

**Toward accessible bioinformatic tools for analyzing residue coevolution and sequence-
fitness relationships in Fluc family proteins**

by

Fox Christopher Baudelaire

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
(Molecular, Cellular, and Developmental Biology)

University of Michigan
May 2023

Thesis committee:

Associate Professor Randy Stockbridge
Professor Gary Huffnagle
Professor Ursula Jakob

Fox Christopher Baudelaire

fblaire@umich.edu

ORCID ID: 0000-0002-5551-6851

© Fox Baudelaire 2023

Acknowledgments

The author first wishes to express gratitude to the members of the committee who agreed to review this thesis, including Dr. Randy Stockbridge, Dr. Ursula Jakob, and Dr. Gary Huffnagle. I am further grateful to the committee for accommodating changes to the focus of the work when my research interests came into clearer view. I am also thankful for the opportunity to work with my colleagues in the Stockbridge lab: Olive Burata and Rachael Lucero in particular stand out to me as outstanding researchers who also devote ample time in helping the lab run smoothly. I will miss their camaraderie, words of encouragement, and contributions to a sense of community in the group. I am also thankful to Carla Peralta, whose collaboration with me had helped me to overcome feelings of detachment in research that affected me in the earlier half of my time in the lab. Of course, my transition to the activities of a graduate student also greatly benefited from working with Chris Macdonald, whose own studies of the Fluc channels prefigured the present work. Additional talented members of the Stockbridge lab I wish to recognize include Chia-Yu Kang, Achala Chittor, Trevor Yeh, Ever O'Donnell, Kemal Demirer, and Michal Ruprecht. I am also thankful for Dr. Laura Olsen and Mary Carr, who were sources of counsel as I prepared my application to the department and the University of Michigan. Next, I must extend my gratitude to the undergraduates I have served in my two semesters as a Graduate Student Instructor, time which allowed me to reflect on what it was like to be a beginner and which will fortify my growth as an educator capable of making a real difference in the lives of students. I am reminded of several outstanding teachers who were a positive and lasting influence in my upbringing and especially my time at Don Estridge High Tech Middle School – the golden age of my schooling – which helped to instill me with an abiding sense of curiosity and resilience that uplifted me amid challenges that Carl Sagan may have recognized when he quipped that “every kid starts out as a natural-born scientist, and then we beat it out of them. A few trickle through the system with their wonder and enthusiasm for science intact.” I would especially like to thank my high school biology teacher, Fred Hock, who taught at Atlantic Community High School in Palm Beach County and who first dazzled me with the great ideas in the field of biochemistry. Even though hard times fell on the latter years of my adolescence, I held fast to that sense of wonder as I began a new life in Boston. To do acknowledgements full justice would require quite a few more pages, but I would still like to express thanks to the many people whom I met while I called the Boston area home, starting with coworkers from my job as a bookstore clerk at Logan International Airport, especially Lynn Sweezy, Kerry Breymann, as well as Saory Khong and sister Sina Khong. From my time as a student at Bunker Hill Community College, I am grateful to various faculty, staff, and other affiliates, notably Katy Abel, Kathleen O'Neill, Wick Sloane, Libby Dunphy, Karen Atkinson. From my subsequent time as an undergraduate at Brandeis University, I thank professors Dr. Michael Hagan and Dr. Melissa Kosinski-Collins for their continued support. Other friends from college and from Boston include Christine Spinelli and Victor Zúñiga, as well as Bill Hagar, Jessica Wei, Marissa Farkas, Alex Sun, Johnson Agyapong, Elyse Hahn, Nick Hanson-Holtry, Hairuo Sun, and Sara Hamadani. Last but not least, I would like to thank my half-brother Scott Aguirre and his family, whose presence has positively influenced my life since we re-established contact in 2019. Thank you, Scott.

Table of Contents

Acknowledgments	iv
List of Figures.....	vii
List of Abbreviations	ix
Abstract.....	xi
1 Introduction.....	1
1.1 Oligomericity and membrane proteins in structural biology	1
1.2 Weak acid accumulation effect and toxicity of fluoride	2
1.3 Introduction to the structure and function of Fluc family proteins.....	3
1.4 Gene duplication is the putative origin of distinct topologies in Fluc family proteins	6
1.5 Residue coevolution and its implications for structural biology and for Fluc proteins.....	10
1.6 Fitness landscapes model evolutionary fitness as a function of genotype or phenotype.....	11
1.7 Overview of thesis aims	12
2 A program for the evaluation of charge bias in Fluc loop regions	13
2.1 Background: Influence of charge bias on membrane protein orientation	13
2.2 Applicable principles of software development	14
2.3 Methods.....	15
2.3.1 Preparation of a multi-sequence alignment file	15
2.3.2 View of script and accompanying inline documentation using Jovian and Jupyter notebook integration	
15	
2.4 Structure and synopsis of program.....	16
2.5 Discussion of expected output.....	23
2.5.1 Reflection on the development and optimization of the program	23
2.5.2 Utility of program for understanding charge bias characteristics	25
3 A program for the quantification and visualization of mutational tolerance	26
3.1 Background: The single-site substitution matrix as a depiction of mutational tolerance	26
3.2 EVcouplings is a model computational framework for evolutionary couplings analysis	27
3.3 Methods.....	30
3.3.1 Generating Fluc variant information and accompanying CSV file with EVmutation	31
3.3.2 Use of R, RStudio, and Markdown to write and document an early version of the program	34

3.3.3	Interpretation of population variances among Fluc variant frequencies as mutational tolerance scores	34
3.4	Structure and synopsis of program with example output	35
3.5	Discussion.....	40
3.5.1	Reflection on the development and optimization of the program	40
3.5.2	Analysis of mutational tolerance in Fluc transmembrane domains and internal loops	40
3.5.3	Placing computed mutational tolerances in the context of Fluc topology evolution	43
3.5.4	Toward a re-implementation of program in Python	44
3.5.5	Suggestions for further development.....	45
4	Future directions in the investigation of Fluc variant fitness.....	46
4.1	On the advantages of modeling epistatic interactions	46
4.2	On challenges encountered when applying EVcouplings to Fluc variants other than Fluc-Bpe	47
4.3	Toward a deep mutational scanning methodology for investigating Fluc variant fitness	49
4.4	Prospects for resolving the Fluc evolutionary trajectory in greater detail.....	50
	Appendix.....	55
	Bibliography	56

List of Figures

EQUATION 1: Proportional relationship of $[H^+]$ and $[F^-]$ across a membrane.....	3
FIGURE 1: Structural aspects of Fluc-Ec2.....	4
FIGURE 2: Divergence subsequent to gene duplication.....	7
FIGURE 3: Paralog interference and its effects.....	8
FIGURE 4: Distinct topologies of Fluc proteins.....	9
FIGURE 5: Depictions of a fitness landscape	12
FIGURE 6: Screenshot of charge bias script in Jovian	16
FIGURE 7: Fluc homolog dataset as a CSV file in Excel	17
FIGURE 8: General workflow of a DMS experiment.....	27
FIGURE 9: Stages of the EVcouplings computational pipeline.....	28
FIGURE 10: Sequence conservation analysis of Fluc-Bpe	29
FIGURE 11: Single-site substitution matrix of Fluc-Bpe	30
FIGURE 12: EVcouplings alignment and homology search options.....	31
FIGURE 13: Screenshot of result from EVcouplings query of Fluc-Bpe.....	33
EQUATION 2: Definition of population variance for a given position.....	34
FIGURE 14: Histogram of frequency distribution of $\{\sigma^2\}$ in Fluc-Bpe residues 9-123.....	37
FIGURE 15: View of text file containing mutational tolerance scores.....	39
FIGURE 16: Histograms for Fluc-Bpe domains of interest	41
TABLE 1: Summary of mutational tolerance spreads for Fluc-Bpe domains of interest	42
FIGURE 17: Sequence alignment of selected FEX sequences with Fluc-Bpe.....	43
FIGURE 18: Screenshot of result from EVcouplings query of <i>A. thaliana</i> FEX.....	47
FIGURE 19: Screenshot of result from EVcouplings query of <i>S. cerevisiae</i> FEX.....	48
FIGURE 20: Schematic of small-scale growth assay to test relative fitness with <i>Ec2</i>	49
FIGURE 21: Influence of genotype and environment on phenotype and fitness in <i>lac</i> pathway.....	52

FIGURE 22: Different levels of perspective in a fitness landscape..... 53
FIGURE A1: Flowchart of script to evaluate charge bias 55
FIGURE A2: Outline of script to compute mutational tolerances..... 55

List of Abbreviations

Abbreviation	Definition
ASPEN	Accuracy through Subsampling of Protein Evolution
CADD	Combined Annotation–Dependent Depletion; bioinformatics program
CLC	Chloride channel
CRAN	Comprehensive R Archive Network
CSV	Comma-separated values; file format
DCA	Direct coupling analysis
DFE	Distribution of fitness effects
DMS	Deep Mutational Scanning
FASTA	"FAST-All"; file format for protein and nucleotide sequences
FEX	Fluoride exporter
GEBA	Genomic Encyclopedia of Bacteria and Archaea
GUI	Graphical user interface
HMM	Hidden Markov model
HMMER	software for protein sequence homology analysis; uses HMMs
HTML	Hypertext Markup Language
IPTG	Isopropyl β -D-1-thiogalactopyranoside
MAFFT	Multiple Alignment using Fast Fourier Transform; sequence alignment tool
MAVE	Multiplex assays of variant effect
MRF	Markov random field
MSA	Multi-sequence alignment
MUSCLE	Multiple Sequence Comparison by Log-Expectation; alignment software
PDB	Protein Data Bank
PYMOL	Python-based molecular graphics program; used for protein structures
SIFT	Sorting Intolerant From Tolerant; bioinformatics program for mutagenesis
TEM-1	Temoniera in Greece; gene or TEM-1 β -lactamase
TM	Transmembrane
TMD	Transmembrane domain
URL	Uniform Resource Locator
WT	Wild-type
ZIP	archive file format with extension .zip

It is important that students bring a certain ragamuffin, barefoot irreverence to their studies; they are not here to worship what is known, but to question it.

Jacob Bronowski

Justice is what love looks like in public.

Cornel West

Abstract

Members of the Fluc family of membrane channel proteins, found in all three domains of biological classification, allow organisms to prevent the buildup of fluoride ion inside cells and thereby counteract fluoride toxicity. Export through these proteins is passive yet extremely selective for the substrate. Like nearly all membrane proteins, Fluc protomers exhibit internal repeat symmetry, which is thought to result from either the association of structurally similar domains or gene duplication. Gene duplication events are of special significance to the Fluc family because duplication is considered the earliest mechanism of major genetic variation that allowed for topologically distinct Fluc channels to evolve. In prokaryotes, the Fluc channels assemble as proper dimers and may have a dual topology (monomer is equally capable of insertion in either of two orientations with respect to the membrane) or a fixed topology (monomer is biased towards one orientation). Fixed-topology channels are obligate Fluc heterodimers thought to have arisen due to sequence divergence in the two copies of Fluc-encoding gene following a duplication event. In eukaryotes, the homologous FEX family proteins are monomeric, but retain a pseudosymmetry suggestive of a gene fusion event in an ancestor, joining two previously separate copies of Fluc-encoding gene. Accordingly, the dual-topology state is considered the ancestral phenotype; the more subfunctionalized phenotypes evolved later. Multiple independent gene duplications in the Fluc family have been retained over evolutionary time, and the Fluc/FEX proteins as a whole are thought to have undergone a general, three-step evolutionary trajectory: (1) gene duplication, (2) sequence divergence, and (3) gene fusion. With attention to ease of access and other principles of software development, the present work develops computational tools using Python and R for two purposes: evaluating the bias in membrane protein orientation for each member of a diverse set of prokaryotic Fluc homologs and quantifying mutational tolerance in Fluc residues from computed statistical correlations between likely pairs of co-evolving residues. For the latter of these ends, we apply the validated computational framework EVcouplings to Fluc-Bpe, the dual-topology Fluc channel found in *Bordetella pertussis* whose structure and function have become well characterized in recent years. We generate a single-site substitution matrix that illustrates the effects of amino acid substitution for nearly the entirety of the Fluc-Bpe primary sequence and we describe means by which this result may be validated by experiment. Having estimated *prima facie* the mutational lability of the Fluc-Bpe sequence space, we consider paralog dynamics to posit a more detailed yet provisional narrative of the evolution of post-duplication Fluc topologies. We stress the importance of further studies in the elaboration and testing of this narrative, including efforts to chart the Fluc evolutionary pathway in phylogenetic terms. Finally, we draw attention to particular limitations of the adaptationist paradigm in evolutionary theory and we advise nuance in the modeling of sequence-fitness relationships using the fitness landscape metaphor that is prominently used throughout the discipline.

1 Introduction

Themes that may be considered important for a description of the relationship between phenotype and fitness include an organism's environment, the genotype corresponding to a certain phenotype, and the interplay between these factors. In a lab setting that controls for many selective pressures of microbial life, features of greater relevance in the investigation of microbial fitness are those operating at the molecular scale. It is, for example, easy to control the temperature and medium in which a microorganism is cultivated, but a more challenging task to exert similar control over the expression of its genes. Some cases of auxotrophy offer simple examples as to whether a microorganism is entirely dependent on some aspect of its environment in order to grow or survive: a microbe that normally depends on the *trpE* gene to synthesize tryptophan but is deficient in functional *trpE* will resort completely to the uptake of tryptophan from its environment to survive. However, even in this scenario, fitness is not an all-or-nothing outcome. *E. coli* in turn depends on three different permeases (a class of membrane proteins) to accomplish the active transport of tryptophan, with each permease gene residing in its own operon [1]. Beyond that, the intracellular level of tryptophan in *E. coli* is affected by tryptophanase, an enzyme that degrades tryptophan to indole [2]. In this work, we consider a family of membrane channel proteins responsible for the export of fluoride ion, the Fluc family of proteins, which has representatives in all three domains of biological classification. This chapter will introduce key details about the structure and function of Fluc family proteins, as well as the genetic contexts that give rise to them. We necessarily introduce questions about the evolutionary history of Fluc family proteins (hereafter "Fluc proteins" or "Flucs"). More purposefully, we regard the Flucs as a model system for understanding the evolution of certain membrane protein topologies – roughly, the number and orientation of a protein's transmembrane domains – and the extent to which mutation can be tolerated in Flucs. To understand these properties, this thesis is guided by a larger inquiry into how computational and quantitative methods can shed light on them.

1.1 Oligomericity and membrane proteins in structural biology

For the purposes of this work, the terms "oligomeric protein" and "multidomain protein" are used synonymously. Although the exact definition of "oligomeric" may vary according to the specific isoforms and protein variants considered, it is estimated that 30-50% of all proteins oligomerize [3], while genomic analyses have estimated that more than 70% of eukaryotic proteins are multidomain [4]. However, a 2019 analysis reports that 65.3% of solved structures in the Protein Data Bank (PDB) are of single-domain proteins [5], also noting that many computational approaches are optimized for the structural prediction of single-domain proteins. The study and characterization of membrane proteins have historically been a challenge in structural biology owing to the hydrophobic conditions of the membrane which render its

environment very different from that of the cell's interior and exterior. Such conditions also make membrane proteins difficult to crystallize, although it remains an open question as to whether the overrepresentation of single-domain proteins in the PDB is entirely due to missing information from proteins not amenable to current experimental methods in structural biology. In early 2023, the number of experimentally validated structures in the PDB surpassed 200,000.

Except for plasma membranes and the membranes of intracellular bodies, cell membranes consist primarily of glycerophospholipids. Due to their geometry, the arrangement of such lipids in a bilayer is energetically favorable in aqueous conditions. Typically, more than half of the mass of most cellular membranes comes from membrane proteins [6]. Compared to other proteins, the structure of inner membrane proteins may appear simple; they are comprised of alpha-helices threaded successively across the membrane and arranged in bundles [7]. This apparent simplicity is belied by the cellular processes responsible for expressing a membrane protein, as well as the asymmetry originating from the protein's topology, the specific arrangement of its primary and secondary structure with respect to the membrane. This topology in turn affects the protein's own interactions with the membrane, small molecules, and other proteins.

Because the structural determination of membrane proteins has been a challenge even for modern experimental methods, it is difficult to provide a precise estimate of what fraction of membrane proteins are oligomeric. Still, oligomericity is of note to membrane proteins because of the apparent ease with which membrane proteins may diffuse along the lipid bilayer compared to the volume of a cell's interior; all else being constant, oligomeric interactions may be more likely to evolve between membrane-associated proteins than between soluble proteins in general. Phenomena such as cooperativity and allostery emerge from oligomerization. A 2005 analysis on a subset of the PDB suggests that about 65% of membrane proteins consist of more than one membrane-spanning subunit [8]. Aside from the consequences of membrane protein structure on function – which may implicate such diverse processes as transport, signaling, and energy production – investigations of this structure also can offer insights into the evolutionary history of membrane protein families, with promise for a more complete description of the path taken by the Flucs toward the provenance of extant family representatives.

1.2 Weak acid accumulation effect and toxicity of fluoride

Although concentrations of fluoride ion (F^-) in natural sources can vary widely according to factors such as weathering and deposition in soils, as well as the surrounding geology and climate of areas with natural bodies of water, such concentrations have generally increased due to human activity [9]. In developing countries as well as industrialized ones such as the United States, there are water supplies where the

concentration of fluoride is high enough to cause fluorosis, greater than 1.5 mg/L [9]. In contrast, the U.S. Public Health Service recommends a concentration of 0.7 mg/L for the prevention of dental caries [10].

Unlike other hydrohalic acids, HF is a weak acid; its short H-F bond is difficult to dissociate. With a pK_a of 3.4, HF can diffuse easily across cell membranes [11]. For a cell surrounded by conditions that are acidic relative to the cytoplasm, HF can dissociate more readily in the interior of the cell than in the cell's surroundings, exemplifying the weak acid accumulation effect [11]. Since F^- cannot diffuse back through the membrane, this effect traps F^- within the cell and contributes to a decrease in cytoplasmic pH. Given a fixed total concentration of HF (protonated and deprotonated) across a membrane, we can define a proportional relationship to describe this process (Equation 1). For acidic surroundings with pH 5.5 and a microbe maintaining a cytoplasmic pH of 7.5, the concentration of fluoride inside the cell is 100 times greater than in the environment [11].

$$\frac{[F^-]_{in}}{[F^-]_{out}} = \frac{[H^+]_{out}}{[H^+]_{in}}$$

Equation 1. Proportional relationship of H^+ and F^- concentrations across a cell membrane.

Once inside a cell, F^- can interfere with all kinds of processes: it can readily associate with metals and interfere with metalloenzyme processes, and this is considered the primary means of fluoride toxicity [12]. Indeed, fluoride exhibits a high affinity for Mg^{+2} ions, thus having the effect of inhibiting enzymes that carry out phosphoryl transfer [13]. Direct contact with HF or inhalation of HF also can lead to such toxic effects as tissue damage, bone demineralization, and respiratory problems [14].

The molecular mechanisms by which microorganisms counteract fluoride toxicity were not well understood until Baker et al. showed that the absence of the *crcB* RNA motif in *E. coli* led to an increase in F^- sensitivity for tested Δ *crcB* strains; the lower sensitivity observed in strains retaining functional *crcB* was observed due to the binding of this motif to F^- and the consequently activated expression of *crcB* genes, leading to the correct inference that *crcB* proteins acted as fluoride transporters [15].

1.3 Introduction to the structure and function of Fluc family proteins

In 2015, Stockbridge et al. published the first crystallographic structures for bacterial fluoride transporters encoded by *crcB*, revealing a “double-barreled” architecture housing two tunnels capable of fluoride conductance [16]. Such a channel can also be encoded by a *crcB* variant known as *Ec2*, occurring in *E. coli* virulence plasmid. The associated family of channel proteins, now known as the Flucs, represents the bacterial isoforms of the protein. As orthologs of *crcB* have been known to occur among bacteria, archaea, and eukaryotes, the eukaryotic isoforms of the transporter have become known as the FEX (fluoride

exporter) protein family [17]. A Fluc channel is dimeric and the aforementioned double-barreled structure stems from the assembly of two polypeptides, arranged in an antiparallel orientation, each bearing four transmembrane (TM) alpha-helices [18]. The TM regions correspond to 80-85% of the primary sequence [18]. Crystallization of the Fluc encoded by *Ec2* (Fluc-Ec2) was assisted by monobodies and achieved by R.B. Stockbridge and colleagues under C. Miller; the structure was deposited in the PDB with the identifier 5A43. In the same group, Last et al. demonstrated that two phenylalanine residues in each Fluc monomer are needed for fluoride binding and permeation [18]. Two exposed pores constitute the two tunnels of the dimeric channel, as illustrated in Figure 1 below.

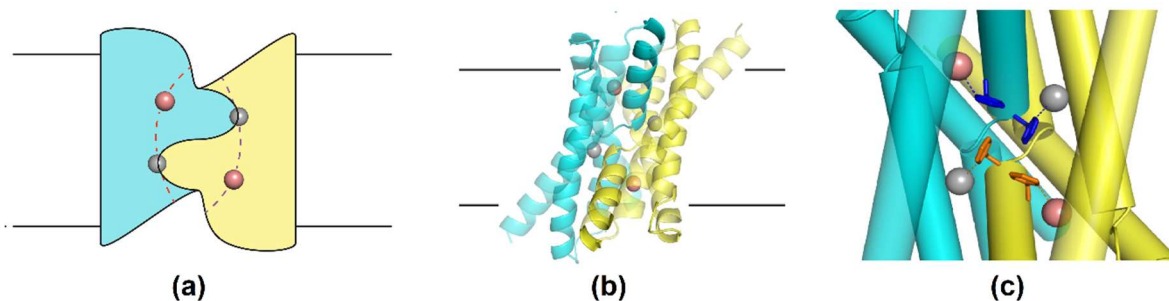


Figure 1. Structural aspects of Fluc-Ec2. Modified from Last et al. (2016) [18].

