My Thesis or Dissertation Title

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Cedric Landerer

December 2018

To my mother

Acknowledgments

First and foremost I want to thank my Adviser, Michael Gilchrist for his long lasting patience.

Abstract

Abstract text goes here...

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This is a very short guide to an unofficial thesis/dissertation template for the University of Tennessee. It has been updated to meet the specifications as of 2017 but can be easily altered as the guidelines are changed. This template requires a basic knowledge of LaTeX and should cover the basic requirements in terms of required packages and functionality.

1.1 Disclaimer

This template is distributed with ABSOLUTELY NO WARRANTY. It serves as a guideline and constitutes a basic structure for a thesis/dissertation. The user assumes full responsibility for formatting and typesetting their document and for verifying that all the thesis requirements set by the University of Tennessee are met. Please refer to the most recent UT thesis guide http://gradschool.utk.edu/thesesdissertations/formatting/ or contact the thesis consultant (http://gradschool.utk.edu/thesesdissertations/). Please report any bugs to the thesis consultant.

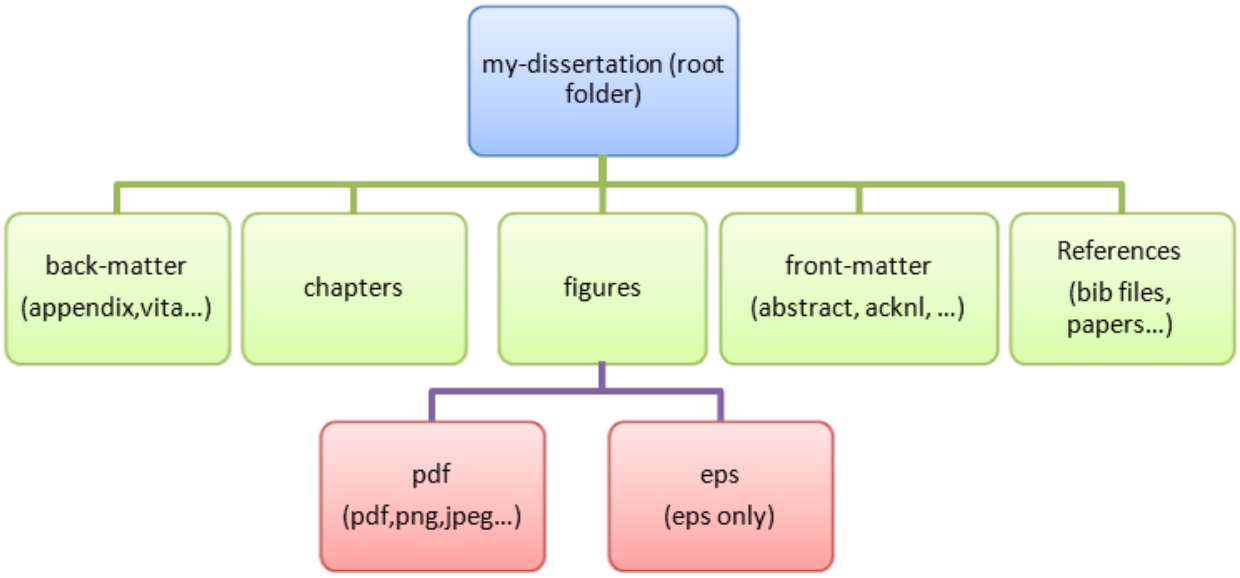


Figure 1.1: UT thesis template folder structure.

1.2 Getting started

The general structure of this template is based on the tree shown in Figure 1.1. The titles of the folders are self descriptive and should guide you to proper file placement. Note that this is only a suggested model that could be modified to fit your own organizational structure.

There are two important files in this template: "my-dissertation.tex" and "utthesis.cls". The "utthesis.cls" is the class file that contains the settings, definitions, packages, and macros for this template to work properly and is located in the root directory. This file constitutes the document class for the template. It is based on the report class and provides some customized functionality. It will also generate a title page for you. In certain cases, one of the packages included in this template may conflict with a package that you are adding. You will have to resolve this conflict by either removing the package that is not being used or by modifying some settings with either packages. The packages that are preloaded in this class file are: amsmath, amsthm, amssymb, setspace, geometry, hyperref, and color.

The "my-dissertation.tex" file is the main file for your thesis/dissertation. This is where you bring all of the pieces together. Each individual section of your dissertation should be its own .tex file saved in the proper place. For example, a chapter for your dissertation should be saved in the "chapters" folder. Whereas your acknowledgments file should be saved in

the “front-matter” folder. The “my-dissertation.tex” file is the file you compile to make your dissertation. It’ll call all of the included files and compile the document properly. You may want to change the name of the file to something like “my-name-dissertation.tex”. Next, invoke the proper options for the “utthesis” document class. This class will take all the options for the report class in addition to two options: thesis/dissertation and monochrome. If you are writing a thesis, you must use ”thesis” otherwise, use ”dissertation” or omit that option because dissertation is the default setting. The monochrome option converts all your document to monochrome - except figures. This is very useful when printing your document. Because this dissertation has colored hyperlinks, these will look washed out when printed on a monochrome printer. Therefore, it is handy to have a monochrome copy of your thesis for print.

Now you are ready to fill in the proper values corresponding to your title, name, degree, etc. This can be done in the following section:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TO DO: FILL IN YOUR INFORMATION BELOW - READ THIS SECTION CAREFULLY
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\title{My Thesis or Dissertation Title}  % title of thesis/dissertation
\author{Smokey Volunteer}                % author's name
\copyrightYear{2017}                     % copyright year of your
                                           thesis/dissertation
\graduationMonth{May}                    % month of graduation for your
                                           thesis/dissertation
\degree{Doctor of Philosophy}   % degree: Doctor of Philosophy, Master of
                                  Science, Master of Engineering...
\university{The University  of Tennessee, Knoxville} % school name
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

1.3 References

The bibliography style used in this template is "apalike". It is an author-year style based on the APA specification. Here are a few examples. T. Hungerford wrote a book on Algebra, [**?**]. In 1999, D. F. Anderson and P. S. Livingston wrote the defining paper on zero-divisor graphs of commutative rings in [**?**]. You can also point out specific theorems in papers, such as the fact that the zero-divisor graph always has diameter less than or equal to 3, [**?** , Theorem 2.3]. You can also list several references at once. For example, for more on zero-divisor graphs see [**?** **?**]. However, you can change this style to any format you'd like. The code in the "my-dissertation.tex" file is

```
\utbiblio{#1}{apalike}{references-dissertation}
```

The first entry ("#1") must remain there. It deletes the title "Bibliography" from being printed again at the top of the bibliography page. The title "Bibliography" should only appear on the title page. The second entry can be changed to any natbib style you'd like. For example, plainnat, humannat, etc. The third entry is the name of your bibliography .tex file. Remember to run BibTeX in order to compile the bibliography correctly. For more information, visit http://merkel.texture.rocks/Latex/natbib.php.

