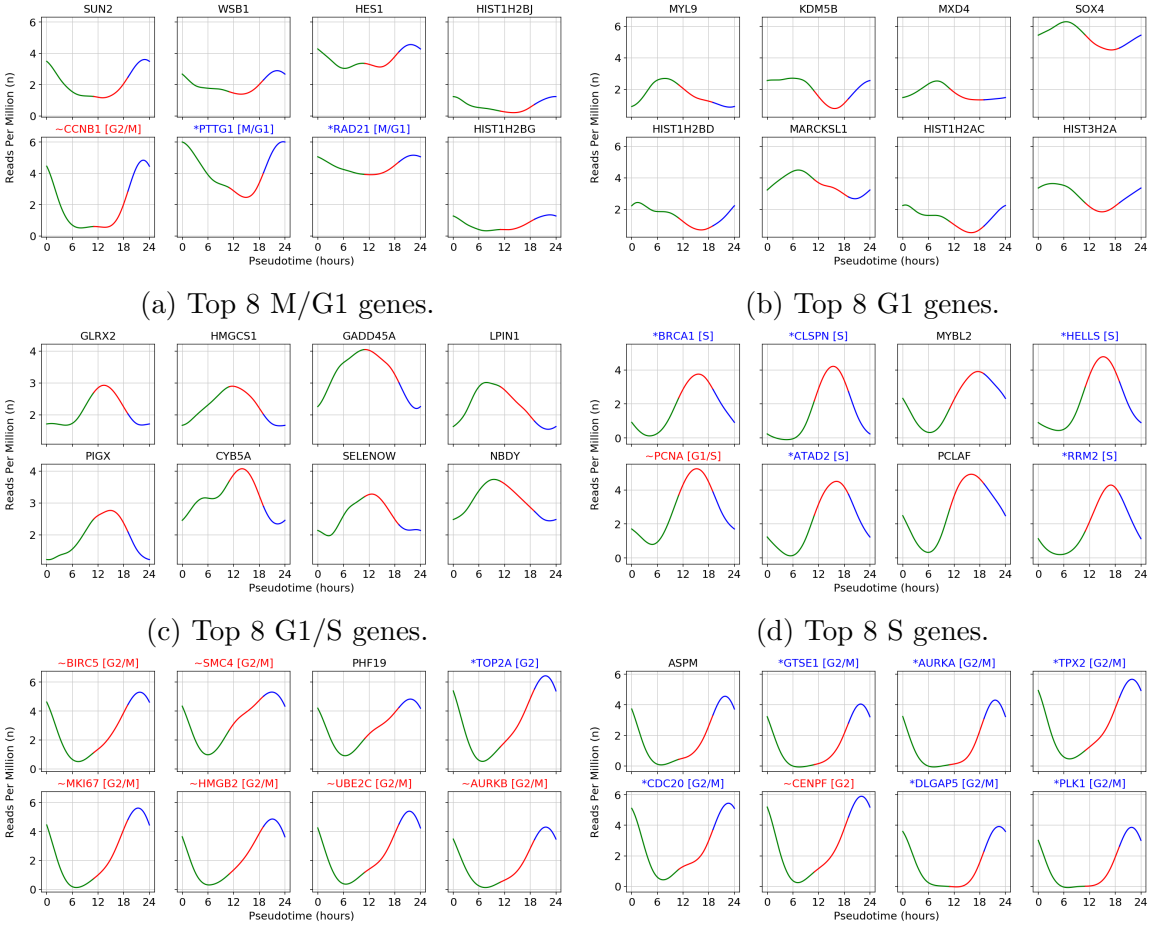


Figure 4.13: The final cell cycle phase classification for all significant genes in the HaCat data set.

Table 4.5: The number of assigned genes to each of the cell cycle phases.

Cell Cycle Phase	Assigned Genes
S	783
G2	179
G2/M	101
G1	94
G1/S	58
M/G1	24



(a) Top 8 M/G1 genes.

(b) Top 8 G1 genes.

(c) Top 8 G1/S genes.

(d) Top 8 S genes.

(e) Top 8 G2 genes.

(f) Top 8 G2/M genes.

Figure 4.14: **Top cyclic genes for HaCat cells**. Top 8 genes within each of the assigned phases, ranked by their length in the PLS loading plane. Genes prepended with a * marked in blue identify marker genes with their correct phase. Genes prepended with $\sim$ marked in red signify marker genes that are not classified as expected.

4.6 Functional Enrichment

We perform a functional enrichment anlysis of the genes found to have significant cell cycle periodicity in the previous chapter. We perform the enrichment on each of the assigned phases. We present only the top 10 significant terms, where they exist. More extensive tables can be found in Appendix C.3.

The significant terms in M/G1 seems to be mostly related to functions associated with late G2/M. We know from the previous inspection of expression profiles that many of the genes assigned to M/G1 resembled more the behaviour of G2/M. Hence, significant terms

for G2/M here should therefore be expected.

No significant terms for the genes assigned to G1 is found for this data set and the significant terms found in G1/S has no obvious explaination. The significant terms for genes assigned to S clearly shows expected functions for this phase. From distributions in the phase assignment, we remember that G2 and G2/M were close to identical. A consequence of this is that there is left a gap between G2 and S where ideally G2 should have a distribution present. Hence, genes that are towards late G2 can be assigned to S. Effects from this can also be viewed from the significant terms in G2 and G2/M where both show similar terms. These are terms that represent functions assoicated with G2/M.

Table 4.6: The 10 most significant terms in the functional enrichment of HaCat cells classified as M/G1 by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0033044	regulation of chromosome organization	342	7	1.939e-04
2	GO:BP	GO:0051276	chromosome organization	1220	10	1.065e-03
3	GO:BP	GO:2001251	negative regulation of chromosome organization	145	5	1.266e-03
4	GO:BP	GO:0033047	regulation of mitotic sister chromatid segregation	69	4	2.306e-03
5	GO:BP	GO:0000278	mitotic cell cycle	1015	9	2.494e-03
6	GO:BP	GO:2001252	positive regulation of chromosome organization	173	5	3.027e-03
7	GO:BP	GO:0033045	regulation of sister chromatid segregation	81	4	4.395e-03
8	GO:BP	GO:0006323	DNA packaging	205	5	6.958e-03
9	GO:BP	GO:0051984	positive regulation of chromosome segregation	28	3	8.464e-03
10	GO:BP	GO:0051983	regulation of chromosome segregation	103	4	1.147e-02

g:Profiler (biit.cs.ut.ee/gprofiler)

Table 4.7: Significant terms in the functional enrichment of Hacat cells classified as G1/S by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0009628	response to abiotic stimulus	1245	16	6.829e-04
2	GO:BP	GO:0009314	response to radiation	450	10	1.306e-03

g:Profiler (biit.cs.ut.ee/gprofiler)

Table 4.8: The 10 most significant terms in the functional enrichment of Hacat cells classified as S by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0006260	DNA replication	282	100	1.269e-64
2	GO:BP	GO:0006259	DNA metabolic process	926	160	3.348e-56
3	GO:BP	GO:0006261	DNA-dependent DNA replication	153	67	2.301e-49
4	GO:BP	GO:0006281	DNA repair	545	116	6.694e-49
5	GO:BP	GO:0006974	cellular response to DNA damage stimulus	849	142	2.561e-47
6	GO:BP	GO:0007049	cell cycle	1850	215	3.998e-47
7	GO:BP	GO:0022402	cell cycle process	1383	170	9.541e-39
8	GO:BP	GO:0033554	cellular response to stress	2052	205	3.027e-34
9	GO:BP	GO:0051276	chromosome organization	1220	150	1.854e-33
10	GO:BP	GO:0000278	mitotic cell cycle	1015	130	3.862e-30

g:Profiler (biit.cs.ut.ee/gprofiler)

Table 4.9: The 10 most significant terms in the functional enrichment of Hacat cells classified as G2 by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0051301	cell division	605	52	8.687e-32
2	GO:BP	GO:0007049	cell cycle	1850	80	1.448e-30
3	GO:BP	GO:0007059	chromosome segregation	314	39	6.533e-29
4	GO:BP	GO:0000278	mitotic cell cycle	1015	59	2.205e-27
5	GO:BP	GO:0022402	cell cycle process	1383	66	2.773e-26
6	GO:BP	GO:1903047	mitotic cell cycle process	871	53	5.221e-25
7	GO:BP	GO:0000280	nuclear division	427	36	1.730e-20
8	GO:BP	GO:0098813	nuclear chromosome segregation	261	29	2.661e-19
9	GO:BP	GO:0048285	organelle fission	469	36	4.230e-19
10	GO:BP	GO:0051276	chromosome organization	1220	54	6.509e-19

g:Profiler (biit.cs.ut.ee/gprofiler)

Table 4.10: The 10 most significant terms in the functional enrichment of Hacat cells classified as G2/M by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0051301	cell division	605	42	4.701e-33
2	GO:BP	GO:1903047	mitotic cell cycle process	871	46	1.593e-31
3	GO:BP	GO:0000278	mitotic cell cycle	1015	48	5.138e-31
4	GO:BP	GO:0140014	mitotic nuclear division	284	31	3.635e-29
5	GO:BP	GO:0007049	cell cycle	1850	56	1.744e-27
6	GO:BP	GO:0022402	cell cycle process	1383	50	4.200e-27
7	GO:BP	GO:0000280	nuclear division	427	33	2.458e-26
8	GO:BP	GO:0048285	organelle fission	469	33	5.267e-25
9	GO:BP	GO:0010564	regulation of cell cycle process	797	36	5.288e-21
10	GO:BP	GO:0007017	microtubule-based process	745	35	7.462e-21

g:Profiler (biit.cs.ut.ee/gprofiler)

4.7 Additional Data Sets

We present the results obtained by applying our method on the two data sets described in Chapter 3.1.4. We perform sequence alignment using STARsolo with the same parameter set as for the HaCat cells. In addition, we use the same reference genome and the same gene annotation file.

4.7.1 293t Results

We apply our method for identification of cyclic gene expressions on the 293t cells. The cells origin are described in Chapter 3.1.4.

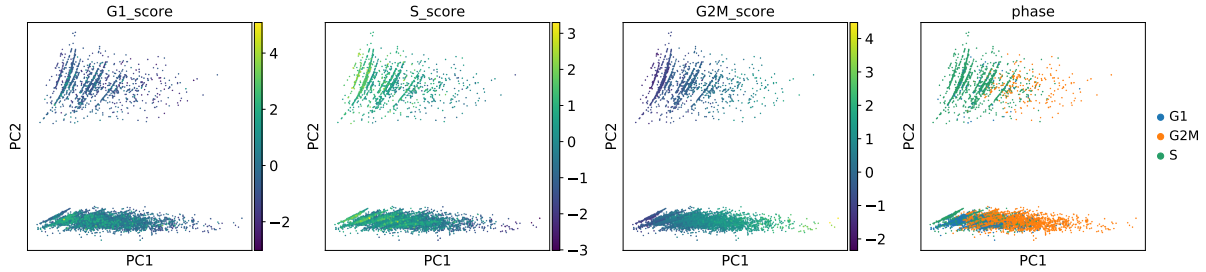
Pseudotime Ordering

After initial pre-processing and quality control (see Appendix A.1), we are left with 11447 cells and 5430 genes classified as highly variable. Here, only 18 of the marker genes are present as highly variable genes. Three are G1, six are S and nine are G2/M. The cells are given a score for each of the cell cycle phases. From Figure 4.15a, we see that there are some peculiar patterns from the PCA analysis. As we can see in Figure 4.15b, these patterns are not explained by any of the usual quality control metrics. PC1 seems to be related to the cell cycle phase scores, with G2/M associated with higher PC1 values, S score with lower PC1 values and G1 in the middle.

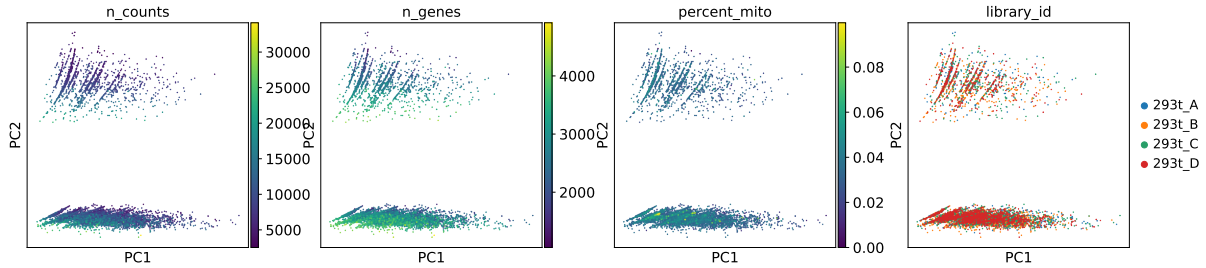
After filtering on the phase scores, we can see that the projection in 4.15c looks much better with a clear separation of all three phase scores. We note that there are still some effects left. Especially for the G1 cells, where two clusters can be seen. We are left with a total of 629 cells where 356 are in G1, 195 in S and 78 in G2/M. The cells are ordered by their angle and we calculate each cells corresponding time point in the estimated 24 hour duration of the human cell cycle. Additional figures can be found in Appendix A.2.

Since these patterns were not present during pre-processing, we suspected that they might occur due to the low input of marker genes in our PCA analysis. In Appendix B.1, we can see the results of performing a PCA on the HaCat cells with the same set of marker

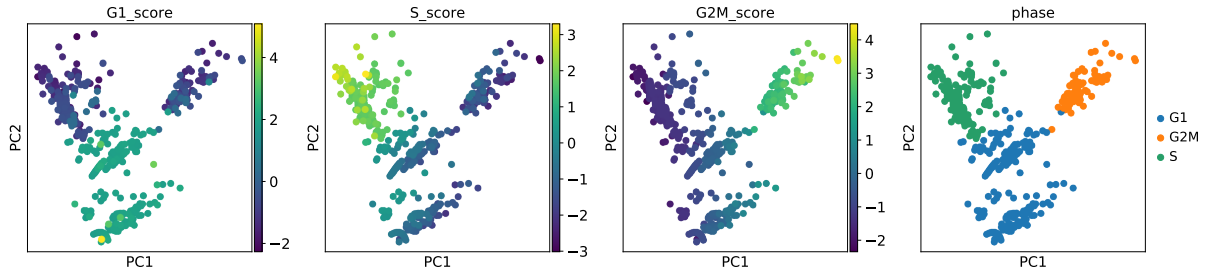
genes that were present in 293t. We observe similar behaviour.



(a)



(b)



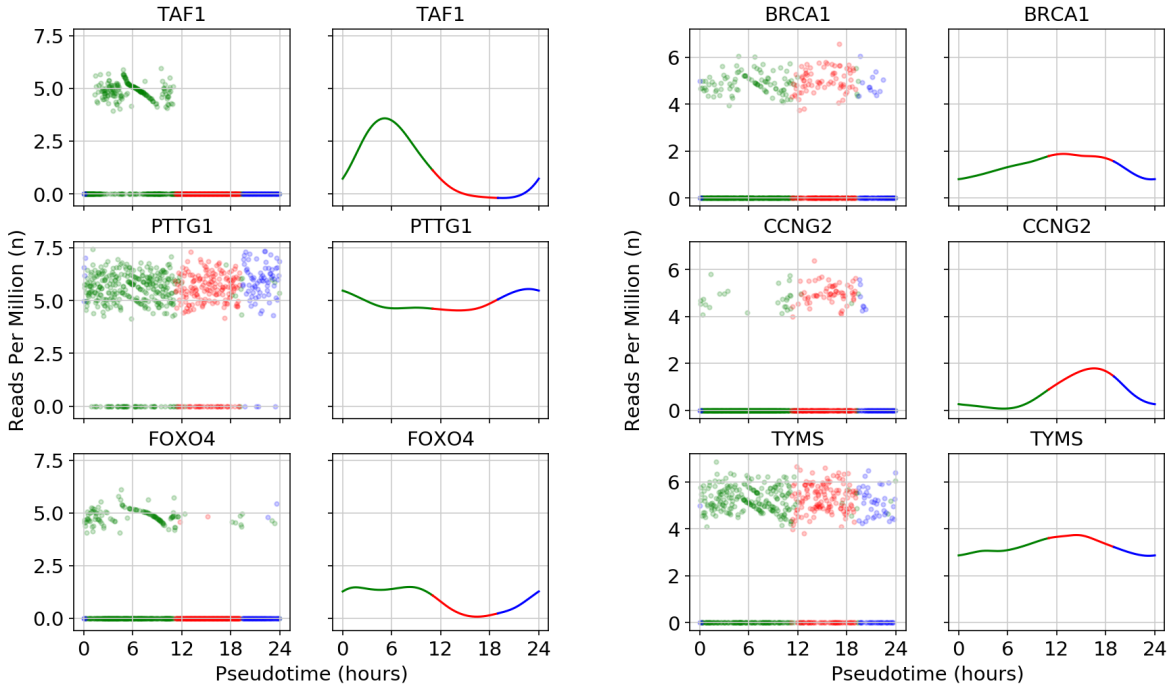
(c)

Figure 4.15: **PCA analysis of the 293t cells**. The PCA projection of the 293t data set with only the highly variable marker genes as input. 4.15a) The PCA projection after cell cycle phase scoring. The projections are colored by G1 score, S score, G2/M score and assigned phase. 4.15b) The PCA projection colored by number of reads, numbder of detected genes, percent mitochondrial reads and library id. 4.15c) The PCA projection after filtering on cell cycle phase score.

Gene Expression Modelling

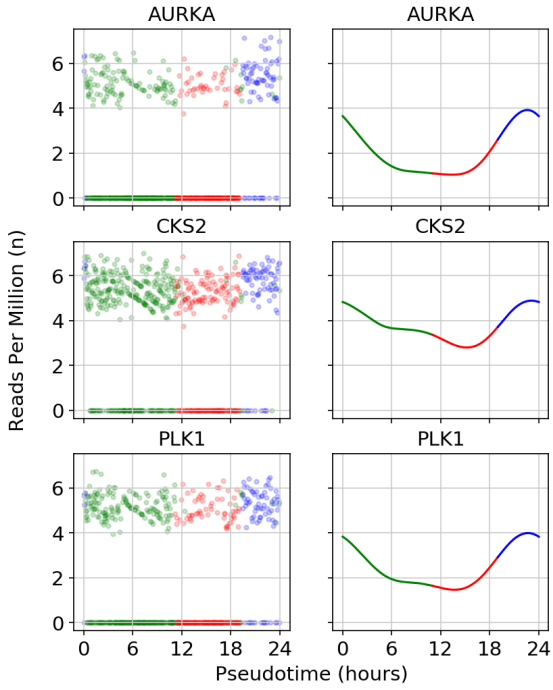
We find an optimal smoothing parameter of $p = 0.15$ for the smoothing cubic spline model in this data set. Figure 4.16 show the spline model applied to a selection of marker genes. We observe some artefacts in the scatter plots where some expression levels are clustered together in stripes. This is most visible in TAF1 and FOXO4. The effect seems to be mostly located in the G1 phase. This is also where we found the most distinct patterns in the PCA analysis of the 293t.

The constant gene expression levels in the scatter plots are more present here than in the HaCat cells, but that the spline still performs well due to distribution of zero expressions in the ordering.

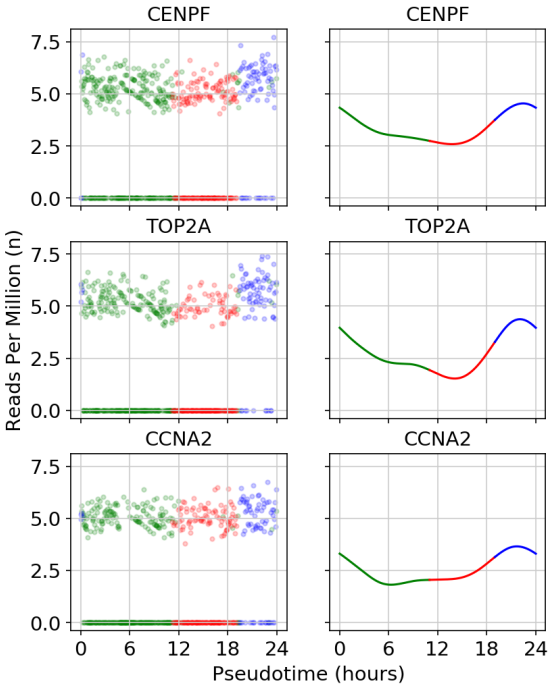


(a)

(b)



(c)



(d)

Figure 4.16: **Gene expression model for 293t cells.** Scatter plots of the gene expression through the pseudotime ordering compared to the resulting spline model for each of the genes. The scatter plot is colored by the cell classification in Chapter 4.3.1 while the spline model is colored by modelled phase after the time. For the scatter plot, cells classified as G1 are green, red for S and blue for G2/M. In the spline model, the color is green between 0 and 11 hours (G1), red for 11 to 19 hours (S) and blue for 19 to 24 hours (G2/M). We show a comparison for a selection of marker genes in 4.16a) M/G1 (PTTG1) and G1. 4.16b) S. 4.16d) G2. 4.16c) G2/M.

Cell Cycle Periodic Genes

We use the spline model to subsample the gene expressions in the data set on an hourly interval and apply the PLS regression to a cell cycle periodic sine and cosine response. With a significance level corresponding to a p-value of 5%, we find the cutoff value for the distance from the origin in the loading plane to be $r = 0.92$. There 1295 genes that are above this level. Figure 4.17 shows the progression of the cell cycle phase assignment in the loading plane, using the same method as for the HaCat cells. The variance for the G2 distribution must be increased over 24 iterations before there are no out-of-order assignments between S and G2. As we see in Figure 4.17e and 4.17f, using this method on the out of order assignments in G1/S and S is futile. The reason for this is that the mean of the phase distribution for the G1/S marker genes lies between S and G2. This is solved by admitting that this method is unable to separate the two phases in the loading plane and we combine the assignments from both to one phase, S. The final assignments in the loading plane can be seen in Figure 4.18.

The number of genes assigned to each phase can be seen in Table 4.11. Genes assigned to G2/M make up roughly 1/3 of all the significant genes in the data set. Of the 128 marker genes, 94 show significant cell cycle periodicity, where 65 are assigned to their expected phase. Figure 4.19 shows the top eight genes in each phase, ranked by their distance from the origin in the loading plane.

The top eight genes in M/G1 show much of the same behaviour as for the HaCat cells. Many show expression profiles expected of G2/M. Three of the genes here are also known to be G2/M and one is known to be S. The other phases do better. Also here, G2 have marker genes that are known to be in G2/M while G2/M have seven marker genes among the top eight that are classified to their expected phase.

Also here we see that phase assignment by the probability distribution of marker show issues with assignments in the boundary of the cell cycle phases.

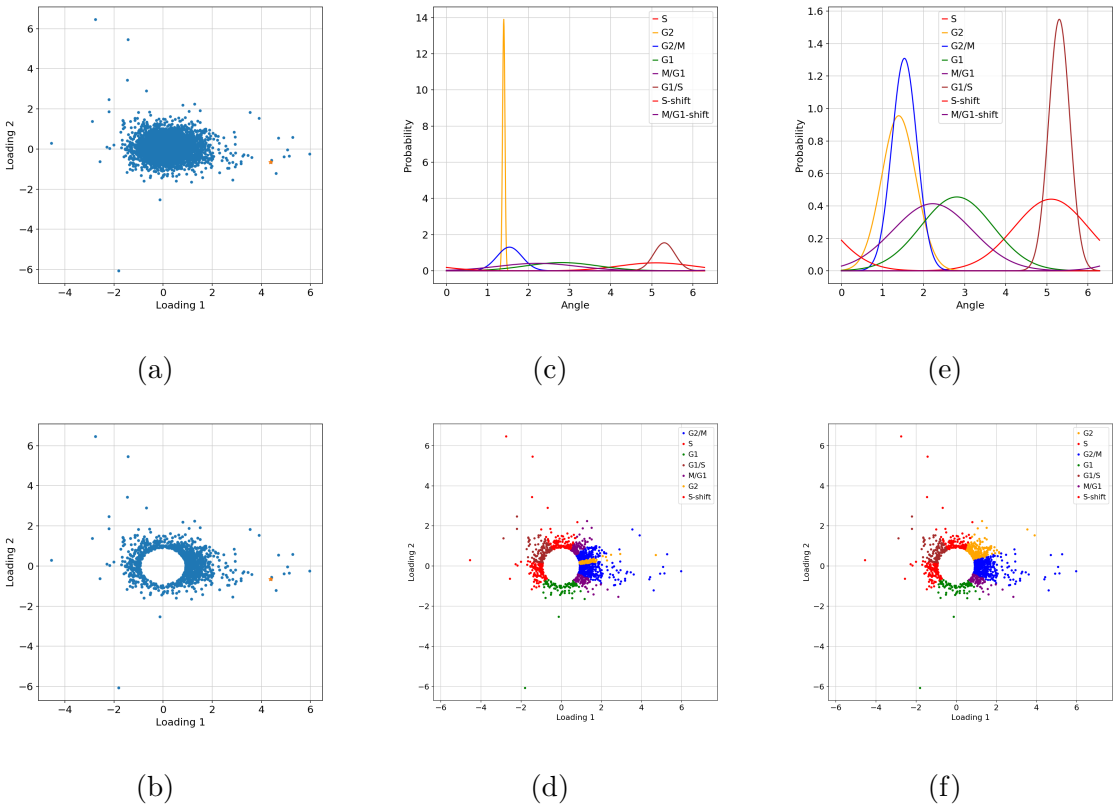


Figure 4.17: **PLS regression on the 293t cells**. The phases appended with `-shift` is where we have accounted for the periodicity of the phases. 4.17a) Position of all 18957 in the loading plane. TOP2A, is marked in yellow. 4.17b) The position in the loading plane of the 1295 genes that show significant cell cycle periodicity. TOP2A is marked in yellow. 4.17c) The initial phase angle distribution of the marker genes. 4.17d) The initial assignment of cell cycle phase. 4.17e) The phase angle distribution of the marker genes after adjusting variances according to out-of-order assignments. 4.17f) Assignment after adjusting variances according to out-of-order assignments.

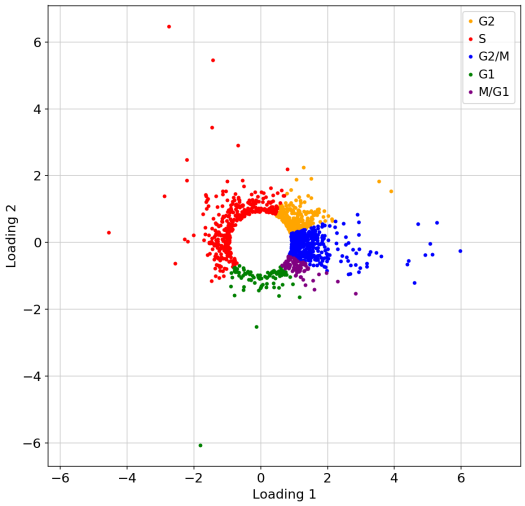
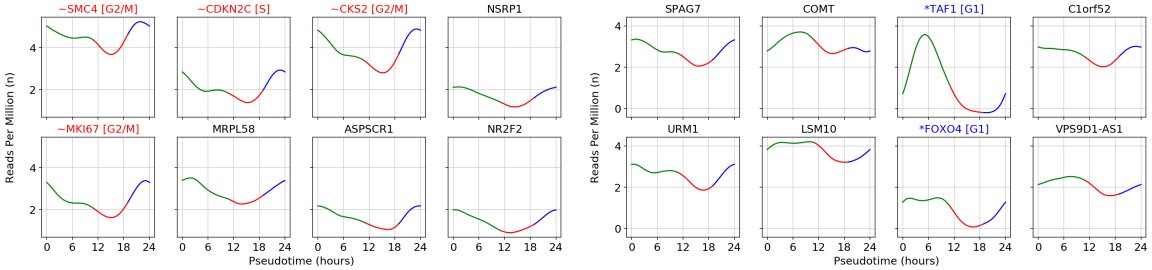


Figure 4.18: The final cell cycle phase classification for all significant genes in the 293t data set.

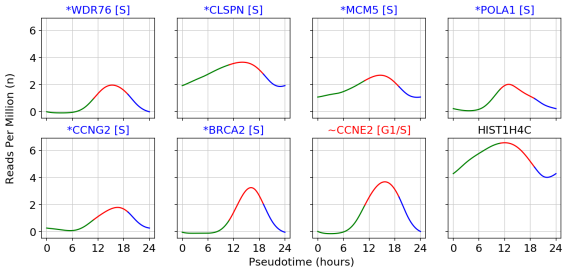
Table 4.11: The number of assigned genes to each of the cell cycle phases for the 293t cells.

Cell Cycle Phase	Assigned Genes
G2/M	461
S	364
G2	288
G1	94
M/G1	88

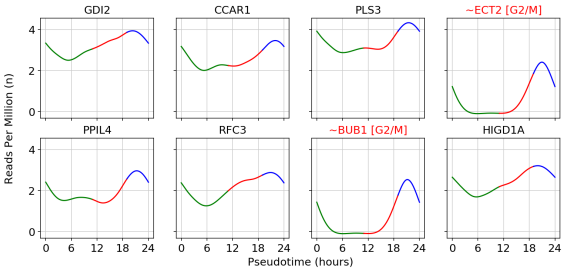


(a) Top 8 M/G1 genes.

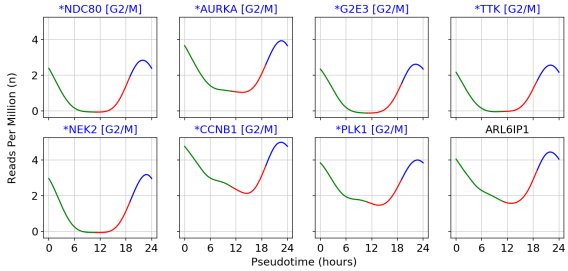
(b) Top 8 G1 genes.



(c) Top 8 S genes.



(d) Top 8 G2 genes.



(e) Top 8 G2/M genes.

Figure 4.19: **Top cyclic genes for 293t cells**. Top 8 genes within each of the assigned phases, ranked by their length in the PLS loading plane. Genes prepended with a * marked in blue identify marker genes with their correct phase. Genes prepended with ∼ marked in red signify marker genes that are not classified as expected. Note: In this data set G1/S and S was merged into one phase, S, because the G1/S distribution was out of order

Functional Enrichment

We perform functional enrichment using gprofiler2 for all the significant genes in each of the assigned phases. We present only the top 10 significant terms, where they exist. More extensive tables can be found in Appendix C.4.

There are no significant terms for M/G1 and G1/S show one significant term, RNA processing. This term is related to processing pre mRNA to mature mRNA. It is not directly specific to cell cycle processes. For the genes in S, the top 10 significant terms are strongly correlated to biological process involved in the synthesis of DNA. G2 also shows

significant terms for RNA processing together with other mRNA production processes. In addition, generic cell cycle terms are present and some that are more related to G2/M, like mitotic cell cycle. G2/M shows significant terms that are expected in this phase.

