Population Genetic Analyses of Genomic Data 1

USN: 303039534

1 Exercises

1.1 Phylogenetics

We write a python script that constructs a tree graph based from the data structure provided, storing the length of each edge and the label of each node. We also write a topological sort function that will sort the nodes non-uniquely such that no pair of nodes a and b with indices i and j where $i < j$ will b be a parent of a. Using this function, we modify include in our parsing function a subroutine that counts for each node a how many total descendents it has and how many of these are leafnodes and internal nodes respectively. We also annotate each node with its total length from the rootnode.

These numbers are given in table /reftab:treesum sorted from fewest to most total descendants. This also identifies node 5 as the parent node and nodes 8,14,15,2,11,6,13 as leafnodes.

node	cumulative daughter nodes	internal nodes	leafnodes	distance from root
8	0	0	0	8.2
14	0	0	0	8.2
15	0	0	0	8.2
2	0	0	0	8.2
11	0	0	0	8.2
6	0	0	0	8.2
13	0	0	0	8.2
12	2	2	0	5.5
4	2	2	0	7.1
7	2	2	0	3.5
3	4	3	1	3.7
10	4	3	1	5.2
1	10	6	4	2.2
5	14	8	6	0

Table 1: daughter nodes

b) We include in the final column of table /reftab:treesum the length from the rootnode of each node. This shows that the length to every leafnode from the root is 8.2 units. The tree is thus consistent with an evolutionary tree where branches represent time since all current individuals (leafnodes) are equidistant from their combined most recent common ancestor (the rootnode).

We can also show this visually by plotting the tree such that the length along the y-axis corresponds to the length L between two nodes. We label each node with its node label and give the result in figure 5

c)

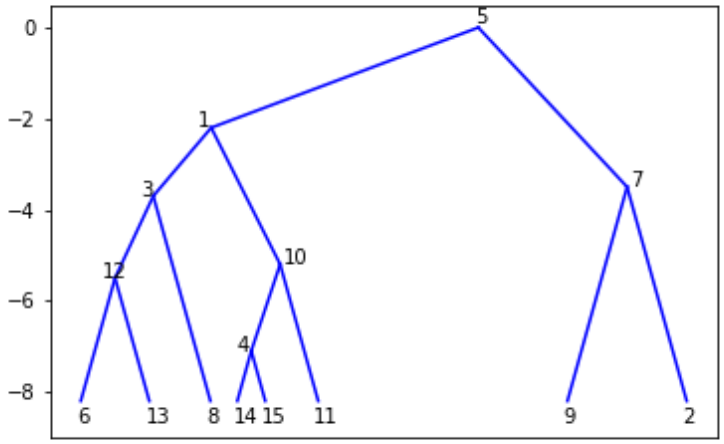
given our topological sort function, we can now easily print out a relabelled tree where every daughter node has a lower number than its parent, simply by labelling each node according to it's position in the reverse of the topologically sorted list.

we can also make this plot a relabelled graph

1.2 Time-dependent selection

We consider an organisms with 1000 genes each of which can be fixed in state 0 or 1. We let f_i^a denote the fitness advantage of gene i in state a and draw it from an exponential distribution with parameter 100. At t=0, we assume state 1 to be beneficial and let $f_i^0 = -f_i^1$, $f_i^1 > 0$. We assume that no more than one gene is polymorphic at any time. Given a mutation in a gene, the probability of fixation is given by

$$p_{fix}(f_i^a) = \frac{1 - e^{-2f_i^a}}{1 - e^{-2Nf_i^a}} \tag{1}$$



(a) Plot of mean fitness against allele frequency p

Figure 1

new label	original label	cumulative daughter nodes	internal nodes	leafnodes	distance from root
5	8	0	0	0	8.2
8	14	0	0	0	8.2
7	15	0	0	0	8.2
4	2	0	0	0	8.2
3	11	0	0	0	8.2
2	6	0	0	0	8.2
1	13	0	0	0	8.2
6	12	2	2	0	5.5
9	4	2	2	0	7.1
10	7	2	2	0	3.5
11	3	4	3	1	3.7
12	10	4	3	1	5.2
13	1	10	6	4	2.2
14	5	14	8	6	0

Table 2: daughter nodes

Where N is the population size. We assume the lifespan of each polymorphism to be

$$\min\left(10, \frac{2\log\left(N-1\right)}{919 f_i^a}\right) \tag{2}$$

We let N = 100 and use the natural logarithm.

