Assignment4

March 25, 2024

1 Assignment 4: Solutions

[94]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy.stats import nbinom
from scipy.integrate import solve_ivp
import random
from scipy.special import comb
from scipy.linalg import expm
```

1.1 1. Molecular evolution under the HKY 84

Consider the following ancestor and descendant viral sequences separated by $\Delta t =$ 1year. Assume that the mutation/substitution of nucleotides in this virus occurs according to the HKY84 model.

Ancestor:	G	C	G	C	C	C	A	G	C	C	A	G	G	G	A	G	G	A	C	G
Descendant:	A	A	C	C	T	C	A	G	C	C	C	C	G	A	A	C	G	A	C	G

Part A: What is the empirical stationary distribution $\vec{\pi}$?

There are actually two ways of calculating this depending on how big of an assumption you are willing to make.

Method 1: Here we assume that the 40 sites of data is a lot of data in which case the empirical distribution can be calculated by simply counting the proportion of cases in which an 'A', 'C', 'G' or 'T' is in the data. This is usually the assumption that is made in natural systems because we usually have long sequences.

To figure out the empirical stationary distribution we look at the representation of each nucleotide in the two observed sequences.

The observed frequencies are:

$$\left\{\pi_A = \frac{1}{4}, \pi_C = \frac{2}{5}, \pi_G = \frac{13}{40}, \pi_T = \frac{1}{40}\right\}$$

Method 2: But 40 sites aren't actually that many sites, so to be more accurate we could use a beta distribution. If we were to use the beta distribution independently for estimating the proportion of

'A', 'C', 'G' or 'T' we wouldn't get an answer that summed because the proportion of each is not independent. Just as the beta distribution is related to the binomial distribution, in this case, we need the equivalent of the beta distribution but for the multinomial distribution; this distribution is known as the Dirichlet Distribution

$$\left\{\pi_A = \frac{1}{4}, \pi_C = \frac{17}{44}, \pi_G = \frac{7}{22}, \pi_T = \frac{1}{22}\right\}$$

Using this method we obtain the above-estimated frequencies:

Part B: What is the empirical (aka observed) probability of transitioning between A, C,G, and T in 1 year?

Again we can take two approaches. The first assuming that we have 'a lot' of data. In this case, we simply count the number of observed transitions between states $X \to Y$ and divide it by the number of possible cases in which this could have happened (e.g., the number of sites in state X in the ancestral sequence.

Ordering sites as $\{T, C, A, G\}$ we would have:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{7} & \frac{5}{7} & \frac{1}{7} & \frac{0}{7} \\ \frac{0}{4} & \frac{1}{4} & \frac{3}{4} & \frac{0}{4} \\ \frac{0}{9} & \frac{3}{9} & \frac{3}{9} & \frac{4}{9} \end{bmatrix}$$

For this, we can use the beta distribution. Let's start by considering the probability of going from an A to state $X = \{A, C, G, \text{ or} T\}$ in one year. In this case there are four "observations" of an 'A' in the ancestral sequence. The probability of staying an 'A' occurred 3 times

Now, there is something a little hoki about this matrix (especially the T row) because we didn't account for the fact that no T's were observed. This again would need to be corrected for uncertainty.

Part C: Using the HKY84 model and assuming that at most a single mutation occurs at a given base in a year, what is the transition probability matrix?

One-step Matrix:

$$P_1 = \begin{bmatrix} \pi_A(-\beta) - \alpha\pi_C - \beta\pi_G + 1 & \alpha\pi_C & \pi_A\beta & \beta\pi_G \\ \alpha\pi_T & \pi_A(-\beta) - \beta\pi_G - \alpha\pi_T + 1 & \pi_A\beta & \beta\pi_G \\ \beta\pi_T & \beta\pi_C & -\beta\pi_C - \alpha\pi_G - \beta\pi_T + 1 & \alpha\pi_G \\ \beta\pi_T & \beta\pi_C & \alpha\pi_A & -\alpha\pi_A - \beta\pi_C - \beta\pi_T + 1 \end{bmatrix}$$

1.2 2. Molecular evolution under the GTR model**

Consider a GTR model with the following parameters:

$$\pi_A = 0.27, \pi_C = 0.25, \pi_G = 0.33, \pi_T = 0.15$$
$$a = 0.1, b = 0.2, c = 0.21, d = 0.12, e = 0.23, f = 0.15$$

Part A: In the GTR model with these parameters, what is the stationary distribution?

$$Q = \begin{bmatrix} -0.1225 & 0.025 & 0.066 & 0.0315 \\ 0.027 & -0.1011 & 0.0396 & 0.0345 \\ 0.054 & 0.03 & -0.1065 & 0.0225 \\ 0.0567 & 0.0575 & 0.0495 & -0.1637 \end{bmatrix}$$

[28]:
```
Q2=np.array([[-0.1225,0.025,0.066,0.0315],[0.027,-0.1011,0.0396,0.0345],
             [0.054,0.03,-0.1065,0.0225],[0.0567,0.0575,0.0495,-0.1637]])
print(Q2)
```

