MATH50010 Coursework

Ivan Kirev CID:01738166

17/12/2020

1. (2 marks) For the data given, compute the overall proportion of the four different bases in the sequence.

```
library(seqinr)
dat <- read.fasta("sequences.fasta")
genome = dat[[1]]
# calculate the number of each basis in the genome, then divide by the length of
# the whole sequence to get the proportion
table_1 = count(genome, 1)/getLength(genome)
print(table_1)
```

```
##
##         a         c         g         t
## 0.3052582 0.1995605 0.2005196 0.2946616
```

2. (3 marks) In the DNA double helix, the bases G and C are always bound together on opposite strands. Produce a plot to illustrate how the proportion of the sequence that is a G or a C varies across the genome using a sliding window of size 1000 bases. More precisely,

- choose a region of the genome of length (say) 100000 bases
- For each window of length 1000 bases, compute the proportion of G+C.
- Plot the proportion G+C against position.

```
# select the fist 100000 bases
n = genome[1:1e+05]

# for each window of 1000 bases calculate the propotion of G + C
proportion = c()
for (i in c(1:99000)) {
    proportion = c(proportion, GC(n[i:(i + 1000)]))
}

# plot these proportions agains position
plot(c(1:99000), proportion, xlab = "Position", ylab = "Proportion of G+ C")
```

3. (15 marks) A very naive model of the DNA sequence would be to regard each base pair as being chosen independently of its surroundings.

a) Using the base proportions already calculated, determine the proportions q_{ij} of each pair of consecutive bases $i, j \in \{A, C, G, T\}$ that would be expected under an independence model.

(a) Under the independence model, we would expect that the proportion of each base pair q_{ij} is

$$\frac{(\text{proportion of i}) \cdot (\text{proportion of j})}{\text{total number of bases}}$$

```
# put the count of bases for a, c, g, t in a vector q_0
q_0 = as.vector(table_1)

# calculate each $q_{ij}$
qij = c(q_0[1] * q_0[1], q_0[1] * q_0[2], q_0[1] * q_0[3], q_0[1] * q_0[4], q_0[2] *
    q_0[1], q_0[2] * q_0[2], q_0[2] * q_0[3], q_0[2] * q_0[4], q_0[3] * q_0[1], q_0[3] *
    q_0[2], q_0[3] * q_0[3], q_0[3] * q_0[4], q_0[4] * q_0[1], q_0[4] * q_0[2], q_0[4] *
    q_0[3], q_0[4] * q_0[4])
print(qij)
```

```
##  [1] 0.09318260 0.06091749 0.06121026 0.08994790 0.06091749 0.03982440
##  [7] 0.04001579 0.05880283 0.06121026 0.04001579 0.04020811 0.05908544
## [13] 0.08994790 0.05880283 0.05908544 0.08682549
```

b) Now, we'll assess the strength of the evidence against this simple null model. To do this, you'll need to compute the log likelihood ratio statistic. This compares the (log of) the likelihood under the null model and the more general model where we estimate the probabilities for each pair $i, j \in \{A, C, G, T\}$ as $\widehat{p}_{ij} = \frac{n_{ij}}{n-1}$, where n_{ij} is the number of times the consecutive pair ij is seen, and n is the length of the sequence, so that there are $n - 1$ (overlapping) pairs.

We assume under both hypotheses that pairs are sampled from a multinomial distribution (omitting a multinomial coefficient that cancels in the ratio)

$$L(\widehat{p}_{ij}) \propto \prod_{i,j} \widehat{p}_{ij}^{n_{ij}}, \qquad L(\widehat{q}_{ij}) \propto \prod_{ij} \widehat{q}_{ij}^{n_{ij}},$$

Note that this is not quite right - consecutive pairs overlap, so it is not entirely reasonable to regard the sampling as multinomial. This turns out not to matter - see d).

The log likelihood ratio statistic is then

$$\log L(\widehat{p}_{ij}) - \log L(\widehat{q}_{ij}).$$

Note: when computing the log likelihood ratio statistic, be mindful of the underflow errors - careful use of logs avoids this.

