CodonU: A Python Package For Codon Usage Analysis

A Thesis submitted
in partial fulfillment of the requirement for the degree of

Bachelor Of Technology

By

Souradipto Choudhuri

Roll No. 00419031

Under the Guidance of

Keya Sau, Ph.D.

Associate Professor

Department Of Biotechnology

Haldia Institute of Technology
(Autonomous)

June, 2023

CodonU: A Python Package For Codon Usage Analysis

A Thesis submitted
in partial fulfillment of the requirement for the degree of

Bachelor Of Technology

By

Souradipto Choudhuri

Roll No. 00419031

Under the Guidance of

Keya Sau, Ph.D.

Associate Professor

Department Of Biotechnology

Haldia Institute of Technology
(Autonomous)

June, 2023

CodonU: A Python Package For Codon Usage Analysis

A Thesis submitted
in partial fulfilment of the requirement for the degree of

Bachelor Of Technology

By

Souradipto Choudhuri

Roll No. 00419031

Approved By

Keya Sau, Ph.D. Project supervisor	Suvroma Gupta, Ph.D. Head of the Department

Department Of Biotechnology

Haldia Institute of Technology
(Autonomous)

June, 2023

Certificate By Supervisor

This is to certify that the thesis entitled, "CodonU: A Python Package for Codon Usage Analysis" submitted by Souradipto Choudhuri (00419031) in partial fulfillment for the requirements for the award of Bachelor of Technology Degree in Biotechnology Engineering from Haldia Institute of Technology, Haldia, 721657, is an authentic work carried out by them under my supervision and guidance.

To the best of our knowledge, the matter embodied in the thesis has not been submitted to any other University / Institute for the award of any Degree or Diploma.

Signature:

Supervisor
Keya Sau, Ph.D.
Associate Professor
Department of Biotechnology
Haldia Institute of Technology
Date: June 13, 2023

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Souradipto Choudhuri (00419031)

Date: June 13, 2023

Acknowledgement

On the very outset of this report, I would like to extend my sincere and heartfelt obligation towards all the personages who helped us in this endeavor. Without their active help, cooperation and encouragement, I would not have made headway in the project.

I am ineffably indebted to my supervisor *Prof. Dr. Keya Sau* for her conscientious guidance and encouragement to accomplish this project. I am extremely thankful and pay my gratitude to her for her valuable guidance and support on completion of this project in its present form.

I extend my thanks towards the *Department of Biotechnology* as well as *Haldia Institute of Technology* for giving me this opportunity.

I also acknowledge with a deep sense of reverence, my gratitude towards my parents, who has always supported me morally as well as economically.

At last but not least gratitude goes to all of my friends who directly or indirectly helped me to complete this project report.

Any omission in this brief acknowledgement does not mean a lack of gratitude.

Thanking You,
Souradipto Choudhuri

CodonU
A Python Package for Codon Usage Analysis

Souradipto Choudhuri

Abstract

Codon Usage Analysis has been accompanied by several web servers and independent programs written in several programming languages. Also this diversity speaks for the need of a reusable software that can be helpful in reading, manipulating and acting as a pipeline for such data and file formats. Most popular software for these kind of analyses is CodonW. But it has its limited scopes and a complex pipeline for data analysis. So, we propose CodonU, a package written in python language. It is compatible with existing file formats and can be used solely or with a group of other such packages. The proposed package incorporates various statistical measures necessary for codon usage analysis. The measures vary with nature of the sequences, viz. for nucleotide, Codon Adaptation Index (CAI), Codon Bias Index (CBI) etc. and for protein sequences Gravy score etc. Users can also perform the Correspondence Analysis (COA). This package also provides the liberty to generate graphics to users, and also perform phylogenetic analysis which is out of scope for CodonW. Capabilities of the proposed package were checked thoroughly on a diverse genomic set. Detailed documentation and some examples for this open-source project is available at GitHub: https://www.github.com/SouradiptoC/CodonU

Acronyms

AA	Amino Acid
Aromaticity	Over all aromaticity of the protein
C	C Programming Language
CADD	Computer Aided Drug Design
CAI	Codon Adaptation Index
CBI	Codon Bias Index
CDS	Coding DNA Sequence
COA	Correspondence Analysis
CUA	Codon Usage Analysis
CUTG	Codon usage tabulated from the GenBank genetic sequence data database
$\mathbf{EN_c}$	Effective Number of Codon
$\mathbf{F_{op}}$	Frequency of Optimal Codons
gbk	GenBank File
GRAVY	Overall hydrophobicity of protein
GUI	Graphic User Interface
JS	JavaScript Programming Language
Nuc	Nucleotide
PCA	Principal Component Analysis
Python	Python Programming Language
R	R Programming Language
RSCU	Relative Synonymous Codon Usage

Contents

Approval i

Certification By Supervisor ii

Declaration iii

Acknowledgement iv

Abstract v

Acronyms vi

I Introduction 1

1 Motivation 2
1.1 Introduction . . . 2
1.2 CodonW . . . 2
1.3 Workflow . . . 3
1.4 Proposal . . . 3

2 Objective 4
2.1 Introduction . . . 4
2.2 Using Python Programming Language . . . 4
2.3 Objectives . . . 5

3 Literature Survey 7
3.1 Tools for Correspondence Analysis . . . 7
3.2 Tools for Visualizing COA . . . 7
3.3 Early Databases . . . 8
3.4 Early Tools for CUA . . . 8
3.5 New Tools for CUA . . . 9

II Theoretical Analysis 10

4 Statistical Measures 11
4.1 Biological Viewpoint . . . 11

4.2 Statistical Measures for Nuc Sequence . . . 11
4.2.1 Relative Synonymous Codon Usage . . . 11
4.2.2 Codon Adaptation Index . . . 12
4.2.3 Codon Bias Index . . . 13
4.2.4 Effective Number of Codon . . . 14
4.3 Statistical Measures for AA Sequence . . . 16
4.3.1 Overall hydrophobicity of protein Score . . . 16
4.3.2 Over all aromaticity of the protein Score . . . 16

5 Correspondence Analysis 17
5.1 Introduction . . . 17
5.2 Correspondence Analysis . . . 17
5.2.1 Methodology . . . 17
5.2.2 Interpretation . . . 18
5.2.3 One Example . . . 18
5.3 Principal Component Analysis . . . 18
5.3.1 Methodology . . . 19
5.3.2 Interpretation . . . 19
5.4 COA in CUA . . . 19
5.4.1 Methodology . . . 19
5.4.2 Interpretation . . . 20
5.5 Categorization of COA Based on Sequence . . . 20
5.6 Application of COA in CUA . . . 20

III Implementation 21

6 Implementation 22
6.1 Introduction . . . 22
6.2 Required Packages . . . 22
6.3 Third Party Softwares . . . 22

7 Extractor 23
7.1 Introduction . . . 23
7.2 Functions . . . 23

8 File Handler 25
8.1 Introduction . . . 25
8.2 Functions . . . 25

9 Analyzer 27
9.1 Introduction . . . 27
9.2 Functions . . . 27

10 Correspondence Analysis 32
10.1 Introduction . . . 32
10.2 Functions . . . 32

11 Phylogenetic Analysis **34**
11.1 Introduction . . . 34
11.2 Functions . . . 34

12 Vizualizer **36**
12.1 Introduction . . . 36
12.2 Function . . . 36

IV Conclusion **43**

13 Conclusion **44**

V Bibliography **45**

Bibliography **46**
Articles . . . 46
Books . . . 49
Book Chapters . . . 49
Thesis . . . 50
Softwares, Online Tools, etc. . . . 50

VI Communication **51**

List of Figures

5.1 Example of COA 18

12.1 EN_c and Neutrality Plot 37
12.2 Parity Rule 2 and Phylogenetic Tree Plot 38
12.3 Phylogenetic Tree Plot 39
12.4 COA Plots of Codon 41
12.5 COA Plots of AA 42

List of Code Snippets

7.1 extract_cds . . . 23
7.2 extract_cds_lst . . . 23
7.3 extract_exome . . . 23
7.4 extract_exome . . . 24
8.1 get_gb . . . 25
8.2 write_exome_fasta . . . 25
8.3 write_nucleotide_fasta . . . 25
8.4 write_protein_fasta . . . 26
8.5 set_entrez_param . . . 26
9.1 calculate_cai . . . 27
9.2 caululate_cbi . . . 28
9.3 calculate_enc . . . 29
9.4 calculate_rscu . . . 29
9.5 calculate_aromaticity . . . 30
9.6 calculate_gravy . . . 30
10.1 mca_codon_freq . . . 32
10.2 mca_codon_rscu . . . 33
10.3 mca_aa_freq . . . 33
11.1 generate_phylo_input . . . 34
11.2 phy_clustal_w . . . 34
11.3 phy_clustal_o . . . 35
12.1 plot_enc . . . 36
12.2 plot_neutrality . . . 36
12.3 plot_pr2 . . . 37
12.4 plot_phy_dnd . . . 37
12.5 plot_phy_fas . . . 38
12.6 plot_phy_nex . . . 38
12.7 plot_mca_codon_freq . . . 39
12.8 plot_mca_rscu . . . 40
12.9 plot_mca_aa_gravy . . . 41
12.10 plot_mca_aa_aroma . . . 42

Part I

Introduction

Chapter 1

Motivation

1.1 Introduction

Codon Usage Analysis (CUA) is a topic of investigation for many decades now. This is nothing but various biases observed in the genome of various species. In a very simplified way, synonymous codons are sets of codons that code for the same Amino Acid (AA). While the resulting protein remains unchanged, the preferential usage of particular codons can vary across species, genes, and even within different regions of a single gene. This phenomenon has been the subject of investigation in the field of bioinformatics for several decades.

The topic of the thesis is in silico analysis of CUA. The term '*In Silico*' has a fascinating origin. Tracing the etymology of the word, in silico shares it origin with *in vitro* and *in vivo*. In vitro means the experiments that are done in a controlled environment mimicking living organism. Just as so, *silico* in the word 'in silico' refers to the silicon used in computer chips. The word was first used by Christopher Gale Langton (as reported by Hameroff 1987). He used the term for describing artificial life while pitching about a workshop on the topic at Los Alamos National Laboratory in 1987. Though it was not written in a scientific literature until 1990. It was first used in a scientific literature by Sieburg 1990. The phrase in silico originally applied only to computer simulations that modeled natural or laboratory processes (in all the natural sciences), and did not refer to calculations done by computer generically. But now it also envelops the calculations. 'In silico' is highly used in Systems' biology, Network biology, CADD etc in modern days.

1.2 CodonW

The problem of CUA was addressed by various researchers in the in silico approach. But arguably, most popular among them was developed by Peden 1999. Peden developed the tool as his Ph.D. project. He also made it public as a software named as `CodonW` (Pedan 2005).

`CodonW` was well received by the scientific community from its release. It was written in `C`. It could calculate CAI, CBI, EN_c, F_{op} for Nuc sequences. For AA sequences, it can calculate GRAVY score and Aromaticity score. In case of COA, it can perform codon usage, amino acid usage and RSCU.

1.3 Workflow

Though `CodonW` was very popular and still in use for various research projects, the workflow needed for working with `CodonW` is intertwined. A workflow using `CodonW` may consist of:

1. First the gbk needs to be downloaded from genome database (NCBI 1992).
2. Then the downloaded files are uploaded to coderet (A. Bleasby 2000). From there the CDS sections are exported.
3. The exported sequences are then fed in `CodonW`.
4. The results are then interpreted by the user.

The interpretation can not be done with standalone `CodonW`. Various other softwares need to be used for statistical significance checking and phylogenetic analysis like `SPSS` (IBM. Corp 2009) and `Mega` (Tamura, Stecher, and Kumar 2021; MEGA Dev Team 2021) respectively. Also `CodonW` uses much more disk space and yeilds various files which can not be manipulated with current softwares.

1.4 Proposal

Previous section speaks for the need of a novel software for CUA. Also the limitations faced by `CodonW` is not mitigated by its developer. When I originally started to work on CUA of *Staphylococcus sp.*, I faced the complex workflow and come to know about the limitations faced. These serve as the motivation for developing `CodonU`. Details of the mentioned software is discussed in the later chapters.

Chapter 2

Objective

2.1 Introduction

The motivation for writing `CodonU` was driven by the need to simplify and enhance the process of codon usage analysis in genomic research. The existing tools and software available at the time had limitations and complexities that hindered efficient analysis and hindered the exploration of codon usage patterns.

One of the key motivations was to provide researchers with a user-friendly and comprehensive package that consolidates various steps in codon usage analysis. By merging these steps and streamlining the workflow, `CodonU` aims to reduce the burden on researchers and enable them to focus more on the interpretation and analysis of results.

Another motivation was to address the challenges associated with file compatibility and data integration. `CodonU` was designed to be compatible with existing file formats, making it easier for researchers to handle and manipulate their data. This compatibility also allows for seamless integration with other software packages, providing researchers with a flexible and powerful analytical environment.

2.2 Using Python Programming Language

`CodonU` was written in the Python (Python Dev Team 2001) due to several compelling reasons. Python is a popular and widely-used language in the field of scientific research and data analysis, offering a range of benefits that align with the objectives of `CodonU` development.

Firstly, Python is known for its simplicity and readability, making it easier for developers and users to understand and work with the codebase. This simplicity enables efficient coding, debugging, and maintenance of the software, which is crucial for a tool like `CodonU` that aims to be user-friendly and accessible.

Secondly, Python has a rich ecosystem of libraries and packages specifically designed for scientific computing and bioinformatics, such as Biopython (Cock et al. 2009; Biopython Dev Team 2021), NumPy (Harris et al. 2020; NumPy Dev Team 2023), Pandas (McKinney 2010; Pandas Dev Team 2022), and Matplotlib (Hunter 2007; Matplotlib Dev Team 2023). These libraries provide pre-built functions and tools that greatly facilitate the implementation of various functionalities within

`CodonU`. Leveraging these existing packages not only saves development time but also ensures that `CodonU` benefits from the well-established and extensively tested functionalities of these libraries. Furthermore, Python offers excellent cross-platform compatibility, allowing `CodonU` to run seamlessly on different operating systems. This versatility is essential for ensuring broad accessibility and enabling researchers to utilize `CodonU` on their preferred computing environment.