```
[[-0.1225  0.025   0.066   0.0315]
 [ 0.027  -0.1011  0.0396  0.0345]
 [ 0.054   0.03   -0.1065  0.0225]
 [ 0.0567  0.0575  0.0495 -0.1637]]
```

[29]:
```
np.sum(Q2, axis=1)
```

[29]:
```
array([ 0.0000000e+00,  6.9388939e-18, -6.9388939e-18,  0.0000000e+00])
```

Find the eigenvectors and eigenvalues. Note that the first eigenvalue is the leading eigenvalue.

[30]:
```
eigenvalues, eigenvectors = np.linalg.eig(Q2)
print(eigenvalues)
```

```
[-2.77555756e-17 -1.25200000e-01 -1.98092752e-01 -1.70507248e-01]
```

[31]:
```
AMtrx=eigenvectors
DMtrx=np.diag(eigenvalues)
AMtrxInv=np.linalg.inv(eigenvectors)
np.round(AMtrx@DMtrx@AMtrxInv,4)
```

[31]:
```
array([[-0.1225,  0.025 ,  0.066 ,  0.0315],
       [ 0.027 , -0.1011,  0.0396,  0.0345],
       [ 0.054 ,  0.03  , -0.1065,  0.0225],
       [ 0.0567,  0.0575,  0.0495, -0.1637]])
```

[32]:
```
np.round(AMtrxInv[0],4)/np.sum(np.round(AMtrxInv[0],4))
```

[32]:
```
array([0.27, 0.25, 0.33, 0.15])
```

So the stationary distribution is:

$$\pi = \begin{bmatrix} 0.27 & 0.25 & 0.33 & 0.15 \end{bmatrix}$$

This checks out with what we would expect given the parameters we put in for π

Part B: Using the above model simulate a random ancestral genome with 50 base pairs.

[47]:
```
def randGenomeGTR(piVec):
    nucleotides = [0, 1, 2, 3] #(0:A, 1:C, 2: T, 3: G)
    nBase=50
    seq = random.choices(nucleotides, weights=piVec,k=nBase)
    return seq
```

[50]:
```
ances=randGenomeGTR([0.27,0.25,0.33,0.15])
np.array(ances)
```

[50]:
```
array([2, 1, 1, 3, 0, 1, 3, 2, 1, 0, 2, 2, 1, 0, 3, 2, 2, 3, 1, 0, 1, 1,
       0, 1, 1, 0, 0, 1, 0, 3, 1, 2, 1, 2, 2, 3, 1, 2, 0, 2, 0, 2, 0, 0,
       0, 1, 0, 1, 0, 0])
```

Part C: Using the genome in part B as your initial condition, simulate molecular evolution in this genome for $t = 10$ units of time. What is the final genome sequence?

[64]:
```
def simGTR(tMax):
    QMtrx=np.array([[-0.1225,0.025,0.066,0.0315],[0.027,-0.1011,0.0396,0.0345],
             [0.054,0.03,-0.1065,0.0225],[0.0567,0.0575,0.0495,-0.1637]])
    ##Simulate a random sequence
    nucleotides = [0, 1, 2, 3] #(0:A, 1:C, 2: T, 3: G)
    nBase=50
    seq = ances
    initSeq=np.array(seq)
    print(f"Initial sequences {initSeq}")
    t=0
    # print(round(t,3))
    print(seq)
    # Calcualte the time until the first event
    rateTot=sum(-QMtrx[seq[i],seq[i]] for i in range(nBase))
    Deltat=np.random.exponential(scale=1/rateTot)
    while t+Deltat <= tMax:
        ## Update time
        t+=Deltat
        # Choose a focal base
        weight1=np.array([-QMtrx[seq[i],seq[i]]/rateTot for i in range(nBase)])
        base=np.random.choice(range(nBase), p=weight1)
        #Choose a nucleotide
        i=seq[base] #current nucleotide at the focal base
        altNeuc=np.array([j for j in range(4) if j != i])
        weight2=np.array([QMtrx[i,j] for j in range(4) if j != i])
        neuc=np.random.choice(altNeuc, p=weight2/np.sum(weight2))
        ## Update Sequence
        seq[base]=neuc
        # print(round(t,3))
        # print(seq)
        # Draw new Delta t
        rateTot=sum(-QMtrx[seq[i],seq[i]] for i in range(nBase))
        Deltat=np.random.exponential(scale=1/rateTot)
    print(f"Final sequences {seq}")
    print(f"Sequence change{initSeq-np.array(seq)}")
    return seq
```

[65]:
```
simGTR(10);
```