Figure 1b depicts the ribbon diagram for Fluc-Ec2 and its two constituent protomers in different colors (yellow and cyan). Figure 1c is a closeup of the interior of the protein, with phenylalanine side chains at positions 80 and 83, emphasized as the stick representation. In all three panels of Fig. 1, the spheres colored grey and pink represent F^- as part of the structure. The two bound fluoride ions within the same channel are separated by a distance of 12 Å [19]. Within a single monomer, positions 80 and 83 each contribute to a different pore in the complete channel. The Last et al. study also showed that for wild-type (WT) Fluc-Ec2, it was possible to abolish fluoride export in both channels through either of the mutations F80I or F83I, whereas for concatemeric versions of Fluc-Ec2 consisting of one WT monomer linked to one mutated monomer (F80I or F83I), fluoride conductance could be selectively ablated in either pore, without blocking conduction in the other [18]. The motif forming the fluoride coordination sphere, of which F80 and F83 are an essential part, has been dubbed the “phenylalanine box” [20]; the phenylalanine box and several surrounding residues constitute a non-helical segment that disrupts the third TM helix and forms part of the dimer interface [15]. Each pore is formed by side chains that are contributed by each Fluc monomer. From the observed spatial arrangement of fluoride ion within Fluc, it has been proposed that F^- coordinates to electropositive edges on the ring systems of each Phe side chain in what are known as anion-quadrupole interactions [21].

Recordings of current in single FEX channels in *S. cerevisiae* have been used to estimate the rate of fluoride throughput at 106 ions/s, comparable to the estimate for heteromeric Fluc in *Lactobacillus*

acidophilus from bulk liposome flux assays (over 10⁵ ions/s) [22]. Also, Fluc proteins exhibit a remarkable degree of selectivity for fluoride (more than 100-fold over chloride ion) [22]. The dehydration of F⁻ within the channel pore is important for selectivity and especially the discrimination of F⁻ over Cl⁻. An investigation employing liposome flux assays and electrophysiology experiments as well as X-ray crystallography showed that a series of residues (most of them hydrogen-bond donors) line each pore in the channel, belong to a part of the fourth TM helix collectively named a “polar track,” and are key actors in this desolvation [23]. The means by which Fluc overcomes the enthalpic cost of displacing the F⁻ solvation shell (about 125 kcal/mol, among the highest for any species of ±1 formal charge) [24], along with other details of the energetics for the fluoride export mechanism, have not yet been elucidated. Still, it is most likely that fluoride export through a Fluc protein (and its eukaryotic counterpart, FEX protein) proceeds via electrodiffusion, diffusion that is biased by an electric field. That fluoride export is a thermodynamically passive process is not surprising given the weak acid accumulation effect and the fact that the export of an anionic species such as F⁻ is favored by the negative-inside membrane potential maintained by most cells.

Analysis of electron densities occurring in the crystal structure of the *Bordetella pertussis* Fluc (Fluc-Bpe, PDB: 5NKQ) has found the presence of a sodium ion (which for *in vitro* purposes, is likely sourced from sodium compounds in buffers and crystallization conditions) that helps ensure the conformation of Fluc optimal for F⁻ conductance [25]. The Na⁺ inhere within a tetrahedral, tetradentate complex at a juncture of two TM helices, contributed from each monomer’s third TM helix; each of these two helices bears Gly77 and Thr80 which coordinate to Na⁺ via their respective backbone carbonyl oxygens [20].

In an investigation of a Fluc-Bpe single mutant (N43S) that exhibits weaker Na⁺ binding, Ernst. et al. found this mutant’s dimerization to be very thermodynamically favorable, calculating a ΔG° of -10.3 ± 0.4 kcal/mol [26].

Another group of proteins that allow bacteria to contend with fluoride stress is a fluoride-specific subclass of CLC (chloride channel) proteins which couple F⁻ export to H⁺ import [27]. Their discovery precedes that of the Flucs.

In the sections to follow, we will draw attention to two concepts pertinent to a better understanding of how Fluc structure and function evolved: residue coevolution and sequence conservation. Since a functional Fluc must be assembled as a dimer (Fluc monomer in isolation does not support F⁻ transport), we also will discuss matters of Fluc assembly and especially the influence of charge bias in the orientation assumed by Fluc with respect to the membrane. Apropos, we will examine two types of

mutation and their consequences in the Fluc evolutionary trajectory: copy-number mutation (specifically, tandem gene duplication) and point mutation.

1.4 Gene duplication is the putative origin of distinct topologies in Fluc family proteins

Gene duplication is well appreciated as a source of genetic novelty distinct from that produced by point mutations. Yet, one might think of events which cause genetic variation at the molecular level as existing in a spectrum ranging from single-nucleotide polymorphisms to short tandem repeats, then to tandem gene duplication, and finally to whole-genome duplication. Whole-genome duplication events by definition result in the duplication of all genomic segments. However, we will concentrate on tandem gene duplication, an example of small-scale duplication that may be caused by, for example, retrotransposition or unequal crossing over [28]. Gene duplication is of particular interest to this work because the duplication of a gene for an oligomeric protein entails notable structural and functional consequences for the protein, the Flucs being no exception. There moreover exists a variety of theoretical accounts aiming to describe the processes which lead to the retention of gene duplicates. For the purposes of this work, we will focus on a model of divergence that postulates three general types of outcomes, with the understanding that these fates are not necessarily mutually exclusive (in fact, they usually are not). Following the discussion put forth by Kuzmin et al. and Figure 2, these outcomes are (1) neofunctionalization, (2) subfunctionalization, and (3) dosage amplification or back-up compensation [29]. Consider an arbitrary duplication event which produces a new copy of some ancestral gene, made to exist in series with respect to the original gene. Following this tandem duplication event, the two copies of the gene are considered paralogs. Immediately after duplication, the paralogs are identical in sequence. In neofunctionalization, one of the paralogs accumulates mutations in such a way that it gains a new biological function not afforded by the ancestral gene. Generally, for neofunctionalization to be sustained, any deleterious consequences of these mutations must be outweighed by the advantage made possible by this fate. In subfunctionalization, both paralogs undergo selection such that the original set of functions previously achieved by the ancestral gene can now only be conserved when both paralogs are retained. By contrast, in a scenario that strictly involves dosage amplification and/or back-up compensation, effects resulting from the increase in gene dosage are offset by the cell (retention of the duplicate is tolerated) and/or there exists some adaptive benefit of having a duplicate which could give the organism robustness against loss of function [29].

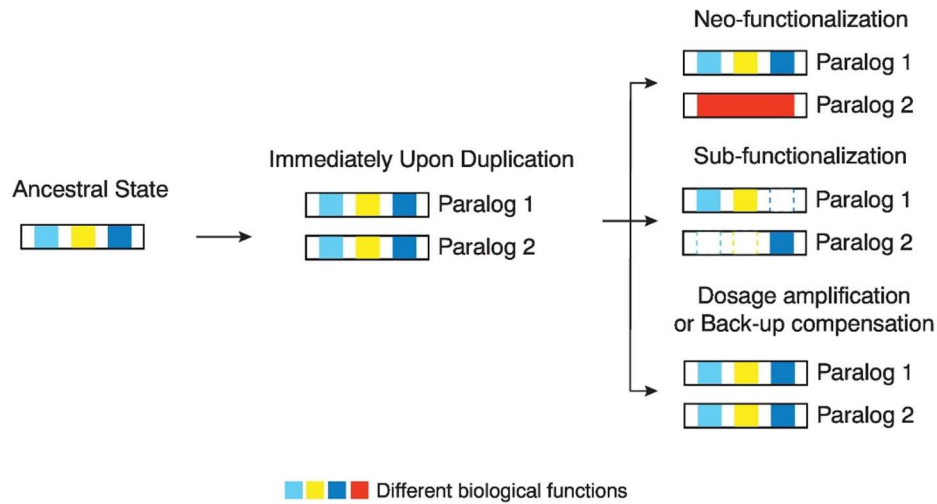


Figure 2. Divergence subsequent to gene duplication. Modified from Kuzmin et al. (2022) [29].

In cases where gene duplication does lead to a net fitness advantage, the extent to which duplication can relieve the cell from selective pressures can hardly be overstated. Prior to the duplication event, fitness for the ancestral state could have been rather dependent on whatever sequence features were important for normal protein function; mutations which obviate these features would not be tolerated by natural selection. After gene duplication, fitness becomes less dependent on these features due to, potentially, the robustness afforded by a redundant copy; mutation of such features on one paralog is more likely to be tolerated by natural selection. In turn, purifying selection may render a mutated paralog nonfunctional (e.g., the gene becomes a pseudogene and is eventually deleted) or may allow the paralog to be retained and functional.

We now turn to the effects of gene duplication on oligomeric proteins. Consider an ancestral, homodimeric protein encoded by a single copy of a gene. After a duplication event, it then becomes possible for a *paralogous heteromer* to assemble [30]. At first, the heterodimer is indistinguishable from either of the two possible homodimers that can form from either of the two paralogs alone. Should the duplication be tolerated, the two paralogs can undergo diversifying mutations, leading to distinct gene products (in this scenario, two distinct forms of the protein). However, the new relationship between the paralogs, via the oligomeric interactions within the paralogous heteromer, is a bidirectional one. As Figure 3 illustrates, an entirely *negative* effect of sequence divergence could make the paralogous heteromer distinguishable from the two possible heteromers, with negligible differences in fitness among the three possible dimers, but an entirely *positive* effect must amount to a coevolution of the two protomers, with a pronounced fitness advantage in the heteromer over the two homomers [30]. These effects and their intermediate cases may be precipitated by a dominant-negative effect in which function-damaging mutations in one paralog effectively “poison” complete oligomers [31]. In our hypothetical scenario, this would diminish function in two out of

the three possible dimer assemblies. Selective pressure against these deleterious mutations can enforce the aforementioned negative effect of sequence divergence (Figure 3a1) while the positive effect of sequence divergence leading to coevolution (Figure 3a2) could be achieved through a continuous selective pressure favoring the subfunctionalized heteromer over an evolutionarily significant period of time [30]. The foregoing phenomena are all examples of paralog (or paralogue) interference.

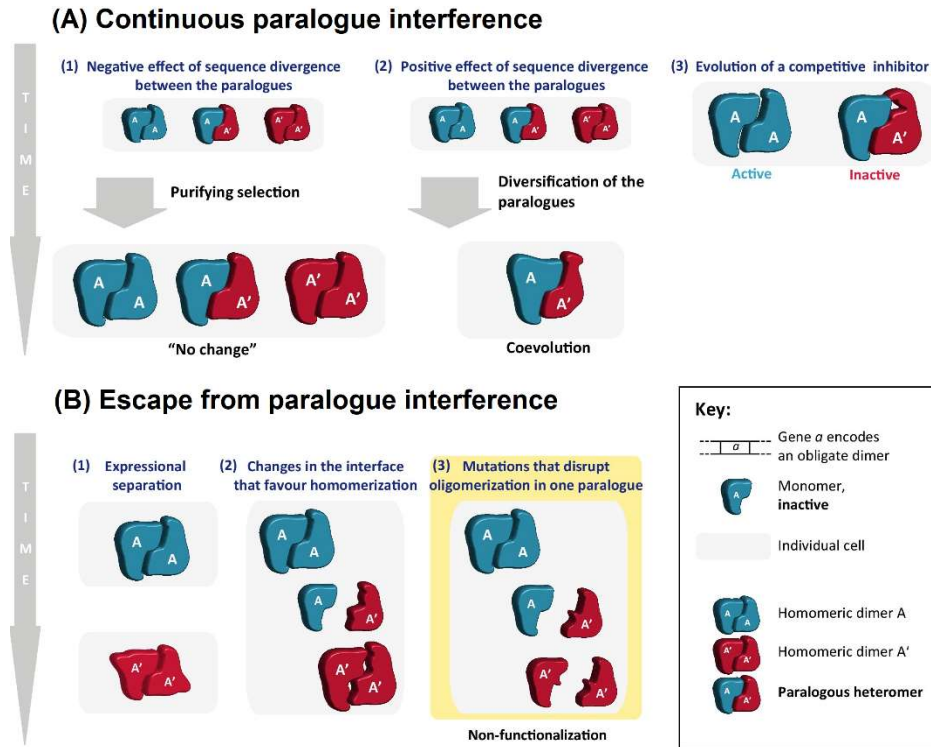


Figure 3. Paralogue interference and its effects. Modified from E. Kaltenecker and D. Ober (2015) [30].

This monograph is concerned primarily with the effects of mutations in Fluc proteins which enabled the subfunctionalization evidenced in the fixed-topology Flucs; however, it is relevant to note other outcomes that would result from paralog interference, as Figure 2 and Figure 3 illustrate. Briefly stated, paralog interference can be discontinued at which point the cross-interaction between distinct monomers is no longer feasible. For example, mutations at the dimeric interface could lead to a situation where only homodimers are able to assemble (Figure 3b2). In yet another kind of dominant-negative effect, interface-damaging mutations in one paralog could disrupt all oligomerization for said paralog (Figure 3b3).

For several reasons, it has been suggested that the topologies observed in extant Fluc (and FEX) proteins followed an evolutionary trajectory which began with gene duplication and led to subfunctionalization [7]. Like almost all membrane proteins comprised of more than one transmembrane helix, Flucs exhibit an “internal repeat” structure in which two domains of the same polypeptide are similar in fold. The specific type of repeat seen in the Flucs is an inverted repeat architecture; there are domains which not only share a tertiary structure, but are oriented in opposite ways with respect to a twofold axis parallel to the membrane [32]. This architecture exists in all three types of topologies observed in extant Fluc and FEX proteins: (1) dual topology, (2) fixed topology, and (3) inverted monomeric repeat. These are illustrated in Figure 4.

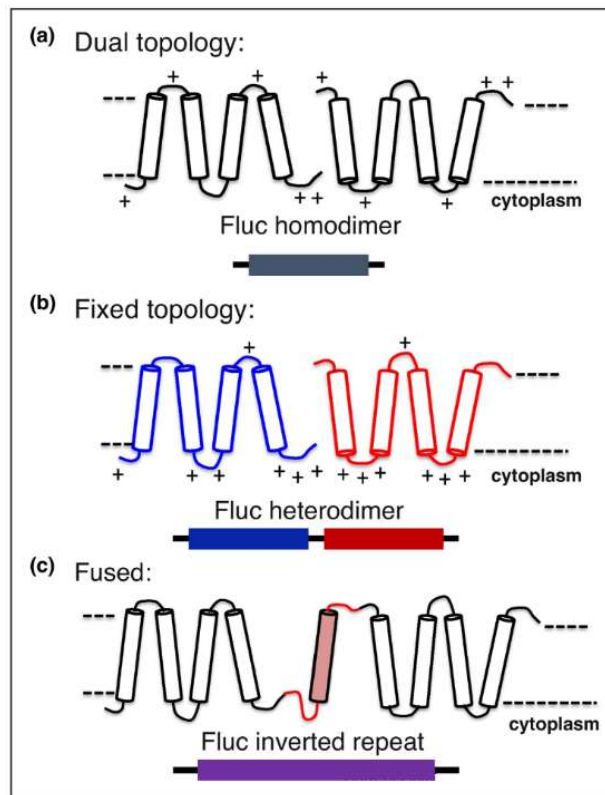


Figure 4. Distinct topologies of Fluc proteins. Modified from Macdonald and Stockbridge (2017) [20].

Also represented in Figure 4 are the three distinct genotypes known to correspond to these topologies. The inverted monomeric repeat, which is only found in eukaryotes and includes an additional TM helix, is thought to have resulted from gene fusion [7]; sequencing of FEX in *S. cerevisiae* has shown that the entire channel is encoded by a single open reading frame. In the fixed-topology phenotype, there are two distinct copies of Fluc gene in series; one encodes Fluc monomer capable of inserting itself in the N_{IN}/C_{IN} orientation (called the “in” orientation for short) whereas the other gene encodes Fluc that adopts the N_{OUT}/C_{OUT} orientation (abbreviated “out”). Fixed-topology Flucs are obligate heterodimers; functional dimeric Fluc can only be assembled as a heteromer. The fixed topology phenotype may have arisen due to

subfunctionalization. It is conceivable that a subfunctionalization process which gave rise to the fixed topology involved the accumulation of complementary mutations in two paralogs of Fluc-encoding gene. In contrast, the dual-topology state is considered the ancestral phenotype because it is associated with a single gene encoding a Fluc monomer with no preference in orientation. Dual-topology Fluc is equally capable of inserting itself in either the “in” or the “out” orientation. The difference in feasible orientations between fixed-topology Flucs and dual-topology Flucs can be explained by the tendency for loops with excess positive charge to end up facing the cytoplasm, a characteristic discussed further in Section 2.1.

1.5 Residue coevolution and its implications for structural biology and for Fluc proteins

Our discussion has reviewed in some detail the concept of a functional link ensuing from the duplication of a gene encoding oligomeric protein: the mutual influence between a pair of paralogs due to the interactions of their corresponding protomers. We also implied that a causative mechanism of subfunctionalization, leading to a fixed topology, is the coevolution of Fluc protomers enabling the optimization of protein-protein interactions at the dimer interface or the overall function of the protein.

A first step toward understanding the consequences of sequence divergence in post-duplication Flucs might consider sequence conservation. It is not controversial to say that the total absence of sequence divergence has neutral consequences for fitness, but the effect of any particular sequence of mutations on fitness is not trivial to evaluate. The total number of permutations for a 100-residue protein in which no wild-type amino acid positions are conserved is 19^{100} , many orders of magnitude greater than the number of atoms in the observable universe. Extensive structural and functional characterization can help researchers narrow down to some idea of exactly which residues are essential and for what purposes, but the problem remains a daunting one. Gene duplication alone is likely to have relaxed the selective pressure on Fluc protomers which previously enforced perfect symmetry: the independent exposure of two paralogs to point mutations means that such mutations could have acted to preserve the tertiary folds but allow the primary sequence to diverge (thereby generating a pseudosymmetry in post-duplication Flucs), an effect that has also been attributed to the kind of divergence seen more generally in membrane protein internal repeats [32].

This is a salient example of how sequence homology is not enough to infer structural homology, or, by extension, functional homology. Additionally, it is challenging for computational approaches to determine from sequence conservation alone which parts of a protein are essential for function. We must therefore appeal to a concept more elaborate than sequence conservation: residue coevolution.

Residue coevolution has been described as the trend that a pair of co-evolving residues for a protein family should occur in spatial proximity [33]; however, it may be more precisely defined as a phenomenon in which mutation of one residue is, for whatever reason, correlated with a compensatory mutation in some

other residue. This correlation may indeed be due to interaction in 3D space, although there may be other reasons. Spatial proximity may be reasonably implied, but not assumed, in a case of residue coevolution: in fact, a study of more than 4000 protein families found that 25% of coevolving residue pairs were separated by more than 5 Å, and 3% of coevolving pairs were separated by more than 15 Å [33]. Anishchenko et al. also found that 35% of cases in the latter category occurred at homo-oligomeric interfaces [33].

Advancements in our understanding of residue coevolution have been a boon for protein structural prediction. Methods that identify pairs of co-evolving residues, also called evolutionary couplings, have been used to accurately represent soluble proteins as well as membrane proteins, including Fluc [34]. Many statistical models based on residue coevolution for predicting 3D contacts in protein and nucleic acid structures – such as SIFT, PolyPhen-2, and CADD – portray some position i in the protein sequence as a target of coevolution independent of background positions j [35]. This approach led to incorrect predictions and was superseded around 2017, with the introduction of a new algorithm based on protein epistatic interactions, explicitly modeling pairwise interactions between positions i and j and other context [35]. This method, named EVmutation, was reported by Hopf et al. under D. Marks and C. Sander and has since inspired similar modifications to structural prediction algorithms. EVmutation was eventually incorporated into the group's larger computational framework for identifying and analyzing evolutionarily coupled residues, EVCouplings [36], which plays a significant background role in the work presented in Chapter 3.

An understanding of how mutation at coupled positions influences secondary and tertiary structure is especially important for the Fluc given that the ancestral dual-topology phenotype is expressed as a homodimer; evolutionary couplings could act over large distances. More concretely, both experimental strategies (namely, deep mutational scanning) and computational apparatus deployed to probe evolutionary couplings could help explain variations in mutational tolerance within Fluc protein and potentially serve to reveal fine-grained steps taken in the Fluc evolutionary pathway.

1.6 Fitness landscapes model evolutionary fitness as a function of genotype or phenotype

The concept of a fitness landscape was introduced by Sewall Wright in 1932 [37] to represent evolutionary fitness as a function of genotype, with two axes representing two genetic traits and a third axis for fitness. In many depictions such as those in Figure 5, this metaphor has been taken to mean that genetic traits can be regarded as continuous variables, as can fitness.