Additionally, Python's popularity within the scientific community makes it a preferred choice for collaborative research and future development. Its widespread adoption means that other researchers and developers are more likely to be familiar with Python, increasing the potential for collaboration, code contributions, and the incorporation of new features into `CodonU`.

Overall, choosing Python as the programming language for `CodonU` combines its simplicity, extensive library support, cross-platform compatibility, and strong community engagement. These factors collectively contribute to the usability, flexibility, and future growth potential of `CodonU` as a valuable tool for codon usage analysis in genomic research.

2.3 Objectives

The primary objective of developing CodonU was to address the challenges and limitations faced by researchers in performing codon usage analysis. The following objectives guided the development of CodonU:

1. Simplifying Workflow: CodonU aimed to simplify the complex and time-consuming workflow involved in codon usage analysis. By integrating various steps and functionalities into a single software package, CodonU streamlines the analysis process, reducing the effort and expertise required from users.

2. Enhanced User-Friendliness: The objective was to create a user-friendly tool that enables researchers, even those without extensive bioinformatics background, to perform codon usage analysis effectively. CodonU provides an intuitive interface and clear documentation, making it accessible to a wide range of users.

3. Interoperability and Compatibility: CodonU sought to ensure interoperability and compatibility with existing file formats and software tools commonly used in codon usage analysis. By supporting widely used formats and incorporating popular third-party software, CodonU facilitates seamless integration into existing research pipelines.

4. Statistical Measures and Analysis: CodonU aimed to provide a comprehensive set of statistical measures and analysis methods relevant to codon usage analysis. These measures include CAI, CBI, COA, and more. By offering a range of analytical capabilities, CodonU enables researchers to gain valuable insights into codon usage patterns.

5. Visualization Capabilities: The objective was to empower users with visualization tools for effective data exploration and presentation. CodonU enables

the generation of high-resolution graphics, suitable for publication and sharing, aiding in the interpretation and communication of codon usage analysis results.

Overall, the objective of CodonU was to provide a comprehensive, user-friendly, and flexible software tool that simplifies codon usage analysis, promotes interoperability, and facilitates valuable insights into genomic research.

Chapter 3

Literature Survey

3.1 Tools for Correspondence Analysis

Codon Usage Analysis as previously said, has been a topic of discussion for many decades now. There are various practical application for CUA. One of the most earliest review works on practical usage of CUA was done by P. M. Sharp and Cowe 1991. The work by Nesti et al. 1995 also brought forward the usage of CUA for studying the phylogenetic relation of 31 organisms. But these works were tedious in nature because of lack of softwares. Most of the calculations had to be done by hand and only the complex calculations are done by computer. This was partially due to technological constrains of that time.

Correspondence Analysis is a part of principal component analysis. This in term is part of Multi variate analysis. In early 80s or 90s, after the advancements in computers, various computer programs were written by scholars for solving the problems of mentioned field. But they were also complex and time taking. One of the famous programs named `DECORANA` (`DE`trended `COR`respondence `ANA`lysis) for correspondence analysis was written by Hill 1979. Later `DECORANA` was updated and put forward by Hill and Gauch 1980. Now `DECORANA` is also available as `R` package (Oksanen 2022). Another software named `CORAN` was also popular for COA. It was developed and described by Lebart, Morineau, and Warwick 1984, and was popularized by a review on it by Duncombe 1985. But no dedicated software for using Correspondence Analysis in Codon Usage Analysis was developed at that time.

3.2 Tools for Visualizing COA

Researchers who were working on CUA needed the visualization of the COA data rather than just the numerical values. The proposal for building softwares which can help visualize the COA data was done by many but the proposal of Thioulouse 1989 was most popular. Within a year of the proposal, two softwares named `MacMul` and `GraphMu` was announced by Thioulouse 1990. The former was a software for `Macintosh` machines, which can calculate the COA data. The latter, one of first of its kind, was a software which can help visualize the data and produce graphics. With the advancements of world wide web, the developers put their effort in developing

a variant of the program which can work online. As a result of their dedication and effort, NetMul was finally developed by Thioulouse and Chevenet 1996.

3.3 Early Databases

Despite the upgradation in computational tools, the usage of such tools were scarcely noticed for CUA. One of the problems was integrating two ends of the distant fields. A common parser was needed for reading the sequence data and then manipulating it. Also the availability of genetic data had caused serious hindrance in such analyses. This problem was later mitigated with the development of the database GenBank by NCBI 1992. A more specific database for CUA was proposed when genbank database was set up, and shortly it was initiated Wada et al. 1990. After more than half a decade of its announcement, the database named CUTG (available at http://www.kazusa.or.jp/codon/) for CUA was developed by Yasukazu Nakamura, Wada, et al. 1996, quite the same time when NetMul was developed. It became popular among researchers and a detailed report including the vision and future implementations was published in a series of works by Yusuke Nakamura, Gojobori, and Ikemura 1997; Yusuke Nakamura, Gojobori, and Ikemura 1998; Yasukazu Nakamura, Gojobori, and Ikemura 1999; Yasukazu Nakamura, Gojobori, and Ikemura 2000.

3.4 Early Tools for CUA

Some programs were written to to calculate various statistical measures viz. CAI, CBI etc. However, feeding data to these tools were complex. Most general programs for this usage was done by CODONS by Lloyd and P. Sharp 1992 written in FORTRAN (ANSI 1978). One of the most promising softwares for analysis of biological data and make it publishable was GCG developed by university of Wisconsin in 1994 (Womble 1999).

First breakthrough for softwares in CUA was CodonW by Peden 1999. A software named GCUA was developed a year earlier by McInerney 1998 but was not that much popular as CodonW. In the 2000s, with growing popularity of web servers, SMS (Sequence Manipulation Suite) was implemented by Stothard 2000. The suite is developed in JS, which was optimal for implementation. In this suite, Codon Usage (available at https://www.bioinformatics.org/sms2/codon_usage.html) was implemented which can calculate the usage bias. During the same time, famous suite for bioinformatics analysis named EMBOSS was developed by Rice, Longden, and A. J. Bleasby 2000. Coderet mentioned earlier is a part of this suite.

One problem with softwares and web servers at this point was the absence of GUI. For mitigating this problem, Jemboss was developed by Carver and A. J. Bleasby 2003. This was the GUI based EMBOSS. Another GUI based tool named ACUA (Automated Codon Usage Analysis) was implemented by Vetrivel, Arunkumar, and Dorairaj 2007.

3.5 New Tools for CUA

With advancements in computational power and popularity of modern programming languages, various researchers tried to implement softwares for CUA with new philosophies. There were various problems with previously used tools. Most of them were written in `FORTRAN` or `C`. They work fine but is not efficient if seen from the perspective of time consumption. Also, with advancements in hardware, old tools became hard to use. The popular `CodonW` was developed for the 32 bit machines. Also new generation of researchers are habituated with modern programming languages. Hence it is hard for them to upgrade the old tools. Rather, development of new tools are easy for them. Some of recent advancements in the field are the proposal and implementation of `CUBAP` (`C`odon `U`sage `B`iases `A`cross `P`opulations) web tool developed by Hodgman et al. 2020. It is an interactive web portal for analyzing codon usage bias. Various programs like `codon-usage` (Diament 2023) now exist for CUA but most of them are difficult to work with and written in languages that are not very common to the researchers of CUA.

Part II

Theoretical Analysis

Chapter 4

Statistical Measures

4.1 Biological Viewpoint

The genetic code consists of 64 codons including 3 stop codons (viz. *UAA*, *UAG*, *UGA*). Out of 61 codon, *Met* and *Trp*, are encoded by a single codon (*AUG* and *UGG*, respectively). The remaining 59 codons encode 18 amino acids. This subset of codons that encode for the same amino acid is known as *synonymous* codons. Within the subset of synonymous codons, a bias may be observed for the preference of a single codon, which can be species-specific and is referred to as the *preferred* codon. When analyzing codon bias, two hypothetical events are often considered. These are:

- *No Bias* (H^0): The first hypothetical event assumes that all 20 amino acids are encoded equally by the 61 codons. In this scenario, there is no observed bias in the codon usage for encoding amino acids, hence the term "no bias" event.
- *Extreme Bias* (H^*): The second hypothetical event assumes that all 20 amino acids are encoded by only 20 codons, with extreme bias in the codon usage for encoding amino acids. This scenario is referred to as the "extreme bias" event.

4.2 Statistical Measures for Nuc Sequence

For Nuc sequence, the measures which are taken account of are RSCU, CAI, CBI and EN_c. Details about these measures are discussed in section.

4.2.1 Relative Synonymous Codon Usage

Theoretical Aspect

The concept of RSCU was introduced by P. M. Sharp and Li 1987. RSCU is calculated as the ratio of the observed frequency of a codon to the expected frequency of the codon, assuming no bias in the codon usage. This metric is widely used to

evaluate codon bias and is often used as a starting point for further analyses. Hence,

$$RSCU_{ij} = \frac{x_{ij}}{\frac{1}{n_i}\sum_{j=1}^{n_i} x_{ij}} \tag{4.1}$$

where x_{ij} is the observed frequency of j^{th} codon for i^{th} amino acid. n_i is the number of other codons present in the subset of synonymous codons for i^{th} amino acid. The minimum value of RSCU is 0. If $RSCU > 1$, it indicates a positive bias for that particular codon, while $RSCU < 1$ indicates a negative bias. If $RSCU = 1$, it indicates that the codon is used as expected under the assumption of no bias.

Values of RSCU

Proof. Let's consider a H^0 situation. Hence in complete random state all codons for a SF group will be used equally. So for i^{th} amino acid, if there exists n codon (SFn), then

$$\frac{1}{n}\sum_{j=1}^{n} x_{ij} = \frac{n \times x_{ij}}{n} = \bar{x} = x_{ij}$$

Hence from Equation 4.1 and the above mentioned result,

$$\begin{aligned} RSCU_{H^0} &= \frac{x_{ij}}{\frac{1}{n}\sum_{j=1}^{n} x_{ij}} \\ &= \frac{x_{ij}}{\bar{x}} \\ &= \frac{x_{ij}}{x_{ij}} = 1 \end{aligned}$$

If $RSCU > 1$, then $x_{ij} > \bar{x}$ and there exists a positive correlation and vice-versa. Hence,

$$RSCU = \begin{cases} > 1, & +ve \text{ correlation} \\ < 1, & -ve \text{ correlation} \end{cases}$$

□

4.2.2 Codon Adaptation Index

Theoretical Aspect

CAI is a quantitative measure that provides a more precise assessment of codon usage bias in comparison to RSCU. It is computed as the ratio of the geometric mean of the observed RSCU values to the maximum possible geometric mean of RSCU. This measure of bias was first proposed by P. M. Sharp and Li 1987. Hence,

$$CAI = \frac{CAI_{obs}}{CAI_{max}} \tag{4.2}$$

where

$$CAI_{obs} = \left(\prod_{k=1}^{L} RSCU_k\right)^{\frac{1}{L}} \tag{4.3}$$

and

$$CAI_{max} = \left(\prod_{k=1}^{L} RSCU_{kmax} \right)^{\frac{1}{L}} \tag{4.4}$$

where $RSCU_k$ is the RSCU value for k^{th} codon, $RSCU_{kmax}$ is the maximum RSCU value for the amino acid encoded by k^{th} codon. L is the number of codons present in the gene. The value of CAI lies between 0 (extreme bias) to 1 (no bias).

Values of CAI

Proof. Let's consider a H^0 situation. Then as discussed in previous section, $RSCU_{H^0} = 1$. Hence from Equation 4.3 and Equation 4.4,

$$CAI_{obsH^0} = CAI_{maxH^0} = 1$$

Hence form Equation 4.2 and previously discussed results,

$$CAI_{H^0} = \frac{CAI_{obsH^0}}{CAI_{maxH^0}} = \frac{1}{1} = 1$$

Again let's consider a H^* situation. Then, for a subset of synonymous codons, only the preferred codon will be present and rest of elements will have a frequency of 0. Hence, if k is a synonymous but not preferred codon,

$$\begin{aligned} & RSCU_{H^*} && = \tfrac{0}{\frac{1}{n}\sum_{j=1}^{n_i} x_{ij}} = 0 \\ \implies & CAI_{obsH^*} && = 0 \\ \implies & CAI_{H^*} && = \tfrac{0}{CAI_{maxH^*}} = 0 \end{aligned}$$

Hence,

$$CAI = \begin{cases} 0, & \text{if } H^* \\ 1, & \text{if } H^0 \end{cases}$$

□

4.2.3 Codon Bias Index

Theoretical Aspect

The CBI is a measure of codon bias from the perspective of a specific AA. It was initially proposed by Bennetzen and Hall 1982. Due to its quantitative nature and ease of interpretation, it gained popularity quickly. CBI is calculated as the ratio of the occurrence of the preferred codon minus the occurrence of the preferred codon in a non-biased situation to the occurrence of synonymous codons minus the latter part of the numerator. Hence,

$$CBI = \frac{n_{opt} - n_0}{n_{syn} - n_0} \tag{4.5}$$

where n_{opt} is the occurrence of preferred codon, n_{syn} is the total occurrence of other codons in the synonymous subset, and n_0 is expected occurrence in no bias situation. The value of CBI lies between 0 (extreme bias) to 1 (no bias).

