Musical Proteins: The Intersection of Coding, Science and Art

By: Gabrielle Kerkow

Table of Contents

Part I

Problem Overview

Researchers have found that there exists many similarities in the layered structures of both proteins and music. By assigning each amino acid to a particular note, point mutations can be easily ‘heard,’ as humans have evolved a keen ear for detecting differences between melodies. Our goals with this project are to determine a method of mapping amino acids to any scale of choice; to produce a short playback of a recognizable melody in order to demonstrate the code’s efficacy; and to produce a melody resulting from a short peptide sequence.

Program Design

Bioinformatics is, first and foremost, a field interested in compiling and simplifying extensive data structures into digestible chunks. Changes in large protein structures can be difficult to spot, or taxing to model. In contrast, the human ear is much more finely attuned to noticing changes in melodies than the human eye is at playing spot-the-difference. The act of converting protein sequences into identifiable music is called ‘sonification,’ and remains a relatively unexplored method/heuristic for understanding protein structure, mainly due to its room for creative expression. In this project, we sought to create a tool that can produce music from amino acid sequences, and that can do so using an arbitrarily assigned conversion table between residues and a corresponding note. Our aims were for this tool to be flexible enough to incorporate any desired music scale, standard or nonstandard, and include or exclude any number of amino acids. Because dictionaries in Python operate like mathematical functions, it is even possible to map multiple amino acids to the same note.

The simplest method of arranging the amino acids was by size; larger amino acids were assigned to lower notes, in sequential order. Here, we have produced two alternative versions of the code, which only differ by their assignments of amino acids to notes along a musical scale.

The first, “main.py,” assigns each amino acid onto a C Major pentatonic scale, ranging from C2 to A5. This serves the purpose of producing a melody that sounds coherent even when given random protein sequences, as each note naturally harmonizes with the others. However, it has the downside of producing notes that may be octaves apart, resulting in ‘jumps’ that may seem musically unappealing.

The second, “alt.py,” assigns each amino acid onto a full twelve-tone scale, ranging from C4 to G5. This was to facilitate a wider variety of melodies across a narrower range of frequencies, at the cost of being musically disjointed and, to some, unpleasant.

Results & Output

Each of the two .py files produces a different melody; the protein sequence input for main.py is Small Protein mntS, a 42-amino acid long protein found in E. coli; and the input for alt.py is a sequence that ‘reverse engineers’ the arpeggiated opening of J.S. Bach’s *Jesu bleibet meine freude*, a popular wedding and church chorale. This melody was chosen because it did not necessitate changing the length of each note.

One particularly useful aspect of this tool is its ability to recognize repeats or motifs within an amino acid sequence - short tandem repeats within larger genetic material would sound like musical motifs, and back-to-back repeats sound like a note twice as long as usual.

The mntS-generated melody can be heard here.

The Bach melody can be heard here.

Part II

Code Roadmap

The protein’s amino acid sequence will be mapped to specific frequencies that are then converted to waves and accumulated to create a WAV file to generate music. To begin the process, a code to create WAV files needs to be composed. The reference code used for this project was generated by Zach Denton in his tutorial, “Generate Audio with Python” (1).

