Bioinformatica con R — Esercizi

Complemento al testo: "*Bioinformatica con R*" di Crescenzio Gallo, con licenza Creative Commons Attribution 3.0.

Crescenzio Gallo
crescenzio.gallo@unifg.it

Università degli Studi di Foggia

Dipartimenti di Area Medica

25 marzo 2020 00:47

Indice

Q2.6 *Quali sono le tre parole di 4 basi più frequenti (4-meri) nel genoma del batterio* Chlamydia trachomatis *ceppo* D/UW-3/CX *(accession NCBI* NC_000117*) e quante volte si verificano nella sua sequenza?*

Q2.7 *Utilizzare la funzione* dotPlot() *del pacchetto R SeqinR per tracciare un dotplot della fosfoproteina del virus della rabbia e della fosfoproteina del virus* Mokola*, utilizzando una dimensione della finestra di 10 e una soglia di 5.*

Q2.8 *Utilizzare la funzione* makeDotPlot1() *per fare un dotplot della fosfoproteina del virus della rabbia e della fosfoproteina del virus* Mokola*, impostando l'argomento "dotsize" a 0,1.*

Q2.9 *Utilizzare la funzione* dotPlot() *del pacchetto SeqinR per creare un dotplot della fosfoproteina del virus della rabbia e della fosfoproteina del virus* Mokola*, utilizzando una dimensione della finestra di 3 e una soglia di 3. Utilizzare la funzione* makeDotPlot3() *in Appendice per creare lo stesso dotplot, con una dimensione della finestra (x) di 3 e una soglia (y) di 3. I due grafici sono simili o diversi, e si può spiegare perché?*

3. Database di sequenze biologiche

Q3.1 *Quali informazioni sulla sequenza del virus della rabbia (accession NCBI* NC_001542*) si possono ottenere dalle sue annotazioni nella banca dati delle sequenze NCBI?*

Q3.2 *Quante sequenze nucleotidiche delle sequenze del batterio* Chlamydia trachomatis *sono presenti nella banca dati NCBI?*

Q3.3 *Quante sequenze nucleotidiche sono presenti dal batterio* Chlamydia trachomatis *nel database di sequenze NCBI RefSeq?*

Q3.4 *Quante sequenze nucleotidiche sono state presentate all'NCBI da Matthew Berriman?*

Q3.5 *Quante sequenze nucleotidiche dei vermi nematodi sono presenti nel database RefSeq?*

Q3.6 *Quante sequenze nucleotidiche per i geni del collagene dei vermi nematodi sono presenti nella banca dati NCBI?*

Q3.7 *Quante sequenze di mRNA per i geni del collagene dei vermi nematodi sono presenti nel Database NCBI?*

Q3.8 *Quante sequenze proteiche per le proteine del collagene dei vermi nematodi sono presenti nel database NCBI?*

Q3.9 *Qual è l'accession NCBI per il genoma del* Trypanosoma cruzi*?*

Q3.10 *Quante specie di vermi nematodi completamente sequenziati sono rappresentati nel database NCBI del genoma?*

Q3.11 *Quante sequenze proteiche del virus della rabbia sono presenti nel database NCBI delle proteine?*

Q3.12 *Qual è l'accession NCBI per il genoma del virus* Mokola*?*

4. Allineamento a coppie di sequenze

Q4.1 *Qual è il punteggio per l'allineamento globale ottimale tra la proteina* Brugia malayi Vab-3 *e la proteina* Loa loa Vab-3*, quando si utilizza la matrice di punteggio* BLOSUM50*, una penalità di apertura del gap di -10 e una penalità di estensione del gap di -0.5?*

Q4.2 *Utilizzare la funzione* printPairwiseAlignment() *per visualizzare l'allineamento globale ottimale tra la proteina* Brugia malayi Vab-3 *e la proteina* Loa loa Vab-3*, utilizzando la matrice di punteggio* BLOSUM50*, una penalità di apertura del gap di -10 e una penalità di estensione del gap di -0,5.*

Q4.3 *Quale punteggio globale di allineamento si ottiene per le due proteine* Vab-3*, quando si utilizza la matrice di allineamento* BLOSUM62*, una penalità di apertura del gap di -10 e una penalità di estensione del gap di -0,5?*

Q4.4 *Qual è la significatività statistica dell'allineamento globale ottimale per le proteine* Brugia malayi *e* Loa loa Vab-3 *realizzato con la matrice di punteggio* BLOSUM50*, con una penalità di apertura del gap di -10 e una penalità di estensione del gap di -0,5?*

Q4.5 *Qual è il punteggio ottimale di allineamento globale tra la proteina* Brugia malayi Vab-6 *e la proteina* Mycobacterium leprae chorismate lyase*?*

5. Allineamento multipli e alberi filogenetici

Q5.1 *Calcolare un albero filogenetico "unrooted" con bootstrap, utilizzando il metodo dell'evoluzione minima.*

Q5.2 *Calcolare le distanze genetiche tra le seguenti proteine* NS1 *di diversi ceppi di virus* Dengue*: proteina* NS1 virus 1 *(UniProt* Q9YRRR4*), proteina* NS1 virus 2 *(UniProt* Q9YP96*), proteina* NS1 virus 3 *(UniProt* AAB52248*) e proteina* NS1 virus 4 *(UniProt* Q6TFL5*). Quali sono le proteine più strettamente correlate e quali sono le meno correlate, in base alle distanze genetiche?*

Q5.3 *Costruire un albero filogenetico "unrooted" delle proteine* NS1 *dei virus* Dengue 1, 2, 3 *e* 4*, utilizzando l'algoritmo Neighbour-Joining. Quali sono le proteine più strettamente correlate, in base all'albero? In base ai valori di bootstrap nell'albero, quanto siamo sicuri del risultato?*

Q5.4 *Costruire un albero filogenetico "unrooted" delle proteine* NS1 *dei virus* Dengue 1-4*, basato su un allineamento filtrato delle quattro proteine (mantenendo colonne di allineamento in cui almeno il 30% delle lettere non sono vuote e in cui almeno il 30% delle coppie di lettere sono identiche). Questo differisce dall'albero basato sull'allineamento non filtrato (nel quesito* Q5.3*)? Si può spiegarne il motivo?*

Q5.5 *Costruire un albero filogenetico "rooted" delle proteine* NS1 *del virus* Dengue *basato su un allineamento filtrato, utilizzando la proteina del virus* Zika *come outgroup. Quali sono le proteine del virus* Dengue *più strettamente correlate, in base all'albero? Quali sono le informazioni aggiuntive che questo albero fornisce, rispetto all'albero "unrooted" del quesito* Q5.3*?*

6. Ricerca genica (*gene-finding*) computazionale

Q6.1 *Quanti ORF sono presenti sul forward strand del genoma del virus* DEN-1 Dengue *(accession NCBI* NC_001477*)?*

Q6.2 *Quali sono le coordinate dell'ORF più a destra (nella direzione 3', cioè l'ultimo) nella forward strand del genoma del virus* Dengue DEN-1*?*

Q6.3 *Qual è la sequenza proteica predetta dell'ORF più a destra (nella direzione 3', cioè l'ultimo) nel forward strand del genoma del virus* Dengue DEN-1*?*

Q6.4 *Quanti ORF ci sono di 30 nucleotidi o più nel forward strand della sequenza del genoma del virus* Dengue DEN-1*?*

Q6.5 *Quanti ORF più lunghi di 248 nucleotidi sono presenti nel forward strand della sequenza genomica del virus* Dengue DEN-1*?*

Q6.6 *Se un ORF è lungo 248 nucleotidi, quale sarà la lunghezza in aminoacidi della sua sequenza proteica predetta?*

Q6.7 *Quanti ORF ci sono sul forward strand del genoma del virus della rabbia (accession NCBI* NC_001542*)?*

Q6.8 *Qual è la lunghezza del più lungo ORF tra il 99% degli ORF più lunghi in 10 sequenze casuali della stessa lunghezza e composizione della sequenza del genoma del virus della rabbia?*

Q6.9 *Quanti ORF ci sono nel genoma del virus della rabbia che sono più lunghi della soglia di lunghezza trovata nel quesito Q6.8?*

7. Genomica comparativa

Q7.1 *Quanti geni di* Mycobacterium ulcerans *sono presenti nella versione attuale della banca dati Ensembl Bacteria?*

Q7.2 *Quanti geni codificanti le proteine della* Leishmania major *hanno ortologhi nel* Plasmodium falciparum*?*

Q7.3 *Quanti geni codificanti le proteine della* Leishmania major *hanno ortologhi di tipo uno-a-uno nel* Plasmodium falciparum*?*

Q7.4 *Quanti geni della* Leishmania major *hanno domini Pfam?*

Q7.5 *Quali sono i 5 domini Pfam più comuni nei geni della* Leishmania major*?*

Q7.6 *Quante copie di ciascuno dei 5 domini Pfam più comuni nei geni della* L. major *sono presenti nel* P. falciparum*?*

Q7.7 *Quante copie di ciascuno dei 5 domini Pfam più comuni nei geni del* P. falciparum *sono presenti nella* L. major*?*

8. Modelli nascosti di Markov (*Hidden Markov Models*)

Q8.1 *Qual è la probabilità della sequenza del genoma mitocondriale del* Brugia malayi *(accession NCBI* NC_004298*), secondo un modello multinomiale in cui le probabilità di A, C, G e T (p_A, p_C, p_G e p_T) sono uguali alla frazione di A, C, G e T del genoma mitocondriale dello* Schistosoma mansoni*?*

Q8.2 *Generare una sequenza di DNA casuale lunga* n *basi utilizzando un modello multinomiale in cui le probabilità p_A, p_C, p_G e p_T sono impostate pari alla frazione di A, C, G e T nel genoma mitocondriale dello* Schistosoma mansoni.

Q8.3 *Utilizzare la funzione* generateSeqWithMultinomialModel() *per calcolare una sequenza casuale di 20 lettere, utilizzando un modello multinomiale con p_A=0,28, p_C=0,21, p_G=0,22 e p_T=0,29.*

Q8.4 *Creare una matrice di transizione per un modello di Markov di una sequenza DNA generando le probabilità a piacere per ciascun nucleotide "precedente".*

Q8.5 *Usare la funzione* generatemarkovseq() *(vedi Appendice testo di riferimento) per generare una sequenza di 20 nucleotidi usando il modello di Markov descritto nella matrice di transizione tr_matrix del quesito precedente, usando le probabilità di partenza Π_A = 0,2; Π_C = 0,3; Π_G = 0,2; Π_T = 0,3).*

Q8.6 *Definire una matrice di transizione per un modello Hidden Markov basato sui due stati* "AT" *e* "GC"*, con probabilità p_{AT} = (0,68; 0,32) di cambiare stato se il precedente era* "AT"*, p_{GC} = (0,11; 0,89) di cambiare stato se il precedente era* "GC".

Q8.7 *Definire una matrice di emissione per lo stesso HMM precedente che contiene le probabilità di scegliere i quattro nucleotidi A/C/G/T p_{A_AT} = 0,4, p_{C_AT} = 0,1, p_{G_AT} = 0,1 e p_{T_AT} = 0,4 per lo stato* "AT"*, e p_{A_GC} = 0,1, p_{C_GC} = 0,4, p_{G_GC} = 0,4 e p_{T_GC} = 0,1 per lo stato* "GC".

Q8.8 *Usare la funzione* generatehmmseq() *per generare una sequenza di 10 nucleotidi utilizzando l'HMM del quesito precedente e probabilità di partenza uniformi (cioè Π_{Low} = Π_{High} = 0,5).*

Q8.9 *Date le matrici di transizione e di emissione dell'HMM precedente, e la sequenza di DNA* "TACCGATACGCGATGCTAGC"*, utilizzare la funzione* viterbi() *per verificare lo stato dell'HMM che più probabilmente ha generato il nucleotide in ogni posizione della sequenza.*

9. Grafi di interazione proteica

Q9.1 *Quanti vertici (proteine) e connessioni (interazioni proteina-proteina) ci sono nel grafo di de Lichtenberg? Vi sono circa 6.600 geni previsti nel genoma* S. cerevisiae*: è lo stesso numero di vertici nel grafo dei dati di de Lichtenberg? In caso contrario, spiegarne il motivo.*

Q9.2 *Qual è il numero minimo, massimo e medio di interazioni per le proteine nel grafo dei dati di de Lichtenberg? Si rileva un esempio di proteina hub? Fare un istogramma del numero di interazioni per le proteine* S. cerevisiae *nel dataset di de Lichtenberg.*

Q9.3 *Realizzare un grafo random con lo stesso numero di vertici e connessioni del grafo dei dati di de Lichtenberg. Qual è il grado minimo, massimo e medio dei vertici del grafo random? C'è una differenza nel grado minimo, massimo e medio dei vertici del grafo random rispetto all'altro grafo? Confrontare l'istogramma della distribuzione dei gradi per i due grafi.*

Q9.4 *Quante componenti connesse esistono nel grafo dei dati di de Lichtenberg? Quante componenti connesse contengono solo 2 proteine? Fare un grafico della componente connessa più grande del grafo del dataset di de Lichtenberg.*

Q9.5 *Con quali proteine interagisce la proteina del lievito* YPR119W*, nel dataset di de Lichtenberg? Tracciare un grafico della componente connessa a cui appartiene la proteina* YPR119W*: si riesce a vedere* YPR119W *nel grafico? Tracciare le comunità in questa componente connessa: a quali comunità appartiene* YPR119W*? A quali complessi proteici potrebbe appartenere la proteina* YPR119W*? Si riesce a trovare qualcosa sulla natura delle interazioni tra* YPR119W *e le proteine con cui interagisce?*

10. Estrazione delle caratteristiche strutturali delle proteine

Q10.1 *Reperire l'accession number della proteina* Myoglobin *(mioglobina umana) e dire quanti atomi compongono la proteina; cercare inoltre mediante BLAST le eventuali proteine simili.*

11. Analisi di dati da microarray

Q11.1 *Eseguire la normalizzazione RMA (Robust Multichip Average) sul dataset* GSE1297 *della banca dati Gene Expression Omnibus (GEO).*

1. Analisi statistica delle sequenze biologiche — Parte I

Q1.1 ***Quali sono gli ultimi venti nucleotidi della sequenza genomica del virus*** **Dengue*****?***

Per rispondere a questa domanda, è necessario prima installare il pacchetto R *seqinr* e scaricare la sequenza del genoma *Dengue* DEN-1 dal database NCBI (*accession* NC_001477) e salvarla come file "den1.fasta" nella directory di lavoro. Quindi, per trovare la lunghezza della sequenza del genoma del virus, possiamo digitare nella console R:

```
> library("seqinr")
> dengue <- read.fasta(file="den1.fasta")
> denguеseq <- dengue[[1]]
> length(denguеseq)
[1] 10735
```

Questo ci dice che la lunghezza della sequenza è di 10735 nucleotidi. Pertanto, gli ultimi 20 nucleotidi sono da 10.716 a 10.735. È possibile estrarre la sequenza di questi nucleotidi digitando:

```
> denguеseq[10716:10735]
[1] "c" "t" "g" "t" "t" "g" "a" "a" "t" "c" "a" "a" "c" "a" "g" "g" "t" "t" "c" "t"
```

Q1.2 ***Qual è la lunghezza in nucleotidi della sequenza genomica del ceppo*** **TN** ***del batterio*** **Mycobacterium leprae** ***(accession*** **NC_002677*****)?***

Nota: *Mycobacterium leprae* è un batterio responsabile della lebbra, classificata dall'OMS come una malattia tropicale trascurata. Poiché la sequenza del genoma è una sequenza di DNA, se si sta recuperando la sua sequenza attraverso il sito web del NCBI, sarà necessario cercarla nel database NCBI dei nucleotidi.

Per rispondere a questa domanda, è necessario prima di tutto recuperare la sequenza del genoma del *Mycobacterium leprae TN* dalla banca dati NCBI o, in alternativa, utilizzando la funzione `retrieventrezseqs()` in R. Quindi, possiamo ottenere la lunghezza 3.268.203 della sequenza del genoma del *Mycobacterium leprae TN* dal vettore *lepraseq*:

```
> leprae <- retrieventrezseqs("NC_002677","nucleotide")
[1] "Retrieving sequence NC_002677 ..."
> lepraeseq <- leprae[[1]]
> length(lepraeseq)
[1] 3268203
```

Q1.3 ***Quanti dei quattro nucleotidi A, C, T e G, e di ogni altro simbolo, sono presenti nella sequenza del genoma del batterio*** **Mycobacterium leprae TN*****?***

Nota: Altri simboli oltre ai quattro nucleotidi A/C/T/G possono apparire in una sequenza. Essi corrispondono a posizioni nella sequenza che non sono chiaramente una base e che sono dovute alle incertezze di sequenziamento. Ad esempio, il simbolo

"N" significa "aNy base", mentre "R" significa "A oppure G" (puRine). La tabella di simboli è consultabile su https://www.bioinformatics.org/sms/iupac.html.

Supponendo che la sequenza del batterio richiesta sia già disponibile nel vettore *lepraeseq*, basta digitare:

```
> table(lepraeseq)
lepraeseq
     A      C      G      T
687041 938713 950202 692247
```

Q1.4 ***Qual è il contenuto GC della sequenza genomica del*** **Mycobacterium leprae TN*****, quando (a) tutti i nucleotidi non-A/C/T/G sono inclusi, (b) i nucleotidi non-A/C/T/G sono scartati?***

Suggerimento: Guardare la pagina di aiuto della funzione GC() per scoprire come si comporta con i nucleotidi non-A/C/T/T/G (parametro "exact").

```
> GC(lepraeseq)
[1] 0.5779675
> GC(lepraeseq, exact=FALSE)
[1] 0.5779675
```

La funzione `GC()` dà come risultato 0,5779675 (cioè 57,79675%). Questo è il contenuto di GC sia quando i nucleotidi non-A/C/T/T/G sono presi in considerazione che quando non lo sono (essendovi solo basi canoniche nella sequenza).

La lunghezza della sequenza *M. leprae* è 3.268.203 bp, con 687.041 A, 938.713 C, 950.202 G e 692.247 T. Quindi, per calcolare il contenuto di GC considerando solo le A, C, T e G, possiamo anche digitare:

```
> (938713+950202)/(938713+950202+687041+692247)
[1] 0.5779675
```

Q1.5 ***Quanti dei quattro nucleotidi A, C, T e G sono presenti nel complemento della sequenza del genoma del*** **Mycobacterium leprae TN*****?***

Suggerimento: È necessario prima cercare la funzione per calcolare il complemento di una sequenza. Quindi, ricordarsi di usare la funzione help() per scoprire quali sono gli argomenti e il risultato di tale funzione. Come si comporta la funzione con i simboli diversi dai quattro nucleotidi A, C, T e G? I numeri di A, C, T e G nella sequenza complementare sono quelli che ci aspetteremmo?

Per prima cosa è necessario cercare una funzione per calcolare il complemento inverso, ad es. digitando: `help.search("complement")`. Troveremo che esiste la funzione `seqinr::comp()` che complementa una sequenza di acido nucleico. Ciò significa che è una funzione del pacchetto SeqinR.

Scopriamo come usare questa funzione digitando: `help("comp")`. L'aiuto dice: "I valori non definiti vengono restituiti come NA". Ciò significa che il complemento di simboli non-A/C/T/T/G sarà restituito come NA.

Per trovare il numero di A, C, T e G nel complemento inverso sella sequenza digitiamo quindi:

```
> complepraeseq <- comp(lepraeseq)
> table(complepraeseq)
complepraeseq
     a      c      g      t
692247 950202 938713 687041
```

Si noti che nella sequenza *M. leprae* abbiamo 687.041 A (mentre nel complemento abbiamo 687.041 T), 938.713 C (mentre nel complemento abbiamo 938.713 G), 950.202 G (mentre nel complemento abbiamo 950.202 C) e 692.247 T (mentre nel complemento abbiamo 692.247 A).

Q1.6 ***Quante occorrenze delle parole DNA "CC", "CG" e "GC" si verificano nella sequenza genomica del* Mycobacterium leprae TN*?***

```
> seqinr::count(tolower(lepraeseq), 2)

    aa     ac     ag     at     ca     cc     cg     ct     ga     gc     gg
149718 206961 170846 159516 224666 236971 306986 170089 203397 293261 243071
    gt     ta     tc     tg     tt
210473 109259 201520 229299 152169
```

Abbiamo indicato espressamente il pacchetto (*seqinr*) dal quale utilizzare la funzione `count()`; inoltre abbiamo forzato la sequenza in lettere minuscole, così come richiesto dalla funzione.

Il conteggio richiesto è: 236.971 CC, 306.986 CG, 293.261 GC.

Q1.7 ***Quante occorrenze delle parole DNA "CC", "CG" e "GC" si verificano nei (a) primi 1.000 e (b) ultimi 1.000 nucleotidi della sequenza del genoma Mycobacterium leprae TN?***

Il comando: `length(lepraeseq)` dà come risultato la lunghezza della sequenza, cioè 3.268.203 bp. Quindi, i primi 1.000 nucleotidi avranno gli indici nel range 1:1.000, mentre gli ultimi 1.000 avranno gli indici nel range (length(lepraeseq)-1.000+1):(length(lepraeseq)), cioè 3.267.204-3.268.203:

```
> count(lepraeseq[1:1000],2)
aa ac ag at ca cc cg ct ga gc gg gt ta tc tg tt
78 95 51 49 85 82 92 54 68 63 39 43 42 73 31 54
> count(lepraeseq[(length(lepraeseq)-1000+1):(length(lepraeseq))],2)
aa ac ag at ca cc cg ct ga gc gg gt ta tc tg tt
70 85 44 55 94 81 87 50 53 75 49 51 36 72 48 49
```

Per verificare che le sequenze che abbiamo esaminato siano lunghe 1.000 nucleotidi, possiamo digitare:

```
> length(lepraeseq[1:1000])
[1] 1000
> length(lepraeseq[3267204:3268203])
[1] 1000
```

Q1.8 ***Qual è la lunghezza (in numero totale di coppie di basi) del genoma mitocondriale dello*** **Schistosoma mansoni** ***(accession NCBI*** **NC_002545*****), e quanti A, C, G e T contiene?***

Nota: *Schistmosoma mansoni* è un verme parassita responsabile della causa della schistosomiasi, classificata dall'OMS come malattia tropicale trascurata.

Per rispondere alla prima parte della domanda, recuperiamo innanzitutto in una variabile vettore *seq* la sequenza del genoma mitocondriale dello *Schistosoma mansoni* tramite la funzione `retrieventrezseqs()` e quindi calcoliamo la lunghezza della sequenza dal vettore *seq*:

```
> seqs <- retrieventrezseqs("NC_002545","nucleotide")
[1] "Retrieving sequence NC_002545 ..."
> seq <- seqs[[1]]
> length(seq)
[1] 14415
```

Il genoma mitocondriale dello *Schistosoma mansoni* è lungo 14.415 bp.

Per calcolare la quantità di ciascun nucleotide A/C/G/T basta usare la funzione `table()`:

```
> table(seq)
seq
   A    C    G    T
3654 1228 3307 6226
```

Vi sono 3645 A, 1228 C, 3307 G e 6226 T.

Q1.9 ***Qual è la lunghezza del genoma mitocondriale del*** **Brugia malayi** ***(accession NCBI*** **NC_004298*****), e quanti A, C, G e T contiene?***

Nota: Il *Brugia malayi* è un verme parassita responsabile della causa della filariosi linfatica, classificata dall'OMS come malattia tropicale trascurata.

Per rispondere alla prima parte della domanda, recuperiamo innanzitutto in una variabile vettore *seq* la sequenza del genoma mitocondriale del *Brugia malayi* tramite la funzione `retrieventrezseqs()` e quindi calcoliamo la lunghezza della sequenza dal vettore *seq*:

```
> seqs <- retrieventrezseqs("NC_004298","nucleotide")
[1] "Retrieving sequence NC_004298 ..."
> seq <- seqs[[1]]
> length(seq)
[1] 13657
```

Il genoma mitocondriale del *Brugia malayi* è lungo 13.657 bp.

Per calcolare la quantità di ciascun nucleotide A/C/G/T basta usare la funzione `table()`:

```
> table(seq)
seq
   A    C    G    T
2950 1054 2297 7356
```

Vi sono 2950 A, 1054 C, 2297 G e 7356 T.

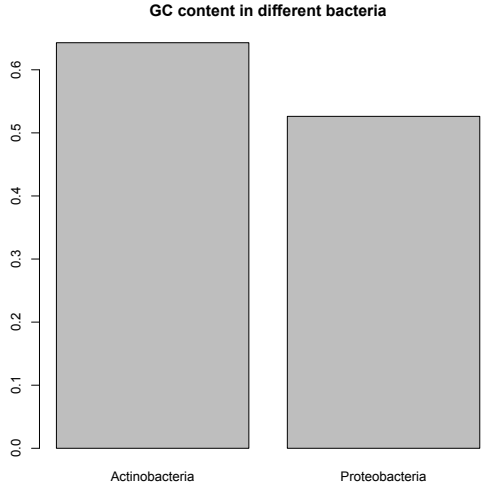
Q1.10 ***Calcolare e confrontare la frazione di GC del*** **Mycobacterium tubercolosis** ***(*****Actinobacterium*****; accession NCBI*** **JX303316*****) e dell'*****Escherichia coli** ***(*****Proteobacterium*****; accession NCBI*** **JN707683*****).***

Ritroviamo per prima cosa le sequenze nucleotidiche richieste per i due organismi:

```
> myActino <- retrieventrezseqs("JX303316","nucleotide")
[1] "Retrieving sequence JX303316 ..."
[1] "   ... found sequence for JX303316 with ID 405113461"
> table(myActino)
myActino
   A    C    G    T
 679 1084 1190  584
> myProteo <- retrieventrezseqs("JN707683","nucleotide")
[1] "Retrieving sequence JN707683 ..."
[1] "   ... found sequence for JN707683 with ID 375162523"
> table(myProteo)
myProteo
   A    C    G    T
1017 1033 1087  892
```

Usiamo quindi la funzione `seqinr::GC()` per calcolare la frazione di GC nel *Mycobacterium tubercolosis* e nell'*Escherichia coli*; creiamo e visualizziamo infine un semplice grafico a barre per il confronto del contenuto di GC nelle due sequenze:

```
> GCActino <- GC(myActino[[1]])
[1] 0.6429177
> rho(tolower(myActino), wordsize = 2)
       aa        ac        ag        at        ca        cc        cg        ct
1.0052859 1.1151847 0.8494599 1.0795944 1.0526959 0.8641342 1.1821127 0.8215482
       ga        gc        gg        gt        ta        tc        tg        tt
1.2741898 0.9325251 0.8119837 1.1912802 0.3390462 1.2574718 1.2167349 0.8506404
> GCProteo <- GC(myProteo[[1]])
[1] 0.5253227
> rho(tolower(myProteo), wordsize = 2)
       aa        ac        ag        at        ca        cc        cg        ct
1.2819145 1.0395684 0.8165872 0.8529440 0.8247499 0.8837314 1.2382100 1.0452924
       ga        gc        gg        gt        ta        tc        tg        tt
1.1228075 0.9546778 0.7298939 1.2427445 0.7285563 1.1458854 1.2635262 0.8205223
> barplot(c(Actinobacteria=GCActino, Proteobacteria=GCProteo), main="GC content in
different bacteria")
```



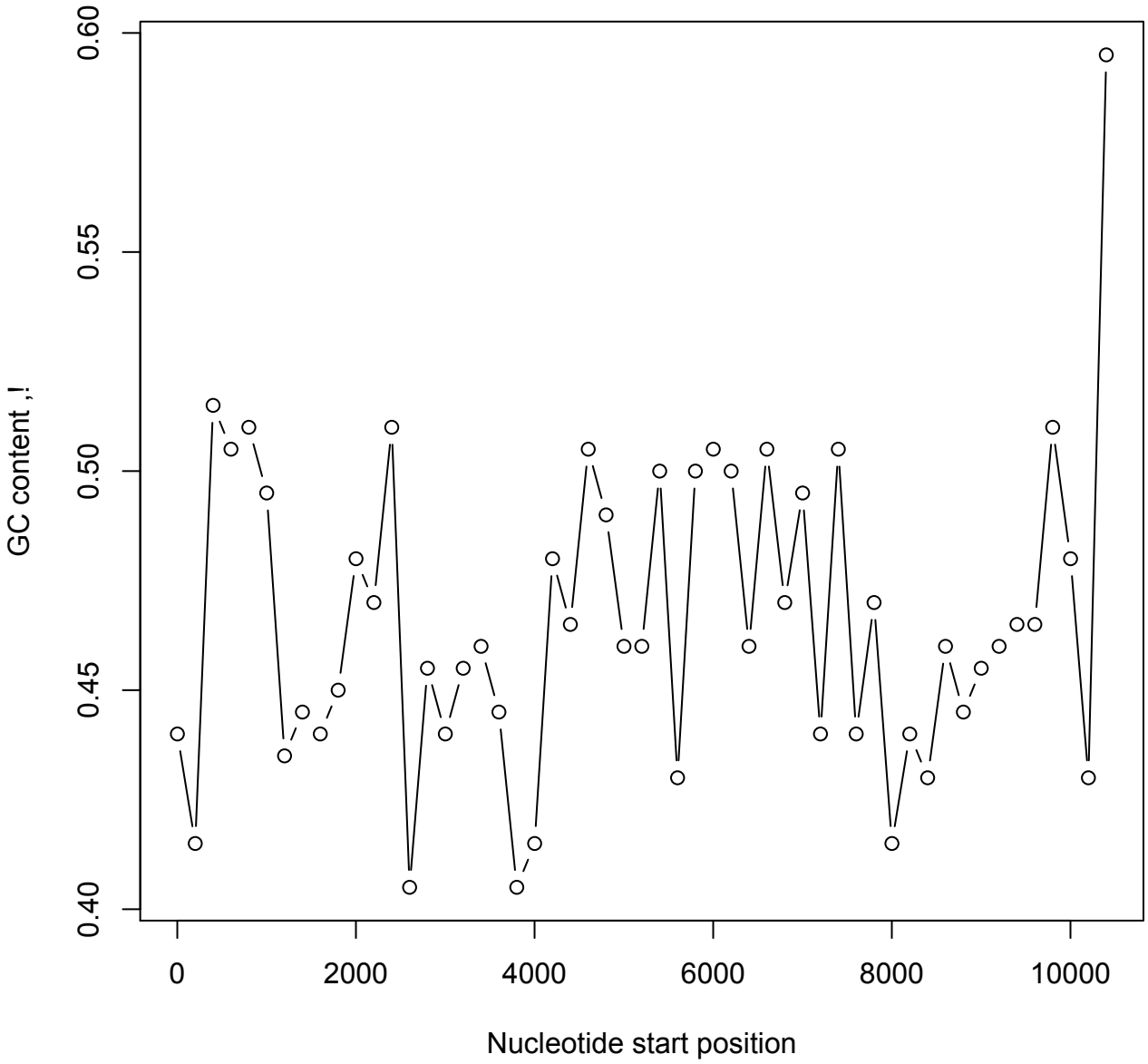
Inoltre, i valori Rho dei dinucleotidi GC+CG indicano un contenuto GC effettivo inferiore a quello atteso.