Values of CBI

Proof. Let's consider a H^0 situation. Then by the consideration we assume that every codon can be considered as the preferred codon. Hence, if there are n codons present, then

$$n_{opt} = n_{syn} = n_0 = n \quad \text{and} \quad \lim_{n\to 0} \frac{n}{n} = 1$$

Hence from Equation 4.5 and the above stated fact,

$$CBI_{H^0} = \frac{n_0 - n_0}{n_0 - n_0} = 1$$

Again let's consider a H^* situation, and our codon of interest is not the preferred codon. Then $n_{opt} = 0$ and $n_{syn} >> n_0$ considering $n_{syn} >> 1$. Hence from Equation 4.5,

$$\begin{aligned} CBI_{H^*} &= \frac{0 - n_0}{n_{syn} - n_0} \\ &= -\frac{n_0}{n_{syn} - n_0} \quad < 0 \\ &\approx 0 \end{aligned}$$

Hence

$$CBI = \begin{cases} 0, & \text{if } H^* \\ 1, & \text{if } H^0 \end{cases}$$

□

4.2.4 Effective Number of Codon

$\mathrm{EN_c}$ is a measure of codon usage bias that takes into account both the number of synonymous codons for each AA and their relative frequencies. It was first introduced by Wright 1990 and has since been widely used in bioinformatics research. Codons can be classified into different groups based on the number of synonymous codons that encode a particular AA. This grouping is known as the *synonymous family* or *SF* categorization. For example, AAs encoded by only one synonymous codon belong to SF1, while those encoded by two synonymous codons belong to SF2, and so on. The number of elements in each SF category is denoted by F_1, F_2, F_3, etc.

ENc is defined as the effective number of codons used by a gene, taking into account the different frequencies of codons in each SF category. Mathematically, it is calculated as:

$$ENc = \frac{2}{F_1} + \frac{9}{F_2} + \frac{1}{F_3} + \frac{5}{F_4} + \frac{3}{F_6} \tag{4.6}$$

where F_i is the arithmetic mean of homozygosity for SF type i, and is defined as:

$$F_i = \frac{n \sum_{i=1}^{k} p_i^2 - 1}{n - 1} \quad \text{where} \quad p_i = \frac{n_i}{n} \tag{4.7}$$

where k is the number of codons present in the subset and n is total number of codons present. If there are total 4 synonymous codons present in the subset (SF4) then, $k = 4$ $n_1 + n_2 + n_3 + n_4 = n$, $k = 4$, $p_1 = \frac{n_1}{n}$ and so forth.

It is worth noting at this point that SF3 contains only *Ile* which is encoded from AUU, AUC and AUA. It is not highly unlikely that any of the mentioned codons may be absent in a gene. This is known as *missing codon* problem. Wright proposed to compute F_3 by calculating the average of F_2 and F_4. But no proof was provided by him, and it was based on intuition. Later it was pointed out that though it looks correct at first glance, but is wrong by Fuglsang 2004. He proposed that, in case of missing codon,

$$\begin{aligned} F_3 &= (\tfrac{2}{F_2} - 1)^{-1} && \text{if } F_2 \neq 0 \\ F_3 &= \tfrac{(\frac{2}{F_2}-1)^{-1}+(\frac{2}{3F_4}+\frac{1}{3})^{-1}}{2} && \text{if } [F_2, F_4] \neq 0 \\ F_3 &= \tfrac{(\frac{2}{F_2}-1)^{-1}+(\frac{2}{3F_4}+\frac{1}{3})^{-1}+(\frac{2}{5F_6}+\frac{3}{5})^{-1}}{3} && \text{if } [F_2, F_4, F_6] \neq 0 \end{aligned}$$

CodonU implements this function. The value of ENc lies between 20 (extreme bias) to 61 (no bias).

Values of $\mathbf{EN_c}$

Proof. Let's consider a H^0 situation. If there exists k (SFk) codons in the subset and total number of codons is n, then

$$\sum_{i=1}^{k} n_i = n \quad \text{and} \quad p_i = \frac{n_i}{n} = \frac{n_i}{k \times n_i} = \frac{1}{k}$$

Hence for that subset according to Equation 4.7,

$$\begin{aligned} F &= \frac{n \sum_{i=1}^{k} p_i^2 - 1}{n - 1} \\ &= \sum_{i=1}^{k} p_i^2 \qquad \text{as } n >> 1 \\ &= k \times p_i^2 \\ &= k \times \left(\frac{1}{k}\right)^2 \\ &= \frac{1}{k} \end{aligned}$$

Hence $F_1 = 1$, $F_2 = \frac{1}{2}$, $F_3 = \frac{1}{3}$, $F_4 = \frac{1}{4}$, $F_6 = \frac{1}{6}$. Then by Equation 4.6,

$$\begin{aligned} ENc_{H^0} &= \frac{2}{F_1} + \frac{9}{F_2} + \frac{1}{F_3} + \frac{5}{F_4} + \frac{3}{F_6} \\ &= 2 + 18 + 3 + 20 + 18 \\ &= 61 \end{aligned}$$

Again let's consider a H^* situation. For SFk, in this case,

$$p_i = \begin{cases} 0, & \text{if codon is not preferred} \\ 1, & \text{if codon is preferred} \end{cases}$$

Hence for the subset according to Equation 4.7,

$$\begin{aligned} F &= \frac{n\sum_{i=1}^{k} p_i^2 - 1}{n-1} \\ &= \sum_{i=1}^{k} p_i^2 \qquad \text{as } n >> 1 \\ &= 1 \end{aligned}$$

Then by Equation 4.6,

$$\begin{aligned} ENc_{H^*} &= \frac{2}{F_1} + \frac{9}{F_2} + \frac{1}{F_3} + \frac{5}{F_4} + \frac{3}{F_6} \\ &= 2 + 9 + 1 + 5 + 3 \\ &= 20 \end{aligned}$$

Hence

$$ENc = \begin{cases} 20, & \text{if } H^* \\ 61, & \text{if } H^0 \end{cases}$$

□

4.3 Statistical Measures for AA Sequence

4.3.1 Overall hydrophobicity of protein Score

Gravy score is a measure of the hydrophobicity of a protein sequence. While the measures mentioned previously apply to nucleotide sequences, the gravy score is specifically for protein sequences. There are various scales available to compute this score, but the most widely used scale was proposed by Kyte and Doolittle 1982. CodonU implements this scale to calculate the gravy score.

4.3.2 Over all aromaticity of the protein Score

Aromaticity score is a metric used to assess the abundance of aromatic amino acids, such as *Phe*, *Tyr*, and *Trp*, in a protein sequence. This score provides insight into the aromatic nature of the protein. Various scales have been proposed to calculate this score, and CodonU employs the scale developed by Lobry and Gautier 1994.

Chapter 5

Correspondence Analysis

5.1 Introduction

Correspondence Analysis (COA) and Principal Component Analysis (PCA) are part of multivariate calculus. These techniques are widely used in various fields, including statistics, data science, social sciences, and marketing research. Below is an overview of COA and PCA, followed by a step-by-step explanation of their methodologies. The discussion on interpretation of results, applications, and practical considerations are after that. Later we will discuss its application in the field of CUA.

5.2 Correspondence Analysis

COA is a technique used to analyze categorical data and explore the relationships between different categories. It is particularly useful when dealing with large contingency tables, where rows and columns represent categories and cells contain frequency counts or proportions. COA aims to summarize and visualize the association patterns within the data (Greenacre 2017; Husson and Josse 2010).

5.2.1 Methodology

COA involves the following steps:

- Preprocessing: Transforming the original contingency table into an appropriate format (e.g., frequency counts or proportions).
- Computing row and column profiles: Calculating the marginal frequencies or proportions for rows and columns.
- Calculating standardized residuals: Assessing the deviations between observed and expected cell frequencies.
- Dimensionality reduction: Constructing a low-dimensional space using singular value decomposition (SVD) or other matrix factorization techniques.
- Visualizing results: Plotting the row and column points in the reduced space using scatter plots or biplots.

5.2.2 Interpretation

The interpretation of COA results involves analyzing the proximity of categories in the plot, the relative positions of points, and the contribution of variables to the dimensions. CA is commonly applied in market research, linguistics, social sciences, and ecology. It helps identify underlying patterns, relationships, and associations in categorical data.

5.2.3 One Example

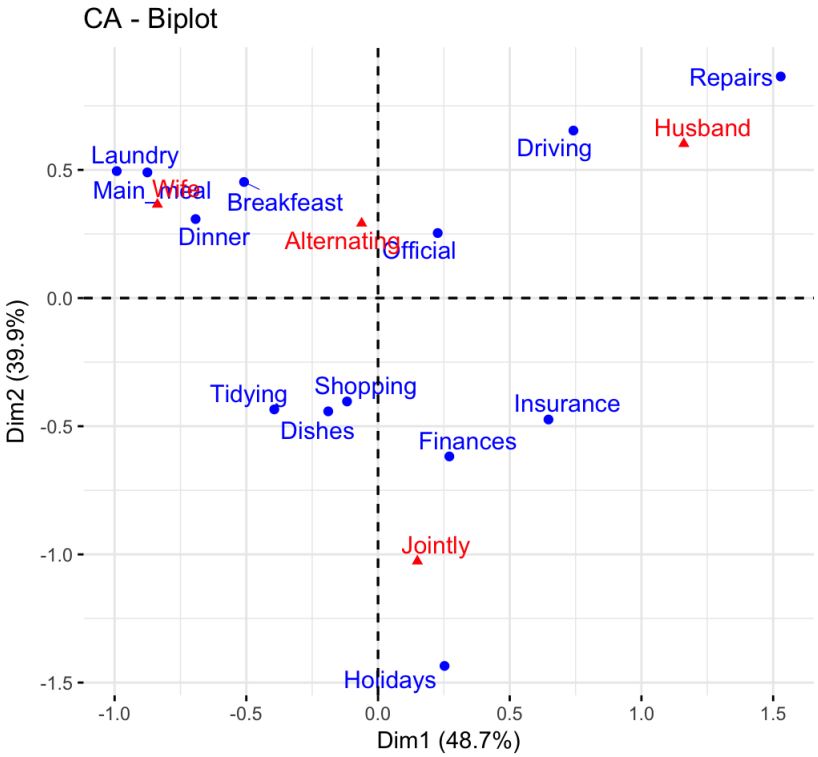


Figure 5.1: Example of COA

The picture shows the results of a correspondence analysis on a contingency table that shows the frequency of occurrence of two categorical variables, gender and political party affiliation. The rows of the contingency table are represented by the blue points in the biplot, and the columns are represented by the red points. The distances between the points are proportional to the association between the variables.

5.3 Principal Component Analysis

PCA is a dimensionality reduction technique used to transform high-dimensional data into a lower-dimensional space while preserving the most important information. PCA aims to identify the principal components that capture the maximum variability in the data (Jolliffe 2002; Abdi and Williams 2010).

5.3.1 Methodology

PCA involves the following steps:

- Data standardization: Scaling variables to have zero means and unit variances.
- Computing the covariance matrix: Calculating the covariance or correlation matrix of the standardized data.
- Eigenvalue decomposition: Obtaining eigenvalues and eigenvectors of the covariance matrix.
- Selection of principal components: Sorting eigenvectors based on eigenvalues and selecting the top components.
- Dimensionality reduction: Constructing the reduced-dimensional space by projecting the data onto the selected components.

5.3.2 Interpretation

Interpreting PCA results involves understanding the contribution of variables to each principal component, analyzing loadings, and examining the variance explained by each component. PCA is widely used in fields such as finance, genetics, image processing, and pattern recognition. It aids in data visualization, data compression, and feature extraction.

5.4 COA in CUA

Codon usage refers to the frequency of occurrence of different codons in the coding sequences of genes. Understanding codon usage patterns provides insights into various biological processes such as gene expression, translation efficiency, and evolutionary dynamics. COA offers a powerful statistical approach to analyze and visualize codon usage data, aiding in the identification of underlying patterns and factors influencing codon preferences (P. M. Sharp and Li 1987; Angov 2011).

5.4.1 Methodology

The steps involved in performing COA for CUA are as follows:

- Data preparation: Create a codon usage table where rows represent genes and columns represent codons. The table entries can be counts, relative frequencies, or other appropriate measures.
- Preprocessing: Normalize the data to remove biases arising from gene lengths or other factors. Common normalization methods include RSCU and CAI.
- Dimensionality reduction: Apply COA to the codon usage table, typically using singular value decomposition or other matrix factorization techniques.
- Visualization: Plot the genes and codons in a low-dimensional space, such as a scatter plot or biplot, to explore the relationships between codons and genes.

5.4.2 Interpretation

The interpretation of COA results in codon usage analysis involves analyzing the proximity of codons and genes in the plot, identifying clusters or groups of genes with similar codon usage patterns, and exploring the contributions of codons to the dimensions. COA helps in understanding the factors shaping codon usage preferences, such as mutational biases, selection pressures, or gene expression levels.

5.5 Categorization of COA Based on Sequence

The application of COA differs for Nuc and AA sequences (Lobry 2018).

- In the case of DNA or RNA, COA works by calculating the relative frequencies of the nucleotides or codons and plotting them in a multidimensional space. The distances between the points in the space represent the degree of similarity between the sequences. Mathematically, this works as a $\mathbb{R}^{59} \rightarrow \mathbb{R}^2$ projection. CodonU implements this projection in terms of codon frequency and codon RSCU value.
- In the case of AA sequences, COA works by calculating the frequency of occurrence of each AA and plotting them in a multidimensional space. The distances between the points in the space represent the degree of similarity between the sequences based on their AA composition. Mathematically, this works as a $\mathbb{R}^{20} \rightarrow \mathbb{R}^2$ projection. CodonU implements this projection in terms of AA frequency.

5.6 Application of COA in CUA

The application COA in specific contexts of codon usage analysis, such as:

- Comparative Genomics: Exploring codon usage variations across different species or genomes, identifying evolutionary trends, and understanding the impact of selective pressures.
- Gene Expression Analysis: Investigating codon usage patterns in relation to gene expression levels, translational efficiency, or regulatory mechanisms.
- Viral Evolution and Vaccine Development: Analyzing codon usage biases in viral genomes, identifying codon preferences for efficient protein expression, and designing synthetic genes for vaccine development.
- Phylogeny Analysis: With the help of D^2 statistics, phylogeny analysis can be done.