```
Initial sequences [1 2 1 2 1 0 2 2 1 2 3 2 2 0 2 0 1 3 1 0 1 1 0 2 1 1 2 2 2 1 3
0 1 2 0 2 3
 3 2 1 1 2 2 2 1 2 2 1 3 2]
[1, 2, 1, 2, 1, 0, 2, 2, 1, 2, 3, 2, 2, 0, 2, 0, 1, 3, 1, 0, 1, 1, 0, 2, 1, 1,
2, 2, 2, 1, 3, 0, 1, 2, 0, 2, 3, 3, 2, 1, 1, 2, 2, 2, 1, 2, 2, 1, 3, 2]
Final sequences [1, 0, 0, 2, 3, 3, 1, 1, 2, 2, 0, 2, 2, 2, 3, 0, 1, 2, 3, 2, 0,
2, 0, 0, 2, 2, 2, 0, 0, 1, 2, 0, 1, 1, 0, 2, 3, 0, 2, 1, 2, 1, 2, 0, 2, 2, 3, 2,
3, 1]
Sequence change[ 0  2  1  0 -2 -3  1  1 -1  0  3  0  0 -2 -1  0  0  1 -2 -2  1
-1  0  2
 -1 -1  0  2  2  0  1  0  0  1  0  0  0  3  0  0 -1  1  0  2 -1  0 -1 -1
  0  1]
```

Part D: (Challenge for 795) Repeat the simulation in Part C 20 times. Plot the number of mutations (an integer counter) over time for each of these replicate runs. The rate at which mutations occur in a genome is known as the molecular clock**, estimate the** molecular clock rate** in this model.**

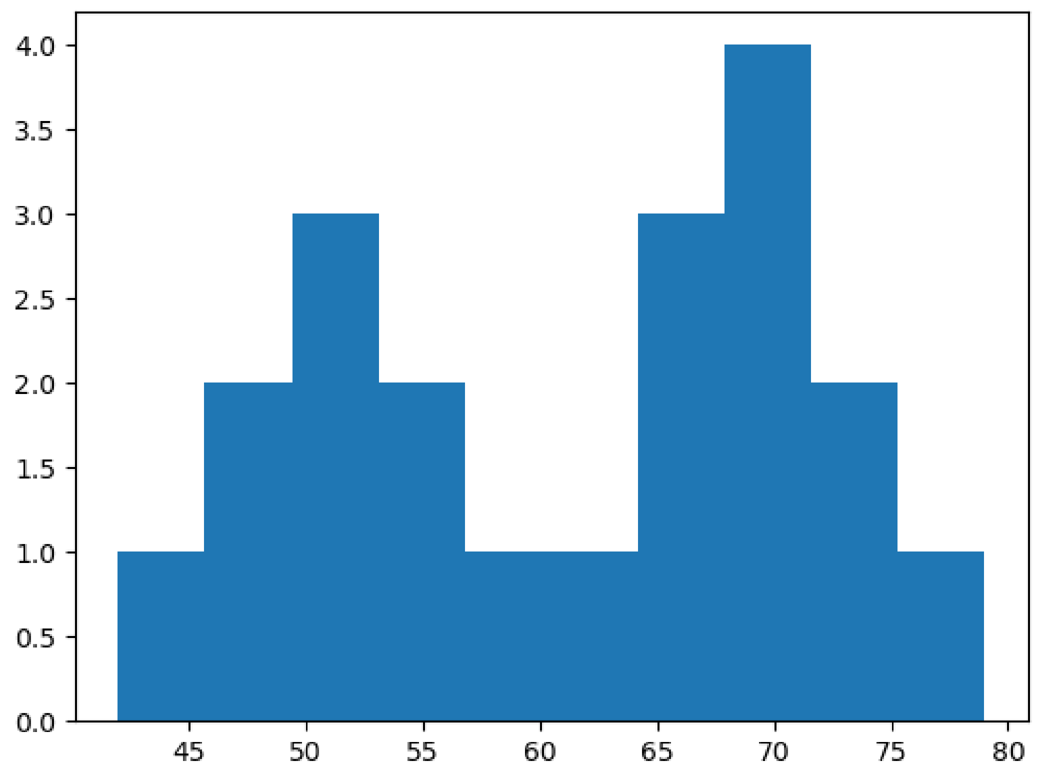
[78]:
```
def simGTR_D(tMax):
    QMtrx=np.array([[-0.1225,0.025,0.066,0.0315],[0.027,-0.1011,0.0396,0.0345],
             [0.054,0.03,-0.1065,0.0225],[0.0567,0.0575,0.0495,-0.1637]])
    ##Simulate a random sequence
    nucleotides = [0, 1, 2, 3] #(0:A, 1:C, 2: T, 3: G)
    nBase=50
    seq = ances
    # initSeq=np.array(seq)
    # print(f"Initial sequences {initSeq}")
    t=0
    # print(round(t,3))
    # print(seq)
    # Calcualte the time until the first event
    rateTot=sum(-QMtrx[seq[i],seq[i]] for i in range(nBase))
    Deltat=np.random.exponential(scale=1/rateTot)
    nMut=0 # Counter for part D
    while t+Deltat <= tMax:
        ## Update time
        t+=Deltat
        # Choose a focal base
        weight1=np.array([-QMtrx[seq[i],seq[i]]/rateTot for i in range(nBase)])
        base=np.random.choice(range(nBase), p=weight1)
        #Choose a nucleotide
        i=seq[base] #current nucleotide at the focal base
        altNeuc=np.array([j for j in range(4) if j != i])
        weight2=np.array([QMtrx[i,j] for j in range(4) if j != i])
        neuc=np.random.choice(altNeuc, p=weight2/np.sum(weight2))
        ## Update Sequence
        seq[base]=neuc
        # print(round(t,3))
        # print(seq)
        # Draw new Delta t
        rateTot=sum(-QMtrx[seq[i],seq[i]] for i in range(nBase))
        Deltat=np.random.exponential(scale=1/rateTot)
        nMut=nMut+1
    # print(f"Final sequences {seq}")
    # print(f"Sequence change{initSeq-np.array(seq)}")
    return nMut
```

[81]:
```
# Create a dictionary to store results
results_dict = {}