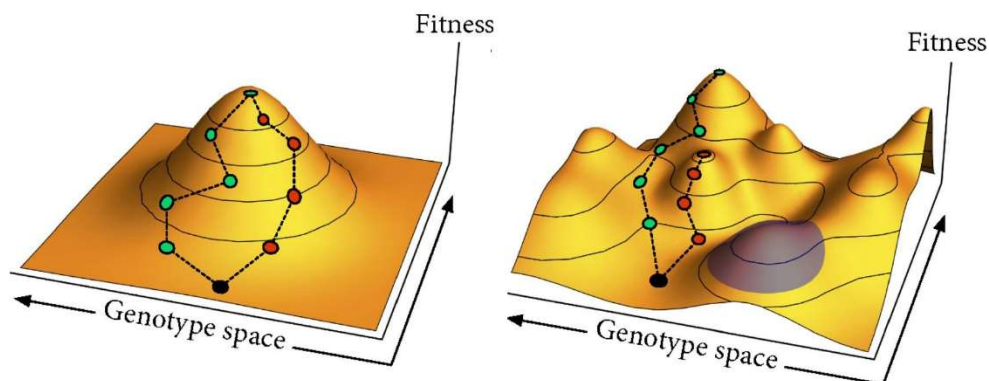


Figure 5. Example depictions of a fitness landscape. Modified from J. Van Cleve and D.B. Weissman. [38].

It has been typical to model an evolutionary pathway as a path through the landscape toward some optimal level of fitness. The concept has since been refined and expanded in many different ways, such as in the consideration of phenotypic rather than genotypic traits, or the depiction of the landscape as a contour map or heatmap. Of interest to our present investigation of Flucs is a representation we refer to as a single-site substitution matrix, a depiction of the effects due to amino acid substitution at all positions within the protein (deleterious, advantageous, or neutral), further defined in Chapter 3. We introduce this device in order to reconcile the estimated mutational propensity of Fluc residues with our incumbent, three-step evolutionary account and mechanisms of subfunctionalization.

1.7 Overview of thesis aims

We surmise that the Flucs have undergone a partial subfunctionalization in which certain overlapping functions in Fluc have become partitioned whereas others were retained. Over evolutionary time, orientation has been partitioned so that whereas ancestral Fluc monomer is dual-topology, this function became divided into two separate and complementary functions, each assumed by a distinct Fluc descendant. The Phe box motif, however, is an example of a function that appears to have been retained among the fixed-topology Flucs. The investigation documented herein seeks to help answer two questions related to this subfunctionalization:

- (1) How can we more closely examine sequence divergence in post-duplication Flucs?
- (2) How can we more closely evaluate mutational tolerance in pre-duplication Flucs?

We employ computational means to address these and related questions. The first question is considered more specifically in the context of charge bias and the differences observed in this property among a relatively small set of bacterial and archaeal Flucs, while our treatment of the second question involves the

use of EVcouplings to estimate this tolerance in a larger selection of Fluc homologs, vis-à-vis statistical correlations between coevolving residues in Fluc. In Chapter 2, we present a Python script for the simple evaluation of charge bias in Fluc loop regions. Following our account of this program and of motivating principles of software development, Chapter 3 demonstrates the use of web-based EVcouplings to effectively simulate a set of Fluc single-mutant frequencies; Chapter 3 also showcases an R script that was used to compute and analyze mutational tolerance in Fluc and its constituent domains. We discuss technical challenges associated with the development of both programs, efforts to make these programs accessible, and potential for further use. In Chapter 4, we discuss broader theoretical implications for this work in the investigation of fitness in the Fluc family and elucidating the Fluc evolutionary pathway. Chapter 4 also concludes this thesis with a discussion of theoretical concepts relevant to future work, as well as a description of an ongoing experimental project to compare fitness among Fluc variants expressed from a set of engineered plasmids in *E. coli*.

2 A program for the evaluation of charge bias in Fluc loop regions

In this chapter, we present a script written to classify among a set of homologous Fluc protein sequences, which Flucs in this set assume the “in” orientation, the “out” orientation, or have dual topology. The inputs of this program include a FASTA file listing all such sequences along with their unique identifiers, and a dataset corresponding to these sequences which also lists important sequence information, as will be explained in the following subsections. The output of this program is a new dataset which preserves the input and which denotes the applicable orientation for each Fluc sequence.

Python is a general-purpose language in widespread use and popularity. Python is open-source: it may be used, modified, and distributed freely. Its functionality can be extended with the import of libraries and modules, as this script demonstrates.

2.1 Background: Influence of charge bias on membrane protein orientation

Depending on the organism, Fluc proteins exhibit the dual-topology phenotype or the fixed-topology phenotype. As stated in Section 1.4, the former describes Fluc protomers which are equally adept at orienting themselves in the “in” orientation as well as the “out” orientation. In the fixed topology phenotype, there are qualitatively two different types of Fluc monomer; the bias in orientation emerges from the presence of positively charged residues (namely, Arg and Lys) at connecting loops or the termini of the polypeptide. The idea that placement of Arg and Lys at such positions is a strong determinant of membrane protein orientation, now validated extensively by experiment, was articulated as the *positive-inside rule* by G. von Heijne [39]. Dual-topology membrane proteins, however, are a notable exception [40]. For those

proteins which do show an orientation bias, the rule holds: it is likely that there is an excess of positively charged residues on cytoplasm-facing loops and termini. The current state of knowledge yet suggests a conservative interpretation: except for proteins consisting of a single transmembrane alpha-helix, passing through the membrane only once, it is not categorically the case that an excess of negatively charged residues occurs in loops and termini exposed to the outside surroundings of the membrane.

2.2 Applicable principles of software development

The user audience of primary interest to this work includes structural biology researchers not yet well acquainted with uses of computer programming in their activities. Apropos, there are certain principles of software development which have guided the development of the script described here and the script presented in Chapter 3. These principles are defined briefly here.

Ease of access: This refers to the ease with which users can access and use a piece of software. Naturally, to achieve greater ease of access it is important to remove barriers or difficulties associated with this use. Computer programming can be daunting to new users due to an uncertainty about how best to get started. Jupyter Notebook is a web-based, interactive computing platform that can be set up in a number of ways. To run a notebook from scratch requires several preparatory steps, but certain other platforms already take care of these and incorporate heavy use of graphical interfaces that are easy to navigate. The auxiliary platform discussed here for hosting and collaboration with Jupyter notebooks is Jovian, a platform which also offers learning resources in such areas as data science and machine learning.

Legibility: As it relates to code, this principle emphasizes the need to make code concise and readable. This entails adherence to practices such as consistent formatting (of indentations, spacing, and line breaks, for example), modularity, and comments. The scripts discussed in this work have a modular design in the sense that the code is organized in discrete, self-contained units, each with its own intended objective. Comments, prefaced with a hash symbol in the scripts presented here, also serve the important purpose of clarifying code where necessary or presenting other relevant information.

Inline documentation: Documentation is written text or illustration that accompanies code in order to describe its operation or use. Inline documentation refers to that which is somehow embedded within the code itself. Comments are a form of such documentation. However, a benefit afforded by the Jupyter notebook platform is the ability to add dedicated annotations to provide information without compromising legibility; excessive comments can make code more difficult to read. The utilization of Jupyter notebooks and the Markdown typesetting language is cast with this functionality in mind.

Version control: This concept is concerned with appropriately tracking changes to code over time, in order to facilitate collaboration and error correction. The Jovian platform is especially useful for this purpose since hosted Jupyter notebooks can also be shared with other Jovian users, with the revision history of these notebooks also recorded by the platform. Version control options in Jovian may be used to identify and track errors in code, thereby facilitating the correction of this code as needed.

On a further note, Jupyter notebooks allow for the interactive visualization of data sets and plots, a functionality of benefit to bioinformatics and molecular biology.

2.3 Methods

2.3.1 Preparation of a multi-sequence alignment file

A set of homologous Fluc protein sequences already in use was previously harvested from the Joint Genome Institute's GEBA genome project [41]; consisting of 1034 distinct sequences of bacterial and archaeal Flucs, this dataset was saved as an unaligned FASTA file entitled `geba_crcbs.fasta`. On multiple occasions, the popular alignment algorithms MAFFT and MUSCLE were deployed in attempts to generate multi-sequence alignment (MSA) files from the original FASTA file. These attempts were successful, yet it was later discovered that the EVcouplings framework (usage discussed in Chapter 3) could also generate an alignment of Fluc homologs. The MAFFT algorithm (Version 7) [42] was run at the command-line level using Windows Powershell, whereas a version of the MUSCLE algorithm was run in the open-source software Jalview [43].

2.3.2 View of script and accompanying inline documentation using Jovian and Jupyter notebook integration

Having revealed our means of generating a serviceable MSA file to be a main input for this program, we now present a closer look at the use of Jovian and the integration of a Jupyter notebook in which the script has been written and annotated.

A Jupyter notebook was created to host the script and its annotations, previewed in Jovian as shown in Figure 6. Collectively, these annotations offer inline documentation within the program; they are recapitulated in the synopsis of the script as presented in Section 2.4. The version of Python used in this work is version 3.9.6.

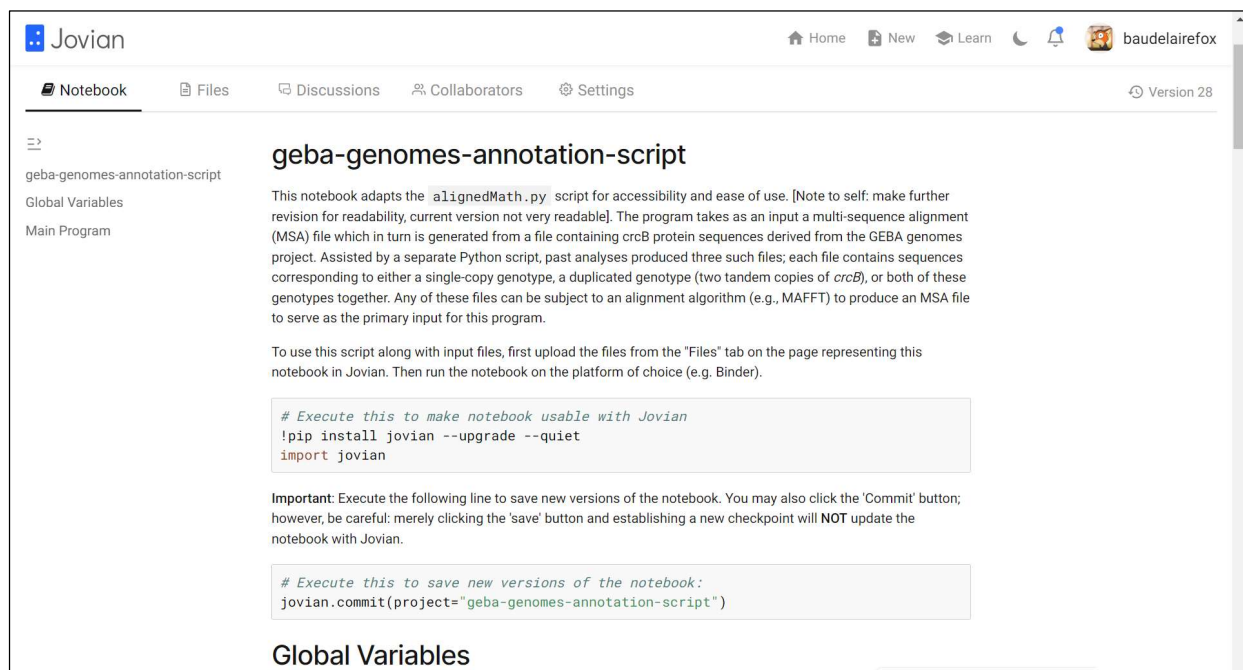


Figure 6. Screenshot of a recent version of the charge bias script as a Jupyter Notebook, viewed using the Jovian online platform and website.

2.4 Structure and synopsis of program

The script begins with commands that enable the notebook to be used with Jovian. Although the notebook may be initialized independently of Jovian, the Jovian platform offers convenient links to run a notebook on such web-based environments as Binder, Google Colab, or Kaggle. Using the platform website, these options also automate a handshake between Jovian and the platform of choice. (Note: Whether running the notebook locally or on these web-based platforms, it is necessary that all required inputs, including the input MSA file, are accessible by the program). For simplicity, we will follow the script as though it were being run in a web-based environment already authenticated with Jovian. A flowchart summarizing this script is presented in Figure A1 of the Appendix.

```
!pip install jovian --upgrade --quiet
import jovian
```

The following line may be executed as frequently as desired to save the notebook with Jovian:

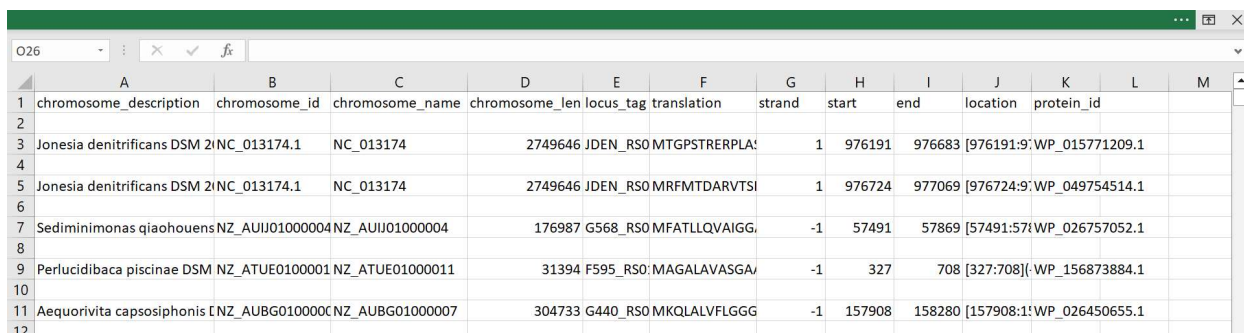
```
jovian.commit(project="geba-genomes-annotation-script")
```

The next lines of the script serve to import the Python csv module and instantiate several global variables.

```
import csv
inputGenesFileName = 'geba_aligned.fasta' # Can modify to match actual file name
inputFileName = 'geba_crcb_genes.csv' # Use output from other program
outputFileName = 'aligned.csv'
locusTagIndex = 4
```

The primary input is the MSA file, the name of which is stored here in the variable `inputGenesFileName`. As mentioned earlier, the MSA file is a FASTA alignment file generated from previous analysis of harvested Fluc sequences.

Serving as another input is a CSV file, stored as `inputFileName`, containing key information about the genomes sampled for the alignment, in the format illustrated as shown in Figure 7. From left to right, the attributes represented in this dataset are `chromosome_description`, `chromosome_id`, `chromosome_name`, `chromosome_len`, `locus_tag`, `translation`, `strand`, `start`, `end`, `location`, and `protein_id`. Exactly 1033 instances of the *crcB* gene are represented in the dataset of `geba_crcb_genes.csv`. Our particular copy of this CSV file was generated from a prior analysis conducted by C. Macdonald in the Stockbridge group, entitled `geba_crcb_genes.csv`. Notice in Figure 7 that the fifth column of this CSV file was made to store values for a certain attribute known as the genomic locus tag: the script stores the index for this column as the variable `locusTagIndex`.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	chromosome_description	chromosome_id	chromosome_name	chromosome_len	locus_tag	translation	strand	start	end	location	protein_id		
2													
3	Jonesia denitrificans DSM 2	NC_013174.1	NC_013174	2749646	JDEN_RS0	MTGPSTRERPLA	1	976191	976683	[976191:9:WP_015771209.1			
4													
5	Jonesia denitrificans DSM 2	NC_013174.1	NC_013174	2749646	JDEN_RS0	MRFMTDARVTSI	1	976724	977069	[976724:9:WP_049754514.1			
6													
7	Sediminimonas qiaohouensis	NZ_AUIJ01000004	NZ_AUIJ01000004	176987	G568_RS0	MFATLLQVAIGG	-1	57491	57869	[57491:57:WP_026757052.1			
8													
9	Perlucidibaca piscinae DSM	NZ_ATUE01000011	NZ_ATUE01000011	31394	F595_RS0	MAGALAVASGA	-1	327	708	[327:708]:WP_156873884.1			
10													
11	Aequorivita capsosiphonis	ENZ_AUBG01000007	ENZ_AUBG01000007	304733	G440_RS0	MKQLALVFLGGG	-1	157908	158280	[157908:1:WP_026450655.1			
12													

Figure 7. Dataset from `geba_crcb_genes.csv` as viewed in Microsoft Excel.

The script then proceeds to declare certain other important variables.

```

lettersToCount = ['R', 'K']      # Arginine and lysine are specified here.

addRanges = [[1, 31], [85, 95], [144, 205]]
# A list of lists demarcating the regions of a protein sequence that correspond
# to loop regions occurring inside of the membrane.

subRanges = [[47, 69], [117, 129]]
# Another list of lists, but for loop regions occurring on outside of membrane.

```

As the comments indicate, the `addRanges` and `subRanges` variables store lists corresponding to loop regions occurring within the consensus sequence of the MSA file. In other words, these ranges are specific to the consensus sequence recorded in the MSA file. For the purposes of this script, the C-terminal and N-terminal regions in their entirety are considered loop regions. These regions correspond to the first and third lists stored in `addRanges`.

There are five functions written in the remainder of the script. We will discuss these in the order that they appear in the script.

```

def countInRange(string, chars, num_range):
    runningTotal = 0
    cropString = string[(num_range[0]-1): num_range[1]]
    for char in chars:
        runningTotal += cropString.count(char)
    return runningTotal

```

This function consists of three arguments, `string`, `chars`, and `num_range`. There are two local variables: `runningTotal` and `cropString`. The parameters passed into the function as `chars` and `num_range` are lists. What is stored in `num_range` is the result of slicing the string stored as `string` according to a slicing syntax involving the value of the first entry in `num_range`, having subtracted 1 from this value, as well as the second entry in `num_range`. The for loop iterates over every entry in the `chars` list, updating the value of `runningTotal` with each iteration.

The next function includes parameters which evaluate the `countInRange` function.

```

def testCountInRange():
    shouldBeOne = countInRange('hello', ['l', 'x'], [1,3])

```

```

print(shouldBeOne)
shouldBeTwo = countInRange('hello xx there', ['l', 'x'], [4, 7])
print(shouldBeTwo)

```

As the name implies, the function `testCountInRange` does not actually play a role in the analysis but serve only as an option for the user to determine whether the function is working as intended. The result of calling `countInRange` for the two sets of parameters shown above will store the values 1 and 2 in the local variables `shouldBeOne` and `shouldBeTwo`, respectively. For example, passing the string 'hello' and the lists ['l', 'x'] and [1,3] into `countInRange` will result in the slicing string [0:2]. The result of slicing the string is 'hel', which is stored as the variable `cropString`. The for loop serves to count the number of times that each value in the `chars` list appears in `cropString`. In this case, because there is only one instance of 'l', the value of `shouldBeOne` becomes 1. A similar line of reasoning will show that the value of `shouldBeTwo` will assume a value of 2.

The next function is defined similarly as `countInRange`.

```

def countInRanges(string, chars, ranges):
    totalCount = 0
    for num_range in ranges:
        totalCount += countInRange(string, chars, num_range)
    return totalCount

```

This function returns the total number of instances that a string in the `chars` list occurs in `string`, over all of the different `num_range` lists included as the `ranges` argument. Therefore, the `ranges` argument is a list of lists. Note also that the counter variable `totalCount` is updated for each iteration of the for loop.

```

def getSequenceDictionary():
    numPos = 0
    outDict = {}
    with open(inputGenesFileName, 'r') as inFile:
        target = ''
        line = inFile.readline().rstrip()
        while True:
            target = line[1:]
            readingSequence = True

```

```

sequence = ''
while readingSequence:
    line = inFile.readline().strip()
    if '>' in line:
        addScore = countInRanges(sequence, lettersToCount, addRanges)
        subScore = countInRanges(sequence, lettersToCount, subRanges)
        totalScore = addScore - subScore + 1
        print('{} , {} , {}'.format(target, totalScore, sequence))
        if totalScore > 0:
            numPos += 1
            outDict[target] = 'in'
        else:
            numPos -= 1
            outDict[target] = 'out'
        break
    else:
        sequence += line
    if not line:
        ## first process last line
        ##
        addScore =
            countInRanges(sequence, lettersToCount, addRanges)
        subScore =
            countInRanges(sequence, lettersToCount, subRanges)
        totalScore = addScore - subScore + 1
        print('{} , {} , {}'.format(target, totalScore, sequence))
        if totalScore > 0:
            numPos += 1
            outDict[target] = 'in'
        else: # if totalScore < 0:
            numPos -= 1
            outDict[target] = 'out'
        print('we had a pos/neg balance of {}'.format(numPos))
    return outDict

```

This function is defined to include some counter variable numPos and a dictionary named outDict.

The with statement is accompanied by a file object returned by open(), into which inputGenesFileName is passed along with the flag 'r'. This flag selects a mode for open() so as to only read the file in question. Other modes enable open() to open a file for writing and editing.

The first `while` loop is preceded by declaration of the variables `target` and `line`. The variable `target` is initialized as an empty string whereas a method is invoked in the declaration of the variable `line`. This method, using `rstrip()`, effectively removes any whitespace that occurs at the end of the string returned by `readline()`.

Within the first `while` loop, `target` is redefined to store the result of slicing the string `line` as indicated by `line[1:]`. The `readingSequence` variable stores the Boolean value `True`, which will allow the nested `while` loop to run. Preceding this nested `while` loop is an initialization of the variable `sequence` as an empty string.