Part III

Implementation

Chapter 6

Implementation

6.1 Introduction

`CodonU` is implemented in Python. The package is further divided in 6 parts or subpackages for easy usage and maintenance. The role of each subpackage is to perform a special type work which can be reflected in their names. The subpackages are: `extractor`, `file_handler`, `analyzer`, `correspondence_analysis`, `phylogenetic_analysis`, `vizualizer`.

6.2 Required Packages

Preference for using Python has already been discussed and most of used packages have also been discussed. Apart from them, `Bio.Phylo` (Talevich et al. 2012) is used for displaying and constructing phylogenetic trees. `CAI` (Lee 2018b; Lee 2018a) is used for calculating CAI and RSCU. `SciPy` (Virtanen et al. 2019; Scipy Dev Team 2023) is used for calculating the trends. `Prince` (Halford 2020) is used for PCA.

6.3 Third Party Softwares

`CodonU` does not implement phylogenetic analysis directly. Instead, it utilizes popular and readily available software, namely `Clustal` by Larkin et al. 2007; Sievers and Higgins 2018. `CodonU` supports two variants of `Clustal`, namely, `ClustalW` (also known as `ClustalX`) and `ClustalΩ`. Users can perform phylogenetic analysis using these software variants, by having the corresponding binary files in their system, which can be obtained from the official website of Clustal at http://www.clustal.org. The choice of not implementing phylogenetic analysis from scratch and using popular software ensures the reliability and accuracy of the analysis, while also reducing the burden of implementation and maintenance on the developers.

Chapter 7

Extractor

7.1 Introduction

The objective of this package is to extract CDS and AA information. The details are discussed below. Basic function names and some code snippets are provided for ease of readers.

7.2 Functions

`extract_cds`: This is used to extract the CDS from a given sequence.

```
def extract_cds(record: SeqRecord, feature_location: FeatureLocation, cds_no: int =
     0) -> SeqRecord:
    """
    Returns the CDS as a Sequence Record object
    """
    cds = SeqRecord(
        seq=extract_cds_seq(record.seq, feature_location),
        id=f"{record.id} {record.annotations['organism']}",
        name=f"{record.annotations['organism']}",
        description=f"CDS_{cds_no}"
    )
    return cds
```

Code Snippet 7.1: extract_cds

`extract_cds_lst`: This is used to extract the CDS location list from a given sequence.

```
def extract_cds_lst(record: SeqRecord) -> tuple[Any, ...]:
    """
    Extracts the list of features if their type is CDS
    """
    cds_lst = [cds for cds in record.features if cds.type == 'CDS' and 'pseudo' not
     in cds.qualifiers.keys()]
    return tuple(cds_lst)
```

Code Snippet 7.2: extract_cds_lst

`extract_exome`: This is used to extract the exome.

```
def extract_exome(nuc_file_path: str, organism_name: str) -> SeqRecord:
    """
    Extracts the exome from given nucleotides
    """
    records = parse(nuc_file_path, 'fasta')
    m_seq = MutableSeq('')
```

```
    for record in records:
        m_seq += record.seq[:-3]
    records = parse(nuc_file_path, 'fasta')
    *_, lst_record = records
    m_seq += lst_record.seq[-3:]
    exome = SeqRecord(
        seq=m_seq,
        id=lst_record.id,
        name=organism_name,
        description=f'whole exome of {organism_name}'
    )
    return exome
```

Code Snippet 7.3: extract_exome

`extract_prot`: This is used to extract the protein sequence.

```
def extract_prot(feature: SeqFeature, organism_name: str, cds_no: int = 0) ->
    SeqRecord:
    """
    Extracts protein sequences and return them for writing
    """
    if 'product' in feature.qualifiers.keys():
        description = f"{feature.qualifiers['product'][0]} CDS_{cds_no}"
    else:
        description = f"{feature.qualifiers['note'][0]} CDS_{cds_no}"
    prot = SeqRecord(
        seq=extract_prot_seq(feature),
        id=f"{feature.qualifiers['protein_id'][0]}",
        name=f"{organism_name}",
        description=description
    )
    return prot
```

Code Snippet 7.4: extract_exome

Chapter 8

File Handler

8.1 Introduction

The objective of this package is to handle files. The details are discussed below. Basic function names and some code snippets are provided for ease of readers.

8.2 Functions

`get_gb`: This is used to retrieve gbk file.

```
def get_gb(accession_id: str) -> SeqRecord:
    """
    Gets the Sequence Record object from a given accession number
    """
    handle = Entrez.efetch(db='nucleotide', id=accession_id, rettype='gb', retmode='text')
    record = read(handle, 'gb')
    return record
```

Code Snippet 8.1: get_gb

`write_exome_fasta`: Creates fasta file of exome.

```
def write_exome_fasta(file_name: str, nuc_file_path: str, organism_name: str) -> None:
    """
    Creates a fasta file of exome if not exists previously or is empty
    """
    if not is_file(file_name) or is_file_empty(file_name):
        with open(file_name, 'w') as out_file:
            exome = extract_exome(nuc_file_path, organism_name)
            write(exome, out_file, 'fasta')
    print(f"Exome file for {organism_name} created successfully")
```

Code Snippet 8.2: write_exome_fasta

`write_nucleotide_fasta`: Creates fasta file of nucleotide.

```
def write_nucleotide_fasta(file_name: str, cds_lst: tuple, record: SeqRecord, organism_name: str) -> None:
    """
    Creates a fasta file of nucleotides if not exists previously or is empty
    """
    if not is_file(file_name) or is_file_empty(file_name):
        with open(file_name, 'w') as out_file:
            for i in range(len(cds_lst)):
                cds = extract_cds(record, cds_lst[i], i + 1)
                write(cds, out_file, 'fasta')
```

```
10      print(f"Nucleotide file for {organism_name} created successfully")
```

Code Snippet 8.3: write_nucleotide_fasta

`write_protein_fasta`**:** Creates fasta file of protein.

```
1  def write_protein_fasta(file_name: str, cds_lst: tuple, organism_name: str) -> None
       :
2      """
3      Creates a fasta file of proteins if not exists previously or is empty
4      """
5      if not is_file(file_name) or is_file_empty(file_name):
6          with open(file_name, 'w') as out_file:
7              for i in range(len(cds_lst)):
8                  cds = extract_prot(cds_lst[i], organism_name, i + 1)
9                  write(cds, out_file, 'fasta')
10     print(f"Protein file for {organism_name} created successfully")
```

Code Snippet 8.4: write_protein_fasta

`set_entrez_param`**:** Set entrez parameter.

```
1  def set_entrez_param(email: str | None = None, api_key: str | None = None) -> None:
2      """
3      Sets entrez parameters
4      """
5      set_entrez_email(email)
6      set_entrez_api_key(api_key)
```

Code Snippet 8.5: set_entrez_param

Chapter 9

Analyzer

9.1 Introduction

The objective of this package is to analyze sequence data and calculate various statistical measures. The details are discussed below. Basic function names and some code snippets are provided for ease of readers.

9.2 Functions

`calculate_cai`: This is used to calculate CAI.

```
def calculate_cai(handle: str, genetic_code_num: int, min_len_threshold: int = 200,
     gene_analysis: bool = False,
                  save_file: bool = False, file_name: str = 'CAI_report',
    folder_path: str = 'Report') -> \
        dict[str, float | dict[str, float]]:
    """
    Calculates cai values for each codon
    """
    filterwarnings('ignore')
    cai_dict = dict()
    records = parse(handle, 'fasta')
    reference = filter_reference(records, min_len_threshold)
    if gene_analysis:
        for i, seq in enumerate(reference):
            cai_val_dict = dict()
            for codon in unambiguous_dna_by_id[genetic_code_num].forward_table:
                cai_val = CAI(codon, reference=[seq], genetic_code=genetic_code_num
    )
                cai_val_dict.update({codon: cai_val})
            cai_dict.update({f'gene_{i + 1}': cai_val_dict})
        if save_file:
            name = file_name + '.xlsx'
            make_dir(folder_path)
            file_path = join(folder_path, name)
            if is_file_writeable(file_path):
                df = pd.DataFrame.from_records(
                    [
                        (gene, codon, cai_val)
                        for gene, cai_vals in cai_dict.items()
                        for codon, cai_val in cai_vals.items()
                    ],
                    columns=['Gene', 'Codon', 'CAI_vals']
                )
                df.to_excel(file_path, float_format='%.4f', columns=df.columns)
            print(f'The CAI score file can be found at: {abspath(file_path)}')
    else:
```

```
        for codon in unambiguous_dna_by_id[genetic_code_num].forward_table:
            cai_val = CAI(codon, reference=reference, genetic_code=genetic_code_num
    )
            cai_dict.update({codon: cai_val})
        if save_file:
            name = file_name + '.xlsx'
            make_dir(folder_path)
            file_path = join(folder_path, name)
            if is_file_writeable(file_path):
                df = pd.DataFrame(cai_dict.items(), columns=['Codon', 'CAI_vals'])
                df.to_excel(file_path, float_format='%.4f', columns=df.columns)
            print(f'The CAI score file can be found at: {abspath(file_path)}')
    return cai_dict
```

Code Snippet 9.1: calculate_cai

`calculate_cbi`: This is used to calculate CBI.

```
def calculate_cbi(handle: str, genetic_code_num: int, min_len_threshold: int = 66,
    gene_analysis: bool = False,
                      save_file: bool = False, file_name: str = 'CBI_report',
    folder_path: str = 'Report') -> \
        dict[str, tuple[float, str] | dict[str, tuple[float, str]]]:
    """
    Calculates cbi values for each amino acid
    """
    records = parse(handle, 'fasta')
    reference = filter_reference(records, min_len_threshold)
    filterwarnings('ignore')
    cbi_dict = dict()
    if gene_analysis:
        for i, seq in enumerate(reference):
            cbi_val_dict = dict()
            for aa in unambiguous_dna_by_id[genetic_code_num].protein_alphabet:
                cbi_val = cbi(aa, reference=[seq], genetic_code=genetic_code_num)
                cbi_val_dict.update({aa: cbi_val})
            cbi_dict.update({f'gene_{i + 1}': cbi_val_dict})
        if save_file:
            name = file_name + '.xlsx'
            make_dir(folder_path)
            file_path = join(folder_path, name)
            if is_file_writeable(file_path):
                df = pd.DataFrame.from_records(
                    [
                        (gene, aa, cbi_val[0], cbi_val[1])
                        for gene, cbi_vals in cbi_dict.items()
                        for aa, cbi_val in cbi_vals.items()
                    ],
                    columns=['Gene', 'AA', 'CBI_vals', 'Preferred_Codon']
                )
                df.to_excel(file_path, float_format='%.4f', columns=df.columns)
            print(f'The CBI score file can be found at: {abspath(file_path)}')
    else:
        for aa in unambiguous_dna_by_id[genetic_code_num].protein_alphabet:
            cbi_val = cbi(aa, reference, genetic_code_num)
            cbi_dict.update({aa: cbi_val})
        if save_file:
            name = file_name + '.xlsx'
            make_dir(folder_path)
            file_path = join(folder_path, name)
            if is_file_writeable(file_path):
                df = pd.DataFrame.from_records(
                    [
                        (aa, cbi_vals[0], cbi_vals[1])
                        for aa, cbi_vals in cbi_dict.items()
                    ],
                    columns=['AA', 'CBI_vals', 'Preferred_Codon']
                )
                df.to_excel(file_path, float_format='%.4f', columns=df.columns)
            print(f'The CBI score file can be found at: {abspath(file_path)}')
```

```
    return cbi_dict
```

Code Snippet 9.2: caululate_cbi

`calculate_enc`: This is used to calculate EN_c.

```
def calculate_enc(handle: str, genetic_code_num: int, min_len_threshold=200,
    gene_analysis: bool = False,
                      save_file: bool = False, file_name: str = 'ENc_report',
    folder_path: str = 'Report') -> \
        float or dict[str, float]:
    """
    Calculates ENc value for a given sequences
    """
    records = parse(handle, 'fasta')
    references = filter_reference(records, min_len_threshold)
    filterwarnings('ignore')
    if gene_analysis:
        enc_dict = dict()
        for i, seq in enumerate(references):
            enc_dict.update({f'gene_{i + 1}': enc([seq], genetic_code_num)})
        if save_file:
            name = file_name + '.xlsx'
            make_dir(folder_path)
            file_path = join(folder_path, name)
            if is_file_writeable(file_path):
                df = pd.DataFrame(enc_dict.items(), columns=['Gene', 'ENc_val'])
                df.to_excel(file_path, float_format='%.4f', columns=df.columns)
            print(f'The ENc score file can be found at: {abspath(file_path)}')
        return enc_dict
    else:
        if save_file:
            name = file_name + '.xlsx'
            make_dir(folder_path)
            file_path = join(folder_path, name)
            if is_file_writeable(file_path):
                df = pd.DataFrame({'Genome': enc(references, genetic_code_num)}.
    items(), columns=['Genome', 'ENc_vals'])
                df.to_excel(file_path, float_format='%.4f', columns=df.columns)
            print(f'The ENc score file can be found at: {abspath(file_path)}')
        return enc(references, genetic_code_num)
```

