2016

Effects of K80 molecular evolution model simulated by python3

GEN 812

MITCHELL FRIEND

Introduction

Mutations, in Biology, refers to nucleic acid changes in the genomic sequence of an organism. Due to the nature of biological forces (Evolution, Selection, Fitness, etc.) most heritable mutations, germline mutations, are silent and the overall rate is lower compared to somatic cells. Somatic cells are not heritable. In both germline and somatic cells, mutations accumulate overtime. The somatic cell mutation rate is three-fold larger than that germline mutation (Drake, 1969).

When mutations do occur, there are two types of nucleotide mutations, transitions and transversions. Transitions are a change in nucleotide but identical base, purine to purine (A ↔ G) or pyrimidine to pyrimidine (C↔T). Transversion are changes in nucleotide and base, (purine ↔ pyrimidine) (A → T, C), (T → G, A), (G → C, T), and (C → A,G). Transition rates are more common than transversion rates, about 67% to 33% respectively (Keller, 2007). Deleterious mutations that cause premature stop codons rarely propagate due to the protein loosing loss of function in a single point mutation. These codons, therefore, tend to combat mutation to the genome of an organism due to the damaging nature of this mutation. Thus nature selects against mutations to genetic code that cause amino acid codons to change to stop codons and these codons are less prone to genetic drift.

Changes in single nucleotides in a genome are known as SNPs (Single nucleotide polymorphisms). Two types of SNPs, that only affect protein coding regions, are synonymous and non-synonymous mutations. Synonymous SNP cause that codon nucleotide sequence to change but the codon still codes for the same amino acid. If the codon ATC, which codes for isoleucine, undergoes a transition mutation to ATT, the codon mutated but still codes for Isoleucine. A nonsynonymous SNP also changes the codon nucleotide sequence but also causes the codon to code for a different amino acid. The codon AAT codes for Asparagine, if a transverison mutation changes the codon to AAA, the codon now codes for Lysine.

In this study, a hemoglobin gene was put through a simulation that simulated the effects of an evolutionary mechanism known as genetic drift. The program is designed to mimic the K80 model of molecular evolution where transition and transversion are the types of mutations (Kimura, 1980).

Methods

Outline

*** = **change in outline**

- Import modules

- Create dictionaries
  (Codon, polarity, transversion and translation)

- Define Cds_class
  - prot(self):

- Use for loop to parse through a sequence and take 3 elements in ascending order translate them amino acid using codon dictionary and create new string containing amino acid sequence.

- mutate(self, n):
  - *** use for loop
    Backup the previous sequence in case mutation causes stop codon.
  - Create variable for probability of transversion /translation using np.random.choice *** and variable to choose random nucletide in sequence using numpy.
  - Use transversion and translation dictionary to change the previosu sequence. ***Use np.choice and Boolean statement and correct probability to choose between mutation.
  - *** detect new sequence for stop codons

- mute_prot(self):
  - Use for loop parse through mutated sequence find codons and translate codons and add amino acid to empty string just like prot(self):.

- *** prot_mute(self):
  - Simply uses for loop and zip to compare two AA sequences and count the positions that have different amino acids.

- count_snps(self):
  - compare mutated sequence to previous seq. if nucletide is different as a given position, find the codon translate codon in both sequences using codon dictionary, compare Amino acid. If different add 1 to nonsynonymous counter if same add 1 to synonymous counter.
  - *** use np.floor to find codon in nucleotide sequence

- diff(self):
  - takes previous amino acid sequence and compares it to the mutated sequence using zip and checks for polarity differences using polarity dictionary.
  - Uses for loop and if the amino acid is the same returns the int 0.
    elif amino acid is not the same in both sequences, compares polarity of each amino acid using polarity dictionary if both amino acids are have equal dictionary values returns the int 1.
    else returns the int 4.
  - Appends all int values to empty list and then return that list.

Outline – graphing

- Heatmap
  - *** import python file contain and other modules on jupyter.
    - ***calls mutate and diff classes and mutates the original heme gene x times with y SNP mutations per x.

- Errorbar
  - *** takes the average score of x trials of each amino acid site after y amount of mutations done to the original heme seq.
  - Graphs the average score of the amino acid site and uses error bars that depict a range with 3sigma confidence.

- Nucleotide difference
  - *** uses class mutate and diff to compare the total amount of difference at each position and graphs the amount of nucleotide changes given x amount of mutations

- Syn vs nonsyn mutations
  - *** takes two empty lists. Uses mutate to mutate the original sequence x times and uses count synps to add the total synonymous and nonsynonymous mutations in the mutated sequences after x mutations and appends that value to the empty list.