We note that this particular system can be implemented in two different ways. In the first implementation, we consecutive mutations between the 0 and 1 states of a gene to be independent such that whenever a mutation occurs in a gene in state 1, we draw a new fitness $f_i^0 = -|f_i|$ and whenever a mutation occurs in state 0, we draw a new fitness $f_i^1 = +|f_i|$.

In the other implementations, the relative fitnesses of states 0 and 1 are fixed and we draw all f_i^1 at the beginning of the simulation corresponding to the system switching between two pre-defined alleles over the course of the simulation. On one hand, this justifies always having $f_i^0 < 0$ and $f_i^1 > 0$, but on the other hand it is unlikely that a second mutation in gene i will exactly reverse the effect of the first mutation. In the following, we consider both of these interpretations.

The number of allele 1 over the 1000 loci for these two approaches have been plotted in figures 3a and 5a. We see that

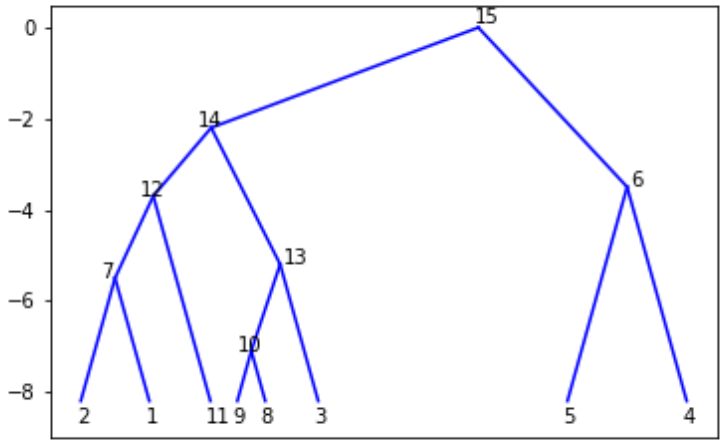


Figure 2: relabelled tree

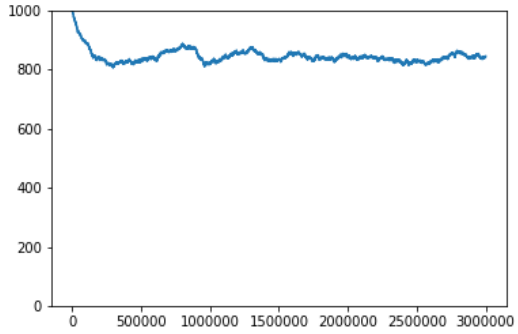
in both cases, an equilibrium state of $n_1 \approx 800$ has been reached after 500,000 timesteps.

$$n1 \cdot p(a = 1 \rightarrow a = 0) = n2 \cdot p(a = 0 \rightarrow a = 1) \tag{3}$$

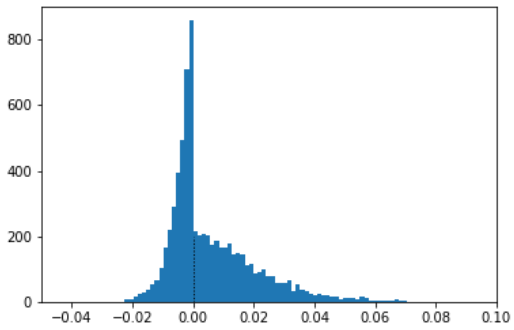
This implies that the integral of the right side of the above figure must be equal to the integral of the left side. We can thus calculate the expected n1 analytically.

b)

We start sampling the f values that lead to successful fixation after t=500,000 and run the simulation for an additional 2,500,000 timesteps. We then plot histograms showing the frequency of fixation for a given value of f in figures 3b and 5b.