Table 4.12: Significant terms in the functional enrichment of 293t cells classified as G1 by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0006396	RNA processing	925	15	1.738e-02

g:Profiler (biit.cs.ut.ee/gprofiler)

Table 4.13: The 10 most significant terms in the functional enrichment of 293t cells classified as S by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0006260	DNA replication	282	40	3.315e-21
2	GO:BP	GO:0006261	DNA-dependent DNA replication	153	28	3.469e-17
3	GO:BP	GO:0006259	DNA metabolic process	926	62	1.801e-16
4	GO:BP	GO:0044786	cell cycle DNA replication	66	18	1.436e-13
5	GO:BP	GO:0033260	nuclear DNA replication	55	16	2.514e-12
6	GO:BP	GO:0007049	cell cycle	1850	82	6.830e-12
7	GO:BP	GO:0022402	cell cycle process	1383	68	2.134e-11
8	GO:BP	GO:0006974	cellular response to DNA damage stimulus	849	51	4.246e-11
9	GO:BP	GO:0044770	cell cycle phase transition	624	41	1.160e-09
10	GO:BP	GO:0006281	DNA repair	545	38	1.458e-09

g:Profiler (biit.cs.ut.ee/gprofiler)

Table 4.14: The 10 most significant terms in the functional enrichment of 293t cells classified as G2 by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0006396	RNA processing	925	44	3.693e-08
2	GO:BP	GO:0007049	cell cycle	1850	59	6.702e-05
3	GO:BP	GO:0008380	RNA splicing	431	24	1.191e-04
4	GO:BP	GO:0022613	ribonucleoprotein complex biogenesis	455	24	3.265e-04
5	GO:BP	GO:1903047	mitotic cell cycle process	871	35	3.758e-04
6	GO:BP	GO:0000278	mitotic cell cycle	1015	38	6.367e-04
7	GO:BP	GO:0006364	rRNA processing	204	15	1.336e-03
8	GO:BP	GO:0010564	regulation of cell cycle process	797	32	1.361e-03
9	GO:BP	GO:0016071	mRNA metabolic process	852	33	1.993e-03
10	GO:BP	GO:0022402	cell cycle process	1383	45	2.366e-03

g:Profiler (biit.cs.ut.ee/gprofiler)

Table 4.15: The 10 most significant terms in the functional enrichment of 293t cells classified as G2/M by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0051301	cell division	605	67	7.191e-23
2	GO:BP	GO:0000278	mitotic cell cycle	1015	82	1.880e-19
3	GO:BP	GO:0000819	sister chromatid segregation	185	36	2.939e-19
4	GO:BP	GO:0140014	mitotic nuclear division	284	43	6.108e-19
5	GO:BP	GO:0007059	chromosome segregation	314	45	6.150e-19
6	GO:BP	GO:0007049	cell cycle	1850	114	1.100e-18
7	GO:BP	GO:1903047	mitotic cell cycle process	871	74	1.749e-18
8	GO:BP	GO:0022402	cell cycle process	1383	93	6.080e-17
9	GO:BP	GO:0000070	mitotic sister chromatid segregation	153	31	8.328e-17
10	GO:BP	GO:0000280	nuclear division	427	49	1.367e-16

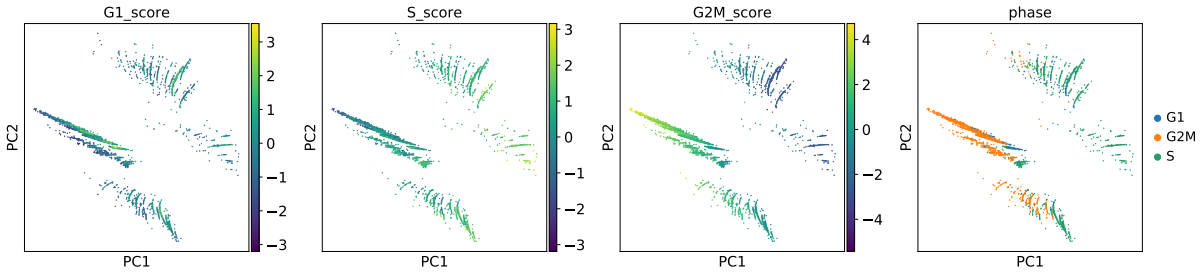
g:Profiler (biit.cs.ut.ee/gprofiler)

4.7.2 Jurkat Results

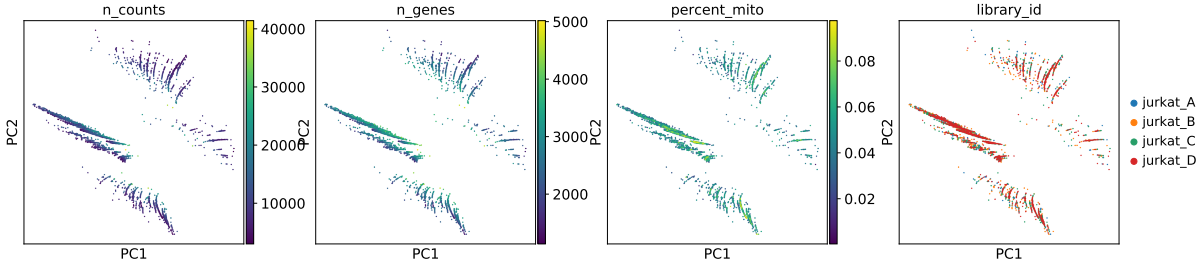
We apply our method for identification of cyclic gene expressions on the Jurkat cells. The cells origin are described in Chapter 3.1.4.

Pseudotime Ordering

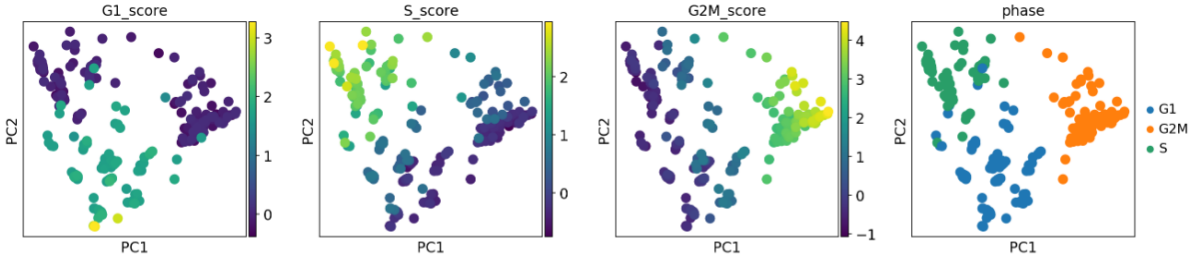
After pre-processing of the Jurkat cells (Appendix A.3), we are left with 12681 viable cells and 5246 genes classified as highly variable. Only 13 of our marker genes are present among the highly variable genes (three in G1, six in S and four in G2/M). The cell cycle phase scores are calculated and we apply a filter based on the scores. Figure 4.20 shows the resulting PCA projections before and after filtering, with only the highly variable marker genes as input. Similar behaviour as for 293t cells and the HaCat cells in Appendix B.1 can be observed, but also here we are able to get a separation of the phases after filtering. The cells are ordered based on their angle in PC1,PC2 plane and we assign them a time point in the cell cycle. Additional figures are presented in Appendix A.4. We have 137 cells classified as G1, 102 for G2/M and 73 for S.



(a)



(b)

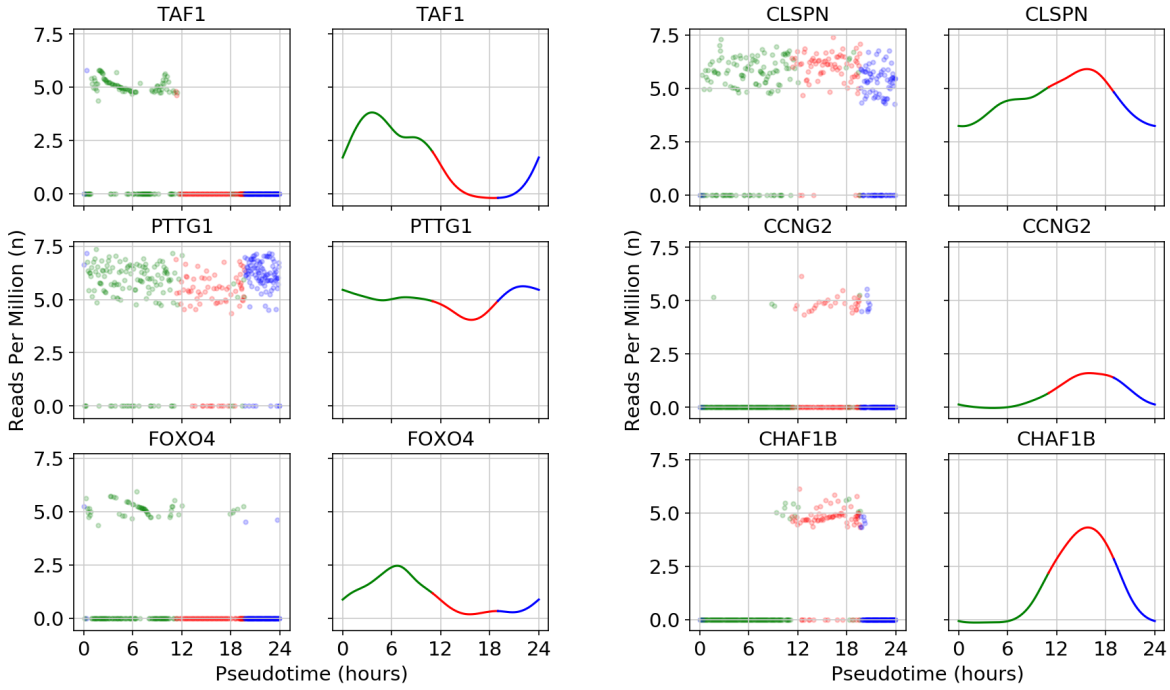


(c)

Figure 4.20: **PCA analysis of Jurkat cells**. The PCA projection of the Jurkat data set with only the highly variable marker genes as input. 4.20a) The PCA projection after cell cycle phase scoring. The projections are colored by G1 score, S score, G2/M score and assigned phase. 4.20b) The PCA projection colored by number of reads, number of detected genes, percent mitochondrial reads and library id. 4.20c) The PCA projection after filtering on cell cycle phase score.

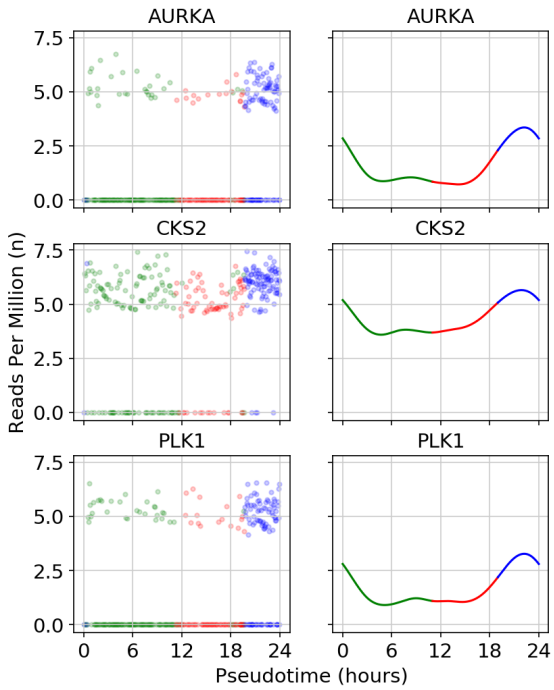
Gene Expression Modelling

We find an optimal smoothing parameter of $p = 0.25$ for the Jurkat cells. Figure 4.21 shows the spline on a selection of marker genes in each of the phases. The same trends as in the HaCat and 293t cells are present. The variation of the modelled gene expression patterns are mainly driven by the fraction of zeros within each region. Both CCNG2 and CHAF1B show very distinct activity in the S phase. Many of the other marker genes have distinct bumps around the same timepoint at ∼9 hours, but the spline model show distinct activity in all phases.

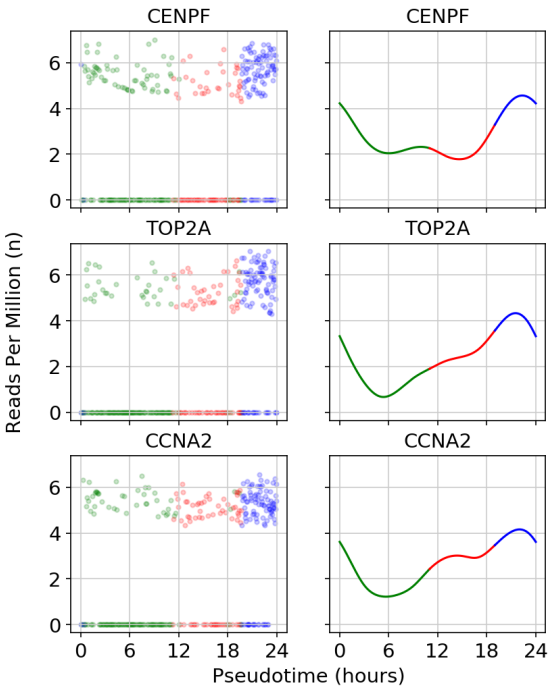


(a)

(b)



(c)



(d)

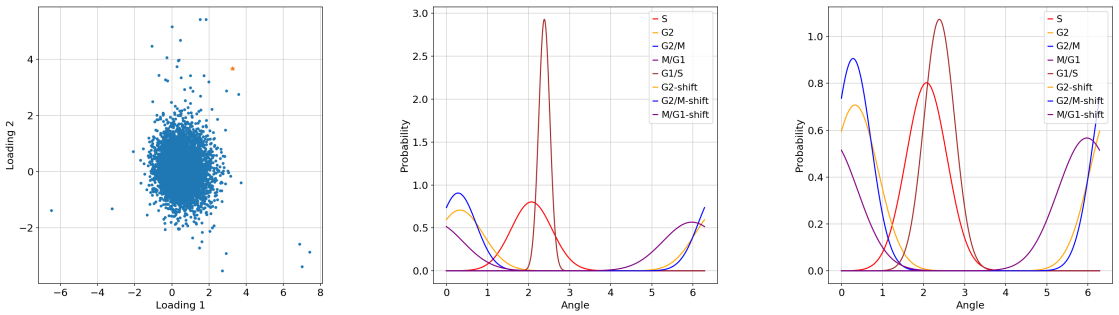
Figure 4.21: **Gene expression model for Jurkat cells.** Scatter plots of the gene expression through the pseudotime ordering compared to the resulting spline model for each of the genes. The scatter plot is colored by the cell classification in Chapter 4.3.1 while the spline model is colored by modelled phase after the time. For the scatter plot, cells classified as G1 are green, red for S and blue for G2/M. In the spline model, the color is green between 0 and 11 hours (G1), red for 11 to 19 hours (S) and blue for 19 to 24 hours (G2/M). The comparison is presented for marker genes in 4.21a) M/G1 (PTTG1) and G1. 4.21b) S. 4.21d) G2. 4.21c) G2/M.

Cell Cycle Periodic Genes

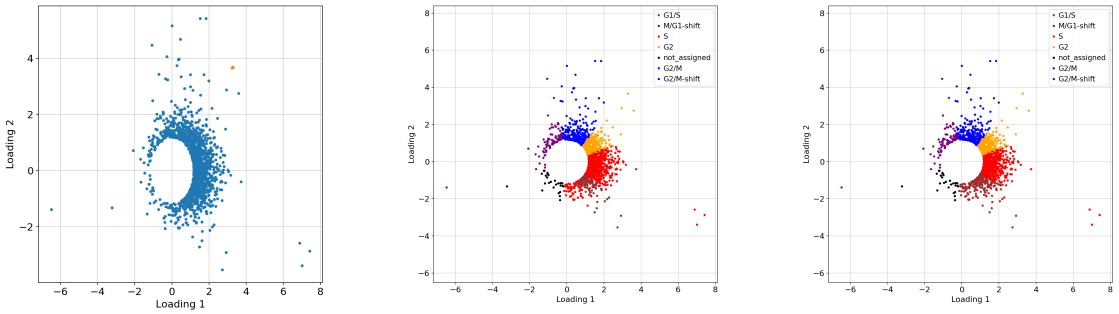
A PLS regression is performed on the data set. Again, the spline model for all the genes in the data set is subsampled on hourly intervals and the resulting data set of 25 observations is used as input to the PLS. We find a cutoff of $r = 1.18$, corresponding to a significance a $p = 5\%$, for the distance from the origin in the PLS loading plane. A total of 1478 genes show significant cell cycle periodicity. The significant genes are assigned a cell cycle phase according to the phase angle distribution of the marker genes. The result can be seen in Figure 4.22. Because of out of order assignments with G1/S and S, the variance of G1/S is increased incrementally by 25% over the course of nine iterations. As with the other data sets, marker genes in G2 and G2/M have similar distributions also for the Jurkat cells. We are still left with one region of genes that are not assigned. This region lies between M/G1 and G1/S. By inspection, we confirm that these genes show expression profiles expected from G1 and assign them manually to this phase. The final result is presented in Figure 4.23.

The number of genes within each cell cycle phase is presented in Table 4.16. Of the 128 marker genes, we have 88 present as significant cell cycle periodic genes. Of these 88, 56 are classified to be in their expected phases. Figure 4.24 presents the top eight genes within each of the assigned phases, ranked by their length in the loading plane.

M/G1 has genes assigned from both G2/M and G1 and several of the expression profiles here show G2/M signatures. The rest of the phases show expression profiles that is expected from their respective phases, but the phase assignments yield boundary issues also here.



(a) (c) (e)



(b) (d) (f)

Figure 4.22: **PLS regression on the Jurkat cells**. The phases appended with `-shift` is where we have accounted for the periodicity of the phases. 4.22a) Position of all 17867 in the loading plane. TOP2A, is marked in yellow. 4.22b) The position in the loading plane of the 1478 genes that show significant cell cycle periodicity. TOP2A is marked in yellow. 4.22c) The initial phase angle distribution of the marker genes. 4.22d) The initial assignment of cell cycle phase. 4.22e) The phase angle distribution of the marker genes after adjusting variances according to out-of-order assignments. 4.22f) Assignment after adjusting variances according to out-of-order assignments.

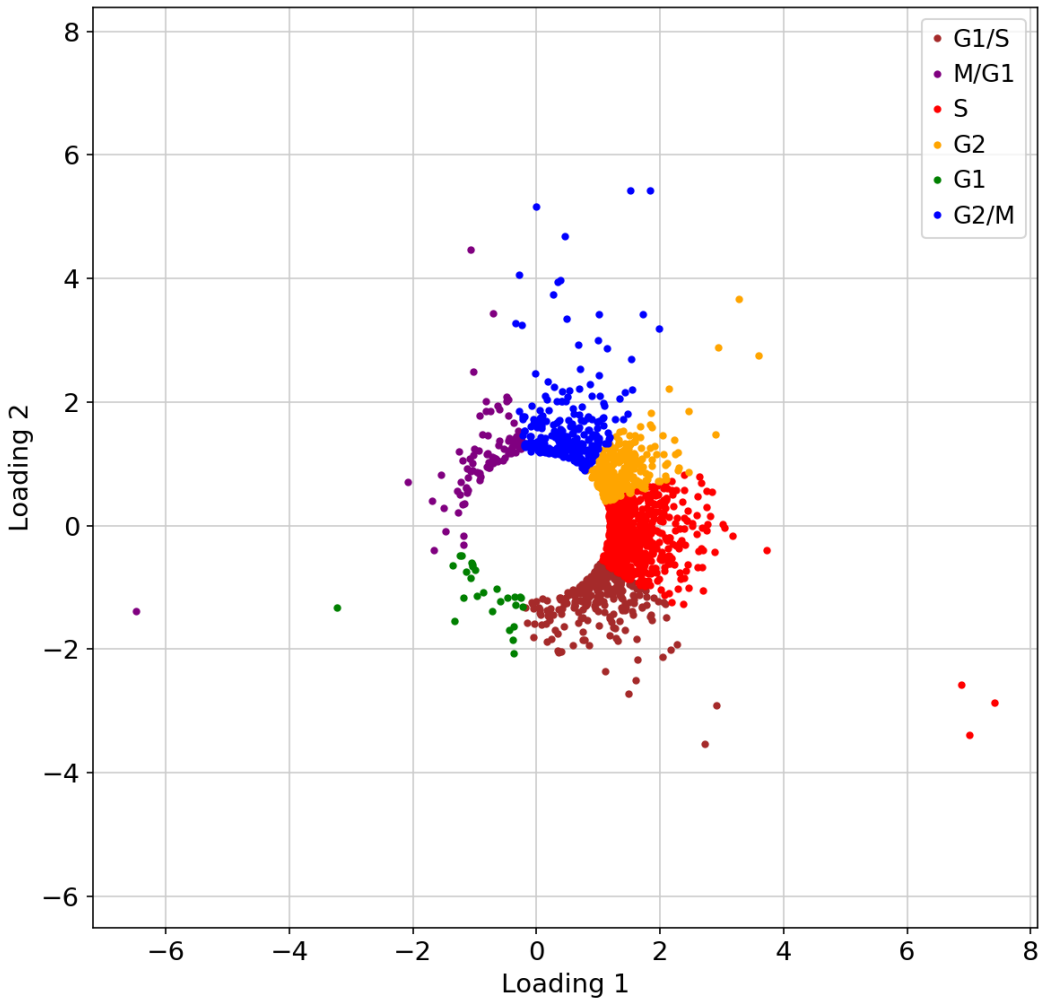
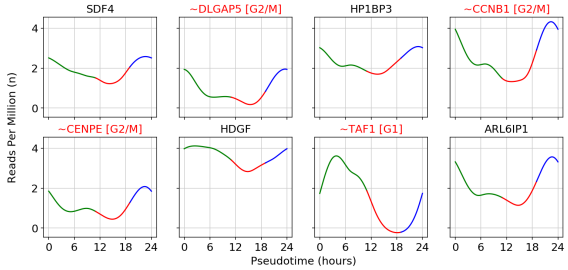


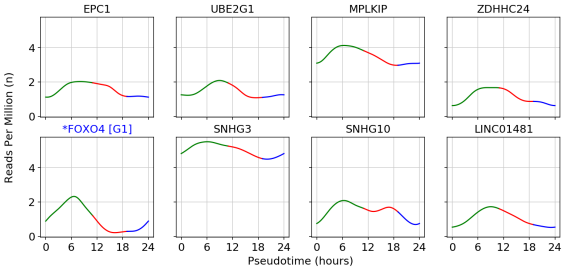
Figure 4.23: The final cell cycle phase classification for all significant genes in the Jurkat data set.

Table 4.16: Number of assigned genes to each of the cell cycle phases for the Jurkat cells.

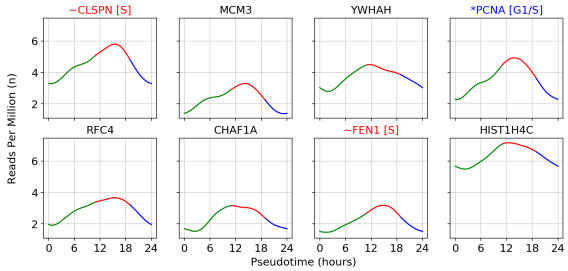
Cell Cycle Phase	Assigned Genes
S	598
G2	270
G1/S	249
G2/M	245
M/G1	89
G1	27



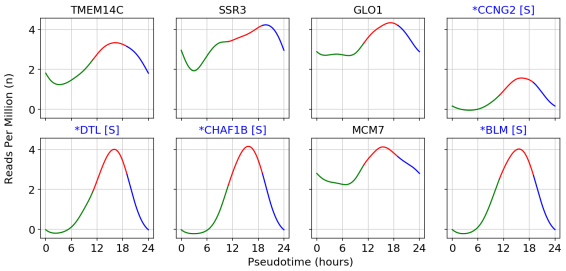
(a) Top 8 M/G1 genes.



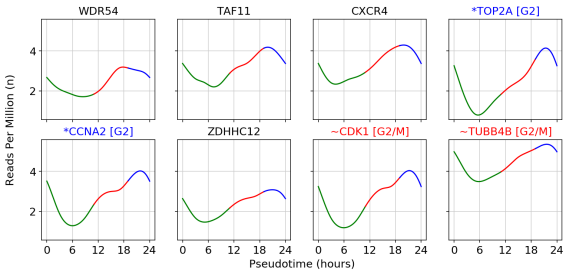
(b) Top 8 G1 genes.



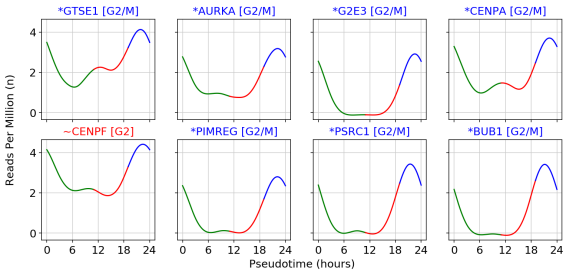
(c) Top 8 G1/S genes.



(d) Top 8 S genes.



(e) Top 8 G2 genes.



(f) Top 8 G2/M genes.

Figure 4.24: **Top genes for Jurkat cells.** Top 8 genes within each of the assigned phases, ranked by their length in the PLS loading plane. Genes prepended with a * marked in blue identify marker genes with their correct phase. Genes prepended with ∼ marked in red signify marker genes that are not classified as expected.

Functional Enrichment

The significant genes in the Jurkat data set are analyzed using funtional enrichment with gprofiler2 and a significance threshold of 0.05. We perform the analysis of the genes phase-by-phase and the 10 most significant terms are presented, where they exist, in Table 4.17-4.22. Extensive tables are presented in Appendix C.5.

Again we find G2/M related terms for the genes in M/G1. Genes in G1 have significant terms for G0-G1 regulation. In G1/S we find terms related to G1/S transisition together with DNA replication and DNA repair in addition to general cell cycle terms. For S we have RNA processing other terms related to RNA production together with general cell

cycle terms and one term for DNA replication. G2 has its majority of terms related to RNA production. For genes in G2/M, several terms for cell division are present together with general cell cycle terms.

Table 4.17: The most significant terms in the functional enrichment of Jurkat cells classified as M/G1 by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0051276	chromosome organization	1220	19	4.846e-03
2	GO:BP	GO:0018205	peptidyl-lysine modification	399	11	4.963e-03
3	GO:BP	GO:0033554	cellular response to stress	2052	25	7.494e-03
4	GO:BP	GO:0016569	covalent chromatin modification	480	11	2.877e-02
5	GO:BP	GO:0006325	chromatin organization	804	14	3.906e-02

g:Profiler (biit.cs.ut.ee/gprofiler)

Table 4.18: Significant terms in the functional enrichment of Jurkat cells classified as G1 by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0070317	negative regulation of G0 to G1 transition	41	3	1.454e-02
2	GO:BP	GO:0070316	regulation of G0 to G1 transition	45	3	1.930e-02
3	GO:BP	GO:0045023	G0 to G1 transition	48	3	2.348e-02

g:Profiler (biit.cs.ut.ee/gprofiler)

Table 4.19: The 10 most significant terms in the functional enrichment of Jurkat cells classified as G1/S by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0006259	DNA metabolic process	926	49	2.337e-14
2	GO:BP	GO:0006260	DNA replication	282	24	5.244e-10
3	GO:BP	GO:0006974	cellular response to DNA damage stimulus	849	39	7.072e-09
4	GO:BP	GO:0006281	DNA repair	545	31	7.265e-09
5	GO:BP	GO:0000082	G1/S transition of mitotic cell cycle	272	22	1.519e-08
6	GO:BP	GO:0006261	DNA-dependent DNA replication	153	17	2.740e-08
7	GO:BP	GO:0044843	cell cycle G1/S phase transition	292	22	6.146e-08
8	GO:BP	GO:0051276	chromosome organization	1220	45	2.037e-07
9	GO:BP	GO:1903047	mitotic cell cycle process	871	36	1.018e-06
10	GO:BP	GO:0007049	cell cycle	1850	56	1.308e-06

g:Profiler (biit.cs.ut.ee/gprofiler)

Table 4.20: The 10 most significant terms in the functional enrichment of Jurkat cells classified as S by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0006396	RNA processing	925	81	7.111e-15
2	GO:BP	GO:0043933	protein-containing complex subunit organization	2212	129	1.215e-10
3	GO:BP	GO:0016071	mRNA metabolic process	852	69	1.439e-10
4	GO:BP	GO:0008380	RNA splicing	431	46	2.210e-10
5	GO:BP	GO:0006397	mRNA processing	512	49	2.102e-09
6	GO:BP	GO:0006260	DNA replication	282	35	2.942e-09
7	GO:BP	GO:1903047	mitotic cell cycle process	871	66	1.100e-08
8	GO:BP	GO:0044770	cell cycle phase transition	624	53	2.481e-08
9	GO:BP	GO:0022402	cell cycle process	1383	88	3.458e-08
10	GO:BP	GO:0044772	mitotic cell cycle phase transition	581	50	6.166e-08

g:Profiler (biit.cs.ut.ee/gprofiler)

Table 4.21: The 10 most significant terms in the functional enrichment of Jurkat cells classified as G2 by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0000377	RNA splicing, via transesterification reactions with bulged adenosine as nucleophile	342	20	1.416e-04
2	GO:BP	GO:0000398	mRNA splicing, via spliceosome	342	20	1.416e-04
3	GO:BP	GO:0043933	protein-containing complex subunit organization	2212	61	1.603e-04
4	GO:BP	GO:0000375	RNA splicing, via transesterification reactions	345	20	1.633e-04
5	GO:BP	GO:0008380	RNA splicing	431	22	3.350e-04
6	GO:BP	GO:0016032	viral process	823	31	8.964e-04
7	GO:BP	GO:0065003	protein-containing complex assembly	1897	53	9.763e-04
8	GO:BP	GO:0006396	RNA processing	925	33	1.251e-03
9	GO:BP	GO:0044403	symbiont process	886	32	1.445e-03
10	GO:BP	GO:0044419	interspecies interaction between organisms	933	33	1.515e-03

g:Profiler (biit.cs.ut.ee/gprofiler)

Table 4.22: The 10 most significant terms in the functional enrichment of Jurkat cells classified as G2/M by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0000278	mitotic cell cycle	1015	56	1.149e-17
2	GO:BP	GO:0007049	cell cycle	1850	75	4.824e-17
3	GO:BP	GO:0051301	cell division	605	43	7.643e-17
4	GO:BP	GO:1903047	mitotic cell cycle process	871	49	2.408e-15
5	GO:BP	GO:0140014	mitotic nuclear division	284	26	1.006e-11
6	GO:BP	GO:0000280	nuclear division	427	30	8.682e-11
7	GO:BP	GO:0007017	microtubule-based process	745	39	1.424e-10
8	GO:BP	GO:0048285	organelle fission	469	31	1.633e-10
9	GO:BP	GO:0022402	cell cycle process	1383	54	2.087e-10
10	GO:BP	GO:0051726	regulation of cell cycle	1213	50	2.780e-10

g:Profiler (biit.cs.ut.ee/gprofiler)

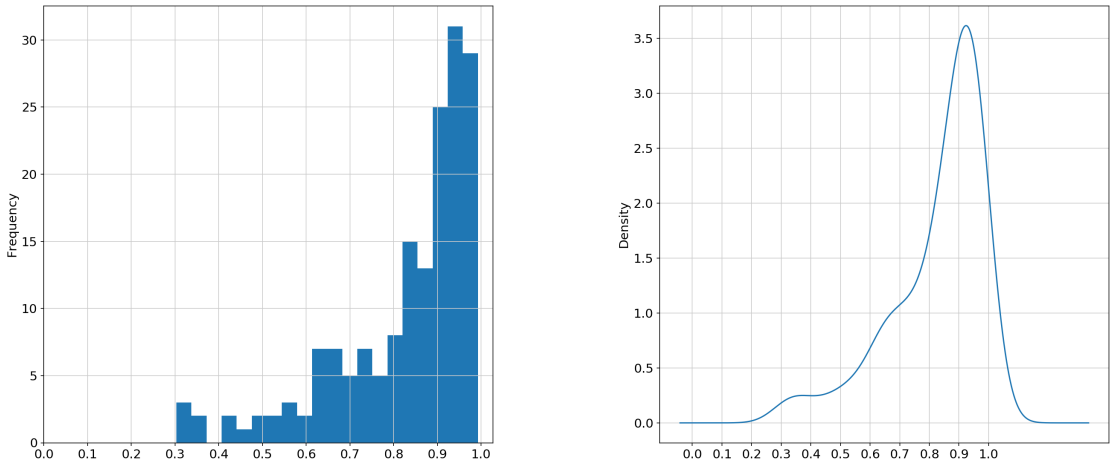
4.8 Aggregated Results

We will now analyze the combined results from the three data sets to identify genes with consistent expression profiles in the HaCat, 293t and Jurkat cells. We find 169 significant cell cycle periodic genes that are present in all three data sets. The mean of the upper triangular correlation matrix is found for each of the genes and we apply a filter to the genes based on this. The mean peak times of the common genes will be calculated from the spline models and used to estimate the genes phase.