In a sense, the likelihood ratio is a measure of whether the data are better explained by the alternative model or the null model. But we have to make the comparison carefully - under the alternative model we are estimating more free parameters - this gives us more wiggle room to accommodate the data. So even if the data *were* generated under the null model, it is possible that the alternative model would fit better. This line of thinking tells us how to calibrate the test. We generate many data sets from the null model. For each data set, we estimate the parameters for the null model and the alternative model, and compute the likelihood ratio just as we did for our real dataset.

Sure enough, we will find that the likelihood ratio statistic is often greater than one: data simulated under the null model will actually have higher probability under the alternative model. But if the likelihood ratio is much greater than we would expect for data simulated under the null model, then we have evidence against the null hypothesis.

First, we will calculate n_{ij} and $\hat{p}_{ij}$.

```
# calculate the number of each pair in the genome
table_2 = count(genome, 2)
table_2
```

```
##
##    aa    ac    ag    at    ca    cc    cg    ct    ga    gc    gg    gt    ta
## 98508 50762 47920 52021 53053 36674 26743 46450 40885 37134 36758 48926 56765
##    tc    tg    tt
## 38350 52282 93162
```

```
# calculate $n_{ij}$
nij = as.vector(table_2)

# calculate $p_{ij}$
pij = nij/(getLength(genome) - 1)
```

Now we assume that under both hypotheses the pairs are sampld from a multinomial distribution and thus we have

$$L(\hat{p}_{ij}) \propto \prod_{i,j} \hat{p}_{ij}^{n_{ij}}, \; L(\hat{q}_{ij}) \propto \prod_{i,j} \hat{q}_{ij}^{n_{ij}}.$$

The log-likelihood ratio is then

$$\begin{aligned}\frac{\log L(\hat{p}_{ij})}{\log L(\hat{q}_{ij})} = \log \frac{\prod_{i,j} \hat{p}_{ij}^{n_{ij}}}{\prod_{i,j} \hat{q}_{ij}^{n_{ij}}} &= \log \prod_{i,j} \hat{p}_{ij}^{n_{ij}} - \log \prod_{i,j} \hat{q}_{ij}^{n_{ij}} \\ &= \sum_{i,j} \log(\hat{p}_{ij}^{n_{ij}}) - \sum_{i,j} \log(\hat{q}_{ij}^{n_{ij}}) \\ &= \sum_{i,j} \left(n_{ij} \log\left(\hat{p}_{ij}\right) - n_{ij} \log\left(\hat{q}_{ij}\right) \right) \\ &= \sum_{i,j} n_{ij} \left(\log\left(\frac{\hat{p}_{ij}}{\hat{q}_{ij}}\right) \right)\end{aligned}$$

Now we will calculate this log-likelihood ratio for our genome:

```
# set initial value for the sum
log_likelihood = 0

# calculate the sum from above
for (i in 1:length(pij)) {
    log_likelihood = log_likelihood + nij[i] * (log(pij[i]) - log(qij[i]))
}
print(log_likelihood)
```

```
## [1] 15368.99
```

Since this is less than 0, we deduce that $\log L(\hat{p}_{ij}) > \log L(\hat{q}_{ij})$ and thus $L(\hat{p}_{ij}) > L(\hat{q}_{ij})$. The alternative model appears to be much better fit since the log-likelihood ratio is much greater than zero.

c) Use the `permutation` function to generate random permutations of the DNA sequence. These shuffled sequences are samples from the null distribution, because their ordering is entirely random. For each permuted sequence, compute the likelihood ratio as in b) This should allow you to reject the null hypothesis very comfortably: check that this is the case.