(a) $n1$ as a function of simulated time



(b) Frequency of fitnesses f that fix in the population

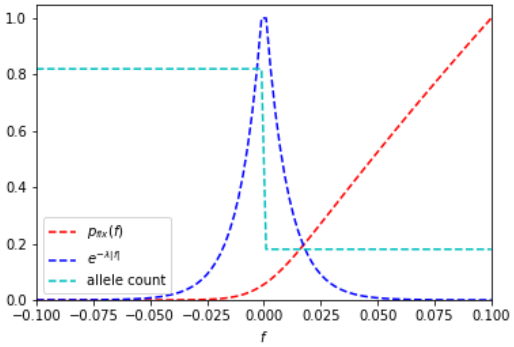
Figure 3: Simulation results when drawing a new f_i whenever a mutation occurs.

We can explain the shape of the distribution of frequencies of fixation in the case of resampling by putting together three separate distributions (figure 4a). These are
1. the number of alleles in states 0 and 1 respectively since a $0 \rightarrow 1$ transition is associated with $f > 0$ and vice versa
2. The probability with which we draw a given f value if in state 0 for f¿0 and in state 1 for f¡1. This is given by the exponential distribution $e^{-\lambda|f|}$ with $\lambda = 100$.
3. the probability with which a mutation fixates given by equation 1. If we multiply these three probability distributions, we arrive at a theoretical distribution that resembles the simulated data very closely (figure 4b suggesting that

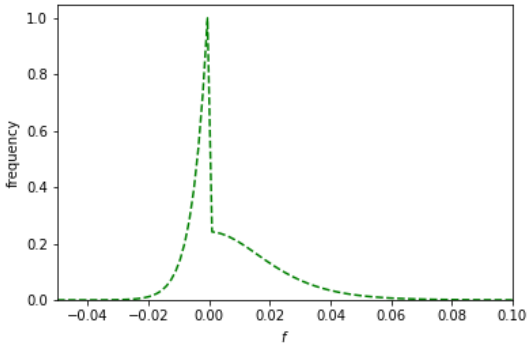
this is a good explanation for the observed data. The frequency distribution is thus given by

$$y \propto n_1 * exp(-|f|) * \frac{1 - e^{-2f}}{1 - e^{-2Nf}} f < 0 \tag{4}$$

$$y \propto n_0 * exp(-|f|) * \frac{1 - e^{-2f}}{1 - e^{-2Nf}} f > 0 \tag{5}$$



(a) Distributions of alleles (cyan), frequency of drawing a given fitness (blue) and probability of fixation (red) as a function of fitness f.



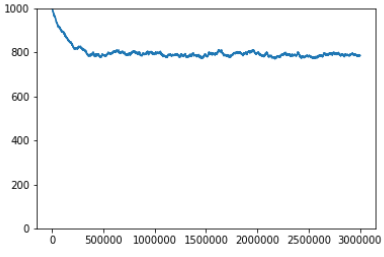
(b) Product of the three distributions in (a). This distributions explains the observed form of the simulate data.

Figure 4

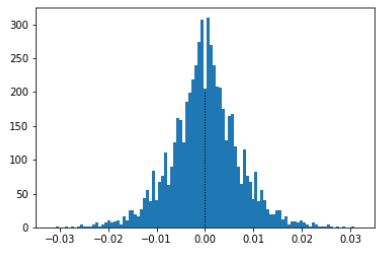
If instead we fix the values of f at the beginning of the simulation, consecutibe mutations are no longer independent and we isntead observe a symmetric distribution of frequencies of fixation (figure 5b). Considering the set of all mutations in a single site i. Whenever it is in state 1, the probability of fixation is $p_1 = p_{fix}(f_0^i)$ and whenever it is in state 0 the probability of fixation is $p_0 = p_{fix}(f_1^i)$. This probability can be interpreted as a frequency of switching, and the average time between switching is thus the inverse of this frequency. However, because the state must switch back from 1 to 0 to switch from 0 to 1, in the limit of $t \to \infty$ where many switches occur, the average switching time for site i is given by the mean switching time from $0 \to 1$ and $1 \to 0$. Denoting this mean switching time τ_i,

$$\tau_i \propto \frac{1}{2p_1} + \frac{1}{2p_0} = \frac{1 - e^{-2Nf_i^0}}{2 - 2e^{-2f_i^0}} + \frac{2 - 2e^{-2Nf_i^1}}{1 - e^{-2f_i^1}} \tag{6}$$