The frequency plot and kernel density estimate for the correlation factor are presented in Figure 4.25. We use a cut-off to filter the bi-modal distribution observed. This corresponds to a correlation factor of 0.7. By inspection, the gene with the lowest correlation factor after filtering yields satisfactory results. After filtering, we are left with 138 genes that have consistent expression profiles in all the data sets. Of these, 68 are marker genes, which gives us 70 new candidate genes. The set of new candidate genes is further divided into a list for genes with a mean peak time in S ($11.0 \leq \mu_{\text{peak}} < 19$) and one for G2/M ($19.0 \leq \mu_{\text{peak}}$). None of the present genes have a peaktime in G1 (below 11.0). We find that 34 of the 70 are assigned to S and 36 to G2/M.

The gene expression profiles for the top eight marker genes, new candidate genes in S and new candidate genes in G2/M are presented in Figure 4.26-4.28. For the top eight genes in S, we see that many of the genes have different characteristics between the data sets, but that the trends are mostly consistent for all the expression profiles.

The top 15 genes in the marker genes, S and G2/M are listed in Table 4.23-4.25. The full tables are provided in Appendix C.6.



(a) (b)

Figure 4.25: **Correlation factor.** 4.25a) The frequency plot for the correlation factors of the common genes for the three data sets. 4.25b) Kernel density estimate for the correlation factor of the common genes for the three data sets.

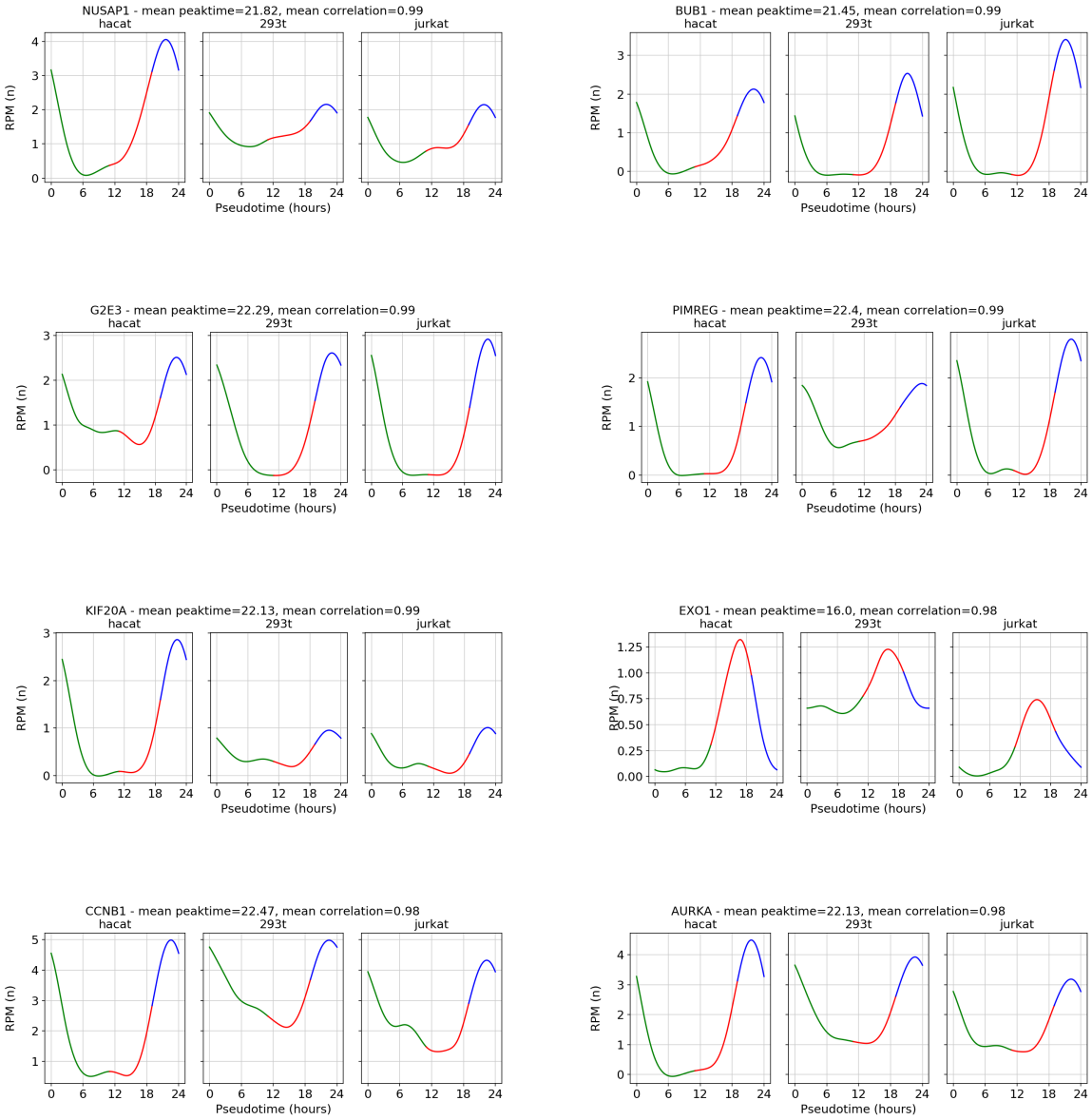


Figure 4.26: Top 8 marker genes ranked by mean correlation across datasets.

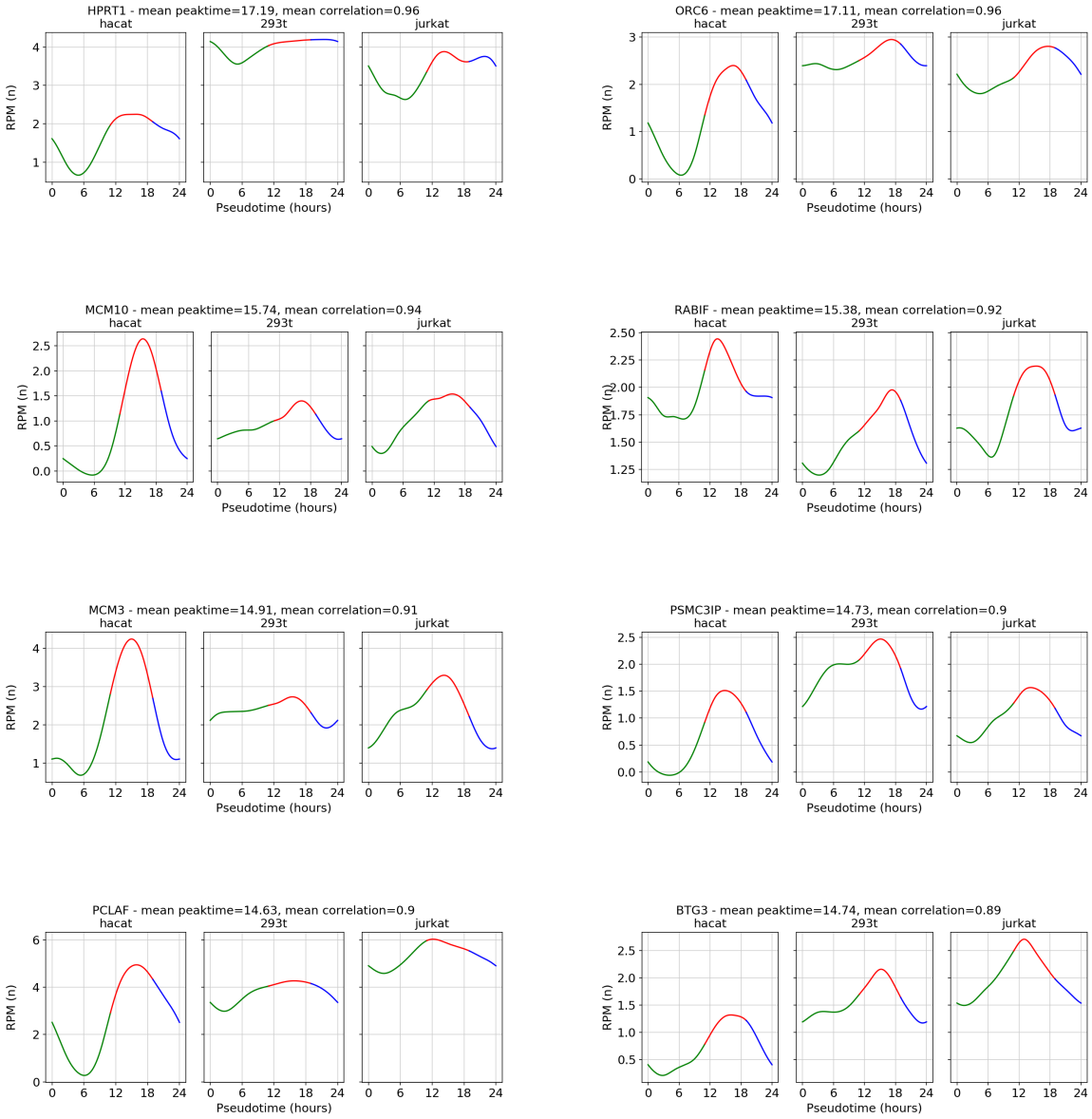


Figure 4.27: Top 8 new candidate genes classified as S by mean peaktime across datasets, ranked by mean correlation across datasets.

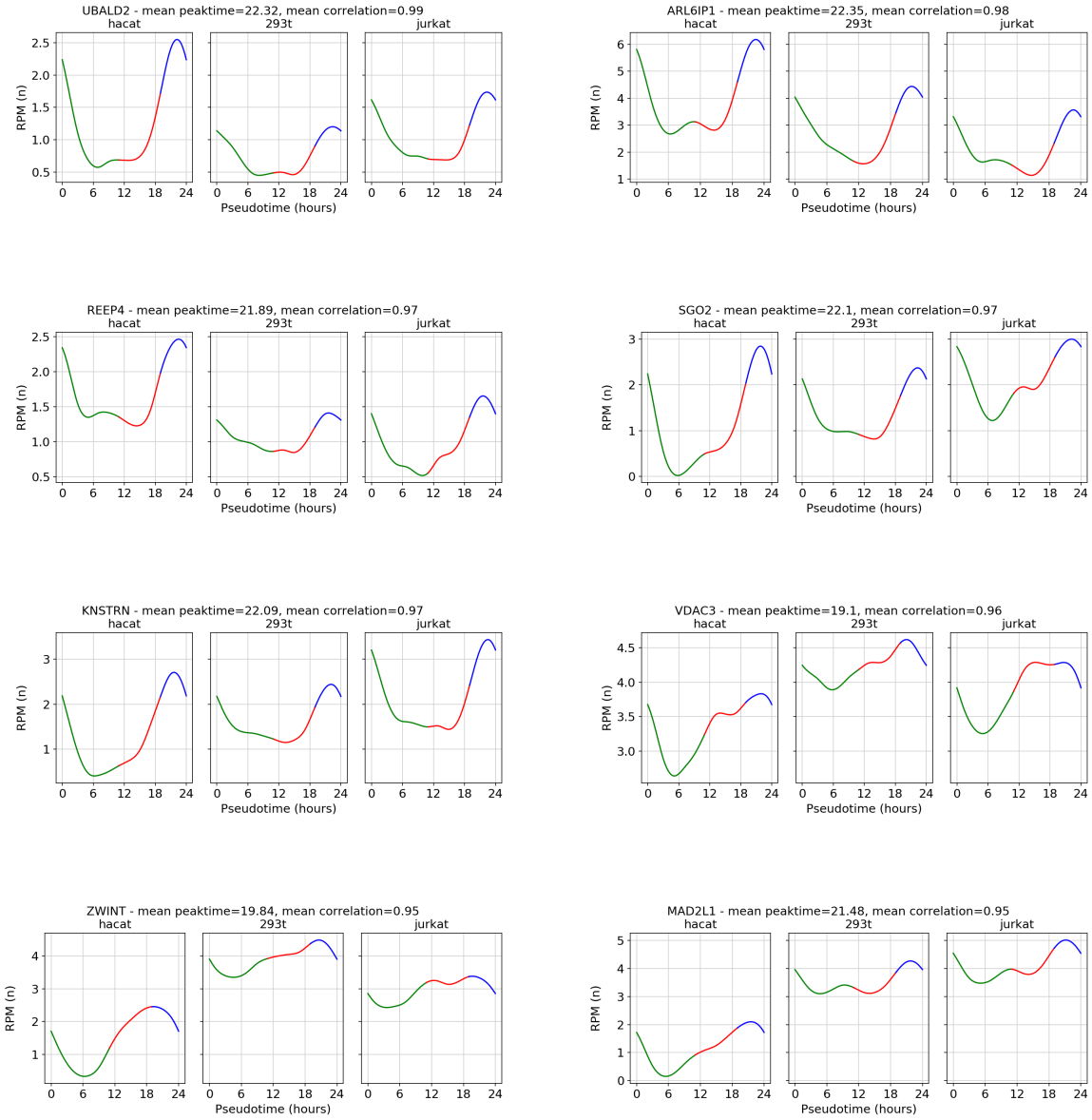


Figure 4.28: Top 8 new candidate genes classified as G2/M by mean peaktime across datasets, ranked by mean correlation across datasets.

Table 4.23: Top 15 marker genes ranked by correlation across datasets.

Gene Symbol	Mean Loading	Mean Correlation	Mean Peaktime	Rank
NUSAP1	3.58	0.99	21.82	1
BUB1	4.22	0.99	21.45	2
G2E3	4.30	0.99	22.29	3
PIMREG	3.58	0.99	22.40	4
KIF20A	2.33	0.99	22.13	5
EXO1	1.35	0.98	16.00	7
CCNB1	5.51	0.98	22.47	8
AURKA	5.25	0.98	22.13	10
NDC80	3.32	0.98	21.73	11
MCM6	2.97	0.98	15.18	12
PLK1	4.65	0.98	22.23	13
HMMR	3.83	0.98	21.90	14
NUF2	3.11	0.97	22.05	15
CCNE2	3.53	0.97	15.71	17
CENPF	5.56	0.97	22.31	20

Table 4.24: Top 15 new candidate genes classified as S by mean peaktime across datasets ranked by mean correlation across datasets.

Gene Symbol	Mean Loading	Mean Correlation	Mean Peaktime	Rank
HPRT1	1.58	0.96	17.19	25
ORC6	2.12	0.96	17.11	29
MCM10	2.48	0.94	15.74	50
RABIF	1.28	0.92	15.38	62
MCM3	3.16	0.91	14.91	68
PSMC3IP	2.13	0.90	14.73	72
PCLAF	4.01	0.90	14.63	76
BTG3	1.70	0.89	14.74	81
VRK1	1.90	0.89	18.55	82
ACD	1.82	0.89	16.14	83
UBE2T	2.91	0.89	18.59	84
CHAF1A	3.44	0.88	13.51	89
NUDT8	2.06	0.88	13.82	90
UROS	1.34	0.88	14.39	91
DNAJC9	2.62	0.88	15.74	92

Table 4.25: Top 15 new candidate genes classified as G2/M by mean peaktime across datasets ranked by mean correlation across datasets.

Gene Symbol	Mean Loading	Mean Correlation	Mean Peaktime	Rank
UBALD2	2.17	0.99	22.32	6
ARL6IP1	4.37	0.98	22.35	9
REEP4	1.58	0.97	21.89	16
SGO2	3.10	0.97	22.10	18
KNSTRN	2.92	0.97	22.09	19
VDAC3	1.61	0.96	19.10	26
ZWINT	2.35	0.95	19.84	34
MAD2L1	2.10	0.95	21.48	37
KPNA2	3.45	0.95	22.13	41
NAA50	1.70	0.94	19.38	47
DEPDC1B	1.83	0.93	21.56	56
SPC25	2.10	0.93	21.65	58
ASPM	4.01	0.91	22.47	63
PRR11	2.91	0.91	22.18	64
BUB3	2.62	0.91	21.38	66

4.8.1 Functional Enrichment

Functional enrichment is performed by using gprofiler2. We perform the queries on the list of new candidate genes assigned to S and to G2/M. All the significant terms are presented in Table 4.26 and in Table 4.27.

Here both phases show significant terms related to their functions in the cell cycle together with generic cell cycle terms. For S, we have DNA replication related terms. Pyrimidine nucleoside triphosphate metabolic process is related to production of DNA compounds. For G2/M, chromatid and chromosome segregation terms are the most significant. We also have terms related to cell division. This indicates that several of our new candidate genes have known cell cycle functions.

Table 4.26: Significant terms in the functional enrichment of the new candidate genes classified as S by mean peaktime across data sets.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0006260	DNA replication	282	10	1.009e-07
2	GO:BP	GO:0006259	DNA metabolic process	926	13	7.207e-06
3	GO:BP	GO:0007049	cell cycle	1850	15	5.003e-04
4	GO:BP	GO:0006261	DNA-dependent DNA replication	153	6	5.937e-04
5	GO:BP	GO:0022402	cell cycle process	1383	13	8.240e-04
6	GO:BP	GO:0006270	DNA replication initiation	41	4	1.680e-03
7	GO:BP	GO:0009147	pyrimidine nucleoside triphosphate metabolic process	22	3	1.554e-02
8	GO:BP	GO:0000278	mitotic cell cycle	1015	10	1.710e-02
9	GO:BP	GO:1903047	mitotic cell cycle process	871	9	3.646e-02

g:Profiler (biit.cs.ut.ee/gprofiler)

Table 4.27: Significant terms in the functional enrichment of the new candidate genes classified as G2/M by mean peaktime across data sets.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0000819	sister chromatid segregation	185	8	3.349e-06
2	GO:BP	GO:0007059	chromosome segregation	314	9	1.042e-05
3	GO:BP	GO:0007049	cell cycle	1850	17	1.462e-05
4	GO:BP	GO:0051301	cell division	605	11	1.778e-05
5	GO:BP	GO:0000070	mitotic sister chromatid segregation	153	7	2.659e-05
6	GO:BP	GO:0000278	mitotic cell cycle	1015	13	3.734e-05
7	GO:BP	GO:0098813	nuclear chromosome segregation	261	8	4.978e-05
8	GO:BP	GO:0000280	nuclear division	427	9	1.479e-04
9	GO:BP	GO:0022402	cell cycle process	1383	14	1.712e-04
10	GO:BP	GO:0051276	chromosome organization	1220	13	3.263e-04
11	GO:BP	GO:0048285	organelle fission	469	9	3.285e-04
12	GO:BP	GO:1903047	mitotic cell cycle process	871	11	7.274e-04
13	GO:BP	GO:0140014	mitotic nuclear division	284	7	1.808e-03
14	GO:BP	GO:0070601	centromeric sister chromatid cohesion	12	3	2.761e-03
15	GO:BP	GO:0007062	sister chromatid cohesion	58	4	8.719e-03

g:Profiler (biit.cs.ut.ee/gprofiler)

Chapter 5

Discussion

5.1 Sequence Alignment

The gene expression quantifaction from STARsolo and CellRanger yield almost identical results in both the number of reads per biotype and the number of genes detected per biotype. By manipulating the paramterers of STASRsolo, we were able to reduce the amount of reads and genes mapped to pseudogenes significantly while the numbers changed little for protein coding reads and for genes. The decrease in reads and genes mapped to pseudogenes was reduced slightly by restricting the number of mismatched bases allowed. As expected, when the number of allowed multimapping alignments was reduced, pseudogenes decreased drastically. Hence, a strict multimapping policy for single cell alignments might be a good policy in single cell experiments. A draw back for CellRanger is the lack of support for altering the parameters to alignment procedures. Users have little to no freedom to tweak parameters according to their particular experiments.

In the benchmark metrics we also find favourable results for STARsolo. It was capable of producing identical output at a 11.54 speedup in runtime. We were able to improve the runtime for CellRanger by disabling the secondary analysis. This gave a speedup of 20.7%, but STARsolo still outperforms it by a factor of 9.57 in runtime. CellRanger also produce a high volume of I/O during execution. This volume is not present in the resulting output files of CellRanger. It consists of intermedite files that are used during

runtime. It is clear that STARsolo outperforms CellRanger on all the measured metrics in this benchmark, but many users may find the secondary analysis available through CellRanger useful.

5.2 Pseudotime Ordering

With the exception of the HaCat cells, there are some clear issues with distinct artefacts in the PCA that is the basis of the ordering. The observed structures does not seem to be related to the common nuisance factors like total number of reads or batch effects. By performing a PCA on the HaCat cells with a low input of marker genes, we were to reproduce the behavior. In spite of the visible effects, the method is still able to produce an ordering for the cells inn all three data sets.

Also in the comparison of the mean expression levels within each modelled phase, the HaCat data set shows the best performance. Here, 85% of the marker genes have their highest mean expression in the expected phase against 70% for 293t and 78% for Jurkat. The issue is mainly with predicting the G1 marker genes and for 293t and Jurkat, to some extent, S. For G2/M, all data sets show an almost perfect prediction with the expression levels. The exception being for 293t where two G2/M marker genes show higher mean expressions in other phases.

The weaker results in G1 and S might indicate some issues with classification of G1 and S cells, or they might indicate that the marker genes are not perfectly compatible with this type of experiment. Some of the marker genes show very little differences in variability in expresseion levels between the modelled phases. Broadly, we evaluate the overall performance of this method to be good enough for our purposes, with some issues that needs further research.

5.3 Gene Expression Modelling

The optimal smoothing parameter found for all data sets lie in the lower ranges ($p = 0.15$ for HaCat and 293t, $p = 0.25$ for Jurkat). This is a weak indication that our ordering method is able to produce close to smooth trajectories for the control points of the gene expression levels of our marker genes.

From the scatter plots of the gene expressions per cell, we could see that for several of the marker genes the expression values for the cells that have a non-zero expression do not change significantly throughout the ordering. In these genes, our method for producing the control points are still able to model variability through the phases of the ordering. This is because the density of cells that have zero expression varies through the time intervals in the ordering. Hence, when the means from the time intervals that are used as control points for the splines, we are able to model the variation in a way that is more conforming to the expected behaviour of a cyclic gene expression. At the same time, some marker genes in all data sets of genes that do show very distinct expression levels, exclusively in their respective phases.

We also note that for the scatter plots in 293t and Jurkat, we can observe some artefacts in the expression levels in the form stripes. This does affect the spline model in itself, but may point to some issues with the ordering. These stripes may very well be a direct effect from the patterns observed in the PCA in the two data sets.

5.4 Identification of Cell Cycle Periodic Genes

Our method is able to identify a similar amount of periodic gene expressions in all three data sets. We found 1239 for Hacat, 1295 for 293t and 1478 for Jurkat. These are at the same levels, or higher, than reported findings in previous synchronization experiments. For example, in [Pen+13] 1249 periodic expressions were found and 874 are reported in [Whi+02].

The method used for phase assignments mostly produce assignments that are in order with respect to the cell cycle. In all three data sets G2 and G2/M have very similar

distributions which leads to many of the marker genes in these two phases being miss-assigned to one or the other. Most phases show some miss-assignments of marker genes in almost all phases. For the 293t data set, the mean phase angle of the G1/S marker genes were between the mean phase angle of G2 and S. Hence, this assignment method failed to separate G1/S and S and the two phases merged into S. We also observe that there are issues with the assignments of M/G1 in all three data sets. Many of the expressions assigned to M/G1 show profiles more consistent with G2 or G2/M. This method for classification has some issues regarding boundaries between all cell cycle phases, but it is mostly able to predict a phase that fits with the observed expression profile.

All three data sets show different distributions in the number of assigned genes to each phase, but most of the periodic genes are classified to S, G2 or G2/M. The number of genes assigned to G1 or M/G1 are consistently among the lowest across all data sets.

5.5 Functional Enrichment

The biological process enrichment generally gives expected significant terms for S and G2/M in all three data sets. For S, most significant terms are related to DNA replication, DNA repair, chromosome organization. For G2/M, we typically see terms for cell division, nuclear division, organelle fission, chromatid and chromosome segregation and mitotic cell cycle. For the HaCat data set, significant terms in G2 are more similar to what we find in the enrichment for G2/M in the data sets, while for 293t and Jurkat we find significant terms for RNA splicing and RNA processing.

From our enrichment for biological processes, we see that our method is able to detect genes related to cell cycle and largely assign phases conforming to their biological functions. Some of the anomalies observed in our enrichment come from the boundary issues in phase assignment discussed in the results. This is most clear from the enrichment of M/G1. There is very little biology confirmed by the enrichment of G1 genes, but G1 is also the phase that generally has the fewest assignments in our experiment.

5.6 Aggregate Results

By requiring consistency in the expression profiles among the data sets analysed, we reduce the set of interest down to 138 genes. Because of our similarity requirement, we can confidently assign a phase by their modelled peaktime. This eliminates the phase boundary issues observed by the assignment by the phase probability density functions. The functional enrichment analysis of these sets yields fewer significant terms because we have less genes as input in our queries. The retrieved terms are strongly related to their respective phases without any noise from neighbouring phases. This indicates that many of our new candidate genes have known cell cycle functions. It also indicates that phase assignment by modelled peaktime from the smoothing cubic spline model may be a better strategy.

Chapter 6

Conclusion

In our analysis, we have shown that STARsolo and CellRanger produce almost identical results and that STARsolo does this in a highly effective manner. CellRanger provides functionality for quality reports and secondary analysis that does not exist in STARsolo. CellRanger is arguably more user friendly than STARsolo, but for many users the technicalities of STARsolo will be straightforward. The fast run time of STARsolo and its parameter flexibility is a huge benefit for developers that want to experiment with different parameters and investigate the effects on their data.

By using single cell RNA-seq data from HaCat, 293t and Jurkat cells, we have derived a model for identifying genes that show cyclic expression profiles with period matching the cell cycle. We use dimensional reduction through Principal Component Analysis with marker genes as input variables. By investigating expression levels in the marker genes with a scoring scheme, cells that are proliferating are identified. Each cell is assigned an order with respect to the cell cycle by using the angles in the principal component plane. The strategy is shown to be sensitive for a low input of marker genes, producing clustering artefacts in the Principal Component Analysis. The method is able to produce a ordering of the cells in spite of the visible artefacts, implying that the method is robust. We have developed method using smoothing cubic splines to model the gene expression levels through the ordering as a function of cyclic time. The method is able to produce smooth expression profiles that are periodic in pseudotime.

Partial Least Squares regression is used to identify genes that show cyclic expression profile with period matching the cell cycle. Here, the developed method uses the smoothing cubic spline model of the gene expression profiles as denoiser. The smoothing cubic spline models of each gene expression is subsampled on hourly intervals through the pseudotime and used as the input observation for the Partial Least Squares regression. We use a sine and cosine function with period equal to the cell cycle as the response variable. The sine and cosine are evaluated at the timepoints corresponding to the observations used for the smoothing cubic smoothing spline model. The genes are analyzed in the plane of the scaled loading weights retrieved from the Partial Least Squares regression model. We determine a threshold for the distance from the origin in this plane corresponding to a False Discovery Rate of 5% by performing a permutation analysis on the response variables. By obtaining the phase angle probability density function of the marker genes in the plane of the scaled loading weights, we assign each gene above the threshold a their most probable phase. This method of phase assignment has been shown in this study to produce miss-assignments of marker genes to neighboring phases in all three data sets.

The method is able to determine cell cycle periodic genes in the same numbers as has been previously reported from bulk RNA sequencing synchronization experiments. We find 1239 cell cycle periodic genes for the HaCat cells, 1295 for 293t and 1478 for the Jurkat cells corresponding to a p-value of 5%. We perform a functional enrichment analysis for biological processes on the cell cycle periodic genes in each data set for each of the assigned phases. At a significance threshold corresponding to $p = 5\%$ genes assigned to S and G2/M generally retrieves significant terms related phase-specific processes across all data sets. With few exceptions, the enrichment analysis retrieves cell cycle related terms for most phases in all data sets. M/G1 and G2 frequently have significant terms from G2/M as an effect of the miss-assignments from phase angle probability density function. The abundance of significant terms related to the cell cycle shows that our method is able to identify genes with known cell cycle functions.

A set of new candidate genes are produced by performing a correlation analysis on the gene expression profiles from the smoothing cubic spline model on significant cell cycle periodic genes that are present in all three data sets. A requirement for similarity is found by analyzing the kernel density estimate distribution of the mean upper triangular

correlation matrix for each of the genes present in all three data sets. We find a threshold for the mean correlation factor by filtering the bi-modal distribution observed. In total 138 genes that are above the mean correlation threshold, of which 68 are among our marker genes. The 70 new candidate genes are assigned phase by the mean peaktime obtained from the three smoothing cubic spline gene expression models. We assign 34 genes to S and 36 to G2/M. Functional enrichment analysis is performed on the set of new candidate genes in each of the phases. Here, all significant terms are either general terms or specific to the functions related to the respective cycles.

In this experiment we have derived a method for identifying cell cycle periodic genes within a significance threshold using single cell RNA-seq data. The method for obtaining a smoothing cubic spline model for gene expression profiles through pseudotime has proven to be robust across all three data sets. Two different strategies for assigning cell cycle phase to genes with significant cell cycle periodic expressions have been developed. In this process, we have developed a library of utility routines. These are published and are freely available together with working examples implemented in jupyter notebooks and data. All significant cell cycle periodic genes HaCat, 293t and Jurkat cells are published together with the set of genes present in all three data sets.

Chapter 7

Future work

In our experiments, we saw that the pseudotime ordering based on PCA was sensitive to a low input of marker genes. By utilizing other techniques, we can omitt the requirement of marker genes completely. Single cell RNA-seq is a field of rapid technological increments and new software emerge weekly. During the work on this experiment software such has Cyclum[Lia+19] has emerged. Cyclum uses a machine learning approach that is claimed to be able to efficiently infer latent cell-cycle trajectories from scRNA-seq gene expression data by characterizing circular trajectories in the high-dimensional gene expression space. Cyclum is implemented in TensorFlow[Aba+16] which is a portable computing environment that can easily be run on Graphical Processing Units.

Some of the steps in the methods from the utility library developed for this experiment should also be automated. This should be a fairly straight forward task. Most parts of the method run automatically with parameters that are dynamically set based on output from previous steps. The aligning of G1 to start at order zero after the phase angle assignment in the principal component has only been semi-automated. The user must specify how many orders should be shifted and in which direction. The user must also specify if the order should be reversed. The cell cycle phase assignment of the significant cell cycle periodic genes are also semi-automated. If full automation is achieved, the workflow can be implemented as a pipeline. This allows for easily acquiring of significant cell cycle periodic gene expressions from more data sets.

In this experiment, the results from each of the data sets was only analyzed using functional enrichment. This confirmed that we were able to retrieve known cell cycle genes by our method, but the exciting potential lies in analyzing the gene expressions that are not known to have particular functions. These should be identified and further analyzed.

When analyzing the genes that were present in all three data sets, we saw the mean peaktime performed better than assignment by phase angle probability density function. A new analysis should be done within each of three data sets to determine if this is more stable approach. The overlap analysis has the strength of using the mean peaktime across all three data sets for each gene, but in our analysis where we compared the top eight genes in each phase in the data sets, we saw many examples on genes that were classified to a phase but had the distinct characteristics from another phase in its expression profiles.

A similar correlation analysis as was done in this experiment could also be repeated by analyzing the overlapping genes from the data sets in pairs. This would enable us to expand our list of candidate genes. These candidate genes can in turn be used as marker genes in new experiments to, both to test their validity and to investigate if they will yield better results.