2. Analisi statistica delle sequenze biologiche — Parte II

Q2.1 ***Disegnare un grafico sliding window del contenuto di GC nel genoma del virus*** **Dengue DEN-1*****, utilizzando una finestra di 200 nucleotidi. Si vede qualche regione con un contenuto di DNA insolito nel genoma (ad es. un picco alto o uno troppo basso)?***

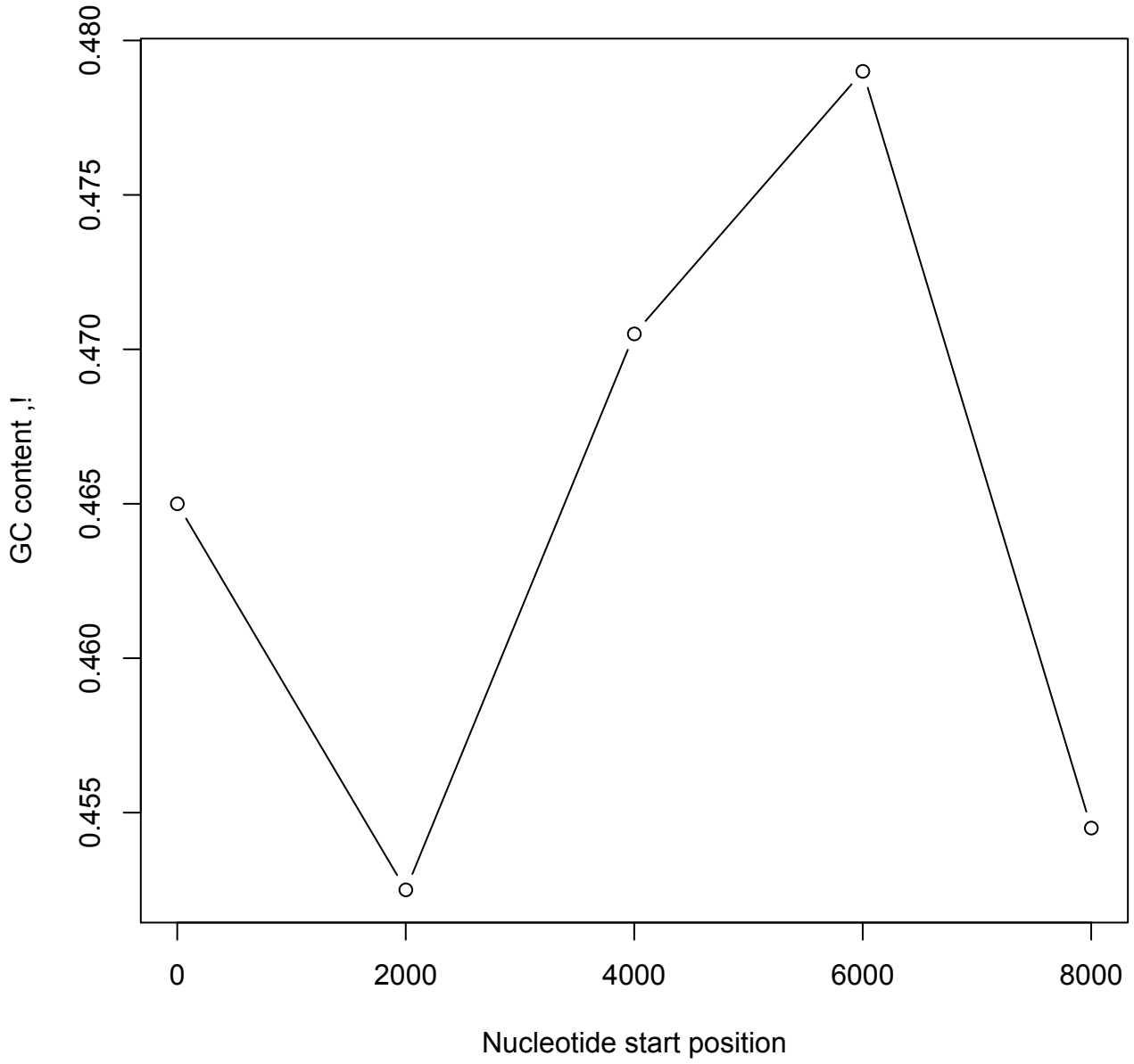
Suggerimento: Esaminare ogni grafico tracciato. In quale posizione (in coppie di basi) del genoma si trova (approssimativamente) il più grande cambiamento nel contenuto di GC locale? Confrontare i grafici *sliding window* del contenuto di GC creati usando dimensioni di finestra di 200 e 2.000 nucleotidi. In che modo le dimensioni della finestra influenzano la capacità di rilevare le differenze all'interno del genoma del virus Dengue?

Per fare questo, è necessario prima scaricare la sequenza del virus *Dengue DEN-1* dal database NCBI. Per fare ciò, seguire i passi del quesito 1.Q1 e memorizzare la sequenza nella variabile *dengueseq*; quindi fare un grafico *sliding window* con una dimensione della finestra di 200 nucleotidi utilizzando la funzione `slidingwindowplot(200, dengueseq)`:



Il contenuto di GC varia da circa il 45% a circa il 50% in tutto il genoma del virus *Dengue DEN-1*, con alcuni abbassamenti notevoli a circa 2.500 e 4.000 basi lungo la sequenza, dove il contenuto di GC scende a circa il 40%. Non c'è una forte differenza tra l'inizio e la fine del genoma, anche se da circa 4.000–7.000 basi il contenuto di GC è abbastanza alto (circa il 50%), e da circa 2.500–3.500 e 7.000–9.000 basi il contenuto di GC è relativamente basso (circa 43–47%).

Possiamo anche fare un grafico a finestra scorrevole del contenuto di GC usando una dimensione della finestra di 2000 nucleotidi mediante `slidingwindowplot(2000, dengueseq)`:



In questa immagine è molto più evidente che il contenuto di GC è relativamente alto da circa 4.000–7.000 basi, e più basso su entrambi i lati (da 2.500–3.500 e 7.000–9.000 basi).

Q2.2 ***Disegnare un grafico sliding window del contenuto di GC nella sequenza del genoma per il ceppo*** **TN** ***del batterio*** **Mycobacterium leprae** ***(accession*** **NC_002677***) utilizzando una dimensione della finestra***

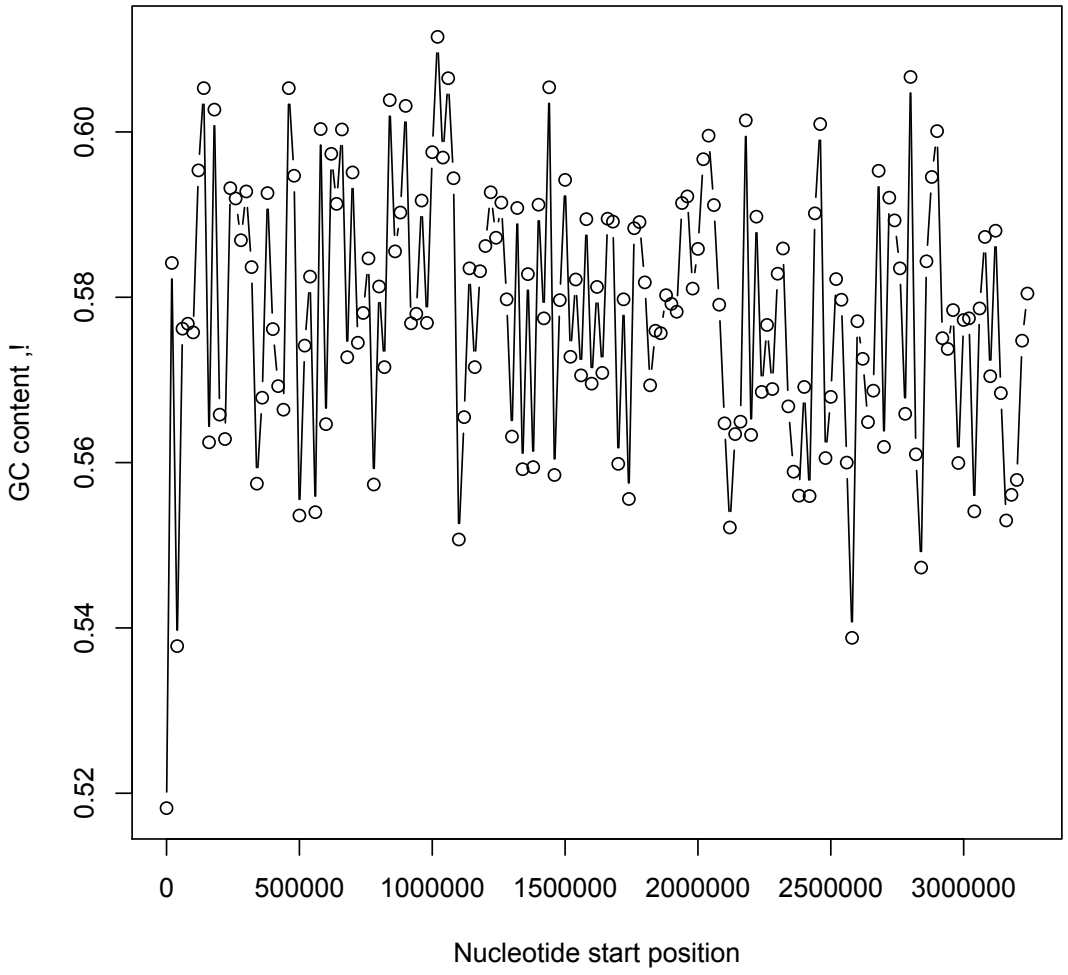
di 20.000 nucleotidi. Si vede qualche regione con un contenuto di DNA insolito nel genoma (ad es. un picco alto o uno troppo basso)?

Suggerimento: Esaminare ogni grafico tracciato. Annotare la posizione approssimativa corrispondente al punto più alto o più basso del grafico. Perché è stata scelta una finestra di 20.000 nucleotidi? Cosa si vede se si utilizza una dimensione della finestra molto più piccola (ad esempio 200 nucleotidi) o una dimensione della finestra molto più grande (ad es. 200.000 nucleotidi)?

Per fare questo, è necessario prima scaricare la sequenza del virus *Mycobacterium leprae* dal database NCBI, ad esempio con i comandi:

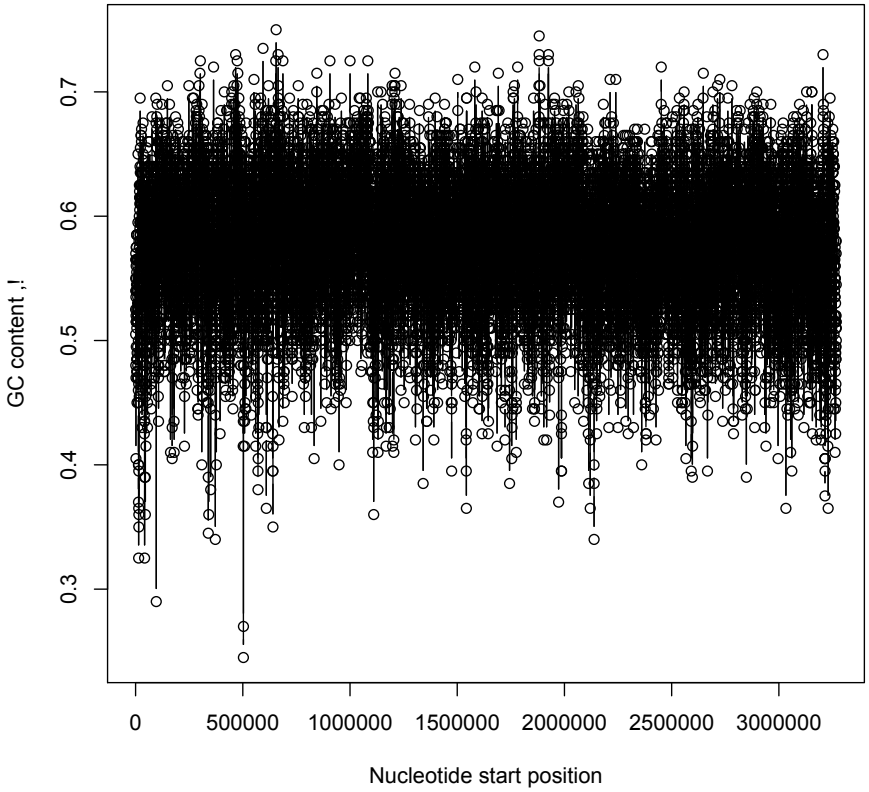
```
> leprae <- retrieventrezseqs("NC_002677", "nucleotide")
[1] "Retrieving sequence NC_002677 ..."
> lepraeseq <- leprae[[1]]
```

che memorizzano la sequenza nella variabile *lepraeseq*. Quindi tracciamo un grafico sliding window con una dimensione della finestra di 20.000 nucleotidi con il comando `slidingwindowplot(20000, lepraeseq)`:

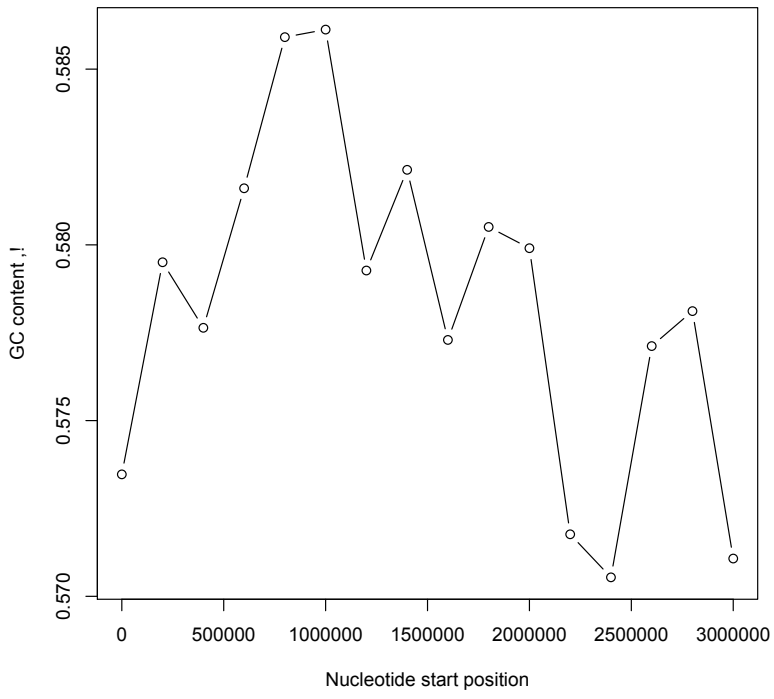


Vediamo il picco più alto del contenuto di GC a circa 1 Mb nel genoma *M. leprae*. Vediamo anche minimi nel contenuto di GC a circa 1,1 Mb e a circa 2,6 Mb.

Se invece utilizziamo una finestra di 200 nucleotidi con il comando `slidingwindowplot(200, lepraeseq)` il grafico diventa molto disordinato, e non possiamo facilmente distinguere i picchi e i minimi del contenuto di GC:



Se infine tracciamo un grafico con una finestra di 200.000 nucleotidi (mediante il comando `slidingwindowplot(200000, lepraeseq)`), il risultato è molto approssimato e non possiamo distinguere in dettaglio i picchi e i minimi nella distribuzione di GC:



Q2.3 ***Scrivere una funzione per calcolare il contenuto di AT di una sequenza di DNA (cioè la frazione dei nucleotidi della sequenza che sono A o T). Qual è il contenuto di AT del genoma del*** **Mycobacterium leprae TN*?***

Suggerimento: Utilizzare la funzione count() per creare una tabella contenente il numero di A, T, G e C nella sequenza. Si ricordi che la funzione count() produce un oggetto tabella, ai quali elementi si può accedere utilizzando le doppie parentesi quadre. Si nota una relazione tra il contenuto AT del genoma del *Mycobacterium leprae TN* e il suo contenuto GC?

Ecco una funzione per calcolare il contenuto di AT di una sequenza genomica:

```
> AT <- function(inputseq) {
+ inputseq <- tolower(inputseq)
+ mytable <- count(inputseq, 1) # make a table with the count of A, C, G, and T
+ mylength <- length(inputseq) # find the length of the whole sequence
+ myA <- mytable[1] # number of As in the sequence
+ myT <- mytable[4] # number of Ts in the sequence
+ myAT <- (myA + myT)/mylength
+ return(myAT) }
```

Possiamo quindi utilizzare la funzione per calcolare il contenuto di AT del genoma *M. leprae*:

```
> AT(lepraeseq)
[1] 0.4220325
```

Si osservi che il contenuto di AT è 1 — (contenuto di GC), cioè (contenuto di AT) + (contenuto di GC) = 1:

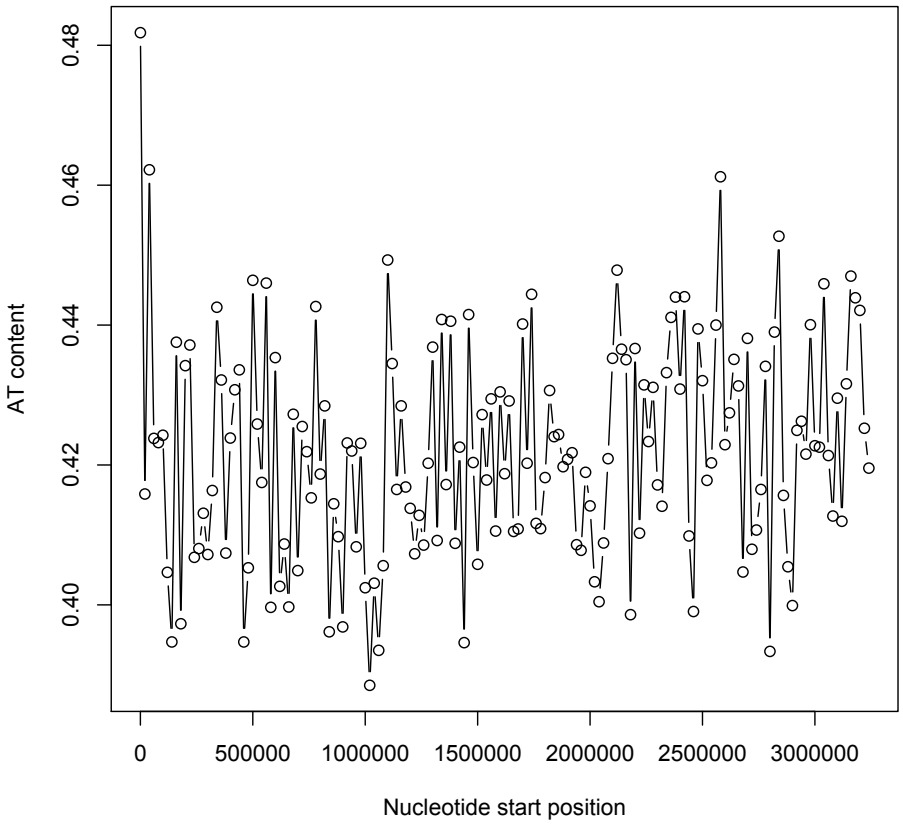
```
> GC(lepraeseq)
[1] 0.5779675
> 0.4220325 + 0.5779675
[1] 1
```

Q2.4 ***Scrivere una funzione per disegnare un grafico sliding window del contenuto di AT e usarla per fare un grafico sliding window del contenuto di AT lungo il genoma del*** **Mycobacterium leprae TN*, utilizzando una dimensione della finestra di 20.000 nucleotidi. Si nota una relazione tra il grafico del contenuto di GC e quello del contenuto di AT lungo il genoma del*** **Mycobacterium leprae*?***

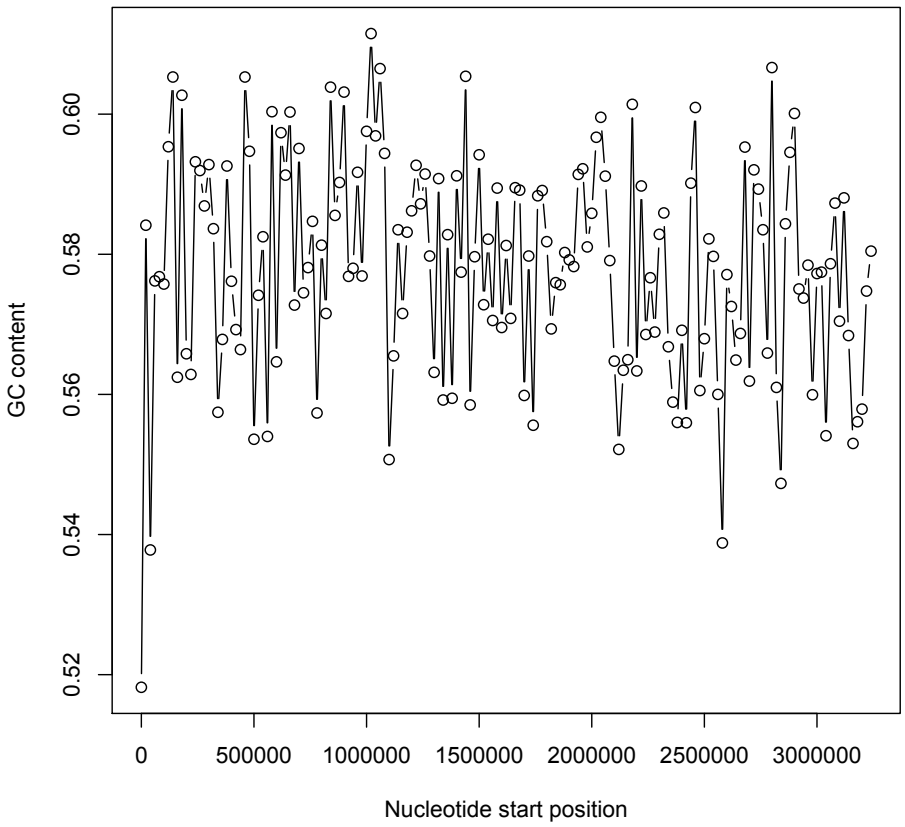
La funzione per realizzare un grafico *sliding window* di contenuti AT è la seguente:

```
> slidingwindowplotAT <- function(windowsize, inputseq) {
+ starts <- seq(1, length(inputseq)-windowsize, by = windowsize)
+ n <- length(starts)
+ chunkATs <- numeric(n)
+ for (i in 1:n) {
+     chunk <- inputseq[starts[i]:(starts[i]+windowsize-1)]
+     chunkAT <- AT(chunk)
+     chunkATs[i] <- chunkAT
+ }
+ plot(starts,chunkATs,type="b",xlab="Nucleotide start position",ylab="AT ,!content")
+ }
```

Possiamo quindi utilizzare questa funzione per realizzare un grafico *sliding window* con una dimensione della finestra di 20.000 nucleotidi digitando `slidingwindowplotAT(20000, lepraeseq)`:



Questa è l'immagine speculare del grafico del contenuto di GC (essendo il contenuto di AT in una sequenza uguale a 1 meno il suo contenuto di GC) ottenuto con la funzione `slidingwindowplot(20000, lepraeseq)`:



Q2.5 ***La parola di 3 nucleotidi*** **"GAC"** ***ha un contenuto di GC sovrarappresentato o sottorappresentato nella sequenza del genoma del*** **Mycobacterium leprae TN*?***

Nota: Qual è la frequenza di questa parola nella sequenza? Qual è la frequenza prevista di questa parola nella sequenza? Qual è il valore di ρ (Rho) per questa parola? Come si può capire se esiste già una funzione R per calcolare ρ?

Possiamo ottenere il conteggio di ciascuna delle parole di 3 nucleotidi nella sequenza *lepraeseq* digitando:

```
> count(lepraeseq, 3)

  aaa   aac   aag   aat   aca   acc   acg   act   aga   agc   agg   agt   ata   atc
32093 48714 36319 32592 44777 67449 57326 37409 31957 62473 38946 37470 25030 57245
  atg   att   caa   cac   cag   cat   cca   ccc   ccg   cct   cga   cgc   cgg   cgt
44268 32973 52381 64102 64345 43838 64869 46037 87560 38504 78120 82057 89358 57451
  cta   ctc   ctg   ctt   gaa   gac   gag   gat   gca   gcc   gcg   gct   gga   ggc
29004 39954 64730 36401 43486 61174 40728 58009 66775 80319 83415 62752 44002 81461
  ggg   ggt   gta   gtc   gtg   gtt   taa   tac   tag   tat   tca   tcc   tcg   tct
47651 69957 33139 60958 65955 50421 21758 32971 29454 25076 48245 43166 78685 31424
  tga   tgc   tgg   tgt   tta   ttc   ttg   ttt
49318 67270 67116 45595 22086 43363 54346 32374
```

Ci sono 61.174 GAC nella sequenza. Il numero totale di parole di 3 nucleotidi si può calcolare digitando:

```
> count(lepraeseq, 1)
    a      c      g      t
687041 938713 950202 692247
```

La lunghezza della sequenza è di 3.268.203 bp, per cui la frequenza osservata di GAC è 61.174/3.268.203 = 0,018718. La frequenza di G è 950.202/3.268.203 = 0,2907414; la frequenza di A è 687.041/3.268.203 = 0,2102198; la frequenza di C è 938.713/3.268.203 = 0,2872260. La frequenza prevista di GAC è quindi 0,2907414×0,2102198×0,2872260 = 0,01755514.

Il valore di *Rho* è quindi la frequenza osservata/frequenza attesa: ρ = 0,01871794 / 0,01755514 = 1,066237. Cioè, ci sono circa 1,1 volte il numero di GAC rispetto al valore atteso. Ciò significa che il GAC è leggermente sovrarappresentato in questa sequenza. La differenza da 1 è comunque così piccola che potrebbe non essere statisticamente significativa.

Possiamo cercare una funzione per calcolare il *Rho* digitando:

```
> help.search("rho")
```

ade4::neig	Neighbourhood Graphs
ade4::rhone	Physico-Chemistry Data
base::getHook	Functions to Get and Set Hooks for Load, Attach, Detach and Unload
igraph::ego_size	Neighborhood of graph vertices
prodlim::neighborhood	Nearest neighborhoods for kernel smoothing
prodlim::print.prodlim	Print objects in the prodlim library
R.utils::Non-documented objects	Non-documented objects
R.utils::GString$getBuiltinRhome	Gets the path where R is installed
seqinr::rho	Statistical over- and under- representation of dinucleotides in a sequence
stats::cor.test	Test for Association/Correlation Between Paired Samples

…

Nel pacchetto *SeqinR* è presente una funzione `rho()`. Ad esempio, possiamo usarla per calcolare il *Rho* per parole di lunghezza 3 nel genoma *M. leprae* digitando:

```
> rho(lepraeseq, wordsize=3)

      aaa       aac       aag       aat       aca       acc       acg       act
1.0570138 1.1742862 0.8649101 1.0653761 1.0793820 1.1899960 0.9991680 0.8949893
      aga       agc       agg       agt       ata       atc       atg       att
0.7610323 1.0888781 0.6706048 0.8856096 0.8181874 1.3695545 1.0462815 1.0697245
      caa       cac       cag       cat       cca       ccc       ccg       cct
1.2626819 1.1309452 1.1215062 1.0487995 1.1444773 0.5944657 1.1169725 0.6742135
      cga       cgc       cgg       cgt       cta       ctc       ctg       ctt
1.3615987 1.0467726 1.1261261 0.9938162 0.6939044 0.6996033 1.1197319 0.8643241
      gaa       gac       gag       gat       gca       gcc       gcg       gct
1.0355868 1.0662370 0.7012887 1.3710523 1.1638601 1.0246015 1.0512300 1.0855155
      gga       ggc       ggg       ggt       gta       gtc       gtg       gtt
0.7576632 1.0266049 0.5932565 1.1955191 0.7832457 1.0544820 1.1271276 1.1827465
      taa       tac       tag       tat       tca       tcc       tcg       tct
0.7112314 0.7888126 0.6961501 0.8135266 1.1542345 0.7558461 1.3611325 0.7461477
      tga       tgc       tgg       tgt       tta       ttc       ttg       ttt
1.1656391 1.1636701 1.1469683 1.0695410 0.7165237 1.0296334 1.2748168 1.0423929
```

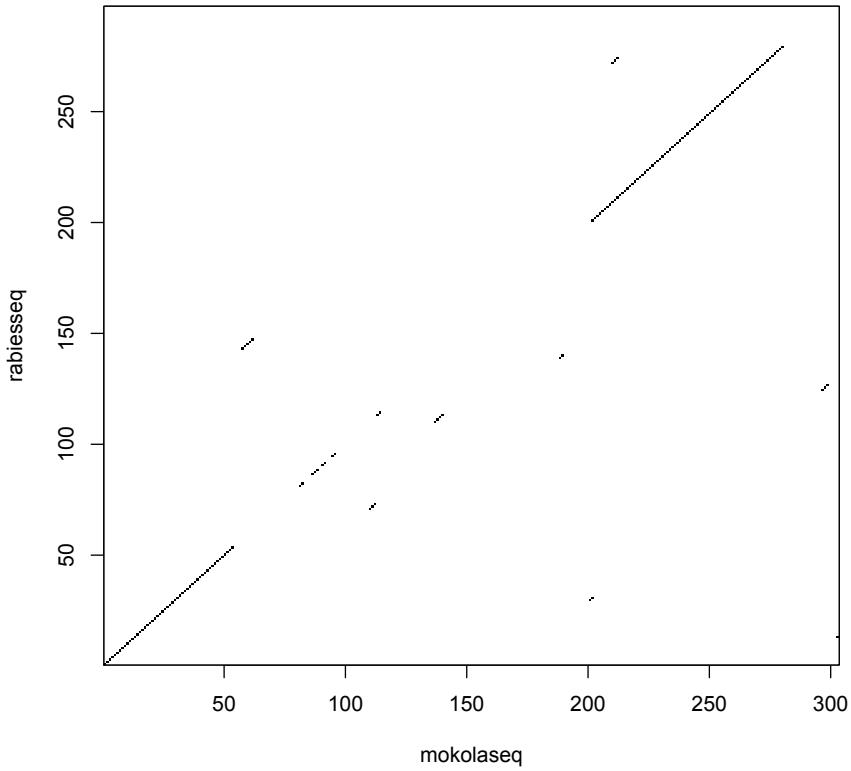
Il valore di *Rho* per la parola "GAC" è pari a 1,0662370, in accordo con il nostro calcolo precedente.