First, we will write a function that calculates the log-likelihood ratios of the two models given a genome.

```
# function that returns the log-likelihood ratio
likelihood_function = function(genome) {

    # here q_0 and qij will be the same since they don't change on permtations of the
    # initial genome
    q_0 = as.vector(count(genome, 1)/getLength(genome))
    qij = c(q_0[1] * q_0[1], q_0[1] * q_0[2], q_0[1] * q_0[3], q_0[1] * q_0[4], q_0[2] *
        q_0[1], q_0[2] * q_0[2], q_0[2] * q_0[3], q_0[2] * q_0[4], q_0[3] * q_0[1],
        q_0[3] * q_0[2], q_0[3] * q_0[3], q_0[3] * q_0[4], q_0[4] * q_0[1], q_0[4] *
            q_0[2], q_0[4] * q_0[3], q_0[4] * q_0[4])

    # calculate nij and pij using the same method as before
    nij = as.vector(count(genome, 2))
    pij = nij/(getLength(genome) - 1)

    # calculate the log-likelihood ratio as before
    log_likelihood = 0
    for (i in 1:length(pij)) {
        log_likelihood = log_likelihood + nij[i] * (log(pij[i]/qij[i]))
    }
```

```
    # return log-likelihood ratio
    return(log_likelihood)
}
```

Now we create 100 permutations of the genome and use our functions to calculate the log-likelihood ratio of each permutation.

```
# initialize vector in which we collect the ratios
ratios = c()

# calculate log-likelihood ratio for 100 permutations
for (i in 1:100) {
    new_genome = permutation(genome)
    ratios = c(ratios, likelihood_function((new_genome)))
}
print(ratios)
```

```
##   [1] 6.051795 6.813399 4.272074 3.804028 3.875951 2.503837 4.597311 7.087805
##   [9] 3.934704 4.519134 2.536618 3.619024 2.801176 2.141299 4.744477 6.305542
##  [17] 2.560623 7.204221 6.138319 5.623130 4.175630 2.330783 5.280681 4.818341
##  [25] 7.105737 3.305240 3.634428 4.529051 2.652729 2.541573 4.028865 5.074076
##  [33] 6.749959 5.700949 6.260849 4.852558 2.113358 2.912804 2.178333 4.292846
##  [41] 3.127414 7.561774 4.434568 2.774314 2.830230 4.632277 3.629841 4.696656
##  [49] 9.125857 6.518973 1.582326 1.587667 2.113184 3.242801 4.372613 3.365983
##  [57] 1.517700 5.829628 6.337302 4.491081 6.738937 6.370295 4.540660 6.885055
##  [65] 3.461451 7.683284 8.085949 4.235706 6.797354 1.055281 9.142713 5.440112
##  [73] 4.132337 3.743349 2.352279 3.708991 7.571638 3.215267 2.405198 7.587870
##  [81] 2.161668 9.618295 5.077833 5.038548 4.891905 2.560695 3.523248 3.762335
##  [89] 4.257762 6.229062 3.383610 2.706938 6.633839 3.611940 9.379551 1.684718
##  [97] 4.284028 3.596169 5.266034 3.955552
```

We see that the log-likelihood ratios are greater then zero, so the likelihood ratios are all greater than one and therefore data simulated under the null model will actually have higher probability under the alternative model. However in our original genome the magnitude of the log-likelihood ratio was much greater that we would expect for data simulated under the null model (it was 15368.99, while under the null model our observations are mostly between 1 and 10). Therefore, we can easily reject the null hypothesis.