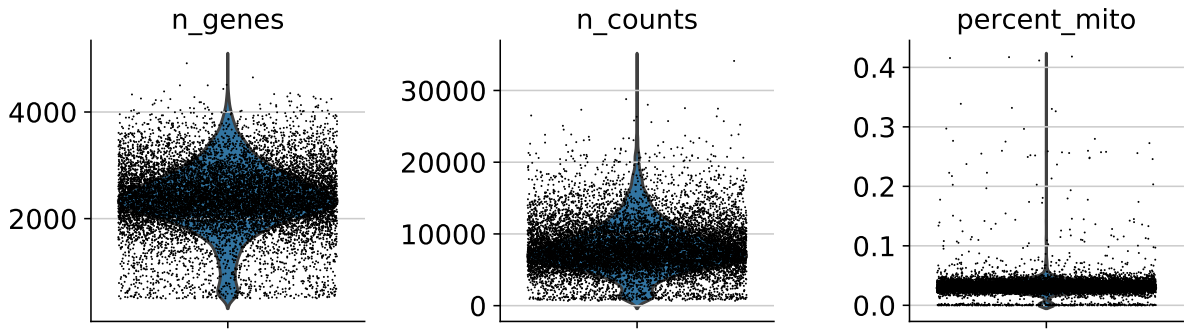
Appendix A

Extra Material

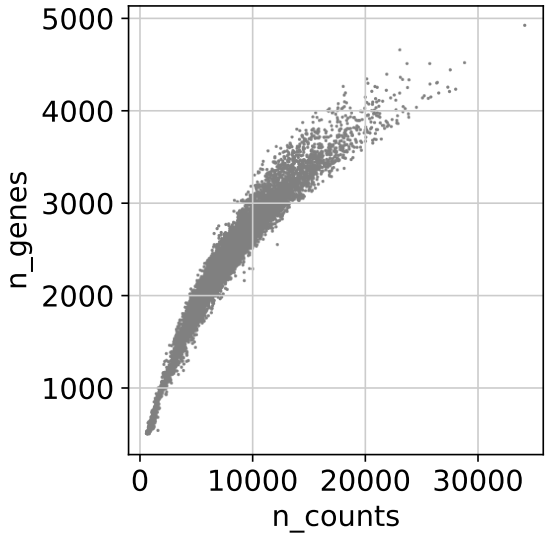
A.1 Pre-Processing of 293t cells

After initial filtering, we have 11978 viable cells and 18957 detected genes. Figure A.1 shows kernel density estimates for detected genes, number of reads and percent mitochondrial reads together with scatter plots for number of reads against detected genes and number of reads against percent mitochondrial read. Figure A.2 shows the same plots after applying a hard filter on 2500 reads and 10% mitochondrial reads.

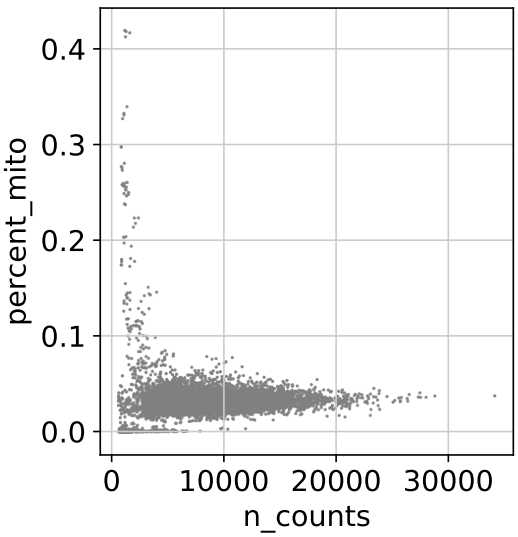
We are left with 11447 cells. The data is RPM normalized and transformed with log(RPM+1). We find that the data set has 5430 genes that are classified as highly variable. We use the data set with these genes as input to a PCA. In Figure A.3a we see that PC1 is correlated to a high number of reads and a high number of detected genes. There are no clear patterns for the percent of mitochondrial reads nor are there any visible batch effects. We perform a regression on the number of reads per cell and perform a new projection. The result can be seen in A.3b. Now, there are no clear patterns for the number of reads or for the number of detected genes along any of the principal components.



(a)

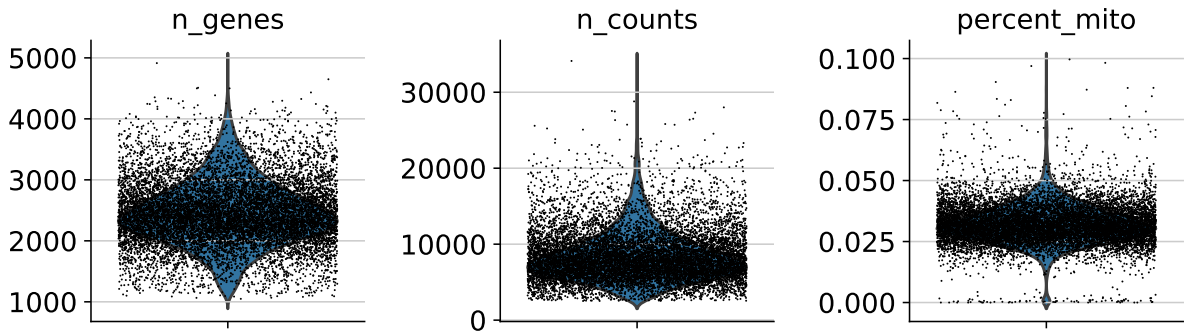


(b)

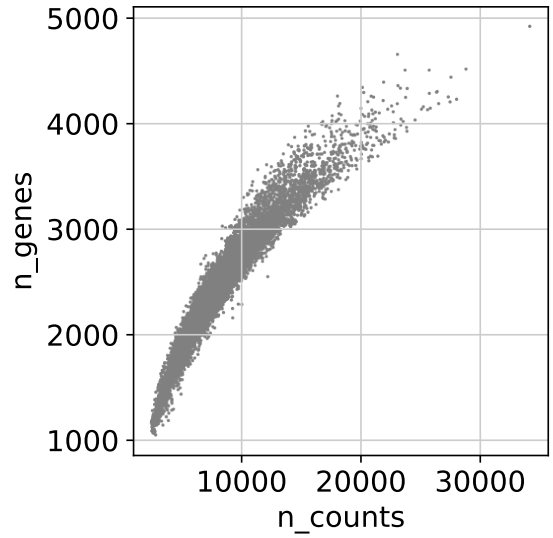


(c)

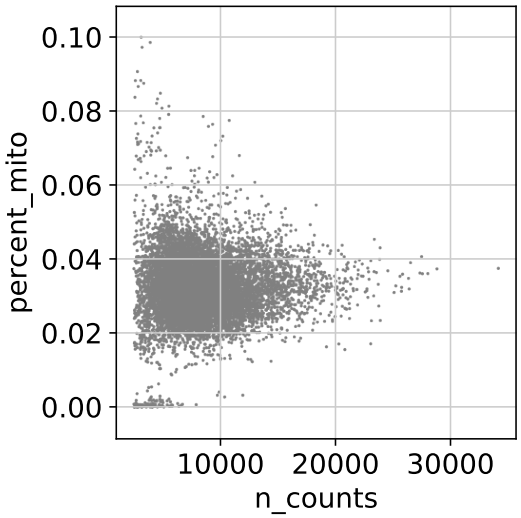
Figure A.1: **Pre-processing of 293t cells.** A.1a) Kernel density estimates for number of genes detected per cell (n_genes), number of reads per cell (n_counts) and percent mitochondrial reads per cell (percent_mito). A.1b) Number of genes detected versus number of reads per cell. A.1c) Percent mitochondrial reads versus number of reads per cell.



(a)

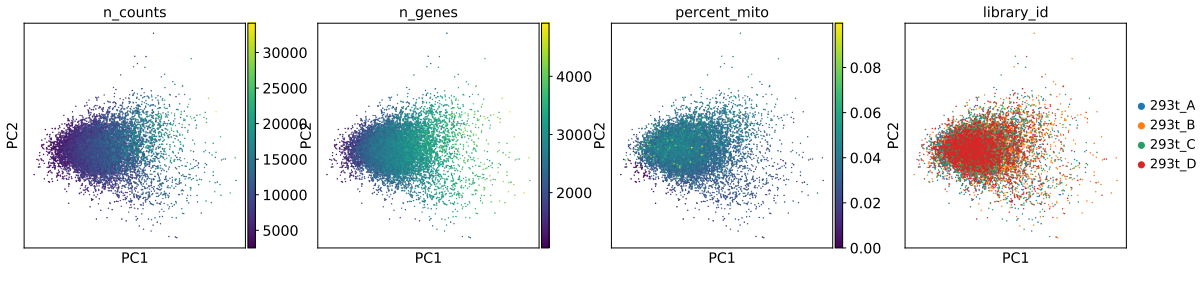


(b)

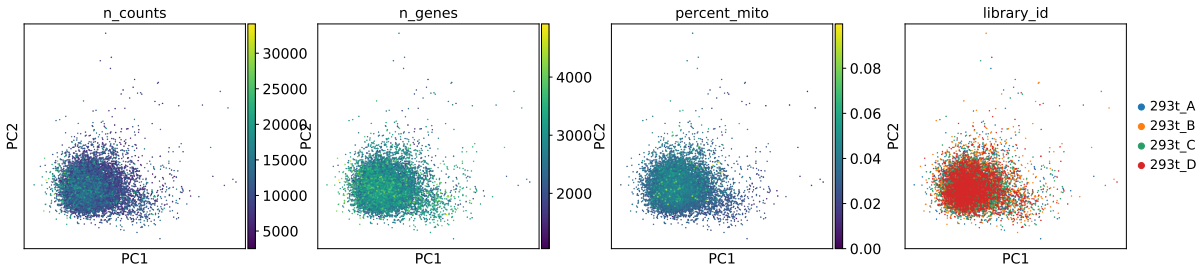


(c)

Figure A.2: **Pre-processing of 293t cells.** A.1a) Kernel density estimates after filtering for number of genes detected per cell (n_genes), number of reads per cell (n_counts) and percent mitochondrial reads per cell (percent_mito). A.1b) Number of genes detected versus number of reads per cell after filtering. A.1c) Percent mitochondrial reads versus number of reads per cell after filtering.



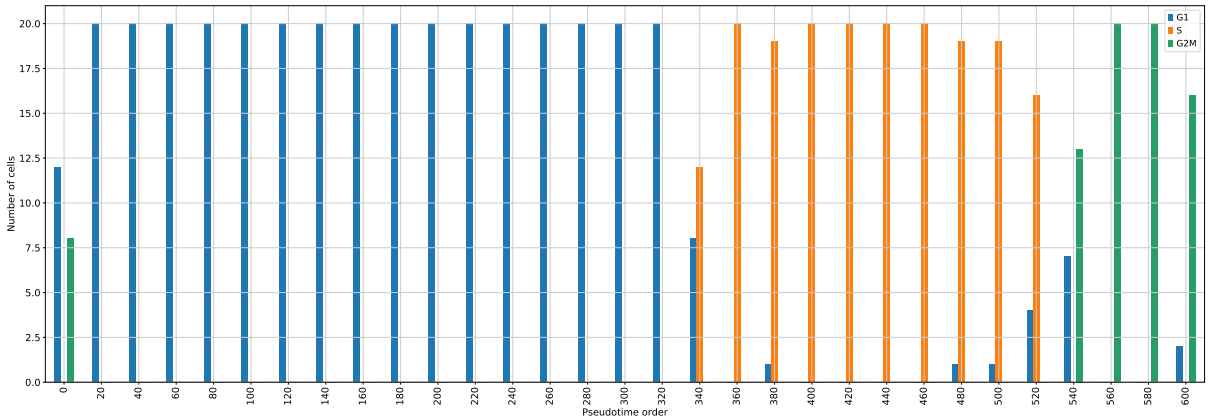
(a)



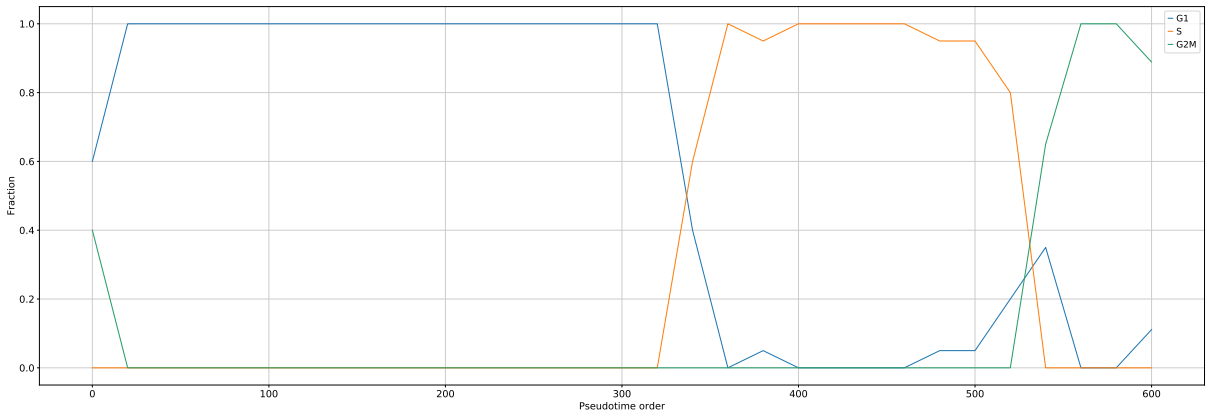
(b)

Figure A.3: **PCA analysis of 293t cells.** Projection of the data set into the PC1,PC2 plane. The cells are colored by number of reads (n_counts), the number of detected genes (n_genes), fraction of mitochondrial reads (percent_mito) and batch id (library_id). A.3a) Initial PCA projection of the 293t data set. A.3b) PCA projection after regressing out number of reads.

A.2 Pseudotime Ordering of 293t cells



(a)



(b)

Figure A.4: **Ordering of 293t cells.** The pseudotime ordering after rearranging so that the G2/M-G1 boundary is at zero. We find that the phase boundaries are located at order 337 for G1-S and 545 for S-G2/M. A.4a) Number of cells per phase for each bin of size 20 through the pseudotime ordering of the cells. A.4b) Linear curves representing the fraction of cells in each phase along the psedotime ordering.

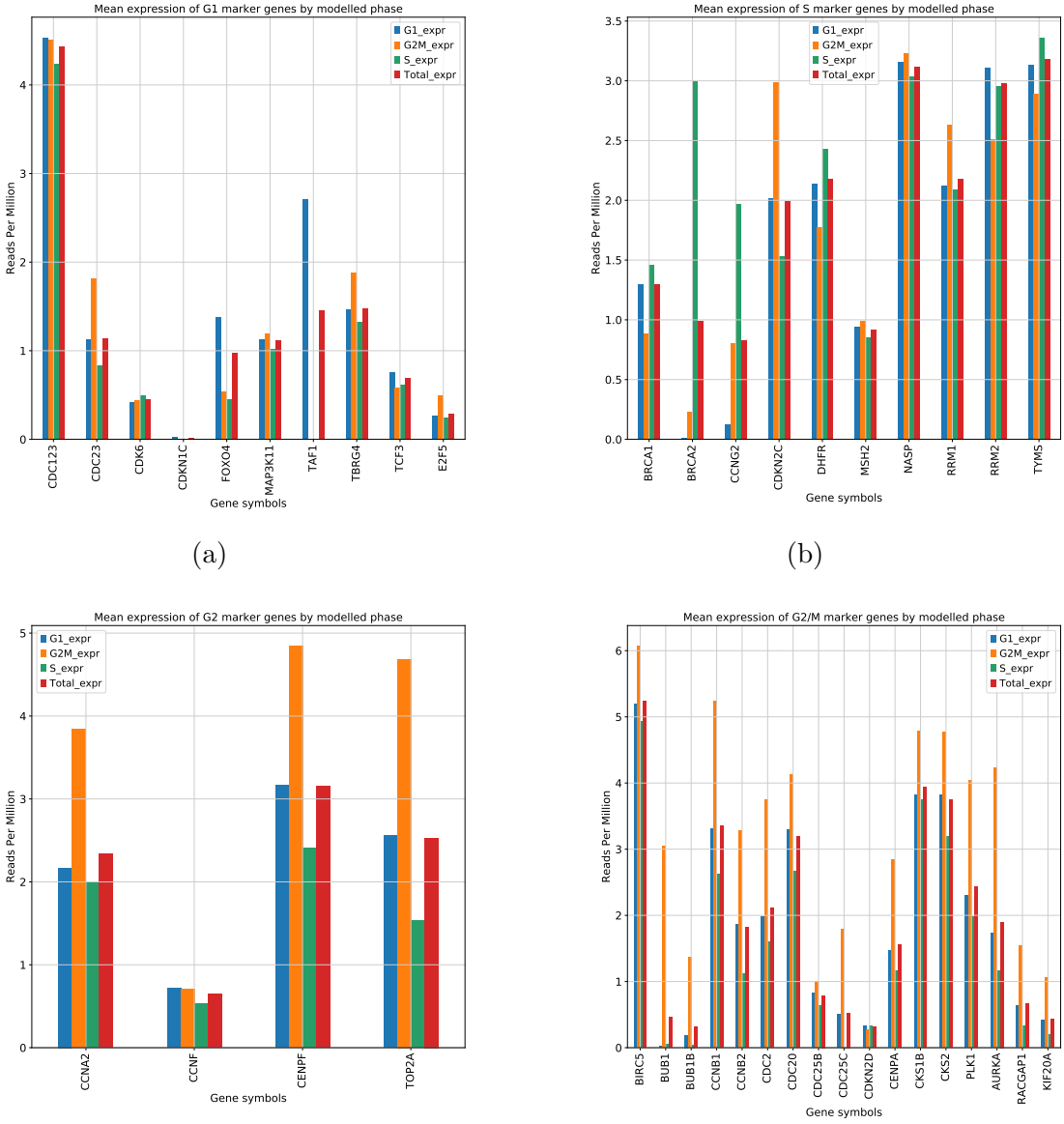


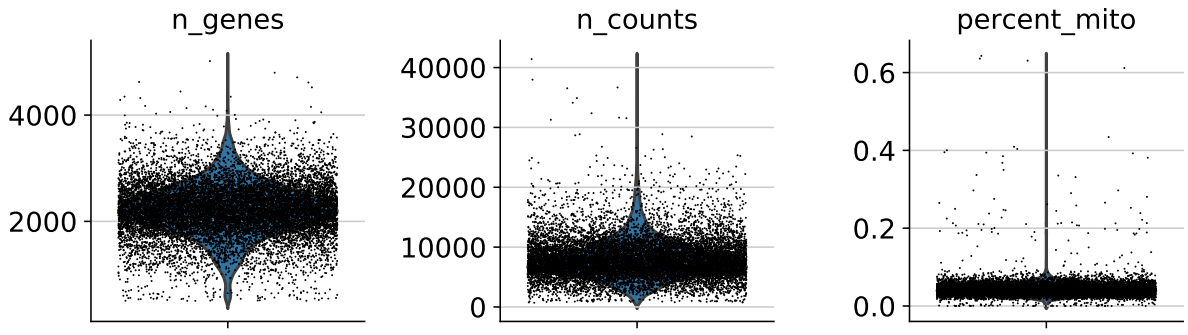
Figure A.5: **Mean expressions of 293t cells.** Mean expression of marker genes in the modelled phases along the pseudotime ordering. We use the full data set of all 18975 RPM normalized genes in the ordering. Roughly $\sim 70\%$ of the marker have their highest mean expression in the expected modelled phase. A.5a) Mean expressions for G1 marker genes. A.5b) Mean expressions for S marker genes. A.5c) Mean expressions for G2 marker genes. A.5d) Mean expressions for G2/M marker genes.

A.3 Pre-Processing of Jurkat cells

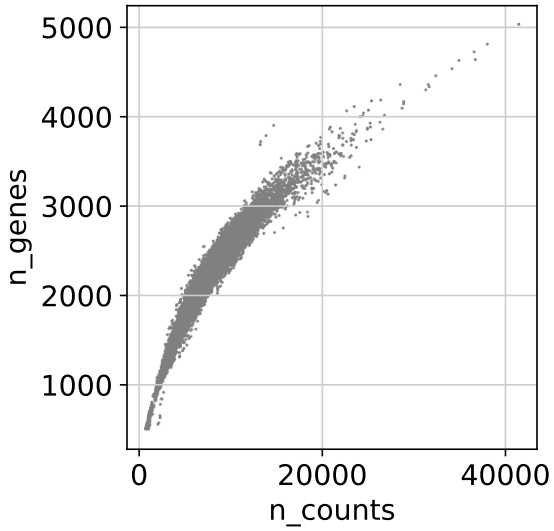
After initial filtering, we have 13154 viable cells and 17867 detected genes. Figure A.6 shows kernel density estimates for detected genes, number of reads and percent mito-

chondrial reads together with scatter plots for number of reads against detected genes and number of reads against percent mitochondrial read. Figure A.7 shows the same plots after applying a hard filter on 3000 reads and 10% mitochondrial reads.

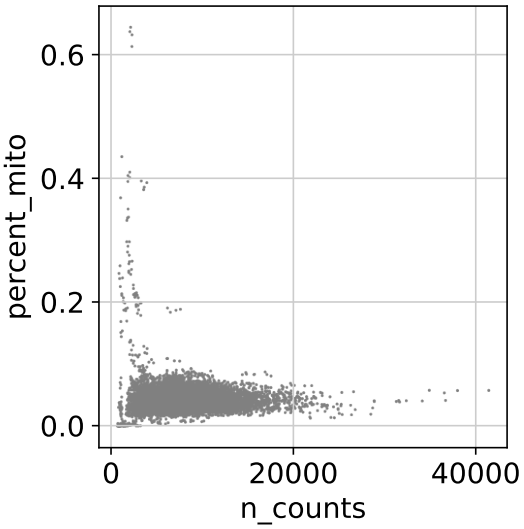
We are left with 12681 cells. The data is RPM normalized and transformed with log(RPM+1). We find that the data set has 5246 genes that are classified as highly variable. We use the data set with these genes as input to a PCA. In Figure A.8a we see that PC1 is correlated to a high number of reads and a high number of detected genes. There are no clear patterns for the percent of mitochondrial reads nor are there any visible batch effects. We perform a regression on the number of reads per cell and perform a new projection. The result can be seen in A.8b. Now, there are no clear patterns for the number of reads or for the number of detected genes along any of the principal components.



(a)

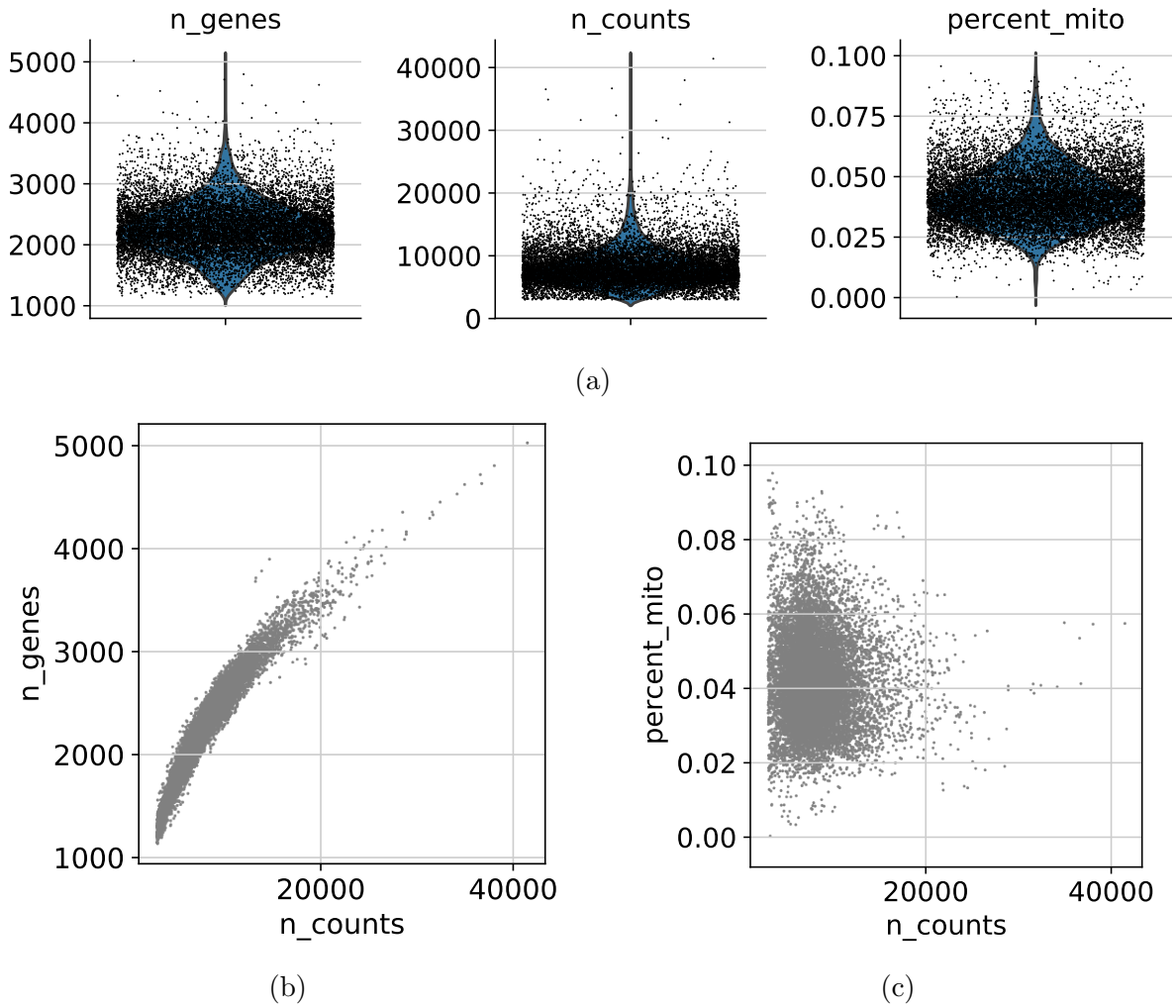


(b)



(c)

Figure A.6: **Pre-processing of Jurkat cells.** A.6a) Kernel density estimates for number of genes detected per cell (n_genes), number of reads per cell (n_counts) and percent mitochondrial reads per cell (percent_mito). A.6b) Number of genes detected versus number of reads per cell. A.6c) Percent mitochondrial reads versus number of reads per cell.

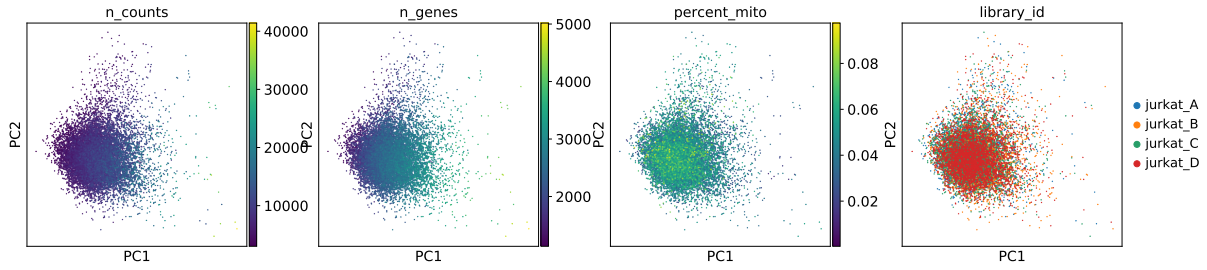


(a)

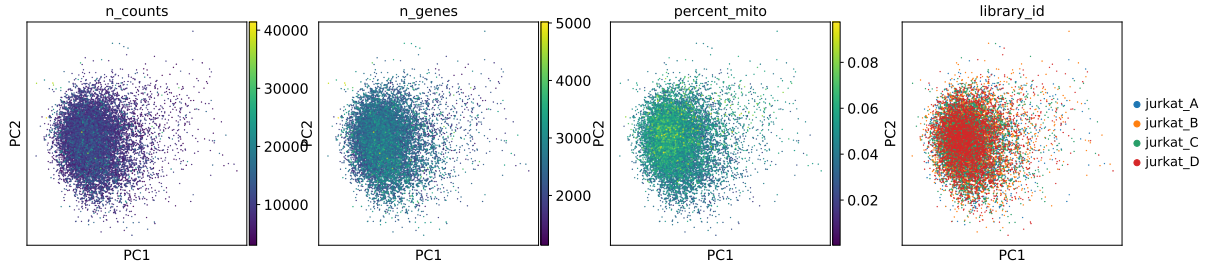
(b)

(c)

Figure A.7: **Pre-processing of Jurkat cells.** A.6a) Kernel density estimates after filtering for number of genes detected per cell (n_genes), number of reads per cell (n_counts) and percent mitochondrial reads per cell (percent_mito). A.6b) Number of genes detected versus number of reads per cell after filtering. A.6c) Percent mitochondrial reads versus number of reads per cell after filtering.



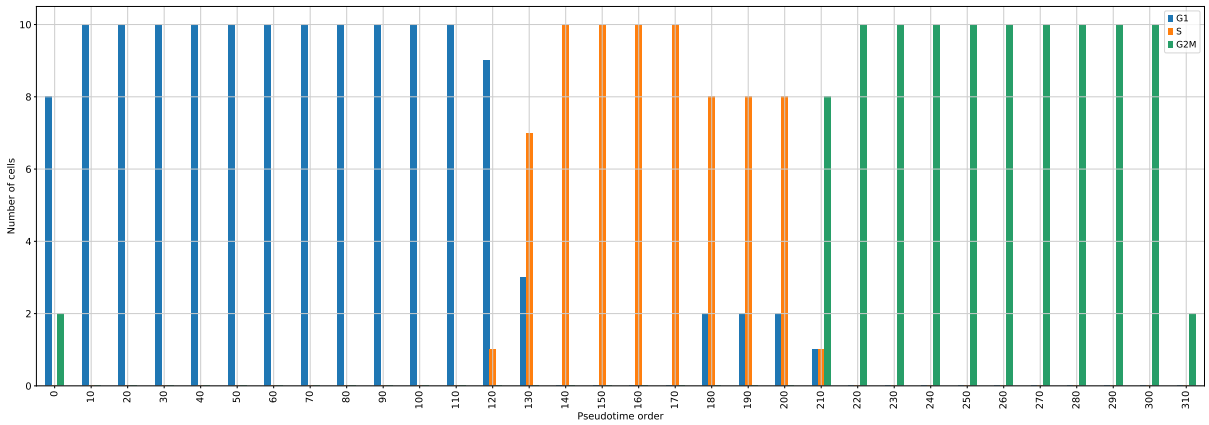
(a)



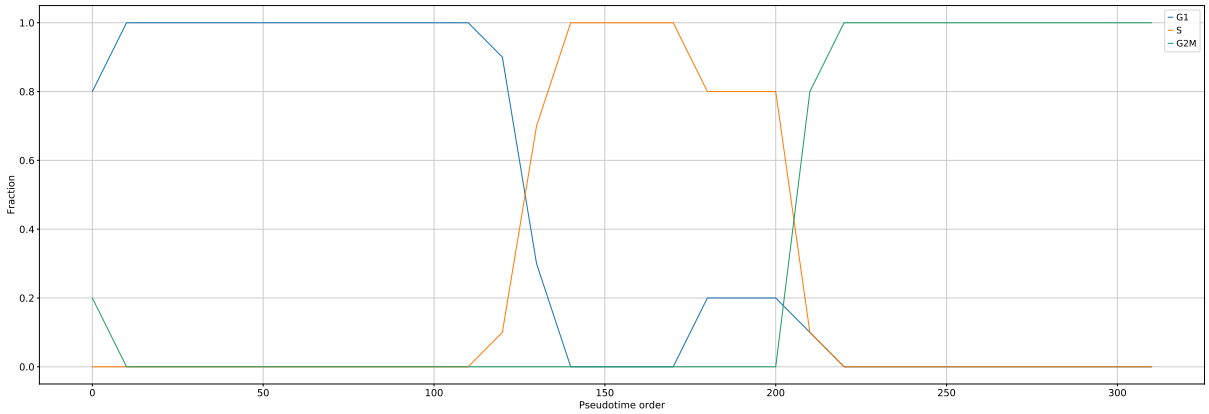
(b)

Figure A.8: **PCA for Jurkat cells.** Projection of the data set into the PC1,PC2 plane. The cells are colored by number of reads (n_counts), the number of detected genes (n_genes), fraction of mitochondrial reads (percent_mito) and batch id (library_id). A.8a) Initial PCA projection of the 293t data set. A.8b) PCA projection after regressing out number of reads.

[H]



(a)



(b)

Figure A.9: **Ordering of Jurkat cells.** The pseudotime ordering after rearranging so that the G2/M-G1 boundary is at zero. We find that the phase boundaries are located at order 127 for G1-S and 196 for S-G2/M. A.9a) Number of cells per phase for each bin of size 20 through the pseudotime ordering of the cells. A.9b) Linear curves representing the fraction of cells in each phase along the psedotime ordering.