Q2.6 ***Utilizzare la funzione*** **dotPlot()** ***del pacchetto SeqinR per tracciare un dotplot della fosfoproteina del virus della rabbia e della fosfoproteina del virus*** **Mokola*****, utilizzando una dimensione della finestra di 10 e una soglia di 5.***

Per prima cosa dobbiamo recuperare le sequenze della fosfoproteina del virus della rabbia (*accession UniProt* P06747) e della fosfoproteina del virus *Mokola* (*accession UniProt* P0C569), cosa che possiamo fare usando *rentrez*; quindi tracciamo il grafico richiesto:

```
> source("retrieventrezseqs.R")
> rabies <- retrieventrezseqs("P06747","protein")
[1] "Retrieving sequence P06747 ..."
[1] "   ... found sequence for P06747 with ID 133685"
> rabiesseqseq <- rabies[[1]]
> mokola <- retrieventrezseqs("P0C569","protein")
[1] "Retrieving sequence P0C569 ..."
[1] "   ... found sequence for P0C569 with ID 158517729"
> mokolaqseq <- mokola[[1]]
> seqinr::dotPlot(mokolaseq, rabiesseq, wsize=10, nmatch=5)
```

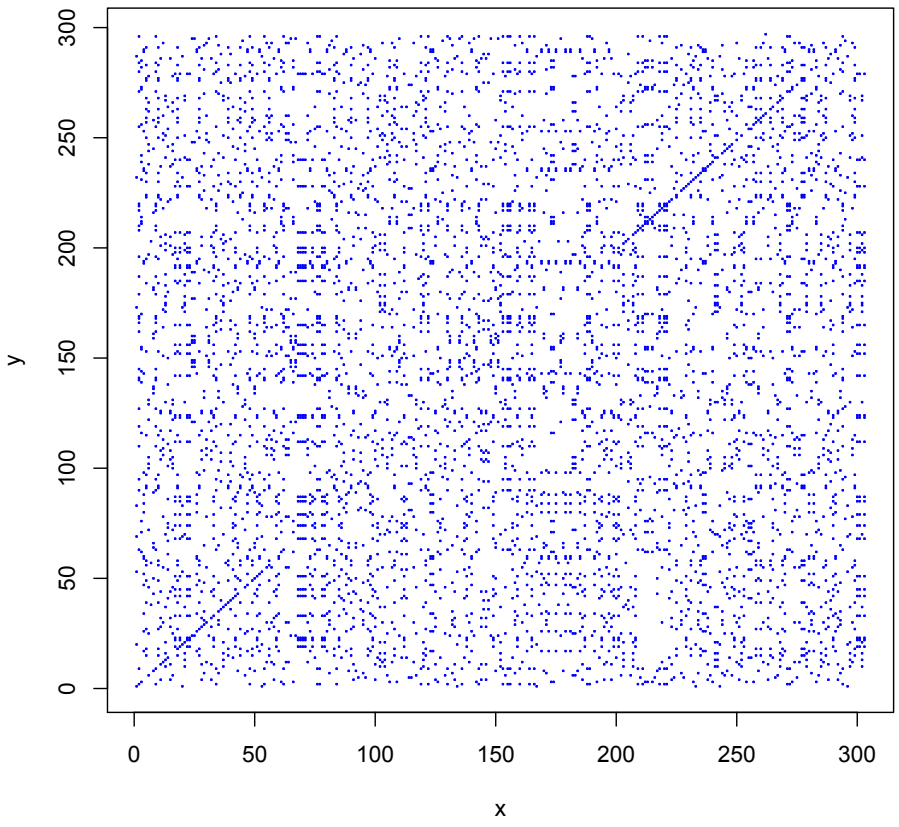
Si può vedere che c'è una regione di similarità che copre circa 60–70 aminoacidi all'inizio delle due proteine, poi c'è una regione di similarità a circa 210–280 in ciascuna delle due proteine. C'è anche una debole quantità di somiglianza in una regione a circa 85–100.



Q2.7 ***Utilizzare la funzione*** **makeDotPlot1()** ***per fare un dotplot della fosfoproteina del virus della rabbia e della fosfoproteina del virus*** **Mokola**, ***impostando l'argomento "dotsize" a 0,1.***

Usiamo la funzione `makeDotPlot1()` in Appendice per fare il dotplot richiesto digitando:

```
> makeDotPlot1(mokolaseq, rabiesseq, dotsize=0.1)
```



Come nel quesito Q2.6 si può vedere che c'è una regione di similarità che copre circa 60–70 amminoacidi all'inizio delle due proteine, poi c'è una regione di similarità a circa 210–280 in ciascuna delle due proteine.

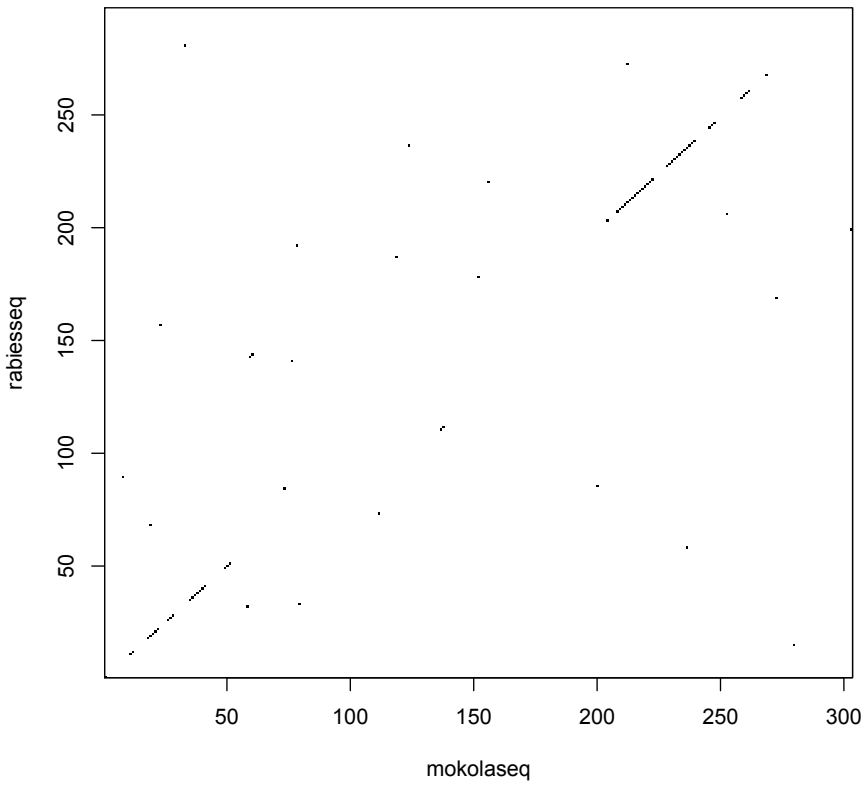
Ci sono molti punti fuori scala nell'immagine perché un punto viene tracciato in ogni posizione in cui le due sequenze sono identiche in una lettera (mentre nel Q2.6 abbiamo tracciato solo un punto all'inizio di una finestra di 10 lettere, e solo se 5 o più posizioni su 10 nella finestra erano identiche).

Il fatto che ci siano così tanti punti nell'immagine rende difficile vedere la debole regione di somiglianza vista nel Q2.6, alle posizioni circa 85–100 nelle due proteine.

Q2.8 ***Utilizzare la funzione* dotPlot() *del pacchetto SeqinR per creare un dotplot della fosfoproteina del virus della rabbia e della fosfoproteina del virus* Mokola*, utilizzando una dimensione della finestra di 3 e una soglia di 3. Utilizzare la funzione* makeDotPlot3() *in Appendice per creare lo stesso dotplot, con una dimensione della finestra (x) di 3 e una soglia (y) di 3. I due grafici sono simili o diversi, e si può spiegare perché?***

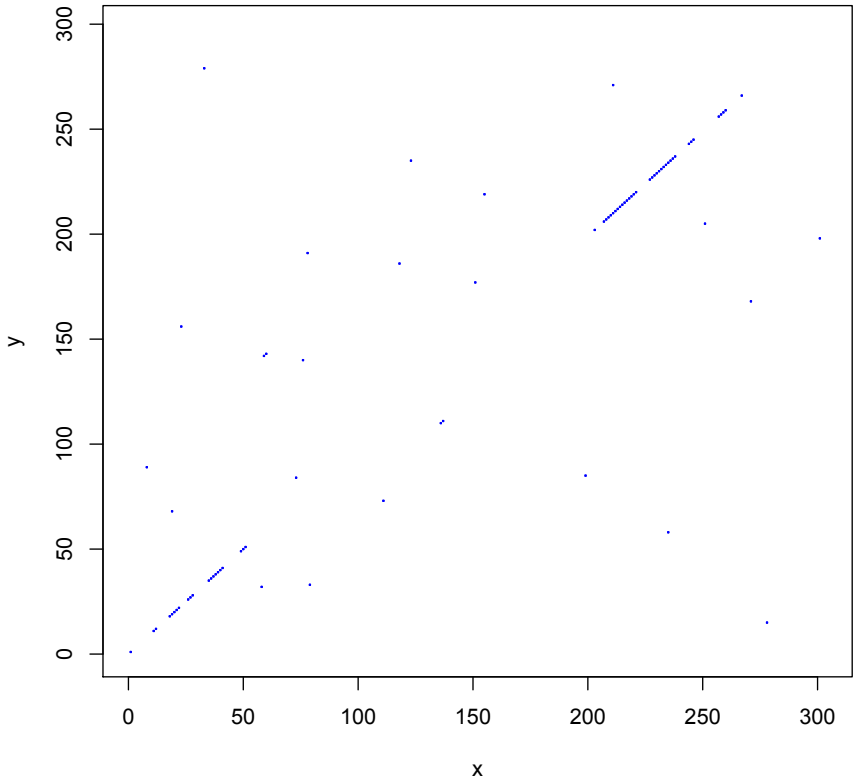
Possiamo usare la funzione `dotPlot()` di *SeqinR* per fare un dotplot delle fosfoproteine del virus della rabbia e del virus *Mokola*, usando una dimensione della finestra di 3 e una soglia di 3, digitando:

```
> seqinr::dotPlot(mokolaseq, rabiesseq, wsize=3, nmatch=3)
```



Utilizziamo ora la funzione `makeDotPlot3()` per fare un dotplot delle proteine del virus della rabbia e del virus *Mokola*, utilizzando una dimensione della finestra di 3 e una soglia di 3:

```
> makeDotPlot3(mokolaseq, rabiesseq, windowsize=3, threshold=3, dotsize=0.1)
```



Le due immagini sono le stesse, come dev'essere, in quanto entrambe tracciano un punto nella prima posizione di una finestra di 3 lettere se tutte e 3 le lettere di quella finestra sono identiche nelle due sequenze.

3. Database di sequenze biologiche

Q3.1 ***Quali informazioni sulla sequenza del virus della rabbia (accession NCBI* NC_001542*) si possono ottenere dalle sue annotazioni nella banca dati delle sequenze NCBI?***

Nota: Cosa riporta nei campi DEFINITION e ORGANISM il suo record NCBI?

Per fare questo, è necessario andare sul sito web NCBI e digitare l'*accession* della sequenza del genoma del virus della rabbia (NC_001542) nella casella di ricerca e premere <Search>.

Nella pagina dei risultati della ricerca, si vede un '1' accanto alla parola 'Nucleotide', che significa che c'è un riscontro per un record di sequenza nel database NCBI 'Nucleotide', che contiene sequenze di DNA e RNA. Se si clicca sulla parola 'Nucleotide', si arriva al record di sequenza per il genoma del virus della rabbia:

Rabies virus, complete genome

NCBI Reference Sequence: NC_001542.1

FASTA Graphics

Go to:

```
LOCUS       NC_001542               11932 bp ss-RNA     linear   VRL 08-DEC-2008
DEFINITION  Rabies virus, complete genome.
ACCESSION   NC_001542
VERSION     NC_001542.1  GI:9627197
DBLINK      Project: 15144
KEYWORDS    .
SOURCE      Rabies virus
  ORGANISM  Rabies virus
            Viruses; ssRNA negative-strand viruses; Mononegavirales;
            Rhabdoviridae; Lyssavirus.
REFERENCE   1  (bases 5388 to 11932)
  AUTHORS   Tordo,N., Poch,O., Ermine,A., Keith,G. and Rougeon,F.
  TITLE     Completion of the rabies virus genome sequence determination:
            highly conserved domains among the L (polymerase) proteins of
            unsegmented negative-strand RNA viruses
  JOURNAL   Virology 165 (2), 565-576 (1988)
   PUBMED   3407152
REFERENCE   2  (bases 1 to 5500)
  AUTHORS   Tordo,N., Poch,O., Ermine,A., Keith,G. and Rougeon,F.
  TITLE     Walking along the rabies genome: is the large G-L intergenic region
            a remnant gene?
  JOURNAL   Proc. Natl. Acad. Sci. U.S.A. 83 (11), 3914-3918 (1986)
   PUBMED   3459163
```

Nella pagina web sopra si possono vedere i campi DEFINITION, ORGANISM e REFERENCE del record NCBI:

- DEFINITION: virus della rabbia, genoma completo.
- ORGANISM: Virus della rabbia.
- REFERENCE: Ci sono diversi documenti; il primo è descritto di seguito.

- AUTHORS: Tordo, N., Poch, O., Ermine, A., Keith, G. e Rougeon, F.
- TITLE: Completion of the rabies virus genome sequence determination: highly conserved domains among the L (polymerase) proteins of unsegmented negative-strand RNA viruses (Completamento della determinazione della sequenza del genoma del virus della rabbia: domini altamente conservati tra le proteine L (polimerasi) dei virus RNA con *strand* negativo non segmentato).
- JOURNAL: Virology 165 (2), 565–576 (1988)

Ci sono anche alcuni altri riferimenti, per gli articoli pubblicati sulla sequenza del genoma del virus della rabbia.

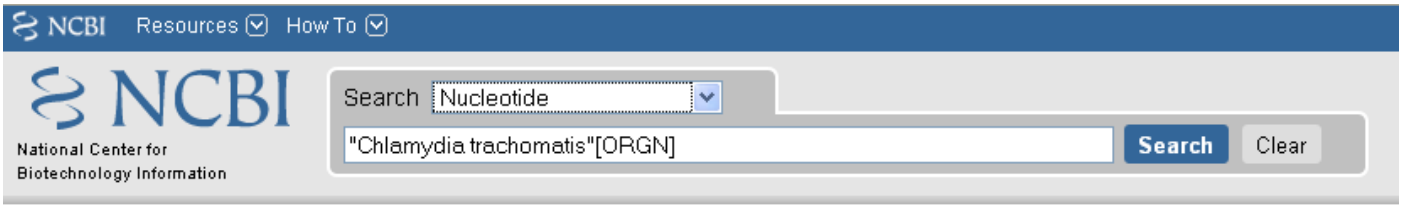
Un modo alternativo per recuperare le annotazioni per la sequenza del virus della rabbia è quello di utilizzare il pacchetto R *SeqinR*. Poiché il virus della rabbia è un virus, la sequenza del suo genoma dovrebbe trovarsi nel sub-database ACNUC "refseqViruses". Pertanto, possiamo eseguire la seguente query per recuperare le annotazioni per la sequenza del genoma del virus della rabbia (*accession* NC_001542):

```
> library(seqinr) # load the SeqinR R package
> choosebank("refseqViruses") # select the ACNUC sub-database to be searched
> rabies <- query("rabies", "AC=NC_001542") # specify the query
> annots <- getAnnot(rabies$req[[1]]) # retrieve the annotations
> annots[1:20]
 [1] "LOCUS       NC_001542              11932 bp ss-RNA     linear   VRL 08-DEC-2008"
 [2] "DEFINITION  Rabies virus, complete genome."
 [3] "ACCESSION   NC_001542"
 [4] "VERSION     NC_001542.1  GI:9627197"
 [5] "DBLINK      Project: 15144"
 [6] "KEYWORDS    ."
 [7] "SOURCE      Rabies virus"
 [8] "  ORGANISM  Rabies virus"
 [9] "            Viruses; ssRNA negative-strand viruses; Mononegavirales;"
[10] "            Rhabdoviridae; Lyssavirus."
[11] "REFERENCE   1  (bases 5388 to 11932)"
[12] "  AUTHORS   Tordo,N., Poch,O., Ermine,A., Keith,G. and Rougeon,F."
[13] "  TITLE     Completion of the rabies virus genome sequence determination:"
[14] "            highly conserved domains among the L (polymerase) proteins of"
[15] "            unsegmented negative-strand RNA viruses"
[16] "  JOURNAL   Virology 165 (2), 565-576 (1988)"
[17] "   PUBMED   3407152"
[18] "REFERENCE   2  (bases 1 to 5500)"
[19] "  AUTHORS   Tordo,N., Poch,O., Ermine,A., Keith,G. and Rougeon,F."
[20] "  TITLE     Walking along the rabies genome: is the large G-L intergenic region"
> closebank()
```

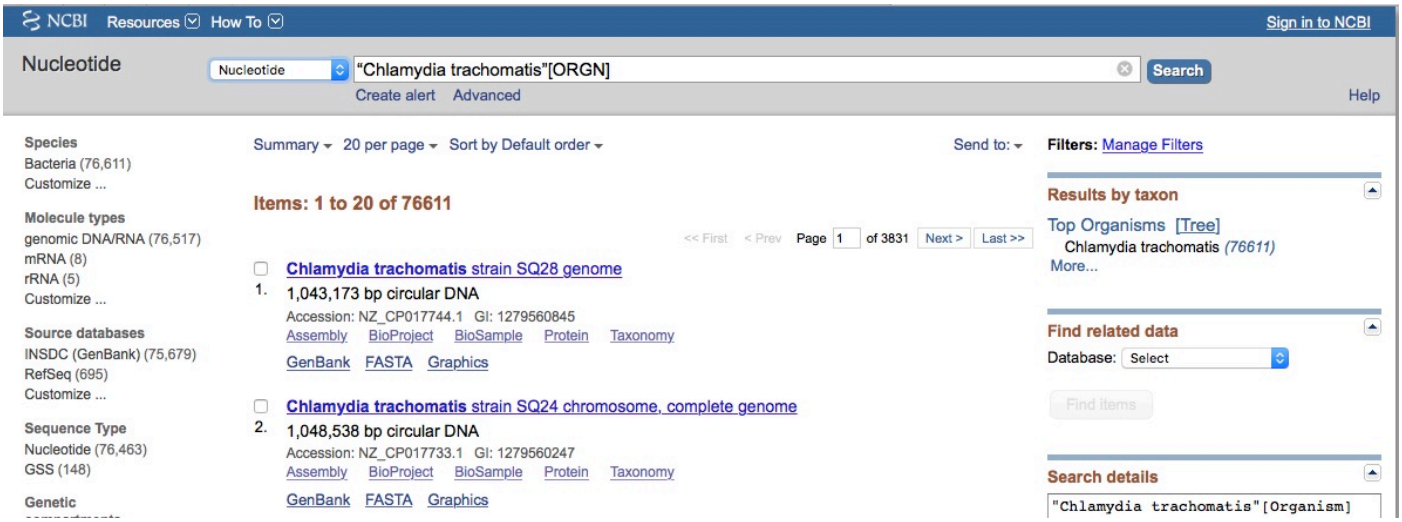
Q3.2 ***Quante sequenze nucleotidiche delle sequenze del batterio*** **Chlamydia trachomatis** ***sono presenti nella banca dati NCBI?***

Nota: Il batterio *Chlamydia trachomatis* è responsabile del tracoma, classificato dall'OMS come malattia tropicale trascurata.

Per rispondere a questa domanda, possiamo andare sul sito web NCBI e selezionare "Nucleotide" dall'elenco a discesa in alto nella pagina web, poiché si desidera cercare le sequenze di nucleotidi (DNA o RNA). Quindi nella casella di ricerca, digitare "Chlamydia trachomatis"[ORGN] e premere <Search>:



Qui [ORGN] specifica che interessa l'organismo, cioè il nome della specie in latino. La pagina dei risultati fornisce un elenco dei risultati dei record di sequenza nel database dei nucleotidi NCBI:



In alto a sinistra vediamo che sono state trovate 76.611 sequenze in totale, di cui (vedi la sezione "Sequence Type" più in basso) 76.463 sono sequenze di DNA o RNA, e 148 sono sequenze di DNA provenienti da Genome Sequence Surveys (GSS), cioè da progetti di sequenziamento del genoma (alla data di verifica). Si noti che ci sono nuove sequenze che vengono aggiunte al database continuamente, quindi se si effettua nuovamente il controllo si troverà molto probabilmente un numero maggiore di sequenze.

Se si cerca "Chlamydia trachomatis"[ORGN] scegliendo "All Databases" dall'elenco a discesa si ottengono 76.611 hit nel database "Nucleotide" e 177 nel database "BioProject".

Si noti anche che se si cerca "Chlamydia trachomatis", senza usare [ORGN] per specificare l'organismo, si ottengono 102.805 risultati nel database dei nucleotidi e 216 in "BioProject", ma alcuni di questi potrebbero non essere sequenze di *Chlamydia trachomatis*: alcuni potrebbero essere sequenze di altre specie per le quali il record di sequenza NCBI contiene da qualche parte la frase "Chlamydia trachomatis".

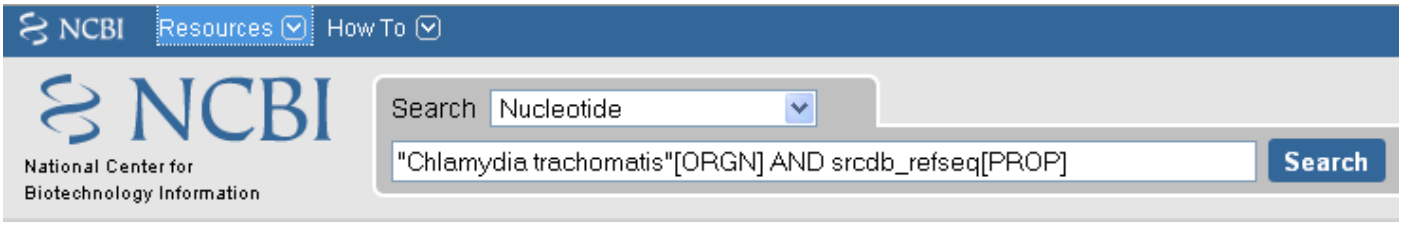
Un modo alternativo per cercare sequenze nucleotidiche del batterio *Chlamydia trachomatis* è quello di usare il pacchetto *rentrez*. Volendo trovare le sequenze nucleotidiche, il database corretto da cercare è "nucleotide". In questo modo, possiamo effettuare la nostra ricerca digitando:

```
> r_search <- entrez_search(db='nucleotide', term='"Chlamydia trachomatis"[ORGN]')
> r_search
Entrez search result with 76611 hits (object contains 20 IDs and no web_history object)
 Search term (as translated):  "Chlamydia trachomatis"[Organism]
```

Abbiamo trovato 76.611 sequenze nucleotidiche di *Chlamydia trachomatis*, ottenendo (non necessariamente) lo stesso numero di sequenze del sito web NCBI.

Q3.3 ***Quante sequenze nucleotidiche sono presenti dal batterio*** **Chlamydia trachomatis** ***nel database di sequenze NCBI RefSeq?***

Per rispondere a questa domanda, è necessario andare sul sito web NCBI e selezionare "Nucleotide" dall'elenco a discesa in alto nella pagina web, poiché si desidera cercare sequenze nucleotidiche. Poi, nella casella di ricerca digitare "'Chlamydia trachomatis'[ORGN] AND srcdb_refseq[PROP]" e premere <Search>:



Qui [ORGN] specifica l'organismo, e [PROP] specifica una proprietà delle sequenze (in questo caso che appartengono alla sottosezione *RefSeq* della banca dati NCBI).

All'inizio alla pagina dei risultati c'è scritto "Items: 1 to 20 of 695", quindi ci sono 695 sequenze corrispondenti [al 4 Gennaio 2020], che sarà probabilmente più alto in futuro, a causa delle sequenze continuamente aggiunte al database.

Si noti che le sequenze cecate nel quesito Q3.2 sono tutte sequenze DNA e RNA di *Chlamydia trachomatis* nel database NCBI. Le ricerca nel quesito attuale è relativa alle sequenze DNA e RNA di *Chlamydia trachomatis* nella parte *RefSeq* del database NCBI, che è una sottosezione del database per dati di alta qualità curati manualmente.

Il numero di sequenze in *RefSeq* è molto inferiore al numero totale di sequenze di *C. trachomatis*, in parte perché le sequenze di bassa qualità non vengono mai aggiunte a *RefSeq*, ma anche perché i curatori di *RefSeq* probabilmente non hanno avuto il tempo di aggiungere tutte le sequenze di alta qualità (si tratta di un processo che richiede tempo, poiché i curatori aggiungono informazioni aggiuntive ai record di sequenze NCBI in *RefSeq*, come riferimenti a documenti che discutono una particolare sequenza).

Un modo alternativo per cercare sequenze nucleotidiche in *RefSeq* utilizza il pacchetto R *rentrez*. Vogliamo trovare sequenze nucleotidiche del batterio *Chlamydia trachomatis* in *RefSeq*, quindi il database NCBI corretto da cercare è "nucleotide", l'organismo è "Chlamydia trachomatis" e il sotto-database "srcdb_refseq". In questo modo, possiamo effettuare la nostra ricerca digitando:

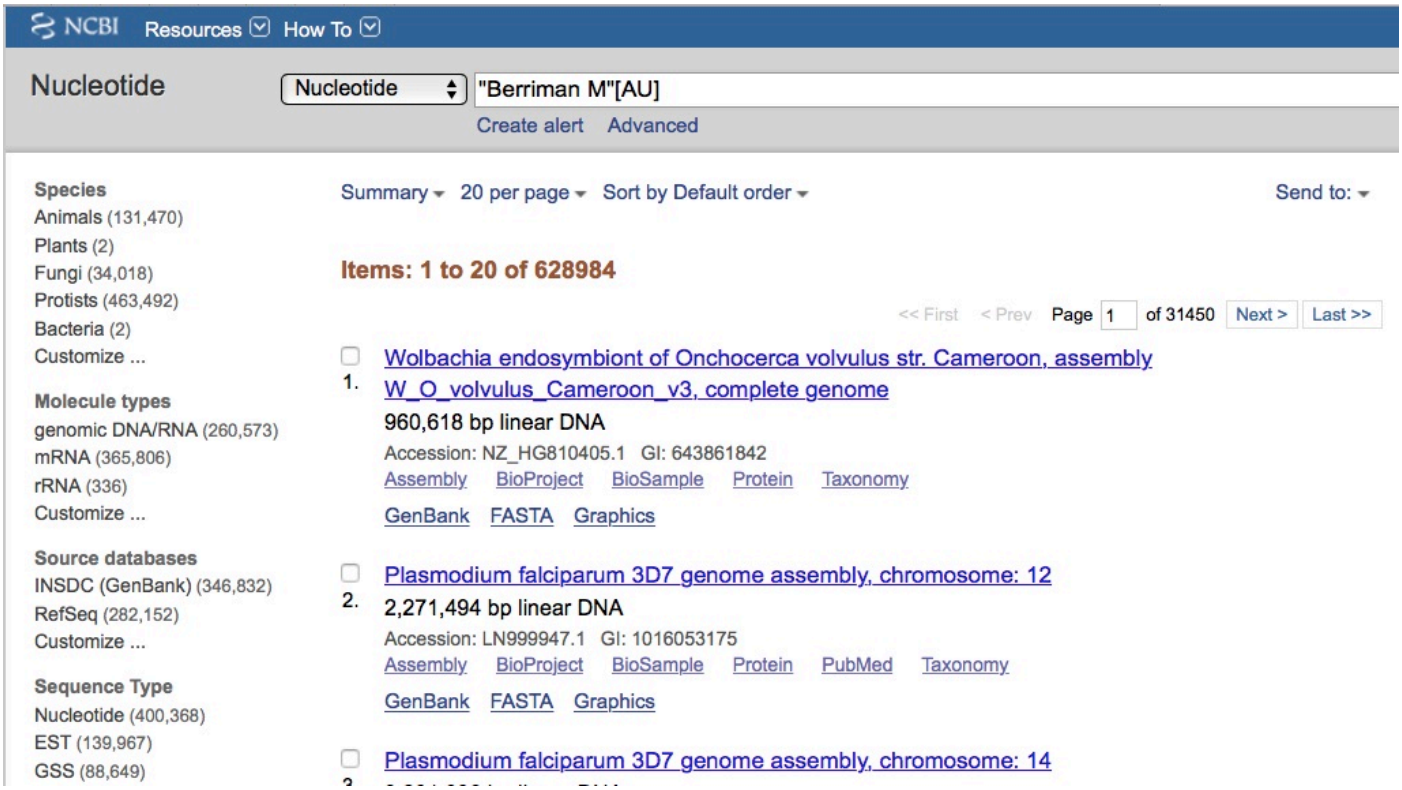
```
> r_search <- entrez_search(db='nucleotide', term='"Chlamydia trachomatis"[ORGN] AND
srcdb_refseq[PROP]')
> r_search
Entrez search result with 695 hits (object contains 20 IDs and no web_history object)
 Search term (as translated):  "Chlamydia trachomatis"[Organism] AND srcdb_refseq ...
```

Se si desidera recuperare la sequenza completa del primo risultato, occorre prima leggere il suo ID (che corrisponde alla proprietà *id* della variabile *r_search*), quindi recuperare la sequenza in formato FASTA in base all'ID e infine scriverla in un file di testo (ad es. "chlamydia_t.fasta"):

```
r_search$ids[1]
[1] “1279560845"
> f <- entrez_fetch(db="nucleotide", id=1279560845, rettype="fasta")
> write(f, file="chlamydia_t.fasta")
```

Q3.4 ***Quante sequenze nucleotidiche sono state presentate all’NCBI da Matthew Berriman?***

Per rispondere a questa domanda, andare sul sito web NCBI e selezionare in alto il database "Nucleotide", trattandosi di una ricerca di sequenze nucleotidiche. Quindi, nella casella di ricerca digitare "Berriman M"[AU] e premere <Search> ([AU] specifica il nome dell’autore, cioè della persona che ha inviato la sequenza alla banca dati NCBI, oppure ha scritto un documento che descrive la sequenza):



In alto nella pagina dei risultati, c’è scritto [al 4 Gennaio 2020]: "Items: 1 to 20 of 628984". Ciò significa che 628.984 sequenze nucleotidiche sono state presentate al database della NCBI da qualcuno chiamato "Berriman M.", o sono state descritte in un documento da qualcuno con tale nome. Di queste, 260.573 sono sequenze di DNA/RNA, 365.806 sono mRNA e 336 sono rRNA.