# Calculate and save results for specified indices
for index in range(20):
    results_dict[index] = simGTR_D(10)
```

[82]:
```
nMutVec = np.array([results_dict[i] for i in range(20)])
plt.hist(nMutVec)
```

[82]:
```
(array([1., 2., 3., 2., 1., 1., 3., 4., 2., 1.]),
 array([42. , 45.7, 49.4, 53.1, 56.8, 60.5, 64.2, 67.9, 71.6, 75.3, 79. ]),
 <BarContainer object of 10 artists>)
```



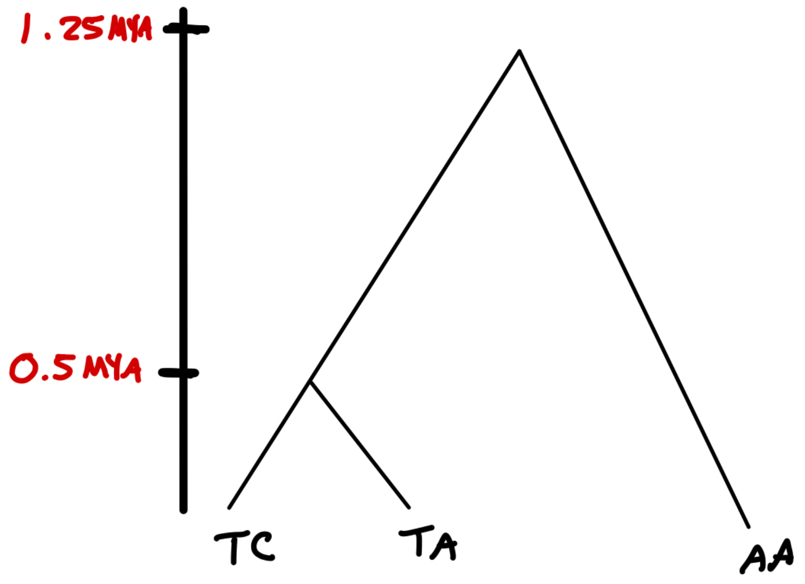
To calculate the molecular clock rate we want to know how many mutations occur per unit time so taking the average we have: 6.115 mut/unit time