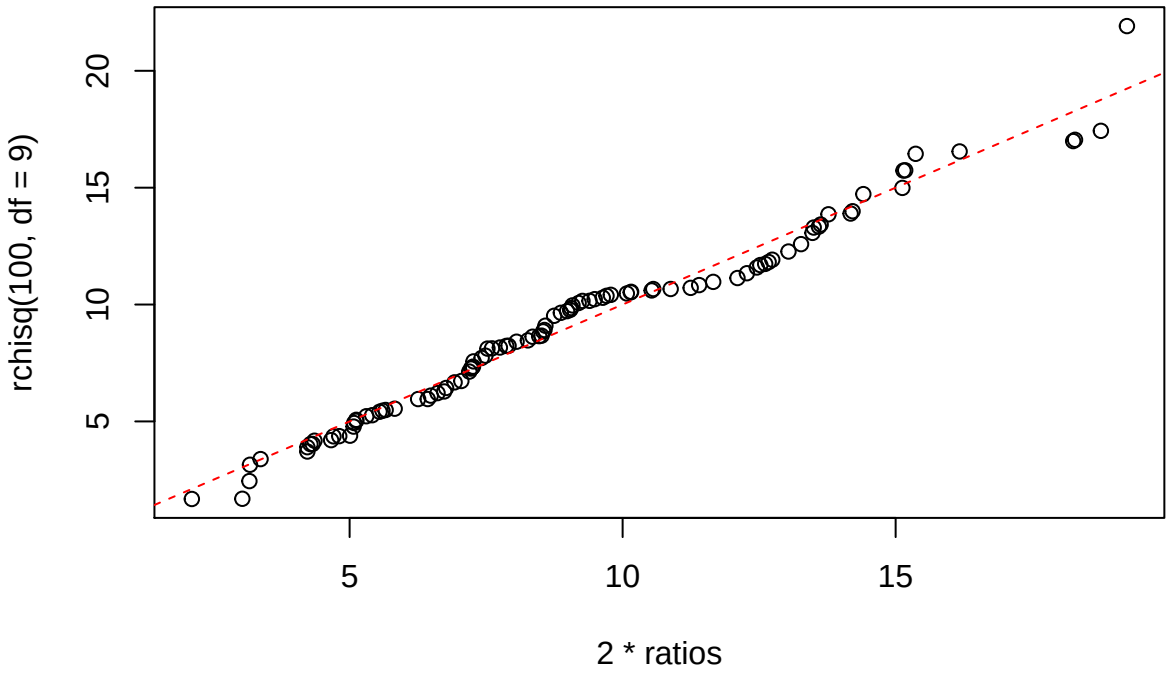
d) When, as here, we have lots of data, the null distribution of the likelihood ratio test statistic has a nice form. *Wilks' theorem* tell us that (twice) the likelihood ratio test statistic should have a chi-square distribution, with r degrees of freedom, where r is the difference between the number of free parameters estimated under the alternative hypothesis and the number of free parameters estimated under the null hypothesis. Use e.g. a QQ-plot to compare the distribution of (twice) the likelihood ratio to a chi-square distribution, specifying the degrees of freedom carefully. Comment on the result.

Under the alternative hypothesis, we have the constraint that the sum over j of each $\hat{p}_{ij}$ equals one for each $i = \{a, c, g, t\}$, so for each of these four groups with sum one, we need to only fix three of them, since the last one will be easily calculated using the constraint. Therefore we need to fix $3 \cdot 4 = 12$ parameters.

With the null hypothesis there are 3 free parameters. This is because we only need to fix three of the parameters q_{ij} to be able to determine uniquely all sixteen parameters.

Hence the difference of free parameters is $12 - 3 = 9$, so using Wilk's theorem we conclude that twice the likelihood ratio test statistic should follow a chi-square distribution with 9 degrees of freedom.

```r
# qqplot
qqplot(2 * ratios, rchisq(100, df = 9))
abline(a = 0, b = 1, col = "red", lty = 2)
```



4. (10 marks) a) Show that if $(x_0, x_1, x_2, \ldots x_n)$ is a realization from the Markov chain with transition matrix P, then the log likelihood for P has the form

$$l(P) = \log \Pr(X_0 = x_0) + \sum_{i,j \in \mathcal{E}} n_{ij} \log p_{ij}.$$

Let $(x_0, x_1, x_2, ..., x_n)$ be a realization from the Markov chain. We can compute its probability as follows:

$$\begin{aligned} P(X_n = x_n, X_{n-1} = x_{n-1}, ..., X_1 = x_1) &= P(X_0 = x_o) \prod_{t=1}^{n} P(X_t = x_t | X_{t-1} = x_{t-1}, X_{t-2} = x_{t-2}, ..., X_1 = x_1) \\ &= P(X_0 = x_o) \prod_{t=1}^{n} P(X_t = x_t | X_{t-1} = x_{t-1}) \end{aligned}$$

where we have used the definition of conditional probability and the Markov Property in the second line. We can rewrite this in terms of the transition probabilities to get that the likelihood of P is

$$L(P) = P(X_0 = x_0) \prod_{t=1}^{n} p_{x_{t-1} x_t} = P(X_0 = x_0) \prod_{i,j} p_{ij}^{n_{ij}}$$