Code Snippet 9.3: calculate_enc

`calculate_rscu`: This is used to calculate RSCU.

```
def calculate_rscu(handle: str, genetic_code_num: int, min_len_threshold: int =
    200, gene_analysis: bool = False,
                      save_file: bool = False, file_name: str = 'RSCU_report',
    folder_path: str = 'Report') -> \
        dict[str, float | dict[str, float]]:
    """
    Calculates rscu values for each codon
    """
    records = parse(handle, 'fasta')
    references = filter_reference(records, min_len_threshold)
    if gene_analysis:
        rscu_dict = dict()
        for i, seq in enumerate(references):
            rscu_dict.update({f'gene_{i + 1}': RSCU([seq], genetic_code_num)})
        if save_file:
            name = file_name + '.xlsx'
            make_dir(folder_path)
            file_path = join(folder_path, name)
            if is_file_writeable(file_path):
                df = pd.DataFrame.from_records(
                    [
                        (gene, codon, rscu_val)
                        for gene, rscu_vals in rscu_dict.items()
                        for codon, rscu_val in rscu_vals.items()
                    ],
```

```
                    columns=['Gene', 'Codon', 'RSCU_vals']
                )
                df.to_excel(file_path, float_format='%.4f', columns=df.columns)
            print(f'The RSCU score file can be found at: {abspath(file_path)}')
    else:
        reference = filter_reference(records, min_len_threshold)
        rscu_dict = RSCU(reference, genetic_code_num)
        if save_file:
            name = file_name + '.xlsx'
            make_dir(folder_path)
            file_path = join(folder_path, name)
            if is_file_writeable(file_path):
                df = pd.DataFrame.from_records(
                    [
                        (codon, rscu_val)
                        for codon, rscu_val in rscu_dict.items()
                    ],
                    columns=['Codon', 'RSCU_vals']
                )
                df.to_excel(file_path, float_format='%.4f', columns=df.columns)
            print(f'The RSCU score file can be found at: {abspath(file_path)}')
    return rscu_dict
```

Code Snippet 9.4: calculate_rscu

`calculate_aromaticity`: This is used to calculate Aromaticity.

```
def calculate_aromaticity(handle: str, min_len_threshold: int = 66, gene_analysis: bool = False,
                          save_file: bool = False, file_name: str = 'Aroma_report', folder_path: str = 'Report') -> \
        dict[str, float] | float:
    """
    Calculates the aromaticity score for a given protein sequence
    """
    records = parse(handle, 'fasta')
    references = filter_reference(records, min_len_threshold)
    if gene_analysis:
        aroma_dict = dict()
        for i, seq in enumerate(references):
            aroma_dict.update({f'prot_seq{i + 1}': aromaticity(seq)})
        if save_file:
            name = file_name + '.xlsx'
            make_dir(folder_path)
            file_path = join(folder_path, name)
            if is_file_writeable(file_path):
                df = pd.DataFrame(aroma_dict.items(), columns=['Protein_name', 'Aroma_score'])
                df.to_excel(file_path, float_format='%.4f', columns=df.columns)
            print(f'The Aromaticity score file can be found at: {abspath(file_path)}')
        return aroma_dict
    else:
        seq = ''.join([str(_seq) for _seq in references])
        if save_file:
            name = file_name + '.xlsx'
            make_dir(folder_path)
            file_path = join(folder_path, name)
            if is_file_writeable(file_path):
                df = pd.DataFrame({'Prot_seq': aromaticity(seq)}.items(), columns=['Protein_name', 'Aroma_score'])
                df.to_excel(file_path, float_format='%.4f', columns=df.columns)
            print(f'The Aromaticity score file can be found at: {abspath(file_path)}')
        return aromaticity(seq)
```

Code Snippet 9.5: calculate_aromaticity

`calculate_gravy`: This is used to calculate GRAVY.

```python
def calculate_gravy(handle: str, min_len_threshold: int = 66, gene_analysis: bool = False, save_file: bool = False,
                    file_name: str = 'GRAVY_report', folder_path: str = 'Report') -> dict[str, float] | float:
    """
    Calculates the gravy score for a given protein sequence
    """
    records = parse(handle, 'fasta')
    references = filter_reference(records, min_len_threshold)
    filterwarnings('ignore')
    if gene_analysis:
        gravy_dict = dict()
        for i, seq in enumerate(references):
            gravy_dict.update({f'prot_seq{i + 1}': gravy(seq)})
        if save_file:
            name = file_name + '.xlsx'
            make_dir(folder_path)
            file_path = join(folder_path, name)
            if is_file_writeable(file_path):
                df = pd.DataFrame(gravy_dict.items(), columns=['Protein_name', 'Gravy_score'])
                df.to_excel(file_path, float_format='%.4f', columns=df.columns)
            print(f'The GRAVY score file can be found at: {abspath(file_path)}')
        return gravy_dict
    else:
        seq = ''.join([str(_seq) for _seq in references])
        if save_file:
            name = file_name + '.xlsx'
            make_dir(folder_path)
            file_path = join(folder_path, name)
            if is_file_writeable(file_path):
                df = pd.DataFrame({'Prot_seq': gravy(seq)}.items(), columns=['Protein_name', 'Gravy_score'])
                df.to_excel(file_path, float_format='%.4f', columns=df.columns)
            print(f'The GRAVY score file can be found at: {abspath(file_path)}')
        return gravy(seq)
```

Code Snippet 9.6: calculate_gravy

Chapter 10

Correspondence Analysis

10.1 Introduction

The objective of this package is to help in correspondence analysis. The details are discussed below. Basic function names and some code snippets are provided for ease of readers.

10.2 Functions

`mca_codon_freq`: This is used to perform COA with codon frequency data.

```
def mca_codon_freq(handle: str, genetic_table_num: int, min_len_threshold: int =
    200, n_components: int = 59) -> \
        tuple[pd.DataFrame, np.ndarray]:
    """
    Calculates the contingency table and the inertia from codon frequency of gene
    """
    records = parse(handle, 'fasta')
    references = filter_reference(records, min_len_threshold)
    codons = [codon for codon, _ in unambiguous_dna_by_id[genetic_table_num].
    forward_table.items()]
    gene_names = [f'gene_{i}' for i in range(len(references))]
    contingency_table = pd.DataFrame(index=gene_names, columns=codons)
    for idx, gene in enumerate(references):
        sequences = ((sequence[i:i + 3].upper() for i in range(0, len(sequence), 3)
    ) for sequence in [gene])
        _codons = chain.from_iterable(sequences)
        counts = Counter(_codons)
        for codon in codons:
            try:
                contingency_table[codon][gene_names[idx]] = counts[codon]
            except KeyError:
                warn = NoCodonWarning(codon)
                warn.warn()
                contingency_table[codon][gene_names[idx]] = 0
    pca = PCA(random_state=42, n_components=n_components)
    pca.fit(contingency_table)
    print('The inertia for respective components are:')
    for idx, inertia in enumerate(pca.explained_inertia_):
        print(f'Axis {idx + 1}: {inertia}')
    return contingency_table, pca.explained_inertia_
```

Code Snippet 10.1: mca_codon_freq

`mca_codon_rscu`: This is used to perform COA with codon RSCU data.

```python
def mca_codon_rscu(handle: str, genetic_table_num: int, min_len_threshold: int = 200, n_components: int = 59) -> \
        tuple[pd.DataFrame, np.ndarray]:
    """
    Calculates the contingency table and the inertia from RSCU of every codon of every genes
    """
    codons = [codon for codon, _ in unambiguous_dna_by_id[genetic_table_num].forward_table.items()]
    rscu_dict = calculate_rscu(handle, genetic_table_num, min_len_threshold, gene_analysis=True)
    gene_names = list(rscu_dict.keys())
    contingency_table = pd.DataFrame(index=gene_names, columns=codons)
    for gene in gene_names:
        for codon in codons:
            contingency_table[codon][gene] = rscu_dict[gene][codon]
    pca = PCA(random_state=42, n_components=n_components)
    pca.fit(contingency_table)
    print('The inertia for respective components are:')
    for idx, inertia in enumerate(pca.explained_inertia_):
        print(f'Axis {idx + 1}: {inertia}')
    return contingency_table, pca.explained_inertia_
```

Code Snippet 10.2: mca_codon_rscu

`mca_aa_freq`: This is used to perform COA with AA frequency data.

```python
def mca_aa_freq(handle: str, genetic_table_num: int, min_len_threshold: int = 66, n_components: int = 20) -> \
        tuple[pd.DataFrame, np.ndarray]:
    """
    Calculates the contingency table and the inertia from amino acid frequency of protein
    """
    records = parse(handle, 'fasta')
    references = filter_reference(records, min_len_threshold)
    aas = [aa for aa, _ in unambiguous_dna_by_id[genetic_table_num].back_table.items()]
    aas.remove(None)  # as None is also returned to the back_table
    prot_names = [f'prot_{i}' for i in range(len(references))]
    contingency_table = pd.DataFrame(index=prot_names, columns=aas)
    for idx, prot in enumerate(references):
        for aa in aas:
            contingency_table[aa][prot_names[idx]] = prot.count(aa)
    pca = PCA(random_state=42, n_components=n_components)
    pca.fit(contingency_table)
    print('The inertia for respective components are:')
    for idx, inertia in enumerate(pca.explained_inertia_):
        print(f'Axis {idx + 1}: {inertia}')
    return contingency_table, pca.explained_inertia_
```

Code Snippet 10.3: mca_aa_freq

Chapter 11

Phylogenetic Analysis

11.1 Introduction

The objective of this package is to help in phylogenetic analysis. The details are discussed below. Basic function names and some code snippets are provided for ease of readers.

11.2 Functions

`generate_phylo_input`: This is used to generate the input for phylogenetic analysis.

```
def generate_phylo_input(file_lst: list[str], file_name: str = 'phylo_input',
    folder_path: str = 'Report'):
    """
    Generates the alignment input file
    """
    make_dir(folder_path)
    file_path = join(folder_path, file_name) + '.fasta'
    if is_file_writeable(file_path):
        for in_file in file_lst:
            records = read(in_file, 'fasta')
            write(records, open(file_path, 'a'), 'fasta')
        print(f'The alignment input file can be found at {abspath(file_path)}')
```

Code Snippet 11.1: generate_phylo_input

`phy_clustal_w`: This is used to perform phylogenetic analysis with `ClustalW`.

```
def phy_clustal_w(bin_path: str, handle: str, res_folder_path: str = 'Report'):
    """
    Makes the multiple sequence alignment with ClustalW. For details visit http://
    www.clustal.org/clustal2
    """
    make_dir(res_folder_path)
    identifier = handle.split('/')[-1].split('.')[0]
    report_file_name = f'{identifier}_aligned_w.nex'
    report_file_path = join(res_folder_path, report_file_name)
    if not is_file(report_file_path) or is_file_empty(report_file_path):
        clustalW_cline = ClustalwCommandline(bin_path, infile=handle, outfile=
    report_file_path, output='NEXUS',
                                             outputtree='nexus')
        cmd = clustalW_cline.__str__()
        subprocess.run(cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.
    PIPE, text=True)
        print(f'The alignment file can be can be found at: {abspath(
    report_file_path)}')
```

Code Snippet 11.2: phy_clustal_w

`phy_clustal_o`: This is used to perform phylogenetic analysis with `ClustalΩ`.

```
def phy_clustal_o(bin_path: str, handle: str, res_folder_path: str = 'Report'):
    """
    Makes the multiple sequence alignment with ClustalO. For details visit http://
    www.clustal.org/omega
    """
    make_dir(res_folder_path)
    identifier = handle.split('/')[-1].split('.')[0]
    report_file_name = f'{identifier}_aligned_o.fasta'
    report_file_path = join(res_folder_path, report_file_name)
    if not is_file(report_file_path) or is_file_empty(report_file_path):
        clustalO_cline = ClustalOmegaCommandline(bin_path, infile=handle, infmt='
    fasta', outfile=report_file_path,
                                                 outfmt='fasta', seqtype='DNA')
        cmd = clustalO_cline.__str__()
        subprocess.run(cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.
    PIPE, text=True)
        print(f'The alignment file can be can be found at: {abspath(
    report_file_path)}')
```

Code Snippet 11.3: phy_clustal_o

Chapter 12

Vizualizer

12.1 Introduction

The objective of this package is to help in visualizing the data. The details are discussed below. Basic function names and some code snippets are provided for ease of readers.