1.4 Theorem environments

This template contains predefined theorem, lemma, proposition, corollary, and definition environments. For example,

Definition 1.1. *This is your definition.*

Proposition 1.2. *This is an example of a proposition.*

Theorem 1.3 (First theorem)**.** *This is an example theorem.*

Proof for theorem. This is the proof for this theorem. □

Lemma 1.4 (First lemma)**.** *This is the first lemma.*

Proof. This is the proof for this lemma that requires Theorem 1.3. □

Corollary 1.5. *This is the first corollary.*

1.5 Figures and Tables

1.5.1 General Rules

To comply with the 2017 dissertation formatting, figure captions should be placed below the figure and table captions should be placed above the table. Also, if a table or figure takes up more than half the page, then there should be no text on that page (except for the caption of course). Lastly, you must allow tables and figures to float. DO NOT HARD CODE POSITIONS. In addition, no table or figure should go into the margins. If a table or figure does creep into the margins you can either resize it so that it properly fits within the margins, or put it on its own page and make that specific page landscape. See Figure 1.5 for an example. Note the page number location in the example. The code for this is given by:

```
\begin{landscape}
\thispagestyle{mylandscape}
\begin{figure}[h]
\centering
\includegraphics[width=9in]{32303-TheHill-byJoshQueener.jpg}
\caption{This view of The Hill is too wide for a portrait page.}
\label{fig:wide-pic}
\end{figure}
\end{landscape}
```

Be careful about where you place this landscape page, as well as all figures and tables. These objects are not considered part of the text, and thus their placement should not be assigned to a precise location. The general rule to follow is that no text page should have significant white space, with the exception being the last page of a chapter. So if you mention a figure in some paragraph but the figure will not fit on the remainder of the page, continue the text (even if it's a new section) to fill the current page with text and then place the figure on the next page. To see an example of this, consider this page you are reading now. We've mentioned Figure 1.5 in the previous paragraph. However, it requires a new page

and since there is plenty of space on this page, we've filled it with text and delayed the `\begin{landscape}` section of code until the appropriate position.

1.5.2 Single figures

For more information, check out the following page:
http://en.wikibooks.org/wiki/LaTeX/Floats,_Figures_and_Captions

```
\begin{figure}[t for top, b for bottom, h for here]
    % Requires \usepackage{graphicx}
    \centering % center the figure
    \includegraphics[width=5in or 127mm etc...]{figure-name}\\
    \caption{figure caption}\label{figure label}
\end{figure}
```

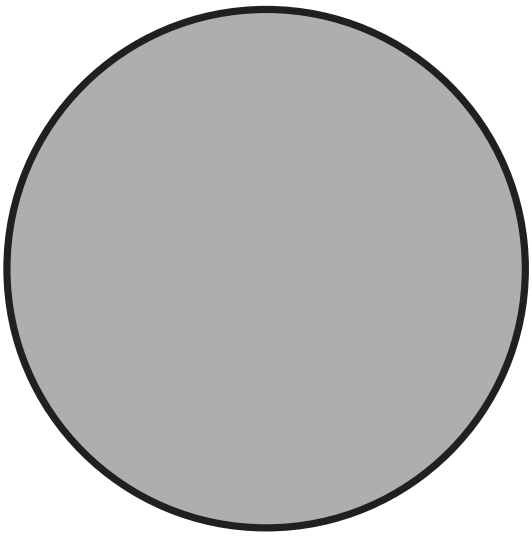


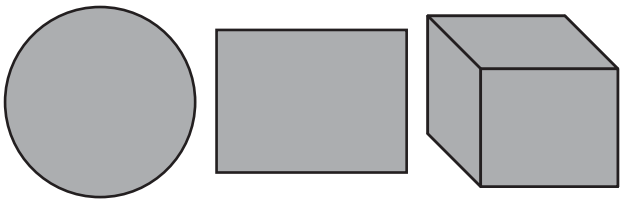
Figure 1.2: Sample caption.

1.5.3 Multipart figures

For multipart figures, you need to use the package "subfig". here's an example:

```
\begin{figure}[h]
```

```
  \centering
  \subfloat[Circle]{\label{fig:figure-a}\includegraphics[width=1.1in]
    {fig02a-circle}}
  \subfloat[Rectangle]{\label{fig:figure-b}\includegraphics[width=1.1in]
    {fig02b-rectangle}}
  \subfloat[Cube]{\label{fig:figure-c}\includegraphics[width=1.1in]
    {fig02c-cube}}
  \caption{Geometric shapes.}
  \label{fig:multipart-figure}
\end{figure}
```



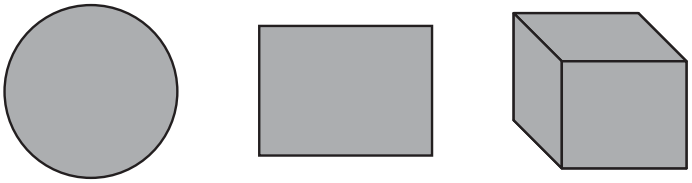
(a) Circle **(b)** Rectangle **(c)** Cube

Figure 1.3: Geometric shapes.

To add some space between the figures above, one can use the usual spacing commands such as "\qquad". For example,

```
\begin{figure}[h]
  \centering
  \subfloat[Circle]{\label{fig:fig-a-space}\includegraphics[width=1in]
    {fig02a-circle}} \qquad
  \subfloat[Rectangle]{\label{fig:fig-b-space}\includegraphics[width=1in]
    {fig02b-rectangle}}\qquad
  \subfloat[Cube]{\label{fig:fig-c-space}\includegraphics[width=1in]
    {fig02c-cube}}\qquad
  \caption{Geometric shapes with space between images.}
  \label{fig:multipart-figure-space}
```

```
\end{figure}
```



(a) Circle **(b)** Rectangle **(c)** Cube

Figure 1.4: Geometric shapes with space between images.

Figure 1.5: This view of The Hill is too wide for a portrait page.

1.5.4 Tables

Again, table captions should be placed above the table. See Table 1.1 for an example and to learn about Smokey's history[1]. For more information about tables, see https://en.wikibooks.org/wiki/LaTeX/Tables.