So the log likelihood of P is

$$\log L(P) = \log P(X_0 = x_0) + \log \prod_{i,j} p_{ij}^{n_{ij}} = \log P(X_0 = x_0) + \sum_{i,j} n_{ij} \log(p_{ij})$$

as required.

b) Show that the maximum likelihood estimators are given by $\widehat{p}_{ij} = \frac{n_{ij}}{n_{i\cdot}}$, where

$$n_{i\cdot} = \sum_{j \in \mathcal{E}} n_{ij}.$$

Hint: you know that for each $i \in \mathcal{E}$, $\sum_{j\in\mathcal{E}} p_{ij} = 1$ - *you can respect these constraints using Lagrange multipliers.*

Since i can take four values, we have four constraint equations

$$\sum_j p_{ij} = 1,$$

so we need four Lagrange multipliers $\lambda_1, \lambda_2, \lambda_3, \lambda_4$. The Lagrangian is

$$\mathcal{L} = \log L(P) - \sum_{i=1}^{4} \lambda_i \Big(\sum_j p_{ij} - 1 \Big).$$

Taking the partial derivatives of $\mathcal{L}$ with respect to pij and setting them to be zero, we get

$$\frac{n_{ij}}{p_{ij}} - \lambda_i = 0 \implies p_{ij} = \frac{n_{ij}}{\lambda_i}.$$

Because of the constraint equations we have that $\sum_j p_{ij} = \sum_j \frac{n_{ij}}{\lambda_i} = 1$, so

$$\lambda_i = \sum_j n_{ij}.$$

Hence if we set $n_{i\cdot} = \lambda_i$ we get the desired result that

$$\hat{p}_{ij} = \frac{n_{ij}}{n_{i\cdot}}.$$

c) Compute the maximum likelihood estimates based on the DNA data, assuming the sequence is a first order Markov chain.

```
# first calculate ni, we have repeated each sum four times since our vector needs
# to have 16 entries and each i takes up four of them.
ni = c(rep(sum(nij[1:4]), 4), rep(sum(nij[5:8]), 4), rep(sum(nij[9:12]), 4), rep(sum(nij[13:16]),
    4))

# now we calculate and print the maximum likelihood estimates
phat = nij/ni
print(phat)
```

```
##  [1] 0.3952795 0.2036908 0.1922869 0.2087428 0.3256384 0.2251043 0.1641480
##  [8] 0.2851093 0.2497511 0.2268376 0.2245408 0.2988705 0.2359712 0.1594204
## [15] 0.2173355 0.3872730
```

5. (15 marks)

a) Write code to simulate n independent realizations of length m using the transition probabilities from the data by maximum likelihood.

```
# create the transition matrix from the maximum likelihood data
P = matrix(phat, nrow = 4, byrow = TRUE)
print(P)
```