```
[88]: np.mean(nMutVec)/10
```

```
[88]: 6.115
```

1.3 3. Tree likelihoods

Consider the following tree topology and corresponding sequences.



Part A: Using a JC69 model with a mutation rate of $\mu = 1.5\frac{\text{mutations}}{\text{Million Years}}$ what is the likelihood of this tree given the molecular model?

Let's start by entering the transition rate matrix and specifying the stationary distribution.

[96]:
```
def QJC69(mu):
    QMtrx=np.array([[-3*mu, mu, mu, mu],
                    [mu,-3*mu, mu, mu],
                    [mu, mu, -3*mu, mu],
                    [mu, mu, mu,-3*mu]])
    return QMtrx
QMtrx=QJC69(1.5)
print(QMtrx)
# Stationary distribution
piVec=[0.25,0.25,0.25,0.25]
```

```
[[-4.5  1.5  1.5  1.5]
 [ 1.5 -4.5  1.5  1.5]
 [ 1.5  1.5 -4.5  1.5]
 [ 1.5  1.5  1.5 -4.5]]
```

From here we have to calculate the transition probability matrix

[100]:
```
def PMtrx(t):
    return expm(QMtrx*t)
print(PMtrx(0.5))
```

```
[[0.2873403  0.23755323 0.23755323 0.23755323]
 [0.23755323 0.2873403  0.23755323 0.23755323]
 [0.23755323 0.23755323 0.2873403  0.23755323]
 [0.23755323 0.23755323 0.23755323 0.2873403 ]]
```

Now we want to calculate the likelihood which is given by:

$$\Pr(\mathcal{D}_1|\mathcal{T},\Theta) * \Pr(\mathcal{D}_2|\mathcal{T},\Theta)$$

where

$$\Pr(\mathcal{D}_1|\mathcal{T},\Theta) = \sum_{Y\in\{A,C,G,T\}} Pr(Y)P_{Y,A}(t_1+t_2) \sum_{X\in\{A,C,G,T\}} P_{X,Y}(t_2)P_{T,X}(t_1)P_{T,X}(t_1)$$

and

$$\Pr(\mathcal{D}_2|\mathcal{T},\Theta) = \sum_{Y\in\{A,C,G,T\}} Pr(Y)P_{Y,A}(t_1+t_2) \sum_{X\in\{A,C,G,T\}} P_{X,Y}(t_2)P_{C,X}(t_1)P_{A,X}(t_1)$$

- we use the following indexes for the nucleotides $\{T:0, C:1, A:2, G:3\}$
- Here $Pr(Y)$ is the probability of having state Y at the root given by the stationary distribution $\Pr(Y) = \pi_Y = 0.25$ assuming that evolution has occurred for a long time before ever having reached the root.

```
[101]: t1=0.5;
       t2=1.25-0.5;
```

```
[103]: Lik1=np.sum([piVec[y]*PMtrx(t1+t2)[2,y]*np.
        ↪sum([PMtrx(t2)[x,y]*PMtrx(t1)[0,x]*PMtrx(t1)[0,x] for x in range(4)]) for y␣
        ↪in range(4)])
       print(Lik1)

       Lik2=np.sum([piVec[y]*PMtrx(t1+t2)[2,y]*np.
        ↪sum([PMtrx(t2)[x,y]*PMtrx(t1)[1,x]*PMtrx(t1)[2,x] for x in range(4)]) for y␣
        ↪in range(4)])
       print(Lik2)