Code – python program

```
#Genetic Drift Simulator
"""
This program takes a given nucletide sequences for a protein coding gene and simpulates
genetic drift over time.
written by Mitchell Friend
"""
#import modules
import numpy as np
import seq_tools as st
import re
import sys

codon_dictionary = {
        'ATA': 'I', 'ATC': 'I', 'ATT': 'I', 'ATG': 'M',
        'ACA': 'T', 'ACC': 'T', 'ACG': 'T', 'ACT': 'T',
        'AAC': 'N', 'AAT': 'N', 'AAA': 'K', 'AAG': 'K',
        'AGC': 'S', 'AGT': 'S', 'AGA': 'R', 'AGG': 'R',
        'CTA': 'L', 'CTC': 'L', 'CTG': 'L', 'CTT': 'L',
        'CCA': 'P', 'CCC': 'P', 'CCG': 'P', 'CCT': 'P',
        'CAC': 'H', 'CAT': 'H', 'CAA': 'Q', 'CAG': 'Q',
        'CGA': 'R', 'CGC': 'R', 'CGG': 'R', 'CGT': 'R',
        'GTA': 'V', 'GTC': 'V', 'GTG': 'V', 'GTT': 'V',
        'GCA': 'A', 'GCC': 'A', 'GCG': 'A', 'GCT': 'A',
        'GAC': 'D', 'GAT': 'D', 'GAA': 'E', 'GAG': 'E',
        'GGA': 'G', 'GGC': 'G', 'GGG': 'G', 'GGT': 'G',
        'TCA': 'S', 'TCC': 'S', 'TCG': 'S', 'TCT': 'S',
        'TTC': 'F', 'TTT': 'F', 'TTA': 'L', 'TTG': 'L',
        'TAC': 'Y', 'TAT': 'Y', 'TAA': '*', 'TAG': '*',
        'TGC': 'C', 'TGT': 'C', 'TGA': '*', 'TGG': 'W', }
#codon_dictionary = takes codons and returns amino acid.
polarity_dictionary = {"A": 0, "R": 1, "N": 1, "D": 1, "C": 0,
                       "E": 1, "Q": 1, "G": 0, "H": 1, "I": 0,
                       "L": 0, "K": 1, "M": 0, "F": 0, "P": 0,
                       "S": 1, "T": 1, "W": 0, "Y": 1, "V": 0}
#polarity_dictionary = takes the amino acid as keys and reutrns 1 polar 0 for nonpolar
t_version_dict = {"A": ("T", "C"), "T": ("A", "G"), "G": ("C", "T"), "C": ("A", "G")}
#t_versio_dict  nucletide key and possible mutant nucletide = value
t_lation_dict = {"A": "G", "T": "C", "G": "A", "C": "T"}
#t_lation_dict = nucletide key and possible mutant nucletide = value

heme_cds =
"GTGCATCTGACTCCTGAGGAGAAGTCTGCCGTTACTGCCCTGTGGGGCAAGGTGAACGTGGATGAAGTTGGTGGTGAGGCCCTGGGCAGG
CTGCTGGTGGTCTACCCTTGGACCCAGAGGTTCTTTGAGTCCTTTGGGGATCTGTCCACTCCTGATGCTGTTATGGGCAACCCTAAGGTGA
AGGCTCATGGCAAGAAAGTGCTCGGTGCCTTTAGTGATGGCCTGGCTCACCTGGACAACCTCAAGGGCACCTTTGCCACACTGAGTGAGCT
GCACTGTGACAAGCTGCACGTGGATCCTGAGAACTTCAGGCTCCTGGGCAACGTGCTGGTCTGTGTGCTGGCCCATCACTTTGGCAAAGAA
TTCACCCCACCAGTGCAGGCTGCCTATCAGAAAGTGGTGGCTGGTGTGGCTAATGCCCTGGCCCACAAGTATCAC"

# hemoglobin nucletide sequence

"""
Use this class to take a nucleotide sequence store it as self.seq and translate it to a
protein w/ prot():. then mutate
the nucleotide sequence using mutate():. and tranlate into mute protein with mute_prot():.
count type mute (nonsyn and
sym) with count_snps():. count amino acid difference with mute_prot(): and create list of
polarity changes and total
number of nucleotide differences with diff():
"""
class Cds:
    def __init__(self, seq):
        self.seq = seq
        self.muteseq = (np.copy(st.str2seq(self.seq)))
    def prot(self):
        """:return: returns the translated protein sequence from the nuc sequence """
```

```
        prot_seq = ""
        for i in range(0, len(self.seq), 3): #counts by three over whole nuc seq
            codon = self.seq[i : i + 3] #stores each 3 nucs as a codon
            prot_seq += codon_dictionary[codon] #appends the dictionary value to prot seq
        return prot_seq


    def mutate(self, n):
        """
        :param n: number of nucleotide mutations to original seq per cycle
        :return: self.muteseq mutated sequnce
        """
        for x in range(n):
            backup_mute_seq = np.copy(self.muteseq) #stores previous sequence before
mutations
            single_mute = np.random.choice(range(0, len(self.muteseq))) # randomly
selection a single nuc
            prob_mute = np.random.choice([True, False], p=[0.34, 0.66]) # randomly selects
T(34%) and F(66%)
            if prob_mute == False: #if false is selected tranlation mutation occurs
                old_nuq = self.muteseq[single_mute] #takes element in mutseqslected by