12.2 Function

`plot_enc`: This is used to plot EN_c.

```
def plot_enc(handle: str | Any, genetic_table_num: int, min_len_threshold: int =
    200, organism_name: None | str = None,
                save_image: bool = False, folder_path: str = 'Report',
    gene_analysis: bool = True):
    """
    Plots ENc curve from given fasta file
    """
    filterwarnings('ignore')
    records = parse(handle, 'fasta')
    reference = filter_reference(records, min_len_threshold)
    enc_val_lst = []
    gc3_val_lst = []
    for seq in reference:
        enc_val_lst.append(enc([seq], genetic_table_num))
        gc3_val_lst.append(gc_123(seq)[-1] / 100)

    _plot_enc(enc_val_lst, gc3_val_lst, organism_name, save_image, folder_path,
    gene_analysis)
```

Code Snippet 12.1: plot_enc

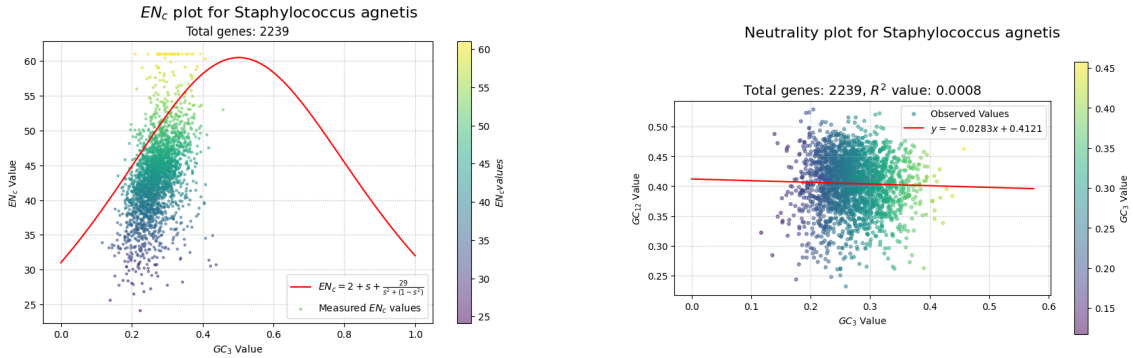
`plot_neutrality`: This is used to plot neutrality.

```
def plot_neutrality(handle: str | Any, min_len_threshold: int, organism_name: str |
     None = None,
                    save_image: bool = False, folder_path: str = '', gene_analysis:
     bool = True):
    """
    Plots neutrality plot from given fasta file
    """
    filterwarnings('ignore')
    records = parse(handle, 'fasta')
    reference = filter_reference(records, min_len_threshold)
    gc_12_lst = []
    gc_3_lst = []
    for seq in reference:
```

```
        _, gc_1, gc_2, gc_3 = gc_123(seq)
        # taking avg of gc_1 and gc_2
        gc_12_lst.append((gc_1 + gc_2) / 2 / 100)
        gc_3_lst.append(gc_3 / 100)

    _plot_neutrality(gc_12_lst, gc_3_lst, organism_name, save_image, folder_path,
    gene_analysis)
```

Code Snippet 12.2: plot_neutrality



(a) EN_c Plot

(b) Neutrality Plot

Figure 12.1: EN_c and Neutrality Plot

plot_pr2: This is used to plot parity rule 2.

```
def plot_pr2(handle: str | Any, min_len_threshold: int, organism_name: str | None =
    None, save_image: bool = False,
                 folder_path: str = '', gene_analysis: bool = True):
    """
    Plots A3/AT3 values against G3/GC3 values from given fasta file
    """
    filterwarnings('ignore')
    records = parse(handle, 'fasta')
    reference = filter_reference(records, min_len_threshold)
    gc3_val_lst = []
    g3_val_lst = []
    at3_val_lst = []
    a3_val_lst = []
    for seq in reference:
        _, _, _, gc3 = gc_123(seq)
        _, _, _, at3 = at_123(seq)
        g_3 = g3(seq)
        a_3 = a3(seq)
        gc3_val_lst.append(gc3 / 100)
        at3_val_lst.append(at3 / 100)
        g3_val_lst.append(g_3 / 100)
        a3_val_lst.append(a_3 / 100)

    _plot_pr2(gc3_val_lst, at3_val_lst, g3_val_lst, a3_val_lst, organism_name,
    save_image, folder_path, gene_analysis)
```

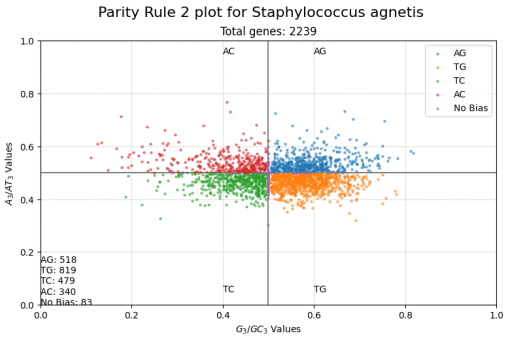
Code Snippet 12.3: plot_pr2

plot_phy_dnd: This is used to plot phylogenetic tree from .dnd file.

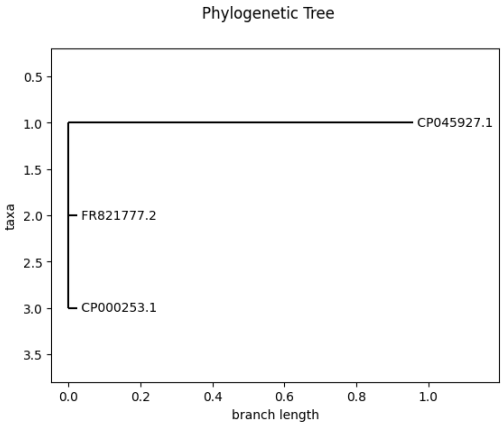
```
def plot_phy_dnd(handle: str, title: str = 'Phylogenetic Tree', save_image: bool =
    False, folder_path: str = 'Report'):
    """
    Plots phylogenetic tree from dnd file
    """
    tree = read(handle, 'newick')
    tree = tree.as_phyloxml()
```

```
    fig, ax = plt.subplots()
    fig.suptitle(title)
    draw(tree, axes=ax)
    if save_image:
        make_dir(folder_path)
        file_name = '_'.join(title.split()) + '_dnd.png'
        file_path = join(folder_path, file_name)
        if is_file_writeable(file_path):
            fig.savefig(file_path, dpi=500)
            print(f'Saved file can be found as {abspath(file_path)}')
    plt.close(fig)
```

Code Snippet 12.4: plot_phy_dnd



(a) Parity Rule 2 Plot



(b) Phylogenetic Plot (.dnd)

Figure 12.2: Parity Rule 2 and Phylogenetic Tree Plot

plot_phy_fas: This is used to plot phylogenetic tree from .fasta file.

```
def plot_phy_fas(handle: str, title: str = 'Phylogenetic Tree', save_image: bool =
    False, folder_path: str = 'Report'):
    """
    Plots phylogenetic tree from fasta file
    """
    cons = DistanceTreeConstructor()
    aln_file = read(open(handle), 'fasta')
    calc = DistanceCalculator('identity')
    distance_matrix = calc.get_distance(aln_file)
    tree = cons.upgma(distance_matrix)
    tree = tree.as_phyloxml()
    fig, ax = plt.subplots()
    fig.suptitle(title)
    draw(tree, axes=ax)
    if save_image:
        make_dir(folder_path)
        file_name = '_'.join(title.split()) + '_fas.png'
        file_path = join(folder_path, file_name)
        if is_file_writeable(file_path):
            fig.savefig(file_path, dpi=500)
            print(f'Saved file can be found as {abspath(file_path)}')
    plt.close(fig)
```

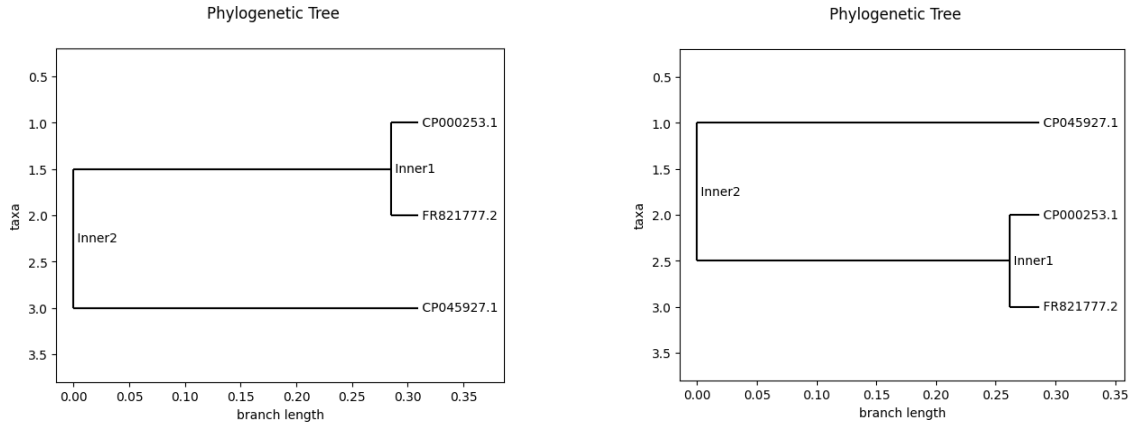
Code Snippet 12.5: plot_phy_fas

plot_phy_nex: This is used to plot phylogenetic tree from .nexus file.

```
def plot_phy_nex(handle: str, title: str = 'Phylogenetic Tree', save_image: bool =
    False, folder_path: str = 'Report'):
    """
```

```
    Plots phylogenetic tree from nexus file
    """
    cons = DistanceTreeConstructor()
    aln_file = read(open(handle), 'nexus')
    calc = DistanceCalculator('identity')
    distance_matrix = calc.get_distance(aln_file)
    tree = cons.upgma(distance_matrix)
    tree = tree.as_phyloxml()
    fig, ax = plt.subplots()
    fig.suptitle(title)
    draw(tree, axes=ax)
    if save_image:
        make_dir(folder_path)
        file_name = '_'.join(title.split()) + '_nex.png'
        file_path = join(folder_path, file_name)
        if is_file_writeable(file_path):
            fig.savefig(file_path, dpi=500)
            print(f'Saved file can be found as {abspath(file_path)}')
    plt.close(fig)
```

Code Snippet 12.6: plot_phy_nex



(a) Phylogenetic Plot (.fasta)

(b) Phylogenetic Plot (.nex)

Figure 12.3: Phylogenetic Tree Plot

`plot_mca_codon_freq`: This is used to plot COA of codon frequency data.

```
def plot_mca_codon_freq(handle: str, genetic_table_num: int, min_len_threshold: int = 200, n_components: int = 59,
                        organism_name: str | None = None, save_image: bool = False, folder_path: str = 'Report'):
    """
    Plots the principal component analysis based on codon frequency
    """
    records = parse(handle, 'fasta')
    references = filter_reference(records, min_len_threshold)
    len_lst = [len(gene) for gene in references]
    max_len = max(len_lst)
    s = [len(gene) / max_len * 100 for gene in references]
    contingency_table, _ = mca_codon_freq(handle, genetic_table_num, min_len_threshold, n_components)
    pca = PCA(random_state=42, n_components=n_components)
    pca.fit(contingency_table)
    plot_df = pca.row_coordinates(contingency_table)
    x = plot_df.iloc[:, 0]
    y = plot_df.iloc[:, 1]
    plt.figure(figsize=(9, 5.25))
    plt.scatter(x, y, s, alpha=0.5, c=len_lst, cmap='viridis', zorder=2)
    plt.grid(True, linestyle=':')
```

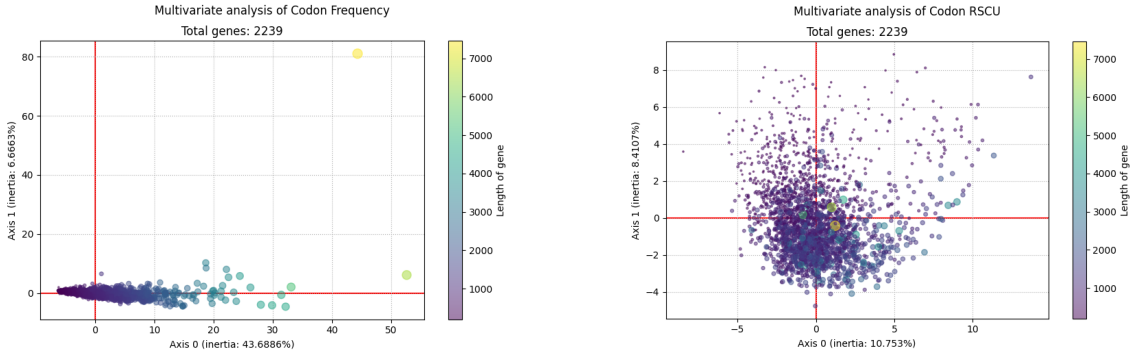
```
    plt.axvline(0, color='red', zorder=1)
    plt.axhline(0, color='red', zorder=1)
    plt.xlabel(f'Axis 0 (inertia: {round(pca.explained_inertia_[0] * 100, 4)}%)')
    plt.ylabel(f'Axis 1 (inertia: {round(pca.explained_inertia_[1] * 100, 4)}%)')
    c_bar = plt.colorbar()
    c_bar.set_label('Length of gene')
    plt.title(f'Total genes: {len(references)}')
    sup_title = f'Multivariate analysis of Codon Frequency of {organism_name}' if
    organism_name else 'Multivariate analysis of Codon Frequency'
    plt.suptitle(sup_title)
    if save_image:
        make_dir(folder_path)
        name = f'Multivariate_analysis_codon_freq_{organism_name}.png' if
    organism_name else 'Multivariate_analysis_codon_freq.png'
        file_name = join(folder_path, name)
        if is_file_writeable(file_name):
            plt.savefig(file_name, dpi=500)
    plt.show()
    plt.close()
```

Code Snippet 12.7: plot_mca_codon_freq

`plot_mca_rscu`: This is used to plot COA of codon RSCU data.

```
def plot_mca_rscu(handle: str, genetic_table_num: int, min_len_threshold: int =
    200, n_components: int = 59,
                  organism_name: str | None = None, save_image: bool = False,
    folder_path: str = 'Report'):
    """
    Plots the principal component analysis based on codon RSCU value
    """
    records = parse(handle, 'fasta')
    references = filter_reference(records, min_len_threshold)
    len_lst = [len(gene) for gene in references]
    max_len = max(len_lst)
    s = [gene_len / max_len * 100 for gene_len in len_lst]
    contingency_table, _ = mca_codon_rscu(handle, genetic_table_num,
    min_len_threshold, n_components)
    pca = PCA(random_state=42, n_components=n_components)
    pca.fit(contingency_table)
    plot_df = pca.row_coordinates(contingency_table)
    x = plot_df.iloc[:, 0]
    y = plot_df.iloc[:, 1]
    plt.figure(figsize=(9, 5.25))
    plt.scatter(x, y, s, alpha=0.5, c=len_lst, cmap='viridis', zorder=2)
    plt.grid(True, linestyle=':')
    plt.axvline(0, color='red', zorder=1)
    plt.axhline(0, color='red', zorder=1)
    plt.xlabel(f'Axis 0 (inertia: {round(pca.explained_inertia_[0] * 100, 4)}%)')
    plt.ylabel(f'Axis 1 (inertia: {round(pca.explained_inertia_[1] * 100, 4)}%)')
    c_bar = plt.colorbar()
    c_bar.set_label('Length of gene')
    plt.title(f'Total genes: {len(references)}')
    sup_title = f'Multivariate analysis of Codon RSCU of {organism_name}' if
    organism_name else 'Multivariate analysis of Codon RSCU'
    plt.suptitle(sup_title)
    if save_image:
        make_dir(folder_path)
        name = f'Multivariate_analysis_rscu_{organism_name}.png' if organism_name
    else 'Multivariate_analysis_rscu.png'
        file_name = join(folder_path, name)
        if is_file_writeable(file_name):
            plt.savefig(file_name, dpi=500)
    plt.show()
    plt.close()
```

Code Snippet 12.8: plot_mca_rscu



(a) COA Plot of Codon Freq

(b) COA Plot of Codon RSCU

Figure 12.4: COA Plots of Codon

`plot_mca_aa_gravy`: This is used to plot COA of AA GRAVY data.