Table 1.1: Smokey's History

Dog	Years	Record	Pct.
Blue Smokey	1953-1954	10-10-1	.500
Smokey II	1955-1963	58-39-5	.597
Smokey III	1964-1977	105-39-5	.729
Smokey IV	1978-1979	12-10-1	.545
Smokey V	1980-1983	28-18-1	.608
Smokey VI	1984-1991	67-23-6	.744
Smokey VII	1992-1994	27-9	.750
Smokey VIII	1995-2003	91-22	.805
Smokey IX	2004-2012	62-53	.539
Smokey X	2013-present	21-17	.552

[1] According to Wikipedia: https://en.wikipedia.org/wiki/Smokey_(mascot)

Chapter 2

AnaCoDa: Analyzing Codon Data with Bayesian mixture models

Abstract

AnaCoDa is an R package for estimating biologically relevant parameters of mixture models, such as selection against translation inefficiency, nonsense error rate, and ribosome pausing time, from genomic and high throughput datasets. **AnaCoDa** provides an adaptive Bayesian MCMC algorithm, fully implemented in C++ for high performance with an ergonomic R interface to improve usability. **AnaCoDa** employs a generic object-oriented design to allow users to extend the framework and implement their own models. Current models implemented in **AnaCoDa** can accurately estimate biologically relevant parameters given either protein coding sequences or ribosome foot-printing data. Optionally, **AnaCoDa** can utilize additional data sources, such as gene expression measurements, to aid model fitting and parameter estimation. By utilizing a hierarchical object structure, some parameters can vary between sets of genes while others can be shared. Genes may be assigned to clusters or membership may be estimated by **AnaCoDa**. This flexibility allows users to estimate the same model parameter under different biological conditions and categorize genes into different sets based on shared model properties embedded within the data. **AnaCoDa** also allows users to generate simulated data which can be used to aid model development and model analysis as well as evaluate model adequacy. Finally, **AnaCoDa** contains a set of visualization routines and the ability to revisit or re-initiate previous model fitting, providing researchers with a well rounded easy to use framework to analyze genome scale data.

Availability:

AnaCoDa is freely available under the Mozilla Public License 2.0 on CRAN (https://cran.r-project.org/web/packages/AnaCoDa/).

2.1 Introduction

AnaCoDa is an open-source software implemented in R [6] that allows researchers to analyze genome-scale data like coding sequences and ribosome footprinting data using evolutionary or analytical models in a Bayesian framework. **AnaCoDa** was developed to analyze selection on synonymous codon usage in the form of ribosome overhead cost ([4, 9, 7]). However, other codon metrics like the codon adaptation index [8] or the effective number of codons [10] are also provided as reference. In addition, three currently unpublished models to analyze coding sequences for evidence of selection against nonsense errors and estimate ribosome pausing times from ribosome footprinting data are included. **AnaCoDa** implements an adaptive Gibbs sampler within a Metropolis-Hastings Monte Carlo Markov Chain (MCMC). This allows for the incorporation of prior knowledge such as observed gene expression levels and easy sampling from the posterior distribution to estimate parameter values and quantify degree of uncertainty. **AnaCoDa** provides a mixture distribution option to all implemented models, combining genes into sets by estimating the posterior probabilities of set membership based on gene-set specific parameters shared by all genes assigned to a given set. **AnaCoDa** provides a generic, mixture distribution option to all implemented models, allowing for the estimation of condition specific parameters or the automatic categorization of data into different sets based on differences in their posterior probabilities of set membership. In addition to the four models provided, **AnaCoDa** provides a modular infrastructure such that additional genome scale or even phylogenetic models can be integrated.

The **AnaCoDa** framework works with **AnaCoDa** requires gene specific data such as codon frequencies obtained from coding sequences or position specific footprint counts. Conceptually, **AnaCoDa** allows for three different types of parameters. The first type are gene specific parameters such as protein synthesis rate or relative functionality. The second type are gene-set specific parameters, such as mutation bias terms or translation error rates. These parameters are shared across genes within a set and can be exclusive to a single set or shared with other sets. While the number of gene sets must be pre-defined by the user, set assignment of genes can be pre-defined or estimated as part of the model fitting. Estimation of the set assignment provides the probability of a gene being assigned to a set allowing

the user to asses the uncertainty in each assignment. The third type are hyperparameters allowing for the construction and analysis of hierarchical model. Hyperparameters control the prior distribution for gene and gene-set specific parameters such as mutation bias or protein synthesis rate.

2.2 Features

AnaCoDa provides an interface written in R, a freely available programming language noted for its ease of use for even inexperienced programmers. As a result, **AnaCoDa** is accessible to researchers with minimal computational experience.

The interface of **AnaCoDa** is designed for quick and efficient data analysis. Generally, the only input needed for fitting a model to the data are protein-coding codon sequences in the form of a FASTA file or a flat-file containing codon counts obtained from ribosome foot-printing experiments. **AnaCoDa** also provides visualization functionality, including plotting functions to compare parameter estimates for different mixture distributions and display codon usage patterns. In addition, diagnostic functions such as those for calculating and visualizing the degree of autocorrelation in the parameter tracess are provided.

Robust and efficient model fitting

AnaCoDa has built-in features designed to improve the robustness and performance of the implemented MCMC approach. For example, the implemented MCMC automatically adapts the proposal width for sampled parameters such that an user defined acceptance range is met, improving sampling efficiency of the MCMC and computational performance. Even though **AnaCoDa** is written in C++, analysis of large datasets and/or complex models can be very computationally intensive. To protect users from computer failures or aid in the collection of additional MCMC samples, **AnaCoDa** can periodically produce output checkpoint files, which can be used to restart an MCMC chain from a previous time point. In addition, **AnaCoDa** automatically thins all parameter traces - meaning only every k^{th} sample is kept - increasing the effective number of samples and reducing its memory foorprint.

Although **AnaCoDa** is provided as an R package, the main computational work is implemented in C++. Because R does not provide native C++ support, Rcpp was employed to expose whole C++ classes as modules to R [3]. Using Rcpp eliminates time consuming data transfers between the R environment and the C++ core during model fitting, resulting in improved computational performance and allows for a fully object-oriented code design [1]. As expected, the runtime of **AnaCoDa** scales linearly with genome size and number of iterations, and scales polynomially with the number of mixture distributions in the data set. The polynomial increase in runtime with the number of mixture distributions is due to the necessity to condition the gene assignment on the estimation of gene specific parameters, such as, protein synthesis rate.