The last function in the script, `writeSequenceResult`, is the most elaborate.

```
def writeSequenceResult(sDict):
    with open(inputFileName, 'r') as inGenesFile:
        with open(outputFileName, 'w') as outFile:
            outFile.write(inGenesFile.readline().rstrip()) # copy header
            newCol = 'direction'
            outFile.write(',')
            outFile.write(newCol)
            outFile.write('\n')
            #header now updated
            reader = csv.reader(inGenesFile, dialect='excel', quotechar='')
            for inLine in reader:
                # print(inLine)
                inLine.pop() #we have an extra/empty item we want to discard.
                inLine[0] = '"{0}"'.format(inLine[0])
                # fix first column with space in it
                outFile.write(','.join(inLine))
                outFile.write(',')
                try:
                    target = inLine[locusTagIndex]
                    outFile.write(sDict.get(target.strip(), 'dual'))
                    # strip because they have a space at the start,
                    # use dual as the default
                    outFile.write('\n')
                except:
                    print('error with that line{}'.format(inLine))
```


This function has a dictionary, `sDict`, as an argument. The first line of the function opens a file object for `inputFileName` in reading mode using the `with` statement. The second line opens a file object for `outputFileName` in writing mode. Both files are opened simultaneously using nested `with` statements.

The third line of the function writes the first line of the input file, as returned by `inGenesFile.readline()`, to the output file, with any trailing whitespace removed using `rstrip()`.

The fourth line defines a new variable, `newCol`, to store the string 'direction', which is then written to the output file preceded by a comma but followed by a newline character ('\n').

The fifth line of the function initializes a `csv.reader` object, called `reader`, which reads from the input file object, `inGenesFile`, with the `dialect` and `quotechar` parameters chosen to optimize the reading of the input according to its format.

For each line in the input file (represented by `inLine`), the penultimate item is removed from the list using the `pop()` method. The first item of the list is then formatted as a string surrounded by double quotes using the `format()` method.

After joining all the remaining elements of `inLine` with commas, the resulting string is written to the output file, followed by a comma. The variable `target` is assigned as the value of the element of `inLine` at `locusTagIndex`.

Finally, the value in the `sDict` dictionary corresponding to the key `target` is written to the output file, with 'dual' as the default value if `target` is not found in `sDict`. A newline character is then added to the output file. If an exception is encountered, a message is printed to the console stating that there was an error with that line.

The concluding lines of code in the script constitute the entry point of the program.

```
if __name__ == '__main__':
    # testCountInRange()
    sequenceDict = getSequenceDictionary()
    writeSequenceResult(sequenceDict)
```

The `if` statement checks if the module is being run as the main program, as opposed to being imported as a module. This is a common Python idiom to ensure that the code within the `if` statement is only run when the module is being executed as the main program.

The two functions `getSequenceDictionary()` and `writeSequenceResult()` are then called, in this order. As advertised, the former function returns a dictionary of sequence identifiers and their corresponding statuses as either “in” or “out” based on the scores calculated from the `countInRanges` function for each sequence in the input CSV file. Then, the function `writeSequenceResult()` takes the dictionary returned by `getSequenceDictionary()` as an argument and writes the output to a file whose name is stored as `outputFileName`. In this case, that name is `aligned.csv`.

2.5 Discussion of expected output

The output of this script would be a new CSV file containing all of the input data (represented in `geba_crcb_genes.csv`, in this case) with an additional column named “direction” appended to it, denoting the new attribute. As we have seen, the values in the “direction” column are based on a sequence dictionary generated from the input file. If a locus tag in the input file is found in the sequence dictionary, the corresponding value in the “direction” column will be either “in” or “out” depending on the value retrieved from the dictionary. If the locus tag is not found in the dictionary, the value in the “direction” column will be “dual,” denoting the dual-topology state as the most likely phenotype.

If there were relatively more instances of the letters “R” and “K” for a given protein sequence in the regions defined by `addRanges`, the corresponding total score (given as `totalScore`) will be higher in value, which would make it more likely for the protein overall to be labeled as “in” in the new `direction` attribute. This is because the function `addRange()` checks for the presence of the amino acids Arg and Lys in the protein sequence at the specified positions, incrementing the variables `addScore` and `subScore` accordingly.

An example output file has not been made available because of several outstanding changes to the script that have as yet not been made; these changes and the reasons for implementing them are discussed in the next subsection. Collaboration on these changes via Jovian is also advisable.

2.5.1 Reflection on the development and optimization of the program

All core functions of the program are drawn primarily from a pre-existing but unoptimized Python script formerly in use by the Stockbridge group. The most significant changes ensued from the adaptation of this

script into a Jupyter notebook and made available to other lab members using Jovian. However, one other important change solved a potential compatibility issue that would have arisen in the original script. The original script named the `num_range` variable as `range`; this naming is not ideal because of the built-in `range()` method that has been available in newer versions (3.x) of Python. In light of this problem, the script was assessed qualitatively for any other compatibility issues involving the use of reserved keywords, although no further issues were identified. As expected, replacing all instances of `range` with the new variable name (`num_range`) did not change the functionality of the script.

More generally, there exist several areas for improvement in the program that are likely worth addressing before further use. These include (1) handling of errors from an empty input, (2) output file formatting, and (3) scalability.

The code as yet implemented does not account for the case where the `line` variable has no stored value. The first is to modify the `getSequenceDictionary()` function so that it can test whether the first line read from the file is an empty string. If such is the case, the program could return an empty dictionary and an optional error message. This could be achieved by inserting a certain `if` statement ahead of the `while` loops, as shown below.

```
def getSequenceDictionary():
    numPos = 0
    outDict = {}
    with open(inputGenesFileName, 'r') as inFile:
        target = ''
        line = inFile.readline().rstrip()
        if not line: # handling of empty file
            return outDict # this dictionary is empty
            # print('Error: empty first line'.format(inLine))
        while True:
...

```

The script could be similarly amended to handle problems with reading the protein sequence. Exclusively, the program inspects for the characters 'R' and 'K', but cases in which the protein sequence includes invalid characters (e.g., characters which do not represent amino acids) are causes for concern and could be more robustly handled. For example, if a protein sequence contains such invalid characters, the code could be further modified to recognize them and return a warning message when applicable.

Next, the output file format is not clearly defined in the code. The current implementation assumes that the output is a CSV file (like the main input), but this is not necessarily the outcome for the code as currently written. To ensure that the output is a valid CSV file, the `writeSequenceResult()` function can be modified to use the `csv.writer` and `csv.reader` modules. The code below offers an example of such a modification.

```
def writeSequenceResult(sDict):
    with open(inputFileName, 'r') as inGenesFile, open(outputFileName,
        'w', newline='') as outFile:
        writer = csv.writer(outFile, dialect='excel')
        header = inGenesFile.readline().rstrip().split(',') # copy header
        header.append('direction')
        writer.writerow(header)
        reader = csv.reader(inGenesFile, dialect='excel')
        for inLine in reader:
            inLine.pop() # we have an extra/empty item we want to discard.
            inLine[0] = "{0}".format(inLine[0])
            # fix first column with space in it
            target = inLine[locusTagIndex].strip()
            inLine.append(sDict.get(target, 'dual')) # use dual as the default
            writer.writerow(inLine)
```

Finally, the current version of the program processes all gene sequences in memory at once, which could be problematic if the input file is very large. The code could be modified to process sequences one at a time to improve scalability.

2.5.2 Utility of program for understanding charge bias characteristics

Despite there being exceptions to the positive-inside rule, it is generally reliable to determine the orientation of most TMDs by the simple counting of positively charged residues in cytoplasmic loops [44], as demonstrated by this script. Membrane potential can differ according to species as well as various metabolic and environmental conditions, yet our algorithm for evaluating charge bias is robust because we have limited our analysis to homologs of the same protein (Fluc), rather than different membrane proteins across different species.

The script could also prove a useful starting point for the annotation of phylogenetic trees for the Flucs: the reference genomes represented by the GEBA project were selected to maximize phylogenetic and physiological diversity [41]. Further analysis of the program output, along with an understanding of

mutational tolerance in Fluc, may be used to inform phylogenetic assignment and refine notions of what possible mutations could feasibly have led to the emergence of fixed-topology Flucs.

3 A program for the quantification and visualization of mutational tolerance

In this chapter, we present a script to compute a mutational tolerance score according to a definition of mutational tolerance as a population variance: this population variance applies to a given position on the protein of interest (here, Fluc) and is calculated among the single-mutant frequencies determined for all single-mutant Fluc variants with respect to this position. We introduce the use of EVcouplings to generate these variant frequencies and produce useful visualizations of Fluc sequence conservation and the simulated variant frequencies as a heatmap. The script also is capable of writing the set of computed mutational tolerance scores to a new text file; an attempt was made to overwrite a part of the Fluc-Bpe PDB file with these scores, although this functionality would have to be implemented in a future version of the program, as discussed in subsequent sections. The specific part of the PDB file to be overwritten is the file’s “b-factor” column, a part of the file denoting a measure of spatial uncertainty, the eponymous b-factor for each atom. Section 3.4 includes reference to a pre-existing, open-source Python script which may guide further development of the program.

R was introduced in the 1990s and has since grown in popularity for fields involving data analysis, data visualization, and statistical computing. In regard to the latter, base R incorporates a wide range of functions for this purpose. Like Python, the functionality of R can be extended through the use of packages, which are typically installed from the centralized R repository CRAN (Comprehensive R Archive Network). The version of R used in this work is version 4.2.1.

To date, the most popular development environment for R is RStudio; the version of RStudio used to compose the script for this program is RStudio 2022.07.2, build 576.

3.1 Background: The single-site substitution matrix as a depiction of mutational tolerance

A single-site substitution matrix is a representation of the effect that single substitutions have on a protein’s function, over the entire sequence space of the protein. The matrix is typically depicted as a heatmap with position along the horizontal and choice of amino acid on the vertical, as well as coloring to indicate the effect of the mutation. The effect chosen may be ligand binding, catalysis, or something else. If the effect is the frequency with which a mutant occurs, this in turn could be considered a measure of fitness afforded by the mutant, given a certain set of conditions. The results of relatively new experimental methodologies which assess thousands of mutations at a time – namely deep mutational scans and MAVEs (Multiplex Assays of Variant Effects) – are typically represented as single-site substitution matrices.

In a deep mutational scanning (DMS) experiment, one begins with a library of protein variants which can be expressed in an appropriate selection system. For example, if the activity of interest was the binding of a protein of interest to some ligand, this system could involve a protein display combined with a type of competitive binding assay. Most appropriate for understanding the effect of Fluc mutation on fitness would be a bacterial cell culture system where growth depends on Fluc activity. In Figure 8, the “selection” graph represents the changes in the frequency of each variant due to competitive selection – the actual selection occurring, for example, in a single competitive growth assay. The selection pressure could be imposed by a fixed concentration of NaF in the growth media. The post-selection library resulting from this selection is then subject to high-throughput DNA sequencing, with the resulting data analysis yielding functional scores for each variant that would allow one to gauge which mutations prove to be deleterious, neutral, or advantageous [45]. The resulting single-site substitution matrix in such an experiment could be more generically called a “mutational landscape” or a “fitness landscape,” although the latter term predates this technology and has been used in many other senses since its coinage [37].

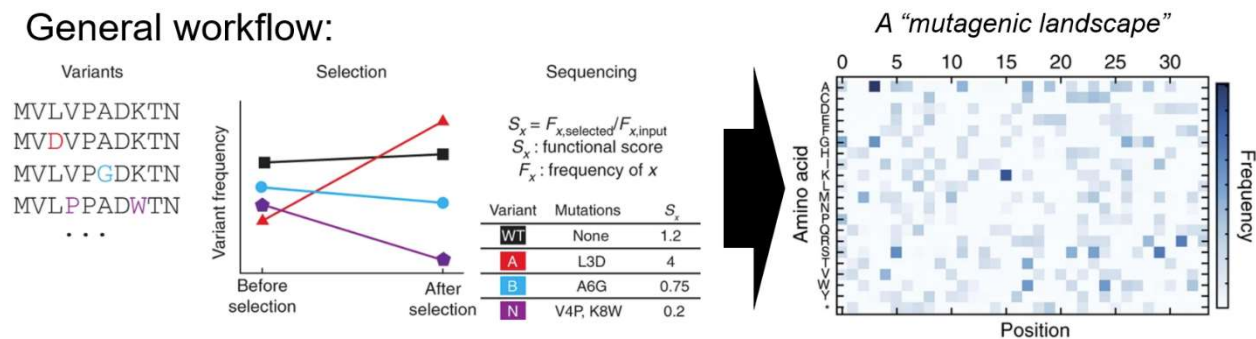


Figure 8. General workflow of a DMS experiment. Modified from D.M. Fowler et al. (2014) [45].

Complementary to DMS and related approaches are computational means of modeling the effects of mutation. EVcouplings is an example of such an approach, capable even of generating an *in silico* version of a single-site substitution matrix. Moreover, EVcouplings exploits the principle of evolutionary conservation to predict mutational effects [35], originating from sequence data across potentially thousands of different species, which is generally not an option for a single DMS experiment. We will observe the use of EVcouplings for producing a single-site substitution matrix and for purposes more germane to the function of a program for quantifying mutational tolerance in Fluc-Bpe.

3.2 EVcouplings is a model computational framework for evolutionary couplings analysis

In Chapter 1, we discussed the value of EVcouplings as a tool for the analysis of residue coevolution and how it has applications for protein structure prediction. Undergirding the development of EVcouplings is

an approach known as direct-coupling analysis (DCA) which invokes maximum entropy modeling to infer the probability distribution of pairwise interactions between residues, derived from a large number of homologous protein sequences, a method that can even account for coupled residues separated by long distances [46]. The computational pipeline encompassed by EVcouplings is represented in Figure 9.

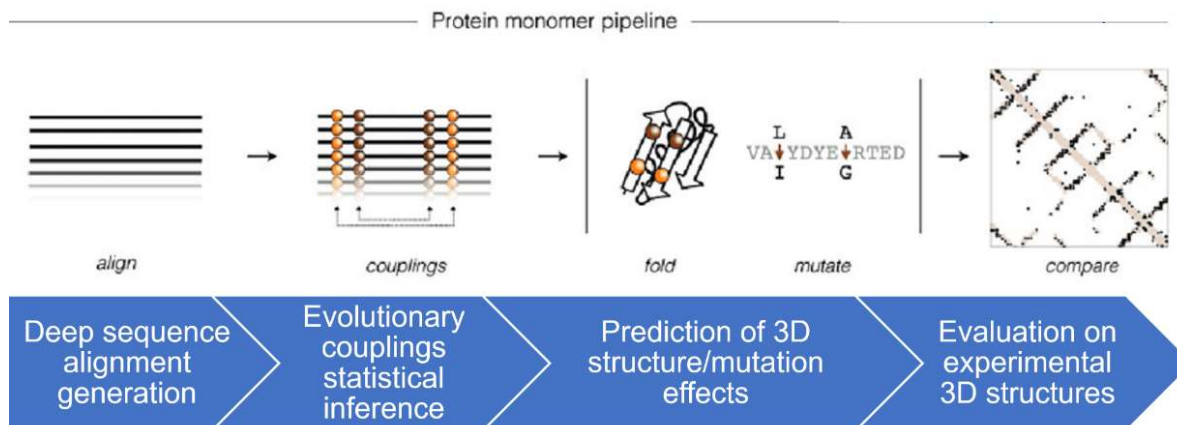
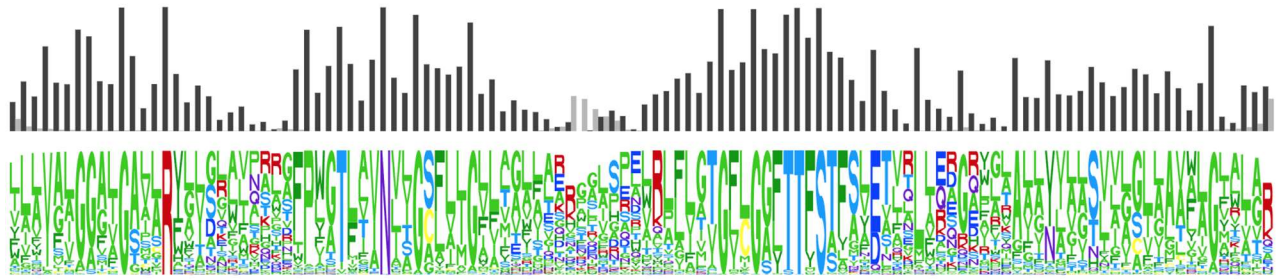


Figure 9. Stages of the EVcouplings computational pipeline. Modified from T.A. Hopf et al. (2019). [36].

Here, we use EVcouplings to evaluate a phenotype of interest (the ability of an organism to survive under conditions of fluoride stress) by proxy: the inferred structural and functional changes in Fluc due to a single mutation. For any relatively strong coupling between residues, a deleterious mutation in one of these residues has a detrimental impact on structure related to that expected from a similar mutation in the coupled position. If the user input and subsequent database search performed by EVcouplings produce metadata of high quality, EVcouplings can then construct a single-site substitution matrix based on the inferred couplings; presented in Figure 10 and Figure 11 are examples of the analyses generated by EVcouplings for Fluc-Bpe.



Position: 82			Position: 85		
Residue	Frequency	Count	Residue	Frequency	Count
Phenylalanine [F]	81.36%	19951	Phenylalanine [F]	69.75%	17103
Tyrosine [Y]	12.77%	3131	Tyrosine [Y]	10.55%	2586
Methionine [M]	2.57%	629	Leucine [L]	8.35%	2048
Valine [V]	1.36%	334	Alanine [A]	5.15%	1263
Leucine [L]	0.9%	220	Tryptophan [W]	1.88%	462
Tryptophan [W]	0.83%	203	Methionine [M]	1.84%	450
Histidine [H]	0.11%	26	Valine [V]	1.58%	388
Threonine [T]	0.04%	9	Histidine [H]	0.29%	70
Isoleucine [I]	0.02%	6	Isoleucine [I]	0.2%	50
Alanine [A]	0.02%	4	Threonine [T]	0.18%	43
Cysteine [C]	0.01%	3	Serine [S]	0.13%	32
Serine [S]	0.01%	2	Cysteine [C]	0.05%	13
Unknown [X]	0.01%	2	Proline [P]	0.01%	3
			Asparagine [N]	0.01%	2
			Unknown [X]	0.01%	2
			Glutamic Acid [E]	0.01%	2
			Glutamine [Q]	0%	1

Figure 10. Sequence conservation analysis of Fluc-Bpe using EVcouplings.

The sequence conservation information reported by EVcouplings is consistent with our current understanding of Fluc. As discussed further in Section 3.3.1, these visualizations are readily obtained from EVcouplings given the Fluc-Bpe protein sequence. However, for the purposes of the script, we instead use another output of this analysis for use as an input. The main output of the script is a set of mutational tolerance scores and a histogram to summarize their distribution.

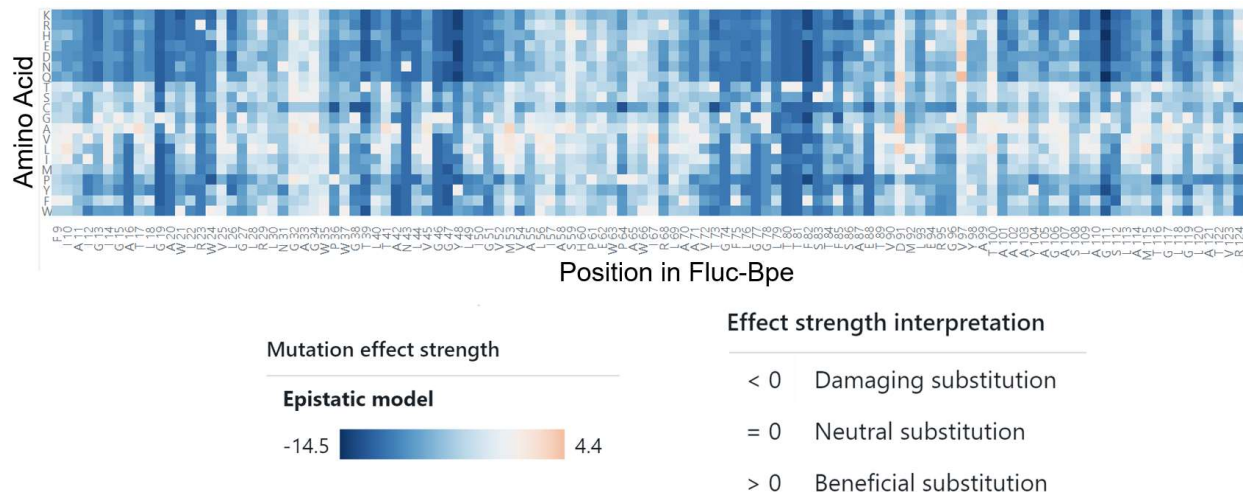


Figure 11. Single-site substitution matrix of Fluc-Bpe generated with EVcouplings.

It is noteworthy that, as shown in Figure 11, EVcouplings predicted that all substitutions at positions 82 and 85 led to a damaging effect: these positions are the locations of Phe in the phenylalanine box motif occurring in Fluc-Bpe.

3.3 Methods

We emphasize that the specific function of the EVcouplings pipeline used for this program serves to transform some protein sequence as input and conduct a homology search through a protein sequence database (e.g., from UniProt) to assemble a multi-sequence alignment, based on coupled positions within the sequence space of the protein. UniRef90 is the database for which the alignment and homology search discussed in the previous sections was executed; web-based EVcouplings also allows for a query performed with UniRef100 or with an option denoted “MGnify + UniProt,” yet at the command-line level, the user may input virtually any database of interest.

For a homology search, UniRef90 is an apt choice because it is redundancy-reduced and contains isofunctional clusters of homologous protein sequences. Compared to using a database that is not redundancy-reduced but generates similar results, a query on UniRef90 is likely to require less running time for the computation. As of May 2023, the UniProtKB database contains nearly 250 million sequence entries; UniRef90 is approximately 60% the size of the UniProtKB record [47].