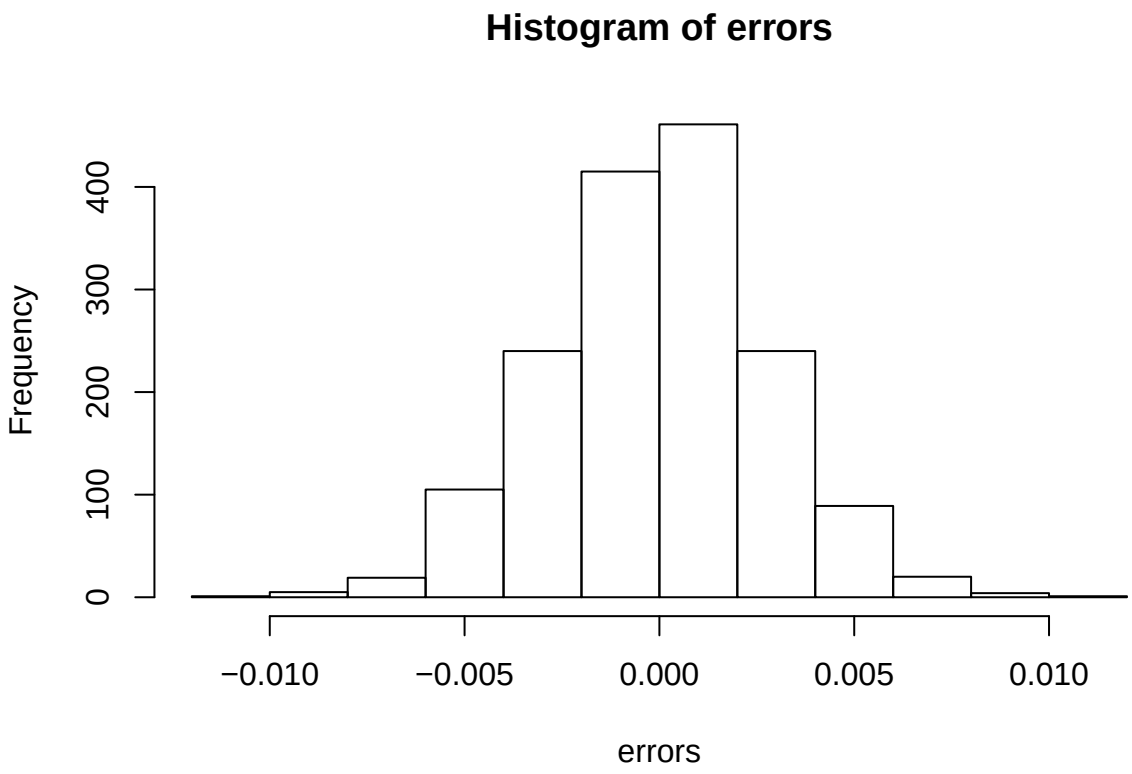
```
##           [,1]      [,2]      [,3]      [,4]
## [1,] 0.3952795 0.2036908 0.1922869 0.2087428
## [2,] 0.3256384 0.2251043 0.1641480 0.2851093
## [3,] 0.2497511 0.2268376 0.2245408 0.2988705
## [4,] 0.2359712 0.1594204 0.2173355 0.3872730
```

```
# create a function that returns a sample of size m
simulation = function(m, initial_value = 1) {
    x = vector(length = m)
    x[1] = initial_value
    for (i in 2:m) {
        x[i] = sample(x = 4, size = 1, prob = P[x[i - 1], ])
    }

    # since our result is a vector with numbers 1,2,3,4, we each number into letters
    # from the genome
    x = plyr::mapvalues(x, from = c(1, 2, 3, 4), to = c("a", "c", "g", "t"))
    return(x)
}
```

```
# create n different samples of size m and store them in a list
m = 1e+05
n = 100
realizations = list(rep(0), n)
for (i in 1:n) {
    realizations[i] = list(simulation(m, initial_value = 1))
}
```