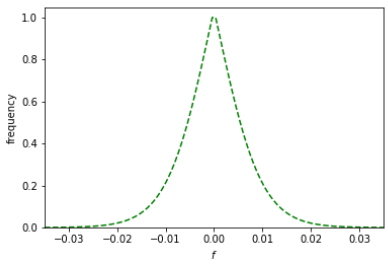
The average frequency of switching for site i is thus $\omega \propto \frac{1}{\tau_i}$. Since the distribution of fitnesses is given by $e^{-\lambda|f_i|}$, the probability distribution of fixation as a function of f is therefore given by $\frac{1}{\tau_i} e^{-\lambda|f_i|}$. This has been plotted in figure 5c and we see that it fits very well with the simulated data.



(a) $n1$ as a function of simulated time



(b) Frequency of fitnesses f that fix in the population



(c) Predicted frequencies of fixation given by $\frac{e^{-\lambda|f_i|}}{\tau_i}$

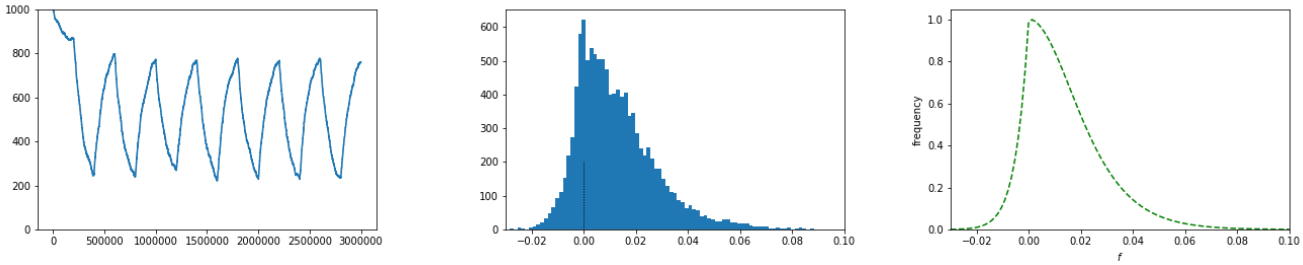
.

Figure 5: Simulation results when drawing a new f_i whenever a mutation occurs.

c)

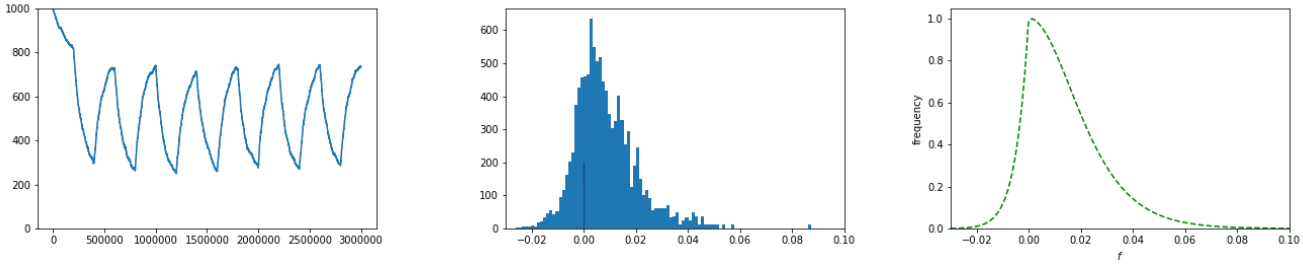
We now let the environment change at a frequency determined by a poisson distribution with rate $\tau = 5 * 10^(-6)$. Upon a change in environment, fitnesses of alleles 1 and 0 reverse such that $f_i^1 < 0$ and $f_i^0 > 0$ for all i. We see in figures 6a and 7a that this leads to oscillations in n_1 as allele 1 becomes alternatingly more and less fit compared to allele 0 at each site.