Following our general discussion of how EVcouplings achieves this end, the next few sections will focus on how an R script can be applied to an output of the EVmutation algorithm to compute mutational tolerance scores. The parameters used by EVmutation to generate this output are described in Section 3.3.1.

3.3.1 Generating Fluc variant information and accompanying CSV file with EVmutation

We present here a walkthrough of our initial analysis on Fluc-Bpe facilitated by EVcouplings and its web-based GUI (graphical user interface), initially performed May 2022 but recurrently validated in the months since. The main input for this analysis can take the form of a UniProt accession number or entry name, a FASTA sequence, or a raw protein sequence, to name a few types. Here, we provided EVcouplings the UniProt accession code Q7VYU0, which in the database has the associated protein name “Putative fluoride ion transporter CrcB” and organism name “Bordetella pertussis (strain Tohama I / ATCC BAA-589 / NCTC 13251)”.

From the user submission page, there are also a set of parameters that can be adjusted, but which were left unchanged from their default values for the purposes of this work. The parameters for the multi-sequence alignment and homology search are noteworthy and include the following: Bitscore, Alignment coverage of target, Sequences, Seqs/L, and Quality. Options for changing these parameters are presented is shown in Figure 12. We define these terms in the following paragraphs.

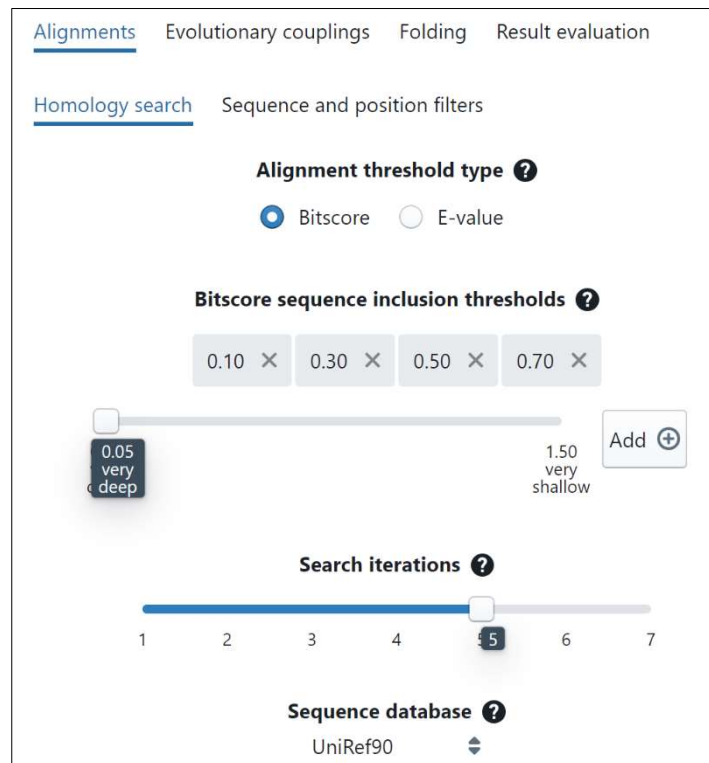


Figure 12. EVcouplings alignment and homology search options.

Bitscore (or bitscore inclusion threshold): A measure of the quality of an alignment, with a higher score representing a better alignment. An alignment can be improved if the length of the alignment can be increased or if matches between aligned characters can be improved. The bitscores provided by

EVcouplings represent threshold values. Each alignment is assigned a bitscore against each threshold; usually, an alignment with a bitscore clearing a higher threshold will have an alignment of better quality than an alignment clearing a lower bitscore threshold. Also, an alignment that clears a higher bitscore threshold will often be comprised of a smaller set of sequences than an alignment that has cleared a lower threshold. According to documentation, it is advisable to select the alignment that clears the highest threshold and which also has the highest quality rating reported by EVcouplings (see “Quality”). A desirable alignment will offer sufficient coverage of the target sequence and a high enough number of sequences that make up the alignment. By default, there are four bitscore thresholds set by EVcouplings: 0.1, 0.3, 0.5, 0.7. However, values ranging from 0.05 (“very deep”) to 1.50 (“very shallow”) can be used. In EVcouplings, bitscore inclusion threshold values are length-normalized, meaning that they are comparable between different proteins (therefore, different input sequences) and between different databases; length-normalized bitscore thresholds are recommended for use as comparable measures of evolutionary distance.

Alignment coverage of target: As shown in Figure 12, blue bars shown in web-based EVcouplings represent the parts of the target sequence for which a reliable alignment could be generated. Positions within the target sequence with an excessive number of gaps are represented by a thin black line. In the advanced settings, this constraint is specified by a “position filter.” This setting allows the program to only use sequence positions with a certain occupancy (measured as a percentage) in statistical inference. For example, a filter set to 70% will exclude all sequence positions for which gaps occur in more than 30% of sequences at these positions. By default, the position filter is set to 70%. According to EVcouplings documentation, “excluding positions with too many gaps helps to avoid spurious correlations between positions that lack evolutionary information in most identified homologs.” To yield greater coverage, it is suggested to (1) make the criteria for inclusion in the alignment (such as the bitscore inclusion threshold) more stringent, (2) make the position filter less stringent, or (3) specify a different input sequence.

Sequences: The number of identified homologs of the target sequence, after applying the bitscore inclusion threshold and after downweighting similar sequences according to the number of effective sequences tolerated. Here, the number of sequences reported depends on two filters found in the advanced settings, entitled “Removing Similar Sequences” and “Downweighting similar sequences.” As a general principle, for two sequences A and B of increasing similarity (that is, highly redundant sequences), Sequence A is less likely to contribute any informative value to the alignment than Sequence B (and vice versa). When similar enough (set to 80% by default), sequences A and B will be organized by EVcouplings into a cluster that is treated as a single, effective sequence; each sequence in a cluster of n sequences is “reweighted to contribute with weight $1/n$ during inference” ($n = 2$ in this example since there are only two sequences, A

and B; clustering can be performed for n sequences). If a sequence is exceedingly similar (over 90% by default) to another in the alignment, it is removed entirely from further consideration.

Seqs/L: The number of redundancy-reduced identified homologs (i.e., “Sequences”) divided by the number of confidently aligned positions, denoted “L”. According to documentation, this metric is better suited for comparison between proteins of different lengths.

Quality: The quality score assigned by a machine-learning model of the evolutionary couplings associated with a given alignment. The higher the score, the better the quality of the evolutionary couplings dataset. The quality score ranges from 0 to 10. The criteria for quality assessment were discussed extensively by Hopf et al. in 2014 [48].

A screenshot of the result overview obtained from this job is shown in Figure 13.

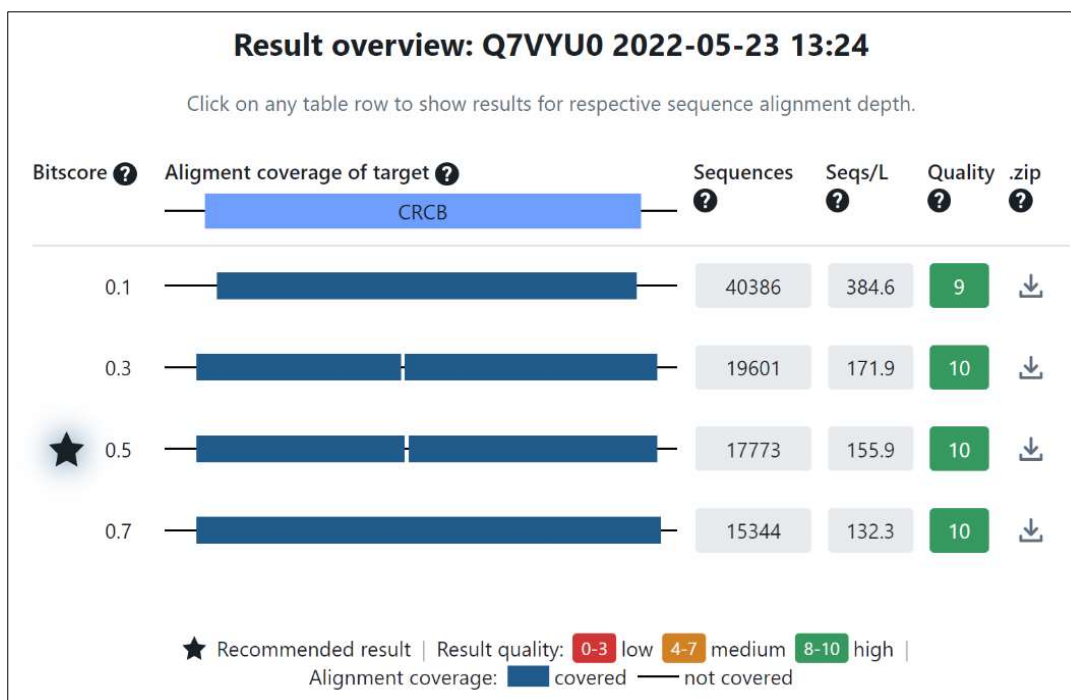


Figure 13. Screenshot of result overview for EVcouplings job on Fluc-Bpe (Uniprot: Q7VYU0).

As can be seen, EVcouplings executed a total of four analyses, one for each bitscore value and calculating high quality scores for all of them. We proceeded with the datasets for the recommended result, the analysis obtained with bitscore 0.5. Accessible with other results from the site-provided ZIP file or individually from navigating through this analysis on the web platform, we recovered a CSV file described as the “predicted single substitution effects,” and named this file `5NKQ_single_mutant_matrix.csv`. It is this CSV file and dataset, generated by EVmutation, which will serve as the input for the main R script. Also, other

outputs of this analysis include the sequence conservation illustration captured in Figure 10 and the single-site substitution matrix in Figure 11.

As of the publication of this thesis, web-based EVcouplings can be accessed at the following URL address:

<https://v2.evcouplings.org>

The information generated with EVcouplings for this analysis can be accessed as downloads from the following URL address: <https://v2.evcouplings.org/results/44570812be4e4227972621393375ad85/>

3.3.2 Use of R, RStudio, and Markdown to write and document an early version of the program

For the purposes of a related course project, an earlier version of the script was written in R, rather than Python. It is this version of the program that is the main subject of discussion, although steps toward a re-implementation of the same program in Python are described in Section 3.6.5. R was also used in order to explore the use of an alternative means of presenting inline documentation: the Markdown markup language was used to create an HTML file which follows the script about as closely the synopsis put forth in Section 3.4, with some visual similarities to the presentation of code in a Jupyter Notebook. Unlike Jupyter Notebook scripts shared via Jovian, however, Markdown by itself offers no options for cloud-based collaboration and version control.

3.3.3 Interpretation of population variances among Fluc variant frequencies as mutational tolerance scores

For the dataset we generated to serve as an input for this program, we will observe that the associated CSV file contains a column of values for the frequency of each variant, as computed by EVcouplings. Any particular frequency is relative; all of the computed variant frequencies for a position i sum to 1. To use simple terms, it is reasonable to define mutational tolerance as a population variance σ_i^2 as written in Equation 2 for a total number of distinct variants N given position i . For our purposes, $N = 19$ for each and every (computable) position i because EVcouplings computes a frequency for every possible amino-acid substitution at said position. (A caveat is that EVcouplings discards simulated data of poor quality, which was observed to happen in the Fluc-Bpe analysis for positions 1 through 8 and for positions 124 through 128; this is discussed further in Section 3.4).

$$\sigma_i^2 = \frac{\sum_{k=1}^N (f_k - \mu_i)^2}{N}$$

Equation 2. Definition of population variance at a position i for N possible variants at this position.

To reiterate, f_k denotes a particular variant frequency occurring at position i while μ_i denotes the population mean, the mean of all variant frequencies in the set $\{f_1, f_2, \dots, f_N\}$ applicable for a position i .

It is worthwhile to question the extent to which interpreting σ_i^2 as a mutational tolerance score is valid. Although the use of σ_i^2 is presented without further justification in our synopsis of the script (Section 3.4), an alternative quantification of mutational tolerance is referenced in Section 4.4.

3.4 Structure and synopsis of program with example output

A bullet-point outline summarizing this program is presented in Figure A2 of the Appendix.

First, we introduce code to read the CSV file and inspect elements of the resulting dataframe.

```
EVdata <- read.csv("5NKQ_single_mutant_matrix.csv")
head(EVdata)
```

Sample output for this code is provided below. Running `dim(EVdata)`, it was found that the dataframe EVdata measures 2166 rows and 9 columns in size.

```
##  segment mutant pos wt subs  frequency column_conservation
## 1      NA    F9A  9  F    A 0.0736405551          0.3027515
## 2      NA    F9C  9  F    C 0.0130725857          0.3027515
## 3      NA    F9D  9  F    D 0.0000000000          0.3027515
## 4      NA    F9E  9  F    E 0.0002655288          0.3027515
## 5      NA    F9G  9  F    G 0.0027184221          0.3027515
## 6      NA    F9H  9  F    H 0.0002602182          0.3027515
##  prediction_epistatic prediction_independent
## 1          -1.718659          -0.2746668
## 2          -5.009361          -2.0033915
## 3          -9.630321          -9.4965702
## 4          -7.145127          -5.8878061
## 5          -5.146645          -3.5735432
## 6          -7.226433          -5.9076794
```

Now, we can consider how to compute the population variance of all individual frequencies. Rows 1 through 19 of EVdata describe the mutation of position 9 in Fluc-Bpe. The next few lines illustrate how to

compute a population variance for all simulated variants generated for position 9. This code constitutes an implementation of Equation 2.

```
var_data <- EVdata[1:19, 6]
n <- length(var_data)
F9_variance <- var(var_data)*(n-1)/n
F9_variance
```

```
## [1] 0.005432733
```

Thus, for simulated F9 mutants we have $\sigma_9^2 = 0.005432733$. To meaningfully compare the extent to which the position is susceptible to mutation, it is necessary to compute the variances for all positions represented in the dataset. Recall that a population variance of zero indicates that all of the data values in a given population are identical. That is, a population variance of exactly zero means that all mutations at a given position occur with the same frequency, which ought to correlate with very low conservation at said position. By contrast, a relatively high population variance is likely to describe a case where one or a few mutants occur with much higher frequencies compared to all other mutants.

We can introduce a nested loop that will allow us to construct a numeric vector that stores computed population variances, for all positions represented by the dataset.

```
k <- 1
variances <- c()
freq_vec <- c()
df_pos <- EVdata[k, 6]

while (k <= nrow(EVdata)){
  # The 'while' condition makes this useable for EVdata of arbitrary length.
  for (i in 1:19){
    # This condition will apply regardless of dataset because EVcouplings
    # effectively simulates 19 point mutations for each position. However
    # this code and its execution in command-line EVcouplings may be modified
    # to change this.
    freq_vec <- c(freq_vec, df_pos)
    if (i == 19){
      pop_var <- var(freq_vec)*(length(freq_vec)-1)/(length(freq_vec))
      variances <- c(variances, pop_var)
    }
    k <- k + 1
  }
}
```

```

    df_pos <- EVdata[k, 6]
  }
}

```

The resulting numeric vector, `variances`, stores the computed population variances for all positions represented by the dataset. However, we must be careful to note that our analysis is only useful for positions 9 through 123 along Fluc-Bpe. EVmutation did not report simulated data for positions 1 through 8 or for positions 124 through 128, likely because the quality of metadata associated with these positions was too low and therefore excluded from the downstream computations and from the CSV file. Consequently, the variances computed are applicable only for positions 9 through 123 inclusive. The set of variances for all positions represented in the dataset can be written as $\{\sigma_9^2, \sigma_{10}^2, \sigma_{11}^2, \dots, \sigma_{123}^2\}$. From the histogram generated by `hist(variances)` and presented in Figure 13 below, we can observe that on a relative scale most of these variances are low. This is consistent with the expectation that for most of the positions in the Fluc-Bpe sequence space, especially those corresponding to the transmembrane regions of the protein, mutation is not tolerated. The single-site substitution matrix for Fluc-Bpe presented in Section 3.3.2 (Figure 11) also supports this interpretation.

Histogram of tolerance [9:123]

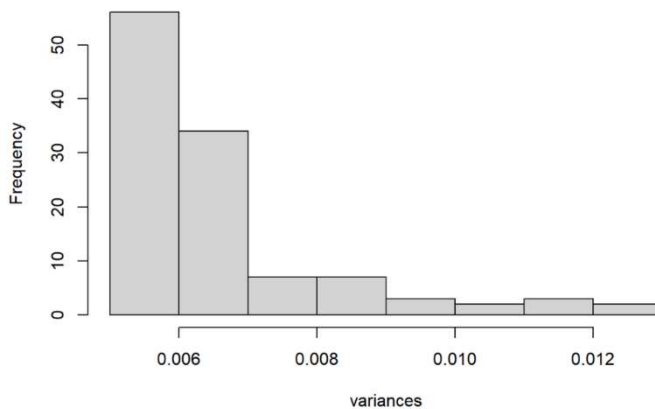


Figure 14. Frequency distribution of $\{\sigma_i^2\}$ calculated from Fluc-Bpe single-mutant variant frequencies in residues 9-123. For a given position in the Fluc-Bpe sequence, σ_i^2 quantifies a mutational tolerance.

The next lines of code in the script were written to handle the PDB file for Fluc-Bpe so that the information in the b-factor column could be overwritten. Note the use of packages `bio3d` [49] and `dplyr` [50].

```

library(bio3d)
library(dplyr)

```


To make the atomic data in the file more tractable for this purpose, we need only initialize the data describing the atoms as a dataframe as shown below.

```
pdb <- read.pdb("5nkq.pdb")
protein_df <- as.data.frame(pdb$atom)
```

```
head(protein_df, rm.alt = TRUE)
```

```
##   type eleno elety  alt resid chain resno insert      x      y      z o      b
## 1 ATOM     1     N <NA>  LEU   A     2  <NA> -16.982 -52.101 55.381 1 39.76
## 2 ATOM     2    CA <NA>  LEU   A     2  <NA> -15.959 -51.186 55.872 1 40.22
## 3 ATOM     3     C <NA>  LEU   A     2  <NA> -15.378 -50.359 54.728 1 43.47
## 4 ATOM     4     O <NA>  LEU   A     2  <NA> -15.150 -49.159 54.874 1 40.43
## 5 ATOM     5    CB <NA>  LEU   A     2  <NA> -14.850 -51.958 56.592 1 39.44
## 6 ATOM     6    CG <NA>  LEU   A     2  <NA> -13.727 -51.133 57.225 1 44.99
##   segid elesy charge
## 1  <NA>     N  <NA>
## 2  <NA>     C  <NA>
## 3  <NA>     C  <NA>
## 4  <NA>     O  <NA>
## 5  <NA>     C  <NA>
## 6  <NA>     C  <NA>
```

Thus, we observe that column 13 of this dataframe is the b-factor column. As noted earlier, PyMOL settings allow for a color-coded visualization of the protein in terms of the b-factor.

The next lines of code allow the variances to be written into a new text file, with each variance occupying its own line. Notice how this is done by coercing the numeric vector into a character vector.

```
variances_char <- as.character(variances)
writeLines(variances_char, con = "newBfactors.txt", sep = "\n", useBytes = FALSE)
```

Inspection of the newly created text file as shown in Figure 15 shows that the file consists of 114 lines, which correspond precisely to positions 9 through 123 in Fluc-Bpe, in the order listed.

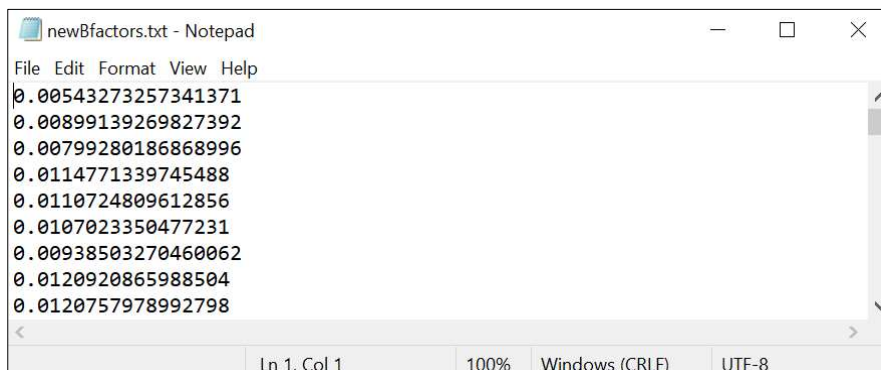


Figure 15. View of the newly created text file newBfactors.txt listing mutational tolerance scores, computed as population variances of Fluc-Bpe single-mutant variant frequencies.

At this point, it was recognized that the length of the protein is not equal to the number of atoms in the PDB file and it is therefore not trivial to overwrite the b-factor column with the variances we computed earlier. To work around this, an attempt was made to work with an additional, open-access Python script (named loadBfacts.py) already available from online repository Figshare [51] and documented in the PyMOL Wiki [52] which serves to replace b-factor values in a PDB file at the amino acid level.

The HTML file also documents this attempt, noting that it is necessary to run the outside Python script independently; it would be more practical here to employ exclusively R or exclusively Python. (However, use of the rpdb package from the CRAN repository may enable the use of PyMOL from an R session). Although steps were taken to translate the R script into Python and to integrate the operations of both scripts into one script, time constraints did not allow for this script to be fully written and presentable in the form of a Jupyter Notebook and made available through Jovian, as was done for the script presented in Chapter 2. A brief discussion of this progress is offered in Section 3.5.4.