Data Simulation

In addition to fitting the models to datasets, **AnaCoDa** can be used to generate simulated data sets as well. On their own, simulated datasets are useful for model development and analysis. Simulating data under different conditions allows the user to explore model behavior and explore theoretical scenarios. Different conditions can include the addition or elimination of parameters, or simply allowing a set of parameter values to vary. Fitting models to simulated data can provide insight into potential pitfalls or shortcomings when fitting observational data and can serve as the basis for evaluating model adequacy of a model fit to observational data [5]. Significant differences between simulated and observational data suggests the current set of parameters or the model as a whole fail to include or adequately represent biological mechanisms underlying the observed data.

Available models

AnaCoDa currently provides codon models for analyzing genome scale data. The ROC model implements and extends the codon usage bias (CUB) models developed by Gilchrist et al. [4], Wallace et al. [9], Shah and Gilchrist [7], which can reliably estimate the strength of selection on ribosome overhead cost, mutation bias and allows for the inference of protein synthesis rates. This model allows for the separation of effects of mutation and selection based on gene ordering by protein synthesis rate, and the addition of a mixture distribution

allows for gene clustering based on mutation bias and selection for translation efficiency. In addition to identifying the most efficient codons, ROC provides estimates of mutation bias allowing the approximation of mutation ratios between codons [4, 9].

The ability to estimate protein synthesis rates in the absence of empirical data is useful for investigating CUB of non-model organisms for which such data is lacking and enables the usage of protein synthesis rate in comparative frameworks or other analyses requiring protein synthesis rate information [2]. Use of the mixture model allows for the investigation of CUB heterogeneity at the genome or gene level. Following the same framework, additional models included in **AnaCoDa** provide estimates of codon-specific nonsense errors rates (FONSE) and ribosome pausing times (PA and PANSE).

Parameters estimated with the evolutionary models ROC and FONSE represent evolutionary averages and do not depend on experimental conditions. In contrast, PA and PANSE estimate the distribution of biologically relevant parameters like ribosome pausing times along a gene from experimental data such as ribosome footprinting data. The distribution can be dependent (PANSE) or independent (PA) of evidence for nonsense errors in the data.

2.3 Supplementary Material

AnaCoDa allows for the estimation of biologically relevant parameters like mutation bias or ribosome pausing time, depending on the model employed. Bayesian estimation of parameters is performed using an adaptive Metropolis-Hasting within Gibbs sampling approach. Models implemented in AnaCoDa are currently able to handle gene coding sequences and ribosome footprinting data.

2.3.1 The AnaCoDa framework

The AnaCoDa framework works with gene specific data such as codon frequencies or position specific footprint counts. Conceptually, AnaCoDa uses three different types of parameters.

- The first type of parameters are **gene specific parameters** such as gene expression level or functionality. Gene-specific parameters are estimated separately for each gene and can vary between potential gene categories or sets.

- The second type of parameters are **gene-set specific parameters**, such as mutation bias terms or translation error rates. These parameters are shared across genes within a set and can be exclusive to a single set or shared with other sets. While the number of gene sets must be pre-defined by the user, set assignment of genes can be pre-defined or estimated as part of the model fitting. Estimation of the set assignment provides the probability of a gene being assigned to a set allowing the user to asses the uncertainty in each assignment.

- The third type of parameters are **hyperparameters**, such as parameters controlling the prior distribution for mutation bias or error rate. Hyperparameters can be set specific or shared across multiple sets and allow for the construction and analysis of hierarchical models, by controlling prior distributions for gene or gene-set specific parameters.

Analyzing protein coding gene sequences

AnaCoDa always requires the following four objects:

- **Genome** contains the codon data read from a fasta file as well as empirical protein synthesis rate in the form of a comma separated (.csv) ID/Value pairs.

- **Parameter** represents the parameter set (including parameter traces) for a given genome. The parameter object also hold the mapping of parameters to specified sets.

- **Model** allows you to specify which model should be applied to the genome and the parameter object.

- **MCMC** specifies how many samples from the posterior distribution of the specified model should be stored to obtain parameter estimates.

2.3.2 AnaCoDa setup

Application of codon model to single genome

In this example we are assuming a genome with only one set of gene-set specific parameters, hence **num.mixtures** = 1. We assign all genes the same gene-set, and provide an initial value for the hyperparameter sphi (s_ϕ). s_ϕ controls the lognormal prior distribution on the gene specific parameters like the protein synthesis rate ϕ. To ensure identifiability the expected value of the prior distribution is assumed to be 1.

$$E[\phi] = \exp\left(m_\phi + \frac{s_\phi^2}{2}\right) = 1 \tag{2.1}$$

Therfore the mean m_ϕ is set to be $-\frac{s_\phi^2}{2}$. For more details see Gilchrist et al. [4].

After choosing the model and specifying the necessary arguments for the MCMC routine, the MCMC is run

```
genome <- initializeGenomeObject(file = "genome.fasta")
parameter <- initializeParameterObject(genome = genome, sphi = 1,
                        num.mixtures = 1,
                        gene.assignment = rep(1, length(genome)))
model <- initializeModelObject(parameter = parameter, model = "ROC")
mcmc <- initializeMCMCObject(samples = 5000, thinning = 10,
                        adaptive.width=50)
runMCMC(mcmc = mcmc, genome = genome, model = model)
```

runMCMC does not return a value, the results of the MCMC are stored automatically in the mcmc and parameter objects created earlier.

> **Please note that AnaCoDa utilizes C++ obejct orientation and therefore employs pointer structures. This means that no return value is necessary for such objects as they are modified within the the runMCMC routine. You will find that after a completed run, the parameter object will contain all necessary information without being directly passed into the MCMC routine. This might be confusing at first as it is not default R behaviour.**

Application of codon model to a mixture of genomes

This case applies if we assume that parts of the genome differ in their gene-set specific parameters. This could be due to introgression events or strand specific mutation difference, horizontal gene transfers or other reasons. We make the assumption that all sets of genes are independent of one another. For two sets of gene-set specific parameter with a random gene assignment we can use:

```
parameter <- initializeParameterObject(genome = genome,
                        sphi = c(0.5, 2), num.mixtures = 2,
                        gene.assignment = sample.int(2,
                                length(genome), replace = T))
gene.assignment = sample.int(2, length(genome), replace = T))
```

To accommodate for this mixing we only have to adjust **sphi**, which is now a vector of length 2, **num.mixtures**, and **gene.assignment**, which is chosen at random here.