single_mute and stores it
                new_nuq = t_lation_dict[old_nuq] # converts slected nuc to corresponding
dictionary value
            else:# True selected and transversion mutation occurs
                old_nuq = self.muteseq[single_mute] #takes element in mutseqslected by
single_mute and stores it
                trans_nuq = t_version_dict[old_nuq]# converts slected nuc to corresponding
dictionary value
                new_nuq = np.random.choice(trans_nuq)# chooses a random dictionary value
            self.muteseq[single_mute] = new_nuq #creates mutated sequence replaces previous
self.muteseq

            if "*" in self.mute_prot():
                self.muteseq = backup_mute_seq #replaces newself.muteseq if * (stopcodon)
and returns the prev seq

    def mute_prot(self):
        """
        :return: returns the translated amino acid sequence from the mutant nucletide
        """
        mute_prot_seq = ""
        for i in range(0, len(self.muteseq), 3):
            codon = self.muteseq[i: i + 3]
            mute_prot_seq += codon_dictionary[st.seq2str(codon)]
        return mute_prot_seq



    def prot_mute(self):
        """
        :return: return the amount of amino acid differences between old and new protein
        """
        if self.prot() != self.mute_prot():
            counter = 0
            x = zip(self.prot(), self.mute_prot())
            for i,j in x:
                if i != j:
                    counter += 1
            return counter

        else:
            return 0

    def count_snps(self):
        """
        :return: returns list where element[0] = synonymous mutation element[1] =
```

```
nonsynonymous
        """
        nuc_location = 0
        syn_mute = 0 #stores total synonymous mutes
        nonsyn_mute = 0 #stores total nonsynonymous mutes
        for x in range(len(self.seq)): #parses through original and mute nucleotide seq and
finds difference
            if self.seq[x] != st.seq2str(self.muteseq[x]):
                nucleotide_location = np.floor(x) #sotres the positon of nucletide that is
different
                codon_location = int(np.floor(nucleotide_location / 3)) # takes positon
divides by 3 rounds lowest int
                start_nuc = codon_location * 3 # returns positon of start nucletide
                codon_triplet= self.seq[start_nuc: start_nuc +3] #saves codon AAA or ATA
etc.)
                mute_codon_trip = st.seq2str(self.muteseq[start_nuc: start_nuc +3]) #saves
mutant codon AAT, ATG etc.)
                if codon_dictionary[codon_triplet] == codon_dictionary[mute_codon_trip]:
                    syn_mute += 1 #add 1 to syn_mute if mute_codon and codon code for the
same amino acid
                else:
                    nonsyn_mute += 1# add 1 to nonsyn_mute if codon and mute codon cod for
different amino acid


        return syn_mute, nonsyn_mute


    def diff(self):
        """
        :return: returns tuple with [0] is changes is polarity of amino acid and [1] total
#of nucletide changes
        """
        x = (self.prot())
        y = (self.mute_prot())
        z = zip(x,y)
        polarity = []
        nuc_differences = 0
        for a, b in z:
            if a == b: #if amino acid in prot and mute_prot is identical adds list value of
0
                polarity.append(0)
            elif a != b:
                if polarity_dictionary[a] == polarity_dictionary[b]: #if AA are diff but
have same polarity, add 1
                    polarity.append(1)
                else:
                    polarity.append(4) # AA different and different polarity add 4
        a = self.seq
        b = st.seq2str(self.muteseq)
        c = zip(a,b)
        for x, y in c:
            if x != y:
                nuc_differences += 1 # if different nucletide add 1
            else:
                nuc_differences += 0 # if same nucletide add 0

        return polarity, nuc_differences
```