From the documentation for the open-access Python script [52], it is a simple matter to call the requisite function according to the syntax given below:

```
loadBfacts mol, [startaa, [source, [visual Y/N]]]
```

Here, the following parameters apply: startaa allows us to indicate that our text file lists values starting with position 9, source refers to the name of the text file containing the replacement values, and source redraws the structure as the putty/sausage representation in PyMOL, having a default value of Y (yes).

When called without error, this function will replace b-factor values for a continuous amino acid sequence starting from the position provided in the parameter `startaa`.

3.5 Discussion

3.5.1 Reflection on the development and optimization of the program

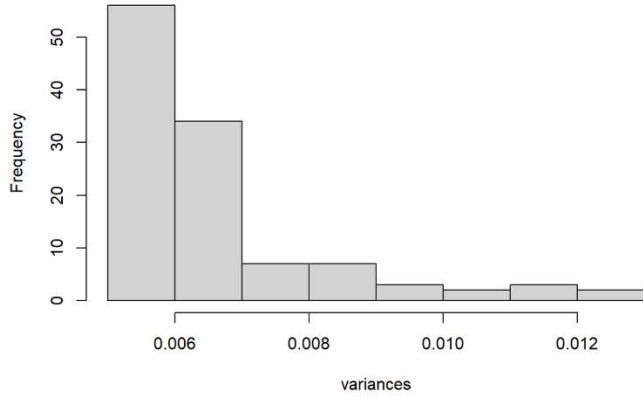
The Markdown formatting offers the script legibility and inline documentation in a manner comparable to that seen with the Jupyter notebook in the previous chapter. To arrive at the logic underpinning the main function of the script required forethought and planning with pencil and paper. As noted earlier, some progress was made in the development of a newer version of the script written entirely in Python. Use of the Jupyter notebook and Jovian integration could be useful should this version come to fruition. Also, since typical values of the b-factor are between 15 and 30 Å², it may be necessary to re-scale the mutational tolerance scores so that the spreads in mutational tolerance can be more easily differentiated according to color in PyMOL as intended.

Since EVcouplings plays a significant background role for this program, further development may explore the effects of changes to EVcouplings parameters, including the input Fluc sequence and selected protein database. More specifically, it may be worthwhile to query the known sequence of a Fluc ancestor (if this sequence is readily available or reconstructed) rather than an extant Fluc speculated to resemble such an ancestor. In addition, certain challenges were encountered in attempts to run web-based EVcouplings with the sequences for FEX proteins in *Arabidopsis thaliana* and in *Saccharomyces cerevisiae*. The outcomes of these attempts are described in Section 4.2.

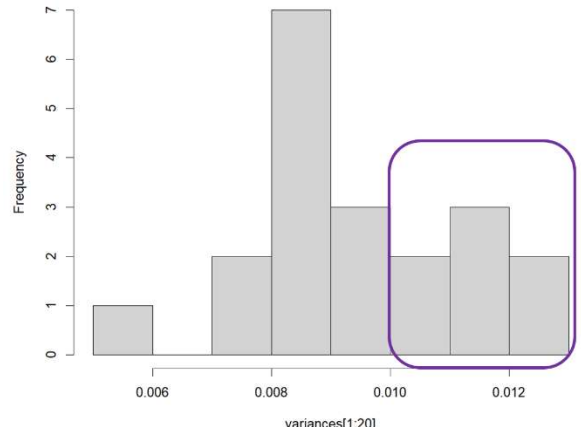
3.5.2 Analysis of mutational tolerance in Fluc transmembrane domains and internal loops

It is straightforward to slice the variances vector to generate histograms for a subset of residues. In addition to the contiguous sequence of positions 9-123, we generated histograms for five additional sets of residues in Fluc-Bpe. Four of these correspond to the four TM helices, denoted TM1 through TM4. The fifth set encompasses the internal loops of Fluc-Bpe, which here is defined to consist of residues 30-35 and residues 60-64. These histograms are presented in Figure 16 below. Note that the histogram shown in Figure 16a is identical to that in Figure 14.

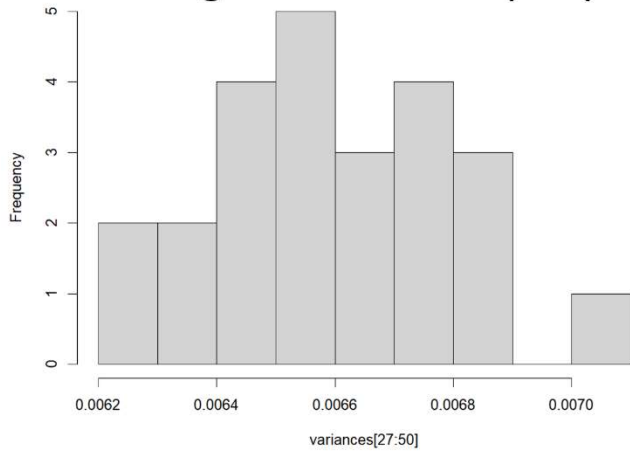
(a) Histogram of tolerance [9:123]



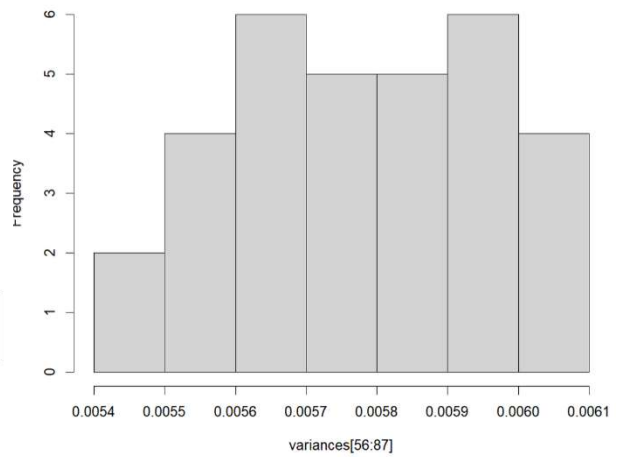
(b) Histogram of tolerance (TM1)



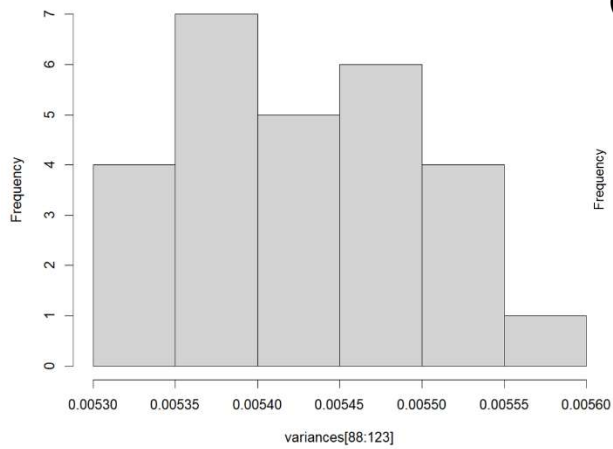
(c) Histogram of tolerance (TM2)



(d) Histogram of tolerance (TM3)



(e) Histogram of tolerance (TM4)



(f) Histogram of tolerance (internal LRs)

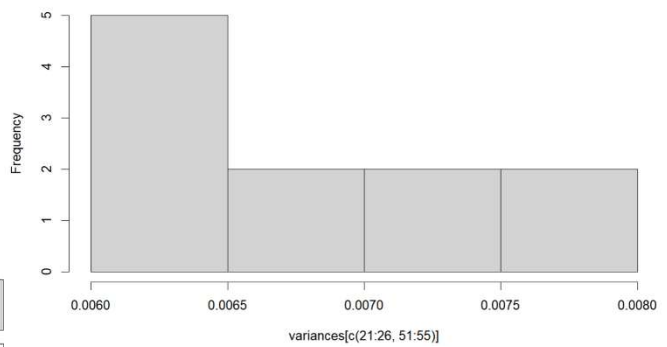


Figure 16. Histograms for Fluc-Bpe residues 9-123 and subsequences corresponding to four TM helices (TM1–TM4) and internal loop regions. Emphasis added in Panel (b) to indicate high tolerance values.

From the bins of these histograms, we can read the spread of mutational tolerance applicable to Fluc-Bpe as a whole and to domains of interest. Table 1 summarizes the spreads (i.e., ranges) of mutational tolerance computed, with each spread interval corresponding to the histograms in Figure 16 and to the relevant positions in the protein sequence.

Specified positions in Fluc-Bpe	Structural domain or feature	Spread of mutational tolerance
Residues 9-123	contiguous Fluc-Bpe sequence	$[5.0 \times 10^{-3}, 1.3 \times 10^{-2}]$
Residues 9-29	TM1	$[5.0 \times 10^{-3}, 1.3 \times 10^{-2}]$
Residues 36-59	TM2	$[6.2 \times 10^{-3}, 7.1 \times 10^{-3}]$
Residues 65-96	TM3	$[5.4 \times 10^{-3}, 6.1 \times 10^{-3}]$
Residues 97-123	TM4	$[5.3 \times 10^{-3}, 5.6 \times 10^{-3}]$
Residues 30-35 and Residues 60-64	internal loop regions	$[6.0 \times 10^{-3}, 8.0 \times 10^{-3}]$

Table 1. Summary of mutational tolerance spreads computed for structural features of Fluc-Bpe.

We found that the distribution of tolerance differed when selecting these subsequences. TM1 was found to contain several of the highest tolerance scores computed (greater than 1×10^{-2}), as emphasized in Figure 16b; even the mode of this distribution occurs in the interval between 8.0×10^{-3} and 9.0×10^{-3} . By comparison, the distributions computed for TM2 and TM3 are somewhat more uniform but have comparable spreads. TM4 was shown to have the narrowest spread and the greatest number of positions with tolerances lower than 5.6×10^{-3} . Finally, the internal loop regions are represented by only a few positions, yet high tolerance scores (up to 8.0×10^{-3}) were calculated here also.

The distribution for TM1 suggests that the unusually high scores are not due to outliers; exactly half of the positions in TM1 yielded scores greater than 7.0×10^{-3} . The result appears to contradict the general expectation that mutation in TM helices would not be tolerated, as might be interpreted from the single-site substitution matrix shown in Figure 10. Such intolerance is most conspicuous for R23, a position for which the EVcouplings job found 100% conservation among all Fluc homologs considered. However, a closer inspection of the matrix reveals that certain single substitutions in TM1, such as I10L and T17A, are predicted to be neutral or even advantageous. Mutation at several positions in the vicinity of R23 appear to be associated with a neutral or slightly damaging effect. Indeed, an alignment of Fluc-Bpe TM helices against the corresponding domains of representative FEX proteins shows that the TM1 primary sequence has greater variation than the sequences of other TM helices [23]; this alignment is shown in Figure 17. We therefore consider the high mutational tolerance scores computed for TM1 of Fluc-Bpe an indication that, following a duplication event which preceded the fused, monomeric structure of FEX, mutation in TM1

(and its paralogous domain) grew less deleterious to function to than did mutation in other TM domains. Further justification of this claim is provided in the next subsection.

3.5.3 Placing computed mutational tolerances in the context of Fluc topology evolution

We argue that the high mutational tolerance scores computed for TM1 of Fluc-Bpe may be reconciled with the greater sequence variation observed in FEX proteins, especially from the viewpoint that Fluc-Bpe is likely to resemble an evolutionary ancestor of FEX. Although a more precise and accurate elaboration of Fluc evolution must be informed by further studies, we tentatively propose that the three-step account of duplication followed by divergence and an eventual fusion event can be reformulated in greater detail. Our considerations are based in part on an alignment reported by McIlwain et al. and pictured in Figure 17.

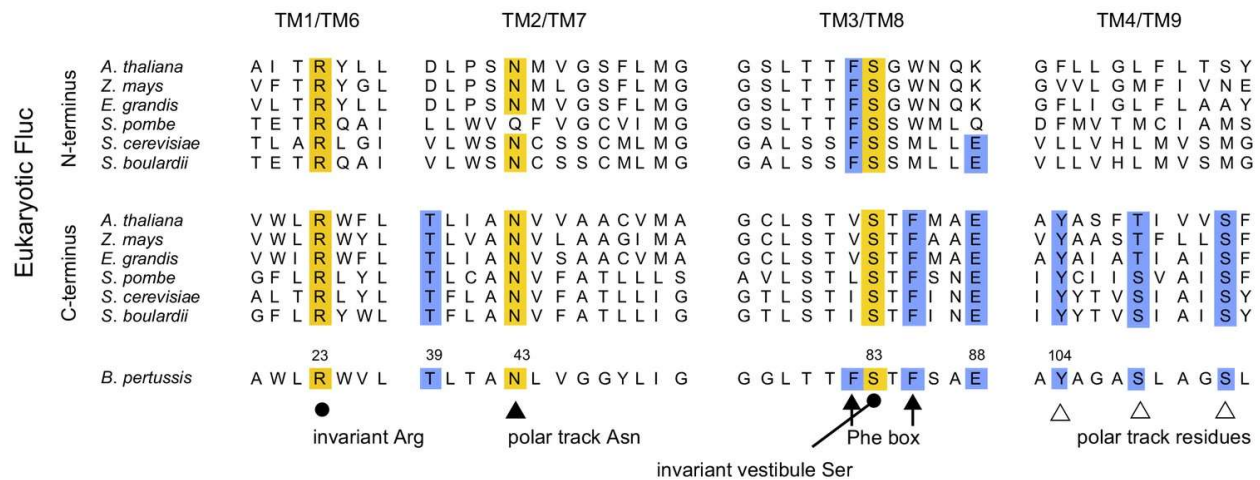


Figure 17. Sequence alignment of Fluc-Bpe with N-terminal, C-terminal domains of FEX proteins (TM helices only). Selected invariant residues are shown in yellow. Modified from McIlwain et al. (2021) [24].

First, we suggest that a gene duplication event in an ancestral, dual-topology Fluc led to intermediates in which the three-dimensional structure of TM domains was retained, but the selective pressure enforcing perfect symmetry and high sequence conservation was relaxed. Evolutionary constraints within TM2, TM3, and TM4 precluded the retention of mutations deleterious to fluoride transport; relatively fewer constraints in TM1 permitted sequence divergence of a greater extent in TM1 compared to divergence in the other TM helices. We can conceive of mutations in the TM domains as having dual effects: (1) preserving or damaging the requisite 3D fold of the domain and (2) enhancing or damaging the capacity for fluoride transport. Post-duplication selective pressures mediated a tradeoff between these effects according to their correlations with mutation at a given position. Controlling for sequence length, the greater sequence variation in TM1 seen in extant post-duplication Flucs suggests a larger space of tolerated mutations in some precursor TM1 sequence occurring in post-duplication intermediates. If we assume that the primary determinant of fitness has been the ability to counteract fluoride stress, increased genetic variation within

the tolerated mutational space for TM1 (and its paralogous domain) was likely retained in extant Fluc/FEX proteins, precisely because such variation conferred comparable levels of fitness in such intermediates. Mutation of a similar extent in other TM domains, it may be argued, could have had more deleterious outcomes; the presently observed variation in the sequences of TM2-TM4 of FEX proteins is less pronounced.

In addition, surviving FEX proteins have retained only one functional pore. Note the difference in conserved Phe residues in the phenylalanine box indicated in Figure 17. Whereas the fixed-topology Flucs experienced a partitioning of the orientation assumed by Fluc monomer in the membrane, a reduced selective pressure in post-duplication FEX precursors may also have led to partitioning of the Phe box motif that rendered vestigial the capacity for fluoride transport through one of the two pores. It seems plausible that this vestigiality emerged from genetic drift following the putative gene fusion event which allowed FEX to be monomeric (this monomericity being retained in extant FEX proteins), although this remains unclear. Nevertheless, some conservation of the Phe box motif persists: a serine at position 83 in Fluc-Bpe was found to be invariant among all Fluc/FEX proteins examined by McIlwain et al. [24].

Although our findings based on evolutionary couplings in Fluc-Bpe are ampliative and add detail to a provisional account of the Fluc evolutionary trajectory, future work could investigate residue coevolution more comprehensively and involve attempts to characterize couplings in FEX proteins. More specifically, it may be worth exploring how evolutionary couplings have changed over time and especially in the transition of Flucs to eukaryotic organisms. Further relevant considerations are offered in the next chapter.

3.5.4 Toward a re-implementation of program in Python

One additional reason for the re-implementation of this script in Python is that R is generally considered less intuitive to newcomers than Python. Thus far, we have adapted the code responsible for iterating through the input dataset to compute mutational tolerances and store the results in a Python list. The suggested Python code for this is presented below.

```
import pandas as pd
import numpy as np

EVdata = pd.read_csv("5NKQ_single_mutant_matrix.csv")

k = 0
variances = []
freq_vec = []
df_pos = EVdata.iloc[k, 5]
```

```

while k < len(EVdata):
    for i in range(1, 20):
        freq_vec.append(df_pos)
        if i == 19:
            pop_var = np.var(freq_vec) * (len(freq_vec) - 1) / len(freq_vec)
            variances.append(pop_var)
        k += 1
        df_pos = EVdata.iloc[k, 5]

```

Note that we import the packages `numpy` and `pandas` to afford a functionality similar to that afforded by dataframes in R. The variable `EVdata` in this new script is a Pandas dataframe. Since indexing in R begins with 1 and indexing in Python begins with 0, we have used the `iloc` method to select values in a manner equivalent to the 0-based indexing in R.

3.5.5 Suggestions for further development

A desirable prospect in the further elaboration of this script is to make the program more expansive. For example, it would be informative for such a version of the program to parse through the observed relative variant frequencies from a competitive growth assay, as measured through next-generation sequencing, to create mutational tolerance histograms for potentially thousands of different Fluc variants, all growing under the same lab-controlled media and conditions. The frequency distribution for the variation in mutational tolerance could in turn be depicted in other modalities available to descriptive statistics, offering new and different ways of visualizing mutational tolerance besides single-site substitution matrices. We have noted that such matrices are typically considered measures of the fitness that different protein variants can afford. However, through experimental methodologies capable of tracking mutation in descendant bacterial strains, modeling the distribution of experimentally observed mutational tolerance among variants could help assess the extent to which positions in the primary sequence appear *susceptible* to mutation, rather than merely tolerant of mutation.

An ancillary function of the current version of the program (the R script and associated Markdown formatting) is the use of a relatively new package in the CRAN repository, `NGLViewer` [53], to demonstrate how it is possible to visualize PDB files.

```

library(NGLViewer)

NGLViewer("5NKQ") %>%
addRepresentation("cartoon")

```


These lines will initialize visualization of a PDB structure either from the PDB directly (given the 4-character code) or from a locally stored PDB file, as an HTML device or widget distinct from the main R environment or console.

4 Future directions in the investigation of Fluc variant fitness

We conclude our disquisition with material relevant to further investigation. This section begins with a rationale for the modeling of epistatic interactions in determining evolutionary couplings, an approach which underpins EVcouplings and has improved methods of protein structure prediction. Next, we summarize obstacles faced in our attempts to use EVcouplings on selected FEX sequences and give advice about how to overcome these challenges. Then, we offer a sketch of related experimental work currently in progress. Lastly, we present a description of miscellaneous concepts and developments that may be useful in representing and testing possible microevolutionary dynamics of the Flucs. It is hoped that our investigation and the ensuing discussion will serve as a springboard for future studies that will enhance our understanding of the Fluc evolutionary trajectory.

4.1 On the advantages of modeling epistatic interactions

The breakthrough in protein structural prediction based on residue coevolution materialized from modeling the context-dependent effects of mutation. Previous methods tended to treat such mutation as having an effect mostly confined to a certain position along the protein: the position of the mutation. These older methods often relied on hidden Markov models (HMMs) that quantified a position's preference for a particular amino acid, but not accounting for interactions between residues. Granted, HMMs are yet very useful for homology searches and sequence alignments; the HMM-based software HMMER is a dependency of EVcouplings. However, the innovation introduced by newer algorithms for evolutionary couplings and their downstream effects on structure took form in models based on Markov random fields (MRFs). The pairwise interactions inherent to these fields capture epistasis and are essentially in correspondence with pairs of co-evolving residues within a protein of interest. Residue coevolution analyses have improved remarkably as a result of MRF-based models [54].

4.2 On challenges encountered when applying EVcouplings to Fluc variants other than Fluc-Bpe

Initial attempts to analyze two eukaryotic Fluc homologs using the web-based EVcouplings tool, in a manner similar to the analysis we report for Fluc-Bpe, proved unsuccessful due to lower quality results obtained from passing as inputs the sequences for FEX proteins found in *A. thaliana* and *S. cerevisiae*. These sequences were queried using the Uniprot identifiers Q8RYE2 and Q08913, respectively. Results for these queries are shown in Figure 18 and Figure 19.