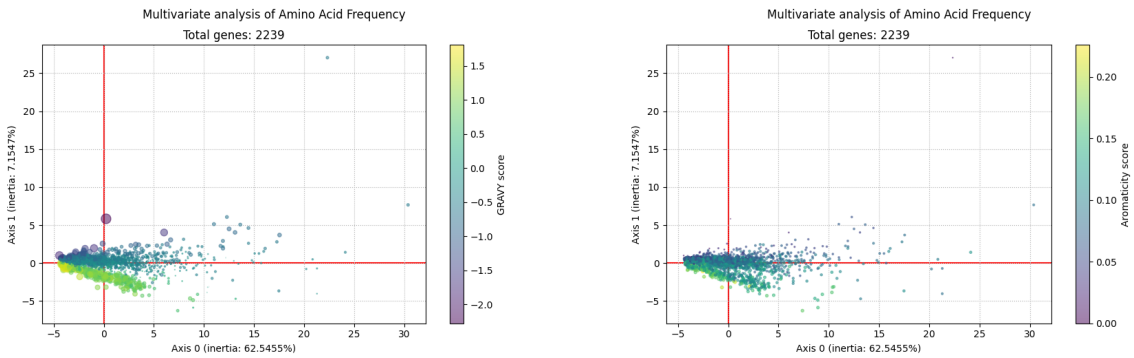
```
def plot_mca_aa_gravy(handle: str, genetic_table_num: int, min_len_threshold: int = 66, n_components: int = 20,
                      organism_name: str | None = None, save_image: bool = False, folder_path: str = 'Report'):
    """
    Plots the principal component analysis based on amino acid frequency with GRAVY score scale
    """
    records = parse(handle, 'fasta')
    references = filter_reference(records, min_len_threshold)
    gravy = [ProteinAnalysis(str(prot_seq)).gravy() for prot_seq in references]
    s = [(g ** 2) * 20 for g in gravy]
    contingency_table, _ = mca_aa_freq(handle, genetic_table_num, min_len_threshold, n_components)
    pca = PCA(random_state=42, n_components=n_components)
    pca.fit(contingency_table)
    plot_df = pca.row_coordinates(contingency_table)
    x = plot_df.iloc[:, 0]
    y = plot_df.iloc[:, 1]
    plt.figure(figsize=(9, 5.25))
    plt.scatter(x, y, s, alpha=0.5, c=gravy, cmap='viridis', zorder=2)
    plt.grid(True, linestyle=':')
    plt.axvline(0, color='red', zorder=1)
    plt.axhline(0, color='red', zorder=1)
    plt.xlabel(f'Axis 0 (inertia: {round(pca.explained_inertia_[0] * 100, 4)}%)')
    plt.ylabel(f'Axis 1 (inertia: {round(pca.explained_inertia_[1] * 100, 4)}%)')
    c_bar = plt.colorbar()
    c_bar.set_label('GRAVY score')
    plt.title(f'Total genes: {len(references)}')
    sup_title = f'Multivariate analysis of Amino Acid Frequency of {organism_name}' if organism_name else 'Multivariate analysis of Amino Acid Frequency'
    plt.suptitle(sup_title)
    if save_image:
        make_dir(folder_path)
        name = f'Multivariate_analysis_aa_gravy_{organism_name}.png' if organism_name else 'Multivariate_analysis_aa_gravy.png'
        file_name = join(folder_path, name)
        if is_file_writeable(file_name):
            plt.savefig(file_name, dpi=500)
    plt.show()
    plt.close()
```

Code Snippet 12.9: plot_mca_aa_gravy

`plot_mca_aa_aroma`: This is used to plot COA of AA Aromaticity data.

```python
def plot_mca_aa_aroma(handle: str, genetic_table_num: int, min_len_threshold: int = 66, n_components: int = 20,
                      organism_name: str | None = None, save_image: bool = False, folder_path: str = 'Report'):
    """
    Plots the principal component analysis based on amino acid frequency with aromaticity score scale
    """
    records = parse(handle, 'fasta')
    references = filter_reference(records, min_len_threshold)
    aroma = [ProteinAnalysis(str(prot_seq)).aromaticity() for prot_seq in references]
    s = [a * 50 for a in aroma]
    contingency_table, _ = mca_aa_freq(handle, genetic_table_num, min_len_threshold, n_components)
    pca = PCA(random_state=42, n_components=n_components)
    pca.fit(contingency_table)
    plot_df = pca.row_coordinates(contingency_table)
    x = plot_df.iloc[:, 0]
    y = plot_df.iloc[:, 1]
    plt.figure(figsize=(9, 5.25))
    plt.scatter(x, y, s, alpha=0.5, c=aroma, cmap='viridis', zorder=2)
    plt.grid(True, linestyle=':')
    plt.axvline(0, color='red', zorder=1)
    plt.axhline(0, color='red', zorder=1)
    plt.xlabel(f'Axis 0 (inertia: {round(pca.explained_inertia_[0] * 100, 4)}%)')
    plt.ylabel(f'Axis 1 (inertia: {round(pca.explained_inertia_[1] * 100, 4)}%)')
    c_bar = plt.colorbar()
    c_bar.set_label('Aromaticity score')
    plt.title(f'Total genes: {len(references)}')
    sup_title = f'Multivariate analysis of Amino Acid Frequency of {organism_name}' if organism_name else 'Multivariate analysis of Amino Acid Frequency'
    plt.suptitle(sup_title)
    if save_image:
        make_dir(folder_path)
        name = f'Multivariate_analysis_aa_aroma_{organism_name}.png' if organism_name else 'Multivariate_analysis_aa_aroma.png'
        file_name = join(folder_path, name)
        if is_file_writeable(file_name):
            plt.savefig(file_name, dpi=500)
    plt.show()
    plt.close()
```

Code Snippet 12.10: plot_mca_aa_aroma



(a) COA Plot of AA GRAVY

(b) COA Plot of AA Aromaticity

Figure 12.5: COA Plots of AA

Part IV

Conclusion

Chapter 13

Conclusion

In conclusion, this thesis project aimed to develop `CodonU`, a comprehensive tool for CUA. `CodonU` addresses the need for an efficient, user-friendly, and versatile software package that integrates various steps of codon usage analysis, provides statistical measures, and offers visualization capabilities. Through the implementation and evaluation of `CodonU`, several key findings and achievements have been made.

Firstly, `CodonU` successfully handles various file formats, allowing users to input organism names and accession IDs and automatically fetch gbk files, extract CDS, and save them in the required format. This streamlines the data acquisition process and eliminates the need for manual data retrieval and manipulation. Secondly, `CodonU` incorporates essential statistical measures for codon usage analysis, including CAI, CBI, etc. These measures provide valuable insights into the preferences and biases in codon usage within an organism's genome, aiding in the understanding of genetic codes and their implications.

Furthermore, `CodonU` employs COA to explore associations between codon usage patterns and factors of interest. Moreover, `CodonU` offers visualization capabilities, generating high-resolution graphics suitable for publication. These graphics enable researchers to effectively communicate and present their findings, enhancing the visibility and impact of their research in the scientific community.

Looking ahead, `CodonU` holds great potential for future implications in genomic research and evolutionary studies. Its modular architecture allows for further development, expansion, and integration of additional features and functionalities. Additionally, `CodonU` can serve as a valuable resource for researchers, enabling them to perform comprehensive codon usage analysis and gain deeper insights into the genetic codes and their evolutionary implications.

In conclusion, `CodonU` simplifies and enhances the process of codon usage analysis, providing researchers with a powerful and accessible tool. The development of `CodonU` contributes to the advancement of genomic research and offers new avenues for exploration in the field of codon usage analysis.

Part V

Bibliography

Bibliography

Articles

Abdi, Hervé and Lynne Williams (July 2010). "Principal component analysis". In: *Wiley Interdisciplinary Reviews: Computational Statistics* 2.4, pp. 433–459. DOI: 10.1002/wics.101. URL: https://doi.org/10.1002/wics.101.

Angov, E. (June 2011). "Codon usage: Nature's roadmap to expression and folding of proteins". In: *Biotechnology Journal* 6.6, pp. 650–659. DOI: 10.1002/biot.201000332. URL: https://doi.org/10.1002/biot.201000332.

Bennetzen, J L and B D Hall (Mar. 1982). "Codon selection in yeast." In: *Journal of Biological Chemistry* 257.6, pp. 3026–3031. DOI: 10.1016/s0021-9258(19)81068-2. URL: http://dx.doi.org/10.1016/s0021-9258(19)81068-2.

Carver, Tim and Alan J. Bleasby (Sept. 2003). "The design of Jemboss: a graphical user interface to EMBOSS". In: *Bioinformatics* 19.14, pp. 1837–1843. DOI: 10.1093/bioinformatics/btg251. URL: https://doi.org/10.1093/bioinformatics/btg251.

Cock, Peter J. A. et al. (June 2009). "Biopython: freely available Python tools for computational molecular biology and bioinformatics". In: *Bioinformatics* 25.11, pp. 1422–1423. DOI: 10.1093/bioinformatics/btp163. URL: https://doi.org/10.1093/bioinformatics/btp163.

Duncombe, P. (Jan. 1985). "Multivariate Descriptive Statistical Analysis: Correspondence Analysis and Related Techniques for Large Matrices." In: *Journal of the Royal Statistical Society*. DOI: 10.2307/2981514. URL: https://doi.org/10.2307/2981514.

Fuglsang, Anders (May 2004). "The 'effective number of codons' revisited". In: *Biochemical and Biophysical Research Communications* 317.3, pp. 957–964. DOI: 10.1016/j.bbrc.2004.03.138. URL: http://dx.doi.org/10.1016/j.bbrc.2004.03.138.

Harris, Charles B. et al. (Sept. 2020). "Array programming with NumPy". In: *Nature* 585.7825, pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

Hill, Martin and Hugh G. Gauch (Oct. 1980). "Detrended correspondence analysis: An improved ordination technique". In: *Vegetatio* 42.1-3, pp. 47–58. DOI: 10.1007/bf00048870. URL: https://doi.org/10.1007/bf00048870.

Hodgman, Matthew R et al. (Nov. 2020). "CUBAP: an interactive web portal for analyzing codon usage biases across populations". In: *Nucleic Acids Research* 48.19, pp. 11030–11039. DOI: 10.1093/nar/gkaa863. URL: https://doi.org/10.1093/nar/gkaa863.

Hunter, J.D. (May 2007). "Matplotlib: A 2D Graphics Environment". In: *Computing in Science and Engineering* 9.3, pp. 90–95. DOI: 10.1109/mcse.2007.55. URL: https://doi.org/10.1109/mcse.2007.55.

Kyte, Jack and Russell F. Doolittle (May 1982). "A simple method for displaying the hydropathic character of a protein". In: *Journal of Molecular Biology* 157.1, pp. 105–132. DOI: 10.1016/0022-2836(82)90515-0. URL: http://dx.doi.org/10.1016/0022-2836(82)90515-0.

Larkin, Mark A. et al. (Nov. 2007). "Clustal W and Clustal X version 2.0". In: *Bioinformatics* 23.21, pp. 2947–2948. DOI: 10.1093/bioinformatics/btm404. URL: https://doi.org/10.1093/bioinformatics/btm404.

Lee, Benjamin D. (Oct. 2018b). "Python Implementation of Codon Adaptation Index". In: *Journal of open source software* 3.30, p. 905. DOI: 10.21105/joss.00905. URL: https://joss.theoj.org/papers/10.21105/joss.00905.pdf.

Lloyd, Andrew T. and P. Sharp (June 1992). "CODONS: A Microcomputer Program for Codon Usage Analysis". In: *Journal of Heredity* 83.3, pp. 239–240. DOI: 10.1093/oxfordjournals.jhered.a111205. URL: https://doi.org/10.1093/oxfordjournals.jhered.a111205.

Lobry, Jean R. and Christian Gautier (Aug. 1994). "Hydrophobicity, expressivity and aromaticity are the major trends of amino-acid usage in 999 Escherichia coli chromosome-encoded genes". In: *Nucleic Acids Research* 22.15, pp. 3174–3180. DOI: 10.1093/nar/22.15.3174.

McInerney, J.O. (1998). "GCUA (General Codon Usage Analysis)". In: *Bioinformatics* 14.4, pp. 372–373. URL: http://mcinerneylab.com/software/gcua/.

McKinney, Wes (Jan. 2010). "Data Structures for Statistical Computing in Python". In: *Proceedings of the Python in Science Conferences*. DOI: 10.25080/majora-92bf1922-00a. URL: https://doi.org/10.25080/majora-92bf1922-00a.

Nakamura, Yasukazu, Takashi Gojobori, and Toshimichi Ikemura (Jan. 1999). "Codon usage tabulated from the international DNA sequence databases; its status 1999". In: *Nucleic Acids Research* 27.1, p. 292. DOI: 10.1093/nar/27.1.292. URL: https://doi.org/10.1093/nar/27.1.292.

— (Jan. 2000). "Codon usage tabulated from international DNA sequence databases: status for the year 2000". In: *Nucleic Acids Research* 28.1, p. 292. DOI: 10.1093/nar/28.1.292. URL: https://doi.org/10.1093/nar/28.1.292.

Nakamura, Yasukazu, Ken-nosuke Wada, et al. (Jan. 1996). "Condon usage tabulated from the international DNA sequence databases". In: *Nucleic Acids Research* 24.1, pp. 214–215. DOI: 10.1093/nar/24.1.214. URL: https://doi.org/10.1093/nar/24.1.214.