A.4 Pseudotime Ordering of Jurkat cells

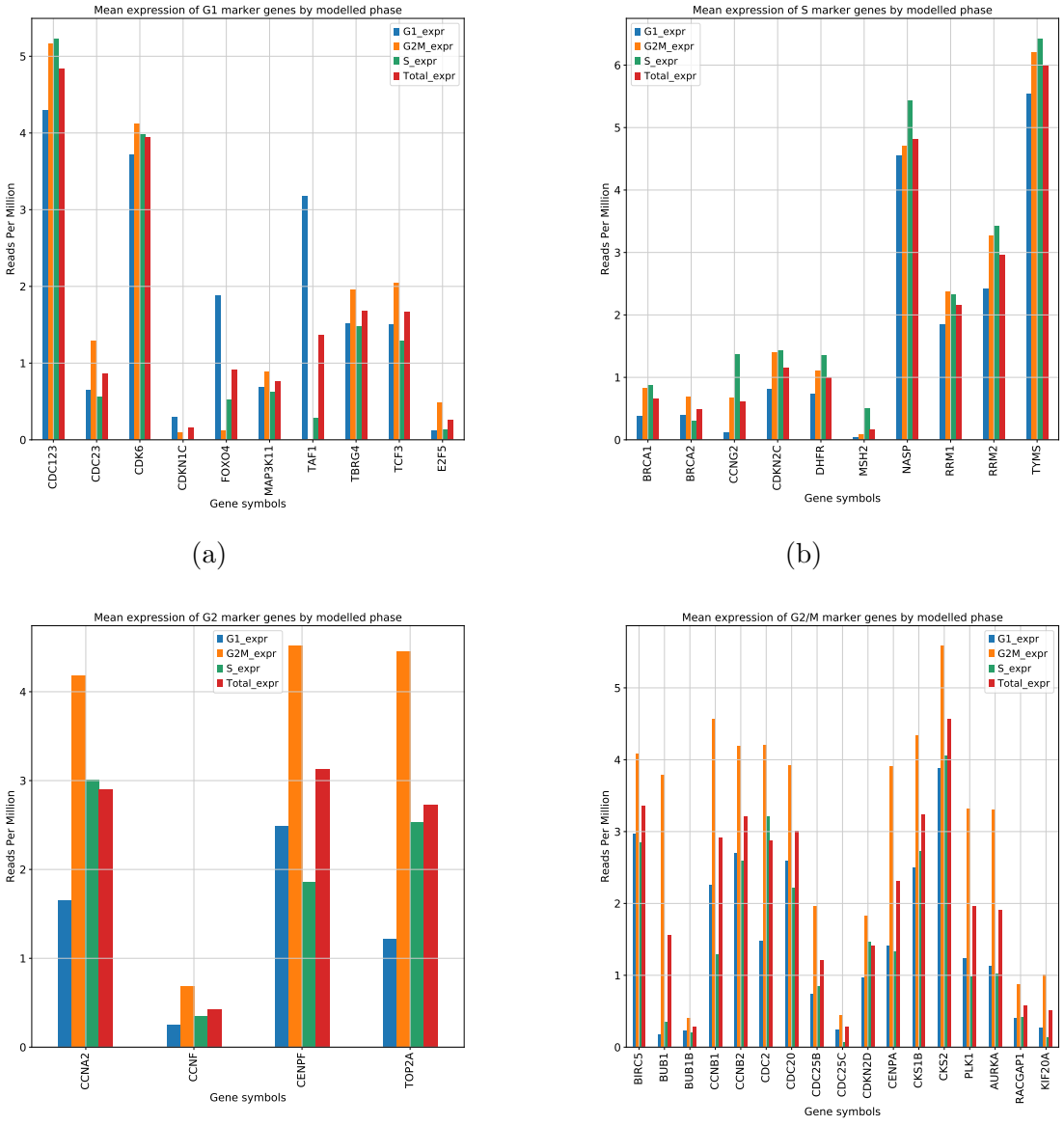


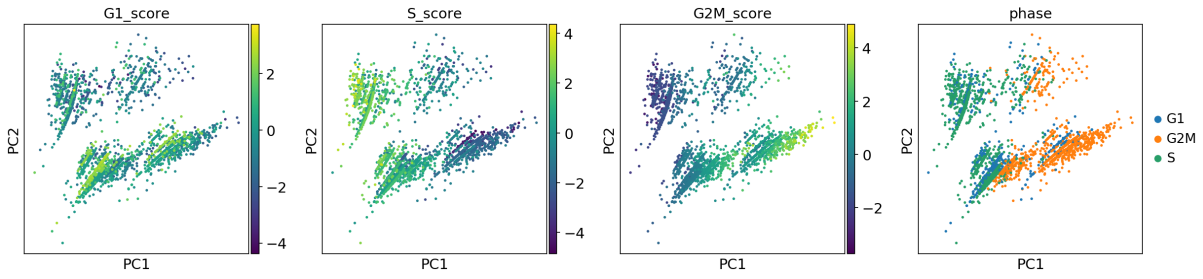
Figure A.10: **Mean expressions for Jurkat cells.** Mean expression of marker genes in the modelled phases along the pseudotime ordering. We use the full data set of all 18975 RPM normalized genes in the ordering. Roughly $\sim 78\%$ of the marker have their highest mean expression in the expected modelled phase. A.10a) Mean expressions for G1 marker genes. A.10b) Mean expressions for S marker genes. A.10c) Mean expressions for G2 marker genes. A.10d) Mean expressions for G2/M marker genes.

Appendix B

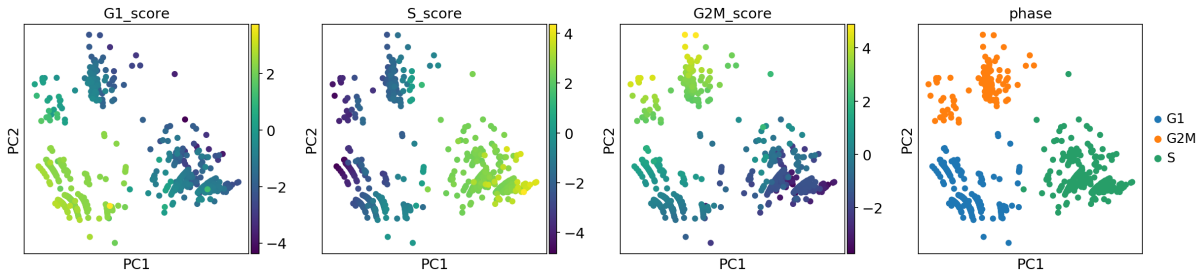
Figures

B.1 Low Input PCA for HaCAT

We performed a PCA analysis with low input of marker genes on the HaCat cells to compare their behaviour with the observed patterns in the 293t cells and the Jurkat cells. We see that the patterns clearly are related amount of marker genes present. Here, we used the same set of marker genes on the HaCat cells as those that were used for 293t.



(a) Before filtering on cell cycle phase scores.



(b) After filtering on cell cycle phase scores.

Figure B.1: The result of a low input of marker genes in the PCA analysis of HaCat cells. The HaCat cells show similar behaviour as the 293t and Jurkat cells.

Appendix C

Tables

C.1 Barcode Whitelists for STARsolo

The following whitelists can be found within the folder structure of a CellRanger installation. Specifically, they are found in the following folder:

`cellranger-3.0.2/cellranger-cs/3.0.2/lib/python/cellranger/barcodes/`.

Table C.1: Whitelists used for cell barcode validation depending on the 10X chemistry version.

10X Chemistry	Barcode Whitelist
10X Genomics Chromium Single Cell V1	737K-april-2014_rc.txt
10X Genomics Chromium Single Cell V2	737K-august-2016.txt
10X Genomics Chromium Single Cell V3	3M-february-2018.txt

C.2 Marker Genes for Cell Cycle Identifaction

The following set of genes are used as the set of marker genes in our cell cycle scoring. The source whitfield_2002 refers to the study [Whi+02] and regev_lab_2016 to [Tir+16].

Table C.2: The list of phase specific genes presented with their Ensembl id, gene symbol, associated phase and their source.

Ensembl	gene_symbols	phase	source
ENSG00000012048	BRCA1	S	whitfield_2002
ENSG00000139618	BRCA2	S	whitfield_2002
ENSG00000138764	CCNG2	S	whitfield_2002
ENSG00000123080	CDKN2C	S	whitfield_2002
ENSG00000228716	DHFR	S	whitfield_2002
ENSG00000095002	MSH2	S	whitfield_2002
ENSG00000132780	NASP	S	whitfield_2002
ENSG00000167325	RRM1	S	whitfield_2002
ENSG00000171848	RRM2	S	whitfield_2002
ENSG00000176890	TYMS	S	whitfield_2002
ENSG00000145386	CCNA2	G2	whitfield_2002
ENSG00000162063	CCNF	G2	whitfield_2002
ENSG00000117724	CENPF	G2	whitfield_2002
ENSG00000131747	TOP2A	G2	whitfield_2002
ENSG00000089685	BIRC5	G2/M	whitfield_2002
ENSG00000169679	BUB1	G2/M	whitfield_2002
ENSG00000156970	BUB1B	G2/M	whitfield_2002
ENSG00000134057	CCNB1	G2/M	whitfield_2002
ENSG00000157456	CCNB2	G2/M	whitfield_2002
ENSG00000170312	CDC2	G2/M	whitfield_2002
ENSG00000117399	CDC20	G2/M	whitfield_2002
ENSG00000101224	CDC25B	G2/M	whitfield_2002
ENSG00000158402	CDC25C	G2/M	whitfield_2002
ENSG00000129355	CDKN2D	G2/M	whitfield_2002
ENSG00000115163	CENPA	G2/M	whitfield_2002

Continued on next page

Table C.2: The list of phase specific genes presented with their Ensembl id, gene symbol, associated phase and their source.

Ensembl	gene_symbols	phase	source
ENSG00000173207	CKS1B	G2/M	whitfield_2002
ENSG00000123975	CKS2	G2/M	whitfield_2002
ENSG00000166851	PLK1	G2/M	whitfield_2002
ENSG00000087586	AURKA	G2/M	whitfield_2002
ENSG00000161800	RACGAP1	G2/M	whitfield_2002
ENSG00000112984	KIF20A	G2/M	whitfield_2002
ENSG00000151465	CDC123	G1	whitfield_2002
ENSG00000094880	CDC23	G1	whitfield_2002
ENSG00000105810	CDK6	G1	whitfield_2002
ENSG00000129757	CDKN1C	G1	whitfield_2002
ENSG00000184481	FOXO4	G1	whitfield_2002
ENSG00000173327	MAP3K11	G1	whitfield_2002
ENSG00000147133	TAF1	G1	whitfield_2002
ENSG00000136270	TBRG4	G1	whitfield_2002
ENSG00000071564	TCF3	G1	whitfield_2002
ENSG00000133740	E2F5	G1	whitfield_2002
ENSG00000164754	RAD21	M/G1	whitfield_2002
ENSG00000164611	PTTG1	M/G1	whitfield_2002
ENSG00000150630	VEGFC	M/G1	whitfield_2002
ENSG00000100526	CDKN3	M/G1	whitfield_2002
ENSG00000105173	CCNE1	G1/S	whitfield_2002
ENSG00000175305	CCNE2	G1/S	whitfield_2002
ENSG00000164045	CDC25A	G1/S	whitfield_2002
ENSG00000093009	CDC45L	G1/S	whitfield_2002
ENSG00000094804	CDC6	G1/S	whitfield_2002
ENSG00000124762	CDKN1A	G1/S	whitfield_2002
ENSG00000101412	E2F1	G1/S	whitfield_2002
ENSG00000073111	MCM2	G1/S	whitfield_2002
ENSG00000149308	NPAT	G1/S	whitfield_2002

Continued on next page

Table C.2: The list of phase specific genes presented with their Ensembl id, gene symbol, associated phase and their source.

Ensembl	gene_symbols	phase	source
ENSG00000132646	PCNA	G1/S	whitfield_2002
ENSG00000163950	SLBP	G1/S	whitfield_2002
ENSG00000100297	MCM5	S	regev_lab_2016
ENSG00000168496	FEN1	S	regev_lab_2016
ENSG00000104738	MCM4	S	regev_lab_2016
ENSG00000076248	UNG	S	regev_lab_2016
ENSG00000131153	GINS2	S	regev_lab_2016
ENSG00000076003	MCM6	S	regev_lab_2016
ENSG00000144354	CDCA7	S	regev_lab_2016
ENSG00000143476	DTL	S	regev_lab_2016
ENSG00000198056	PRIM1	S	regev_lab_2016
ENSG00000276043	UHRF1	S	regev_lab_2016
ENSG00000119969	HELLS	S	regev_lab_2016
ENSG00000049541	RFC2	S	regev_lab_2016
ENSG00000117748	RPA2	S	regev_lab_2016
ENSG00000111247	RAD51AP1	S	regev_lab_2016
ENSG00000112312	GMNN	S	regev_lab_2016
ENSG00000092470	WDR76	S	regev_lab_2016
ENSG00000012963	UBR7	S	regev_lab_2016
ENSG00000077514	POLD3	S	regev_lab_2016
ENSG00000156802	ATAD2	S	regev_lab_2016
ENSG00000051180	RAD51	S	regev_lab_2016
ENSG00000174371	EXO1	S	regev_lab_2016
ENSG00000075131	TIPIN	S	regev_lab_2016
ENSG00000136982	DSCC1	S	regev_lab_2016
ENSG00000197299	BLM	S	regev_lab_2016
ENSG00000118412	CASP8AP2	S	regev_lab_2016
ENSG00000162607	USP1	S	regev_lab_2016
ENSG00000092853	CLSPN	S	regev_lab_2016

Continued on next page

Table C.2: The list of phase specific genes presented with their Ensembl id, gene symbol, associated phase and their source.

Ensembl	gene_symbols	phase	source
ENSG00000101868	POLA1	S	regev_lab_2016
ENSG00000159259	CHAF1B	S	regev_lab_2016
ENSG00000136492	BRIP1	S	regev_lab_2016
ENSG00000129173	E2F8	S	regev_lab_2016
ENSG00000164104	HMGB2	G2/M	regev_lab_2016
ENSG00000137804	NUSAP1	G2/M	regev_lab_2016
ENSG00000175063	UBE2C	G2/M	regev_lab_2016
ENSG00000088325	TPX2	G2/M	regev_lab_2016
ENSG00000080986	NDC80	G2/M	regev_lab_2016
ENSG00000143228	NUF2	G2/M	regev_lab_2016
ENSG00000148773	MKI67	G2/M	regev_lab_2016
ENSG00000120802	TMPO	G2/M	regev_lab_2016
ENSG00000013810	TACC3	G2/M	regev_lab_2016
ENSG00000129195	PIMREG	G2/M	regev_lab_2016
ENSG00000113810	SMC4	G2/M	regev_lab_2016
ENSG00000169607	CKAP2L	G2/M	regev_lab_2016
ENSG00000136108	CKAP2	G2/M	regev_lab_2016
ENSG00000178999	AURKB	G2/M	regev_lab_2016
ENSG00000138160	KIF11	G2/M	regev_lab_2016
ENSG00000143401	ANP32E	G2/M	regev_lab_2016
ENSG00000188229	TUBB4B	G2/M	regev_lab_2016
ENSG00000075218	GTSE1	G2/M	regev_lab_2016
ENSG00000138182	KIF20B	G2/M	regev_lab_2016
ENSG00000123485	HJURP	G2/M	regev_lab_2016
ENSG00000111665	CDCA3	G2/M	regev_lab_2016
ENSG00000112742	TTK	G2/M	regev_lab_2016
ENSG00000142945	KIF2C	G2/M	regev_lab_2016
ENSG00000100401	RANGAP1	G2/M	regev_lab_2016
ENSG00000010292	NCAPD2	G2/M	regev_lab_2016

Continued on next page

Table C.2: The list of phase specific genes presented with their Ensembl id, gene symbol, associated phase and their source.

Ensembl	gene_symbols	phase	source
ENSG00000126787	DLGAP5	G2/M	regev_lab_2016
ENSG00000184661	CDCA2	G2/M	regev_lab_2016
ENSG00000134690	CDCA8	G2/M	regev_lab_2016
ENSG00000114346	ECT2	G2/M	regev_lab_2016
ENSG00000137807	KIF23	G2/M	regev_lab_2016
ENSG00000072571	HMMR	G2/M	regev_lab_2016
ENSG00000134222	PSRC1	G2/M	regev_lab_2016
ENSG00000011426	ANLN	G2/M	regev_lab_2016
ENSG00000143815	LBR	G2/M	regev_lab_2016
ENSG00000175216	CKAP5	G2/M	regev_lab_2016
ENSG00000138778	CENPE	G2/M	regev_lab_2016
ENSG00000102974	CTCF	G2/M	regev_lab_2016
ENSG00000117650	NEK2	G2/M	regev_lab_2016
ENSG00000092140	G2E3	G2/M	regev_lab_2016
ENSG00000139354	GAS2L3	G2/M	regev_lab_2016
ENSG00000094916	CBX5	G2/M	regev_lab_2016

C.3 Functional Enrichment of HaCat cells

We list up to the 60 most significant terms in our functional enrichment of the genes found to be significantly cell cycle periodic. The results are listed per data set per cell cyle phase.

Table C.3: Significant terms in the functional enrichment of Hacat cells classified as M/G1 by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0033044	regulation of chromosome organization	342	7	1.939e-04
2	GO:BP	GO:0051276	chromosome organization	1220	10	1.065e-03
3	GO:BP	GO:2001251	negative regulation of chromosome organization	145	5	1.266e-03
4	GO:BP	GO:0033047	regulation of mitotic sister chromatid segregation	69	4	2.306e-03
5	GO:BP	GO:0000278	mitotic cell cycle	1015	9	2.494e-03
6	GO:BP	GO:2001252	positive regulation of chromosome organization	173	5	3.027e-03
7	GO:BP	GO:0033045	regulation of sister chromatid segregation	81	4	4.395e-03
8	GO:BP	GO:0006323	DNA packaging	205	5	6.958e-03
9	GO:BP	GO:0051984	positive regulation of chromosome segregation	28	3	8.464e-03
10	GO:BP	GO:0051983	regulation of chromosome segregation	103	4	1.147e-02
11	GO:BP	GO:0065004	protein-DNA complex assembly	243	5	1.592e-02
12	GO:BP	GO:0033043	regulation of organelle organization	1275	9	1.635e-02
13	GO:BP	GO:0007346	regulation of mitotic cell cycle	671	7	1.747e-02
14	GO:BP	GO:0098813	nuclear chromosome segregation	261	5	2.249e-02
15	GO:BP	GO:2000816	negative regulation of mitotic sister chromatid separation	40	3	2.528e-02
16	GO:BP	GO:1905819	negative regulation of chromosome separation	41	3	2.726e-02
17	GO:BP	GO:0033048	negative regulation of mitotic sister chromatid segregation	43	3	3.150e-02
18	GO:BP	GO:0030261	chromosome condensation	43	3	3.150e-02
19	GO:BP	GO:0071824	protein-DNA complex subunit organization	282	5	3.266e-02
20	GO:BP	GO:0033046	negative regulation of sister chromatid segregation	45	3	3.617e-02
21	GO:BP	GO:0051985	negative regulation of chromosome segregation	46	3	3.866e-02
22	GO:BP	GO:0071103	DNA conformation change	301	5	4.467e-02
23	GO:BP	GO:0007049	cell cycle	1850	10	4.580e-02

g:Profiler (biit.cs.ut.ee/gprofiler)

Table C.4: Significant terms in the functional enrichment of Hacat cells classified as G1/S by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0009628	response to abiotic stimulus	1245	16	6.829e-04
2	GO:BP	GO:0009314	response to radiation	450	10	1.306e-03

g:Profiler (biit.cs.ut.ee/gprofiler)

Table C.5: The 60 most significant terms in the functional enrichment of Hacat cells classified as S by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0006260	DNA replication	282	100	1.269e-64
2	GO:BP	GO:0006259	DNA metabolic process	926	160	3.348e-56
3	GO:BP	GO:0006261	DNA-dependent DNA replication	153	67	2.301e-49
4	GO:BP	GO:0006281	DNA repair	545	116	6.694e-49
5	GO:BP	GO:0006974	cellular response to DNA damage stimulus	849	142	2.561e-47
6	GO:BP	GO:0007049	cell cycle	1850	215	3.998e-47
7	GO:BP	GO:0022402	cell cycle process	1383	170	9.541e-39
8	GO:BP	GO:0033554	cellular response to stress	2052	205	3.027e-34
9	GO:BP	GO:0051276	chromosome organization	1220	150	1.854e-33
10	GO:BP	GO:0000278	mitotic cell cycle	1015	130	3.862e-30
11	GO:BP	GO:1903047	mitotic cell cycle process	871	117	1.247e-28
12	GO:BP	GO:0044786	cell cycle DNA replication	66	35	2.400e-28
13	GO:BP	GO:0006310	DNA recombination	280	64	2.236e-27
14	GO:BP	GO:0033260	nuclear DNA replication	55	30	1.758e-24
15	GO:BP	GO:0006302	double-strand break repair	241	55	3.861e-23
16	GO:BP	GO:0044772	mitotic cell cycle phase transition	581	85	1.970e-22
17	GO:BP	GO:0044770	cell cycle phase transition	624	88	3.567e-22
18	GO:BP	GO:0051726	regulation of cell cycle	1213	128	1.980e-21
19	GO:BP	GO:0000723	telomere maintenance	163	43	2.822e-20
20	GO:BP	GO:0000724	double-strand break repair via homologous recombination	128	38	1.109e-19
21	GO:BP	GO:0000725	recombinational repair	129	38	1.513e-19
22	GO:BP	GO:0032201	telomere maintenance via semi-conservative replication	28	20	4.479e-19
23	GO:BP	GO:0032200	telomere organization	176	43	7.676e-19
24	GO:BP	GO:0010564	regulation of cell cycle process	797	92	4.643e-17
25	GO:BP	GO:0071103	DNA conformation change	301	53	1.113e-16
26	GO:BP	GO:0007346	regulation of mitotic cell cycle	671	82	1.999e-16
27	GO:BP	GO:0000082	G1/S transition of mitotic cell cycle	272	49	1.106e-15
28	GO:BP	GO:0000375	RNA splicing, via transesterification reactions	345	55	2.471e-15
29	GO:BP	GO:0000377	RNA splicing, via transesterification reactions with bulged adenosine as nucleophile	342	54	8.037e-15
30	GO:BP	GO:0000398	mRNA splicing, via spliceosome	342	54	8.037e-15
31	GO:BP	GO:0044843	cell cycle G1/S phase transition	292	49	2.357e-14
32	GO:BP	GO:0006270	DNA replication initiation	41	20	2.373e-14
33	GO:BP	GO:0045786	negative regulation of cell cycle	644	76	3.717e-14
34	GO:BP	GO:0008380	RNA splicing	431	60	5.228e-14
35	GO:BP	GO:0006397	mRNA processing	512	66	6.724e-14
36	GO:BP	GO:0071897	DNA biosynthetic process	194	39	9.917e-14
37	GO:BP	GO:0006275	regulation of DNA replication	113	30	1.349e-13
38	GO:BP	GO:0000075	cell cycle checkpoint	220	41	2.606e-13
39	GO:BP	GO:0006396	RNA processing	925	92	8.274e-13
40	GO:BP	GO:0006289	nucleotide-excision repair	109	28	3.761e-12
41	GO:BP	GO:0032508	DNA duplex unwinding	85	25	3.870e-12
42	GO:BP	GO:0032392	DNA geometric change	95	26	7.295e-12
43	GO:BP	GO:0010948	negative regulation of cell cycle process	361	51	7.877e-12
44	GO:BP	GO:0090329	regulation of DNA-dependent DNA replication	55	20	2.625e-11
45	GO:BP	GO:0016071	mRNA metabolic process	852	84	3.050e-11
46	GO:BP	GO:0006297	nucleotide-excision repair, DNA gap filling	23	14	3.172e-11
47	GO:BP	GO:1901990	regulation of mitotic cell cycle phase transition	442	56	4.099e-11
48	GO:BP	GO:0042769	DNA damage response, detection of DNA damage	38	17	4.278e-11
49	GO:BP	GO:0031570	DNA integrity checkpoint	159	32	6.547e-11
50	GO:BP	GO:0036297	interstrand cross-link repair	53	19	1.651e-10
51	GO:BP	GO:1901987	regulation of cell cycle phase transition	478	57	3.265e-10
52	GO:BP	GO:0051052	regulation of DNA metabolic process	347	47	4.830e-10
53	GO:BP	GO:0051054	positive regulation of DNA metabolic process	193	34	6.063e-10
54	GO:BP	GO:0007059	chromosome segregation	314	44	8.737e-10
55	GO:BP	GO:0006301	postreplication repair	52	18	1.518e-09
56	GO:BP	GO:0000731	DNA synthesis involved in DNA repair	52	18	1.518e-09
57	GO:BP	GO:0098813	nuclear chromosome segregation	261	39	2.524e-09
58	GO:BP	GO:0007093	mitotic cell cycle checkpoint	170	31	2.671e-09
59	GO:BP	GO:0019985	translesion synthesis	41	16	3.374e-09
60	GO:BP	GO:0045005	DNA-dependent DNA replication maintenance of fidelity	41	16	3.374e-09

g:Profiler (biit.cs.ut.ee/gprofiler)

Table C.6: The 60 most significant terms in the functional enrichment of Hacat cells classified as G2 by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0051301	cell division	605	52	8.687e-32
2	GO:BP	GO:0007049	cell cycle	1850	80	1.448e-30
3	GO:BP	GO:0007059	chromosome segregation	314	39	6.533e-29
4	GO:BP	GO:0000278	mitotic cell cycle	1015	59	2.205e-27
5	GO:BP	GO:0022402	cell cycle process	1383	66	2.773e-26
6	GO:BP	GO:1903047	mitotic cell cycle process	871	53	5.221e-25
7	GO:BP	GO:0000280	nuclear division	427	36	1.730e-20
8	GO:BP	GO:0098813	nuclear chromosome segregation	261	29	2.661e-19
9	GO:BP	GO:0048285	organelle fission	469	36	4.230e-19
10	GO:BP	GO:0051276	chromosome organization	1220	54	6.509e-19
11	GO:BP	GO:0000819	sister chromatid segregation	185	25	2.113e-18
12	GO:BP	GO:0140014	mitotic nuclear division	284	29	2.942e-18
13	GO:BP	GO:0000070	mitotic sister chromatid segregation	153	23	8.132e-18
14	GO:BP	GO:0000226	microtubule cytoskeleton organization	543	34	4.342e-15
15	GO:BP	GO:0051726	regulation of cell cycle	1213	46	6.998e-13
16	GO:BP	GO:0010564	regulation of cell cycle process	797	37	1.938e-12
17	GO:BP	GO:0007017	microtubule-based process	745	35	9.252e-12
18	GO:BP	GO:1902850	microtubule cytoskeleton organization involved in mitosis	125	16	1.409e-10
19	GO:BP	GO:0006323	DNA packaging	205	19	2.327e-10
20	GO:BP	GO:0071103	DNA conformation change	301	22	3.246e-10
21	GO:BP	GO:0007346	regulation of mitotic cell cycle	671	30	3.864e-09
22	GO:BP	GO:0044772	mitotic cell cycle phase transition	581	28	3.894e-09
23	GO:BP	GO:0007051	spindle organization	170	16	1.747e-08
24	GO:BP	GO:0044770	cell cycle phase transition	624	28	2.131e-08
25	GO:BP	GO:0051983	regulation of chromosome segregation	103	13	4.039e-08
26	GO:BP	GO:0007088	regulation of mitotic nuclear division	185	16	6.339e-08
27	GO:BP	GO:0051304	chromosome separation	91	12	1.452e-07
28	GO:BP	GO:0051783	regulation of nuclear division	210	16	4.246e-07
29	GO:BP	GO:0030261	chromosome condensation	43	9	5.024e-07
30	GO:BP	GO:0007052	mitotic spindle organization	104	12	7.139e-07
31	GO:BP	GO:1905818	regulation of chromosome separation	63	10	8.892e-07
32	GO:BP	GO:0007010	cytoskeleton organization	1315	38	1.453e-06
33	GO:BP	GO:0071459	protein localization to chromosome, centromeric region	22	7	2.088e-06
34	GO:BP	GO:0033047	regulation of mitotic sister chromatid segregation	69	10	2.251e-06
35	GO:BP	GO:0071174	mitotic spindle checkpoint	36	8	3.081e-06
36	GO:BP	GO:0071173	spindle assembly checkpoint	36	8	3.081e-06
37	GO:BP	GO:0031577	spindle checkpoint	36	8	3.081e-06
38	GO:BP	GO:0007094	mitotic spindle assembly checkpoint	36	8	3.081e-06
39	GO:BP	GO:0045841	negative regulation of mitotic metaphase/anaphase transition	38	8	4.898e-06
40	GO:BP	GO:0007091	metaphase/anaphase transition of mitotic cell cycle	56	9	6.063e-06
41	GO:BP	GO:1902100	negative regulation of metaphase/anaphase transition of cell cycle	39	8	6.112e-06
42	GO:BP	GO:2000816	negative regulation of mitotic sister chromatid separation	40	8	7.577e-06
43	GO:BP	GO:0010965	regulation of mitotic sister chromatid separation	58	9	8.382e-06
44	GO:BP	GO:0044784	metaphase/anaphase transition of cell cycle	58	9	8.382e-06
45	GO:BP	GO:1905819	negative regulation of chromosome separation	41	8	9.338e-06
46	GO:BP	GO:0033045	regulation of sister chromatid segregation	81	10	1.126e-05
47	GO:BP	GO:0043486	histone exchange	60	9	1.145e-05
48	GO:BP	GO:0051306	mitotic sister chromatid separation	61	9	1.332e-05
49	GO:BP	GO:0033048	negative regulation of mitotic sister chromatid segregation	43	8	1.394e-05
50	GO:BP	GO:0034501	protein localization to kinetochore	17	6	1.682e-05
51	GO:BP	GO:0034080	CENP-A containing nucleosome assembly	44	8	1.690e-05
52	GO:BP	GO:0061641	CENP-A containing chromatin organization	44	8	1.690e-05
53	GO:BP	GO:0000910	cytokinesis	168	13	1.859e-05
54	GO:BP	GO:0033046	negative regulation of sister chromatid segregation	45	8	2.039e-05
55	GO:BP	GO:0051985	negative regulation of chromosome segregation	46	8	2.448e-05
56	GO:BP	GO:0031055	chromatin remodeling at centromere	47	8	2.925e-05
57	GO:BP	GO:0008608	attachment of spindle microtubules to kinetochore	33	7	4.783e-05
58	GO:BP	GO:0030071	regulation of mitotic metaphase/anaphase transition	53	8	7.850e-05
59	GO:BP	GO:0045839	negative regulation of mitotic nuclear division	53	8	7.850e-05
60	GO:BP	GO:0006336	DNA replication-independent nucleosome assembly	54	8	9.140e-05

g:Profiler (biit.cs.ut.ee/gprofiler)

Table C.7: The 60 most significant terms in the functional enrichment of Hacat cells classified as G2/M by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0051301	cell division	605	42	4.701e-33
2	GO:BP	GO:1903047	mitotic cell cycle process	871	46	1.593e-31
3	GO:BP	GO:0000278	mitotic cell cycle	1015	48	5.138e-31
4	GO:BP	GO:0140014	mitotic nuclear division	284	31	3.635e-29
5	GO:BP	GO:0007049	cell cycle	1850	56	1.744e-27
6	GO:BP	GO:0022402	cell cycle process	1383	50	4.200e-27
7	GO:BP	GO:0000280	nuclear division	427	33	2.458e-26
8	GO:BP	GO:0048285	organelle fission	469	33	5.267e-25
9	GO:BP	GO:0010564	regulation of cell cycle process	797	36	5.288e-21
10	GO:BP	GO:0007017	microtubule-based process	745	35	7.462e-21
11	GO:BP	GO:0000070	mitotic sister chromatid segregation	153	21	7.766e-21
12	GO:BP	GO:0000819	sister chromatid segregation	185	22	1.515e-20
13	GO:BP	GO:0007059	chromosome segregation	314	25	2.295e-19
14	GO:BP	GO:0000226	microtubule cytoskeleton organization	543	30	2.737e-19
15	GO:BP	GO:0051726	regulation of cell cycle	1213	40	8.114e-19
16	GO:BP	GO:0098813	nuclear chromosome segregation	261	23	1.404e-18
17	GO:BP	GO:0007346	regulation of mitotic cell cycle	671	31	9.086e-18
18	GO:BP	GO:0044770	cell cycle phase transition	624	30	1.463e-17
19	GO:BP	GO:0044772	mitotic cell cycle phase transition	581	29	2.670e-17
20	GO:BP	GO:1902850	microtubule cytoskeleton organization involved in mitosis	125	17	2.661e-16
21	GO:BP	GO:0007051	spindle organization	170	18	2.134e-15
22	GO:BP	GO:0051983	regulation of chromosome segregation	103	15	1.437e-14
23	GO:BP	GO:1901987	regulation of cell cycle phase transition	478	24	8.241e-14
24	GO:BP	GO:0051783	regulation of nuclear division	210	18	9.723e-14
25	GO:BP	GO:1901990	regulation of mitotic cell cycle phase transition	442	23	1.886e-13
26	GO:BP	GO:0007088	regulation of mitotic nuclear division	185	17	2.371e-13
27	GO:BP	GO:0007052	mitotic spindle organization	104	14	6.006e-13
28	GO:BP	GO:0007010	cytoskeleton organization	1315	34	4.517e-12
29	GO:BP	GO:0044839	cell cycle G2/M phase transition	272	17	1.473e-10
30	GO:BP	GO:0051303	establishment of chromosome localization	76	11	4.567e-10
31	GO:BP	GO:0050000	chromosome localization	77	11	5.305e-10
32	GO:BP	GO:0007091	metaphase/anaphase transition of mitotic cell cycle	56	10	6.386e-10
33	GO:BP	GO:0010965	regulation of mitotic sister chromatid separation	58	10	9.276e-10
34	GO:BP	GO:0044784	metaphase/anaphase transition of cell cycle	58	10	9.276e-10
35	GO:BP	GO:0033045	regulation of sister chromatid segregation	81	11	9.458e-10
36	GO:BP	GO:0051306	mitotic sister chromatid separation	61	10	1.582e-09
37	GO:BP	GO:1905818	regulation of chromosome separation	63	10	2.223e-09
38	GO:BP	GO:0051304	chromosome separation	91	11	3.533e-09
39	GO:BP	GO:0033043	regulation of organelle organization	1275	30	3.691e-09
40	GO:BP	GO:0033047	regulation of mitotic sister chromatid segregation	69	10	5.759e-09
41	GO:BP	GO:0000281	mitotic cytokinesis	71	10	7.752e-09
42	GO:BP	GO:0000086	G2/M transition of mitotic cell cycle	254	15	1.090e-08
43	GO:BP	GO:0051640	organelle localization	612	21	1.551e-08
44	GO:BP	GO:0030071	regulation of mitotic metaphase/anaphase transition	53	9	1.641e-08
45	GO:BP	GO:1902749	regulation of cell cycle G2/M phase transition	217	14	1.777e-08
46	GO:BP	GO:0051225	spindle assembly	107	11	2.156e-08
47	GO:BP	GO:1902099	regulation of metaphase/anaphase transition of cell cycle	55	9	2.333e-08
48	GO:BP	GO:0051310	metaphase plate congression	58	9	3.857e-08
49	GO:BP	GO:0051276	chromosome organization	1220	28	4.473e-08
50	GO:BP	GO:0045787	positive regulation of cell cycle	399	17	6.824e-08
51	GO:BP	GO:0000910	cytokinesis	168	12	1.807e-07
52	GO:BP	GO:0061640	cytoskeleton-dependent cytokinesis	98	10	2.088e-07
53	GO:BP	GO:0070507	regulation of microtubule cytoskeleton organization	184	12	5.201e-07
54	GO:BP	GO:1903046	meiotic cell cycle process	190	12	7.533e-07
55	GO:BP	GO:0090068	positive regulation of cell cycle process	295	14	1.046e-06
56	GO:BP	GO:0010389	regulation of G2/M transition of mitotic cell cycle	200	12	1.359e-06
57	GO:BP	GO:0051321	meiotic cell cycle	252	13	1.635e-06
58	GO:BP	GO:0032886	regulation of microtubule-based process	216	12	3.272e-06
59	GO:BP	GO:0033046	negative regulation of sister chromatid segregation	45	7	6.993e-06
60	GO:BP	GO:0051985	negative regulation of chromosome segregation	46	7	8.212e-06

g:Profiler (biit.cs.ut.ee/gprofiler)

C.4 Functional Enrichment of 293t cells

We list up to the 60 most significant terms in our functional enrichment of the genes found to be significantly cell cycle periodic. The results are listed per data set per cell cyle phase.