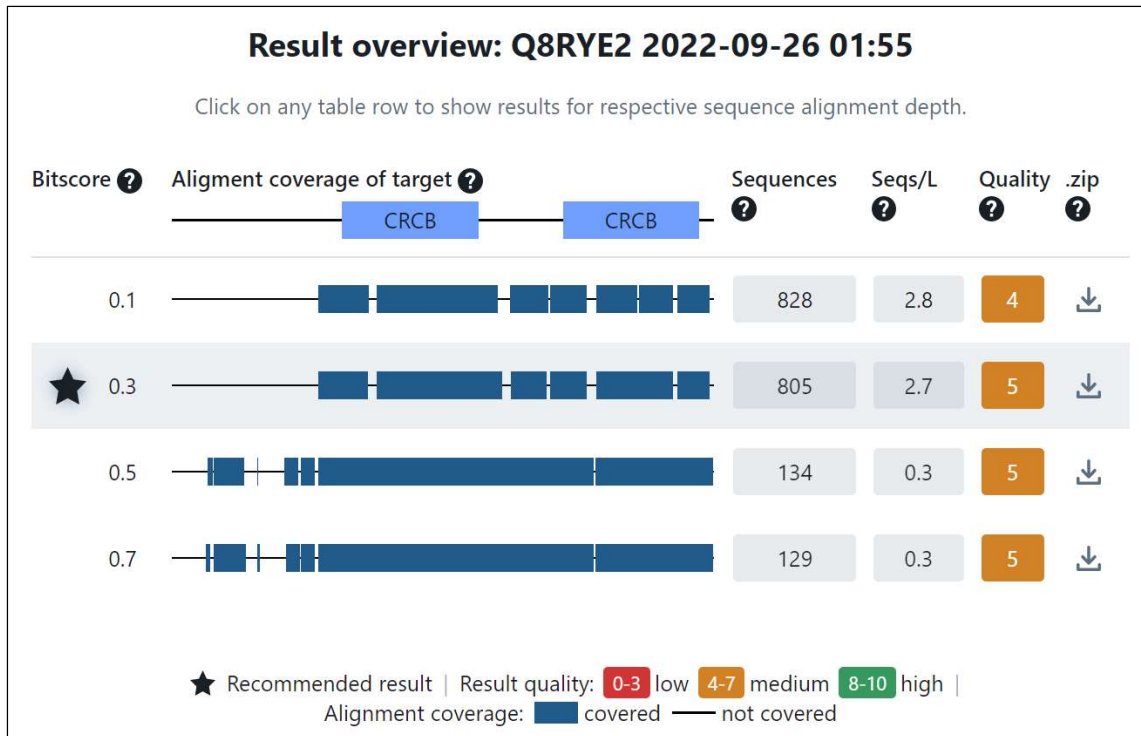


Figure 18. Screenshot of result overview for EVcouplings job on FEX found in *Arabidopsis thaliana*. (Uniprot: Q8RYE2)

Alignment coverage in both of these results is not as extensive as the result obtained with Fluc-Bpe. Also, for each bitscore, the number of homologous sequences detected by the homology search is significantly lower.

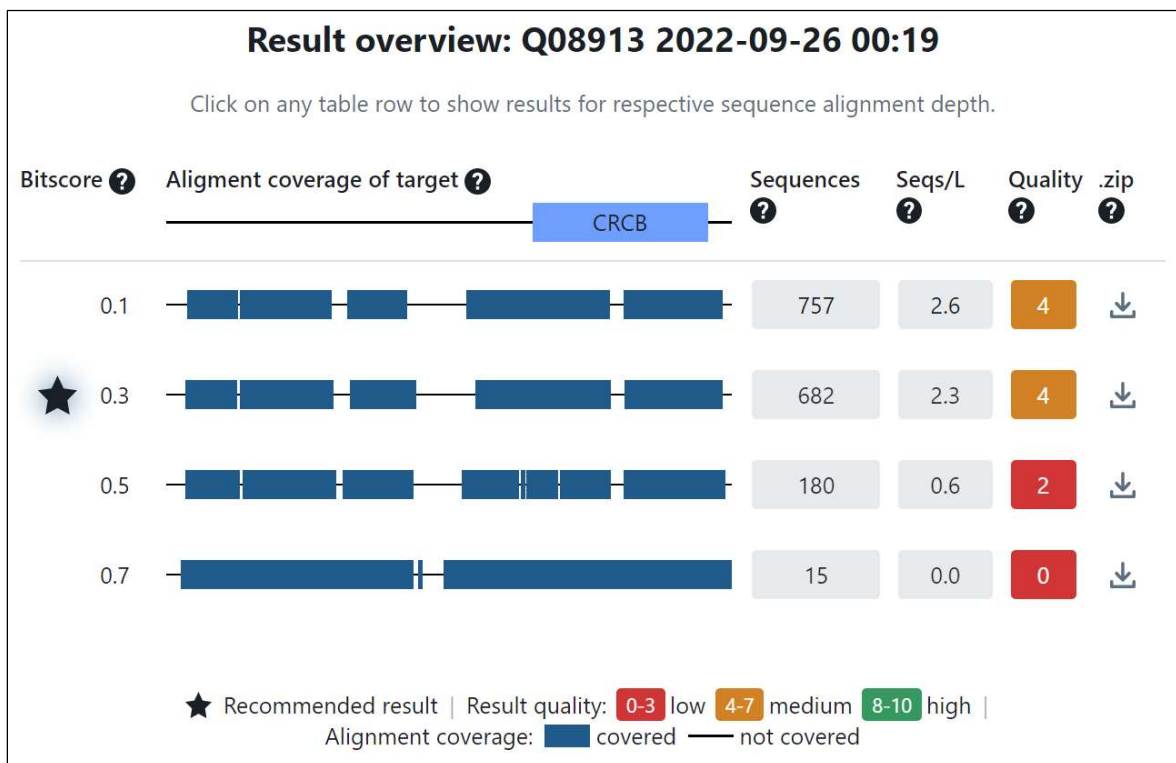


Figure 19. Screenshot of result overview for EVcouplings job on FEX found in *Saccharomyces cerevisiae*. (Uniprot: Q08913)

To troubleshoot this, it may be beneficial to customize the parameters in EVcouplings and to use a more curated database consisting primarily of FEX sequences, rather than one of the larger Uniprot sequence databases. That the genotype for FEX represents the fused form of two formerly distinct paralogs, each of which can resemble Fluc-encoding gene to a highly variable extent, may not be entirely appropriate for the automated alignment and homology search algorithms as they are currently implemented in web-based EVcouplings. It could even be of benefit to independently perform an alignment treating the N-terminal and C-terminal domains of FEX as separate proteins; the resulting alignment should then be formatted appropriately for the homology search function of EVcouplings. At present, such custom settings may only be available from the command-line version of EVcouplings. An attempt to install EVcouplings so that this version could be utilized was only partially fulfilled: it was unclear from existing documentation how to install a particular external dependency named `p1mc`, the tool used by the pipeline to infer evolutionary couplings from MRFs.

These challenges notwithstanding, a point of reference for future work is a study that examines these two FEX proteins, which have a sequence identity of only 25% [55]; researchers provide evidence from alignments, mutagenesis experiments, and structural modeling that FEX channels each contain one functional pore and one vestigial pore [55].

4.3 Toward a deep mutational scanning methodology for investigating Fluc variant fitness

A current project in the Stockbridge group seeks to probe Fluc variant fitness using an engineered plasmid system containing variants of *Ec2*, the Fluc-encoding gene found in *E. coli* virulence plasmid. This project seeks to evaluate relative differences in fitness between Fluc variants through a specialized growth assay that challenges microbial growth with a fixed concentration of NaF. Two variants of interest are those in which one of the two Phe residues in the phenylalanine box motif has been mutated; these are represented in two types of plasmids. The plasmid system also includes a vector made to contain two copies of *Ec2* in tandem, in what is effectively a mimic of an immediate post-duplication genotype. Although cloning procedures to engineer more serviceable versions of this system are in progress, it is hoped that growth assays of *E. coli* transformants will reveal noticeable fitness changes due to gene duplication and single-site substitutions separately. A schematic of this small-scale growth assay is shown in Figure 20.

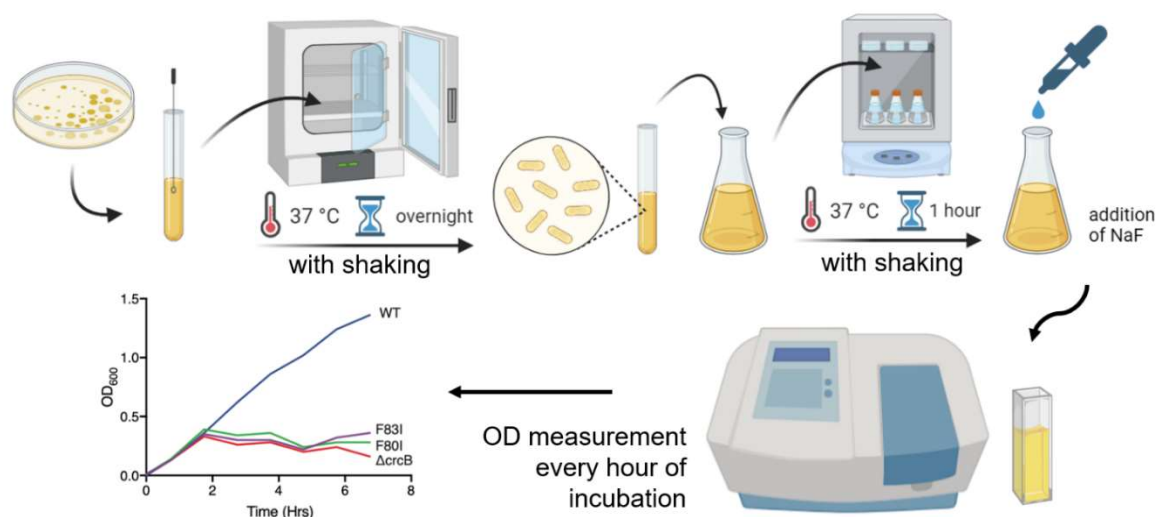


Figure 20. Schematic of small-scale growth assay to test relative fitness in *Ec2*-transformed *E. coli*.

This work is in turn intended to prefigure a large-scale DMS experiment (refer to Section 3.1) in which saturation mutagenesis or a comparable technique is used to create a library of Fluc single-mutant variants, inserted into an appropriate choice of vector to transform into *E. coli* just as in the small-scale growth assay. Since the plasmid system will have made it possible for this vector to contain tandem copies of *Ec2*, these *Ec2* variants will also be made to vary according to whether they occur in a single copy of the gene, or as part of a tandem duplicated genotype. As part of the mutational scan, a menagerie of *E. coli* transformed with this library will be subjected to a selection condition (again, a fixed concentration of NaF) in a competitive growth assay. Next-generation amplicon sequencing will then serve to quantify the abundances of *Ec2* variants after selection; this data will be the basis for empirically determined single-site substitution matrices of Fluc-*Ec2*.

4.4 Prospects for resolving the Fluc evolutionary trajectory in greater detail

In the first chapter, we introduced the notion of a fitness landscape to represent evolutionary fitness as a function of genotype or phenotype. Informed by prior studies as well as our own considerations of conservation and residue coevolution, we offered in Chapter 3 a plausible narrative of the Fluc evolutionary pathway more detailed than the nutshell account summarized as the three stages of gene duplication, divergence, and fusion. Our overarching goal has been to elaborate the phenomena and sequence of events which allowed all three topologies in the Fluc family to remain extant into the present day; that is, we aim to meaningfully discretize steps in the Fluc evolutionary trajectory at a level more discerning than the three aforementioned stages as presently stated.

Also in Chapter 3, we reported an *in silico* sequence-function landscape (the single-site substitution matrix) of Fluc-Bpe, a Fluc whose topology represents an evolutionary antecedent to the fixed-topology and fused monomeric phenotypes. The accurate simulation of amino acid substitutions for an ancestral protein can offer a basis for the reconstruction of plausible phylogenetic representations. This task is not at all trivial, even while protocols for experimental approaches such as DMS may be sound. We believe it has been helpful to model amino-acid substitutions in Fluc using the EVcouplings framework, yet another approach recently developed by R. Sloutsky and K.M. Naegle also seems promising; this methodology consists of an evolutionary reconstruction algorithm that iterates over different but related input protein sequences to quantify the variability associated with these inputs and use this information to optimize the confidence and accuracy of phylogeny inferences [56]. Termed ASPEN (Accuracy through Subsampling of Protein EvolutionN), the methodology is asserted to be an improvement on prior methods which were computationally intensive because they reconstructed many possible versions of sequence divergence, rather than those that are most plausible [56]. In a press release from the University of Massachusetts at Amherst [57], Naegle is quoted with the following analogy:

“If one asks 1,000 people to predict what route a driver took on a past multi-stage trip, pieces shared most across all 1,000 answers are most likely to be true. One can never know the real path – in this case, evolution – but if no predictions agree over 1,000 answers, one has little confidence in their accuracy. But if all 1,000 agree on some stages, confidence is higher that a new model based on it will be true. It still may not be wholly accurate, but it’s probably closer than any single route suggestion,” [Naegle] says.

Absent from the reporting of this new methodology, however, is a definition of fitness. On this subject, we first look toward a 2014 study by Firnberg et al. in which fitness is quantified as a *distribution of fitness effects* (DFE), for the *E. coli* beta-lactamase gene, *TEM-1*, and corresponding protein TEM-1 [58]. The

fitness of a gene is equated with providing some kind of optimum – a “phenotypic signature” – while the fitness of a protein is similarly regarded as “the suitability of a protein to provide a particular function” [58]. Conceptual problems with this definition notwithstanding, the study documents such methods as growth competition experiments, mRNA stability predictions, and a quantification of mutational tolerance as the number of amino acids that are especially likely to occur at a given position [58]. This quantification is derived from the Shannon entropy applicable to the distribution of fitness values for a particular set of mutants, at some position i [58]. These fitness values follow from an asserted proportionality $w \propto v_t$, where w is a raw fitness and v_t is the total catalytic activity of this enzyme in the cell. The total catalytic activity is in turn given by $v_t = v_{sp}P$, where v_{sp} denotes the specific activity of the enzyme and P is the abundance of the enzyme [58]. Researchers concluded that the deleterious effects of mutation are primarily due to a decrease in the specific activity of beta-lactamase rather than a decrease in the abundance of the protein [58]. It would be intriguing, for instance, to have an experiment involving Fluc variant libraries quantifying a mean protein abundance $\langle P \rangle$ from Western blots and a mean total current $\langle v_t \rangle$ so that decreases in Fluc fitness, following Firnberg et al., may be compared to changes in either $\langle v_t \rangle$ or $\langle P \rangle$, separately. This would illustrate the action of mutations which increase stability but undermine fitness, and of mutations which decrease stability but appear to confer adaptive benefit.

To pave the way for future studies, we also find it worthwhile to also examine the concepts of an “evolutionary trajectory” and the fitness landscape metaphor more critically. This also entails a reflection on aspects of evolutionary theory and the meaning of fitness itself [59]. To hearken back to our introduction, we acknowledge that fitness, when equated with the growth rate of an individual microorganism, is the outcome of a delicate interplay between genotype, phenotype, and environment. One attractive portrayal of these interactions as they play out for the *lac* operon and pathway in *E. coli* [60] is offered by Perfeito et al. and reproduced in Figure 21. The operon and some environmental concentration of IPTG modulate a positive feedback loop in which protein production promotes activity and vice versa. Downstream, there is another feedback loop in which, for example, the cost of increased protein production comes at the expense of growth rate, and growth rate itself has an inhibitory effect on protein production and activity.

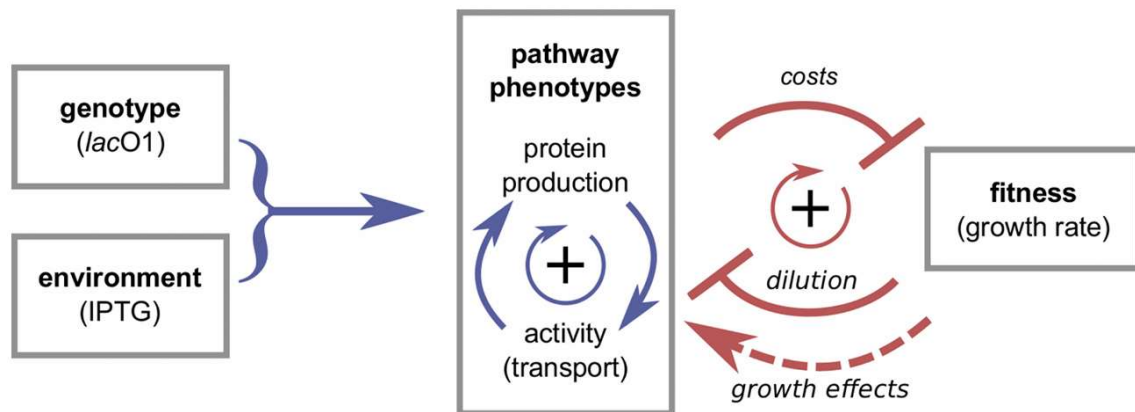


Figure 21. Influence of genotype and environment on phenotypes and fitness in the *lac* expression pathway. From L. Perfeito et al. (2011) [60].

We highlight this point of view to contend that a precise and *fine-grained* description of fitness at the level of individual phenotypes must account for nonlinearities in the relationship between the biochemical intricacies of a phenotype with organism fitness. Furthermore, we argue that this perspective is most useful at the fine-grained level; there are other levels of analysis in which other factors must be considered.

Following relevant contributions from J.F. Wilkins and P. Godfrey-Smith, we ascribe to distinctions in the analysis of fitness which set apart the fine-grained view of adaptation from a coarse-grained view, as well as an intermediate level of grain [61]; this has been described as an act of “zooming in or out” of a fitness landscape. As regards the preceding discussion of the *lac* pathway, we can conceive of a fitness landscape that resembles Figure 22a, in the fine-grained perspective. The position with respect to the vertical axis signifies fitness. One important realization that can be gleaned from this perspective is that natural selection is not the only process by which adaptive benefit can be achieved. For example, horizontal gene transfer can confer adaptive benefit at the level of an individual organism; the bacterial transformations involving the engineered plasmid system described in Section 4.3 are tantamount to such gene transfer mechanisms. By contrast, constraints in the mechanisms of inheritance at the molecular level (including epistasis) could conceivably frustrate adaptation towards some local optimum of fitness. Other relevant processes occur at the level of a population: genetic drift owing to a bottleneck event can lead to an overrepresentation in some genetic variants which could not be achieved by natural selection alone. In other words, it should not be surprising to see that at this level of grain, genetic variation due to processes other than natural selection play an outside role in the distribution of extant species or populations on the adaptive landscape. Natural selection by itself would serve to drive evolutionary change *toward* peaks in a fitness landscape, never *away* from such peaks.

Natural selection is arguably most salient in the intermediate-grained perspective of the fitness landscape, wherein continuous selective pressures (in an unchanging environment) drive adaptation toward a local maximum of fitness, as illustrated in Figure 22b. The net direction of movement on the landscape, for a given population and longer timescales, is toward a fitness peak. In this context, it would even be appropriate to model evolutionary adaptation with an account that appeals *exclusively* to natural selection; differences in adaptation due to mechanisms at the fine-grained level would be too minute to register at this broader level of analysis. Natural selection is the primary means by which evolutionary change concentrates populations around a fitness peak, over an evolutionarily significant period of time.

Finally, with the coarse-grained view depicted in Figure 22c, research can most readily address questions about how organisms of extant species and populations are distributed in some much more expansive space of possible (even imaginary) organisms that would be capable of successful survival and reproduction in the environment represented by the landscape. Points in the landscape viewed from this perspective represent the surviving species or populations relatively suited to this environment. These points occur on local optima and it is possible to have fitness peaks that are entirely unpopulated; due to history, some imaginary and exceptionally fit species may not currently exist (or have ever existed) in this environment.

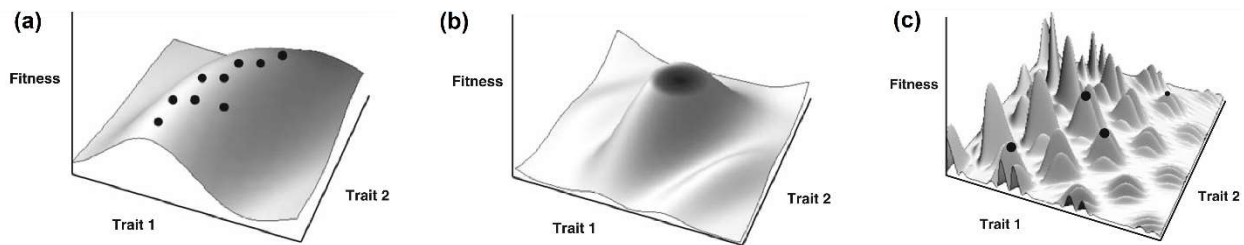


Figure 22. Fine-grained (a), intermediate-grained (b), and coarse-grained (c) perspectives of a fitness landscape. Points in (a) denote individual organisms, points in (b) are individuals concentrated around a local fitness peak, and points in (c) represent extant species or populations. Modified from J.F. Wilkins and P. Godfrey-Smith (2008) [61].

These caveats to the fitness landscape metaphor call attention to the limitations of adaptationism and the metaphor as it is generally presented: the fitness landscape implies that the fitness of a particular phenotype results primarily from an unchanging environment. From our mention of gene expression via the *lac* pathway, however, it can be appreciated that nonlinearities in the sequence-function-fitness relationship persist even in constant environmental conditions such as a fixed concentration of IPTG. A similar level of nuance ought to be recognized in experimental investigations of Fluc survival in the presence of fluoride. Moreover, one should exercise discretion about whether or not independent gene duplication, paralog divergence, and gene fusions in the history of the Flucs were retained due to some fitness advantage.

Caution is also advisable for other events causing genetic variation, although it has been claimed that the rate at which mutation is adaptive is generally lower than the rate at which gene duplication is adaptive [7]. That being said, we admit that our revised narrative of the Fluc evolutionary pathway (like definitions of fitness offered by Firnberg et al. and many others) follows the adaptationist paradigm closely. Criticism against what has been regarded as the uncritical attribution of extant traits to natural selection notably took form in an influential 1979 paper by S.J. Gould and R.C. Lewontin introducing the term “spandrels” to refer to phenotypic features that are best classified not as traits *sui generis* but as byproducts of the adaptation of other traits [62].