In the limit of $t \to \infty$ this averages the populations of state 1 and state 2 which in turn leads to the resampling and fixed fitness strategies being equivalent since consecutive fixations can now occur with the same change in fitness Δf for a single site i in the fixed fitness model. This is evident from figures 6 and 8. However, we do note that we still sample fitness space less thoroughly with fixed fitnesses.



(a) $n1$ as a function of simulated time (b) Frequency of fitnesses f that fix in the population (c) Predicted frequencies of fixation given by $e^{-\lambda|f|}p_{fix}(f)$

Figure 6: Simulated and predicted behavior of system with resampling and changing environment



(a) $n1$ as a function of simulated time (b) Frequency of fitnesses f that fix in the population (c) Predicted frequencies of fixation given by $e^{-\lambda|f|}p_{fix}(f)$

Figure 7: Simulated and predicted behavior of system with constant $|f_i|$ and changing environment

In the long time limit of a changing environment, $\langle n_1 \rangle = \langle n_2 \rangle$ where $\langle\rangle$ denotes a time average. The frequency distribution is thus given by equations 4 and 5 with $n_1 = n_2$, giving

$$y \propto exp(-|f|) * \frac{1 - e^{-2f}}{1 - e^{-2Nf}} \tag{7}$$

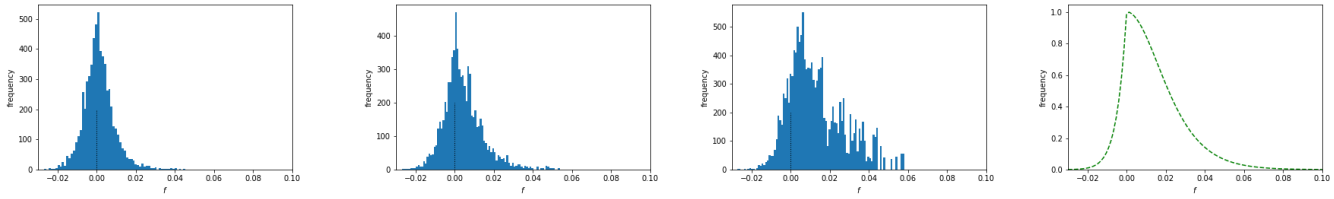
This has been plotted in figures 7c and 6c and we see that it fits well with the data in both cases.

d) As τ becomes large, the frequency of changes in the environment become increasingly frequent, making the time average approximation increasingly accurate. This leads to the distribution of fitnesses that fix becoming increasingly similar to equation 7. Similar results are obtained with resampling. In the present case of constant fitnesses, a better fit to the predicted frequencies would be obtained with an increased number of genes.

This can also be rationalized by looking at the n_1 curves where for small τ we only see a single or a few transitions, whereas for large τ transitions occur very rapidly and the equilibrium state has $n_1 \approx n_2 \approx 500$.

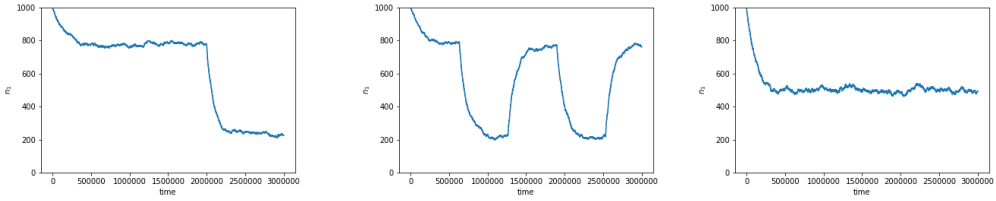
We thus see that as τ increases, the proportion of fixed mutations that are beneficial (i.e. has $f > 0$) increases until it reaches the limiting distribution given by equation 7.

e) *Drosophila* have a relatively short generation time of ≈2 weeks. Changes in the environment due to factors such as changing seasons thus occurs over a timescale that allows for fixation of mutations in small populations within a



(a) $\tau = 10^{-7}$ (b) $\tau = 10^{-6.5}$ (c) $\tau = 10^{-4}$ (d) Predicted

Figure 8: Frequency of fitnesses f that fix in the population for a changing environment with fixed fitnesses $|f|$. As τ increases, this increasingly resembles the predicted distribution resulting from a time average of the populations n_1 and n_0.