Empirical protein synthesis rate values

To use empirical values as prior information one can simply specify an observed.expression.file when initializing the genome object.

```
genome <- initializeGenomeObject(file = "genome.fasta",
                observed.expression.file = "synthesis_values.csv")
```

These observed expression or synthesis values (Φ) are independent of the number of gene-sets. The error in the observed Φ values is estimated and described by sepsilon (s_ϵ). The csv file can contain multiple observation sets separated by comma. For each set of observations an initial s_ϵ has to be specified.

```
# One case of observed data
sepsilon <- 0.1
# Two cases of observed data
sepsilon <- c(0.1, 0.5)
# ...
# Five cases of observed data
```

```
sepsilon <- c(0.1, 0.5, 1, 0.8, 3)
parameter <- initializeParameterObject(genome = genome, sphi = 1,
                        num.mixtures = 1,
                        gene.assignment = rep(1, length(genome)),
                        init.sepsilon = sepsilon)
```

In addition one can choose to keep the noise in the observations (s_ϵ) constant by using the **fix.observation.noise** flag in the model object.

```
model <- initializeModelObject(parameter = parameter, model = "ROC",
fix.observation.noise = TRUE)
```

Fixing parameter types

It can sometime be advantages to fix certain parameters, like the gene specific parameters. For example in cases where only few sequences are available but gene expression measurements are at hand we can fix the gene specific parameters to increase confidence in our estimates of gene-set specific parameters.

We again initialize the **genome**, **parameter**, and **model** objects.

```
genome <- initializeGenomeObject(file = "genome.fasta")
parameter <- initializeParameterObject(genome = genome, sphi = 1,
                        num.mixtures = 1,
                        gene.assignment = rep(1, length(genome)))
model <- initializeModelObject(parameter = parameter, model = "ROC")
```

To fix gene specific parameters we will set the **est.expression** flag to **FALSE**. This will estimate only gene-set specific parameters, hyperparameters, and the assignments of genes to various sets.

```
mcmc <- initializeMCMCObject(samples, thinning=1,
                        adaptive.width=100, est.expression=FALSE,
                        est.csp=TRUE, est.hyper=TRUE, est.mix=TRUE)
```

If we would like to fix gene-set specific parameters we instead disable the **est.csp** flag.

```
mcmc <- initializeMCMCObject(samples, thinning=1,
```

```
                        adaptive.width=100, est.expression=TRUE,
                        est.csp=FALSE, est.hyper=TRUE, est.mix=TRUE)
```

The same applies to the hyper parameters (**est.hyper**),

```
mcmc <- initializeMCMCObject(samples, thinning=1,
                        adaptive.width=100, est.expression=TRUE,
                        est.csp=TRUE, est.hyper=FALSE, est.mix=TRUE)
```

and gene set assignment (**est.mix**).

```
mcmc <- initializeMCMCObject(samples, thinning=1,
                        adaptive.width=100, est.expression=TRUE,
                        est.csp=TRUE, est.hyper=TRUE, est.mix=FALSE)
```

We can use these flags to fix parameters in any combination.

Combining various gene-set specific parameters to a gene-set description.

We distinguish between three simple cases of gene-set descriptions, and the ability to customize the parameter mapping. The specification is done when initializing the parameter object with the **mixture.definition** argument.

We encounter the simplest case when we assume that all gene sets are independent.

```
parameter <- initializeParameterObject(genome = genome,
                        sphi = c(0.5, 2), num.mixtures = 2,
                        gene.assignment = sample.int(2,
                                length(genome), replace = T),
                        mixture.definition = "allUnique")
```

The **allUnique** keyword allows each type of gene-set specific parameter to be estimated independent of parameters describing other gene sets.

In case we want to share mutation parameter between gene sets we can use the keyword **mutationShared**

```
parameter <- initializeParameterObject(genome = genome,
                        sphi = c(0.5, 2), num.mixtures = 2,
                        gene.assignment = sample.int(2,
```

```
                                    length(genome), replace = T),
                            mixture.definition = "mutationShared")
```

This will force all gene sets to share the same mutation parameters.

The same can be done with parameters describing selection, using the keyword **selectionShared**

```
parameter <- initializeParameterObject(genome = genome,
                            sphi = c(0.5, 2), num.mixtures = 2,
                            gene.assignment = sample.int(2,
                                    length(genome), replace = T),
                            mixture.definition = "selectionShared")
```

For more intricate compositions of gene sets, one can specify a custom $n \times 2$ matrix, where n is the number of gene sets, to describe how gene-set specific parameters should be shared. Instead of using the **mixture.definition** argument one uses the **mixture.definition.matrix** argument.

The matrix representation of **mutationShared** can be obtained by

```
# [,1] [,2]
#[1,] 1 1
#[2,] 1 2
#[3,] 1 3
def.matrix <- matrix(c(1,1,1,1,2,3), ncol=2)
parameter <- initializeParameterObject(genome = genome,
                            sphi = c(0.5, 2, 1), num.mixtures = 3,
                            gene.assignment = sample.int(3,
                                    length(genome), replace = T),
                            mixture.definition.matrix = def.matrix)
```

Columns represent mutation and selection, while each row represents a gene set. In this case we have three gene sets, each sharing the same mutation category and three different selection categories. In the same way one can produce the matrix for three independent gene sets equivalent to the **allUnique** keyword.

```
# [,1] [,2]
```

```
#[1,] 1 1
#[2,] 2 2
#[3,] 3 3
def.matrix <- matrix(c(1,2,3,1,2,3), ncol=2)
```

We can also use this matrix to produce more complex gene set compositions.

```
# [,1] [,2]
#[1,] 1 1
#[2,] 2 1
#[3,] 1 2
def.matrix <- matrix(c(1,2,1,1,1,2), ncol=2)
```

In this case gene set one and three share their mutation parameters, while gene set one and two share their selection parameters.

Checkpointing

AnaCoDa does provide checkpointing functionality in case runtime has to be restiricted. To enable checkpointing, one can use the function **setRestartSettings**.

```
# writing a restart file every 1000 samples
setRestartSettings(mcmc, "restart_file", 1000, write.multiple=TRUE)
# writing a restart file every 1000 samples
# but overwriting it every time
setRestartSettings(mcmc, "restart_file", 1000, write.multiple=FALSE)
```

To re-initialize a parameter object from a restart file one can simply pass the restart file to the initialization function:

```
initializeParameterObject(init.with.restart.file="restart_file.rst")
```

Load and save parameter objects

AnaCoDa is based on C++ objects using the Rcpp [3]. This comes with the problem that C++ objects are by default not serializable and can therefore not be saved/loaded with the default R save/load functions.