Table C.8: Significant terms in the functional enrichment of 293t cells classified as M/G1 by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:1903047	mitotic cell cycle process	871	15	1.238e-02

g:Profiler (biit.cs.ut.ee/gprofiler)

Table C.9: Significant terms in the functional enrichment of 293t cells classified as G1 by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0006396	RNA processing	925	15	1.738e-02

g:Profiler (biit.cs.ut.ee/gprofiler)

Table C.10: The 60 most significant terms in the functional enrichment of 293t cells classified as S by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0006260	DNA replication	282	40	3.315e-21
2	GO:BP	GO:0006261	DNA-dependent DNA replication	153	28	3.469e-17
3	GO:BP	GO:0006259	DNA metabolic process	926	62	1.801e-16
4	GO:BP	GO:0044786	cell cycle DNA replication	66	18	1.436e-13
5	GO:BP	GO:0033260	nuclear DNA replication	55	16	2.514e-12
6	GO:BP	GO:0007049	cell cycle	1850	82	6.830e-12
7	GO:BP	GO:0022402	cell cycle process	1383	68	2.134e-11
8	GO:BP	GO:0006974	cellular response to DNA damage stimulus	849	51	4.246e-11
9	GO:BP	GO:0044770	cell cycle phase transition	624	41	1.160e-09
10	GO:BP	GO:0006281	DNA repair	545	38	1.458e-09
11	GO:BP	GO:0044772	mitotic cell cycle phase transition	581	38	1.020e-08
12	GO:BP	GO:1903047	mitotic cell cycle process	871	47	2.307e-08
13	GO:BP	GO:0000082	G1/S transition of mitotic cell cycle	272	25	4.739e-08
14	GO:BP	GO:0032201	telomere maintenance via semi-conservative replication	28	10	6.119e-08
15	GO:BP	GO:0000083	regulation of transcription involved in G1/S transition of mitotic cell cycle	28	10	6.119e-08
16	GO:BP	GO:0000278	mitotic cell cycle	1015	50	1.242e-07
17	GO:BP	GO:0006270	DNA replication initiation	41	11	2.102e-07
18	GO:BP	GO:0044843	cell cycle G1/S phase transition	292	25	2.194e-07
19	GO:BP	GO:0033554	cellular response to stress	2052	77	2.755e-07
20	GO:BP	GO:0051276	chromosome organization	1220	54	1.007e-06
21	GO:BP	GO:0000723	telomere maintenance	163	18	1.710e-06
22	GO:BP	GO:0042769	DNA damage response, detection of DNA damage	38	10	1.882e-06
23	GO:BP	GO:0006301	postreplication repair	52	11	3.374e-06
24	GO:BP	GO:0032200	telomere organization	176	18	5.949e-06
25	GO:BP	GO:0051726	regulation of cell cycle	1213	52	6.734e-06
26	GO:BP	GO:0010564	regulation of cell cycle process	797	40	7.644e-06
27	GO:BP	GO:0007346	regulation of mitotic cell cycle	671	35	3.027e-05
28	GO:BP	GO:0006275	regulation of DNA replication	113	14	3.049e-05
29	GO:BP	GO:0000731	DNA synthesis involved in DNA repair	52	10	5.047e-05
30	GO:BP	GO:0019985	translesion synthesis	41	9	7.572e-05
31	GO:BP	GO:0045786	negative regulation of cell cycle	644	33	1.231e-04
32	GO:BP	GO:0000075	cell cycle checkpoint	220	18	1.957e-04
33	GO:BP	GO:0006297	nucleotide-excision repair, DNA gap filling	23	7	2.253e-04
34	GO:BP	GO:0010948	negative regulation of cell cycle process	361	23	3.285e-04
35	GO:BP	GO:0031570	DNA integrity checkpoint	159	15	3.734e-04
36	GO:BP	GO:0034404	nucleobase-containing small molecule biosynthetic process	102	12	5.943e-04
37	GO:BP	GO:1901990	regulation of mitotic cell cycle phase transition	442	25	8.968e-04
38	GO:BP	GO:0071897	DNA biosynthetic process	194	16	9.210e-04
39	GO:BP	GO:0090329	regulation of DNA-dependent DNA replication	55	9	1.104e-03
40	GO:BP	GO:1901987	regulation of cell cycle phase transition	478	26	1.104e-03
41	GO:BP	GO:0006289	nucleotide-excision repair	109	12	1.233e-03
42	GO:BP	GO:0006283	transcription-coupled nucleotide-excision repair	73	10	1.423e-03
43	GO:BP	GO:0070987	error-free translesion synthesis	19	6	1.496e-03
44	GO:BP	GO:0042276	error-prone translesion synthesis	20	6	2.105e-03
45	GO:BP	GO:0043933	protein-containing complex subunit organization	2212	70	2.204e-03
46	GO:BP	GO:1901988	negative regulation of cell cycle phase transition	268	18	3.595e-03
47	GO:BP	GO:0065003	protein-containing complex assembly	1897	62	3.739e-03
48	GO:BP	GO:0009147	pyrimidine nucleoside triphosphate metabolic process	22	6	3.932e-03
49	GO:BP	GO:0007093	mitotic cell cycle checkpoint	170	14	4.976e-03
50	GO:BP	GO:0000280	nuclear division	427	23	6.034e-03
51	GO:BP	GO:0009116	nucleoside metabolic process	106	11	6.596e-03
52	GO:BP	GO:0006310	DNA recombination	280	18	6.690e-03
53	GO:BP	GO:0045930	negative regulation of mitotic cell cycle	338	20	6.742e-03
54	GO:BP	GO:0045787	positive regulation of cell cycle	399	22	6.850e-03
55	GO:BP	GO:0090407	organophosphate biosynthetic process	596	28	7.163e-03
56	GO:BP	GO:0006296	nucleotide-excision repair, DNA incision, 5'-to lesion	37	7	7.635e-03
57	GO:BP	GO:1901657	glycosyl compound metabolic process	130	12	8.123e-03
58	GO:BP	GO:0007059	chromosome segregation	314	19	8.634e-03
59	GO:BP	GO:0051052	regulation of DNA metabolic process	347	20	1.002e-02
60	GO:BP	GO:0033683	nucleotide-excision repair, DNA incision	40	7	1.321e-02

g:Profiler (biit.cs.ut.ee/gprofiler)

Table C.11: Significant terms in the functional enrichment of 293t cells classified as G2 by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0006396	RNA processing	925	44	3.693e-08
2	GO:BP	GO:0007049	cell cycle	1850	59	6.702e-05
3	GO:BP	GO:0008380	RNA splicing	431	24	1.191e-04
4	GO:BP	GO:0022613	ribonucleoprotein complex biogenesis	455	24	3.265e-04
5	GO:BP	GO:1903047	mitotic cell cycle process	871	35	3.758e-04
6	GO:BP	GO:0000278	mitotic cell cycle	1015	38	6.367e-04
7	GO:BP	GO:0006364	rRNA processing	204	15	1.336e-03
8	GO:BP	GO:0010564	regulation of cell cycle process	797	32	1.361e-03
9	GO:BP	GO:0016071	mRNA metabolic process	852	33	1.993e-03
10	GO:BP	GO:0022402	cell cycle process	1383	45	2.366e-03
11	GO:BP	GO:0016072	rRNA metabolic process	215	15	2.600e-03
12	GO:BP	GO:0046907	intracellular transport	1733	52	3.468e-03
13	GO:BP	GO:0033554	cellular response to stress	2052	58	5.169e-03
14	GO:BP	GO:0006397	mRNA processing	512	23	9.364e-03
15	GO:BP	GO:1901990	regulation of mitotic cell cycle phase transition	442	21	1.056e-02
16	GO:BP	GO:0034470	ncRNA processing	379	19	1.418e-02
17	GO:BP	GO:0000375	RNA splicing, via transesterification reactions	345	18	1.448e-02
18	GO:BP	GO:0070727	cellular macromolecule localization	1925	54	1.575e-02
19	GO:BP	GO:0042254	ribosome biogenesis	287	16	2.096e-02
20	GO:BP	GO:0000154	rRNA modification	33	6	2.133e-02
21	GO:BP	GO:0034613	cellular protein localization	1914	53	2.823e-02
22	GO:BP	GO:0034660	ncRNA metabolic process	473	21	2.966e-02
23	GO:BP	GO:1901987	regulation of cell cycle phase transition	478	21	3.471e-02
24	GO:BP	GO:0051726	regulation of cell cycle	1213	38	4.304e-02
25	GO:BP	GO:0000377	RNA splicing, via transesterification reactions with bulged adenosine as nucleophile	342	17	4.900e-02
26	GO:BP	GO:0000398	mRNA splicing, via spliceosome	342	17	4.900e-02

g:Profiler (biit.cs.ut.ee/gprofiler)

Table C.12: The 60 most significant terms in the functional enrichment of 293t cells classified as G2/M by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0051301	cell division	605	67	7.191e-23
2	GO:BP	GO:0000278	mitotic cell cycle	1015	82	1.880e-19
3	GO:BP	GO:0000819	sister chromatid segregation	185	36	2.939e-19
4	GO:BP	GO:0140014	mitotic nuclear division	284	43	6.108e-19
5	GO:BP	GO:0007059	chromosome segregation	314	45	6.150e-19
6	GO:BP	GO:0007049	cell cycle	1850	114	1.100e-18
7	GO:BP	GO:1903047	mitotic cell cycle process	871	74	1.749e-18
8	GO:BP	GO:0022402	cell cycle process	1383	93	6.080e-17
9	GO:BP	GO:0000070	mitotic sister chromatid segregation	153	31	8.328e-17
10	GO:BP	GO:0000280	nuclear division	427	49	1.367e-16
11	GO:BP	GO:0007052	mitotic spindle organization	104	26	3.504e-16
12	GO:BP	GO:0098813	nuclear chromosome segregation	261	38	7.554e-16
13	GO:BP	GO:0051726	regulation of cell cycle	1213	83	3.657e-15
14	GO:BP	GO:1902850	microtubule cytoskeleton organization involved in mitosis	125	27	4.141e-15
15	GO:BP	GO:0048285	organelle fission	469	49	7.204e-15
16	GO:BP	GO:0051276	chromosome organization	1220	82	1.801e-14
17	GO:BP	GO:0007051	spindle organization	170	29	1.922e-13
18	GO:BP	GO:0051983	regulation of chromosome segregation	103	23	6.875e-13
19	GO:BP	GO:0010564	regulation of cell cycle process	797	60	6.096e-12
20	GO:BP	GO:0007346	regulation of mitotic cell cycle	671	54	1.130e-11
21	GO:BP	GO:0007088	regulation of mitotic nuclear division	185	28	1.606e-11
22	GO:BP	GO:0007017	microtubule-based process	745	57	1.647e-11
23	GO:BP	GO:0010965	regulation of mitotic sister chromatid separation	58	17	4.211e-11
24	GO:BP	GO:0051783	regulation of nuclear division	210	29	6.089e-11
25	GO:BP	GO:0033045	regulation of sister chromatid segregation	81	19	1.071e-10
26	GO:BP	GO:0051306	mitotic sister chromatid separation	61	17	1.071e-10
27	GO:BP	GO:0031145	anaphase-promoting complex-dependent catabolic process	83	19	1.730e-10
28	GO:BP	GO:1905818	regulation of chromosome separation	63	17	1.935e-10
29	GO:BP	GO:0000226	microtubule cytoskeleton organization	543	46	2.283e-10
30	GO:BP	GO:0007091	metaphase/anaphase transition of mitotic cell cycle	56	16	3.944e-10
31	GO:BP	GO:0006396	RNA processing	925	62	3.978e-10
32	GO:BP	GO:0044784	metaphase/anaphase transition of cell cycle	58	16	7.248e-10
33	GO:BP	GO:0033047	regulation of mitotic sister chromatid segregation	69	17	9.990e-10
34	GO:BP	GO:0030071	regulation of mitotic metaphase/anaphase transition	53	15	2.682e-09
35	GO:BP	GO:0033046	negative regulation of sister chromatid segregation	45	14	3.611e-09
36	GO:BP	GO:1902099	regulation of metaphase/anaphase transition of cell cycle	55	15	4.888e-09
37	GO:BP	GO:0051985	negative regulation of chromosome segregation	46	14	5.079e-09
38	GO:BP	GO:0051304	chromosome separation	91	18	1.146e-08
39	GO:BP	GO:2000816	negative regulation of mitotic sister chromatid separation	40	13	1.227e-08
40	GO:BP	GO:1905819	negative regulation of chromosome separation	41	13	1.758e-08
41	GO:BP	GO:0045786	negative regulation of cell cycle	644	47	2.562e-08
42	GO:BP	GO:0033048	negative regulation of mitotic sister chromatid segregation	43	13	3.494e-08
43	GO:BP	GO:0044772	mitotic cell cycle phase transition	581	44	3.676e-08
44	GO:BP	GO:0008380	RNA splicing	431	37	4.780e-08
45	GO:BP	GO:0016071	mRNA metabolic process	852	55	4.790e-08
46	GO:BP	GO:0090307	mitotic spindle assembly	54	14	5.772e-08
47	GO:BP	GO:0044770	cell cycle phase transition	624	45	1.090e-07
48	GO:BP	GO:0033043	regulation of organelle organization	1275	70	1.140e-07
49	GO:BP	GO:0045841	negative regulation of mitotic metaphase/anaphase transition	38	12	1.216e-07
50	GO:BP	GO:1902100	negative regulation of metaphase/anaphase transition of cell cycle	39	12	1.719e-07
51	GO:BP	GO:0051225	spindle assembly	107	18	1.982e-07
52	GO:BP	GO:0000075	cell cycle checkpoint	220	25	2.994e-07
53	GO:BP	GO:0007093	mitotic cell cycle checkpoint	170	22	3.205e-07
54	GO:BP	GO:0008608	attachment of spindle microtubules to kinetochore	33	11	4.076e-07
55	GO:BP	GO:0051784	negative regulation of nuclear division	62	14	4.345e-07
56	GO:BP	GO:0006397	mRNA processing	512	39	4.826e-07
57	GO:BP	GO:0045839	negative regulation of mitotic nuclear division	53	13	6.468e-07
58	GO:BP	GO:2001251	negative regulation of chromosome organization	145	20	7.120e-07
59	GO:BP	GO:1903311	regulation of mRNA metabolic process	328	30	9.171e-07
60	GO:BP	GO:0071173	spindle assembly checkpoint	36	11	1.186e-06

g:Profiler (biit.cs.ut.ee/gprofiler)

C.5 Functional Enrichment of Jurkat cells

We list up to the 60 most significant terms in our functional enrichment of the genes found to be significantly cell cycle periodic. The results are listed per data set per cell cyle phase.

Table C.13: The 60 most significant terms in the functional enrichment of Jurkat cells classified as M/G1 by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0051276	chromosome organization	1220	19	4.846e-03
2	GO:BP	GO:0018205	peptidyl-lysine modification	399	11	4.963e-03
3	GO:BP	GO:0033554	cellular response to stress	2052	25	7.494e-03
4	GO:BP	GO:0016569	covalent chromatin modification	480	11	2.877e-02
5	GO:BP	GO:0006325	chromatin organization	804	14	3.906e-02

g:Profiler (biit.cs.ut.ee/gprofiler)

Table C.14: Significant terms in the functional enrichment of Jurkat cells classified as G1 by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0070317	negative regulation of G0 to G1 transition	41	3	1.454e-02
2	GO:BP	GO:0070316	regulation of G0 to G1 transition	45	3	1.930e-02
3	GO:BP	GO:0045023	G0 to G1 transition	48	3	2.348e-02

g:Profiler (biit.cs.ut.ee/gprofiler)

Table C.15: The 60 most significant terms in the functional enrichment of Jurkat cells classified as G1/S by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0006259	DNA metabolic process	926	49	2.337e-14
2	GO:BP	GO:0006260	DNA replication	282	24	5.244e-10
3	GO:BP	GO:0006974	cellular response to DNA damage stimulus	849	39	7.072e-09
4	GO:BP	GO:0006281	DNA repair	545	31	7.265e-09
5	GO:BP	GO:0000082	G1/S transition of mitotic cell cycle	272	22	1.519e-08
6	GO:BP	GO:0006261	DNA-dependent DNA replication	153	17	2.740e-08
7	GO:BP	GO:0044843	cell cycle G1/S phase transition	292	22	6.146e-08
8	GO:BP	GO:0051276	chromosome organization	1220	45	2.037e-07
9	GO:BP	GO:1903047	mitotic cell cycle process	871	36	1.018e-06
10	GO:BP	GO:0007049	cell cycle	1850	56	1.308e-06
11	GO:BP	GO:0033554	cellular response to stress	2052	59	2.871e-06
12	GO:BP	GO:0022402	cell cycle process	1383	46	3.541e-06
13	GO:BP	GO:0044772	mitotic cell cycle phase transition	581	28	3.889e-06
14	GO:BP	GO:0044770	cell cycle phase transition	624	29	4.448e-06
15	GO:BP	GO:0000278	mitotic cell cycle	1015	38	5.042e-06
16	GO:BP	GO:0006270	DNA replication initiation	41	8	9.605e-05
17	GO:BP	GO:0000083	regulation of transcription involved in G1/S transition of mitotic cell cycle	28	7	1.098e-04
18	GO:BP	GO:0051726	regulation of cell cycle	1213	37	1.581e-03
19	GO:BP	GO:0019985	translesion synthesis	41	7	1.809e-03
20	GO:BP	GO:0090068	positive regulation of cell cycle process	295	16	3.081e-03
21	GO:BP	GO:0007346	regulation of mitotic cell cycle	671	25	3.984e-03
22	GO:BP	GO:0044786	cell cycle DNA replication	66	8	4.386e-03
23	GO:BP	GO:0070987	error-free translesion synthesis	19	5	7.500e-03
24	GO:BP	GO:0010564	regulation of cell cycle process	797	27	9.004e-03
25	GO:BP	GO:0000731	DNA synthesis involved in DNA repair	52	7	9.554e-03
26	GO:BP	GO:0006301	postreplication repair	52	7	9.554e-03
27	GO:BP	GO:0042276	error-prone translesion synthesis	20	5	9.896e-03
28	GO:BP	GO:0034622	cellular protein-containing complex assembly	1102	33	1.016e-02
29	GO:BP	GO:0033260	nuclear DNA replication	55	7	1.403e-02
30	GO:BP	GO:0070647	protein modification by small protein conjugation or removal	1120	33	1.430e-02
31	GO:BP	GO:0031570	DNA integrity checkpoint	159	11	1.480e-02
32	GO:BP	GO:0006268	DNA unwinding involved in DNA replication	11	4	1.826e-02
33	GO:BP	GO:0006397	mRNA processing	512	20	2.300e-02
34	GO:BP	GO:0032508	DNA duplex unwinding	85	8	2.984e-02
35	GO:BP	GO:0006325	chromatin organization	804	26	3.119e-02
36	GO:BP	GO:0009408	response to heat	174	11	3.480e-02
37	GO:BP	GO:0045787	positive regulation of cell cycle	399	17	3.620e-02
38	GO:BP	GO:1901990	regulation of mitotic cell cycle phase transition	442	18	3.727e-02
39	GO:BP	GO:2000045	regulation of G1/S transition of mitotic cell cycle	178	11	4.305e-02
40	GO:BP	GO:1901991	negative regulation of mitotic cell cycle phase transition	249	13	4.609e-02

g:Profiler (biit.cs.ut.ee/gprofiler)

Table C.16: Significant terms in the functional enrichment of Jurkat cells classified as S by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0006396	RNA processing	925	81	7.111e-15
2	GO:BP	GO:0043933	protein-containing complex subunit organization	2212	129	1.215e-10
3	GO:BP	GO:0016071	mRNA metabolic process	852	69	1.439e-10
4	GO:BP	GO:0008380	RNA splicing	431	46	2.210e-10
5	GO:BP	GO:0006397	mRNA processing	512	49	2.102e-09
6	GO:BP	GO:0006260	DNA replication	282	35	2.942e-09
7	GO:BP	GO:1903047	mitotic cell cycle process	871	66	1.100e-08
8	GO:BP	GO:0044770	cell cycle phase transition	624	53	2.481e-08
9	GO:BP	GO:0022402	cell cycle process	1383	88	3.458e-08
10	GO:BP	GO:0044772	mitotic cell cycle phase transition	581	50	6.166e-08
11	GO:BP	GO:0034622	cellular protein-containing complex assembly	1102	74	1.605e-07
12	GO:BP	GO:0000278	mitotic cell cycle	1015	70	1.732e-07
13	GO:BP	GO:0000398	mRNA splicing, via spliceosome	342	36	1.738e-07
14	GO:BP	GO:0000377	RNA splicing, via transesterification reactions with bulged adenosine as nucleophile	342	36	1.738e-07
15	GO:BP	GO:0070125	mitochondrial translational elongation	87	18	2.054e-07
16	GO:BP	GO:0000375	RNA splicing, via transesterification reactions	345	36	2.225e-07
17	GO:BP	GO:0065003	protein-containing complex assembly	1897	107	2.336e-07
18	GO:BP	GO:0070126	mitochondrial translational termination	88	18	2.511e-07
19	GO:BP	GO:0006259	DNA metabolic process	926	64	1.126e-06
20	GO:BP	GO:0051276	chromosome organization	1220	77	1.151e-06
21	GO:BP	GO:0007049	cell cycle	1850	103	1.187e-06
22	GO:BP	GO:0032543	mitochondrial translation	135	21	1.546e-06
23	GO:BP	GO:0044786	cell cycle DNA replication	66	15	1.888e-06
24	GO:BP	GO:0006415	translational termination	103	18	3.743e-06
25	GO:BP	GO:0140053	mitochondrial gene expression	160	22	6.694e-06
26	GO:BP	GO:0010948	negative regulation of cell cycle process	361	34	1.080e-05
27	GO:BP	GO:0033260	nuclear DNA replication	55	13	1.518e-05
28	GO:BP	GO:0006261	DNA-dependent DNA replication	153	21	1.580e-05
29	GO:BP	GO:0006457	protein folding	219	25	2.791e-05
30	GO:BP	GO:0033554	cellular response to stress	2052	105	7.576e-05
31	GO:BP	GO:0006974	cellular response to DNA damage stimulus	849	56	7.591e-05
32	GO:BP	GO:0006414	translational elongation	140	19	9.903e-05
33	GO:BP	GO:0070199	establishment of protein localization to chromosome	27	9	1.245e-04
34	GO:BP	GO:0065004	protein-DNA complex assembly	243	25	2.210e-04
35	GO:BP	GO:0034660	ncRNA metabolic process	473	37	3.041e-04
36	GO:BP	GO:0045786	negative regulation of cell cycle	644	45	3.631e-04
37	GO:BP	GO:0034470	ncRNA processing	379	32	3.954e-04
38	GO:BP	GO:0044839	cell cycle G2/M phase transition	272	26	5.313e-04
39	GO:BP	GO:0010564	regulation of cell cycle process	797	51	8.193e-04
40	GO:BP	GO:0071824	protein-DNA complex subunit organization	282	26	1.072e-03
41	GO:BP	GO:0006336	DNA replication-independent nucleosome assembly	54	11	1.124e-03
42	GO:BP	GO:0007346	regulation of mitotic cell cycle	671	45	1.150e-03
43	GO:BP	GO:0034724	DNA replication-independent nucleosome organization	55	11	1.366e-03
44	GO:BP	GO:0006281	DNA repair	545	39	1.376e-03
45	GO:BP	GO:1902750	negative regulation of cell cycle G2/M phase transition	106	15	1.498e-03
46	GO:BP	GO:1902749	regulation of cell cycle G2/M phase transition	217	22	1.659e-03
47	GO:BP	GO:0000086	G2/M transition of mitotic cell cycle	254	24	1.899e-03
48	GO:BP	GO:0032201	telomere maintenance via semi-conservative replication	28	8	2.638e-03
49	GO:BP	GO:1901987	regulation of cell cycle phase transition	478	35	3.111e-03
50	GO:BP	GO:0043604	amide biosynthetic process	884	53	3.506e-03
51	GO:BP	GO:0071897	DNA biosynthetic process	194	20	4.052e-03
52	GO:BP	GO:0043044	ATP-dependent chromatin remodeling	87	13	4.420e-03
53	GO:BP	GO:1901566	organonitrogen compound biosynthetic process	1864	91	5.731e-03
54	GO:BP	GO:0019439	aromatic compound catabolic process	600	40	5.816e-03
55	GO:BP	GO:0031123	RNA 3'-end processing	149	17	6.013e-03
56	GO:BP	GO:0043624	cellular protein complex disassembly	217	21	6.248e-03
57	GO:BP	GO:0034655	nucleobase-containing compound catabolic process	536	37	6.252e-03
58	GO:BP	GO:0006270	DNA replication initiation	41	9	6.376e-03
59	GO:BP	GO:0019985	translesion synthesis	41	9	6.376e-03
60	GO:BP	GO:0006301	postreplication repair	52	10	6.478e-03

g:Profiler (biit.cs.ut.ee/gprofiler)

Table C.17: The 60 most significant terms in the functional enrichment of Jurkat cells classified as G2 by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0000377	RNA splicing, via transesterification reactions with bulged adenosine as nucleophile	342	20	1.416e-04
2	GO:BP	GO:0000398	mRNA splicing, via spliceosome	342	20	1.416e-04
3	GO:BP	GO:0043933	protein-containing complex subunit organization	2212	61	1.603e-04
4	GO:BP	GO:0000375	RNA splicing, via transesterification reactions	345	20	1.633e-04
5	GO:BP	GO:0008380	RNA splicing	431	22	3.350e-04
6	GO:BP	GO:0016032	viral process	823	31	8.964e-04
7	GO:BP	GO:0065003	protein-containing complex assembly	1897	53	9.763e-04
8	GO:BP	GO:0006396	RNA processing	925	33	1.251e-03
9	GO:BP	GO:0044403	symbiont process	886	32	1.445e-03
10	GO:BP	GO:0044419	interspecies interaction between organisms	933	33	1.515e-03
11	GO:BP	GO:0006397	mRNA processing	512	23	1.646e-03
12	GO:BP	GO:0034622	cellular protein-containing complex assembly	1102	36	2.943e-03
13	GO:BP	GO:1903047	mitotic cell cycle process	871	30	8.642e-03
14	GO:BP	GO:0033554	cellular response to stress	2052	53	1.051e-02
15	GO:BP	GO:0022613	ribonucleoprotein complex biogenesis	455	20	1.227e-02
16	GO:BP	GO:0000819	sister chromatid segregation	185	12	2.259e-02
17	GO:BP	GO:0000278	mitotic cell cycle	1015	32	2.488e-02
18	GO:BP	GO:0006457	protein folding	219	13	2.529e-02
19	GO:BP	GO:0007049	cell cycle	1850	48	2.900e-02
20	GO:BP	GO:1903311	regulation of mRNA metabolic process	328	16	3.036e-02
21	GO:BP	GO:0006974	cellular response to DNA damage stimulus	849	28	4.098e-02
22	GO:BP	GO:0016071	mRNA metabolic process	852	28	4.367e-02
23	GO:BP	GO:0007346	regulation of mitotic cell cycle	671	24	4.605e-02

g:Profiler (biit.cs.ut.ee/gprofiler)

Table C.18: The 60 most significant terms in the functional enrichment of Jurkat cells classified as G2/M by phase distribution of marker genes.