Method – graphing code

Imports

```
In [ ]: import Final_Project
        import numpy as np
        from matplotlib import pyplot as plt
```

Heat Map (figure 1)

```
In [ ]: z = Final_Project.heme_cds
        mycds = Final_Project.Cds(z)
        lis_val = []
        n = 100
        for i in range(n):
            mycds.mutate(3)
            y = mycds.diff()[0]
            lis_val.append(y)
        plt.imshow(lis_val)
        plt.colorbar()
        plt.xlabel('Amino Acid position')
        plt.ylabel('Number of mutation cycles')
        plt.show()
```

Error Bar (figure 2)

```
In [ ]: z = Final_Project.heme_cds
        mycds = Final_Project.Cds(z)
        mute_val = []

        n = 10000
        for i in range(n):

            m = mycds.mutate(180)
            p = mycds.diff()[0]
            mute_val.append(p)
```

```
In [ ]: mean = np.mean(mute_val, 0)
        st_dev = np.std(mute_val, 0)
        std_error = ((3*st_dev)/(np.sqrt(10000)))

        plt.figure(figsize=(20,8))
        plt.plot(mean, '.')

  plt.errorbar(range(146), mean, std_error,linestyle='none')

  plt.xlabel("Amino Acid Position")
  plt.ylabel("Average score of Amino Acid")

  plt.show()
```

```
In [ ]: number_of_mute = []

        i=100
        t = np.logspace(1,4, 100).astype(int)
        for i in np.logspace(1,4, 100).astype(int):
            z = Final_Project.heme_cds
            mycds = Final_Project.Cds(z)
            mycds.mutate(i)
            y = mycds.diff()[1]
            number_of_mute.append(y)
        plt.semilogx(t,number_of_mute)
        plt.show()
```

# of nuc mutation

(fig3)

```
In [ ]: plt.semilogx(t,number_of_mute)
        plt.xlabel("Number of Mutations Implemented to Orginal Sequence")
        plt.ylabel("Number of Nucletides Different from Original Sequence")
        plt.show()

In [ ]: number_of_sym = []
        number_of_nonsym = []
```

```
In [ ]: number_of_sym = []
        number_of_nonsym = []

        i=100
        t = np.logspace(1,4, 100).astype(int)
        for i in np.logspace(1,4, 100).astype(int):
            z = Final_Project.heme_cds
            mycds = Final_Project.Cds(z)
            mycds.mutate(i)
            y = mycds.count_snps()[0]
            u = mycds.count_snps()[1]
            number_of_sym.append(y)
            number_of_nonsym.append(u)
```

 # OF SYN AND NONSYN MUTATIONS (FIGURE 4)

```
In [ ]: plt.semilogx(t,number_of_sym, label='synonmous')
        plt.semilogx(t,number_of_nonsym, label= 'nonsynonmous')
        plt.legend(loc=7)
        plt.xlabel("Number of Mutations Implemented to Orginal Sequence")
        plt.ylabel("Number of Type of Mutation")
        plt.show()
```

Methods – How python can simulate genetic drift

Python to simulate the K80 model of molecular evolution applied over a vast amount of genome duplications in an effort to investigate genetic drift.

The python code written uses a Cds type class. Inside the class there __init__(self, seq): is used to create an instantiation operator so, later the called method will run massive calculations without having to rerun the whole code, thus saving time later on. This method also takes the parameter seq, which will be a nucleotide sequence, and save it to self.seq.

The method prot(self): uses a for loop and passes three elements from self.seq at a time though and then appends the corresponding amino acid from codon_dictionary to an empty list. The method mute_prot(self): utilizes the same code but uses self.mutseq. prot and mut_prote helped simulate genetic drift by being able to compare how amino acid composition changed between generations.

The method mutate(self, n): is used to mutate a random nucleotide in a self.seq n times. Once a random nucleotide is selected, using np.random.choice ([True, False], p= 0.34, 0.66), which takes a parameter and keyword argument, to select True 34% of the timer and False 66% of the time. True represents a transversion mutation and replaces the selected nucleotide with a different base that is randomly selected from the value in t_versrion_dict and False represents a transition mutation and replaces the selected nucleotide with the corresponding base in t_lation_dict. If a stop codon was mutated, meaning self.muteseq contained "*" the previous sequence was returned and no mutation occurred. This is used to show how certain codons are less susceptible to mutations and genetic drift.