(a) $\tau = 10^{-7}$ (b) $\tau = 10^{-6.5}$ (c) $\tau = 10^{-4}$

Figure 9: $n1$ as a function of simulated time for a changing environment with fixed fitnesses $|f|$. As τ increases, oscillations become increasingly fast, allowing us to take a time average of the populations n_1 and n_0.

single season. This scenario is qualitatively similar to the scenario considered above, where e.g. alleles that are more beneficial during the summer can arise and fix during the warmer months while alleles that are more beneficial in the winter can arise and fix during the colder months. As is evident form figure 9b, this will lead to the population having a large proportion of beneficial mutations if the relative timescales of the generation time and change in environment is appropriate.

As is evident from figure 9, this can lead to there being a number of beneficial mutations at any one time, but at a later time these may turn out to be detrimental and thus be cleared from the population. This does not lead to the mean fitness of the population increasing over long timescales, but it does lead to the population being fitter at a given time of year, albeit there will be a delay corresponding to the fixation time of the mutation.

This particular scenario is likely to be more similar to the model with resampling of fitnesses rather than fixed fitnesses since we can imagine there to be a set of mutations $\{1_j\}$ that are more beneficial during the summer and a set of mutations $\{0_j\}$ that are more beneficial during the winter, and while the population can change between these two states, it is unlikely that mutations are exactly reversed. Instead, the population will likely sample genotype space in a more continous manner where mutations in a given gene can have different fitness advantages depending on how it affects the resulting protein and its expression level.

1.3 Ancestral inference

1.4 Neanderthal introgression

Appendix

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 14 11:40:47 2019

@author: kris
"""

import matplotlib.pyplot as plt
import numpy as np
import networkx as nx


def topological_sort(G, relabel = False):
    nodes = [int(n) for n in G.nodes]
    inds = {}
    for i, n in enumerate(nodes): inds[n] = i

    done = False
    while not done:
        done = True
        for i in nodes:
            for j in nodes:
                if j in G.neighbors(i):
                    a = inds[i]
                    b = inds[j]
                    if a > b:
                        inds[i] = b
                        inds[j] = a
                        done = False

    for node, ind in inds.items():
        nodes[ind] = node

    if not relabel: return nodes

    else:
        nodes.reverse()
        for i, n in enumerate(nodes):
            print(n, i)
            G.node[n]['newlabel'] = str(i+1)


        nodes.reverse()
        return nodes, G




def parse_tree(fname = "tree.dat"):
    G = nx.DiGraph()

    f = open(fname, 'r')
    for line in f:
        n, d1, l1, d2, l2 = line.split()
        G.add_node(int(n))
```

```
        G.add_edge(int(n), int(d1), length = float(l1))
        G.add_edge(int(n), int(d2), length = float(l2))
    f.close()

    #topo = topological_sort(G)
    topo, G = topological_sort(G, relabel = True)

    G.root = topo[0]
    G.node[G.root]['rootlength'] = 0
    G.node[G.root]['tier'] = 0


    for node in topo[1:]: #calculate length below root node; start from top
        pred = list(G.predecessors(node))[0]
        G.node[node]['rootlength'] = G[pred][node]['length'] + G.node[pred]['rootlength']
        G.node[node]['tier'] =  G.node[pred]['tier']+1

    topo.reverse() #reverse list

    for node in topo: #calculate number of nodes below each node; start from bottom
        ntot = 0
        nleaves = 0
        nint = 0
        length = 0
        for daughter in G.neighbors(node):
            ntot += G.node[daughter]['ntot']+1 #also add for this node
            nleaves += G.node[daughter]['nleaves']
            nint += G.node[daughter]['nint']
            if len(list(G.neighbors(daughter))) == 0: nleaves += 1 #daughter is a leaf
            else: nint += 1 #daughter is internal