id	source	term_id	term_name	term_size	intersection_size	p_value
1	GO:BP	GO:0000278	mitotic cell cycle	1015	56	1.149e-17
2	GO:BP	GO:0007049	cell cycle	1850	75	4.824e-17
3	GO:BP	GO:0051301	cell division	605	43	7.643e-17
4	GO:BP	GO:1903047	mitotic cell cycle process	871	49	2.408e-15
5	GO:BP	GO:0140014	mitotic nuclear division	284	26	1.006e-11
6	GO:BP	GO:0000280	nuclear division	427	30	8.682e-11
7	GO:BP	GO:0007017	microtubule-based process	745	39	1.424e-10
8	GO:BP	GO:0048285	organelle fission	469	31	1.633e-10
9	GO:BP	GO:0022402	cell cycle process	1383	54	2.087e-10
10	GO:BP	GO:0051726	regulation of cell cycle	1213	50	2.780e-10
11	GO:BP	GO:1902850	microtubule cytoskeleton organization involved in mitosis	125	17	1.096e-09
12	GO:BP	GO:0000226	microtubule cytoskeleton organization	543	32	1.431e-09
13	GO:BP	GO:0000070	mitotic sister chromatid segregation	153	18	2.807e-09
14	GO:BP	GO:0000819	sister chromatid segregation	185	19	7.592e-09
15	GO:BP	GO:0051983	regulation of chromosome segregation	103	15	9.747e-09
16	GO:BP	GO:0007059	chromosome segregation	314	23	4.322e-08
17	GO:BP	GO:0098813	nuclear chromosome segregation	261	21	5.880e-08
18	GO:BP	GO:0007052	mitotic spindle organization	104	14	1.537e-07
19	GO:BP	GO:0033043	regulation of organelle organization	1275	46	3.073e-07
20	GO:BP	GO:0051783	regulation of nuclear division	210	18	5.882e-07
21	GO:BP	GO:0007088	regulation of mitotic nuclear division	185	17	6.373e-07
22	GO:BP	GO:0007051	spindle organization	170	16	1.525e-06
23	GO:BP	GO:0007346	regulation of mitotic cell cycle	671	31	1.572e-06
24	GO:BP	GO:0010564	regulation of cell cycle process	797	33	6.665e-06
25	GO:BP	GO:0033048	negative regulation of mitotic sister chromatid segregation	43	9	7.516e-06
26	GO:BP	GO:0033046	negative regulation of sister chromatid segregation	45	9	1.155e-05
27	GO:BP	GO:0051985	negative regulation of chromosome segregation	46	9	1.420e-05
28	GO:BP	GO:0044772	mitotic cell cycle phase transition	581	27	1.942e-05
29	GO:BP	GO:0044770	cell cycle phase transition	624	28	2.150e-05
30	GO:BP	GO:0007010	cytoskeleton organization	1315	43	2.680e-05
31	GO:BP	GO:0033047	regulation of mitotic sister chromatid segregation	69	10	4.220e-05
32	GO:BP	GO:0033044	regulation of chromosome organization	342	20	4.572e-05
33	GO:BP	GO:0045839	negative regulation of mitotic nuclear division	53	9	5.283e-05
34	GO:BP	GO:0045841	negative regulation of mitotic metaphase/anaphase transition	38	8	5.508e-05
35	GO:BP	GO:1902100	negative regulation of metaphase/anaphase transition of cell cycle	39	8	6.853e-05
36	GO:BP	GO:0010638	positive regulation of organelle organization	622	27	8.046e-05
37	GO:BP	GO:2000816	negative regulation of mitotic sister chromatid separation	40	8	8.472e-05
38	GO:BP	GO:0007091	metaphase/anaphase transition of mitotic cell cycle	56	9	8.735e-05
39	GO:BP	GO:0051276	chromosome organization	1220	40	8.896e-05
40	GO:BP	GO:1905819	negative regulation of chromosome separation	41	8	1.041e-04
41	GO:BP	GO:0044784	metaphase/anaphase transition of cell cycle	58	9	1.201e-04
42	GO:BP	GO:0010965	regulation of mitotic sister chromatid separation	58	9	1.201e-04
43	GO:BP	GO:0045787	positive regulation of cell cycle	399	21	1.227e-04
44	GO:BP	GO:0051306	mitotic sister chromatid separation	61	9	1.891e-04
45	GO:BP	GO:0033045	regulation of sister chromatid segregation	81	10	2.038e-04
46	GO:BP	GO:0044839	cell cycle G2/M phase transition	272	17	2.133e-04
47	GO:BP	GO:0051784	negative regulation of nuclear division	62	9	2.188e-04
48	GO:BP	GO:1905818	regulation of chromosome separation	63	9	2.524e-04
49	GO:BP	GO:0008608	attachment of spindle microtubules to kinetochore	33	7	4.048e-04
50	GO:BP	GO:0000086	G2/M transition of mitotic cell cycle	254	16	4.639e-04
51	GO:BP	GO:0051304	chromosome separation	91	10	6.234e-04
52	GO:BP	GO:0031577	spindle checkpoint	36	7	7.655e-04
53	GO:BP	GO:0007094	mitotic spindle assembly checkpoint	36	7	7.655e-04
54	GO:BP	GO:0071174	mitotic spindle checkpoint	36	7	7.655e-04
55	GO:BP	GO:0071173	spindle assembly checkpoint	36	7	7.655e-04
56	GO:BP	GO:0030071	regulation of mitotic metaphase/anaphase transition	53	8	8.457e-04
57	GO:BP	GO:1902099	regulation of metaphase/anaphase transition of cell cycle	55	8	1.136e-03
58	GO:BP	GO:0051649	establishment of localization in cell	2225	55	2.460e-03
59	GO:BP	GO:1903046	meiotic cell cycle process	190	13	2.645e-03
60	GO:BP	GO:0051225	spindle assembly	107	10	2.849e-03

g:Profiler (biit.cs.ut.ee/gprofiler)

C.6 Cyclic Genes from the Aggregated Results

Table C.19: Marker genes ranked by correlation across datasets.

Gene Symbol	Mean Loading	Mean Correlation	Mean Peaktime	Rank
NUSAP1	3.58	0.99	21.82	1
BUB1	4.22	0.99	21.45	2
G2E3	4.30	0.99	22.29	3
PIMREG	3.58	0.99	22.40	4
KIF20A	2.33	0.99	22.13	5
EXO1	1.35	0.98	16.00	7
CCNB1	5.51	0.98	22.47	8
AURKA	5.25	0.98	22.13	10
NDC80	3.32	0.98	21.73	11
MCM6	2.97	0.98	15.18	12
PLK1	4.65	0.98	22.23	13
HMMR	3.83	0.98	21.90	14
NUF2	3.11	0.97	22.05	15
CCNE2	3.53	0.97	15.71	17
CENPF	5.56	0.97	22.31	20
CENPA	3.96	0.97	22.47	21
TUBB4B	2.58	0.96	21.65	22
CDC20	4.45	0.96	22.61	23
DTL	4.30	0.96	15.55	24
KIF2C	3.05	0.96	22.22	27
TIPIN	1.26	0.96	16.35	28
CCNA2	3.96	0.96	21.62	30
RACGAP1	2.59	0.96	21.58	31
GTSE1	4.47	0.95	22.10	32
KIF11	2.19	0.95	22.11	33
FEN1	3.61	0.95	14.01	35
MCM5	3.58	0.95	14.88	36
CENPE	3.36	0.95	22.37	38

Continued on next page

Table C.19: Marker genes ranked by correlation across datasets.

Gene Symbol	Mean Loading	Mean Correlation	Mean Peaktime	Rank
CHAF1B	3.79	0.95	15.22	39
DLGAP5	4.05	0.95	22.46	40
CCNB2	3.32	0.94	22.19	42
RAD21	1.89	0.94	22.39	43
PSRC1	3.81	0.94	22.05	44
PCNA	4.30	0.94	14.51	45
E2F1	2.85	0.94	15.49	46
CDK1	4.02	0.94	21.55	48
CDC6	2.86	0.94	14.80	49
CDKN3	3.36	0.93	22.50	51
ANP32E	2.09	0.93	21.41	52
UHRF1	2.89	0.93	15.12	53
UBE2C	5.09	0.93	22.07	54
PTTG1	3.01	0.93	22.88	55
CKAP2L	2.45	0.93	22.02	57
CLSPN	4.96	0.92	14.97	59
UNG	2.81	0.92	14.57	60
CKAP2	3.09	0.92	22.22	61
SLBP	2.62	0.91	15.00	65
AURKB	4.61	0.91	22.14	67
CDCA3	3.47	0.91	22.21	69
CBX5	2.83	0.91	19.09	70
GINS2	3.19	0.90	15.76	71
TOP2A	6.54	0.90	21.69	77
NEK2	3.91	0.90	22.38	79
WDR76	3.28	0.89	14.57	80
TPX2	4.22	0.89	22.19	85
CKS2	4.01	0.87	22.17	96
TYMS	2.81	0.85	15.92	102
MKI67	5.21	0.85	21.89	104

Continued on next page

Table C.19: Marker genes ranked by correlation across datasets.

Gene Symbol	Mean Loading	Mean Correlation	Mean Peaktime	Rank
RPA2	1.67	0.84	14.05	106
HMGB2	3.71	0.83	21.81	107
CKS1B	3.41	0.81	22.09	117
HELLS	3.57	0.79	12.89	120
CDC45	1.94	0.75	15.00	127
TACC3	2.58	0.74	22.01	128
HJURP	2.63	0.71	19.96	134
CDCA7	3.25	0.70	14.86	136
SMC4	3.73	0.70	21.28	137
RFC2	2.40	0.70	14.93	138

Table C.20: New candidate genes classified as S by mean peaktime across datasets, ranked by correlation across datasets.

Gene Symbol	Mean Loading	Mean Correlation	Mean Peaktime	Rank
HPRT1	1.58	0.96	17.19	25
ORC6	2.12	0.96	17.11	29
MCM10	2.48	0.94	15.74	50
RABIF	1.28	0.92	15.38	62
MCM3	3.16	0.91	14.91	68
PSMC3IP	2.13	0.90	14.73	72
PCLAF	4.01	0.90	14.63	76
BTG3	1.70	0.89	14.74	81
VRK1	1.90	0.89	18.55	82
ACD	1.82	0.89	16.14	83
UBE2T	2.91	0.89	18.59	84
CHAF1A	3.44	0.88	13.51	89

Continued on next page

Table C.20: New candidate genes classified as S by mean peaktime across datasets, ranked by correlation across datasets.

Gene Symbol	Mean Loading	Mean Correlation	Mean Peaktime	Rank
NUDT8	2.06	0.88	13.82	90
UROS	1.34	0.88	14.39	91
DNAJC9	2.62	0.88	15.74	92
RFC4	2.41	0.87	14.79	93
IER3IP1	1.38	0.87	16.82	94
CARHSP1	2.46	0.87	15.75	95
CYB5R3	1.20	0.87	13.89	97
NME4	1.63	0.85	11.94	100
CDT1	2.97	0.85	13.85	101
NAE1	1.25	0.85	17.51	103
EMG1	1.44	0.85	16.49	105
RAD51C	1.73	0.82	18.75	112
HAUS8	1.38	0.82	13.16	113
HSPB11	1.54	0.82	12.97	114
DCTPP1	2.13	0.81	13.27	115
DUT	2.98	0.81	13.77	116
ACYP1	1.89	0.78	14.01	122
RAD1	1.94	0.77	17.79	124
GEMIN4	1.47	0.73	13.52	129
ASF1B	2.46	0.73	13.75	130
ITGB1BP1	1.68	0.73	14.45	131
UBA2	1.61	0.70	18.39	135

Table C.21: New candidate genes classified as G2/M by mean peaktime across datasets, ranked by correlation across datasets.

Gene Symbol	Mean Loading	Mean Correlation	Mean Peaktime	Rank
UBALD2	2.17	0.99	22.32	6
ARL6IP1	4.37	0.98	22.35	9
REEP4	1.58	0.97	21.89	16
SGO2	3.10	0.97	22.10	18
KNSTRN	2.92	0.97	22.09	19
VDAC3	1.61	0.96	19.10	26
ZWINT	2.35	0.95	19.84	34
MAD2L1	2.10	0.95	21.48	37
KPNA2	3.45	0.95	22.13	41
NAA50	1.70	0.94	19.38	47
DEPDC1B	1.83	0.93	21.56	56
SPC25	2.10	0.93	21.65	58
ASPM	4.01	0.91	22.47	63
PRR11	2.91	0.91	22.18	64
BUB3	2.62	0.91	21.38	66
NELFE	1.70	0.90	21.81	73
CAPZA1	1.43	0.90	21.09	74
PBK	2.87	0.90	21.59	75
DBF4	2.38	0.90	21.77	78
HP1BP3	1.94	0.89	22.00	86
SGO1	2.10	0.89	22.12	87
BRD8	1.55	0.89	21.54	88
PSIP1	1.79	0.86	19.13	98
GPSM2	2.13	0.85	22.13	99
PLIN3	1.33	0.83	23.54	108
COPS3	1.67	0.83	19.16	109
CCDC88A	1.51	0.83	22.33	110
HNRNPR	1.36	0.83	20.04	111
TRABD	1.25	0.80	24.47	118

Continued on next page

Table C.21: New candidate genes classified as G2/M by mean peaktime across datasets, ranked by correlation across datasets.

Gene Symbol	Mean Loading	Mean Correlation	Mean Peaktime	Rank
ARHGAP11A	2.53	0.80	22.31	119
SRRT	1.35	0.79	19.34	121
KIF22	2.13	0.78	21.95	123
H2AFX	2.75	0.76	21.65	125
DESI2	1.45	0.76	22.76	126
AATF	1.31	0.73	21.62	132
UPF3A	1.42	0.72	21.13	133

Appendix D

Code snippets

D.1 Reformatting of CellRanger 1.0.0 fastq Format

The following code was used to reformat fastq files generated by CellRanger 1.0.0.

```
#!/bin/python

import dinopy
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("I1")
parser.add_argument("RA")
parser.add_argument("R1_out")
parser.add_argument("R2_out")
args = parser.parse_args()

ra_reader = dinopy.FastqReader(args.RA)

illumina_index = args.RA.split('_')[1].split('-')[1]

reads = []
umi = []
count = 0
for sequence, name, quality in ra_reader.reads(quality_values=True):
    if count%2 == 0:
        reads.append((sequence,name.replace(b'1:N:0:0','2:N:0:{}'.format(illumina_index).encode()),quality))
    else:
        umi.append((sequence, name, quality))
    count += 1
```

```
with dinopy.FastqWriter(args.R2_out) as r2_writer:
    r2_writer.write_reads(reads)

r1_out = []
I1_reader = dinopy.FastqReader(args.I1)
for bc_reads, umi_reads in zip(I1_reader.reads(quality_values=True), umi):
    r1_out.append((bc_reads[0]+umi_reads[0],bc_reads[1].replace(b'2:N:0:0','1:N:0:{}'.format(illumina_index).encode()),bc_re

with dinopy.FastqWriter(args.R1_out) as r1_writer:
    r1_writer.write_reads(r1_out)
```

D.2 Reference Index for Cellranger

The following build script is used to build a joint reference index of Homo sapiens GRCh38.94 and Mus musculus GRCm38.94 to be used with the `cellranger count` workflow.

```
wget https://ftp.ensembl.org/pub/release-94/gtf/homo_sapiens/Homo_sapiens.GRCh38.94.gtf.gz
gunzip Homo_sapiens.GRCh38.94.gtf.gz


wget https://ftp.ensembl.org/pub/release-94/fasta/mus_musculus/dna/Mus_musculus.GRCm38.dna.primary_assembly.fa.gz
gunzip Mus_musculus.GRCm38.dna.primary_assembly.fa.gz


wget https://ftp.ensembl.org/pub/release-94/gtf/mus_musculus/Mus_musculus.GRCm38.94.gtf.gz
gunzip Mus_musculus.GRCm38.94.gtf.gz


cellranger mkgtf Homo_sapiens.GRCh38.94.gtf Homo_sapiens.GRCh38.94.filtered.gtf \
                 --attribute=gene_biotype:protein_coding \
                 --attribute=gene_biotype:lincRNA \
                 --attribute=gene_biotype:antisense \
                 --attribute=gene_biotype:IG_LV_gene \
                 --attribute=gene_biotype:IG_V_gene \
                 --attribute=gene_biotype:IG_V_pseudogene \
                 --attribute=gene_biotype:IG_D_gene \
                 --attribute=gene_biotype:IG_J_gene \
                 --attribute=gene_biotype:IG_J_pseudogene \
                 --attribute=gene_biotype:IG_C_gene \
                 --attribute=gene_biotype:IG_C_pseudogene \
                 --attribute=gene_biotype:TR_V_gene \
                 --attribute=gene_biotype:TR_V_pseudogene \
```

```
                --attribute=gene_biotype:TR_D_gene \
                --attribute=gene_biotype:TR_J_gene \
                --attribute=gene_biotype:TR_J_pseudogene \
                --attribute=gene_biotype:TR_C_gene


cellranger mkgtf Mus_musculus.GRCm38.94.gtf Mus_musculus.GRCm38.94.filtered.gtf \
                --attribute=gene_biotype:protein_coding \
                --attribute=gene_biotype:lincRNA \
                --attribute=gene_biotype:antisense \
                --attribute=gene_biotype:IG_LV_gene \
                --attribute=gene_biotype:IG_V_gene \
                --attribute=gene_biotype:IG_V_pseudogene \
                --attribute=gene_biotype:IG_D_gene \
                --attribute=gene_biotype:IG_J_gene \
                --attribute=gene_biotype:IG_J_pseudogene \
                --attribute=gene_biotype:IG_C_gene \
                --attribute=gene_biotype:IG_C_pseudogene \
                --attribute=gene_biotype:TR_V_gene \
                --attribute=gene_biotype:TR_V_pseudogene \
                --attribute=gene_biotype:TR_D_gene \
                --attribute=gene_biotype:TR_J_gene \
                --attribute=gene_biotype:TR_J_pseudogene \
                --attribute=gene_biotype:TR_C_gene


cellranger mkref --genome=GRCh38 \
                --fasta=Homo_sapiens.GRCh38.dna.primary_assembly.fa \
                --genes=Homo_sapiens.GRCh38.94.filtered.gtf \
                --genome=GRCm38 \
                --fasta=Mus_musculus.GRCm38.dna.primary_assembly.fa \
                --genes=Mus_musculus.GRCm38.94.filtered.gtf \
                --nthreads=48 \
                --memgb=80 \
                --ref-version=3.0.2
```

D.3 Conversion to AnnData object

The following script is used to convert count matrices from CellRangers count algorithm or STARsolo algorithm to a AnnData object. The AnnData object can loaded directly into a Scanpy environment for analysis.

```
#!/usr/bin/env python
```

```
import warnings
warnings.filterwarnings("ignore", message="numpy.dtype size changed")

import sys
import os
import argparse

import scanpy  as sc
import pandas as pd
import numpy as np
from vpolo.alevin import parser as alevin_parser

GENOME = {'homo_sapiens': 'GRCh38',
          'human': 'GRCh38',
          'hg38': 'GRCh38',
          'GRCh38': 'GRCh38',
          'mus_musculus': 'mm10',
          'mouse': 'mm10',
          'mm10': 'mm10',
          'GRCm38': 'mm10'
}

parser = argparse.ArgumentParser(description=__doc__,
    formatter_class=argparse.RawDescriptionHelpFormatter
    )

parser.add_argument('input',
    help='input file(s)',
    nargs='+'
    )
parser.add_argument('-o',
    '--outfile',
    help='output filename',
    required=True
    )
parser.add_argument('-f',
    '--input-format',
    choices=['cellranger_aggr', 'cellranger',
    'star', 'alevin', 'umitools'],
    default='cellranger_aggr',
    help='input file format'
    )
parser.add_argument('-F',
    '--output-format',
    choices=['anndata', 'loom', 'csvs'],
    default='anndata',
    help='output file format'
    )
parser.add_argument('--sample-info',
```

```
    help='samplesheet info, tab seprated file assumes `Sample_ID` in header',
    default=None
    )
parser.add_argument('--feature-info',
    help='extra feature info filename, tab seprated file assumes `gene_id` in header',
    default=None
    )
parser.add_argument('--genome',
    help='reference genome',
    default=None
    )
parser.add_argument('--filter-org',
    help='filter data (genes) by organism',
    default=None
    )
parser.add_argument('--gex-only',
    help='only keep `Gene Expression` data and ignore other feature types.',
    default=True
    )
parser.add_argument('--normalize',
    help='normalize depth across the input libraries',
    default='none',
    choices=['none', 'mapped']
    )
parser.add_argument('--batch',
    help='column name in `sample-info` with batch covariate',
    default=None
    )
parser.add_argument('--no-zero-cell-rm',
    help='do not remove cells with zero counts',
    action='store_true'
    )
parser.add_argument('-v ',
    '--verbose',
    help='verbose output.',
    action='store_true'
    )


def downsample_gemgroup(data_list):
    """downsample data total read count to gem group with lowest total count
    """
    min_count = 1E99
    sampled_list = []
    for i, data in enumerate(data_list):
        isum = data.X.sum()
        if isum < min_count:
            min_count = isum
            idx = i
```

```
    for j, data in enumerate(data_list):
        if j != idx:
            sc.pp.downsample_counts(data, target_counts = min_count)
        sampled_list.append(data)
    return sampled_list

def read_cellranger(fn, args, rm_zero_cells=True, **kw):
    """read cellranger results

    Assumes the Sample_ID may be extracted from cellranger output dirname,
    e.g ` ... /Sample_ID/outs/filtered_feature_bc_matrix.h5 `
    """
    if fn.endswith('.h5'):
        if not args.genome:
            raise ValueError('loading a .h5 file in scanpy requires the `genome` parameter')
        dirname = os.path.dirname(fn)
        data = sc.read_10x_h5(fn, genome=GENOME[args.genome])
        data.var['gene_symbols'] = list(data.var_names)
        data.var_names = list(data.var['gene_ids'])
    else:
        mtx_dir = os.path.dirname(fn)
        dirname = os.path.dirname(mtx_dir)
        data = sc.read_10x_mtx(mtx_dir,
                gex_only=args.gex_only,
                var_names='gene_ids',
                make_unique=True
                )
        data.var['gene_ids'] = list(data.var_names)
    sample_id = os.path.basename(os.path.dirname(dirname))
    data.obs['library_id'] = [sample_id] * data.obs.shape[0]
    barcodes = [b.split('-')[0] for b in data.obs.index]
    if len(barcodes) == len(set(barcodes)):
        data.obs_names = barcodes
    data.obs.index.name = 'barcodes'
    data.var.index.name = 'gene_id'

    return data

def read_cellranger_aggr(fn, args, **kw):
    data = read_cellranger(fn, args)
    dirname = os.path.dirname(fn)
    if not fn.endswith('.h5'):
        dirname = os.path.dirname(dirname)

    # fix cellranger aggr enumeration to start at 0 (matches scanpy enum)
    barcodes =  [i[0] for i in data.obs.index.str.split('-')]
    barcode_enum = [int(i[1])-1 for i in data.obs.index.str.split('-')]
    data.obs_names = ['-'.join(e) for e in zip(barcodes, barcodes_enum)]
```

```
    aggr_csv = os.path.join(dirname, 'aggregation.csv')
    if os.path.exists(aggr_csv):
        aggr_csv = pd.read_csv(aggr_csv)
        sample_map = dict((i, n) for i,n in enumerate(aggr_csv['library_id']))
        samples = [sample_map[i] for i in barcode_enum]
        data.obs['library_id'] = samples

    return data

def read_star(fn, args, **kw):
    mtx_dir = os.path.dirname(fn)
    dirname = os.path.dirname(mtx_dir)
    data = sc.readwrite._read_legacy_10x_mtx(mtx_dir,
            var_names='gene_ids',
            make_unique=True
            )
    data.var['gene_ids'] = list(data.var_names)
    sample_id = os.path.basename(dirname)
    data.obs['library_id'] = [sample_id] * data.obs.shape[0]
    barcodes = [b.split('-')[0] for b in data.obs.index]
    if len(barcodes) == len(set(barcodes)):
        data.obs_names = barcodes

    return data


def read_alevin(fn, args, **kw):
    avn_dir = os.path.dirname(fn)
    dirname = os.path.dirname(avn_dir)
    if fn.endswith('.gz'):
        df = alevin_parser.read_quants_bin(input_dir)
    else:
        df = alevin_parser.read_quants_csv(input_dir)
    row = {'row_names': df.index.values.astype(str)}
    col = {'col_names': np.array(df.columns, dtype=str)}
    data = AnnData(df.values, row, col, dtype=np.float32)
    data.var['gene_ids'] = list(data.var_names)
    sample_id = os.path.basename(dirname)
    data.obs['library_id'] = [sample_id] * data.obs.shape[0]

    return data

def read_umitools(fn, **kw):
    raise NotImplementedError

READERS = {'cellranger_aggr': read_cellranger_aggr,
    'cellranger': read_cellranger,
    'star': read_star,
    'umitools': read_umitools,
```

```
    'alevin': read_alevin
    }

if __name__ == '__main__':
    args = parser.parse_args()

    reader = READERS.get(args.input_format.lower())
    if reader is None:
        raise ValueError('{} is not a supported input format'.format(args.input_format))
    for fn in args.input:
        if not os.path.exists(fn):
            raise IOError('file does not exist! {}'.format(fn))
    n_input = len(args.input)
    if n_input > 1:
        assert(args.input_format != 'cellranger_aggr')

    if args.sample_info is not None:
        sample_info = pd.read_csv(args.sample_info, sep='\t')
        if not 'Sample_ID' in sample_info.columns:
            raise ValueError('sample_sheet needs a column called `Sample_ID`')
        sample_info.index = sample_info['Sample_ID']
        if args.batch is not None:
            batch_categories = sample_info[batch].astype('category')
    else:
        sample_info = None
        if args.batch is not None:
            raise ValueError('cannot use option `batch` when option `--sample-info` not used')
        batch_categories = None

    if args.feature_info is not None:
        feature_info = pd.read_csv(args.feature_info, sep='\t')
        if not 'gene_id' in feature_info.columns:
            raise ValueError('feature_info needs a column called `gene_id`')

        feature_info.index = feature_info['gene_id']
    else:
        feature_info = None

    data_list = []
    for i, fn in enumerate(args.input):
        fn = os.path.abspath(fn)
        data = reader(fn, args)
        data_list.append(data)

    if len(data_list) > 1:
        if args.normalize == 'mapped':
            data_list = downsample_gemgroup(data_list)

    data = data_list.pop(0)
```

```
if len(data_list) > 0:
    data = data.concatenate(*data_list, batch_categories=batch_categories)
    # clean up feature info (assumes inner join)
    # fixme: maybe outer join support is what we want ?
    if any(i.endswith('-0') for i in data.var.columns):
        keep = [i for i in data.var.columns if i.endswith('-0')]
        data.var  = data.var.loc[:,keep]
        data.var.columns = [i.split('-')[0] for i in data.var.columns]
if sample_info:
    lib_ids = set(data.obs['library_id'])
    for l in lib_ids:
        if l not in sample_info.index:
            raise ValueError('Library `{}` not present in sample_info'.format(l))
    obs = sample_info.loc[data.obs['library_id'],:]
    obs.index = data.obs.index.copy()
    data.obs = data.obs.merge(obs, how='left', on='barcode', copy=False)

if not args.no_zero_cell_rm:
    keep = data.X.sum(1).A.squeeze() > 0
    data = data[keep,:]
    keep = data.X.sum(0).A.squeeze() > 0
    #keep = (data.X != 0).any(axis=0)
    data = data[:,keep]

if feature_info:
    data.var = data.var.merge(feature_info, how='left', on='gene_id', copy=False)

if 'gene_symbols' in data.var.columns:
    mito_genes = data.var.gene_symbols.str.startswith('MT-')
    data.obs['fraction_mito'] = np.sum(data[:, mito_genes].X, axis=1).A1 / np.sum(data.X, axis=1).A1
data.obs['n_counts'] = data.X.sum(axis=1).A1

if args.verbose:
    print(data)
    print(data.X.A.sum())

if args.output_format == 'anndata':
    data.write(args.outfile)
elif args.output_format == 'loom':
    data.write_loom(args.outfile)
elif args.output_format == 'csvs':
    data.write_csvs(args.outpfile)
```

D.4 Modified Cell Cycle Scoring

The following method is a modified version of the scanpy method `scanpy.tl.score_genes_genes_cell_c` The method is modified to take an additional list of G1 marker genes to use in the cell cycle scoring. The classification is performed by the same logic as in the original method. In addition, the parameteres used in the calls to the underlying method `scanpy.tl.score_genes()` are tweaked to account for a low number of total genes in the data set.

```
def my_score_genes_cell_cycle_improved(
        adata,
        g1_genes,
        s_genes,
        g2m_genes,
        copy=False,
        **kwargs):
    """Score cell cycle genes [Satija15]_.
    Given two lists of genes associated to S phase and G2M phase, calculates
    scores and assigns a cell cycle phase (G1, S or G2M). See
    :func:`~scanpy.api.score_genes` for more explanation.
    Parameters
    ----------
    adata : :class:`~anndata.AnnData`
        The annotated data matrix.
    g1_genes : `list`
        List of genes associated with S phase.
    s_genes : `list`
        List of genes associated with S phase.
    g2m_genes : `list`
        List of genes associated with G2M phase.
    copy : `bool`, optional (default: `False`)
        Copy `adata` or modify it inplace.
    **kwargs : optional keyword arguments
        Are passed to :func:`~scanpy.api.score_genes`. `ctrl_size` is not
        possible, as it's set as `min(len(s_genes), len(g2m_genes))`.
    Returns
    -------
    Depending on `copy`, returns or updates `adata` with the following fields.
    **G1_score** : `adata.obs`, dtype `object`
        The score for G1 phase for each cell.
    **S_score** : `adata.obs`, dtype `object`
        The score for S phase for each cell.
    **G2M_score** : `adata.obs`, dtype `object`
        The score for G2M phase for each cell.
    **phase** : `adata.obs`, dtype `object`
```

```
    The cell cycle phase (`S`, `G2M` or `G1`) for each cell.
See also
--------
score_genes
Examples
--------
See this `notebook <https://github.com/theislab/scanpy_usage/tree/master/180209_cell_cycle>`__.
"""
#logg.info('calculating cell cycle phase')
import scanpy as sc

adata = adata.copy() if copy else adata
ctrl_size = min(len(s_genes), len(g2m_genes), len(g1_genes))
s_n_bins = round((len(g1_genes)+len(g2m_genes))/ctrl_size)
g1_n_bins = round((len(s_genes)+len(g2m_genes))/ctrl_size)
g2m_n_bins = round((len(g1_genes)+len(s_genes))/ctrl_size)

# add s-score
sc.tl.score_genes(adata, gene_list=s_genes, score_name='S_score', ctrl_size=ctrl_size, n_bins=s_n_bins, **kwargs)
# add g2m-score
sc.tl.score_genes(adata, gene_list=g2m_genes, score_name='G2M_score', ctrl_size=ctrl_size, n_bins=g2m_n_bins, **kwargs)
# add g1-score
sc.tl.score_genes(adata, gene_list=g1_genes, score_name='G1_score', ctrl_size=ctrl_size, n_bins=g1_n_bins, **kwargs)

if not 'G1_score' in adata.obs.columns:
    print("WARNING: No G1-genes found in data set. Computing G1-score as -sum(S_score,G2M_score)")
    adata.obs['G1_score'] = -adata.obs[['S_score', 'G2M_score']].sum(1)

scores = adata.obs[['S_score', 'G2M_score', 'G1_score']]

# default phase is S
phase = pd.Series('not_assigned', index=scores.index)

# if G2M is higher than S and G1, it's G2M
phase[(scores.G2M_score > scores.S_score) & (scores.G2M_score > scores.G1_score)] = 'G2M'

# if S is higher than G2M and G1, it's S
phase[(scores.S_score > scores.G2M_score) & (scores.S_score > scores.G1_score)] = 'S'

# if G1 is higher than G2M and S, it's G1
phase[(scores.G1_score > scores.G2M_score) & (scores.G1_score > scores.S_score)] = 'G1'



adata.obs['phase'] = phase
#logg.hint('    \'phase\', cell cycle phase (adata.obs)')
return adata if copy else None
```