AnaCoDa however, does provide functions to load and save parameter and mcmc objects. These are the only two objects that store information during a run.

```
#save objects after a run
runMCMC(mcmc = mcmc, genome = genome, model = model)
writeParameterObject(parameter = parameter, file = "parameter.Rda")
writeMCMCObject(mcmc = mcmc, file = "mcmc_out.Rda")
```

As **genome**, and **model** objects are purely storage containers, no save/load function is provided at this point, but will be added in the future.

```
#load objects
parameter <- loadParameterObject(file = "parameter.Rda")
mcmc <- loadMCMCObject(file = "mcmc_out.Rda")
```

2.3.3 File formats

protein coding sequence

Protein coding sequences are provided by fasta file with the default format. One line containing the sequence id starting with > followed by the id and one or more lines containing the sequence. The sequences are expected to have a length that is a multiple of three. If a codon can not be recognized (e.g AGN) it is ignored.

```
>YAL001C
TTGGTTCTGACTCATTAGCCAGACGAACTGGTTCAA
CATGTTTCTGACATTCATTCTAACATTGGCATTCAT
ACTCTGAACCAACTGTAAGACCATTCTGGCATTTAG
>YAL002W
TTGGAACAAAACGGCCTGGACCACGACTCACGCTCT
TCACATGACACTACTCATAACGACACTCAAATTACT
TTCCTGGAATTCCGCTCTTAGACTCAACTGTCAGAA
```

Empirical gene expression

Empirical expression or gene specific parameters are provided in a csv file format. The first line is expected to be a header describing each column. The first column is expected be the gene id, and every additional column is expected to be represent a measurement. Each row corresponds to one gene and contains all measurements for that gene, including missing values.

```
>YAL001C
ORF,DATA_1,DATA_2,...DATA_N
YAL001C,0.254,0.489,...,0.156
YAL002W,1.856,1.357,...,2.014
YAL003W,10.45,NA,...,9.564
YAL005C,0.556,0.957,...,0.758
```

Ribosome foot-printing counts

Ribosome foot-printing (RFP) counts are provided in a csv file format. The first line is expected to be a header describing each column. The columns are expected in the following order gene id, position, codon, rfpcount. Each row corresponds to a single codon with an associated number of ribosome footprints.

```
GeneID,Position,Codon,rfpCount
YBR177C, 0, ATA, 8
YBR177C, 1, CGG, 1
YBR177C, 2, GTT, 8
YBR177C, 3, CGC, 1
```

2.3.4 Analyzing and Visualizating results

Parameter estimates

After we have completed the model fitting, we are interested in the results. AnaCoDa provides functions to obtain the posterior estimate for each parameter. For gene-set specific

parameters or codon specific parameters we can use the function **getCSPEstimates**. Again we can specify for which mixture we would like the posterior estimate and how many samples should be used. **getCSPEstimates** has an optional argument filename which will cause the routine to write the result as a csv file instead of returning a **data.frame**.

```
csp_mat <- getCSPEstimates(parameter = parameter, CSP="Mutation",
                           mixture = 1, samples = 1000)
head(csp_mat)
# AA Codon Posterior 0.025% 0.975%
#1 A GCA -0.2435340 -0.2720696 -0.2165220
#2 A GCC 0.4235546 0.4049132 0.4420680
#3 A GCG 0.7004484 0.6648690 0.7351707
#4 C TGC 0.2016298 0.1679025 0.2387024
#5 D GAC 0.5775052 0.5618199 0.5936979
#6 E GAA -0.4524295 -0.4688044 -0.4356677
getCSPEstimates(parameter = parameter, filename = "mutation.csv",
                           CSP="Mutation", mixture = 1, samples = 1000)
```

To obtain posterior estimates for the gene specific parameters, we can use the function **getExpressionEstimatesForMixture**. In the case below we ask to get the gene specific parameters for all genes, and under the assumption each gene is assigned to mixture 1.

```
phi_mat <- getExpressionEstimates(parameter = parameter,
                           gene.index = 1:length(genome),
                           samples = 1000)
head(phi_mat)
# PHI log10.PHI Std.Error log10.Std.Error 0.025 0.975 log10.025 ...
#[1,] 0.2729446 -0.6188447 0.0001261525 2.362358e-04 0.07331819 ...
#[2,] 1.4221716 0.1498953 0.0001669425 5.194123e-05 1.09593642 ...
#[3,] 0.7459888 -0.1512764 0.0002313539 1.529267e-04 0.31559618 ...
#[4,] 0.6573082 -0.2030291 0.0001935466 1.400333e-04 0.31591233 ...
#[5,] 1.6316901 0.2098120 0.0001846631 4.986347e-05 1.28410352 ...
#[6,] 0.6179711 -0.2286806 0.0001744928 1.374863e-04 0.28478950 ...
```

However we can decide to only obtain certain gene parameters. in the first case we sample 100 random genes.

```
# sampling 100 genes at random
phi_mat <- getExpressionEstimates(parameter = parameter,
                          gene.index = sample(1:length(genome), 100),
                          samples = 1000)
```

Furthermore, AnaCoDa allows to calculate the selection coefficient s for each codon and each gene. We can use the function **getSelectionCoefficients** to do so. Please note, that this function returns the $log(sNe)$.

getSelectionCoefficients returns a matrix with $log(sNe)$ relative to the most efficient synonymous codon.

```
selection.coefficients <- getSelectionCoefficients(genome = genome,
                          parameter = parameter, samples = 1000)
head(selection.coefficients)
# GCA GCC GCG GCT TGC TGT GAC GAT ...
#SAKL0A00132g -0.1630284 -0.008695144 -0.2097771 0 -0.1014373 ...
#SAKL0A00154g -0.8494558 -0.045305847 -1.0930388 0 -0.5285367 ...
#SAKL0A00176g -0.4455753 -0.023764823 -0.5733448 0 -0.2772397 ...
#SAKL0A00198g -0.3926068 -0.020939740 -0.5051875 0 -0.2442824 ...
#SAKL0A00220g -0.9746002 -0.051980440 -1.2540685 0 -0.6064022 ...
#SAKL0A00242g -0.3691110 -0.019686586 -0.4749542 0 -0.2296631 ...
```