The method count_snps(self): is used to calculate the total amount of synonymous and nonsynonymous mutations that occur after the method mutate is returns self.muteseq. A list is returned where element[0] are synonymous mutations and element[1] are nonsynonymous mutations. A for loop was used to compare each individual nucleotide in self.seq and self.muteseq. the imported function np.floor is utilized to find the proper reading frame once a difference is detected between self.seq and self.muteseq. This was achieved by divided the nucletide position by three, and if a float was returned, np.floor rounded that number to the lowest integer value. the codon selected in self.muteseq and self.seq were then translated by codon_dictionary and if the values matched it was recorded as a synonymous mutation and if not, recorded as a nonsynonymous mutation. This allowed for the scenario if two mutation occurred in the same codon ex(AAT → ATC) for this to be recorded as two nonsynonymous mutation rather than just 1. This method ultimately simulates how nonsynonymous and synonymous mutations accumulate overtime.

Diff(self): takes the returned values(the amino acid sequence) from method prot and mute_prot and compares the two different strings using the built-in function zip. A for loop parses over the zipped strings and if the elements are identical it appends 0 to a list. Using elif, If elements are not identical if takes the element from self.seq and self.muteseq and returns the value assigned to that key from the polarity_dictionary. If the returned value is equal then the int 1 is appended to the list and 4 if there are different values. This gives a better look into how genetic drift act on certain amino acid positions and if there is a greater chance for an amino acid position to mutate to an opposite polarity compared to another amino acid.

Diff was also used to compare the self.seq and self.muteseq nucleotide sequences and return an integer that represented the total amount of positions where the nucleotide was different from the original sequence.

Using Jupyter, graphing was done to display how accumulated mutations over vast amounts of time and genome replications can allow for genetic drift to alter the composition of both amino acid and nucleotide structure. Code entered into jupyter utilized the class structure of the imported python file. For example, figure 2., runs the method mutate and diff 10,000 times. This took approximately 15 minutes. This would have taken much longer if the whole file, which contained 7 functions compared to calling on just the two.

Analysis

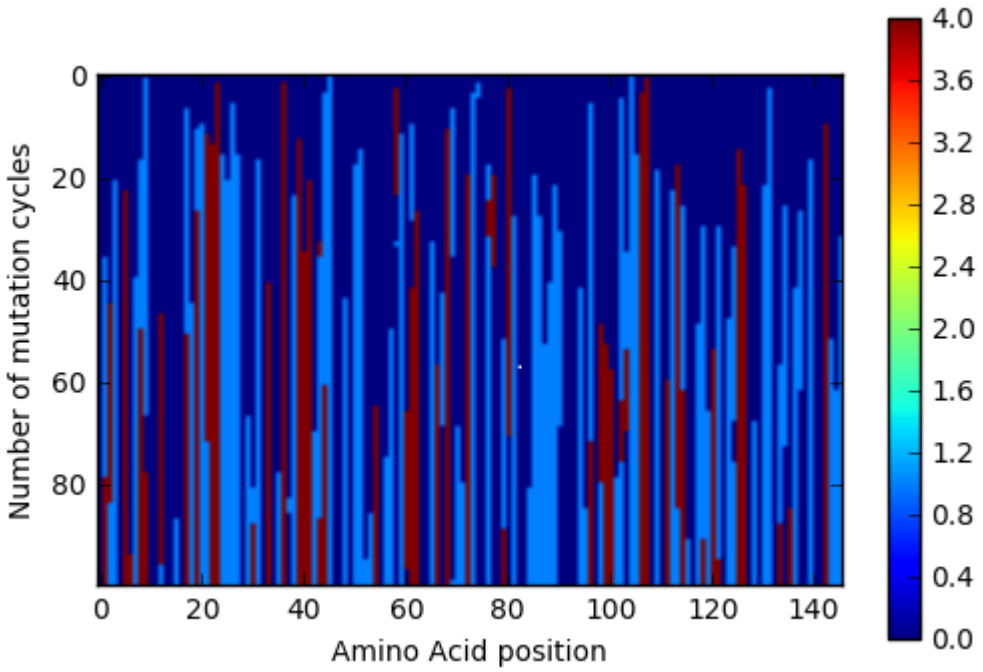

Figure 1. Shows Amino Acids in the heme protein that have not changed (dark blue) changed Amino Acid but no change in polarity (light blue) and Amino Acid changes that also change polarity (red). Each mutation cycle takes into account the chance for three nucleotides to change. 100 cycles allow for there the chance for 300 mutations.