Nota: *Purtroppo il sito web NCBI non permette di cercare "Berriman Matthew"[AU], quindi non possiamo essere sicuri che tutte queste sequenze siano state presentate da quel particolare autore*.

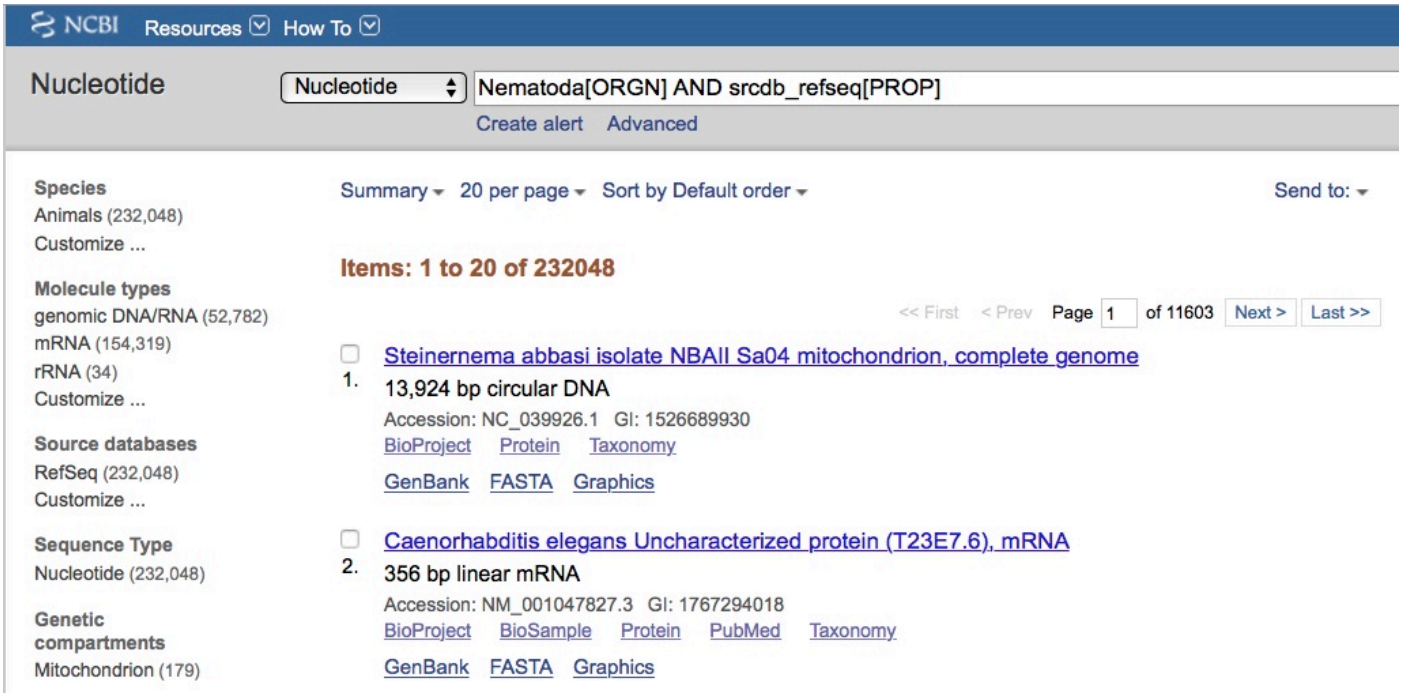
Un modo alternativo per cercare le sequenze nucleotidiche presentate da M. Berriman è quello di utilizzare il pacchetto *rentrez*. Vogliamo trovare le sequenze nucleotidiche, quindi il database NCBI appropriato per la ricerca è "nucleotide". Pertanto, digitiamo:

```
> r_search <- entrez_search(db = 'nucleotide', term = '"Berriman M."[AU]')
> r_search
Entrez search result with 628984 hits (object contains 20 IDs and no web_history
object)
 Search term (as translated):  "Berriman M."[AU]
```

Q3.5 ***Quante sequenze nucleotidiche dei vermi nematodi sono presenti nel database RefSeq?***

Si noti che diversi vermi parassiti nematodi causano malattie tropicali trascurate, tra cui *Brugia malayi* e *Wucheria bancrofti*, che causano la filariosi linfatica; *Loa loa*, che causa la filariosi sottocutanea; *Onchocerca volvulus*, che causa l'oncocercosi; e *Necator americanus*, che causa l'elmintiasi trasmessa dal suolo.

Per rispondere a questa domanda, andare sul sito web NCBI e selezionare il database "Nucleotide". Quindi, nella casella di ricerca, digitare "Nematoda[ORGN] AND srcdb_refseq[PROP]" e premere <Search>:



Qui [ORGN] specifica il gruppo di specie di cui si desidera cercare le sequenze. Nel quesito Q3.3 [ORGN] è stato usato per specificare il nome di un organismo (*Chlamydia trachomatis*). Tuttavia, si può anche usare [ORGN] per specificare il nome di un gruppo di organismi: ad esempio, "Fungi[ORGN]" cercherebbe sequenze di funghi, mentre "Mammalia[ORGN]" cercherebbe sequenze di mammiferi. Il nome del gruppo di specie che si vuole cercare deve essere dato in latino, quindi per cercare sequenze di vermi nematodi si usa il nome latino *Nematoda*.

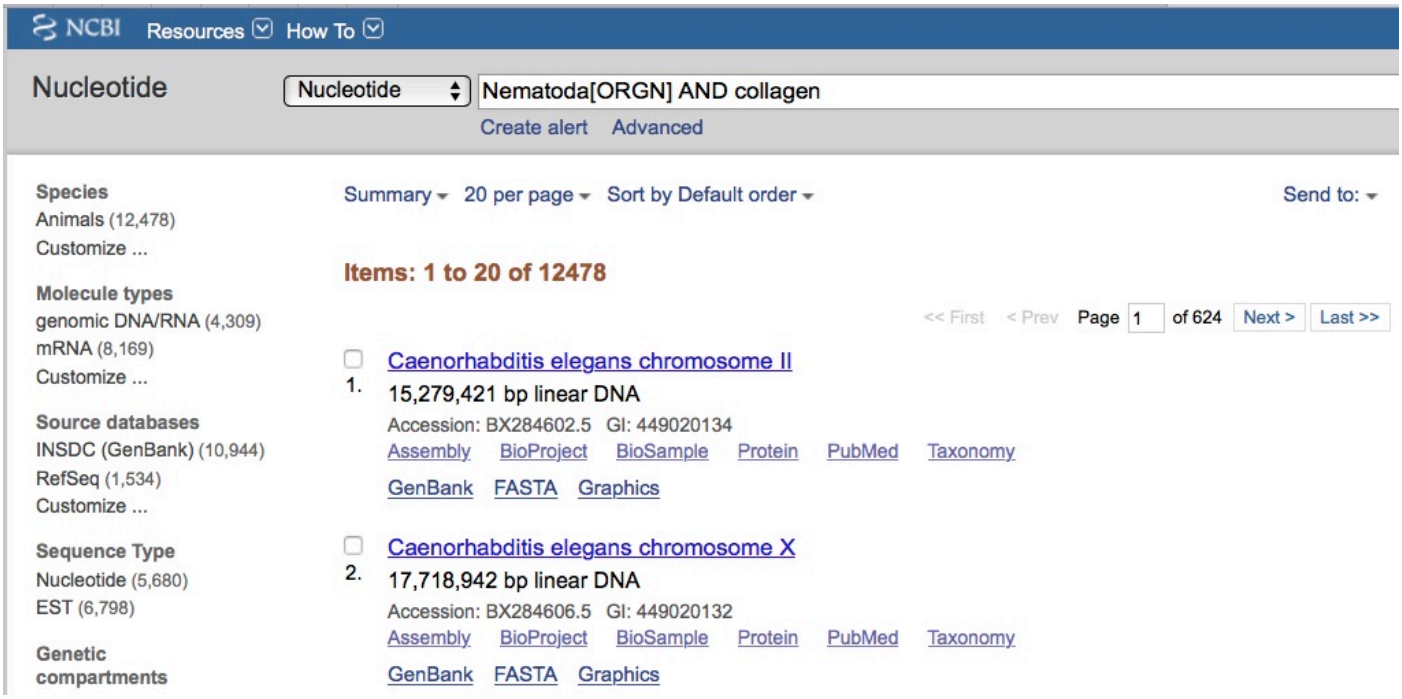
Nella pagina di ricerca c'è scritto in alto "Item: 1 to 20 di 232048" [al 4 Gennaio 2020]. Ciò significa che sono state trovate 232.048 sequenze di DNA o RNA di specie di vermi nematodi nel database *RefSeq*. Queste sequenze provengono probabilmente da una vasta gamma di specie di vermi nematodi, incluso il verme nematode modello *Caenorhabditis elegans*, così come da specie di nematodi parassiti.

In alternativa, per cercare le sequenze nucleotidiche *RefSeq* dai vermi nematodi possiamo utilizzare il pacchetto *rentrez*. Vogliamo trovare le sequenze nucleotidiche che sono in *RefSeq*, quindi il database NCBI appropriato per la ricerca è "nucleotide". Pertanto, digitiamo:

```
> r_search <- entrez_search(db = 'nucleotide', term = 'Nematoda[ORGN] AND
srcdb_refseq[PROP]')
> r_search
Entrez search result with 232048 hits (object contains 20 IDs and no web_history
object)
 Search term (as translated):  "Nematoda"[Organism] AND srcdb_refseq[PROP]
```

Q3.6 ***Quante sequenze nucleotidiche per i geni del collagene dei vermi nematodi sono presenti nella banca dati NCBI?***

Per rispondere a questa domanda, andiamo sil sito web NCBI (https://www.ncbi.nlm.nih.gov) e selezioniamo il database "Nucleotide" dall'elenco a discesa in alto. Quindi, nella casella di ricerca digitiamo "Nematoda[ORGN] AND collagen" e premiamo <Search>:



Qui [ORGN] specifica che vogliamo sequenze di vermi nematodi. La frase "AND collagen" significa che la parola "collagen" deve apparire da qualche parte nelle voci NCBI per quelle sequenze; ad esempio, nel nome della sequenza, o in una descrizione della sequenza, o nel titolo di un articolo che descrive la sequenza, ecc.

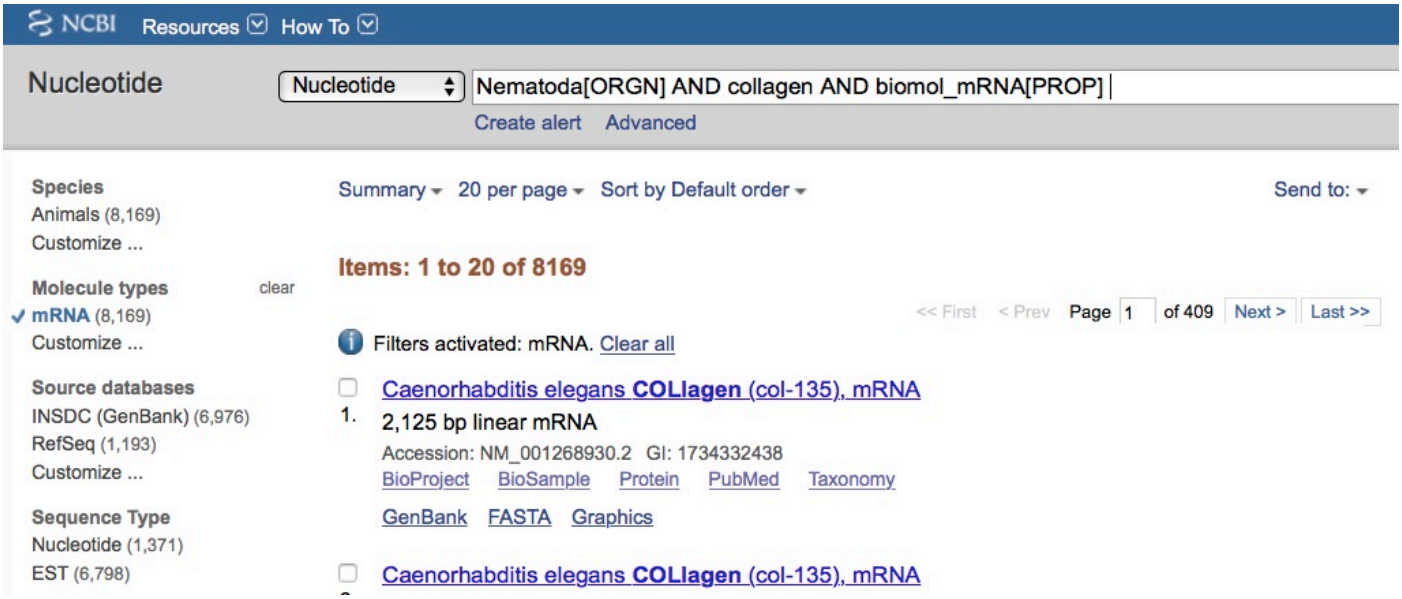
Nella pagina dei risultati, vediamo in alto: "Item: 1 to 20 of 12478", il che significa che sono state trovate 12.478 sequenze nucleotidiche per i geni del collagene dei vermi nematodi, di cui 4.309 sono sequenze di DNA o RNA e 8.169 sono mRNA. Si noti che queste 12.478 sequenze nucleotidiche potrebbero non essere tutte necessariamente per i geni del collagene, poiché alcuni dei record NCBI trovati potrebbero essere per altri geni ma contengono la parola "collagen" da qualche parte nel record NCBI (per esempio, nel titolo di un articolo citato). Tuttavia, un buon numero di essi sono probabilmente sequenze di collagene provenienti da nematodi.

In alternativa, per cercare sequenze nucleotidiche di collagene da vermi nematodi utilizziamo *rentrez*. Il database NCBI appropriato è "nucleotide"; per cercare i geni del collagene, possiamo specificare "collagen" come parola chiave nella nostra query. Pertanto, digitiamo:

```
> r_search <- entrez_search(db = 'nucleotide', term = 'Nematoda[ORGN] AND collagen')
> r_search
Entrez search result with 12478 hits (object contains 20 IDs and no web_history
object)
 Search term (as translated):  "Nematoda"[Organism] AND collagen[All Fields]
```

Q3.7 ***Quante sequenze di mRNA per i geni del collagene dei vermi nematodi sono presenti nel Database NCBI?***

Per rispondere ala domanda, nella casella di ricerca del sito web NCBI digitiamo: "Nematoda[ORGN] AND collagen AND biomol_mRNA[PROP]". Nella stringa di ricerca, "[ORGN]" specifica il nome del gruppo di specie, "collagen" specifica che vogliamo trovare le voci NCBI che includono la parola "collagen" e "[PROP]" specifica una proprietà di quelle sequenze (mRNA, in questo caso):



La pagina di ricerca visualizza: "Item: 1 to 20 of 8169", quindi sono state trovate 8.169 sequenze di mRNA da nematodi che contengono la parola "collagen" nel record NCBI. Delle 8.169, 6.976 sono sequenze in *GenBank* e 1.193 in *RefSeq*.

Si noti che nel Q3.6 abbiamo trovato 12.478 sequenze nucleotidiche di vermi nematodi. In questa interrogazione, abbiamo scoperto che solo 6.169 di queste sequenze sono sequenze di mRNA. Ciò significa che le altre (12.478-6.169 = 6.309) sequenze devono essere sequenze di DNA, o altri tipi di sequenze di RNA (non mRNA) come tRNA o rRNA.

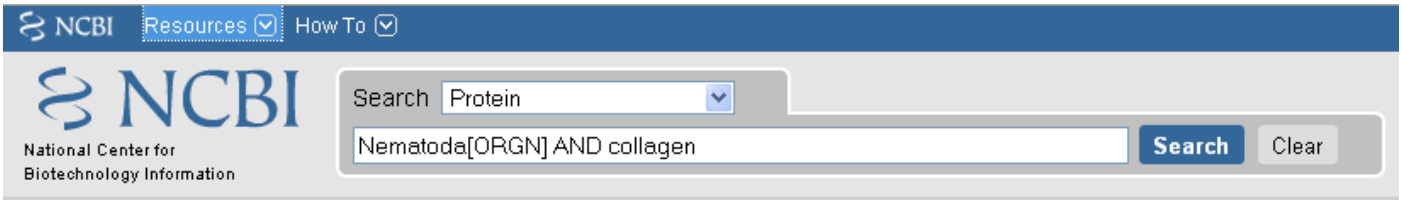
Per effettuare la stessa ricerca tramite *rentrez* ma restringendo "collagen" come parola chiave (e non su tutti i campi), digitiamo:

```
> r_search <- entrez_search(db = 'nucleotide', term = 'Nematoda[ORGN] AND
collagen[KYWD] AND biomol_mRNA[PROP] ')
> r_search
Entrez search result with 10 hits (object contains 10 IDs and no web_history object)
 Search term (as translated):  "Nematoda"[Organism] AND collagen[KYWD] AND biomol ...
```

In tal modo troviamo 10 sequenze di mRNA di nematode etichettate con la parola chiave "collagen", meno sequenze di quelle che si trovano quando si effettua una ricerca sul sito web della NCBI (12.478): la ricerca con la parola chiave "collagen" ha probabilmente più possibilità di raccogliere le vere sequenze di collagene, piuttosto che altre sequenze che contengono la parola "collagen" da qualche parte nelle loro voci NCBI.

Q3.8 ***Quante sequenze proteiche per le proteine del collagene dei vermi nematodi sono presenti nel database NCBI?***

Per rispondere, selezioniamo il database "Protein" sul sito web NCBI in quanto cercano sequenze proteiche. Poi digitiamo nella casella di ricerca: "Nematoda[ORGN] AND collagen" e premiamo <Search>:



La pagina dei risultati riporta: "Items: 1 to 20 of 8343". Ciò significa che sono state trovate 8.343 sequenze proteiche di vermi nematodi che includono la parola "collagen" nelle voci di sequenze NCBI [al 4 Gennaio 2020].

Per effettuare la stessa query utilizzando *rentrez* digitiamo:

```
> r_search <- entrez_search(db = 'protein', term = 'Nematoda[ORGN] AND collagen')
> r_search
Entrez search result with 8343 hits (object contains 20 IDs and no web_history
object)
 Search term (as translated):  "Nematoda"[Organism] AND collagen[All Fields]
```

Q3.9 ***Qual è l'accession NCBI per il genoma del*** **Trypanosoma cruzi*?***

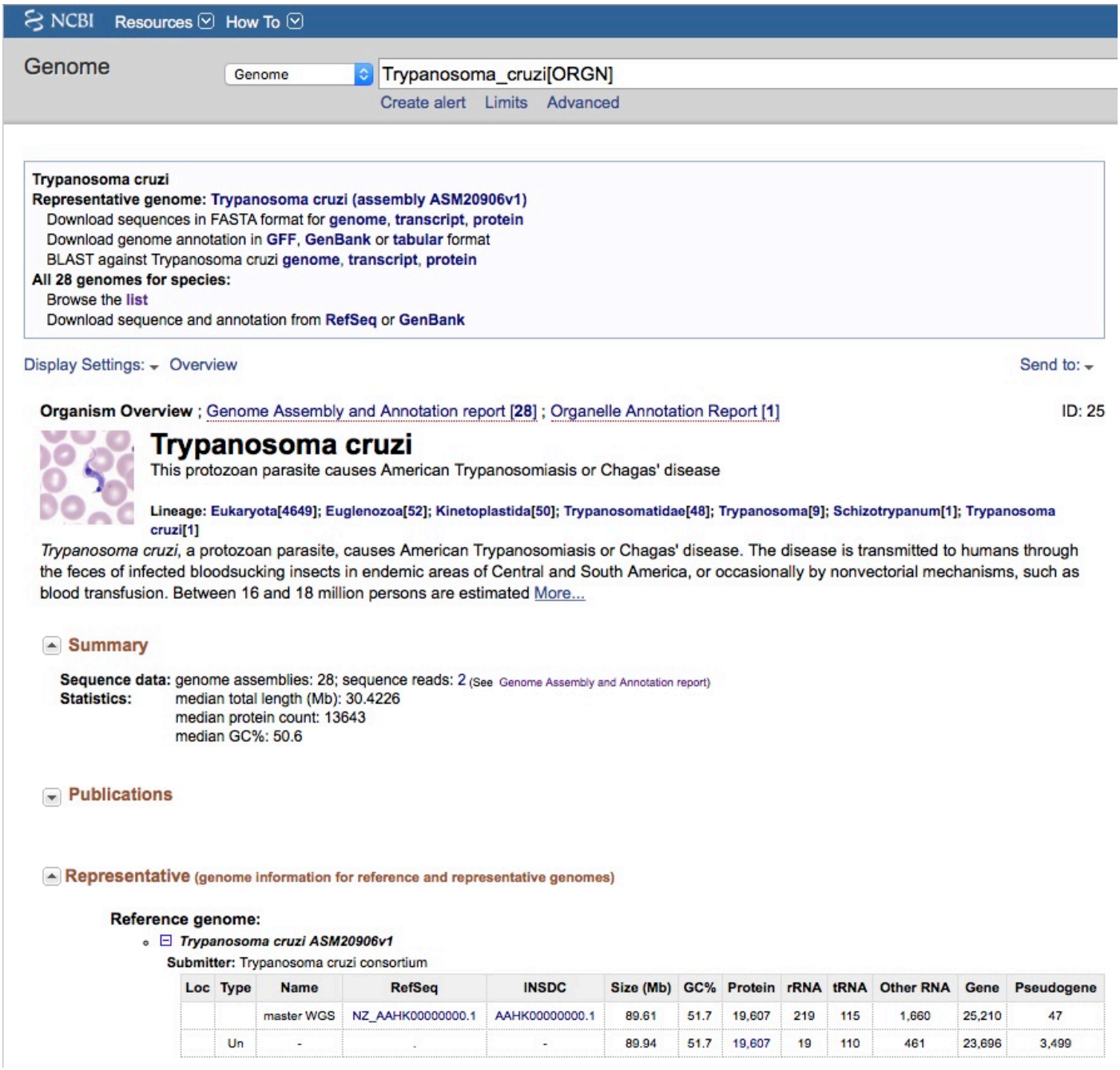
Nota: Il Trypanosoma cruzi causa il morbo di Chagas, classificato dall'OMS come malattia tropicale trascurata.

Ci sono due modi per rispondere a questa domanda. Il primo metodo è quello di andare sul sito web NCBI e selezionare il database "Genome" dall'elenco a discesa, in quanto si desidera cercare le sequenze del genoma. Poi, digitare nella casella di ricerca: "Trypanosoma_cruzi[ORGN]" e premere <Search>:

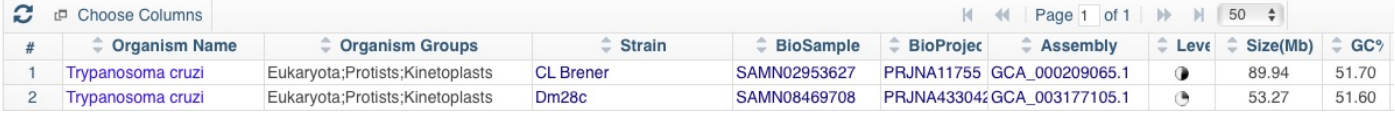
In questo modo si cercherà nel database del genoma del NCBI, che contiene sequenze di genomi completamente sequenziate. La pagina dei risultati elenca un solo record NCBI, la sequenza del genoma per il ceppo di *Trypanosoma cruzi CL Brener*, che ha l'*accession* RefSeq NZ_AAHK0000000000.1 (vedi pagina seguente).

Il secondo metodo per rispondere alla domanda è quello di andare direttamente alla pagina web dei genomi NCBI (https://www.ncbi.nlm.nih.gov/genome?db=Genome).

Clicchiamo sul link 'Browse by Organisms' in alto a sinistra della pagina nella sezione 'Using Genome', quindi sul link 'Eukariotes' (essendo il *Trypanosoma cruzi* una specie eucariotica) per ottenere un elenco completo di tutti i genomi eucarioti che sono stati completamente sequenziati.



Andiamo ora nella casella di ricerca in alto, digitiamo 'Trypanosoma cruzi' e premiamo il pulsante <Search>, ottenendo come risultato un elenco nel quale alla prima riga abbiamo il il ceppo di *Trypanosoma cruzi CL Brener* che ci interessa:



Ci sono anche diversi progetti di sequenziamento del genoma in corso per altri ceppi di *Trypanosoma cruzi*: Dm28c sulla riga 2 e altri (tra cui JR cl. 4, Sylvio X10/1, Y Esmerald cl. 3).

Nella settima colonna della tabella, "Assembly", sono disponibili (se presenti) gli assemblaggi genomici per i ceppi elencati.

Il nome dell'organismo (la seconda colonna "Organism Name") fornisce un link al record NCBI per la sequenza. In questo caso, il link per il ceppo di *Trypanosoma cruzi CL Brener* ci porta al record NCBI per

l'*accession* AAHK0100000000 (in realtà un *accession* per il progetto di sequenziamento del ceppo *T. cruzi CL Brener*, piuttosto che per la sequenza del genoma stesso) da dove è possibile scaricare la sequenza del genoma.

Q3.10 ***Quante specie di vermi nematodi completamente sequenziati sono rappresentati nel database NCBI del genoma?***

Per rispondere a questa domanda, andare alla pagina web NCBI del Genoma (https://www.ncbi.nlm.nih.gov/genome?db=Genome) e, nella casella di ricerca, digitare: "Nematoda[ORGN]" per cercare le sequenze del genoma del verme nematode, usando il nome latino dei vermi nematodi.

La pagina dei risultati riporta: "Items: 1 to 20 of 120", il che indica che sono state trovate 120 sequenze di genomi di vermi nematodi. Se si è interessati alle sole sequenze di genomi non mitocondriali (sequenze di genomi cromosomici), basta digitare 'Nematoda[ORGN] NOT mitochondrion' nella casella di ricerca.

Tra le 120 sequenze del risultato, che sono conunque tutte sequenze del genoma cromosomico per i vermi nematodi, sono incluse le specie *Caenorhabditis elegans*, *Caenorhabditis remanei*, *Caenorhabditis briggsae*, *Loa loa* (che causa la filariosi sottocutanea) e *Brugia malayi* (che causa la filariosi linfatica).

Si noti che quando si effettua una ricerca nel database NCBI Genome, si trovano i record NCBI per i genomi completamente sequenziati (in questo caso genomi di nematodi completamente sequenziati).

Se si è interessati a genomi parzialmente sequenziati, cioè a sequenze di progetti di sequenziamento di genomi ancora in corso, basta selezionare il database NCBI BioProject prima di effettuare la ricerca. Se si cerca "Nematoda[ORGN]" si vedrà che sono in corso progetti di sequenziamento del genoma per molte altre specie di nematodi (vi sono 2.194 progetti al 4 Gennaio 2020), tra cui le specie *Onchocerca volvulus* (che provoca l'oncocercosi), *Wuchereria bancrofti* (che provoca la filariosi linfatica) e *Necator americanus* (che provoca l'elmintiasi trasmessa dal suolo).

Q3.11 ***Quante sequenze proteiche del virus della rabbia sono presenti nel database NCBI delle proteine?***

Possiamo eseguire la ricerca usando il pacchetto *rentrez* sul database "protein":

```
> library(rentrez)
> r_search <- entrez_search("protein", term="rabies_virus[ORGN]")
> r_search
Entrez search result with 30881 hits (object contains 20 IDs and no web_history
object)
 Search term (as translated):  "Rabies lyssavirus"[Organism]
```

Abbiamo 30.881 risultati, il che significa che ci sono 30.881 sequenze proteiche del virus della rabbia nel database [al 4 Gennaio 2020]. Si noti che se si effettua questa ricerca in un secondo momento, è possibile trovare altre sequenze, dato che il database è in continua crescita.

Q3.12 ***Qual è l'accession NCBI per il genoma del virus*** **Mokola*?***

Eseguiamo la ricerca sul sito web NCBI selezionando il database "Genome" e digitando "Mokola_virus[ORGN]" nella casella di ricerca. Dopo aver premuto <Search> otteniamo un risultato nel database del Genoma corrispondente all'*accession* NC_006429, la sequenza del genoma del virus *Mokola*.