Nakamura, Yusuke, Takashi Gojobori, and Toshimichi Ikemura (Jan. 1997). "Codon usage tabulated from the international DNA sequence databases". In: *Nucleic Acids Research* 25.1, pp. 244–245. DOI: 10.1093/nar/25.1.244. URL: https://doi.org/10.1093/nar/25.1.244.

— (Jan. 1998). "Codon usage tabulated from the international DNA sequence databases". In: *Nucleic Acids Research* 26.1, p. 334. DOI: 10.1093/nar/26.1.334. URL: https://doi.org/10.1093/nar/26.1.334.

Nesti, C. et al. (Apr. 1995). "Phylogeny inferred from codon usage pattern in 31 organisms". In: *Bioinformatics* 11.2, pp. 167–171. DOI: 10.1093/bioinformatics/11.2.167. URL: https://doi.org/10.1093/bioinformatics/11.2.167.

Rice, Peter A., Ian Longden, and Alan J. Bleasby (June 2000). "EMBOSS: The European Molecular Biology Open Software Suite". In: *Trends in Genetics* 16.6, pp. 276–277. DOI: 10.1016/s0168-9525(00)02024-2. URL: https://doi.org/10.1016/s0168-9525(00)02024-2.

Sharp, Paul M. and Elizabeth Cowe (Oct. 1991). "Synonymous codon usage in Saccharomyces cerevisiae". In: *Yeast* 7.7, pp. 657–678. DOI: 10.1002/yea.320070702. URL: https://doi.org/10.1002/yea.320070702.

Sharp, Paul M. and Wen-Hsiung Li (1987). "The codon adaptation index-a measure of directional synonymous codon usage bias, and its potential applications". In: *Nucleic Acids Research* 15.3, pp. 1281–1295. DOI: 10.1093/nar/15.3.1281. URL: http://dx.doi.org/10.1093/nar/15.3.1281.

Sievers, Fabian and Desmond G. Higgins (Jan. 2018). "Clustal Omega for making accurate alignments of many protein sequences". In: *Protein Science* 27.1, pp. 135–145. DOI: 10.1002/pro.3290. URL: https://doi.org/10.1002/pro.3290.

Stothard, Paul (June 2000). "The Sequence Manipulation Suite: JavaScript Programs for Analyzing and Formatting Protein and DNA Sequences". In: *BioTechniques* 28.6, pp. 1102–1104. DOI: 10.2144/00286ir01. URL: https://doi.org/10.2144/00286ir01.

Talevich, Eric et al. (Aug. 2012). "Bio.Phylo: A unified toolkit for processing, analyzing and visualizing phylogenetic trees in Biopython". In: *BMC Bioinformatics* 13.1. DOI: 10.1186/1471-2105-13-209. URL: https://doi.org/10.1186/1471-2105-13-209.

Tamura, Koichiro, Glen Stecher, and Sudhir Kumar (Apr. 2021). "MEGA11: Molecular Evolutionary Genetics Analysis Version 11". In: *Molecular Biology and Evolution* 38.7, pp. 3022–3027. DOI: 10.1093/molbev/msab120. URL: https://academic.oup.com/mbe/article-pdf/38/7/3022/38827102/msab120.pdf.

Thioulouse, Jean (Oct. 1989). "Statistical analysis and graphical display of multivariate data on the Macintosh". In: *Bioinformatics* 5.4, pp. 287–292. DOI: 10.1093/bioinformatics/5.4.287. URL: https://doi.org/10.1093/bioinformatics/5.4.287.

— (Nov. 1990). "Macmul and graphmu: Two Macintosh programs for the display and analysis of multivariate data". In: *Computers and Geosciences* 16.8, pp. 1235–1240. DOI: 10.1016/0098-3004(90)90058-2. URL: https://doi.org/10.1016/0098-3004(90)90058-2.

Thioulouse, Jean and François Chevenet (Mar. 1996). "NetMul, a World-Wide Web user interface for multivariate analysis software". In: *Computational Statistics and Data Analysis* 21.3, pp. 369–372. DOI: 10.1016/0167-9473(96)90065-1. URL: https://doi.org/10.1016/0167-9473(96)90065-1.

Vetrivel, Umashankar, Vijayakumar Arunkumar, and Sudarsanam Dorairaj (Oct. 2007). "ACUA: A software tool for automated codon usage analysis". In: *Bioinformation* 2.2, pp. 62–63. DOI: 10.6026/97320630002062. URL: https://doi.org/10.6026/97320630002062.

Virtanen, Pauli et al. (July 2019). "SciPy 1.0: fundamental algorithms for scientific computing in Python". In: *Nature Methods* 17.3, pp. 261–272. DOI: 10.1038/s41592-019-0686-2. URL: https://doi.org/10.1038/s41592-019-0686-2.

Wada, Ken-nosuke et al. (Apr. 1990). "Codon usage tabulated from the GenBank genetic sequence data". In: *Nucleic Acids Research* 18.suppl, pp. 2367–2411. DOI: 10.1093/nar/18.suppl.2367. URL: https://doi.org/10.1093/nar/18.suppl.2367.

Wright, Frank (Mar. 1990). "The 'effective number of codons' used in a gene". In: *Gene* 87.1, pp. 23–29. DOI: 10.1016/0378-1119(90)90491-9. URL: http://dx.doi.org/10.1016/0378-1119(90)90491-9.

Books

Greenacre, Michael (Jan. 2017). *Correspondence Analysis in Practice*. DOI: 10.1201/9781315369983. URL: https://doi.org/10.1201/9781315369983.

Hameroff, Stuart R. (Aug. 1987). *Ultimate Computing: Biomolecular Consciousness and NanoTechnology*. Elsevier Science. ISBN: 978-0444600097. URL: https://shop.elsevier.com/books/ultimate-computing/hameroff/978-0-444-70283-8.

Hill, Martin (Jan. 1979). *DECORANA - A FORTRAN program for detrended correspondence analysis and reciprocal averaging*. URL: http://repositories.tdl.org/tamug-ir/handle/1969.3/24717?show=full.

Husson, François and Julie Josse (Jan. 2010). *Multiple Correspondence Analysis*. DOI: 10.4135/9781412993906. URL: https://doi.org/10.4135/9781412993906.

Jolliffe, I.T. (Oct. 2002). *Principal Component Analysis*. Springer Science and Business Media. DOI: 10.1007/b98835. URL: https://doi.org/10.1007/b98835.

Lebart, Ludovic, Alain Morineau, and Kenneth M. Warwick (Jan. 1984). *Multivariate Descriptive Statistical Analysis*. John Wiley and Sons.

Lobry, Jean R. (Nov. 2018). *Multivariate Analyses of Codon Usage Biases*. ISTE Press - Elsevier. DOI: 10.1016/C2018-0-02165-9. URL: https://doi.org/10.1016/C2018-0-02165-9.

Book Chapters

Sieburg, Hans B. (Aug. 1990). "Physiological Studies in Silico". In: *1990 Lectures In Complex Systems*. Ed. by Lynn Nadel and Daniel L. Stein. CRC Press. ISBN: 978-0429503573. URL: https://www.taylorfrancis.com/chapters/edit/10.1201/9780429503573-15/physiological-studies-silico-hans-sieburg.

Womble, David D. (1999). "GCG:" in: *Bioinformatics Methods and Protocols*. Ed. by Stephen Misener and Stephen A. Krawetz. Totowa, NJ: Humana Press, pp. 3–22. ISBN: 978-1-59259-192-3. DOI: 10.1385/1-59259-192-2:3. URL: https://doi.org/10.1385/1-59259-192-2:3.

Thesis

Peden, John F. (Aug. 1999). "Analysis of codon usage". Nottingham, United Kingdom: University of Nottingham. URL: http://codonw.sourceforge.net/JohnPedenThesisPressOpt_water.pdf.

Softwares, Online Tools, etc.

ANSI (1978). *American National Standard Fortran X3.9-1978 (FORTRAN 77)*. Programming Language. URL: https://www.ibm.com/docs/en/xl-fortran-aix/16.1.0?topic=specifications-fortran-77.

Biopython Dev Team (June 2021). *Biopython*. Python software package. URL: https://pypi.org/project/biopython.

Bleasby, Alan (Nov. 2000). *coderet*. Online Tool. European Bioinformatics Institute. URL: https://www.bioinformatics.nl/cgi-bin/emboss/coderet.

Diament, Alon (2023). *Codon-Usage*. MATLAB software package. URL: https://github.com/alondmnt/codon-usage.

Halford, Max (Oct. 2020). *Prince*. Python software package. URL: https://pypi.org/project/prince.

IBM. Corp (2009). *SPSS*. Software. New Orchard Road, Armonk, NY. URL: https://www.ibm.com/products/spss-statistics.

Lee, Benjamin D. (Oct. 2018a). *CAI*. Python software package. URL: https://pypi.org/project/CAI.

Matplotlib Dev Team (Jan. 2023). *Matplotlib*. Python software package. URL: https://pypi.org/project/matplotlib.

MEGA Dev Team (2021). *Mega 11*. Software. URL: https://www.megasoftware.net/home.

NCBI (1992). *GenBank*. Database. National Library of Medicine. URL: https://www.ncbi.nlm.nih.gov/genome/.

NumPy Dev Team (Feb. 2023). *NumPy*. Python software package. URL: https://pypi.org/project/numpy.

Oksanen, Jari (Oct. 2022). *decorana*. R software package. URL: https://www.rdocumentation.org/packages/vegan/versions/2.6-4/topics/decorana.

Pandas Dev Team (Aug. 2022). *Pandas*. Python software package. URL: https://pypi.org/project/pandas.

Pedan, John F. (2005). *CodonW*. Software. URL: https://sourceforge.net/projects/codonw/.

Python Dev Team (2001). *Pyhton: A Dynamic, Clear, Object Oriented and Open Source Language*. Programming Language. Wilmington, Delaware, United State. URL: https://www.python.org/.

Scipy Dev Team (Jan. 2023). *SciPy*. Python software package. URL: https://pypi.org/project/scipy.

Part VI

Communication

CodonU: A Python Package for Codon Usage Analysis

Souradipto Choudhuri, Keya Sau

Abstract—Codon Usage Analysis has been accompanied by several web servers and independent programs written in several programming languages. Also this diversity speaks for the need of a reusable software that can be helpful in reading, manipulating and acting as a pipeline for such data and file formats. Most popular software for these kind of analyses is CodonW. But it has its limited scopes and a complex pipeline for data analysis. So, we propose CodonU, a package written in python language. It is compatible with existing file formats and can be used solely or with a group of other such packages. The proposed package incorporates various statistical measures necessary for codon usage analysis. The measures vary with nature of the sequences, viz. for nucleotide, codon adaptation index (*CAI*), codon bias index (*CBI*) etc. and for protein sequences Gravy score etc. Users can also perform the correspondence analysis (*COA*). This package also provides the liberty to generate graphics to users, and also perform phylogenetic analysis which is out of scope for CodonW. Capabilities of the proposed package were checked thoroughly on a diverse genomic set. Detailed documentation and some examples for this open-source project is available at GitHub: https://www.github.com/SouradiptoC/CodonU

Index Terms—Codon Bias, Codon Usage, CodonW, Codon Usage Analysis, Correspondence Analysis, Phyogenetic Analysis

1 Introduction

SYNONYMOUS codons are sets of codons that code for the same amino acid. While the resulting protein remains unchanged, the preferential usage of particular codons can vary across species, genes, and even within different regions of a single gene. This phenomenon has been the subject of investigation in the field of bioinformatics for several decades. The first *insilico* analysis of codon usage was conducted by John F. Pedan in his PhD thesis [1] in the late 1990s. Pedan tackled this issue by creating a C language project called 'CodonW' [2]. However, with the rapid advancement of computational technology, the CodonW software has become complex and challenging for new users. Furthermore, its supported functionalities are limited to a certain extent.

Moreover, one couldn't directly feed the data to CodonW. It required pre-processing. A general workflow may had been consistent of following steps:

- Pre-processing
  - Finding and downloading the data in *genbank* (.gb or .gbk) file format from NCBI, or creating a .gb file if working in a customized manner.
  - Upload the downloaded file to coderet [3] for extracting *Coding Sequence* (CDS) and/or translated mRNAs (essentially proteins) in *fasta* (.fasta or .fa) file format.
- CodonW
  - Feeding the extracted CDS and/or protein sequences to CodonW. CodonW itself is a cmd or terminal based tool. And, the users need to specify which operation(s) they want to perform. At last the results can be exported in various file formats such as .coa etc which are specific to CodonW. For more on various files generated by CodonW, please visit the thesis [1].
- Statistical significance and plot generation
  - The exported values need to be checked for statistical significance. This can be done through popular softwares such as SPSS [4].
  - Graphics can not be directly generated from CodonW. The user must do with third-party software(s) (at least in this context) like Microsoft Excel.
- Phylogeny analysis
  - For phylogeny analysis, users must perform multiple sequence alignment first.
  - Once done, they can use third-party webserver (e.g. ClustalW2-Phylogeny etc.) or software (e.g. Mega [5] etc.) for tree generation.

Consequently, there is a growing need for an integrated workflow that can efficiently and accurately analyze codon usage. CodonU can perform all the aforementioned operations with minimal memory consumption. Furthermore, CodonU offers the added benefit of allowing users to perform multiple operations and save only the final results for various analyses and graphics generation.

- *S. Choudhuri, at the time of working on the project was a bachelor's student at Department of Biotechnology, Haldia Institute of Technology, Haldia, West Bengal, India, 721657*
  E-mail: souradipto.choudhuri@hithaldia.ac.in
- *K. Sau, Ph.D., at the time the project was developed was an associate professor with specialization in bioinformatics at Department of Biotechnology, Haldia Institute of Technology, Haldia, West Bengal, India, 721657*
  E-mail: keyasau@hithaldia.ac.in

(Correspondence: Both authors)