        G.node[node]['ntot'] = ntot #total nodes below this node
        G.node[node]['nleaves'] = nleaves #leaves below this node
        G.node[node]['nint'] = nint #internal nodes below this node

    return G

def print_tree(G, relabel = False):

    nodes = np.array([n for n in G.nodes])
    ntots = np.array([G.node[n]['ntot'] for n in G.nodes])
    args = np.argsort(ntots)

    nodesn = nodes[args]
    ntots = ntots[args]
    #for n in nodesn:
    #    print(n,":", G.nodes[n]['ntot'], ' (', G.nodes[n]['nleaves'],
    #          ',', G.nodes[n]['nint'], ')')

    print('\n')

    if relabel:
        for n in nodesn:
            lab = G.node[n]['newlabel']
            print(lab,"&", n, '&', G.nodes[n]['ntot'], '&', G.nodes[n]['nleaves'],
                  '&', G.nodes[n]['nint'], '&', G.nodes[n]['rootlength'], '\\'+'\\')
```

```
    else:
        for n in nodesn:
            print(n,"&", G.nodes[n]['ntot'], '&', G.nodes[n]['nleaves'],
                  '&', G.nodes[n]['nint'], '&', G.nodes[n]['rootlength'], '\\'+'\\')

    print('\n')

    ls = np.array([G.node[n]['rootlength'] for n in G.nodes])
    args = np.argsort(-ls)
    nodesl = nodes[args]
    #for n in nodesl:
    #    print(n,"&", G.nodes[n]['rootlength'])



G = parse_tree()
#nx.draw(G, with_labels=True)
print_tree(G)
print_tree(G, relabel = True)


#assume that ttwo changes do not occur over the course of a single step

def draw_tree(G, relabel = False):

    #topo = list(nx.topological_sort(G))
    topo = list(topological_sort(G))


    xs = {topo[0]: 0}

    plt.figure()
    if relabel:plt.text(-0.05, 0.08, str(G.node[G.root]['newlabel']))
    else: plt.text(-0.05, 0.08, str(G.root))

    slopes = [0.5, 0.4,0.3,0.2,0.1]

    for node in topo:
        y1 = -G.node[node]['rootlength']
        x1 = xs[node]
        daughters = list(G.neighbors(node))
        if len(daughters) == 1:
            y2 = -G.node[daughters[0]]['rootlength']
            x2 = x1
            xs[daughter[0]] = x2
            plt.plot([x1, x1], [y1, y2], 'b-')
            plt.text(x2+0.1, y2, str(daughters[0]))
        elif len(daughters) == 2:
            newxs = [xs[node]-1, xs[node]+1]
            newxs = [-1, 1]
            for i, daughter in enumerate(daughters):
                y2 = -G.node[daughter]['rootlength']
                slope = -1/(y2+1.5)
                x2 = x1+(y1-y2)*newxs[i]*slope#slopes[G.node[node]['tier']]
                #x2 = newxs[i]
                xs[daughter] = x2
```

```
                plt.plot([x1, x2], [y1, y2], 'b-')
                if relabel: s = G.node[daughter]['newlabel']
                else: s = str(daughter)
                if len(list(G.neighbors(daughter))) == 0:
                    plt.text(x2-0.05, y2-0.5, s)
                elif newxs[i] > 0:
                    plt.text(x2+0.03, y2, s)
                else:
                    plt.text(x2-0.16, y2, s)

    plt.xticks([], [])
    plt.ylim(-9, 0.5)
    if relabel: plt.savefig("figures/tree_relabelled.png")
    else: plt.savefig("figures/tree.png")
    plt.show()

draw_tree(G)
draw_tree(G, relabel = True)
```