Q3.13 ***Scaricare da UniProt le sequenze della proteina*** **Brugia malayi Vab-3** ***(accession*** **XP_001899020*) e della proteina*** **Loa loa Vab-3** ***(accession*** **XP_020303145*) e salvarle in formato FASTA.***

Possiamo utilizzare le funzioni del pacchetto *rentrez* per recuperare le sequenze richieste e salvarle in formato FASTA:

```
> # Search for protein Brugia malayi Vab-3
> r_search <- entrez_search(db = "protein", term = "XP_001899020[ACCN]")
> tmp_fasta <- entrez_fetch(db = "protein", id = r_search$ids[1], rettype="fasta")
> write(tmp_fasta, file = "bugiamvab3prot.fasta")
> brugia <- seqinr::read.fasta(file = "bugiamvab3prot.fasta", as.string = FALSE)
> brugiaseq <- brugia[[1]]
> length(brugiaseq)
[1] 153
> brugiaseq
  [1] "M" "K" "L" "I" "V" "D" "S" "G" "H" "T" "G" "V" "N" "Q" "L" "G" "G" "V" "F" "V" "N" "G" "R" "P" "L" "P" "D"
 [28] "S" "T" "R" "Q" "K" "I" "V" "D" "L" "A" "H" "Q" "G" "A" "R" "P" "C" "D" "I" "S" "R" "I" "L" "Q" "V" "S" "N"
 [55] "G" "C" "V" "S" "K" "I" "L" "C" "R" "Y" "Y" "E" "S" "G" "T" "I" "R" "P" "R" "A" "I" "G" "G" "S" "K" "P" "R"
 [82] "V" "A" "T" "V" "S" "V" "C" "D" "K" "I" "E" "S" "Y" "K" "R" "E" "Q" "P" "S" "I" "F" "A" "W" "E" "I" "R" "D"
[109] "K" "L" "L" "H" "E" "K" "V" "C" "S" "P" "D" "T" "I" "P" "S" "A" "V" "V" "E" "A" "I" "I" "V" "I" "N" "Y" "A"
[136] "K" "Q" "N" "N" "N" "L" "L" "D" "R" "F" "I" "L" "P" "F" "S" "K" "L" "D"
> # Search for protein Loa loa Vab-3
> r_search <- entrez_search(db = "protein", term = "XP_020303145[ACCN]")
> tmp_fasta <- entrez_fetch(db = "protein", id = r_search$ids[1], rettype="fasta")
> write(tmp_fasta, file = "loaloavab3prot.fasta")
> loaloa <- seqinr::read.fasta(file = "loaloavab3prot.fasta", as.string = FALSE)
> loaloaseq <- loaloa[[1]]
> length(loaloaseq)
[1] 133
> loaloaseq
  [1] "M" "L" "G" "D" "K" "K" "Y" "S" "G" "H" "T" "G" "V" "N" "Q" "L" "G" "G" "V" "F" "V" "N" "G" "R" "P" "L" "P"
 [28] "D" "S" "T" "R" "Q" "K" "I" "V" "D" "L" "A" "H" "Q" "G" "A" "R" "P" "C" "D" "I" "S" "R" "I" "L" "Q" "V" "S"
 [55] "N" "G" "C" "V" "S" "K" "I" "L" "C" "R" "Y" "Y" "E" "S" "G" "T" "I" "R" "P" "R" "A" "I" "G" "G" "S" "K" "P"
 [82] "R" "V" "A" "T" "V" "S" "V" "C" "D" "K" "I" "E" "S" "Y" "K" "R" "E" "Q" "P" "S" "I" "F" "A" "W" "E" "I" "R"
[109] "D" "K" "L" "L" "H" "E" "K" "V" "C" "S" "P" "D" "T" "I" "P" "S" "I" "H" "V" "G" "F" "S" "I" "H" "F"
```

4. Allineamento a coppie di sequenze

Q4.1 ***Qual è il punteggio per l'allineamento globale ottimale tra la proteina*** **Brugia malayi Vab-3** ***e la proteina*** **Loa loa Vab-3***, quando si utilizza la matrice di punteggio* **BLOSUM50***, una penalità di apertura del gap di* **-10** *e una penalità di estensione del gap di* **-0,5***?*

Possiamo usare il pacchetto R *Biostrings* per rispondere al quesito, digitando:

```
> library(Biostrings) # load the Biostrings package
> data(BLOSUM50) # load the BLOSUM50 scoring matrix
> brugiaseqstring <- seqinr::c2s(c(brugiaseq)) # convert the Brugia sequence to string
> loaseqstring <- seqinr::c2s(c(loaloaseq)) # convert the Loa loa sequence to string
> brugiaseqstring <- toupper(brugiaseqstring) # convert the Brugia sequence to uppercase
> loaseqstring <- toupper(loaseqstring) # convert the Loa loa sequence to uppercase
> myglobalAlign <- pairwiseAlignment(brugiaseqstring, loaseqstring,
      substitutionMatrix = BLOSUM50,
      gapOpening = -10, gapExtension = -0.5, scoreOnly = FALSE) # align the two sequences
> myglobalAlign
Global PairwiseAlignmentsSingleSubject (1 of 1)
pattern: MKLIVD---SGHTGVNQLGGVFVNGRP...DKLLHEKVCSPDTIPSAVVEAIIVINYAKQNNNLLDRFILPFSKLD
subject: M--LGDKKYSGHTGVNQLGGVFVNGRP...DKLLHEKVCSPDTIPSIHVG-----------------FSIHF----
score: 771
```

Il punteggio di allineamento tra la sequenza proteica *Brugia malayi* e la sequenza proteica *Loa loa* è 771.

Q4.2 ***Utilizzare la funzione*** **printPairwiseAlignment()** ***per visualizzare l'allineamento globale ottimale tra la proteina*** **Brugia malayi Vab-3** ***e la proteina*** **Loa loa Vab-3***, utilizzando la matrice di punteggio* **BLOSUM50***, una penalità di apertura del gap di* **-10** *e una penalità di estensione del gap di* **-0,5***.*

La funzione `printPairwiseAlignment()` si trova nell'Appendice del testo di riferimento. La possiamo usare per visualizzare l'allineamento ottenuto nel quesito Q4.1:

```
> printPairwiseAlignment(myglobalAlign)
[1] "MKLIVD---SGHTGVNQLGGVFVNGRPLPDSTRQKIVDLAHQGARPCDISRILQVSNGCV 57"
[1] "M--LGDKKYSGHTGVNQLGGVFVNGRPLPDSTRQKIVDLAHQGARPCDISRILQVSNGCV 58"
[1] " "
[1] "SKILCRYYESGTIRPRAIGGSKPRVATVSVCDKIESYKREQPSIFAWEIRDKLLHEKVCS 117"
[1] "SKILCRYYESGTIRPRAIGGSKPRVATVSVCDKIESYKREQPSIFAWEIRDKLLHEKVCS 118"
[1] " "
[1] "PDTIPSAVVEAIIVINYAKQNNNLLDRFILPF 152"
[1] "PDTIPSIHVG-----------------FSIHF 152"
[1] " "
```

Le due proteine sono molto simili per tutta la loro lunghezza, con pochi *gap* e per lo più identità (pochi disallineamenti).

Q4.3 ***Quale punteggio globale di allineamento si ottiene per le due proteine*** **Vab-3***, quando si utilizza la matrice di allineamento* **BLOSUM62***, una penalità di apertura del gap di* **-10** *e una penalità di estensione del gap di* **-0,5***?*

Anche in questo caso possiamo usare il pacchetto *Biostrings* per rispondere al quesito, digitando:

```
> data(BLOSUM62) # load the BLOSUM62 scoring matrix
> myglobalAlign2 <- pairwiseAlignment(brugiaseqstring, loaseqstring,
      substitutionMatrix = BLOSUM62,
      gapOpening = -10, gapExtension = -0.5, scoreOnly = FALSE) # align the two sequences
> myglobalAlign2
Global PairwiseAlignmentsSingleSubject (1 of 1)
pattern: MKLIVD---SGHTGVNQLGGVFVNGRPLPDSTRQKIVDLAHQGARPCDISRI...FAWEIRDKLLHEKVCSPDTIPSAVVEAIIVINYAKQNNNLLDRFILPFSKLD
subject: M--LGDKKYSGHTGVNQLGGVFVNGRPLPDSTRQKIVDLAHQGARPCDISRI...FAWEIRDKLLHEKVCSPDTIPSIHVG--------------------FSIHF
score: 585
```

Il punteggio di allineamento quando si usa BLOSUM62 è 585, mentre il punteggio quando si usa BLOSUM50 è 771 (vedi quesito Q4.1). Possiamo stampare l'allineamento e vedere se l'allineamento fatto con BLOSUM62 è diverso da quello fatto con BLOSUM50:

```
> printPairwiseAlignment(myglobalAlign2)
[1] "MKLIVD---SGHTGVNQLGGVFVNGRPLPDSTRQKIVDLAHQGARPCDISRILQVSNGCV 57"
[1] "M--LGDKKYSGHTGVNQLGGVFVNGRPLPDSTRQKIVDLAHQGARPCDISRILQVSNGCV 58"
[1] " "
[1] "SKILCRYYESGTIRPRAIGGSKPRVATVSVCDKIESYKREQPSIFAWEIRDKLLHEKVCS 117"
[1] "SKILCRYYESGTIRPRAIGGSKPRVATVSVCDKIESYKREQPSIFAWEIRDKLLHEKVCS 118"
[1] " "
[1] "PDTIPSAVVEAIIVINYAKQNNNLLDRFILPFSKLD 156"
[1] "PDTIPSIHVG--------------------FSIHF 156"
[1] " "
```

L'allineamento fatto con la matrice di punteggio BLOSUM62 (score 585) è leggermente diverso da quello ottenuto con BLOSUM50 (score 771), quindi preferiamo l'allineamento ottenuto con la matrice BLOSUM50.

Q4.4 ***Qual è la significatività statistica dell'allineamento globale ottimale per le proteine*** **Brugia malayi** ***e*** **Loa loa Vab-3** ***realizzato con la matrice di punteggio*** **BLOSUM50*****, con una penalità di apertura del gap di*** **-10** ***e una penalità di estensione del gap di*** **-0,5*****?***

Per rispondere seguiamo un approccio *bootstrap*, producendo prima 1.000 sequenze casuali usando un modello multinomiale in cui le probabilità dei 20 amminoacidi sono impostate uguali alle loro frequenze nella proteina *Brugia malayi Vab-3*.

Faremo questo mediante la funzione `generateSeqsWithMultinomialModel()` (vedi Appendice testo di riferimento) come segue:

```
> randomseqs <- generateSeqsWithMultinomialModel(brugiaseqstring,1000)
```

Questo produce il vettore *randomseqs* contenente 1.000 sequenze casuali, ognuna della stessa lunghezza della proteina *Brugia malati Vab-3*.

Possiamo quindi allineare ciascuna delle 1.000 sequenze casuali con la proteina *Loa loa Vab-3* e memorizzare i punteggi per ciascuno dei 1.000 allineamenti nel vettore *randomscores*:

```
> randomscores <- double(1000) # Create a numeric vector with 1000 elements
> for (i in 1:1000) {
+ score <- pairwiseAlignment(loaseqstring, randomseqs[i],
+ substitutionMatrix = BLOSUM50, gapOpening = -10, gapExtension = -0.5,
+ scoreOnly = TRUE)
+ randomscores[i] <- score
+ }
```

Il punteggio per l'allineamento delle proteine *Brugia malayi* e *Loa loa Vab-3* utilizzando BLOSUM50 con una penalità di apertura del gap di -10 e una penalità di estensione del gap di -0,5 è stato di 771 (vedi quesito Q4.1). Possiamo vedere quale frazione dei 1.000 allineamenti tra le sequenze casuali (della stessa composizione di *Brugia malayi Vab-3*) e *Loa loa Vab-3* ha avuto punteggi uguali o superiori a 771:

```
> sum(randomscores >= 771)
[1] 0
```

Vediamo che nessuno dei 1.000 allineamenti ha avuto punteggi pari o superiori a 771. Quindi, il *p*-value per l'allineamento delle proteine *Brugia malayi* e *Loa loa Vab-3* è 0, e possiamo concludere che il punteggio di allineamento è statisticamente significativo (essendo inferiore a 0,05). Pertanto, è molto probabile che le proteine *Brugia malayi Vab-3* e *Loa loa loa Vab-3* siano omologhe (correlate).

Q4.5 ***Qual è il punteggio ottimale di allineamento globale tra la proteina*** **Brugia malayi Vab-6** ***e la proteina*** **Mycobacterium leprae chorismate lyase*?***

Per calcolare il punteggio di allineamento globale ottimale, dobbiamo prima recuperare la sequenza della proteina *M. leprae chorismate lyase* (*accession UniProt* Q9CD83):

```
> leprae <- retrieventrezseqs("Q9CD83","protein")
[1] "Retrieving sequence Q9CD83 ..."
> lepraeseq <- leprae[[1]]
> length(lepraeseq)
[1] 210
> lepraeseqstring <- seqinr::c2s(lepraeseq)
> lepraeseqstring <- toupper(lepraeseqstring)
```

Possiamo quindi allineare la sequenza della proteina *Brugia malayi Vab-3* alla sequenza della *M. leprae chorismate lyase*:

```
> myglobalAlign3 <- pairwiseAlignment(brugiaseqstring, lepraeseqstring,
      substitutionMatrix = BLOSUM50,
      gapOpening = -10, gapExtension = -0.5, scoreOnly = FALSE)
> myglobalAlign3
Global PairwiseAlignmentsSingleSubject (1 of 1)
pattern: M----------------KLIVDSGHTGVNQLGGVFVNGRPLPDST...VINY---------------AKQNNNLL----DRFILP---FSKLD
subject: MTNRTLSREEIRKLDRDLRILVATNGT-LTRVLNVVANEEIVVDII...TTEYFLRSVFQDTPREELDRCQYSNDIDTRSGDRFVLHGRVFKNL-
score: 59.5
```

Il punteggio di allineamento è di 59,5.

Possiamo visualizzare l'allineamento come segue:

```
> printPairwiseAlignment(myglobalAlign3)
[1] "M-----------------KLIVDSGHTGVNQLGGVFVNGRPLPDSTRQKIVDLAHQGARP 43"
[1] "MTNRTLSREEIRKLDRDLRILVATNGT-LTRVLNVVANEEIVVDIINQQLLDVA-----P 54"
[1] " "
[1] "-------CDISRILQ---VSNGCVSKILCRYYESGTI---RPRAIGG-----SKPRVATV 85"
[1] "KIPELENLKIGRILQRDILLKGQKSGILFVAAESLIVIDLLPTAITTYLTKTHHP-IGEI 113"
[1] " "
[1] "SVCDKIESYKREQ-------PSIFA----WEIRDKLLHEKVCSPDTIPSAVVEAIIVINY 134"
[1] "MAASRIETYKEDAQVWIGDLPCWLADYGYWDLPKRAVGRRY----RIIAGGQPVIITTEY 169"
[1] " "
[1] "----------------AKQNNNLL----DRFILP---FSKL 171"
[1] "FLRSVFQDTPREELDRCQYSNDIDTRSGDRFVLHGRVFKNL 221"
[1] " "
```

L'allineamento non sembra molto buono, contiene molte lacune e disallineamenti e poche corrispondenze.

Nel quesito Q4.4 abbiamo creato un vettore *randomseqs* che contiene 1.000 sequenze casuali generate utilizzando un modello multinomiale in cui le probabilità dei 20 aminoacidi sono impostate uguali alle loro frequenze nella proteina *Brugia malayi Vab-3*.

Per calcolare la significatività statistica dell'allineamento tra *Brugia malayi Vab-3* e *M. leprae chorismate lyase*, possiamo calcolare i punteggi di allineamento delle 1.000 sequenze casuali con *M. leprae chorismate lyase*:

```
> randomscores <- double(1000)
> for (i in 1:1000) {
+ score <- pairwiseAlignment(lepraeseqstring, randomseqs[i], substitutionMatrix="BLOSUM50",
+ gapOpening = -10, gapExtension = -0.5, scoreOnly = TRUE)
+ randomscores[i] <- score
+ }
```

Possiamo quindi vedere quanti dei 1.000 punteggi di allineamento casuali superano il punteggio di allineamento effettivo (59,5) tra *B. malayi Vab-3* e *M. leprae chorismate lyase*:

```
> sum(randomscores >= 59.5)
[1] 28
```

Vediamo che 28 dei 1.000 punteggi per le 1.000 sequenze casuali di *M. leprae chorismate lyase* sono superiori al punteggio di allineamento effettivo di 59,5. Pertanto il *p*-value per il punteggio di allineamento è 28/1.000 = 0,028, inferiore a 0,05, e quindi statisticamente significativo.

Tuttavia, in realtà dovremmo avere qualche dubbio sul fatto che l'allineamento sia statisticamente significativo. Infatti, le proteine *B. malayi Vab-3* e *M. leprae chorismate lyase* non sono note per essere omologhe (correlate), e quindi è probabile che il punteggio relativamente alto di allineamento (59,5) sia dovuto solo al caso.

5. Allineamenti multipli e alberi filogenetici

Q5.1 ***Calcolare un albero filogenetico "unrooted" con bootstrap, utilizzando il metodo dell'evoluzione minima.***

Possiamo cercare le funzioni R che costruiscono un albero utilizzando il metodo dell'evoluzione minima digitando:

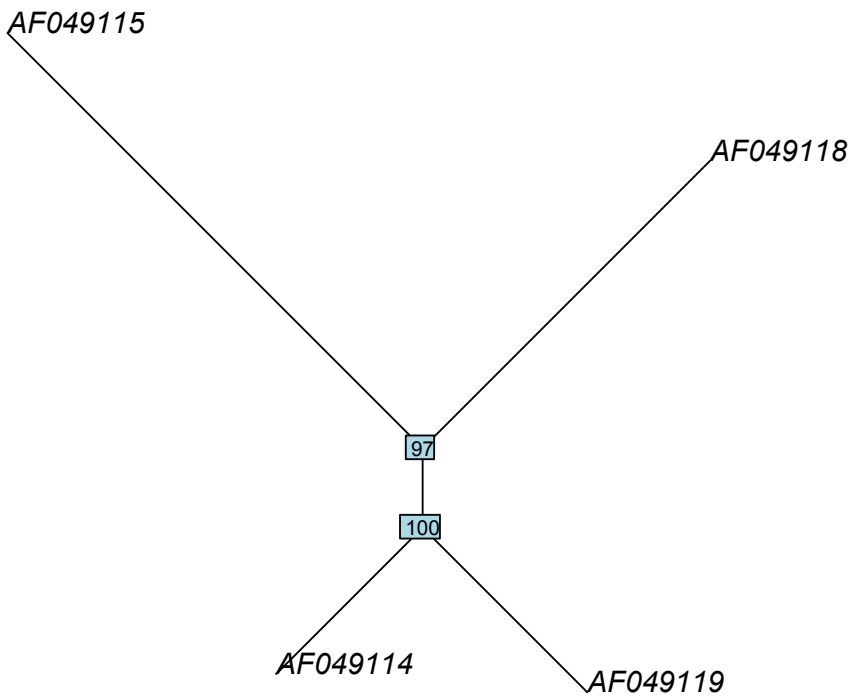
```
> help.search("evolution")
```

```
Help pages:
ade4::EH Amount of Evolutionary History
ade4::skulls Morphometric Evolution
ape::ape-package Analyses of Phylogenetics and Evolution
ape::evonet Evolutionary Networks
ape::FastME Tree Estimation Based on the Minimum Evolution Algorithm
ape::parafit Test of host-parasite coevolution
ggplot2::resolution Compute the "resolution" of a numeric vector
stats::convolve Convolution of Sequences via FFT
```

Troviamo che c'è una funzione "`FastME`" nel pacchetto *ape* per costruire un albero usando il metodo dell'evoluzione minima. Nella pagina di aiuto per questa funzione si vedrà che può essere eseguita digitando `fastme.bal()` o `fastme.ols()`, che sono due versioni diverse dell'algoritmo di minima evoluzione.

Possiamo anche utilizzare la funzione `unrootedMEtree()` (vedi Appendice) per costruire un albero usando l'algoritmo di minima evoluzione (la funzione usa `fastme.bal()`):

```
> virusmRNAaln  <- read.alignment(file = "virusmRNA.phy", format = "phylip")
> virusmRNAalntree <- unrootedMEtree(virusmRNAaln, type="DNA")
```



L'albero risultante è praticamente identico a quello ottenuto con l'algoritmo NJ (cfr. par. 5.12 del testo di riferimento "*Bioinformatica con R*").

Q5.2 ***Calcolare le distanze genetiche tra le seguenti proteine* NS1 *di diversi ceppi di virus* Dengue*: proteina* NS1 virus 1 *(UniProt* Q9YRRR4*), proteina* NS1 virus 2 *(UniProt* Q9YP96*), proteina* NS1 virus 3 *(UniProt* AAB52248*) e proteina* NS1 virus 4 *(UniProt* Q6TFL5*). Quali sono le proteine più strettamente correlate e quali sono le meno correlate, in base alle distanze genetiche?***

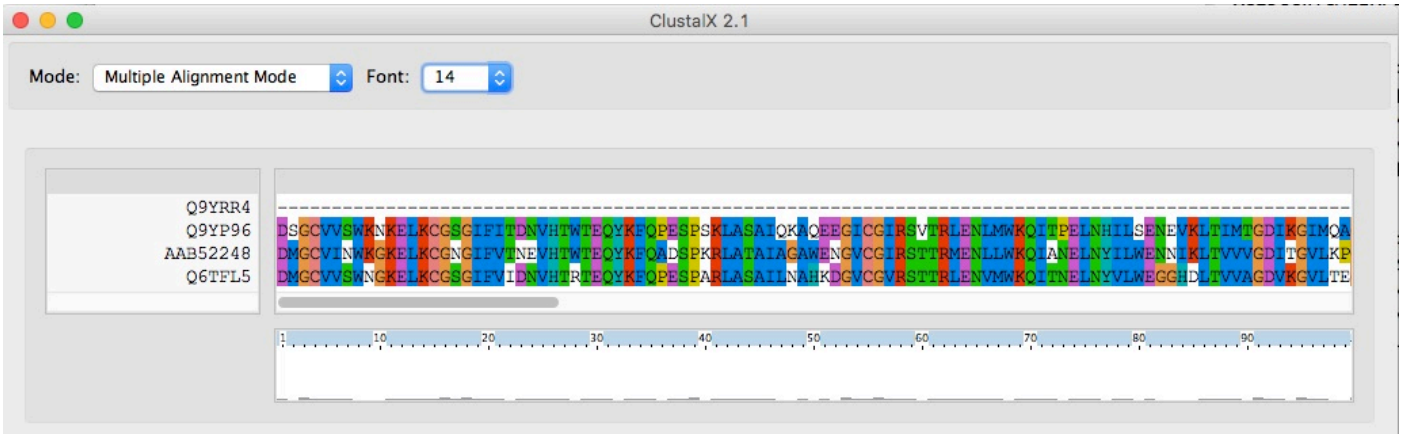
Per recuperare le sequenze in formato FASTA delle quattro proteine richieste, possiamo usare la funzione `retrieventrezseqs()` (vedi Appendice testo di riferimento):

```
> seqnames <- c("Q9YRR4", "Q9YP96", "AAB52248", "Q6TFL5")
> seqs <- retrieventrezseqs(seqnames,"protein")
[1] "Retrieving sequence for protein Q9YRR4 ..."
[1] "   ... found sequence for protein Q9YRR4 with ID 81982182"
[1] "Retrieving sequence for protein Q9YP96 ..."
[1] "   ... found sequence for protein Q9YP96 with ID 81981829"
[1] "Retrieving sequence for protein AAB52248 ..."
[1] "   ... found sequence for protein AAB52248 with ID 968130599"
[1] "Retrieving sequence for protein Q6TFL5 ..."
[1] "   ... found sequence for protein Q6TFL5 with ID 75545977"
```

Possiamo poi scrivere le sequenze in un file in formato FASTA chiamato "NS1.fasta", digitando:

```
> seqinr::write.fasta(seqs, seqnames, file="NS1.fasta")
```

Possiamo quindi utilizzare il software CLUSTAL per effettuare un allineamento multiplo delle sequenze proteiche in NS1.fasta, e memorizzarlo in un file di allineamento in formato PHYLIP chiamato "NS1.phy" (come descritto nel capitolo 5 del testo di riferimento):



Il passo successivo è quello di leggere l'allineamento in formato PHYLIP in R, e calcolare le distanze genetiche tra le sequenze proteiche, digitando:

```
> NS1aln  <- read.alignment(file = "NS1.phy", format = "phylip")
> NS1dist <- dist.alignment(NS1aln)
> NS1dist
            Q9YRR4     Q9YP96     AAB52248
Q9YP96      0.2544567
AAB52248    0.2244097  0.2870302
Q6TFL5      0.3058189  0.3328595  0.3107908
```

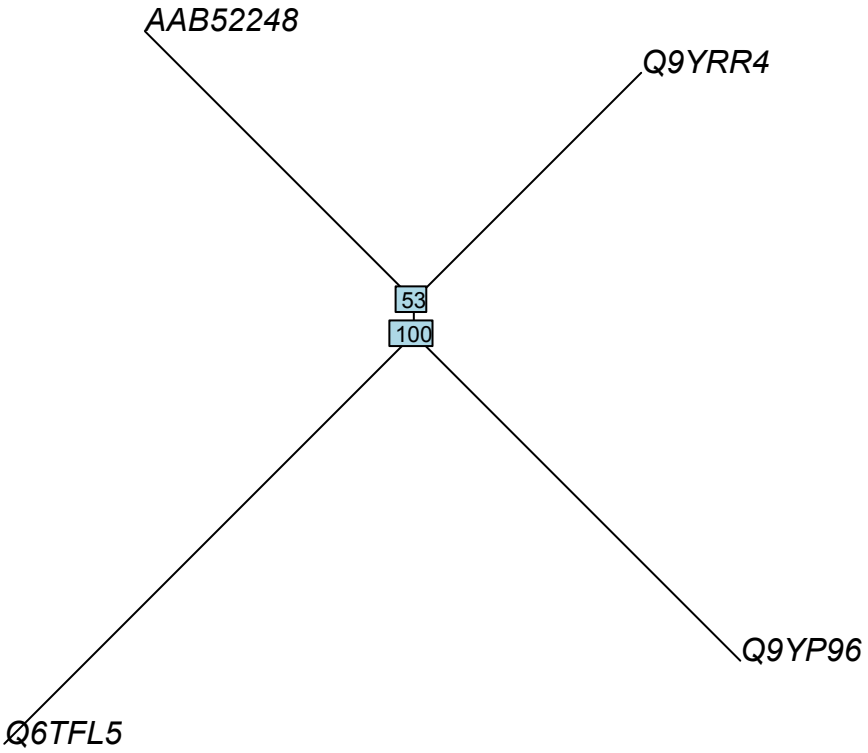
Vediamo che le due sequenze con la maggiore distanza genetica sono Q6TFL5 (*Dengue* virus 4 NS1) e Q9YP96 (*Dengue* virus 2 NS1), che hanno una distanza genetica di circa 0,33. Le due sequenze con la

distanza genetica più piccola sono Q9YRRR4 (*Dengue* virus 1 NS1) e AAB52248 (*Dengue* virus 3 NS1), che hanno una distanza genetica di circa 0,23.

Q5.3 ***Costruire un albero filogenetico "unrooted" delle proteine* NS1 *dei virus* Dengue 1, 2, 3 *e* 4, *utilizzando l'algoritmo Neighbour-Joining. Quali sono le proteine più strettamente correlate, in base all'albero? In base ai valori di bootstrap nell'albero, quanto siamo sicuri del risultato?***

Possiamo costruire un albero filogenetico *unrooted* delle proteine NS1 utilizzando l'algoritmo di *Neighbour-Joining* digitando:

```
> NS1alntree <- unrootedNJtree(NS1aln,type="protein")
Running bootstraps:       100 / 100
Calculating bootstrap values... done.
```



Vediamo nell'albero che Q6TFL5 (*Dengue* virus 4 NS1) e Q9YP96 (*Dengue* virus 1 NS1) sono raggruppati insieme, con un valore di bootstrap del 100%, che è un valore di bootstrap elevato, quindi siamo ragionevolmente sicuri di questo raggruppamento.

Le altre due proteine, AAB52248 (Dengue virus 3 NS1) e Q9YRR4 (Dengue virus 2 NS1) sono raggruppate insieme, ma il valore di bootstrap per il nodo che rappresenta l'antenato di questo clade è solo del 53%.

Non sorprende che Q6TFL5 e Q9YRRR4 siano raggruppate nell'albero filogenetico, essendo le due proteine più vicine quando abbiamo calcolato la distanza genetica (nel quesito Q5.2).

Q5.4 ***Costruire un albero filogenetico "unrooted" delle proteine* NS1 *dei virus* Dengue 1–4, *basato su un allineamento filtrato delle quattro proteine (mantenendo colonne di allineamento in cui almeno il 30%***

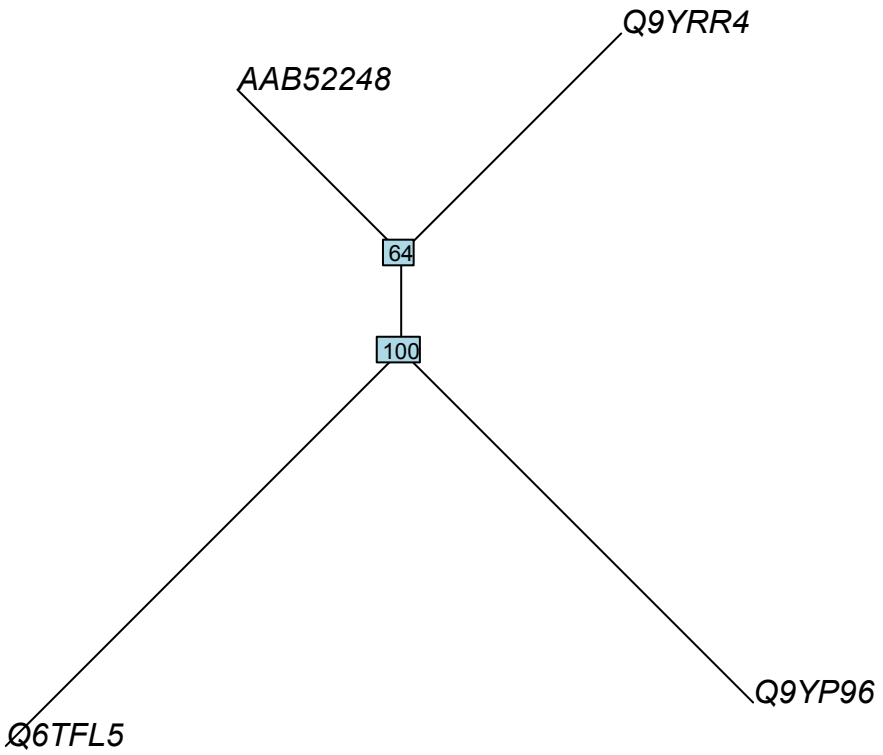
delle lettere non sono vuote e in cui almeno il 30% delle coppie di lettere sono identiche). Questo differisce dall'albero basato sull'allineamento non filtrato (nel quesito* Q5.3*)? Si può spiegarne il motivo?

Per filtrare l'allineamento delle proteine NS1 possiamo utilizzare la funzione `cleanAlignment()` (vedi Appendice testo di riferimento):

```
> cleanedNS1aln <- cleanAlignment(NS1aln, 30, 30)
```

Possiamo quindi costruire un albero senza radice (*unrooted*) in base all'allineamento filtrato:

```
> cleanedNS1alntree <- unrootedNJtree(cleanedNS1aln,type="protein")
Running bootstraps:       100 / 100
Calculating bootstrap values... done.
```



Vediamo che AAB52248 (Dengue virus 3 NS1) e Q9YRRR4 (Dengue virus 1 NS1) sono raggruppati insieme con un bootstrap del 64%, in accordo sia con quanto trovato nell'albero filogenetico basato sull'allineamento non filtrato (vedi quesito Q5.3) che con la matrice di distanza genetica (vedi quesito Q5.2).