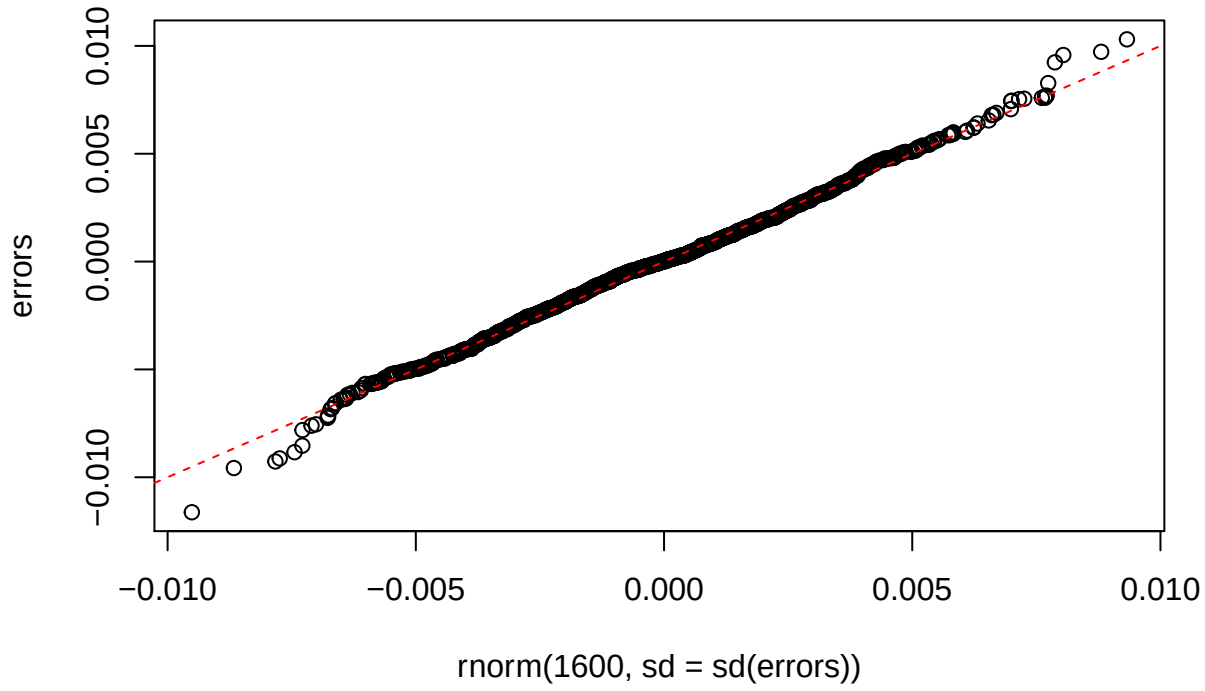
b) For each of the n realizations of the chain, compute the estimates of the transition probabilities. Produce plots to show that the estimators are approximately unbiased, with roughly normal sampling distribution. (You need only plot one entry).

```
# calculate the maximum likelihood estimates for each sample
mle = vector(length = n)
for (i in 1:length(realizations)) {
    newNij = as.vector(count(realizations[[i]], 2))
    newNi = c(rep(sum(newNij[1:4]), 4), rep(sum(newNij[5:8]), 4), rep(sum(newNij[9:12]),
        4), rep(sum(newNij[13:16]), 4))
    mle[i] = list(newNij/newNi)
}

# calculate the difference between the mle for each sample and the mle based on
# the DNA data
error = list(rep(c(0), n))
for (i in 1:n) {
    error[i] = list(mle[[i]] - phat)
}
# create a vector containing all the errors from the 100 samples
errors = c()
for (i in 1:n) {
    errors = c(errors, error[[i]])
}
# create a histogram of all the errors
hist(errors)
```



We can see that the errors are small and seem normally distributed, which indicates that the estimators are unbiased. To further confirm the distribution of the errors, we produce a normal q-q plot.