Using the mutation program generated in python, an amino acid sequence of 146 amino acids long was mutated. Amino acid positions that remain dark blue throughout the 100 mutation cycles show that that amino acid has not mutated from the original sequence. Positions that change dark red show a change in amino acid and in polarity (nonpolar <-> polar). Polarity changes have the chance to be much more deleterious due to the changes these mutations can make in the tertiary structure of the protein. Like in sickle cell anima, one nonsynonymous mutation, from A to T, changes the polarity of a single amino acid and causes a life threatening disease. In future programs it would be interesting to calculate in the probability for polarity changes in in amino acid structure. Areas that are light blue show changes in amino acids the result in a new amino acid but the same polarity. These

mutations seem to be just as common, if not, marginally more common than polarity changes. However, these mutations still can affect tertiary structure of a protein due to the changes in an Amino acid side chain (sulfur side chain <-> non-sulfur side chain).

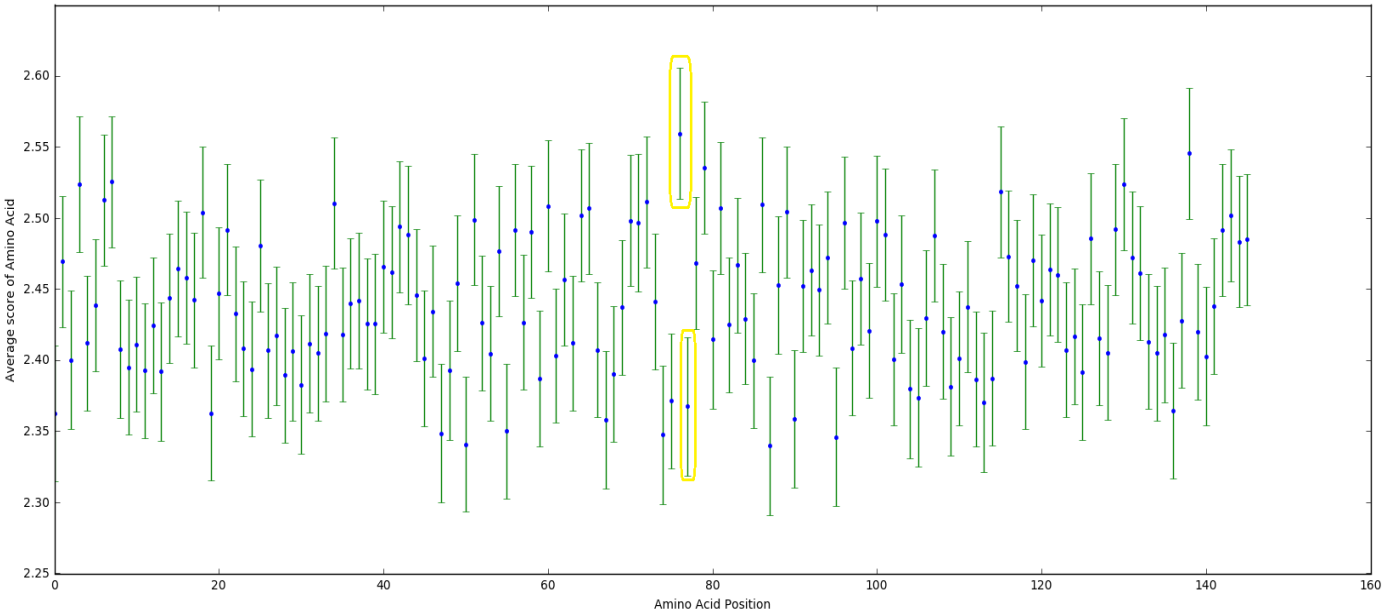

Figure 2. Amino acid position (X-axis) plotted against Average score of Amino Acid (y-axis). Average Score of amino acid represents the average polarity of an amino acid, after 180 nucleotide mutations, over 10,000 trials. The green error bars indicate the standard error for each amino acid score. Standard error was calculated using the sigma rule.

Amino acids with scores closer to 3.0 represent amino acids that have a higher affinity to mutate to an opposite polarity. Scores that are closer to 2.0 have a lower affinity to mutate to an opposite polarity. The two highlighted amino acid positions show a statistically significant difference between polarity scores. This means that the amino acid position, that has a lower score, will have a lower score 99.7% of the time than the amino acid with the higher score. In conclusion, this figure highlights the fact that some amino acids are more susceptible to polarity changes than other amino acids.