Perché verificare la concordanza tra gli allineamenti filtrati e non filtrati è una buona idea visualizzare entrambi gli allineamenti (vedi le pagine seguenti).

Possiamo vedere che l'allineamento originale non filtrato contiene molte sequenze di *gap*. Questo probabilmente aggiunge "rumore" all'analisi filogenetica.

L'allineamento filtrato contiene meno colonne con *gap* (colonne dove due o più sequenze hanno dei *gap*) rispetto all'allineamento non filtrato. È probabile che le colonne con molte sequenze di *gap* nell'allineamento originale non filtrato aggiungessero rumore all'analisi filogenetica e che l'albero filogenetico basato sull'allineamento filtrato sia più affidabile in questo caso.

```
> printMultipleAlignment(NS1aln) # original unfiltered alignment
[1] "------------------------------------------------------------ 0"
[1] "DSGCVVSWKNKELKCGSGIFITDNVHTWTEQYKFQPESPSKLASAIQKAQEEGICGIRSV 60"
[1] "DMGCVINWKGKELKCGNGIFVTNEVHTWTEQYKFQADSPKRLATAIAGAWENGVCGIRST 60"
[1] "DMGCVVSWNGKELKCGSGIFVIDNVHTRTEQYKFQPESPARLASAILNAHKDGVCGVRST 60"
[1] " "
[1] "------------------------------------------------------------ 0"
[1] "TRLENLMWKQITPELNHILSENEVKLTIMTGDIKGIMQAGKRSLRPQPTELKYSWKAWGK 120"
[1] "TRMENLLWKQIANELNYILWENNIKLTVVVGDITGVLKPGKRTLTPPPMELKYSWKTWGK 120"
[1] "TRLENVMWKQITNELNYVLWEGGHDLTVVAGDVKGVLTEGKRALTPPVNDLKYSWKTWGK 120"
[1] " "
[1] "------------------------------------------------------------ 0"
[1] "AKMLSTESHNQTFLIDGPETAECPNTNRAWNSLEVEDYGFGVFTTNIWLKLKEKQDAFCD 180"
[1] "AKIVTAETQNSSFIIDGPNTPECPSASRAWNVWEVEDYGFGVFTTNIWLKLREMYTQLCD 180"
[1] "AKIFTLEARNSTFLIDGPDTSECPNERRAWNFLEVEDYGFGMFTTNIWMKFREGSSEVCD 180"
[1] " "
[1] "----------------DMGYWIESEKNETWKLARASFIEVKTCIWPKSHTLWSNGVWESE 44"
[1] "SKLMSAAIKDNRAVHADMGYWIESALNDTWKIEKASFIEVKNCHWPKSHTLWSNGVLESE 240"
[1] "HRLMSAAVKDERAVHADMGYWIESQKNGSWKLEKASLIEVKTCTWPKSHTLWSNGVLESD 240"
[1] "HRLMSAAIKDQKAVHADMGYWIESSKNQTWQIEKASLIEVKTCLWPKTHTLWSNGVLESQ 240"
[1] " "
[1] "MIIPKIYGGPISQHNYRPGYFTQTAGPWHLGKLELDFDLCEGTTVVVDEHCGNRGPSLRT 104"
[1] "MIIPKNFAGPVSQHNYRPGYHTQIAGPWHLGKLEMDFDFCDGTTVVVTEDCGNRGPSLRT 300"
[1] "MIIPKSLAGPISQHNYRPGYHTQTAGPWHLGKLELDFNYCEGTTVVITENCGTRGPSLRT 300"
[1] "MLIPRSYAGPFSQHNYRQGYATQTMGPWHLGKLEINFGECPGTTVAIQEDCGHRGPSLRT 300"
[1] " "
[1] "TTVTGKIIHEWCCRFCTLPPLRFRGEDGCWYGMEI------------------------ 139"
[1] "TTASGKLITEWCCRSCTLPPLRYRGEDGCWYGMEIRPLKEKEENLVNSLVTA-------- 352"
[1] "TTVSGKLIHEWCCRSCTLPPLRYMGEDGCWYGMEIRPINEKEENMVKSLVSAGSGKVDNF 360"
[1] "TTASGKLVTQWCCRSCAMPPLRFLGEDGCWYGMEIRPLSEKEENMVKSQVTA-------- 352"
[1] " "
[1] "----------------- 182"
[1] "----------------- 395"
[1] "TMGVLCLAILFEEVMRG 420"
[1] "----------------- 395"
[1] " "
```

```
> printMultipleAlignment(cleanedNS1aln) # filtered alignment
[1] "------------------------------------------------------------ 0"
[1] "DSGCVVSWKNKELKCGSGIFITDNVHTWTEQYKFQPESPKLASAIAEGICGIRSVTRLEN 60"
[1] "DMGCVINWKGKELKCGNGIFVTNEVHTWTEQYKFQADSPRLATAIAEGVCGIRSTTRMEN 60"
[1] "DMGCVVSWNGKELKCGSGIFVIDNVHTRTEQYKFQPESPRLASAIAKGVCGVRSTTRLEN 60"
[1] " "
[1] "------------------------------------------------------------ 0"
[1] "LMWKQITPELNHILSENKLTIMGDIKGIMGKRLRPQPELKYSWKAWGKAKMSENQTFLID 120"
[1] "LLWKQIANELNYILWENKLTVVGDITGVLGKRLTPPPELKYSWKTWGKAKITENSSFIID 120"
[1] "VMWKQITNELNYVLWEGDLTVVGDVKGVLGKRLTPPVDLKYSWKTWGKAKITENSTFLID 120"
[1] " "
[1] "-----------------------------------------------DMGYWIESKNT 11"
[1] "GPTECPNRAWNLEVEDYGFGVFTTNIWLKLKECDSKLMSAAIKDRAVHADMGYWIESLNT 180"
[1] "GPTECPSRAWNWEVEDYGFGVFTTNIWLKLRECDHRLMSAAVKDRAVHADMGYWIESKNS 180"
[1] "GPTECPNRAWNLEVEDYGFGMFTTNIWMKFRECDHRLMSAAIKDKAVHADMGYWIESKNT 180"
[1] " "
[1] "WKLARASFIEVKTCWPKSHTLWSNGVWESMIIPKGGPSQHNYRPGYTQTAGPWHLGKLED 71"
[1] "WKIEKASFIEVKNCWPKSHTLWSNGVLESMIIPKAGPSQHNYRPGYTQIAGPWHLGKLED 240"
[1] "WKLEKASLIEVKTCWPKSHTLWSNGVLESMIIPKAGPSQHNYRPGYTQTAGPWHLGKLED 240"
[1] "WQIEKASLIEVKTCWPKTHTLWSNGVLESMLIPRAGPSQHNYRQGYTQTMGPWHLGKLEN 240"
[1] " "
[1] "FCGTTVVVECGRGPSLRTTTVTGKIIHEWCCRFCTLPPLRFGEDGCWYGMEI-------- 123"
[1] "FCGTTVVVECGRGPSLRTTTASGKLITEWCCRSCTLPPLRYGEDGCWYGMEIRPLEKEEN 300"
[1] "FCGTTVVIECGRGPSLRTTTVSGKLIHEWCCRSCTLPPLRYGEDGCWYGMEIRPIEKEEN 300"
[1] "FCGTTVAIECGRGPSLRTTTASGKLVTQWCCRSCAMPPLRFGEDGCWYGMEIRPLEKEEN 300"
[1] " "
[1] "-------- 175"
[1] "LVNSLVTA 360"
[1] "MVKSLVSA 360"
[1] "MVKSQVTA 360"
[1] " "
```

Q5.5 ***Costruire un albero filogenetico "rooted" delle proteine*** **NS1** ***del virus*** **Dengue** ***basato su un allineamento filtrato, utilizzando la proteina del virus*** **Zika** ***come outgroup. Quali sono le proteine del virus*** **Dengue** ***più strettamente correlate, in base all'albero? Quali sono le informazioni aggiuntive che questo albero fornisce, rispetto all'albero "unrooted" del quesito*** **Q5.3*?***

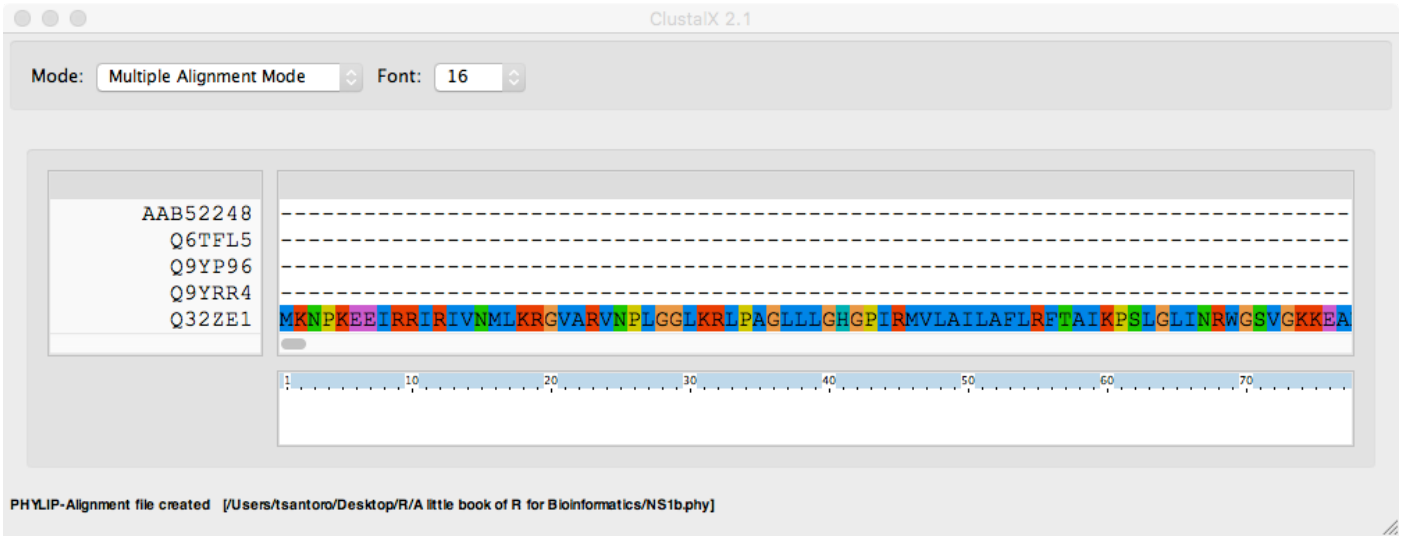
Per prima cosa dobbiamo aggiungere la proteina del virus *Zika* (*accession UniProt* Q32ZE1) alle sequenze:

```
> seqnames <- c("Q9YRR4", "Q9YP96", "AAB52248", "Q6TFL5", "Q32ZE1")
> seqs <- retrieventrezseqs(seqnames,"protein")
[1] "Retrieving sequence Q9YRR4 ..."
[1] "   ... found sequence for protein Q9YRR4 with ID 81982182"
[1] "Retrieving sequence Q9YP96 ..."
[1] "   ... found sequence for protein Q9YP96 with ID 81981829"
[1] "Retrieving sequence AAB52248 ..."
[1] "   ... found sequence for protein AAB52248 with ID 968130599"
[1] "Retrieving sequence Q6TFL5 ..."
[1] "   ... found sequence for protein Q6TFL5 with ID 75545977"
[1] "Retrieving sequence Q32ZE1 ..."
[1] "   ... found sequence for protein Q32ZE1 with ID 123879920"
```

Poi scriviamo le sequenze in un file in formato FASTA chiamato "NS1b.fasta":

```
> seqinr::write.fasta(seqs, seqnames, file="NS1b.fasta")
```

Utilizziamo quindi CLUSTAL per creare un allineamento in formato PHYLIP e lo salviamo come "NS1b.phy":

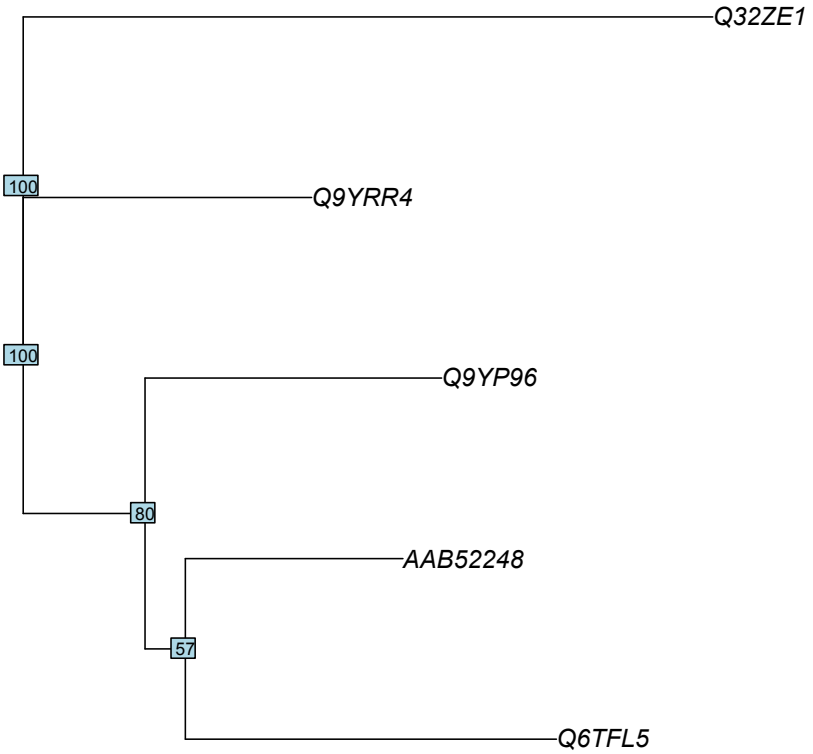


Poi leggiamo l'allineamento in R e scartiamo le colonne inaffidabili dall'allineamento:

```
> NS1baln <- read.alignment(file = "NS1b.phy", format = "phylip")
> cleanedNS1baln <- cleanAlignment(NS1baln, 30, 30)
```

Possiamo quindi costruire un albero filogenetico *rooted*, usando la proteina del virus *Zika* (*accession* Q32ZE1) come outgroup, con la funzione `rootedNJtree()` (vedi Appendice testo di riferimento):

```
> cleanedNS1balntree <- rootedNJtree(cleanedNS1baln, "Q32ZE1", type="protein")
Running bootstraps:       100 / 100
Calculating bootstrap values... done.
```



Vediamo in questo albero che AAB52248 (proteina *NS1* del virus *Dengue 3*) e Q6TFL5 (proteina *NS1* del virus *Dengue 4*) sono raggruppati insieme con bootstrap 57%. La sequenza più vicina è Q9YP96 (proteina *NS1* del virus *Dengue 2*).

La sequenza Q9YRRR4 (proteina *NS1* del virus *Dengue 1*) è la prima che diverge delle quattro proteine *NS1* del virus *Dengue*, in quanto è raggruppata con l'outgroup.

Nel quesito Q5.4 abbiamo trovato che AAB52248 (proteina *NS1* del virus *Dengue 3*) e Q9YRRR4 (proteina *NS1* del virus *Dengue 1*) erano raggruppate in un albero *unrooted*. L'attuale albero *rooted* è coerente con questo; ha AAB52248 e Q9YRRR4 come le due prime proteine *NS1 Dengue* divergenti, in quanto sono più vicine all'outgroup nell'albero.

Di conseguenza l'albero *rooted* ci dice quale delle proteine *NS1* del virus *Dengue* si è ramificata il più presto dagli antenati delle altre proteine, e quale si è ramificata dopo, e così via... Non era possibile rilevare questo dall'albero *unrooted*.

6. Ricerca genica (*gene-finding*) computazionale

Q6.1 ***Quanti ORF sono presenti sul forward strand del genoma del virus*** **Dengue DEN-1** ***(accession NCBI*** **NC_001477*)?***

Per trovare gli ORF sul filamento anteriore (*forward strand*) della sequenza del virus *Dengue* DEN-1, possiamo utilizzare la funzione `findORFsinSeq()`, che a sua volta richiede la funzione `findPotentialStartsAndStops()` (entrambe presenti nell'Appendice del testo di riferimento). Questa funzione richiede una stringa di caratteri come input, che va quindi prima convertita in una stringa di caratteri con `c2s()`:

```
> source("findPotentialStartsAndStops.R")
> source("findORFsinSeq.R")
> source("retrieventrezseqs.R")
> dengue <- retrieventrezseqs("NC_001477","nucleotide")
[1] "Retrieving sequence NC_001477 ..."
[1] "   ... found sequence for protein NC_001477 with ID 9626685"
> denguеseq <- dengue[[1]]
> denguеseqstring <- seqinr::c2s(denguеseq) # Convert the Dengue sequence to a string of characters
> mylist <- findORFsinSeq(denguеseqstring) # Find ORFs in "denguеseqstring"
> orflengths <- mylist[[3]] # Find the lengths of ORFs in "denguеseqstring"
> length(orflengths) # Find the number of ORFs that were found
  [1] 116
```

Risulta che ci sono 116 ORF sul *forward strand* del genoma del virus *Dengue DEN-1*.

Q6.2 ***Quali sono le coordinate dell'ORF più a destra (nella direzione*** **3'** ***, cioè l'ultimo) nel forward strand del genoma del virus*** **Dengue DEN-1*?***

Per rispondere a questa domanda, dobbiamo ottenere le coordinate degli ORF nel genoma del virus *DEN-1* come segue:

```
> denguеseqstring <- seqinr::c2s(denguеseq) # Convert the Dengue sequence to a string of characters
> mylist <- findORFsinSeq(denguеseqstring) # Find ORFs in "denguеseqstring"
> starts <- mylist[[1]] # Start positions of ORFs
> stops <- mylist[[2]] # Stop positions of ORFs
```

Il vettore *starts* contiene le coordinate iniziali dei potenziali codoni di *start*, e il vettore *stops* contiene le coordinate di fine dei potenziali codoni di *stop*. Sappiamo che ci sono 116 ORF sul forward strand (dal quesito Q6.1) e vogliamo le coordinate del 116° ORF. Quindi, digitiamo:

```
> starts[116]
  [1] 10705
> stops[116]
  [1] 10722
```

Questo indica che l'ORF più a destra in direzione 3' ha un potenziale codone di *start* a 10.705—10.707 e un potenziale codone di *stop* a 10.720–10.722. Quindi, le coordinate dell'ORF più a destra sono 10.705–10.722.

Q6.3 ***Qual è la sequenza proteica predetta dell'ORF più a destra (nella direzione 3', cioè l'ultimo) nel forward strand del genoma del virus*** **Dengue DEN-1*?***

Per ottenere la sequenza proteica predetta per il 3'-most ORF (posizioni 10.705–10.722) digitiamo:

```
> myorfvector <- dengueseq[10705:10722] # Get the DNA sequence of the ORF
> seqinr::translate(myorfvector)
  [1] "M" "E" "W" "C" "C" "*"
```

La sequenza proteica predetta dell'ORF è "MEWCC".

Q6.4 ***Quanti ORF ci sono di 30 nucleotidi o più nel forward strand della sequenza del genoma del virus*** **Dengue DEN-1*?***

La funzione `findORFsinSeq()` (vedi Appendice testo di riferimento) restituisce una variabile lista, il cui terzo elemento è un vettore che contiene le lunghezze degli ORF trovati. Così possiamo digitare:

```
> dengueseqstring <- seqinr::c2s(dengueseq) # Convert the Dengue sequence to a string of characters
> mylist <- findORFsinSeq(dengueseqstring) # Find ORFs in "dengueseqstring"
> orflengths <- mylist[[3]] # Find the lengths of ORFs in "dengueseqstring"
> summary(orflengths >= 30)
   Mode   FALSE    TRUE
logical      54      62
```

Questo indica che 62 ORF sul *forward strand* del virus *Dengue DEN-1* sono lunghi 30 o più nucleotidi.

Q6.5 ***Quanti ORF più lunghi di 248 nucleotidi sono presenti nel forward strand della sequenza genomica del virus*** **Dengue DEN-1*?***

Per rispondere al quesito, digitiamo:

```
> summary(orflengths >= 248)
   Mode   FALSE    TRUE
logical     114       2
```

Questo indica che ci sono 2 ORF di almeno 248 nucleotidi sul *forward strand* della sequenza genomica.

Q6.6 ***Se un ORF è lungo 248 nucleotidi, quale sarà la lunghezza in aminoacidi della sua sequenza proteica predetta?***

Se includiamo il codone di *stop* nella lunghezza dell'ORF, significa che le ultime tre basi dell'ORF non codificano alcun amminoacido. Pertanto, la lunghezza dell'ORF che codifica per gli aminoacidi è di 245 bp. Ogni aminoacido è codificato da 3 basi, quindi ci possono essere 245/3 = 81 aminoacidi; cioè, la sequenza proteica predetta sarà lunga 81 aminoacidi.

Q6.7 ***Quanti ORF ci sono sul forward strand del genoma del virus della rabbia (accession NCBI* NC_001542*)?***

Prima di tutto recuperiamo la sequenza del virus della rabbia digitando:

```
> rabies <- retrieventrezseqs("NC_001542","nucleotide")
[1] "Retrieving sequence NC_001542 ..."
[1] "   ... found sequence for NC_001542 with ID 9627197"
> rabiesseq <- rabies[[1]]
> length(rabiesseq)
[1] 11932
```

Troviamo infine gli ORF nel *forward strand* digitando:

```
> rabiesseqstring <- seqinr::c2s(rabiesseq) # Convert the rabies sequence to a string of characters
> rabieslist <- findORFsinSeq(rabiesseqstring) # Find ORFs in "rabiesseqstring"
> rabiesorflengths <- rabieslist[[3]] # Find the lengths of ORFs in "rabiesseqstring"
> length(rabiesorflengths) # Find the number of ORFs that were found
[1] 111
```

Nel *forward strand* vi sono quindi 111 ORF.

Q6.8 ***Qual è la lunghezza del più lungo ORF tra il 99% degli ORF più lunghi in 10 sequenze casuali della stessa lunghezza e composizione della sequenza del genoma del virus della rabbia?***

Generiamo 10 sequenze casuali usando un modello multinomiale in cui le probabilità delle 4 basi sono impostate uguali alle loro frequenze nella sequenza del genoma del virus della rabbia:

```
> randseqs <- generateSeqsWithMultinomialModel(rabiesseqstring, 10) # Generate 10 random
sequences using the multinomial model
> randseqorflengths <- numeric() # Tell R that we want to make a new vector of numbers
> for (i in 1:10) {
+ randseq <- randseqs[i] # Get the i-th random sequence
+ mylist <- findORFsinSeq(randseq) # Find ORFs in "randseq"
+ lengths <- mylist[[3]] # Find the lengths of ORFs in "randseq"
+ randseqorflengths <- append(randseqorflengths, lengths, after =
  length(randseqorflengths))
+ }
```

Per trovare la lunghezza dell'ORF più lungo tra il 99% degli ORF più lunghi nelle 10 sequenze casuali, troviamo il 99° quantile delle lunghezze degli ORF casuali:

```
> quantile(randseqorflengths, probs=c(0.99))
   99%
237
```

Cioè, l'ORF più lungo tra il 99% degli ORF più lunghi nelle sequenze casuali è di 237 nucleotidi.

Q6.9 ***Quanti ORF ci sono nel genoma del virus della rabbia che sono più lunghi della soglia di lunghezza trovata nel quesito*** **Q6.8*****?***

Per rispondere al quesito, digitiamo:

```
> summary(rabiesorflengths > 237)
   Mode   FALSE    TRUE
logical     105       6
```

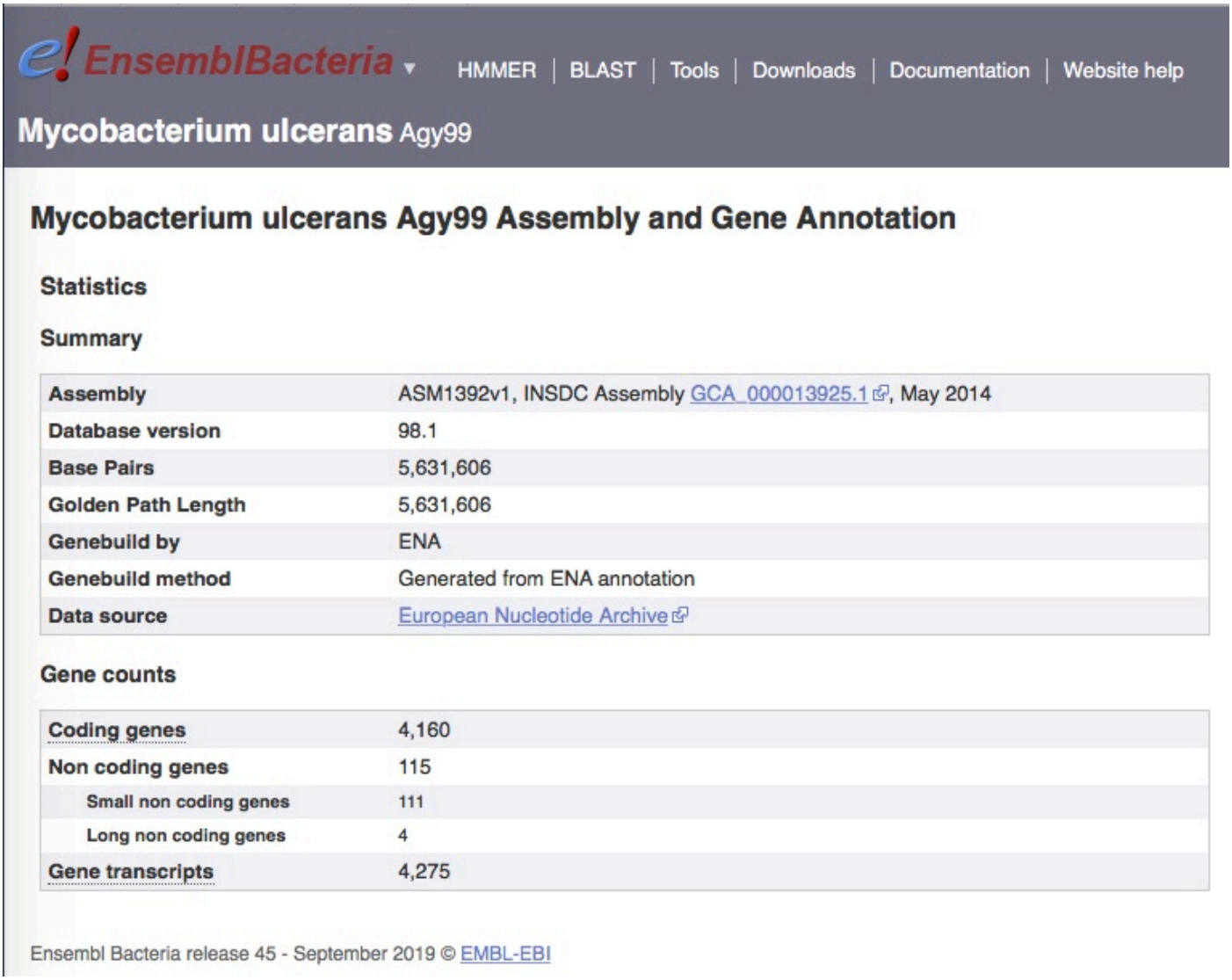
Ci sono 6 ORF nel genoma del virus della rabbia che sono più lunghi della lunghezza di soglia di 237 nucleotidi.

7. Genomica comparativa

Q7.1 ***Quanti geni di*** **Mycobacterium ulcerans** ***sono presenti nella versione attuale della banca dati Ensembl Bacteria?***

Per trovare la risposta basta andare sul sito web EnsemblBacteria (http://bacteria.ensembl.org/), digitare "Mycobacterium ulcerans" nella casella di ricerca intitolata "Search for a genome" e premere <Invio>.

Nella pagina successiva cliccare sul link "Mycobacterium ulcerans Agy99" della prima riga, e infine sul link "Information and statistics". Compare una pagina contenente varie informazioni riassuntive, tra cui il numero totale di geni (4.275) e il numero di geni codificanti proteine (4.160):



Purtroppo ad oggi [4 Gennaio 2020] non è ancora possibile interrogare con il pacchetto *biomaRt* (e *biomartr*) il mart "Bacteria" di Ensembl, quindi abbiamo dovuto risolvere il quesito facendo accesso direttamente al sito web https://bacteria.ensembl.org.

Q7.2 ***Quanti geni codificanti le proteine della*** **Leishmania major** ***hanno ortologhi nel*** **Plasmodium falciparum*****?***

Innanzitutto recuperiamo i geni della *Leishmania major* e del *Plasmodium falciparum* dal mart "Protists" di Ensembl mediante le funzioni del pacchetto *biomaRt*:

```
> ensemblprotists <- useMart("protists_mart", host = "protists.ensembl.org")
> ensemblprotists_datasets <- listDatasets(ensemblprotists)
> ensemblprotists_datasets[,1:2]
                dataset                                                    description
…
9          lmajor_eg_gene                                  Leishmania major genes (ASM272v2)
…
14     pfalciparum_eg_gene                             Plasmodium falciparum 3D7 genes (ASM276v2)
…
> ensembl_lm <- useDataset("lmajor_eg_gene", mart=ensemblprotists)
> ensembl_pf <- useDataset("pfalciparum_eg_gene", mart=ensemblprotists)
> lm_attributes <- listAttributes(ensembl_lm)
> lm_attributenames <- lm_attributes[[1]]
> length(lm_attributenames) # How many attributes are there?
[1] 757
> pf_attributes <- listAttributes(ensembl_pf)
> pf_attributenames <- pf_attributes[[1]]
> length(pf_attributenames) # How many attributes are there?
[1] 759
> lm_genes <- getBM(attributes = c("ensembl_gene_id"), mart=ensembl_lm)
> pf_genes <- getBM(attributes = c("ensembl_gene_id"), mart=ensembl_pf)
> lm_genenames <- lm_genes[[1]]
> length(lm_genenames) # How many genes are in Leishmania major?
[1] 10030
> pf_genenames <- pf_genes[[1]]
> length(pf_genenames) # How many genes are in Plasmodium falciparum?
[1] 5767
> lm_genes2 <- getBM(attributes=c("ensembl_gene_id","gene_biotype"), mart=ensembl_lm)
> pf_genes2 <- getBM(attributes=c("ensembl_gene_id","gene_biotype"), mart=ensembl_pf)
> lm_genenames2 <- lm_genes2[[1]]
> lm_genebiotypes2 <- lm_genes2[[2]]
> table(lm_genebiotypes2) # How many protein-coding genes are in Leishmania major?
lm_genebiotypes2
         ncRNA protein_coding     pseudogene           rRNA          snRNA           tRNA
          1130           8315             94             92            233            166
> pf_genes2 <- getBM(attributes=c("ensembl_gene_id","gene_biotype"), mart=ensembl_pf)
> pf_genenames2 <- pf_genes2[[1]]
> pf_genebiotypes2 <- pf_genes2[[2]]
> table(pf_genebiotypes2) # How many protein-coding genes are in Plasmodium falciparum?
pf_genebiotypes2
             ncRNA nontranslating_CDS     protein_coding         pseudogene               rRNA
               102                   4               5358                153                 44
             snRNA                sRNA               tRNA
                10                  17                 79
```

Sappiamo quindi che la *Leishmania major* ha 10.030 geni (di cui 8.315 codificanti proteine), mentre il *Plasmodium falciparum* ne ha 5.767 (di cui 5.358 codificanti proteine).