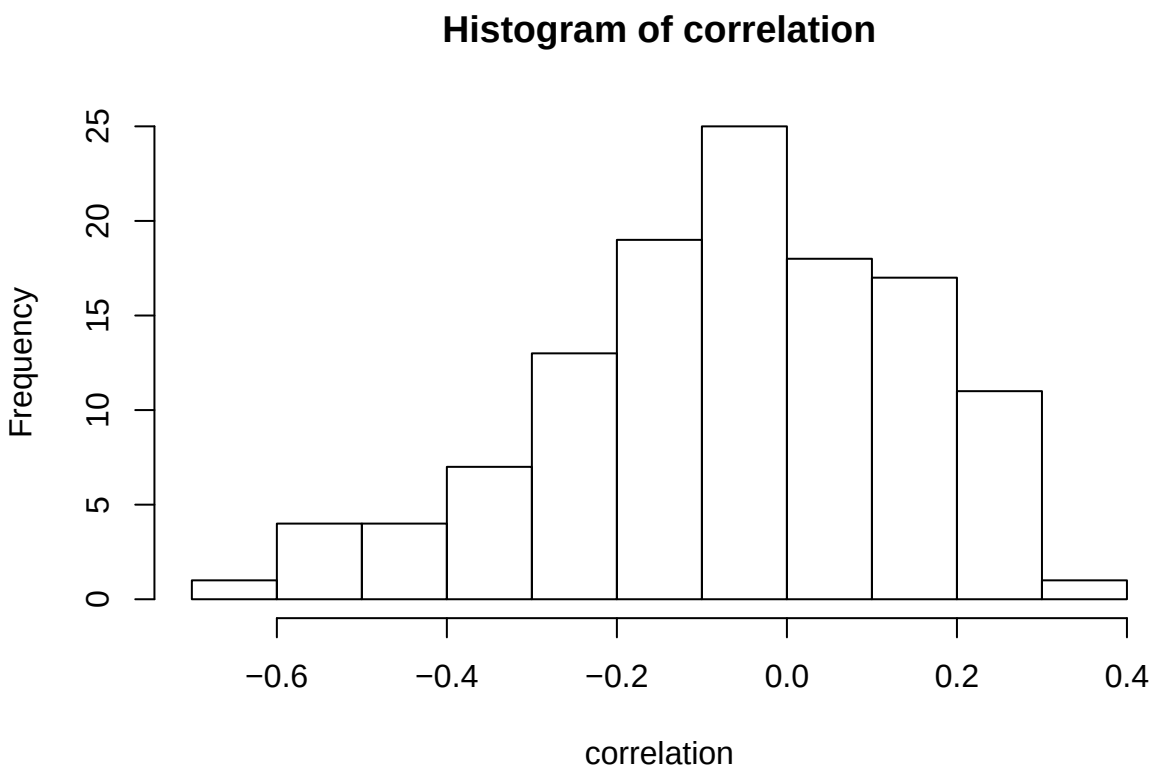
```
# create a qqplot to show that the errors are approximately small and follow a
# normal distribution with standart deviation sd = 0.003

qqplot(rnorm(1600, sd = sd(errors)), errors)
abline(a = 0, b = 1, col = "red", lty = 2)
```



c) Are the estimates of different transition probabilities correlated?

To see if the estimates of different transition probabilities are correlated, we compute the correlation between each two pairs of vectors - one consisting of the ith entries of the 100 realizations (all aa for example) and the other of the jth enties. Thus we have a total of 120 correlations, since we have ommited the cases when i = i (there we will have correlation 1).

```
correlation = c()
for (i in 1:15) {
    for (j in (i + 1):16) {
        vec1 = c()
        vec2 = c()
        for (k in 1:50) {
            vec1 = c(vec1, mle[[k]][i])
            vec2 = c(vec2, mle[[k]][j])
        }
        correlation = c(correlation, cor(vec1, vec2))
    }
}
hist(correlation)
```



We see that most of the correlations are close to zero, which indicate no correlation. In some cases however, we notice a negative correlation of around -0.6. The histogram is skewed to the right which makes sense since the sum for each i of $\sum_j p_{ij}$ is 1, so when one p_{ij} is significantly higher the other three p_{ij}s for that quadruple will go lower, i.e negative correlation will be more common.

6. (5 marks) Suggest (briefly) how the suitability of the Markov model could be evaluated. Are there any obvious incompabitilities between the data and the model? How might the model be improved?

To evaluate the suitability of the Markov model we could consider the proportion of triplets aaa, aac, aag, ect, which should be the products of the proportions of pairs aa, ac, ect, since we have conditional indepence with the Markov model, i.e each outcome depends only on the previous one.
We can check whether this is the case for, lets say the triplet act. Using the count function we can find the proportion of act in the genome:

```
(count(genome, 3)/length(genome))[8]
```

```
##        act
## 0.01854987
```

The proportions of ac and ct are as follows:

```
(count(genome, 2)/length(genome))[2]
```

```
##         ac
## 0.06217831
```

```
(count(genome, 2)/length(genome))[8]
```

```
##         ct
```

```
## 0.05689655
```

But we can see that $0.06217831 \cdot 0.05689655 = 0.003537731$ which is not very close to 0.01854987. Thus the model is not quite correct.
To improve our model we could make a higher order Markov chain - i.e consider when each outcome depends on the previous two or more