Issues with exaggerating the role of adaptation and natural selection may be avoided by defining traits in explicitly phylogenetic terms, as previous phylogenetic studies of Flucs [20] and the ASPEN methodology have exemplified, which would allow adaptation to be distinguished from shared evolutionary histories. Novel phylogenetic investigations should account for divergence in the genomic sequences and primary structures of Flucs subsequent to duplication so as not to overestimate evolutionary distances. Mutational scanning and other functional characterization of the Fluc family can also clarify constraints that could inform further phylogenetic analyses, as can reconstruction of the sequence and function of Fluc ancestors. Examples of structural and functional changes in Flucs which beckon further inquiry include the transition from the dual-topology phenotype to the fixed-topology phenotype, the gene fusion event and emergence of an additional TM helix in the FEX proteins, escapes from paralog interference, and modifications to helix-helix interactions subsequent to gene duplication or fusion.

Appendix

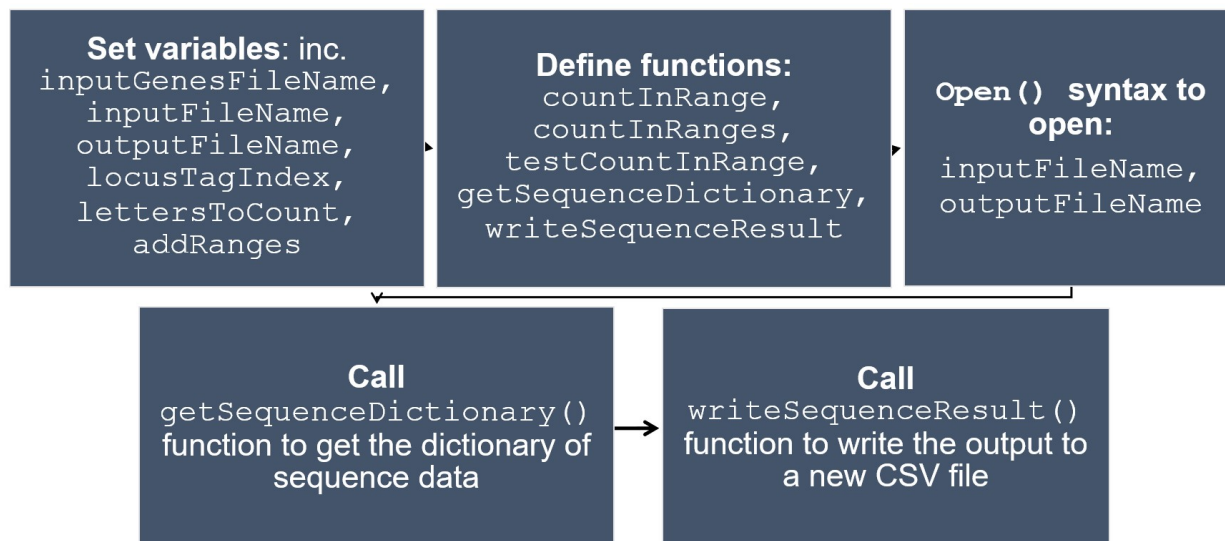


Figure A1. Flowchart of script to evaluate charge bias in Fluc sequences as reported in Chapter 2.

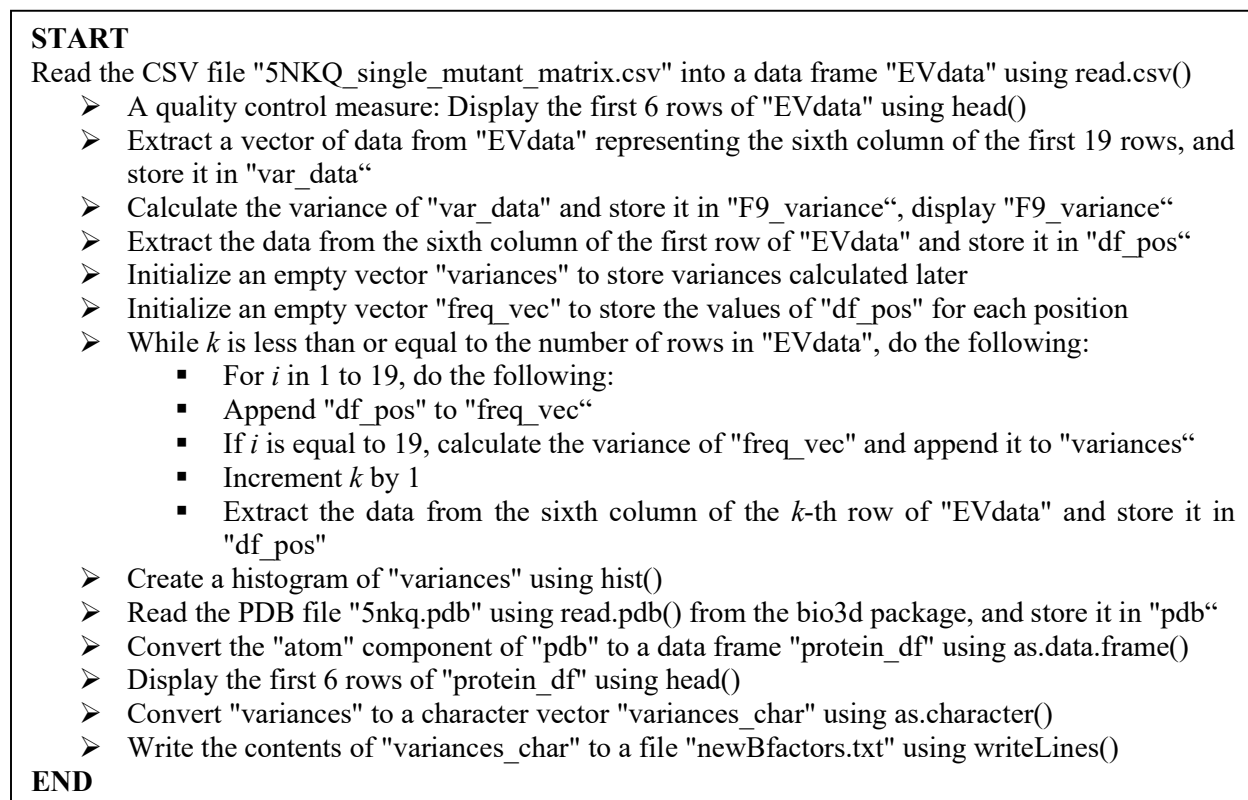


Figure A2. Outline of script to compute and handle mutational tolerances as reported in Chapter 3.

Bibliography

- [1] Y. Charles, V. Horn and P. Gollnick, "Physiological studies of tryptophan transport and tryptophanase operon induction in *Escherichia coli*," *Journal of bacteriology*, vol. 173, no. 19, pp. 6009-6017, 1991.
- [2] R. Chalmers, S. L. Percival, P. R. Hunter, J. Sellwood and P. Wyn-Jones, "Chapter Six - *Escherichia coli*," in *Microbiology of waterborne diseases*, San Diego, Elsevier Academic Press, 2004, pp. 89-117.
- [3] A. Gaber and M. Pavšič, "Modeling and Structure Determination of Homo-Oligomeric Proteins: An Overview of Challenges and Current Approaches," *International journal of molecular sciences*, vol. 22, no. 9081, 2021.
- [4] J.-H. Han, S. Batey, A. A. Nickson, S. A. Teichmann and J. Clarke, "The folding and evolution of multidomain proteins," *Nature Reviews Molecular Cell Biology*, vol. 8, no. 4, pp. 319-330, 2007.
- [5] X. Zhou, J. Hu, Z. Chengxin, G. Zhang and Y. Zhang, "Assembling multidomain protein structures through analogous global structural alignments," *Proceedings of the National Academy of Sciences*, vol. 116, no. 32, pp. 15930-15938, 2019.
- [6] "Cell Membranes," Scitable by Nature Education, 2014. [Online]. Available: <https://www.nature.com/scitable/topicpage/cell-membranes-14052567/>. [Accessed 16 May 2023].
- [7] C. B. Macdonald, "Complexity in the membrane: The Fluc family of fluoride channels and small multidrug resistance family of transporters as models for understanding membrane protein structural and functional evolution." Doctoral thesis. University of Michigan, Ann Arbor, 2021.
- [8] G. E. Tusnády, Z. Dosztányi and I. Simon, "PDB_TM: selection and membrane localization of transmembrane proteins in the protein data bank," *Nucleic acids research*, vol. 33, pp. D275-D278, 2005.
- [9] L. H. Weinstein and A. Davison, *Fluorides in the Environment: Effects on Plants and Animals*, Cambridge: CABI Publishing, 2004.
- [10] U.S. Department of Health and Human Services Federal Panel on Community Water Fluoridation, "US Public Health Service recommendation for fluoride concentration in drinking water for the prevention of dental caries," *Public Health Reports*, vol. 130, no. 4, p. 318-331, 2015.
- [11] C. Ji, R. B. Stockbridge and C. Miller, "Bacterial fluoride resistance, Fluc channels, and the weak acid accumulation effect," *Journal of General Physiology*, vol. 144, no. 3, pp. 257-261, 2014.
- [12] X. Zhang, X. Gao, C. Li, X. Luo and Y. Wang, "Fluoride contributes to the shaping of microbial community in high fluoride groundwater in Qiji County, Yuncheng City, China," *Scientific reports*, vol. 9, no. 1, pp. 1-10, 2019.

- [13] E. Adamek, K. Pawłowska-Góral and K. Bober, "In vitro and in vivo effects of fluoride ions on enzyme activity," *Annales Academiae Medicae Stetinensis*, vol. 51, pp. 69-85, 2005.
- [14] L. S. Newman and K. G. Bird, "Toxic inhalational lung injury," in *Clinical Respiratory Medicine*, 4th ed., Philadelphia, Elsevier Saunders, 2012, pp. 682-689.
- [15] J. Baker, N. Sudarsan, Z. Weinberg, A. Roth, R. B. Stockbridge and R. R. Breaker, "Widespread genetic switches and toxicity resistance proteins for fluoride," *Science*, vol. 335, no. 6065, pp. 233-235, 2012.
- [16] R. B. Stockbridge, L. Kolmakova-Partensky, T. Shane, A. Koide, S. Koide, C. Miller and S. Newstead, "Crystal structures of a double-barrelled fluoride ion channel," *Nature*, vol. 525, no. 7570, pp. 548-551, 2015.
- [17] B. C. McIlwain, M. T. Ruprecht and R. B. Stockbridge, "Membrane exporters of fluoride ion," *Annual review of biochemistry*, vol. 90, pp. 559-579, 2021.
- [18] R. B. Stockbridge, A. Koide, S. Koide and C. Miller, "Proof of dual-topology architecture of Fluc F-channels with monobody blockers," *Nature communications*, vol. 5, no. 1, p. 5120, 2014.
- [19] N. B. Last, S. Sun, M. C. Pham and C. Miller, "Molecular determinants of permeation in a fluoride-specific ion channel," *Elife*, vol. 6, p. e31259, 2017.
- [20] C. B. Macdonald and R. B. Stockbridge, "A topologically diverse family of fluoride channels," *Current opinion in structural biology*, vol. 45, pp. 142-149, 2017.
- [21] N. B. Last, L. Kolmakova-Partensky, T. Shane and C. Miller, "Mechanistic signs of double-barreled structure in a fluoride ion channel," *Elife*, vol. 5, p. e18767, 2016.
- [22] R. B. Stockbridge, J. L. Robertson, L. Kolmakova-Partensky and C. Miller, "A family of fluoride-specific ion channels with dual-topology architecture," *Elife*, vol. 2, p. e01084, 2013.
- [23] M. Cametti and K. Rissanen, "Highlights on contemporary recognition and sensing of fluoride anion in solution and in the solid state," *Chemical Society Reviews*, vol. 42, no. 5, pp. 2016-2038, 2013.
- [24] B. C. McIlwain, R. Gundepudi, B. B. Koff and R. B. Stockbridge, "The fluoride permeation pathway and anion recognition in Fluc family fluoride channels," *Elife*, vol. 10, p. e69482, 2021.
- [25] B. C. McIlwain, K. Martin, E. A. Hayter and R. B. Stockbridge, "An interfacial sodium ion is an essential structural feature of Fluc family fluoride channels," *Journal of molecular biology*, vol. 432, no. 4, pp. 1098-1108, 2020.
- [26] M. Ernst, E. A. Orabi, R. B. Stockbridge, J. D. Faraldo-Gómez and J. L. Robertson, "Dimerization mechanism of an inverted-topology ion channel in membranes," *bioRxiv*, 2023.
- [27] R. B. Stockbridge, H.-H. Lim, R. Otten, C. Williams, T. Shane, Z. Weinberg and C. Miller, "Fluoride resistance and transport by riboswitch-controlled CLC antiporters," *Proceedings of the National Academy of Sciences*, vol. 109, no. 38, pp. 15289-15294, 2012.

- [28] A. B. Reams and J. R. Roth, "Mechanisms of gene duplication and amplification," *Cold Spring Harbor perspectives in biology*, vol. 7, no. 2, p. a016592, 2015.
- [29] E. Kuzmin, J. S. Taylor and C. Boone, "Retention of duplicated genes in evolution," *Trends in Genetics*, vol. 38, no. 1, pp. 59-72, 2022.
- [30] E. Kaltenecker and D. Ober, "Paralogous interference affects the dynamics after gene duplication," *Trends in plant science*, vol. 20, no. 12, pp. 814-821, 2015.
- [31] L. Gerasimavicius, B. J. Livesey and J. A. Marsh, "Loss-of-function, gain-of-function and dominant-negative mutations have profoundly different effects on protein structure," *Nature communications*, vol. 13, no. 1, p. 3895, 2022.
- [32] L. R. Forrest, "Structural symmetry in membrane proteins," *Annual review of biophysics*, vol. 44, pp. 311-337, 2015.
- [33] I. Anishchenko, S. Ovchinnikov, H. Kamisetty and D. Baker, "Origins of coevolution between residues distant in protein 3D structures," *Proceedings of the National Academy of Sciences*, vol. 114, no. 34, pp. 9122-9127, 2017.
- [34] S. Ovchinnikov, H. Park, N. Varghese, P.-S. Huang, G. A. Pavlopoulos, D. E. Kim, H. Kamisetty, N. C. Kyrpides and D. Baker, "Protein structure determination using metagenome sequence data," *Science*, vol. 355, no. 6322, pp. 294-298, 2017.
- [35] T. A. Hopf, J. B. Ingraham, F. J. Poelwijk, C. P. Schärfe, M. Springer, C. Sander and D. S. Marks, "Mutation effects predicted from sequence co-variation," *Nature biotechnology*, vol. 35, no. 2, pp. 128-135, 2017.
- [36] T. A. Hopf, A. G. Green, B. Schubert, S. Mersmann, C. P. Schärfe, J. B. Ingraham, A. Toth-Petroczy, K. Brock, A. J. Riesselman, P. Palmedo, C. Kang, R. Sheridan, E. J. Draizen, C. Dallago, C. Sander and D. S. Marks, "The EVcouplings Python framework for coevolutionary sequence analysis," *Bioinformatics*, vol. 35, no. 9, pp. 1582-1584, 2019.
- [37] D. M. McCandlish, "Visualizing fitness landscapes," *Evolution*, vol. 65, no. 6, pp. 1544-1558, 2011.
- [38] J. Van Cleve and D. B. Weissman, "Measuring ruggedness in fitness landscapes," *Proceedings of the National Academy of Sciences*, vol. 112, no. 24, pp. 7345-7346, 2015.
- [39] G. von Heijne, "Control of topology and mode of assembly of a polytopic membrane protein by positively charged residues," *Nature*, vol. 341, no. 6241, pp. 456-458, 1989.
- [40] M. Rapp, E. Granseth, S. Seppälä and G. von Heijne, "Identification and evolution of dual-topology membrane proteins," *Nature structural & molecular biology*, vol. 13, no. 2, pp. 112-116, 2016.
- [41] S. Mukherjee, R. Seshadri, N. J. Varghese, E. A. Elie-Fadrosh, J. P. Meier-Kolthoff, M. Göker, R. C. Coates, M. Hadjithomas, G. A. Pavlopoulos, D. Paez-Espino, Y. Yoshikuni, A. Visel, W. B. Whitman, G. B. Garrity, J. A. Eisen, P. Hugenholtz, A. Pati, N. N. Ivanova, T. Wojke, H.-P. Klenk and N. C. Kyrpides, "1,003 reference genomes of bacterial and archaeal isolates expand coverage of the tree of life," *Nature biotechnology*, vol. 35, no. 7, pp. 676-683, 2017.

- [42] K. Katoh and D. M. Standley, "MAFFT multiple sequence alignment software version 7: improvements in performance and usability," *Molecular biology and evolution*, vol. 30, no. 4, pp. 772-780, 2013.
- [43] A. M. Waterhouse, J. B. Procter, D. M. Martin, M. Clamp and G. Barton, "Jalview Version 2 — a multiple sequence alignment editor and analysis workbench," *Bioinformatics*, vol. 25, no. 9, pp. 1189-1191, 2009.
- [44] M. Bogdanov, W. Dowhan and H. Vitrac, "Lipids and topological rules governing membrane protein assembly," *Biochimica et Biophysica Acta (BBA) - Molecular Cell Research*, vol. 1843, no. 8, pp. 1475-1488, 2014.
- [45] D. M. Fowler, J. J. Stephany and S. Fields, "Measuring the activity of protein variants on a large scale using deep mutational scanning," *Nature protocols*, vol. 9, no. 9, pp. 2267-2284, 2014.
- [46] F. Morcos, A. Pagnani, B. Lunt, A. Bertolino, D. S. Marks, C. Sander, R. Zecchina, J. N. Onuchic, T. Hwa and M. Weigt, "Direct-coupling analysis of residue coevolution captures native contacts across many protein families," *Proceedings of the National Academy of Sciences*, vol. 108, no. 49, pp. E1293-E1301, 2011.
- [47] B. E. Suzek, H. Huang, P. McGarvey, R. Mazumder and C. H. Wu, "UniRef," *Bioinformatics*, vol. 23, no. 10, pp. 1282-1288, 2007.
- [48] T. A. Hopf, C. P. Schärfe, J. P. Rodrigues, A. G. Green, O. Kohlbacher, C. Sander, A. M. Bonvin and D. S. Marks, "Sequence co-evolution gives 3D contacts and structures of protein complexes," *Elife*, vol. 3, p. e03430, 2014.
- [49] B. J. Grant, A. P. Rodrigues, K. M. ElSawy, J. A. McCammon and L. S. Caves, "Bio3d: an R package for the comparative analysis of protein structures," *Bioinformatics*, vol. 22, no. 21, pp. 2695-2696, 2006.
- [50] H. Wickham, R. François, L. Henry, K. Müller and D. Vaughan, "dplyr: A Grammar of Data Manipulation," 2023.
- [51] P. Gatti-Lafranconi, "Pymol script: loadBfacts.py," figshare, 2014. [Online]. Available: <https://doi.org/10.6084/m9.figshare.1176991.v1> [Accessed 16 May 2023]
- [52] P. Gatti-Lafranconi, "PyMOL Wiki," 22 September 2014. [Online]. Available: https://pymolwiki.org/index.php/Load_new_B-factors. [Accessed 16 May 2023].
- [53] H. Nguyen, D. A. Case and A. S. Rose, "NGLview – interactive molecular graphics for Jupyter notebooks," *Bioinformatics*, vol. 34, no. 7, pp. 1241-1242, 2018.
- [54] C.-S. Jeong and D. Kim, "Structure-based Markov random field model for representing evolutionary constraints on functional sites," *BMC Bioinformatics*, vol. 17, no. 1, pp. 1-11, 2016.
- [55] T. Berbasova, S. Nallur, T. Sells, K. D. Smith, P. B. Gordon, S. L. Tausta and S. A. Strobel, "Fluoride export (FEX) proteins from fungi, plants and animals are 'single barreled' channels containing one functional and one vestigial ion pore," *PloS One*, vol. 12, no. 5, p. e0177096, 2017.

- [56] R. Sloutsky and K. M. Naegle, "ASPEN, a methodology for reconstructing protein evolution with improved accuracy using ensemble models," *Elife*, vol. 8, p. e47676, 2019.
- [57] R. Sloutsky, "A new approach to reconstructing protein evolution," University of Massachusetts Amherst, 17 October 2019. [Online]. Available: <https://www.umass.edu/news/article/new-approach-reconstructing-protein>. [Accessed 16 May 2023].
- [58] E. Firnberg, J. W. Labonte, J. J. Gray and M. Ostermeier, "A comprehensive, high-resolution map of a gene's fitness landscape," *Molecular biology and evolution*, vol. 31, no. 6, pp. 1581-1592, 2014.
- [59] S. el-Showk, "The Meaning of Fitness," Scitable by Nature Education, 10 November 2014. [Online]. Available: https://www.nature.com/scitable/blog/accumulating-glitches/the_meaning_of_fitness/. [Accessed 16 May 2023].
- [60] L. Perfeito, S. Ghozzi, J. Berg, K. Schnetz and M. Lässig, "Nonlinear fitness landscape of a molecular pathway," *PLoS Genetics*, vol. 7, no. 7, p. e1002160, 2011.
- [61] J. F. Wilkins and P. Godfrey-Smith, "Adaptationism and the adaptive landscape," *Biology & Philosophy*, vol. 24, pp. 199-214, 2009.
- [62] S. J. Gould and R. C. Lewontin, "The spandrels of San Marco and the Panglossian paradigm: a critique of the adaptationist programme," *Proceedings of the royal society of London. Series B. Biological Sciences*, vol. 205, no. 1161, pp. 581-598, 1979.