The code begins with importing several modules; “sys”, “wave”, “math”, “struct”, “random”, and functions from “itertools” that are key for code structuring and operation. Importing is followed by a series of defined functions; “grouper” , “sine_wave”, “compute_samples”, and “write_wavefile”. Grouper function uses “iter(iterable)” to run through a list and group it in a specified “n” value. Followed by “zip_longest” to include a “fillvalue” for empty variables if the original list is not a multiple of “n”. This function is useful for filling groups and processing them simultaneously. Sine_wave are ideally used to synthesize audio by frequencies relatively to music. The function sets parameters for a default sine wave and uses if/else statements to calculate waves at specified frequencies, amplitude, and time range over an infinite length. The next step would be to compose a generator to sum up waves so audio channels can output sounds accordingly. An audio channel serves as a single pathway for sound to come from or go to a single point. A default stereo audio has 2 channels, a left and right channel. “Compute_samples” uses “zip(*channel)” to transform the channel to unload data for computation by “(map(sum” where it is then compiled into tuples using the “zip” in “(zip*(map…” for specified “nsample” length, which in this case in infinite as the nsample is set to none. Finally, write_wavefile creates an audio sample comprising an array of parameters such as “framerate”, “sampwidth”, etc. to form the output WAV file. Values within the WAV file are to be in binary representation, which are “max_amplitude” floating-points calculated from sample width bytes and then converted by “struct.pack”. The sample is 16-bit binary representation, which means the sample is converted to “h” integer, or 16 integers made of a combination of 0s and 1s. All samples are then appended to the “bytearray”, or named file “frames_bytes” to be written as audio frames for the WAV file. The WAV file can now be used to execute protein to music note formation.

The program to generate music from a protein’s amino acid sequence begins with defining the sequence of the protein and file name of the melody to be composed. The next series of codes are to set parameters for the program. A normal stereo audio is composed of 2 channels, thus setting “number_channels” to 2. “Sample_bits” and “sample_rate” of basic CD audio quality have bits and rate of 16 and 44100 respectively. With a max range amplitude for a WAV file being [-1.0,1.0], scaling is used to control the volume or amplitude of the audio, with default at 0.5. The next parameter set is the audio duration, with each note to be played a half second using “len(sequence) * 2”. To set parameters for the total number of samples in the WAV file, an equation that cancels time for rate is required, with “sample_rate * total_time”. However, audio samples are typically represented as 16-bit integers, thus implementing integer function in the equation “int(sample_rate * total_time)” to convert the floating-points to integers. Since frequencies are used to create sound or pitch, a dictionary to map notes to frequencies is essential. Notes in the C Major pentatonic scale are mapped to their corresponding frequencies in the first dictionary. The second dictionary is composed of amino acids, with each assigned to a different note from the previous dictionary, based on molecular weight (here, larger amino acids produce lower notes). For the musical notes to be played sequentially rather than simultaneously, a list needs to be created. “Protein_generated_music = [ ]” initiates an empty list where the generated wave functions from the protein sequence will be stored. A for loop is used to produce a set of instructions that will automatically be repeated and applied to each element of the sequence. The first instruction is to retrieve the amino acid in the sequence at current index “`i`”. The second instruction uses the dictionary “acid_notes” to map the amino acids to a musical note which is then stored as “note”. This is followed by “frequency = freq[note]” that uses another dictionary to find the note’s corresponding frequency and storing it in “frequency”. The three instructions tie together with “wave_function” which calls the function “sine_wave” with set parameters to generate a sine wave to represent the musical note. The append function causes the waves to be played sequentially rather than simultaneously by adding the next generated wave in a list behind the previously generated wave. Finally, “compute_samples” accumulates the generated waves to create a single wave and is written into a WAV file by “write_wavefile”. With this code, a musical tune can be generated after inputting a protein’s amino acid sequence.

Program Run Instructions

Step One: Open file main.py in IDLE.

Please note the code will not work if opened in notepad++ or google colab. Here the code is written with annotations explaining each important line of code and what it is responsible for. You can change the sequence if a new protein is desired, as well as the notes corresponding to each amino acid. Refer to annotations for where to make these edits as desired.

Step Two: Click Run.

If no edits are desired the program will run as is, using a major scale as musical notes assigned to the various amino acids. If edits are made, click Save prior to running. Wait until “<<<” appears in the terminal, this may take longer depending on length of sequence but should be completed within 45 seconds. Once this is complete the file name “test.wav” will be available.

Step Three: Open test.wav file.

This file will be available and can be opened on any media player app, and music generated by protein will be heard.

Cited

1. https://zach.se/generate-audio-with-python/
2. https://www.wildlifeacoustics.com/resources/faqs/what-is-an-audio-channel#:~:text=A%20channel%20is%20a%20representation,contain%20multiple%20channels%20of%20data