       Lik=Lik1*Lik2
       print(Lik)
```

```
0.015741181472896857
0.015586278580750235
0.0002453464396267147
```

Part B: What is the most likely ancestral sequence at the root?

First note that we can calculate the most likely state at each site seperately. First he first site we want to calculate for each value of y, we want to calculate:

$$\Pr(Y=y|\mathcal{D}_1,\mathcal{T},\Theta) = \frac{\Pr(\mathcal{D}_1|Y=y,\mathcal{T},\Theta)\Pr(Y=y|\mathcal{T},\Theta)}{\Pr(\mathcal{D}_1|\mathcal{T},\Theta)}$$

where

$$\Pr(\mathcal{D}_1|Y=y,\mathcal{T},\Theta) = P_{y,A}(t_1+t_2)\sum_{X\in\{A,C,G,T\}} P_{X,y}(t_2)P_{T,X}(t_1)P_{T,X}(t_1)$$

and where $\Pr(\mathcal{D}_1|\mathcal{T},\Theta)$ is the first site likelihood from above and $\Pr(Y=y|\mathcal{T},\Theta)=\pi_y=0.25$

[113]:
```
def PrD1(y):
    return PMtrx(t1+t2)[2,y]*np.
 ↪sum([PMtrx(t2)[x,y]*PMtrx(t1)[0,x]*PMtrx(t1)[0,x] for x in range(4)])
def PrD2(y):
    return PMtrx(t1+t2)[2,y]*np.
 ↪sum([PMtrx(t2)[x,y]*PMtrx(t1)[1,x]*PMtrx(t1)[2,x] for x in range(4)])
```

[109]:
```
def PrY1(y):
    return PrD1(y)*0.25/Lik1
site1=np.array([PrY1(y) for y in range(4)])
print(site1)
# The most likely state is state 0 (T)
```

```
[0.25072591 0.24957388 0.25012633 0.24957388]
```

[115]:
```
def PrY2(y):
    return PrD2(y)*0.25/Lik2
site2=np.array([PrY2(y) for y in range(4)])
print(site2)
# The most likely state is state 2 (A)
```

```
[0.2495983  0.25012486 0.25067853 0.2495983 ]
```

The most likely ancestral sequence is TA which is also what we would estimate from parsimony. Note that the answer here does not have to agree with parsimony but is most likely to as long as there are not too many mutations which is the case here.

Part C (Challenge question for 795): Plot the tree likelihood for $0.05 < \mu < 0.75$. What is the maximum likelihood estimate for the mutation rate?

[116]:
```
def PMtrx2(mu, t):
    QMtrx=QJC69(mu)
    return expm(QMtrx*t)
```

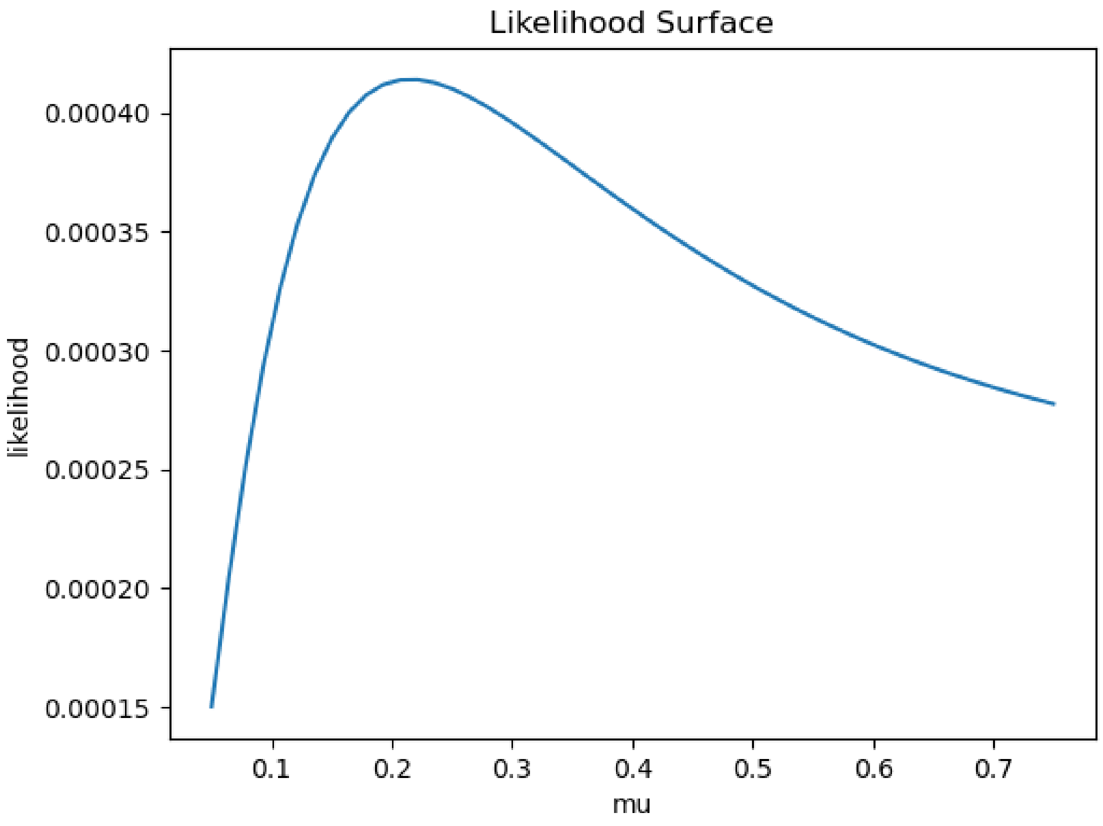
[117]:
```
def treeLike(mu):
    Lik1=np.sum([piVec[y]*PMtrx2(mu,t1+t2)[2,y]*np.
 ↪sum([PMtrx2(mu,t2)[x,y]*PMtrx2(mu,t1)[0,x]*PMtrx2(mu,t1)[0,x] for x in␣
 ↪range(4)]) for y in range(4)])
    Lik2=np.sum([piVec[y]*PMtrx2(mu,t1+t2)[2,y]*np.
 ↪sum([PMtrx2(mu,t2)[x,y]*PMtrx2(mu,1)[1,x]*PMtrx2(mu,t1)[2,x] for x in␣
 ↪range(4)]) for y in range(4)])
    return Lik1*Lik2
```