We can compare these values to the weights from the codon adaptatoin index (CAI) citepSharp1987 or effective number of codons (N_c) [10] by using the functions **getCAI-weights** and **getNcAA**.

```
cai.weights <- getCAIweights(referenceGenome = genome)
head(cai.weights)
# GCA GCC GCG GCT TGC TGT
#0.7251276 0.6282192 0.2497737 1.0000000 0.6222628 1.0000000
nc.per.aa <- getNcAA(genome = genome)
head(nc.per.aa)
```

```
# A C D E F G ...
#SAKL0A00132g 3.611111 1.000000 2.200000 2.142857 1.792453 ...
#SAKL0A00154g 1.843866 2.500000 2.035782 1.942505 1.986595 ...
#SAKL0A00176g 5.142857 NA 1.857143 1.652174 1.551724 3.122449 ...
#SAKL0A00198g 3.800000 NA 1.924779 1.913043 2.129032 4.136364 ...
#SAKL0A00220g 3.198529 1.666667 1.741573 1.756757 2.000000 ...
#SAKL0A00242g 4.500000 NA 2.095890 2.000000 1.408163 3.734043 ...
```

We can also compare the distribution of selection coefficients to the CAI values estimated from a reference set of genes.

```
selection.coefficients <- getSelectionCoefficients(genome = genome,
                          parameter = parameter, samples = 1000)
s <- exp(selection.coefficients)
cai.weights <- getCAIweights(referenceGenome = ref.genome)
codon.names <- colnames(s)
h <- hist(s[, 1], plot = F)
plot(NULL, NULL, axes = F, xlim = c(0,1),
                          ylim = range(c(0,h$counts)),
                          xlab = "s", ylab = "Frequency",
                          main = codon.names[1], cex.lab = 1.2)
lines(x = h$breaks, y = c(0,h$counts), type = "S", lwd=2)
abline(v = cai.weights[1], lwd=2, lty=2)
axis(1, lwd = 3, cex.axis = 1.2)
axis(2, lwd = 3, cex.axis = 1.2)
```

Diagnostic Plots

A first step after every run should be to determine if the sampling routine has converged. To do that, AnaCoDa provides plotting routines to visualize all sampled parameter traces from which the posterior sample is obtained. First we have to obtain the trace object stored within our parameter object. Now we can just plot the trace object. The argument what specifies which type of parameter should be plotted. Here we plot the selection parameter

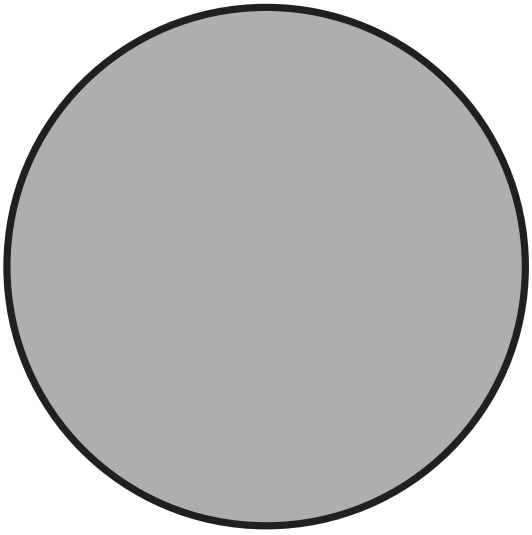


Figure 2.1: Distribution of s for codon GCA for amino acid alanine. Dashed line indicates the CAI weight for GCA. The comparisson provides a more nuanced picture as we can see that the selection on GCA varies across the genome.

$\Delta\eta$ of the ROC model. These parameters are mixture specific and one can decide which mixture set to visualize using the argument **mixture**.

```
trace <- getTrace(parameter)
plot(x = trace, what = "Mutation", mixture = 1)
```

A special case is the plotting of traces of the protein synthesis rate ϕ. As the number of traces for the different ϕ traces is usually in the thousands, a **geneIndex** has to be passed to determine for which gene the trace should be plotted. This allows to inspect the trace of every gene under every mixture assignment.

```
trace <- parameter$getTraceObject()
plot(x = trace, what = "Expression", mixture = 1, geneIndex = 669)
```

We can find the likelihood and posterior trace of the model fit in the **mcmc object**. The trace can be plotted by just passing the **mcmc** object to the **plot** routine. Again we can switch between log(*likelihood*) and log(*posterior*) using the argument **what**. The argument **zoom.window** is used to inspect a specified window in more detail. It defaults to the last 10 % of the trace. The log(*posterior*) displayed in the figure title is estimated over the **zoom.window**.

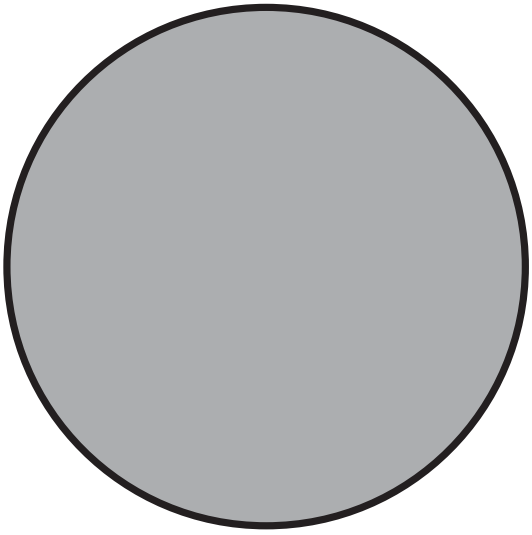


Figure 2.2: Trace plot showing the traces of all 40 codon specific selection parameters organized by amino acid.

```
plot(mcmc, what = "LogPosterior", zoom.window = c(9000, 10000))
```

Model visualization

We can visualize the results of the model fit by plotting the **model** object. For this we require the model and the **genome** object. We can adjust which mixture set we would like to visualize and how many samples should be used to obtain the posterior estimate for each parameter. For more details see Gilchrist et al. [4].

```
# use the last 500 samples from mixture 1 for posterior estimate.
plot(x = model, genome = genome, samples = 500, mixture = 1)
```

As AnaCoDa is designed with the idea to allow gene-sets to have independent gene-set specific parameters, AnaCoDa also provides the option to compare different gene-sets by plotting the parameter object. Here we compare the selection parameter estimated by ROC for seven yeast species.

```
# use the last 500 samples from mixture 1 for posterior estimate.
plot(parameter, what = "Selection", samples = 500)
```

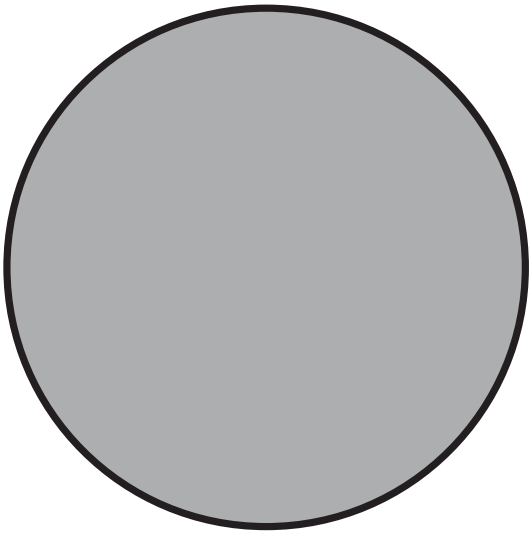


Figure 2.3: Trace plot showing the protein synthesis trace for gene 669.