Vediamo ora se tra gli attributi della *Leishmania major* ne esista qualcuno che ci consenta di cercare i geni ortologhi nel *Plasmodium falciparum*:

```
> subset(lm_attributenames, grepl("pfalciparum", lm_attributenames))
 [1] "pfalciparum_eg_homolog_ensembl_gene"
…
 [9] "pfalciparum_eg_homolog_orthology_type"
…
```

Gli attributi che ci interessano sono il primo, "pfalciparum_eg_homolog_ensembl_gene" (che individua l'ortologo *Plasmodium falciparum* di un gene della *Leishmania major*), e il nono, "pfalciparum_eg_homolog_orthology_type" (che descrive il tipo di relazione ortologica tra un particolare gene della *Leishmania major* e il suo ortologo *Plasmodium falciparum*; ad es. uno-a-molti oppure molti-a-molti).

Possiamo ora recuperare gli identificatori Ensembl degli ortologhi nel *Plasmodium falciparum* di tutti i geni principali della *Leishmania* digitando:

```
> lm_genes <- getBM(attributes = c("ensembl_gene_id",
      "pfalciparum_eg_homolog_ensembl_gene","pfalciparum_eg_homolog_orthology_type"),
      mart=ensembl_lm)
> lm_genenames <- lm_genes[[1]]
> lm_pf_orthologues <- lm_genes[[2]]
> lm_pf_orthologuetypes <- lm_genes[[3]]
```

Il primo elemento della variabile lista *lm_genes* è un vettore di identificatori Ensembl di tutti i principali geni della *Leishmania*, il secondo elemento è un vettore di identificatori Ensembl per i loro ortologhi *Plasmodium falciparum* e il terzo è un vettore con informazioni sui tipi di ortologia (ad es. "ortholog_many2many", cioè molti-a-molti).

Possiamo visualizzare i nomi dei primi 10 geni principali della *Leishmania* e dei loro ortologhi *Plasmodium falciparum*, e dei loro tipi di ortologia, digitando:

```
> lm_genenames[1:10]
 [1] "LMJF_01_0780" "LMJF_01_0350" "LMJF_01_0280" "LMJF_01_0290" "LMJF_01_0335" "LMJF_01_0600"
 [7] "LMJF_01_0220" "LMJF_01_0220" "LMJF_01_0220" "LMJF_01_0220"
> lm_pf_orthologues[1:10]
 [1] ""              ""              ""              ""              ""              ""
 [7] "PF3D7_0528700" "PF3D7_1116300" "PF3D7_0804800" "PF3D7_1423200"
> lm_pf_orthologuetypes[1:10]
 [1] ""                    ""                    ""                    ""
 [5] ""                    ""                    "ortholog_many2many" "ortholog_many2many"
 [9] "ortholog_many2many" "ortholog_many2many"
```

Non tutti i geni principali della *Leishmania* hanno ortologhi *Plasmodium falciparum*. Per scoprire quanti geni della *Leishmania major* hanno ortologhi nel *Plasmodium falciparum*, dobbiamo prima trovare gli indici degli elementi del vettore *lm_pf_orthologues* che non sono vuoti:

```
> myindex <- lm_pf_orthologues!=""
```

Possiamo quindi scoprire il numero e i nomi dei geni del *Leishmania* corrispondenti a questi indici:

```
> lm_genenames2 <- lm_genenames[myindex]
> length(lm_genenames2)
[1] 2538
```

Si vede che 2.538 geni principali della *Leishmania* hanno ortologhi *Plasmodium falciparum*. Di questi, quanti sono geni che codificano proteine? Per rispondere a questa domanda, possiamo unire insieme (mediante la funzione R `merge()`) le informazioni della variabile lista *lm_genes2* (che contiene informazioni sul nome di ogni gene della *Leishmania major* e sul suo tipo) e la variabile lista *lm_genes* (che contiene informazioni sul nome di ogni gene *L. major* e sui suoi ortologhi *P. falciparum*).

```
> lm_genes3 <- merge(lm_genes2, lm_genes)
```

Il primo elemento della variabile lista *lm_genes3* contiene il vettore dei nomi dei geni principali della *Leishmania*, il secondo il vettore dei tipi di questi geni (ad es. "protein_coding", "pseudogene", ecc.) e il terzo elemento un vettore dei nomi degli ortologhi *Plasmodium falciparum*. Possiamo quindi scoprire quanti geni principali della *Leishmania* che codificano proteine sono privi di ortologhi *Plasmodium falciparum* digitando:

```
> lm_genenames <- lm_genes3[[1]]
> lm_genebiotypes <- lm_genes3[[2]]
> lm_pf_orthologues <- lm_genes3[[3]]
> myindex <- lm_pf_orthologues!="" & lm_genebiotypes=="protein_coding"
> lm_genenames2 <- lm_genenames[myindex]
> length(lm_genenames2)
[1] 2538
```

Si vede che ci sono 2.538 geni della *Leishmania* che codificano le proteine principali che hanno ortologhi nel *Plasmodium falciparum*.

Q7.3 ***Quanti geni codificanti le proteine della*** **Leishmania major** ***hanno ortologhi di tipo uno-a-uno nel*** **Plasmodium falciparum*?***

Proseguendo il quesito Q7.2, basta digitare:

```
> myindex <- lm_pf_orthologues!="" & lm_genebiotypes=="protein_coding" &
      lm_pf_orthologuetypes=="ortholog_one2one"
> table(myindex)
myindex
FALSE  TRUE
10502   227
```

Ci sono 227 geni proteon-coding della *L. major* con ortologhi di tipo uno-a-uno nel *P. falciparum*.

Q7.4 ***Quanti geni della*** **Leishmania major** ***hanno dominii Pfam?***

In base a quanto già calcolato nei quesiti precedenti, basta cercare l'attributo riguardante i dominii Pfam e poi filtrare la lista dei geni che hanno un dominio Pfam non vuoto:

```
> subset(lm_attributenames, grepl("pfam", lm_attributenames))
[1] "pfam"       "pfam_start" "pfam_end"
> lm_genes <- getBM(attributes = c("ensembl_gene_id","pfam"), mart=ensembl_lm)
> lm_genenames <- lm_genes[[1]]
> length(lm_genenames)
[1] 9956
> lm_pfamdomains <- lm_genes[[2]]
> summary(lm_pfamdomains!="")
   Mode   FALSE    TRUE
logical    3629    6327
```

Vi sono 6.327 geni *L. major* con un dominio Pfam.

Q7.5 ***Quali sono i 5 domini Pfam più comuni nei geni della*** **Leishmania major*****?***

La variabile vettore *lm_pfamdomains* contiene tutti i domini Pfam dei geni della *L. major*. Per rispondere, basta costruire la tabella dei conteggi per i domini e ordinarla in senso inverso, prelevandone i primi 5 elementi non vuoti:

```
> (sort(table(lm_pfamdomains), decreasing = TRUE))[1:6]
lm_pfamdomains
        PF00069 PF00400 PF00076 PF00271 PF07344
   3629     190      75      71      68      61
```

Quindi, i 5 domini Pfam più comuni (frequenti) nei geni della *L. major* sono: PF00069 (190 geni), PF00400 (75 geni), PF00076 (71 geni), PF00271 (68 geni) e PF07344 (61 geni). Si osservi che la prima posizione corrisponde ai 3.629 geni con dominio Pfam vuoto.

Q7.6 ***Quante copie di ciascuno dei 5 domini Pfam più comuni nei geni della*** **L. major** ***sono presenti nel*** **P. falciparum*****?***

Dobbiamo innanzitutto recuperare i geni del *P. falciparum*, quindi i relativi domini Pfam:

```
> pf_genes <- getBM(attributes = c("ensembl_gene_id","pfam"), mart = ensembl_pf)
> pf_genenames <- pf_genes[[1]]
> length(pf_genenames)
[1] 6655
> pf_pfamdomains <- pf_genes[[2]]
> summary(pf_pfamdomains!="")
   Mode   FALSE    TRUE
logical    2015    4640
```

Vi sono 4.640 geni *P. falciparum* con un dominio Pfam; tra questi il conteggio dei 5 domini Pfam più comuni nella *L. major* sono rilevabili dalla relativa tabella delle frequenze:

```
> tab <- table(pf_pfamdomains)
> tab[["PF00069"]]
[1] 82
> tab[["PF00400"]]
[1] 47
> tab[["PF00076"]]
[1] 63
> tab[["PF00271"]]
[1] 63
> tab[["PF07344"]]
Error in tab[["PF07344"]] : indice fuori limite
```

Vediamo quindi che nei geni del *P. falciparum* ci sono 82 copie del dominio PF00069, 47 copie del dominio PF00400, 63 copie del dominio PF00271 e nessuna copia del dominio PF07344.

Q7.7 ***Quante copie di ciascuno dei 5 domini Pfam più comuni nei geni del*** **P. falciparum** ***sono presenti nella*** **L. major*****?***

Basta eseguire gli stessi comandi del quesito precedente, scambiando gli organismi:

```
> (sort(table(pf_pfamdomains), decreasing = TRUE))[1:6]
pf_pfamdomains
        PF02009 PF00069 PF09687 PF05424 PF15445
   2015     158      82      78      68      66
> tab <- table(lm_pfamdomains)
> tab[["PF02009"]]
Error in tab[["PF02009"]] : indice fuori limite
> tab[["PF00069"]]
[1] 190
> tab[["PF09687"]]
Error in tab[["PF09687"]] : indice fuori limite
> tab[["PF05424"]]
Error in tab[["PF05424"]] : indice fuori limite
> tab[["PF15445"]]
Error in tab[["PF15445"]] : indice fuori limite
```

Nei geni della *L. major* ci sono 190 copie del dominio PF00069 e nessuna copia degli altri quattro domini Pfam più comuni nel *P. falciparum* (PF02009, PF09687, PF05424 e PF15445).

8. Modelli multinomiali e Hidden Markov Models

Q8.1 ***Qual è la probabilità della sequenza del genoma mitocondriale del*** **Brugia malayi** ***(accession NCBI*** **NC_004298***), secondo un modello multinomiale in cui le probabilità di A, C, G e T (p_A, p_C, p_G e p_T) sono uguali alla frazione di A, C, G e T del genoma mitocondriale dello*** **Schistosoma mansoni***?*

Per prima cosa dobbiamo calcolare le frequenze di A, C, G e T nella sequenza mitocondriale di *S. mansoni*. Possiamo farlo calcolando la tabella dei conteggi di A, C, G e T e dividendo i conteggi delle basi per la lunghezza totale della sequenza per ottenere le frequenze:

```
> smansoni <- retrieventrezseqs("NC_002545", db = "nucleotide")
[1] "Retrieving sequence NC_002545 ..."
> smansoniseq <- smansoni[[1]]
> length(smansoniseq)
[1] 14415
> mytable <- table(smansoniseq)
> mytable
smansoniseq
   A    C    G    T
3654 1228 3307 6226
> mytable <- mytable/length(smansoniseq) # Divide the counts by the sequence length, to get frequencies
> mytable
smansoniseq
         A          C          G          T
0.25348595 0.08518904 0.22941381 0.43191120
> freqA <- mytable[["A"]] # Get the frequency of A
> freqC <- mytable[["C"]] # Get the frequency of C
> freqG <- mytable[["G"]] # Get the frequency of G
> freqT <- mytable[["T"]] # Get the frequency of T
> probabilities <- c(freqA,freqC,freqG,freqT) # Make a vector containing the frequencies of A/C/G/T
> probabilities
[1] 0.25348595 0.08518904 0.22941381 0.43191120
```

Utilizzeremo la funzione `multinomialprob()` (vedi Appendice) che calcola la probabilità di una sequenza, dato un particolare modello multinomiale (con date probabilità p_A, p_C, p_G, p_T). La funzione prende come argomenti di input un vettore che contiene la sequenza del DNA e un vettore che contiene le probabilità p_A, p_C, p_G, p_T; la usiamo per calcolare la probabilità della sequenza mitocondriale di *B. malayi*, usando un modello multinomiale dove p_A, p_C, p_G, p_T sono impostati pari alla frazione di A, C, G e T nella sequenza mitocondriale di *S. mansoni* (che abbiamo già memorizzato nel vettore *probabilities*, vedi sopra):

```
> bmalayi <- retrieventrezseqs("NC_004298", db = "nucleotide")
[1] "Retrieving sequence NC_004298 ..."
> bmalayiseq <- bmalayi[[1]]
> length(bmalayiseq)
[1] 13657
> multinomialprob(bmalayiseq, probabilities)
[1] 0
```

In questo caso, la probabilità è così piccola che è effettivamente zero.

Q8.2 ***Generare una sequenza di DNA casuale lunga n basi utilizzando un modello multinomiale in cui le probabilità p_A, p_C, p_G e p_T sono impostate pari alla frazione di A, C, G e T nel genoma mitocondriale dello*** **Schistosoma mansoni.**

Nel quesito Q8.1 abbiamo memorizzato le frequenze di A, C, G e T nel genoma mitocondriale di *S. mansoni* in un vettore chiamato *probabilities*:

```
> probabilities
[1] 0.25348595 0.08518904 0.22941381 0.43191120
```

La funzione `generateSeqWithMultinomialModel()` (vedi Appendice) genera una sequenza casuale con un modello multinomiale, avente come argomenti la lunghezza della sequenza da generare e le probabilità delle diverse lettere. Nel nostro caso useremo le frazioni di A, C, G e T del genoma mitocondriale di *S. mansoni* memorizzate nel vettore *probabilities* per generare una sequenza di 10 basi:

```
> generateSeqWithMultinomialModel(10, probabilities)
 [1] "A" "T" "A" "T" "C" "T" "A" "G" "T" "T"
```

Ogni volta che chiamiamo la funzione, essa creerà una sequenza di 10 bp leggermente diversa:

```
> generateSeqWithMultinomialModel(10, probabilities)
 [1] "T" "C" "A" "A" "A" "A" "T" "G" "C" "G"
> generateSeqWithMultinomialModel(10, probabilities)
 [1] "A" "T" "G" "A" "A" "G" "G" "G" "A" "G"
```

Q8.3 ***Utilizzare la funzione*** **generateSeqWithMultinomialModel()** ***per calcolare una sequenza casuale di 20 lettere, utilizzando un modello multinomiale con p_A=0,28, p_C=0,21, p_G=0,22 e p_T=0,29.***

Per prima cosa definiamo un vettore *myprobabilities* contenente le probabilità di A, C, G e T:

```
> myprobabilities <- c(0.28,0.21,0.22,0.29)
```

Quindi possiamo usare la funzione `generateSeqWithMultinomialModel()` per calcolare una sequenza casuale di 20 bp, usando come input il vettore *myprobabilities*:

```
> generateSeqWithMultinomialModel(20, myprobabilities)
 [1] "C" "C" "A" "G" "T" "A" "A" "A" "G" "G" "G" "A" "C" "C" "A" "G" "C" "C" "T" "A"
```

Q8.4 ***Creare una matrice di transizione per un modello di Markov di una sequenza DNA generando le probabilità a piacere per ciascun nucleotide precedente nella sequenza.***

Dopo aver definito i nucleotidi, scegliamo le probabilità nell'ipotesi che ciascuno di essi sia il nucleotide scelto in precedenza, quindi creiamo la matrice di transizione:

```
> nucleotides <- c("A", "C", "G", "T")
> pA <- c(0.3, 0.2, 0.4, 0.1); pC <- c(0.2, 0.3, 0.3, 0.2)
> pG <- c(0.1, 0.2, 0.3, 0.4); pT <- c(0.5, 0.1, 0.1, 0.3)
> tr_matrix <- matrix(c(pA, pC, pG, pT), 4, 4, byrow = TRUE) # Create a 4x4 matrix
> rownames(tr_matrix) <- nucleotides; colnames(tr_matrix) <- nucleotides
> tr_matrix # Print out the transition matrix
    A   C   G   T
A 0.3 0.2 0.4 0.1
C 0.2 0.3 0.3 0.2
G 0.1 0.2 0.3 0.4
T 0.5 0.1 0.1 0.3
```

Ad esempio, l'elemento nella riga 2 ("C") e colonna 3 ("G") della matrice di transizione contiene la probabilità (p_{CG}) di scegliere un particolare nucleotide ("G") nella posizione corrente nella sequenza, dato un particolare nucleotide ("C") nella posizione precedente della sequenza.

Q8.5 ***Usare la funzione*** **generatemarkovseq()** ***(vedi Appendice testo di riferimento) per generare una sequenza di 20 nucleotidi usando il modello di Markov descritto nella matrice di transizione tr_matrix del quesito precedente, usando le probabilità di partenza*** $\Pi_A = 0.2$; $\Pi_C = 0.3$; $\Pi_G = 0.2$; $\Pi_T = 0.3$).

Digitiamo:

```
> init_probs <- c(0.2, 0.3, 0.2, 0.3)
> generatemarkovseq(tr_matrix, init_probs, 20)
 [1] "G" "T" "A" "T" "A" "A" "C" "A" "G" "T" "T" "G" "T" "A" "C" "C" "T" "T" "T" "A"
```

Q8.6 ***Definire una matrice di transizione per un modello Hidden Markov basato sui due stati "AT" e "GC", con probabilità*** $p_{AT} = (0{,}68; 0{,}32)$ ***di cambiare stato se il precedente era "AT",*** $p_{GC} = (0{,}11; 0{,}89)$ ***di cambiare stato se il precedente era "GC".***

```
> hmm_states <- c("AT", "GC") # Define the names of the states
> AT_probs <- c(0.68, 0.32) # Set the probabilities of switching states, where the previous state was "AT"
> GC_probs <- c(0.11, 0.89) # Set the probabilities of switching states, where the previous state was "GC"
> hmm_tr_matrix <- matrix(c(AT_probs, GC_probs), 2, 2, byrow = TRUE) # Create a 2x2 matrix
> rownames(hmm_tr_matrix) <- hmm_states
> colnames(hmm_tr_matrix) <- hmm_states
> hmm_tr_matrix # Print out the transition matrix
     AT   GC
AT 0.68 0.32
GC 0.11 0.89
```

Q8.7 ***Definire una matrice di emissione per lo stesso HMM precedente che contiene le probabilità di scegliere i quattro nucleotidi A/C/G/T*** $p_{A\_AT} = 0{,}4$, $p_{C\_AT} = 0{,}1$, $p_{G\_AT} = 0{,}1$ ***e*** $p_{T\_AT} = 0{,}4$ ***per lo stato "AT", e*** $p_{A\_GC} = 0{,}1$, $p_{C\_GC} = 0{,}4$, $p_{G\_GC} = 0{,}4$ ***e*** $p_{T\_GC} = 0{,}1$ ***per lo stato "GC".***

Definiamo la matrice di emissione come richiesto:

```
> AT_stateprobs <- c(0.4, 0.1, 0.1, 0.4)
> GC_stateprobs <- c(0.1, 0.4, 0.4, 0.1)
> hmm_em_matrix <- matrix(c(AT_stateprobs, GC_stateprobs), 2, 4, byrow = TRUE)
> rownames(hmm_em_matrix) <- hmm_states
> colnames(hmm_em_matrix) <- nucleotides
> hmm_em_matrix
     A   C   G   T
AT 0.4 0.1 0.1 0.4
GC 0.1 0.4 0.4 0.1
```

Per esempio, il valore nella seconda riga (stato "GC") e nella terza colonna (nucleotide "G") della matrice di emissione di cui sopra è 0,4, che corrisponde alla probabilità di scegliere una "G" quando ci si trova nello stato "GC".

Q8.8 ***Usare la funzione*** **generatehmmseq()** ***per generare una sequenza di 10 nucleotidi utilizzando l'HMM del quesito precedente e probabilità di partenza uniformi (cioè*** $\Pi_{AT} = \Pi_{GC} = 0{,}5$***).***

Richiamiamo la funzione `generatehmmseq()` (vedi Appendice testo di riferimento) per generare una sequenza di 10 nucleotidi con il modello nascosto di Markov basato sul quesito precedente:

```
> generatehmmseq(hmm_tr_matrix, hmm_em_matrix, c(0.5,0.5), 10)
[1] "Position 1 , State AT , Nucleotide =  T"
[1] "Position 2 , State AT , Nucleotide =  A"
[1] "Position 3 , State GC , Nucleotide =  C"
[1] "Position 4 , State GC , Nucleotide =  C"
[1] "Position 5 , State AT , Nucleotide =  A"
[1] "Position 6 , State AT , Nucleotide =  G"
[1] "Position 7 , State AT , Nucleotide =  C"
[1] "Position 8 , State AT , Nucleotide =  A"
[1] "Position 9 , State AT , Nucleotide =  A"
[1] "Position 10 , State AT , Nucleotide =  G"
```

Q8.9 ***Date le matrici di transizione e di emissione dell'HMM precedente, e la sequenza di DNA*** **"TACCGATACGCGATGCTAGC"***, utilizzare la funzione* **viterbi()** ***per verificare lo stato dell'HMM che più probabilmente ha generato il nucleotide in ogni posizione della sequenza.***

```
> viterbi(s2c("TACCGATACGCGATGCTAGC"), hmm_tr_matrix, hmm_em_matrix)
[1] "Positions 1 - 2 Most probable state =  AT"
[1] "Positions 3 - 6 Most probable state =  GC"
[1] "Positions 7 - 8 Most probable state =  AT"
[1] "Positions 9 - 13 Most probable state =  GC"
[1] "Positions 14 - 14 Most probable state =  AT"
[1] "Positions 15 - 17 Most probable state =  GC"
[1] "Positions 18 - 18 Most probable state =  AT"
[1] "Positions 19 - 20 Most probable state =  GC"
```

Si noti che la sequenza va fornita alla funzione come vettore di caratteri.

9. Grafi di interazione proteica

Q9.1 ***Quanti vertici (proteine) e connessioni (interazioni proteina-proteina) ci sono nel grafo di de Lichtenberg? Vi sono circa 6.600 geni nel genoma* S. cerevisiae*: è lo stesso numero di vertici nel grafo dei dati di de Lichtenberg? In caso contrario, spiegarne il motivo.***

Nota: de Lichtenberg *et al.* hanno identificato i complessi di interazione proteica nel *Saccharomyces cerevisiae* che si formano durante il ciclo cellulare del lievito. Il dataset delle coppie di proteine interagenti è disponibile al sito web http://www.cbs.dtu.dk/databases/cellcycle/yeast_complexes/binary_interaction_data.txt. Il documento completo di de Lichtenberg *et al.* è disponibile all'indirizzo https://www.researchgate.net/profile/Soren_Brunak/publication/8040724_Dynamic_Complex_Formation_During_the_Yeast_Cell_Cycle/links/55d20abb08aec1b0429dcd47/Dynamic-Complex-Formation-During-the-Yeast-Cell-Cycle.pdf.

Per trovare il numero di nodi (proteine) e connessioni (interazioni) nel grafo del dataset di de Lichtenberg scarichiamo dal sito indicato il file "binary_interaction_data.txt", leggiamo in un grafo R il dataset delle coppie di proteine interagenti e quindi ne esaminiamo la struttura:

```
> thegraph <- makeproteingraph("binary_interaction_data.txt")
> thegraph
A graphNEL graph with undirected edges
Number of Nodes = 300
Number of Edges = 712
```

Quindi nel grafo vi sono 300 vertici e 712 connessioni. Il numero di vertici del grafo è molto inferiore al numero di geni del *S. cerevisiae* ovviamente perché i vertici rappresentano le proteine (sequenze di aminoacidi) per le quali i geni (sequenze di nucleotidi) codificano.

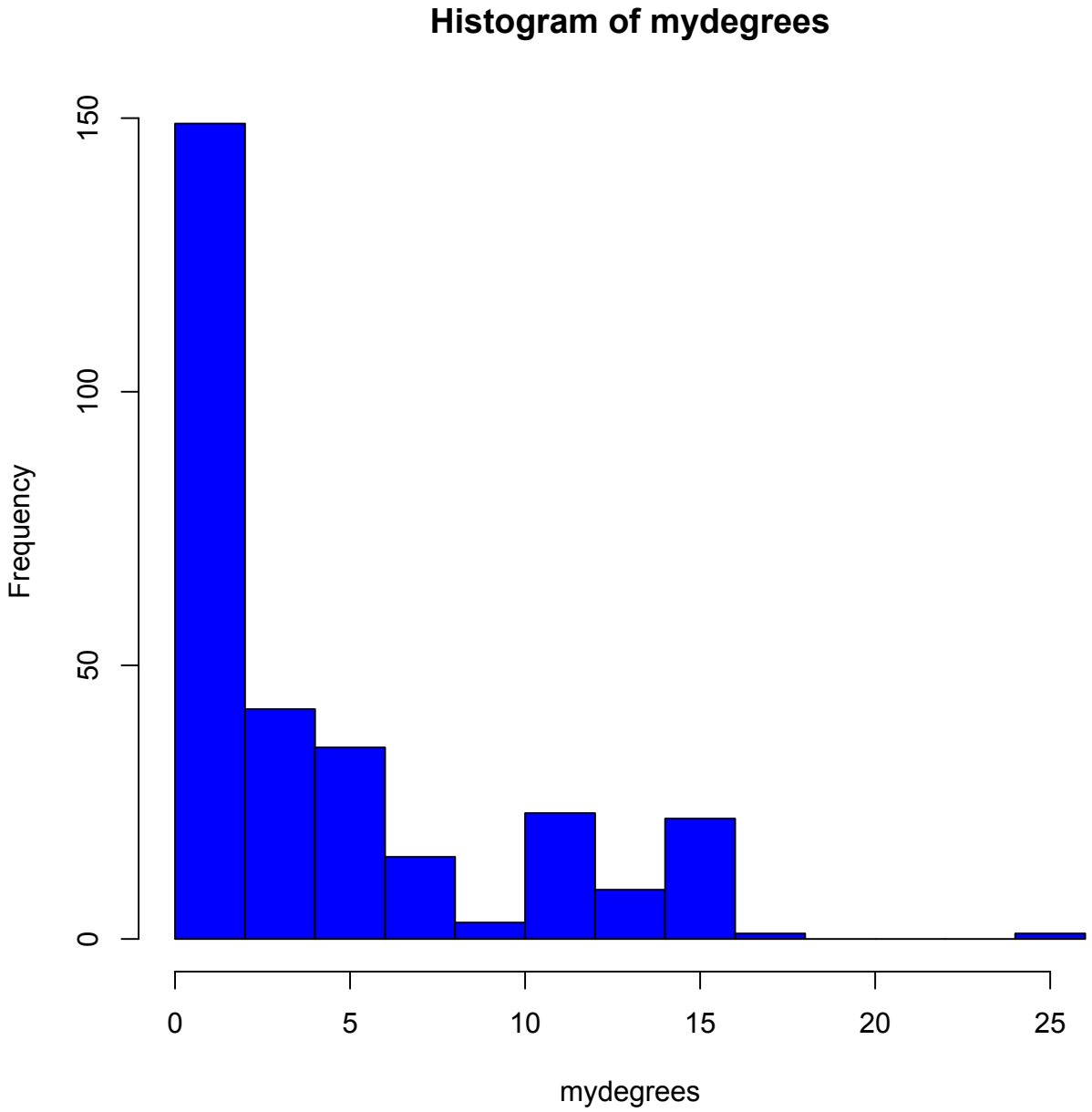
Q9.2 ***Qual è il numero minimo, massimo e medio di interazioni per le proteine nel grafo dei dati di de Lichtenberg? Si rileva un esempio di proteina hub? Fare un istogramma del numero di interazioni per le proteine* S. cerevisiae *nel dataset di de Lichtenberg.***

I numeri di interazioni richiesti corrispondono ai gradi dei nodi del grafo. Quindi per rispondere calcoliamo il grado di ciascun nodo mediante la funzione `degree()` del pacchetto *graph*:

```
> mydegrees <- graph::degree(thegraph)
> table(mydegrees)
mydegrees
 1  2  3  4  5  6  7  8  9 11 12 13 15 16 18 26
97 52 30 12 22 13 11  4  3  9 14  9 19  3  1  1
> min(mydegrees)
[1] 1
> max(mydegrees)
[1] 26
> mean(mydegrees)
[1] 4.746667
```

Il numero minimo di interazioni è quindi 1, quello massimo 26 e quello medio circa 5. Al nodo di grado massimo corrisponde probabilmente una proteina *hub*. Tracciamo infine l'istogramma del numero di interazioni per le proteine del *S. cerevisiae*:

```
> hist(mydegrees, col="blue")
```



Q9.3 ***Realizzare un grafo random con lo stesso numero di vertici e connessioni del grafo dei dati di de Lichtenberg. Qual è il grado minimo, massimo e medio dei vertici del grafo random? C'è una differenza nel grado minimo, massimo e medio dei vertici del grafo random rispetto all'altro grafo? Confrontare l'istogramma della distribuzione dei gradi per i due grafi.***

Realizziamo un grafo random con i 300 nodi e le 712 connessioni del grafo di de Lichtenberg mediante la funzione `makerandomgraph()` (vedi Appendice testo di riferimento):

```
> myrandomgraph <- makerandomgraph(300,712)
```

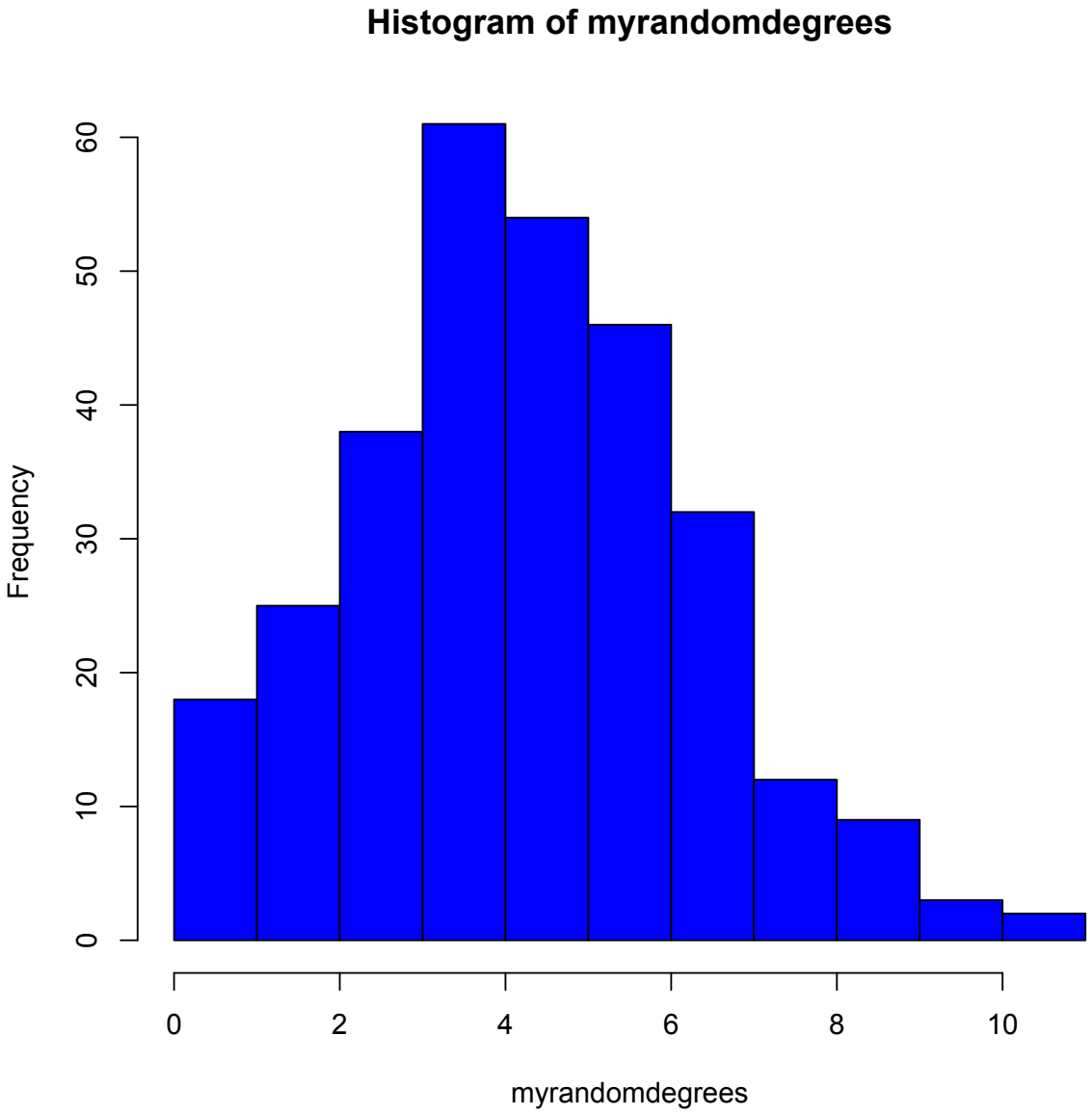
Esaminiamo la struttura dei gradi:

```
> myrandomdegrees <- graph::degree(myrandomgraph)
> table(myrandomdegrees)
myrandomdegrees
 0  1  2  3  4  5  6  7  8  9 10 11
 1 17 25 38 61 54 46 32 12  9  3  2
> min(myrandomdegrees)
[1] 0
> max(myrandomdegrees)
[1] 11
> mean(myrandomdegrees)
[1] 4.746667
```

Il grado minimo è zero (c'è un nodo isolato), quello massimo è 11 (inferiore al grafo di de Lichtenberg) e quello medio (circa 5) è invece identico.