```
[124]: treeLike(0.5)
```

```
[124]: 0.00032728696360801937
```

```
[134]: # Generate x values
       x_values = np.linspace(0.05, 0.75, 50)  # 100 points between 0.05 and 0.5

       # Generate y values by applying the function to each x value
       y_values = np.array([treeLike(x_values[y]) for y in range(50)])
       # Plot the function
       plt.plot(x_values, y_values)
       plt.title('Likelihood Surface')
       plt.xlabel('mu')
       plt.ylabel('likelihood')
       plt.show()
```



Extracting the maximum value.

```
[138]: max_position = np.argmax(y_values)
       x_values[max_position]
```

```
[138]: 0.22142857142857142
```

The maximum mutation rate is $\mu = 0.221$

1.4 4. Models of mutation

Part A: In this course, we have considered several different models of mutation including models of mutation in the coalescent and models of nucleotide evolution. List these different models and describe their assumptions. Which are used for the coalescent and which are used for phylogenetic trees?

Coalescent Models

1. Infinite sites model: Each mutation occurs at a new site in the genome. The mutation rate is specified at the genome-wide scale. Which mutation occurs is random (equivalent to JC69)
2. Infinite alleles model: Each mutation results in a new allele. The whole sequence is treated as an allele, which 'allele' is created is random.

Phylogenetic Models 1. JC69: A nucleotide model. Each mutation is equally likely to occur. 2. K80: A nucleotide model that specifies a separate transition and transversion rate 3. HKY84: A nucleotide model that is similar to K80 but adjusts the rates to create a given stationary distribution. 4. GTR: A six-parameter model that is the MOST general time reversible model that we can create that also gives a specified stationary distribution.

Part B: Coalescent mutation models are based on the assumption that no two mutations affect the same base in the genome either resulting in different alleles or occurring at different sites. Why might this be a reasonable assumption for a coalescent but not for a phylogenetic tree?

The idea of a phylogenetic model of molecular evolution is to consider the possibility that two mutations occur at the same site and hence that evolution can erase itself. This is important over long evolutionary times when many mutations can occur. Over short periods of time however, as within a population, we do not really need to consider the possibility of the erasure of mutations.