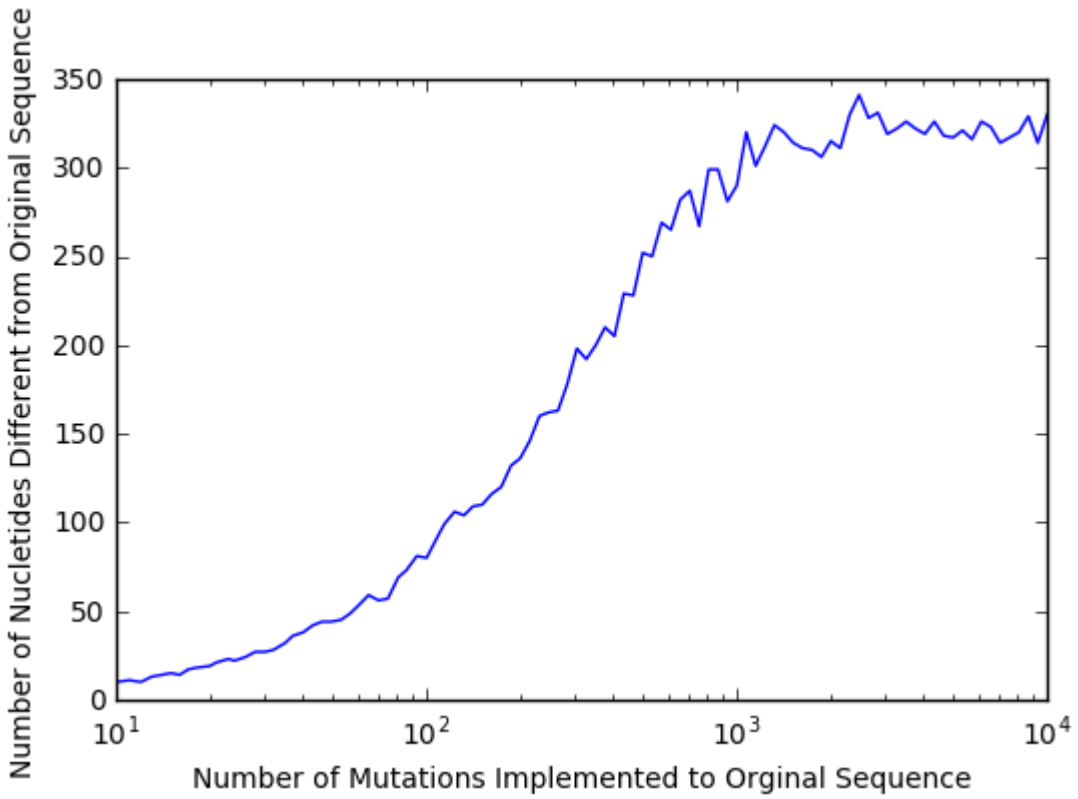

Figure 3. Number of Mutation implemented to original sequence (x-axis) represents how many times the nucleotide sequence was mutated per trail. Number of nucleotides different (y-axis) shows how many nucleotides are different from the original sequence. There are 438 nucleotides.

Figure 3 depicts the total amount of nucleotides different from the original sequence. This means that after X number of mutations if the nucleotide at position Z changed from A to T, C, or G then the Y value increases by 1. If the A nucleotide at position Z stays, or mutates back to, A after X mutations then the Y value does not increase. There is a sharp, exponential like, increase in mutations from 10 to 100 mutations to the original nucleotide sequence. A steady, linear like, increase in mutations from 100 to 1,000 mutations to the original sequence. The number of nucleotides changes plateaus from 1,000 to 10,000 mutations to the sequence. The reason these trends occur is due to the nature of there only being 4 different nucleotides in DNA sequence. The mutated sequence around 75% different from the original sequence after 10,000 mutations. In the last mutation applied the nucleotide has a 25% chance to return to the original and 75% chance that it is different.