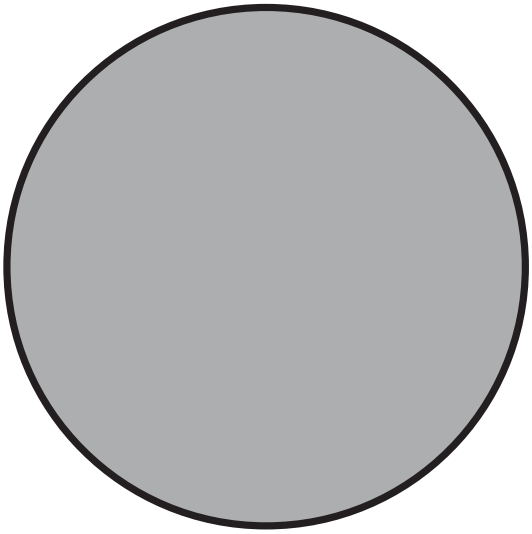


Figure 2.4: Trace plot showing the log(*posterior*) trace for the current model fit

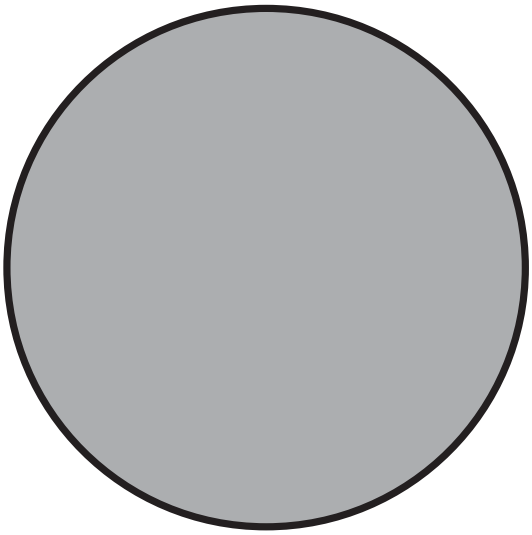


Figure 2.5: Fit of the ROC model for a random yeast. The solid line represent the model fit from the data, showing how synonymous codon frequencies change with gene expression. The points are the observed mean frequencies of a codon in that synthesis rate bin and the wisks indicate the standard diviation within the bin. The codon favored by selection is indicated by a "*". The bottom right panel shows how many genes are contained in heach bin

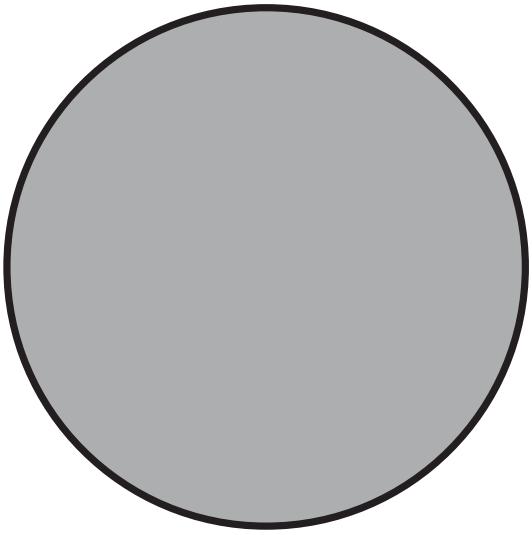


Figure 2.6: Comparisson of the selection parameter of seven yeast species estimated with ROC-SEMPPR.

Chapter 3

Fitness consequences of mismatched codon usage

Chapter 4

Conclusions

Bibliography

[1] Booch, G. (1993). *Object-oriented analysis and design with applications*. Benjamin-Cummings Publishing Co, Redwood City. 3

[2] Dunn, C., Zapata, F., Munro, C., Siebert, S., and Hejnol, A. (2018). Pairwise comparisons across species are problematic when analyzing functional genomic data. *Proc Natl Acad Sci USA*. 4

[3] Edelbuettel, D. and Francois, R. (2011). Rcpp: Seamless r and c++ integration. *Journal of Statistical Software*, 40:1–18. 3, 11

[4] Gilchrist, M., Chen, W., Shah, P., Landerer, C., and Zaretzki, R. (2015). Estimating gene expression and codon-specific translational efficiencies, mutation biases, and selection coefficients from genomic data alone. *Genome Biology and Evolution*, 7:1559–1579. 1, 3, 4, 6, 18

[5] Mi, G., Di, Y., and Schafer, D. (2015). Goodness-of-fit tests and model diagnostics for negative binomial regression of rna sequenceing data. *PLOS ONE*, 10:e0119254. 3

[6] R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. 1

[7] Shah, P. and Gilchrist, M. (2011). Explaining complex codon usage patterns with selection for translational efficiency, mutation bias, and genetic drift. *Proc Natl Acad Sci USA*, 108:10231–6. 1, 3

[8] Sharp, P. (1987). The codon adaptatoin index - a meassure of directional synonymous codon usage bias, and its potential applications. *Nucleic Acids Research*, 15:1281–1295. 1

[9] Wallace, E., Airoldi, E., and Drummond, D. (2013). Estimating selection on synonymous codon usage from noisy experimental data. *Molecular Biology and Evolution*, 30:1438–1453. 1, 3, 4

[10] Wright, F. (1990). The 'effective number of codons' used in a gene. *Gene*, 87:23–29. 1, 15

Appendices

A Summary of Equations

some text here

A.1 Cartesian

some equations here

A.2 Cylindrical

some equations also here

B Summary of Stuff

some text here

B.1 More Things

some equations here

B.2 Other Aspects

some equations also here

Vita

Cedric Landerer graduated from Heinrich-Kleyer Highschool in Frankfurt, Germany in 2006. After that he moved to Munich, Germany to study Bioinformatics in a joined major at the University of Munich and Technical University, Munich. He recieved his Bachelor of Science in 20XX and his Masters of Science in 2013. From there he moved to Knoxville, TN, USA to work on his Ph. D. at the University of Tennessee, Knoxville.