D.5 Generating Cubic Smoothing Splines

The following code is used to generate the cubic smoothing splines from gene expression data from a pseudotime ordering.

```
def subsample_gene(data, gene_name, width=2):
    gene_cat = 'gene_ids' if gene_name.startswith("ENS") else 'gene_symbols'
    if not gene_name in list(data.var[gene_cat]):
        raise ValueError("Gene symbol {} not in data set!".format(gene_name))
    data_test = data[:,data.var[gene_cat] == gene_name]
    m_arr = []
    for i in range(0,24,width):
        m_arr.append(data_test[ (i <= data_test.obs['time']) & (data_test.obs['time'] < i+2),:].X.mean())
    return np.array(m_arr)

def generate_cyclic_control_points(data, gene_name, width=2, smooth=False, smooth_its=1):

    subsample = subsample_gene(data,gene_name,width=width)
    boundary = (subsample[0]+subsample[-1])/2
    x = np.append([np.arange(width/2,24,width)], [subsample], axis=0).T

    x = np.vstack((np.array([0,boundary]), x,np.array([24,boundary])))

    if smooth:
        for k in range(smooth_its):
            xnew = x.copy()
            for i in range(1,len(x)-1):
                xnew[i] = (0.5*x[i-1] + x[i] + 0.5*x[i+1])*0.5
            x = xnew

    x0 = x[1:-1].copy()
    x1 = x[1:-1].copy()

    x0[:,0] -= np.max(x[:,0])

    x1[:,0] += np.max(x[:,0])

    return np.vstack((x0,x,x1)), x

def spline_model2_csaps(data,gene_name,width=2,csaps_smooth=0.3,plot_figs=False,add_boundaries=True):

    import csaps
    if add_boundaries:
        X, x = generate_cyclic_control_points(data, gene_name, width=width, smooth=False)
    else:
        X, x = generate_cyclic_control_points_no_boundaries(data, gene_name, width=width, smooth=False)
```

```
    sp = csaps.UnivariateCubicSmoothingSpline(X[:,0], X[:,1], smooth=csaps_smooth)

    if plot_figs:
        xs = np.linspace(0,24,1000)
        y = sp(xs)
        plt.figure(figsize=(7,7))
        plt.plot(xs,y)
        plt.plot(x[:,0],x[:,1],'*')
        plt.axhline(y=0, color='k')
        plt.xticks(np.arange(0,26,4))

        plt.show()

    return sp, x
```

D.6 Optimal Smoothing Parameter

The following code is used to determine the optimal smoothing parameter to be used with the cubic smoothing splines for a given data set. The genes used to generate the function values are the marker genes from Appendix C.2 labeled whitfield_2002. We use the code listed in Appendix D.5 to generate the splines.

```
def csaps_error(data,ens_list,csaps_smooth=0.3,subsample_width=2):
    ens_list = list(set(list(ens_list)).intersection(list(data.var.index)))
    res = {}
    for i, gene in enumerate(sorted(ens_list)):
        sp, ctr_x = spline_model2_csaps(data,gene,width=subsample_width,csaps_smooth=csaps_smooth,plot_figs=False)

        y_raw = data[:,[gene]].X
        y_est = sp(data.obs['time'])
        res[gene] = {'res_2-norm': np.linalg.norm(y_raw-y_est,axis=0),
                'res_mean': np.mean(abs(y_est-y_raw)),
                'res_var': np.var(abs(y_est-y_raw),ddof=1),
                'sum_curvature': np.sum(csaps_curvature(data,sp=sp)),
                'gene_symbol': data.var.loc[gene]['gene_symbols']}

    return res

def csaps_curvature(data,sp=None,gene=None,width=2,csaps_smooth=0.3):
    if not sp:
        if not gene:
            raise ValueError("If not a csaps spline is defined (sp=) a gene must be specified with gene=")
        sp, x = spline_model2_csaps(data,gene=gene,widht=width,csaps_smooth=csaps_smooth)
```

```
    time = data.obs['time']
    ys = sp(time)
    grady = np.gradient(ys,time)
    grad2y = np.gradient(grady,time)
    denom = np.power(1+np.power(grady,2),1.5)

    return abs(grad2y)/denom

def find_opt_smooth(data_bb,plot=False):
    report_df = pd.read_csv("./data/knownGenes_derived.txt", sep=" ")
    mean_norm = []
    mean_sum_curvature = []
    for i in np.arange(0,1.05,0.05):
        print("Calculating for smooth parameter = {}".format(i))
        for p in ['S','G2/M','G2','G1']:
            residuals = {}
            p_df = report_df[report_df['phase'] == p]
            ens_list = list(set(list(p_df['Ensembl'])).intersection(list(data_bb.var.index)))
            r = mscu.csaps_error(data_bb, ens_list, csaps_smooth=i, subsample_width=2)
            residuals.update(r)
        res_df = pd.DataFrame.from_dict(residuals,orient='index')
        mean_norm.append(res_df['res_2-norm'].mean())
        mean_sum_curvature.append(res_df['sum_curvature'].mean())

    i = np.arange(0,1.05,0.05)
    #scale
    scaled_mean_norm = np.array([x - np.min(mean_norm) for x in mean_norm])
    scaled_mean_norm /= np.max(scaled_mean_norm)
    scaled_mean_curve = np.array([x - np.min(mean_sum_curvature) for x in mean_sum_curvature])
    scaled_mean_curve /= np.max(scaled_mean_curve)

    #plot
    if plot:
        plt.figure(figsize=(9,9))
        plt.plot(i,scaled_mean_norm,label='r: scaled 2-norm error')
        plt.plot(i,scaled_mean_curve,label='c: scaled curvature')
        plt.plot(i,scaled_mean_curve+scaled_mean_norm,label='g = r + c')
        plt.xlabel("Smoothing parameter")
        plt.ylabel("Objective function")
        plt.legend()
        plt.savefig("./figures/objective_function.png")

    #solve
    from scipy.interpolate import interp1d
    t = np.arange(0,1,0.01)
    intp = interp1d(i,scaled_mean_curve+scaled_mean_norm)
    opt_smooth = t[np.where(intp(t) == np.min(intp(t)))[0][0]]
```

```
    return opt_smooth
```

D.7 Partial Least Squares Regression

The following code is used for the Partial Least Squares Regression of the subsampled spline data and for resampling and identification of a significance cut-off value. The PLS implementation `miirPLS` was developed for the study [Pen+13].

```
X <- read.csv("../data/final_hacat/subsampled_splines.csv", row.names=1)
Y <- read.csv("../data/final_hacat/PLS/PLS_Y.csv", row.names=1)

options(warn=-1)
retList = miirPLS(X,Y,ModifiedTtest = 1)
options(warn=0)



options(warn=-1)
#n resamples
n = 1000
#dX number of variables
dX = ncol(X)
#alpha significance level
alpha = 0.05
#nCut cutoff corresponding to alpha
nCut = ceiling(dX*alpha)
R <- matrix(0, dX, n)
rCut <- vector("numeric", length = n)
for (j in 1:n){
    #permute Y
    Yperm <- Y[sample(nrow(Y)),]
    resampleRetList = miirPLS(X, Yperm, ModifiedTtest = 1)
    for (i in 1:dX){
        #compute distance R from origin in PLS-loading plane
        R[i,j] = sqrt(sum(resampleRetList$model$sigLoadings[i,]^2))
    }
    r = sort(R[,j], decreasing = TRUE)
    rCut[j] = r[nCut]
}
options(warn=0)

write.csv(retList$q,
    file="../data/final_hacat/PLS/q.csv",
    row.names=TRUE
    )
```

```
write.csv(retList$model$sigLoadings,
    file="../data/final_hacat/PLS/model_sigloadings.csv",
    row.names=TRUE
    )
write.csv(retList$model$scores,
    file="../data/final_hacat/PLS/model_scores.csv",
    row.names=TRUE
    )
write.csv(retList$model$loadings,
    file="../data/final_hacat/PLS/model_loadings.csv",
    row.names=TRUE
    )
write.csv(retList$model$loadingWeights,
    file="../data/final_hacat/PLS/model_loadingweights.csv",
    row.names=TRUE
    )
write.csv(retList$model$sigLoadings,
    file="../data/final_hacat/PLS/model_sigloadings.csv",
    row.names=TRUE
    )
write.csv(rCut,
    file="../data/final_hacat/PLS/r_cutoff.csv"
    )

miirPLS = function (X, Y, n = NULL, ModifiedTtest = 0.5, sets = NULL, aMax = 2,
    method = "kernel", ridge = 0.1, scrambleSets = F)
{
    library(MASS)
    Y <- as.matrix(Y)
    X <- scale(X, scale = FALSE)
    Y <- scale(Y, scale = FALSE)
    dSets <- length(names(sets))
    if (method == "bridge") {
        mainMod <- calibrate.bridgePLS(X, Y, aMax = aMax, ridge = ridge)
    }
    if (method == "kernel") {
        mainMod <- calibrate.kernelPLS(X, Y, aMax = aMax)
    }
    W <- mainMod$sigLoadings
    t2 <- crossvalidatePLS(W, X, Y, ModifiedTtest, penalty = NULL,
        aMax = aMax, method = method, ridge = ridge)
    Wjk <- t2$Wjk
    Yval <- t2$Yval
    if (!is.null(sets)) {
        for (set in names(sets)) {
            sets[[set]] <- intersect(sets[[set]], colnames(X))
        }
        sets <- sets[unlist(lapply(sets, length) > 3)]
        setStats <- setScores(W, sets, t2$t2, T2null = NULL)
```

```
    TsetReal <- setStats$setT
    print("sets computed")
}
penalty <- t2$penalty
t2 <- t2$t2
names(t2) <- colnames(X)
if (!is.null(n)){
    print("Starting resampling")
    qv <- resamplePLS(W, t2, X, Y, TsetReal, sets, n, ModifiedTtest,
        penalty = penalty, method = method, ridge = ridge, aMax = aMax,
        scrambleSets = scrambleSets)
    print("Resampling finished")
    Tpert = qv$Tpert
    qSet = qv$qSet
    qv = qv$qv
}
else {
    Tpert = NULL
    qSet = NULL
    qv = NULL
}
if (is.null(sets)) {
    Wset <- NULL
    setT <- NULL
}
else {
    Wset <- setStats$setW
    setT <- setStats$setT
}
if (method == "bridge") {
    expVarX <- diag(t(mainMod$scores) %*% mainMod$scores)/sum(diag(X %*%
        t(X)))
}
if (method == "kernel") {
    expVarX <- (colSums(mainMod$loadings * mainMod$loadings) *
        diag(crossprod(mainMod$scores)))/(sum(X^2))
}
dY <- dim(Y)
q2 <- vector("numeric", length = (dY[2] + 2))
for (k in 1:dY[2]) {
    q2[k] <- 1 - sum((Y[, k] - Yval[, k, aMax])^2)/sum((Y[,
        k]^2))
}
q2[dY[2] + 1] <- 1 - sum((Y - Yval[, , aMax])^2)/sum((Y)^2)
q2[dY[2] + 2] <- 1 - sum((mainMod$f)^2)/sum((Y)^2)
if (!is.null(qv)){
    names(qv) <- colnames(X)
}
return(list(T2 = t2, model = mainMod, q = qv, Wjk = Wjk,
```

```
        Tpert = Tpert, sets = sets, qSet = qSet, Wset = Wset,
        setT = setT, Yval = Yval, expVarX = expVarX, q2 = q2,
        X = scale(X, scale = F, center = -1 * attr(X, "scaled:center")),
        method = method, Y = Y))
}

calibrate.bridgePLS = function (X, Y, aMax = 3, ridge = 0.1)
{
    a <- aMax
    dY <- dim(Y)
    dX <- dim(X)
    b <- array(NA, dim = c(dX[2], dY[2], aMax))
    W <- matrix(0, dX[2], a)
    alpha <- 1
    G0 <- svd(Y)
    if (dY[2] == 1) {
        G0 <- G0$u %*% (G0$d) %*% t(G0$u)
    }
    else {
        G0 <- G0$u %*% diag(G0$d) %*% t(G0$u)
    }
    G <- (1 - ridge) * G0 + (ridge) * diag(dY[1])
    H <- t(X) %*% G
    usv <- svd(H)
    Va <- as.matrix(usv$u[, 1:a])
    Da <- X %*% Va
    qa <- t(Y) %*% Da %*% ginv(t(Da) %*% Da)
    f <- Y - Da %*% t(qa)
    Ua <- Y %*% qa
    Q0 <- usv$v
    sf <- alpha * t(Da) %*% Da + (1 - alpha) * t(Ua) %*% Ua
    for (i in 1:a) {
        W[, i] <- sqrt(diag(sf)[i]) * Va[, i]
        if (dY[2] == 1) {
            b[, , i] = Va[, 1:i] %*% (as.matrix(qa[, 1:i]))
        }
        else {
            b[, , i] = Va[, 1:i] %*% t(as.matrix(qa[, 1:i]))
        }
    }
    rownames(W) <- colnames(X)
    fullModel <- list(sigLoadings = W, scores = Da, loadings = Va,
        qa = qa, Ua = Ua, b = b, Q0 = Q0, f = f)
    return(fullModel)
}

calibrate.kernelPLS = function (X, Y, aMax = 3)
{
    Y <- as.matrix(Y)
```

```
a <- aMax
dY <- dim(Y)
dX <- dim(X)
R <- matrix(0, ncol = a, nrow = dX[2])
W <- matrix(0, ncol = a, nrow = dX[2])
P <- matrix(0, ncol = a, nrow = dX[2])
Q <- matrix(0, ncol = a, nrow = dY[2])
B <- array(0, c(dX[2], dY[2], a))
Yhat <- array(0, c(dX[1], dY[2], a))
U <- matrix(0, ncol = a, nrow = dX[1])
T <- matrix(0, ncol = a, nrow = dX[1])
XtY <- crossprod(X, Y)
for (a in 1:aMax) {
    if (dY[2] == 1) {
        w <- XtY/sqrt(c(crossprod(XtY)))
    }
    else {
        if (dY[2] < dX[2]) {
            q = eigen(crossprod(XtY), symmetric = TRUE)$vectors[,
              1]
            w <- XtY %*% q
            w <- w/sqrt(c(crossprod(w)))
        }
        else {
            w <- eigen(tcrossprod(XtY), symmetric = TRUE)$vectors[,
              1]
        }
    }
    r <- w
    if (a > 1) {
        for (j in 1:(a - 1)) {
            r <- r - (P[, j] %*% w) * R[, j]
        }
    }
    t <- X %*% r
    t2 <- c(crossprod(t))
    p <- crossprod(X, t)/t2
    q <- crossprod(XtY, r)/t2
    u <- Y %*% q
    XtY <- XtY - (t2 * p) %*% t(q)
    T[, a] <- t
    R[, a] <- r
    P[, a] <- p
    Q[, a] <- t(q)
    B[, , a] <- R[, 1:a, drop = FALSE] %*% t(Q[, 1:a, drop = FALSE])
    U[, a] <- u
    W[, a] <- w
    Yhat[, , a] <- T[, 1:a] %*% t(Q[, 1:a, drop = FALSE])
}
```

```
    f <- Y - Yhat[, , aMax]
    sc <- sqrt(diag(crossprod(T, T)))
    Wa <- W
    for (a in 1:aMax) {
        Wa[, a] <- W[, a] * sc[a]
        T[, a] <- T[, a]
    }
    rownames(T) <- rownames(X)
    rownames(Wa) <- colnames(X)
    W <- list(sigLoadings = Wa, loadingWeights = W, scores = T,
        loadings = P, qa = Q, Ua = U, b = B, Q0 = NULL, f = f)
    return(W)
}

crossvalidatePLS = function (W, X, Y, ModifiedTtest = 0.5, aMax = 3, method = "kernel",
    ridge = 0.1, penalty = NULL)
{
    library(Hotelling)
    dY <- dim(Y)
    dX <- dim(X)
    t2 <- vector("numeric", length = dX[2])
    Wjk <- array(data = NA, dim = c(dX[2], aMax, dX[1]))
    Yval <- array(data = 0, dim = c(dY[1], dY[2], aMax))
    usv <- svd(X)
    Xc <- usv$u %*% diag(usv$d)
    for (ii in 1:dX[1]) {
        if (method == "bridge") {
            subMod <- calibrate.bridgePLS(scale(Xc[-ii, ], scale = F),
                scale(Y[-ii, ], scale = F), ridge = ridge, aMax = aMax)
        }
        if (method == "kernel") {
            subMod <- calibrate.kernelPLS(scale(Xc[-ii, ], scale = F),
                scale(Y[-ii, ], scale = F), aMax = aMax)
        }
        for (a in 1:aMax) {
            Yval[ii, , a] <- Xc[ii, ] %*% as.matrix(subMod$b[,
                , a])
        }
        Wi <- t(t(subMod$sigLoadings) %*% t(usv$v))
        sol <- svd(crossprod(W, Wi))
        rot <- sol$v %*% t(sol$u)
        Wjk[, , ii] <- Wi %*% rot
    }
    if (ModifiedTtest == 1) {
        t2 <- sqrt(apply(W^2, 1, sum))
    }
    else {
        cvmx <- array(data = NA, dim = c(dX[2], aMax, aMax))
        for (i in 1:dX[2]) {
```

```
            cvmx[i, , ] <- cov(scale(t(as.matrix(Wjk[i, , ])),
                center = TRUE, scale = FALSE))
        }
        if (is.null(penalty)) {
            penalty <- apply(cvmx, c(2, 3), median)
        }
        for (i in 1:dX[2]) {
            cvmx[i, , ] <- (1 - ModifiedTtest) * (cvmx[i, , ]) +
                ModifiedTtest * penalty
        }
        for (i in 1:dX[2]) {
            t2[i] <- hotell(cvmx[i, , ], W[i, ])
        }
    }
    t2 <- as.matrix(t2)
    names(t2) <- colnames(X)
    t2 <- list(t2 = t2, penalty = penalty, Yval = Yval, Wjk = Wjk)
    return(t2)
}

setScores = function (W, sets, T2, T2null = NULL, computeWset = TRUE)
{
    library(vegan)
    dSets <- length(names(sets))
    dW <- dim(W)
    setW <- matrix(0, dSets, dW[2])
    setT <- vector("numeric", length = dSets)
    H <- sqrt(rowSums(W^2))
    for (set in 1:dSets) {
        setCov <- cor(rbind(0, t(W[sets[[set]], ])))
        setCov[setCov < 0] <- 0
        setD <- scale(setCov, center = F, scale = 1/sqrt(T2)[sets[[set]]])
        setD <- scale(t(setD), center = F, scale = 1/sqrt(T2)[sets[[set]]])
        diag(setD) <- 0
        setT[set] <- sum(setD)/(length(sets[[set]])^2 - length(sets[[set]]))
        if (computeWset) {
            setWtemp <- W[sets[[set]], ]
            for (a in 1:dW[2]) {
                setW[set, a] <- weighted.mean(setWtemp[, a],
                  w = colMeans(setD))
            }
            rownames(setW) <- names(sets)
        }
        else {
            setW <- NULL
        }
    }
    names(setT) <- names(sets)
    return(list(setT = setT, setW = setW))
```

```
}

resamplePLS = function (W, Treal, X, Y, TsetReal, sets, n, ModifiedTtest, penalty,
    method = "kernel", ridge = 0.01, aMax = 3, scrambleSets = F)
{
    dY <- dim(Y)
    dX <- dim(X)
    dSet <- length(names(sets))
    dW <- dim(W)
    aMax <- dW[2]
    Tpert <- array(data = NA, dim = c(dX[2], n))
    TsetPert <- array(data = NA, dim = c(dSet, n))
    for (i in 1:n) {
        if (i %in% c(2, 3, 5, 7, 9, 11, 17, 33, 65, 129, 257,
            513, 1025, 2049, 4097)) {
            print(paste("completed", as.character(i - 1), "resamplings"))
        }
        Yind <- sample(1:dY[1])
        Y <- Y[Yind, ]
        Y <- as.matrix(Y)
        if (method == "kernel") {
            pertModel <- calibrate.kernelPLS(X, Y, aMax = aMax)
        }
        if (method == "bridge") {
            pertModel <- calibrate.bridgePLS(X, Y, aMax = aMax,
                ridge = ridge)
        }
        Wpert <- pertModel$sigLoadings
        ValRes <- crossvalidatePLS(Wpert, X, Y, ModifiedTtest,
            penalty = penalty, aMax = aMax, method = method,
            ridge = ridge)
        if (!is.null(sets)) {
            if (scrambleSets) {
                for (set in names(sets)) {
                  sets[[set]] <- sample(colnames(X), size = length(sets[[set]]))
                }
                setStatsPert <- setScores(W, sets, Treal, T2null = NULL)
            }
            else {
                setStatsPert <- setScores(Wpert, sets, ValRes$t2,
                  T2null = NULL, computeWset = F)
            }
            TsetPert[, i] <- setStatsPert$setT
        }
        Tpert[, i] <- ValRes$t2
    }
    qv <- vector("numeric", dX[2])
    print("Computing q values")
    for (i in 1:dX[2]) {
```

```
        n_better <- colSums(Tpert >= Treal[i])
        qv[i] <- median(n_better)/(1 + sum(Treal > Treal[i]))
    }
    if (!is.null(sets)) {
        print("computing set q values")
        qSet <- vector("numeric", dSet)
        for (i in 1:dSet) {
            n_better <- colSums(TsetPert >= TsetReal[i])
            qSet[i] <- median(n_better)/(1 + sum(TsetReal > TsetReal[i]))
        }
        names(qSet) <- names(sets)
    }
    else {
        qSet <- NULL
    }
    return(list(qv = qv, Tpert = Tpert, qSet = qSet))
}
```

D.8 Calculating the Phase Angle Distribution of Marker Genes

The following code is developed for modelling the phase angle distribution of marker genes in the PLS loading plane and to assign a phase for the genes in the PLS loading plane.

```
def compute_angle_phase_distribution_with_resampling(pls_sigload_cutoff,clean=True):
    known_df = pd.read_csv("./data/known_plus_regev.csv",index_col=0)
    z = np.zeros(len(known_df['phase'].unique()))
    angle_dist_df = pd.DataFrame({'mean': z,
            'variance': z,
            'n': z},
            index=known_df['phase'].unique()
            )
    for p in known_df['phase'].unique():
        p_df = known_df[known_df['phase'] == p].copy()
        p_genes = list(set(list(p_df['Ensembl'])).intersection(set(list(pls_sigload_cutoff.index))))
        angle_dist_df.loc[p,'n'] = len(p_genes)
        if len(p_genes) > 2:
            re_means = []
            re_vars = []
            #resampling with leave 10% out
            for i in range(500):
                re_genes = np.random.choice(p_genes, len(p_genes)-int(np.ceil(len(p_genes)*0.1)))
                re_angles = pls_sigload_cutoff.loc[re_genes, 'angles']
```

```
                if re_angles.max() - re_angles.min() > 1.5*np.pi:
                    re_angles[re_angles > np.pi] -= 2*np.pi
                m = np.mean(re_angles)
                v = np.var(re_angles,ddof=1)
                re_means.append(m)
                re_vars.append(v)
            m_v_df = pd.DataFrame({'mean': re_means, 'variance': re_vars})
            m_v_df = m_v_df.sort_values(by=['variance'])
            mean = m_v_df.loc[int(len(m_v_df)*0.5),'mean']
            variance = m_v_df.loc[int(len(m_v_df)*0.5),'variance']
        else:
            p_angles = pls_sigload_cutoff.loc[p_genes, 'angles']
            if p_angles.max() - p_angles.min() > 1.5*np.pi:
                p_angles[p_angles > np.pi] -= 2*np.pi
            mean = np.mean(p_angles)
            variance = np.var(p_angles,ddof=1)

        angle_dist_df.loc[p,'mean'] = mean
        angle_dist_df.loc[p,'variance'] = variance

    if clean:
        angle_dist_df.dropna(inplace=True)
        angle_dist_df = angle_dist_df[angle_dist_df['variance'] != 0]
        angle_dist_df = angle_dist_df[angle_dist_df['variance'] < 1.5 ].copy()

    #make cyclic where needed
    for p in angle_dist_df.index.unique():
        if (angle_dist_df.loc[p,'mean'] + 3*np.sqrt(angle_dist_df.loc[p,'variance']) > 2*np.pi)
                and (np.sqrt(angle_dist_df.loc[p,'variance'])<1):
            shift_df = pd.DataFrame({
                'mean': angle_dist_df.loc[p,'mean']-2*np.pi,
                'variance': angle_dist_df.loc[p,'variance'],
                'n': angle_dist_df.loc[p,'n']
            }, index=['{}-shift'.format(p)])
            angle_dist_df = angle_dist_df.append(shift_df)
        elif (angle_dist_df.loc[p,'mean'] - 3*np.sqrt(angle_dist_df.loc[p,'variance']) < 0)
                and (np.sqrt(angle_dist_df.loc[p,'variance'])<1):
            shift_df = pd.DataFrame({
                'mean': angle_dist_df.loc[p,'mean']+2*np.pi,
                'variance': angle_dist_df.loc[p,'variance'],
                'n': angle_dist_df.loc[p,'n']
            }, index=['{}-shift'.format(p)])
            angle_dist_df = angle_dist_df.append(shift_df)

    return angle_dist_df

def assign_phase(pls_sigload_cutoff,angle_dist_df):
    import scipy
    assigned_phase = []
```

```
    for gene, row in pls_sigload_cutoff.iterrows():
        angle = row['angles']
        z_d = {}
        for p in angle_dist_df.index.unique():
            mean = angle_dist_df.loc[p,'mean']
            scale = np.sqrt(angle_dist_df.loc[p,'variance'])
            z_d[p] = scipy.stats.norm.pdf(angle, mean, scale)
        phase_max = max(z_d, key=z_d.get)
        assigned_phase.append(phase_max if z_d[phase_max]>0.05 else 'not_assigned')
    pls_sigload_cutoff['assigned_phase'] = assigned_phase
    pls_sigload_cutoff['assigned_phase'].value_counts()
```

D.9 Functional Enrichment using gProfiler2

The following code is used to retrieve the significant terms in our functional enrichment analysis. The R script will also export a table of significant terms to images.

```
#!/usr/bin/env Rscript

library(gprofiler2)

args = commandArgs(trailingOnly=TRUE)

df <- read.csv(file=args[1], header=TRUE, sep=",", row.names=1)
if (length(args)==3) {
    df <- df[df$assigned_phase == args[3],]
}



res <- gost(rownames(df),
            organism='hsapiens',
            user_threshold=0.05,
            correction_method=c('g_SCS'),
            domain_scope=c('annotated'),
            sources=c('GO:BP')
            )

res$result <- res$result[res$result$term_size > 10,]
res$result <- res$result[res$result$term_size < 2500,]
```

```
if (nrow(res$result)>60){
    res$result <- res$result[1:60,]
}

publish_gosttable(res,filename=args[2])


if (nrow(res$result)>10){
    res$result <- res$result[1:10,]
    publish_gosttable(res,filename=gsub(".png","_top10.png",args[2]))
}
```

Appendix E

Material Availability

E.1 Single Cell pipeline

As part of this experiment, a Snakemake pipeline was implemented for running STARsolo. This was developed as a part of the existing single cell pipeline at Genomics Core Facility, Norwegian University of Science and Technology.

- **Repository:** https://github.com/gcfntnu/single-cell/tree/star
- **Branch:** star
- **Commit:** 1e6988664923d4dee2f15cf1eba664f5d731e363

E.2 Developed Material

The materiel developed for this experiment can in the git repository below. The count matrices are too large to upload, but will be delivered upon request.

- **Repository:** https://github.com/gcfntnu/single-cell/tree/star

The repository contains the following:

- The lists of genes acquired in our analysis.
- The conda environments used.
- Example notebook for the 293t analysis.
- Example notebook for PLS regression.
- The PLS implementation, miirPLS.
- Our developed utilites.

Bibliography

[Aba+16] Martin Abadi et al. "Tensorflow: A system for large-scale machine learning". In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016, pp. 265–283.

[And+03] GE Andersson et al. "On the origin of mitochondria: a genomics perspective". In: *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences* 358.1429 (2003), pp. 165–179.

[Ash+00] Michael Ashburner et al. "Gene ontology: tool for the unification of biology". In: *Nature genetics* 25.1 (2000), p. 25.

[Con18] Gene Ontology Consortium. "The gene ontology resource: 20 years and still GOing strong". In: *Nucleic acids research* 47.D1 (2018), pp. D330–D338.

[Dob+13] Alexander Dobin et al. "STAR: ultrafast universal RNA-seq aligner". In: *Bioinformatics* 29.1 (2013), pp. 15–21.

[GKK12] Lorenzo Galluzzi, Oliver Kepp, and Guido Kroemer. "Mitochondria: master regulators of danger signalling". In: *Nature reviews Molecular cell biology* 13.12 (2012), p. 780.

[GM00] Cooper GM. *The Cell: A Molecular Approach. 2nd edition.* Sunderland (MA): Sinauer Associates, 2000.

[Grü+18] Björn Grüning et al. "Bioconda: sustainable and comprehensive software distribution for the life sciences". In: *Nature methods* 15.7 (2018), p. 475.

[GSM18] Jonathan A Griffiths, Antonio Scialdone, and John C Marioni. "Using single-cell genomics to understand developmental processes and cell fate decisions". In: *Molecular systems biology* 14.4 (2018).

[Ili+16] Tomislav Ilicic et al. "Classification of low quality cells from single-cell RNA-seq data". In: *Genome biology* 17.1 (2016), p. 29.

[JLB03] Daniel Johansson, Petter Lindgren, and Anders Berglund. "A multivariate approach applied to microarray data for identification of genes with cell cycle-coupled transcription". In: *Bioinformatics* 19.4 (2003), pp. 467–473.

[Jol11] Ian Jolliffe. *Principal component analysis*. Springer, 2011.

[KR12] Johannes Köster and Sven Rahmann. "Snakemake—a scalable bioinformatics workflow engine". In: *Bioinformatics* 28.19 (2012), pp. 2520–2522.

[Lia+19] Shaoheng Liang et al. "Latent periodic process inference from single-cell RNA-seq data". In: *BioRxiv* (2019), p. 625566.

[LT19] Malte D Luecken and Fabian J Theis. "Current best practices in single-cell RNA-seq analysis: a tutorial". In: *Molecular systems biology* 15.6 (2019).

[Mar+10] Gunnar Martinsson et al. "Randomized methods for computing the Singular Value Decomposition (SVD) of very large matrices". In: *Works. on Alg. for Modern Mass. Data Sets, Palo Alto* (2010).

[MM93] Udi Manber and Gene Myers. "Suffix arrays: a new method for on-line string searches". In: *siam Journal on Computing* 22.5 (1993), pp. 935–948.

[Ped+11] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[Pen+13] Javier Pena-Diaz et al. "Transcription profiling during the cell cycle shows that a subset of Polycomb-targeted genes is upregulated during DNA replication". In: *Nucleic acids research* 41.5 (2013), pp. 2846–2856.

[Rau+19] Uku Raudvere et al. "g: Profiler: a web server for functional enrichment analysis and conversions of gene lists (2019 update)". In: *Nucleic acids research* (2019).

[Rei+07] Jüri Reimand et al. "g: Profiler—a web-based toolset for functional profiling of gene lists from large-scale experiments". In: *Nucleic acids research* 35.suppl_2 (2007), W193–W200.

[SAE12] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. "OpenStack: toward an open-source solution for cloud computing". In: *International Journal of Computer Applications* 55.3 (2012), pp. 38–42.

[Sat+15] Rahul Satija et al. "Spatial reconstruction of single-cell gene expression data". In: *Nature biotechnology* 33.5 (2015), p. 495.

[SC75] Fred Sanger and Alan R Coulson. "A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase". In: *Journal of molecular biology* 94.3 (1975), pp. 441–448.

[SD78] Robert M Schwartz and Margaret O Dayhoff. "Origins of prokaryotes, eukaryotes, mitochondria, and chloroplasts". In: *Science* 199.4327 (1978), pp. 395–403.

[Tir+16] Itay Tirosh et al. "Dissecting the multicellular ecosystem of metastatic melanoma by single-cell RNA-seq". In: *Science* 352.6282 (2016), pp. 189–196.

[Whi+02] Michael L Whitfield et al. "Identification of genes periodically expressed in the human cell cycle and their expression in tumors". In: *Molecular biology of the cell* 13.6 (2002), pp. 1977–2000.

[Wol75] Herman Wold. "Path models with latent variables: The NIPALS approach". In: *Quantitative sociology*. Elsevier, 1975, pp. 307–357.

[WT71] Ray Wu and Ellen Taylor. "Nucleotide sequence analysis of DNA: II. Complete nucleotide sequence of the cohesive ends of bacteriophage λ DNA". In: *Journal of molecular biology* 57.3 (1971), pp. 491–511.

[Zhe+17] Grace XY Zheng et al. "Massively parallel digital transcriptional profiling of single cells". In: *Nature communications* 8 (2017), p. 14049.