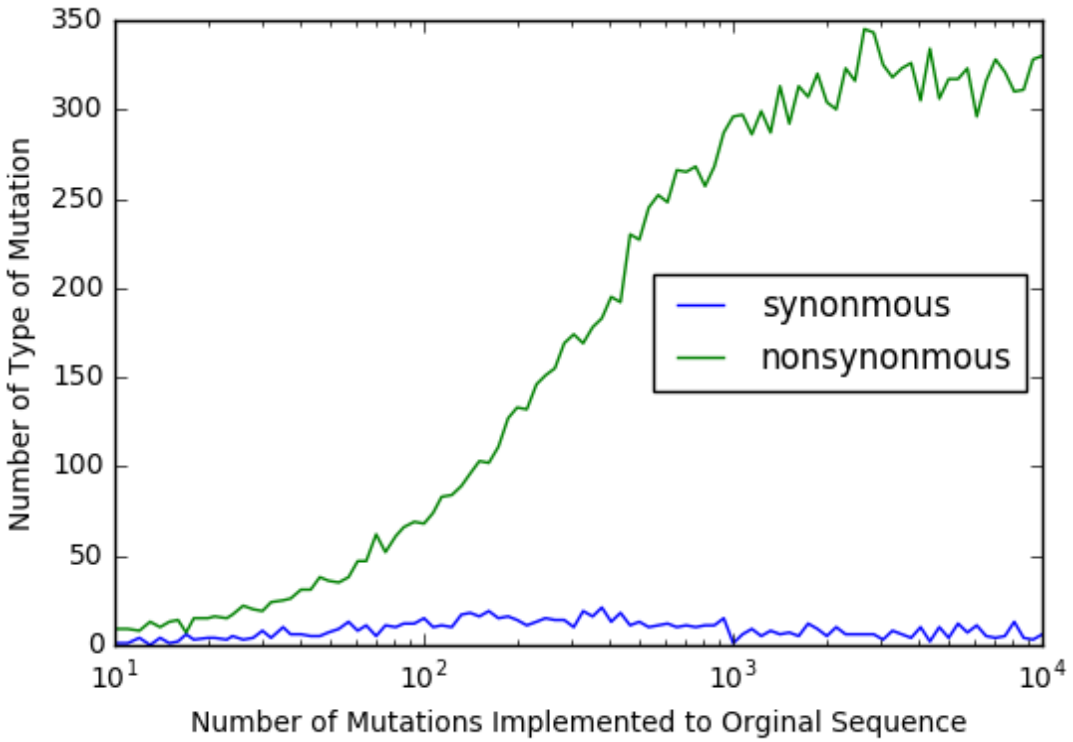

Figure 4. shows the number of nonsynonymous (green) and synonymous (blue) mutations in a mutated sequence at the end of 10 – 10,000 mutations to the original sequence. Number of mutation to original sequence (X-axis). Number of type of mutation in the mutated sequence after X mutations.

Figure 4 shows the number of nonsynonymous mutations that resembles trends similar to figure 3. Nonsynonymous mutations grow exponential like from 10 to 100 mutations. Linear like increase from 100 to 1,000 mutations and plateaus from 1,000 to 10,000. Synonymous mutations slowly increase from 10 to 100 mutations. Plateaus from 100 to 1,000 mutations, and sharply decreases and again plateaus from 1,000 to 10,000.

Nonsynonymous mutations have a higher chance of randomly occurring due to the nature of codons. A single, random, change in a codon triplet has many more opportunities to code for different protein and a lower chance to code for the same protein. As more mutation start to occur in the original sequence, less than 100, both synonymous and nonsynonymous mutations increase. This is because as mutations are accumulating in the original sequence nucleotide changes cause either a different protein to be coded (high chance) for or an identical protein but with a different codon (low chance). This increase in both happens until about 75% of the original sequence is mutated (about the time that number of mutation is equal to the amount of nucleotides in the sequence). After this, synonymous mutations begin to decrease slowly as nonsynonymous mutations begin to replace synonymous mutations.

Conclusion

The K80 model of molecular evolution was successfully applied to the hemoglobin gene using a simulation derived in python. It was concluded that certain amino acid position had a lower incidence of mutating from their original amino acid due to their codon easily being mutated to code for a stop rather than an amino acid (figure 1). Certain amino acids have a higher incidence of changing polarity based on their codon composition (figure 2). The number of positional nucleotide difference that occur in the mutated sequence, after x number of mutations, typically do not change more than 75% of the sequence (figure 3). This also accurately depicts K80 molecular evolution theory where each nucleotide has an equal frequency of occurring, $P_{A,C,G,T} = 0.25$ (kimura, 1980). Lastly, how nonsynonymous mutations accumulate to a similar frequency of nucleotide positional differences and that synonymous mutations increase in frequency then begin to decrease once the amount of mutation a sequence is subjected to passes the number of nucleotides in the sequence (figure 4). For future research, I think if this program implemented code to accurately portray how nonsynonymous mutations in hemoglobin can cause diseases, like sick cell and a variety of other anemic causing diseases, and prevent mutations from occurring that mimic the genomic structure of the diseases in order to give a more precise look at how hemoglobin amino acid and nucleotide structure mutates from genetic drift. In addition, I think code depicting the probability of a somatic mutation actually occurring during a cell division and was plotted against time in years. This simulation could then put into perspective how long it would take and how many cell divisions would occur for the original sequence to undergo 10,000 mutations.

References

Drake, J. W. (1969). Spontaneous Mutation: Comparative Rates of Spontaneous Mutation. Nature, 221(5186).

Keller, I., Bensasson, D., & Nichols, R. A. (2007). Transition-Transversion Bias Is Not Universal: A Counter Example from Grasshopper Pseudogenes. PLoS Genetics, 3(2).

Kimura, M. (1980). A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. Journal of Molecular Evolution, 16(2)