Tracciamo infine l'istogramma del grafo *random*:

```
> hist(myrandomdegrees, col="blue")
```



che presenta una struttura quasi simmetrica "a campana" (essendo casale) rispetto alla distribuzione asimmetrica dei gradi dell'altro grafo.

Q9.4 ***Quante componenti connesse esistono nel grafo dei dati di de Lichtenberg? Quante componenti connesse contengono solo 2 proteine? Fare un grafico della componente connessa più grande del grafo di de Lichtenberg.***

Utilizziamo la funzione `connectedComp()` del pacchetto *RGBL* per isolare le 53 componenti connesse del grafo:

```
> library(RBGL)
> myconnectedcomponents <- connectedComp(thegraph)
> length(myconnectedcomponents)
[1] 53
```

Per elencare le componenti connesse con 2 sole proteine eseguiamo prima il `summary()` della lista *myconnectedcomponents* e quindi facciamo il conteggio:

```
> s = summary(myconnectedcomponents)
> s
   Length Class  Mode
1     3    -none- character
2     3    -none- character
3   104    -none- character
4     6    -none- character
5    19    -none- character
6     2    -none- character
7    30    -none- character
8     2    -none- character
9     3    -none- character
…

> num = as.integer(s[,1]); length(num[num==2])
[1] 31
```

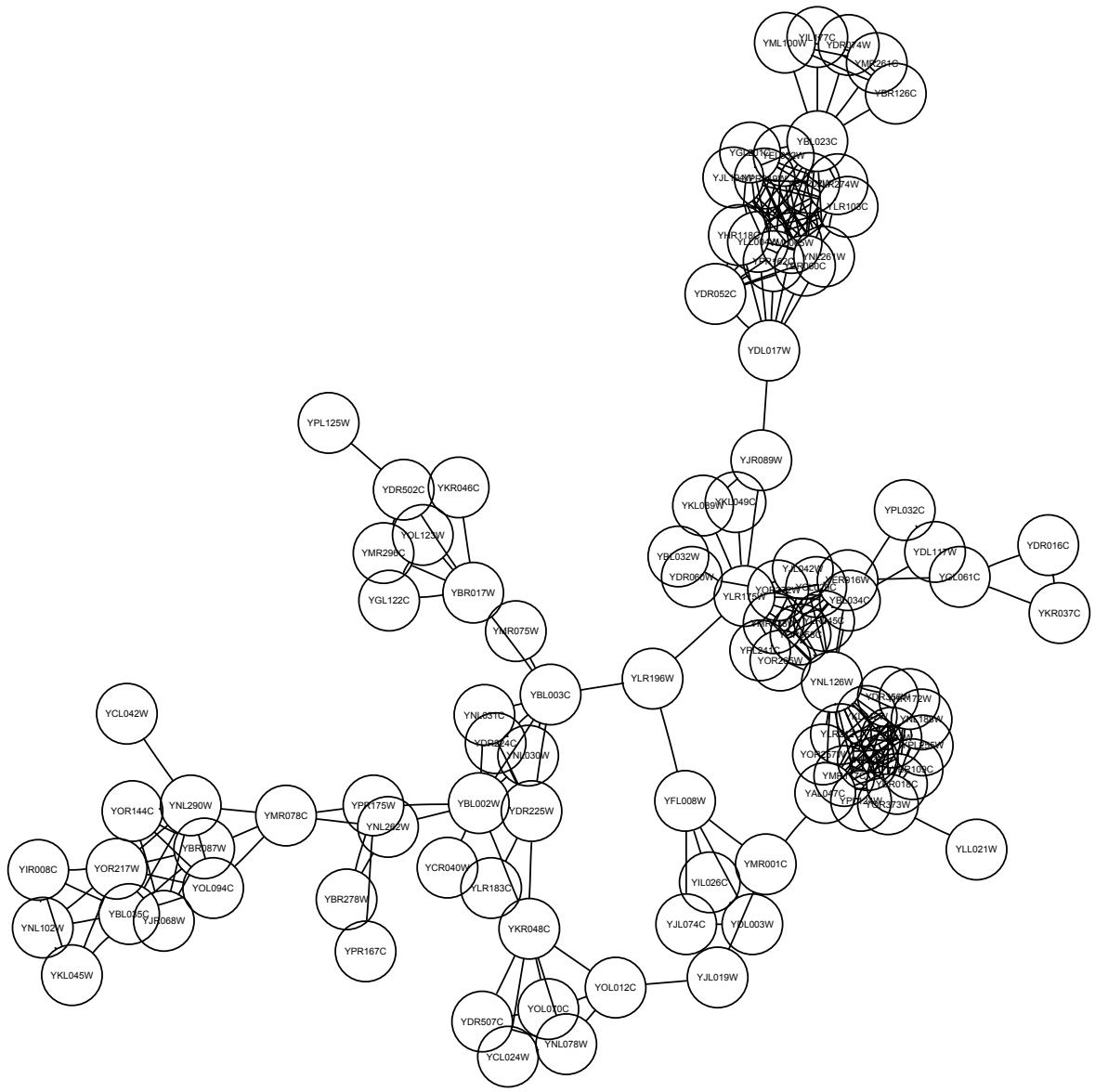
Vi sono 31 componenti connesse con 2 sole proteine.

Per tracciare il grafico richiesto, troviamo prima la componente connessa più grande:

```
> sort(s[,1], decreasing = TRUE, index.return = TRUE)[[1]][1]
    3
"104"
```

che è la terza, con 104 nodi (proteine). Quindi isoliamo la terza componente (che ha 104 nodi e 418 connessioni) e ne tracciamo il grafico mediante le funzioni del pacchetto *Rgraphviz*:

```
> library(Rgraphviz)
> maxcomponent <- myconnectedcomponents[[3]]
> mysubgraph <- subGraph(maxcomponent, thegraph)
> mysubgraph
A graphNEL graph with undirected edges
Number of Nodes = 104
Number of Edges = 418
> mygraphplot <- layoutGraph(mysubgraph, layoutType="neato")
> renderGraph(mygraphplot)
```



Q9.5 ***Con quali proteine interagisce la proteina del lievito*** **YPR119W*****, nel dataset di de Lichtenberg? Tracciare un grafico della componente connessa a cui appartiene la proteina*** **YPR119W*****: si riesce a vedere*** **YPR119W** ***nel grafico? Tracciare le comunità in questa componente connessa: a quali comunità appartiene*** **YPR119W*****? A quali complessi proteici potrebbe appartenere la proteina*** **YPR119W*****? Si riesce a trovare qualcosa sulla natura delle interazioni tra*** **YPR119W** ***e le proteine con cui interagisce?***
Suggerimento: Cercare YPR119W e le proteine con cui interagisce nel database del genoma *Saccharomyces cerevisiae* (https://www.yeastgenome.org). Può essere utile anche guardare la Figura 3 nell'articolo di de Lichtenberg *et al*. Si possono identificare i complessi a cui appartiene YPR119W dalla Figura 1 dell'articolo.

Utilizziamo la funzione `adj()` per scoprire con quali proteine interagisce la proteina YPR119W:

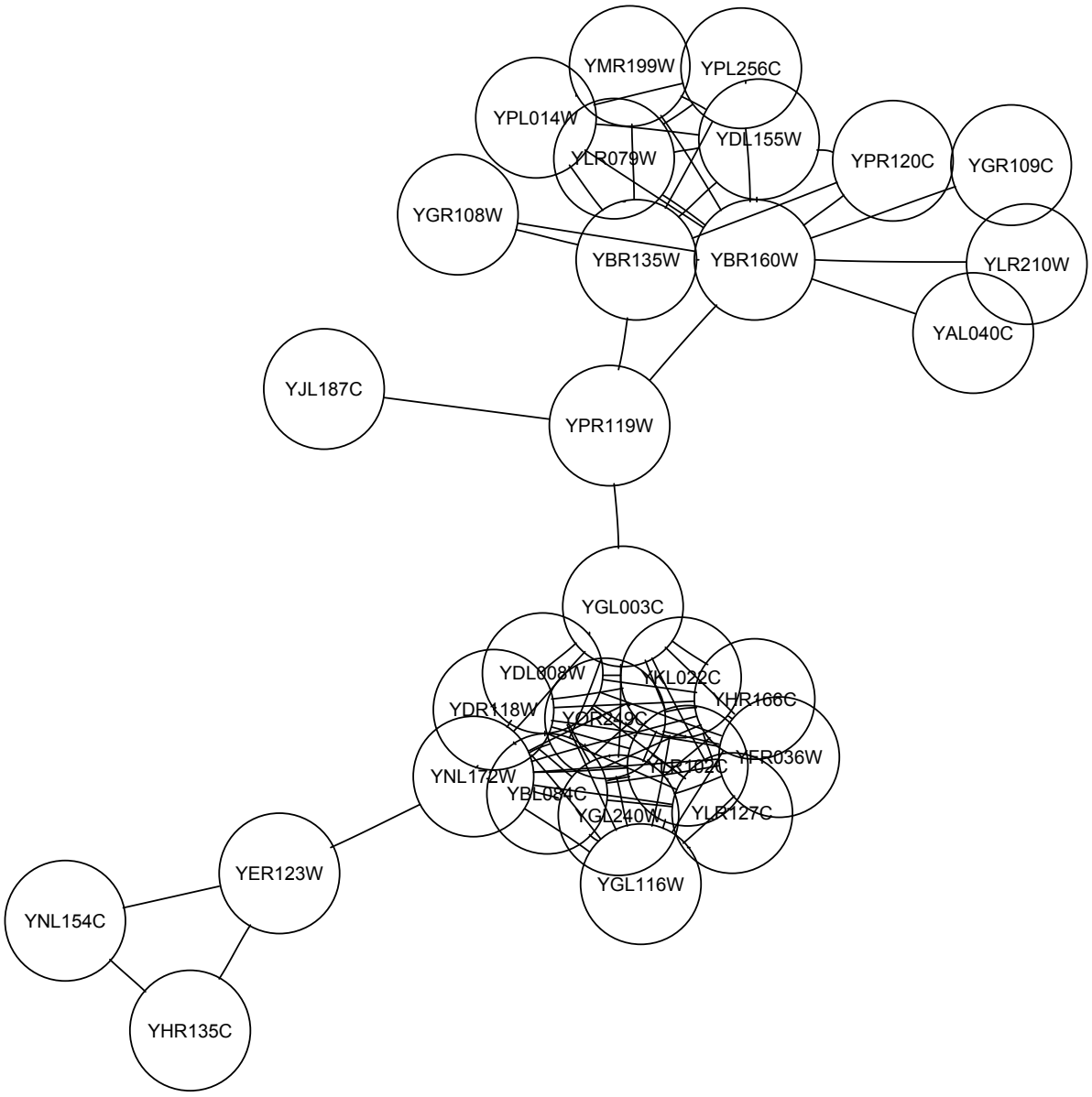
```
> adj(thegraph, "YPR119W")
$YPR119W
[1] "YGL003C" "YJL187C" "YBR135W" "YBR160W"
```

Quindi isoliamo la componente connessa (la n.7, costituita di 30 nodi e 114 connessioni) cui queste proteine appartengono:

```
> for (i in 1:length(myconnectedcomponents)) {
+ myindex <- myconnectedcomponents[[i]]=="YPR119W"
+ tmpcc <- myconnectedcomponents[[i]][myindex]
+ if (length(tmpcc)==1) { print(i) }
+ }
[1] 7
```

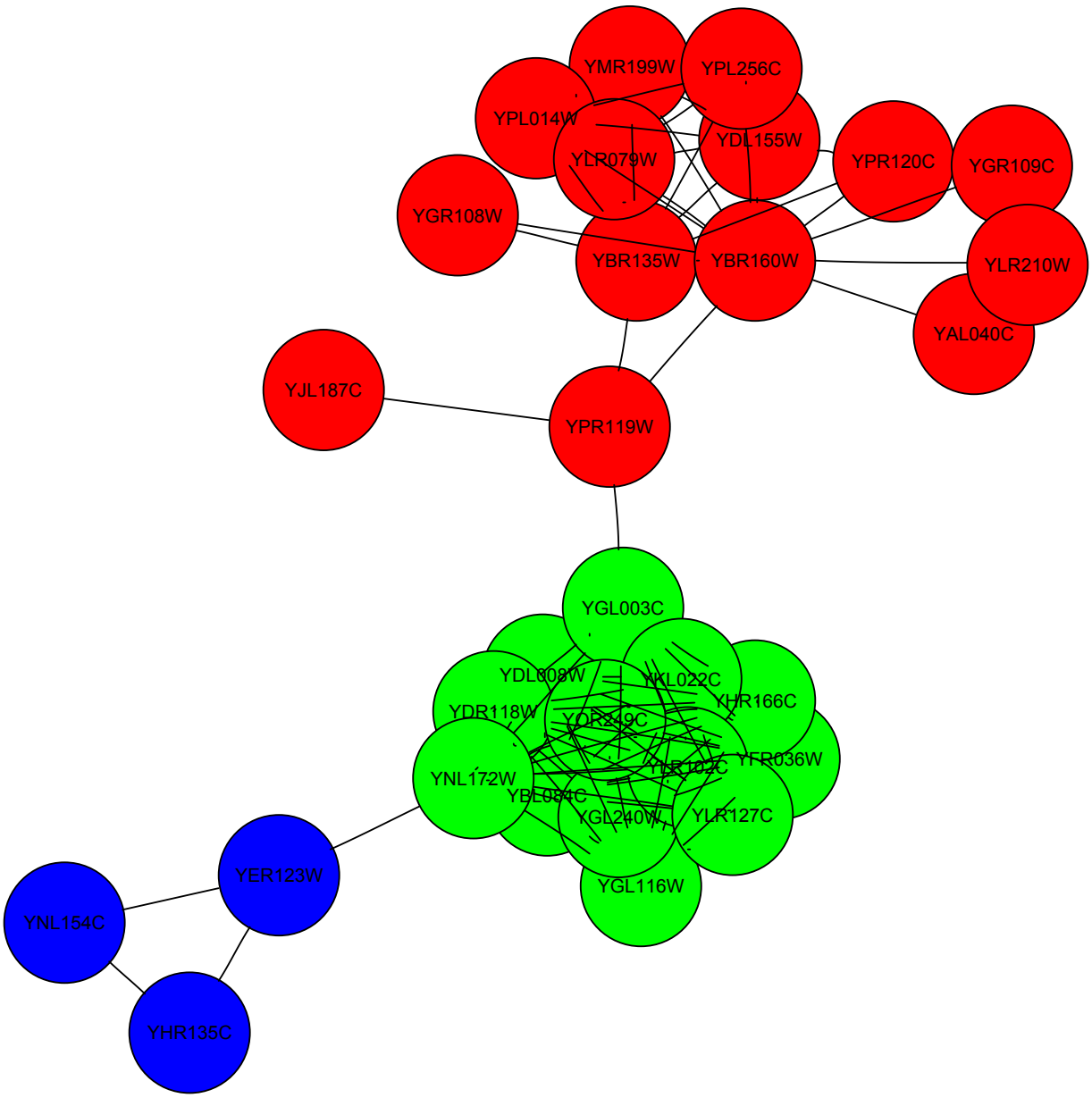
e ne visualizziamo il grafico:

```
> mysubgraph <- subGraph(myconnectedcomponents[[7]], thegraph)
> mysubgraph
A graphNEL graph with undirected edges
Number of Nodes = 30
Number of Edges = 114
> mygraphplot <- layoutGraph(mysubgraph, layoutType="neato")
> renderGraph(mygraphplot)
```



Tracciamo infine il grafico delle comunità della componente connessa mediante la funzione `plotcommunities()` (vedi Appendice testo di riferimento):

```
> findcommunities(mysubgraph, 1)
 [1] "YBL084C" "YBR135W" "YBR160W" "YDL008W" "YDL155W" "YDR118W" "YER123W" "YFR036W"
"YGL003C" "YGL116W"
[11] "YGL240W" "YGR108W" "YHR135C" "YHR166C" "YKL022C" "YLR102C" "YLR127C" "YMR199W"
"YNL154C" "YNL172W"
[21] "YOR249C" "YPL014W" "YPL256C" "YPR119W" "YPR120C" "YAL040C" "YGR109C" "YJL187C"
"YLR079W" "YLR210W"
[1] "Community 1 : YBL084C YDL008W YDR118W YFR036W YGL003C YGL116W YGL240W YHR166C
YKL022C YLR102C YLR127C YNL172W YOR249C"
[1] "Community 2 : YER123W YHR135C YNL154C"
[1] "Community 3 : YBR135W YBR160W YDL155W YGR108W YMR199W YPL014W YPL256C YPR119W
YPR120C YAL040C YGR109C YJL187C YLR079W YLR210W"
[1] "There were 3 communities in the input graph"
> plotcommunities(mysubgraph)
```



La proteina YPR119W appartiene alla comunità più numerosa (in rosso). Per quanto riguarda l'ultima domanda, seguire il suggerimento ed esaminare l'articolo di de Lichtenberg *et al*.

10. Estrazione delle caratteristiche strutturali delle proteine

---

Q10.1 ***Reperire l'accession number della proteina*** **Myoglobin** ***(mioglobina umana) e dire quanti atomi compongono la proteina; cercare inoltre mediante BLAST le eventuali proteine simili.***

Nota: Possiamo cercare su uno qualunque dei siti web contenenti banche dati proteiche, ad es. su Protein Data Bank.

Cercando sul sito web https://www.rcsb.org la parola chiave "Myoglobin" e selezionando la specie "Homo sapiens" troviamo la pagina risultato https://www.rcsb.org/structure/3RGK corrispondente all'accession PDB "3RGK".

Attiviamo quindi la libreria *bio3d* e leggiamo il file PDB della proteina 3RGK, che ci dice anche di quanti atomi (1.159) è costituita la proteina:

```
> library(bio3d)
> pdb <- read.pdb("3RGK")
> pdb

 Call:  read.pdb(file = "3RGK")

   Total Models#: 1
     Total Atoms#: 1317,  XYZs#: 3951  Chains#: 1  (values: A)

     Protein Atoms#: 1159  (residues/Calpha atoms#: 149)
     Nucleic acid Atoms#: 0  (residues/phosphate atoms#: 0)

     Non-protein/nucleic Atoms#: 158  (residues: 104)
     Non-protein/nucleic resid values: [ HEM (1), HOH (100), SO4 (3) ]

   Protein sequence:
      GLSDGEWQLVLNVWGKVEADIPGHGQEVLIRLFKGHPETLEKFDRFKHLKSEDEMKASED
      LKKHGATVLTALGGILKKKGHHEAEIKPLAQSHATKHKIPVKYLEFISEAIIQVLQSKHP
      GDFGADAQGAMNKALELFRKDMASNYKEL

+ attr: atom, xyz, seqres, helix, calpha,
        remark, call
```

Estraiamo la sequenza degli aminoacidi della proteina dall'oggetto *pdb*:

```
> pseq <- aa321(pdb$seqres)
> pseq
  [1] "G" "L" "S" "D" "G" "E" "W" "Q" "L" "V" "L" "N" "V" "W" "G" "K" "V" "E" "A" "D" "I" "P" "G" "H" "G"
 [26] "Q" "E" "V" "L" "I" "R" "L" "F" "K" "G" "H" "P" "E" "T" "L" "E" "K" "F" "D" "R" "F" "K" "H" "L" "K"
 [51] "S" "E" "D" "E" "M" "K" "A" "S" "E" "D" "L" "K" "K" "H" "G" "A" "T" "V" "L" "T" "A" "L" "G" "G" "I"
 [76] "L" "K" "K" "K" "G" "H" "H" "E" "A" "E" "I" "K" "P" "L" "A" "Q" "S" "H" "A" "T" "K" "H" "K" "I" "P"
[101] "V" "K" "Y" "L" "E" "F" "I" "S" "E" "A" "I" "I" "Q" "V" "L" "Q" "S" "K" "H" "P" "G" "D" "F" "G" "A"
[126] "D" "A" "Q" "G" "A" "M" "N" "K" "A" "L" "E" "L" "F" "R" "K" "D" "M" "A" "S" "N" "Y" "K" "E" "L" "G"
[151] "F" "Q" "G"
```

Eseguiamo la ricerca BLAST con la funzione blast.pdb() (ci potrebbe volere un po' di tempo):

```
> pblast <- blast.pdb(pseq)
 Searching ... please wait (updates every 5 seconds) RID = 7CJ1W3YZ016
 .
 Reporting 100 hits
```

Esaminiamo infine la lista delle proteine simili trovate, con le informazioni di allineamento:

```
> pblast$hit.tbl[,1:8]
        queryid subjectids identity alignmentlength mismatches gapopens q.start q.end
1   Query_22247     3RGK_A  100.000             153          0         0       1   153
2   Query_22247     1MYH_A   94.118             153          9         0       1   153
3   Query_22247     1MWC_A   93.464             153         10         0       1   153
4   Query_22247     1MYI_A   93.464             153         10         0       1   153
5   Query_22247     1MYJ_A   92.810             153         11         0       1   153
6   Query_22247     1M6C_A   92.810             153         11         0       1   153
7   Query_22247     1MNJ_A   92.157             153         12         0       1   153
8   Query_22247     1MNK_A   92.157             153         12         0       1   153
9   Query_22247     5YCG_A   90.850             153         14         0       1   153
10  Query_22247     1MNH_A   92.157             153         12         0       1   153
11  Query_22247     1MNI_A   92.157             153         12         0       1   153
12  Query_22247     5YCI_A   88.889             153         17         0       1   153
13  Query_22247     5YCH_A   88.158             152         18         0       2   153
14  Query_22247     5YL3_A   88.235             153         18         0       1   153
15  Query_22247     1AZI_A   88.235             153         18         0       1   153
16  Query_22247     1NZ2_A   88.235             153         18         0       1   153
17  Query_22247     1YMA_A   87.582             153         19         0       1   153
18  Query_22247     1HRM_A   87.582             153         19         0       1   153
19  Query_22247     4DC7_A   88.158             152         18         0       1   152
20  Query_22247     1RSE_A   87.582             153         19         0       1   153
21  Query_22247     1UFP_A   86.184             152         21         0       2   153
22  Query_22247     104M_A   86.184             152         21         0       2   153
23  Query_22247     1NZ3_A   87.582             153         19         0       1   153
24  Query_22247     1XCH_A   87.582             153         19         0       1   153
25  Query_22247     5B85_A   85.526             152         22         0       2   153
26  Query_22247     4QAU_A   85.526             152         22         0       2   153
27  Query_22247     5Z7E_A   87.582             153         19         0       1   153
28  Query_22247     1BJE_A   87.582             153         19         0       1   153
29  Query_22247     5Z7F_A   86.928             153         20         0       1   153
30  Query_22247     109M_A   85.526             152         22         0       2   153
31  Query_22247     3WYO_B   86.928             153         20         0       1   153
32  Query_22247     5XKW_A   84.868             152         23         0       2   153
33  Query_22247     3HC9_A   87.582             153         19         0       1   153
34  Query_22247     4TWV_A   86.928             153         20         0       1   153
35  Query_22247     4H07_A   85.526             152         22         0       2   153
36  Query_22247     4OF9_A   85.526             152         22         0       2   153
37  Query_22247     1CIO_A   84.868             152         23         0       2   153
38  Query_22247     3RJ6_A   86.928             153         20         0       1   153
39  Query_22247     3A2G_A   85.526             152         22         0       2   153
40  Query_22247     5YZF_A   85.526             152         22         0       2   153
41  Query_22247     5B84_A   85.526             152         22         0       2   153
42  Query_22247     4OOD_A   85.526             152         22         0       2   153
43  Query_22247     1MLM_A   84.868             152         23         0       2   153
44  Query_22247     3O89_A   85.526             152         22         0       2   153
45  Query_22247     1J3F_A   85.526             152         22         0       2   153
46  Query_22247     3HEN_A   87.582             153         19         0       1   153
47  Query_22247     2SPO_A   84.868             152         23         0       2   153
48  Query_22247     1CO9_A   84.868             152         23         0       2   153
49  Query_22247     5XKV_A   85.526             152         22         0       2   153
50  Query_22247     1MLQ_A   84.868             152         23         0       2   153
51  Query_22247     3SDN_A   85.526             152         22         0       2   153
52  Query_22247     1CP5_A   84.868             152         23         0       2   153
53  Query_22247     1CH2_A   84.868             152         23         0       2   153
54  Query_22247     1JDO_A   84.868             152         23         0       2   153
55  Query_22247     4PQ6_A   85.526             152         22         0       2   153
56  Query_22247     4IT8_A   85.526             152         22         0       2   153
…
```

11. Analisi di dati da microarray

Q11.1 ***Eseguire la normalizzazione RMA (Robust Multichip Average) sul dataset*** **GSE1297** ***della banca dati Gene Expression Omnibus (GEO).***

Nota: Per eseguire quanto richiesto occorre installare preliminarmente i seguenti package dal repository Bioconductor: GEOQuery, affy, hgu133a.db, hgu133acdf.

Installiamo i package richiesti dal repository Bioconductor:

```
> BiocManager::install("GEOquery") # Get data from NCBI Gene Expression Omnibus (GEO)
> BiocManager::install("affy") # Methods for Affymetrix Oligonucleotide Arrays
> BiocManager::install("hgu133a.db", type = "source") # GSE1297: Platform_title = [HG-U133A]
> BiocManager::install("hgu133acdf")
```

Scarichiamo i file CEL dal dataset remoto nella directory di lavoro e scompattiamoli nella sottodirectory GSE1297, elencandoli:

```
> library("GEOquery")
> getGEOSuppFiles("GSE1297")
> setwd(paste(getwd(),"/GSE1297",sep=""))
> untar("GSE1297_RAW.tar", exdir = "data")
> cels = list.files("data/", pattern = "cel")
> sapply(paste("data", cels, sep = "/"), gunzip)
> cels = list.files("data/", pattern = "cel")
> cels
 [1] "GSM21203.cel" "GSM21204.cel" "GSM21205.cel" "GSM21206.cel" "GSM21207.cel" "GSM21208.cel"
 [7] "GSM21209.cel" "GSM21210.cel" "GSM21211.cel" "GSM21212.cel" "GSM21213.cel" "GSM21214.cel"
[13] "GSM21215.cel" "GSM21216.cel" "GSM21217.cel" "GSM21218.cel" "GSM21219.cel" "GSM21220.cel"
[19] "GSM21221.cel" "GSM21222.cel" "GSM21223.cel" "GSM21224.cel" "GSM21225.cel" "GSM21226.cel"
[25] "GSM21227.cel" "GSM21228.cel" "GSM21229.cel" "GSM21230.cel" "GSM21231.cel" "GSM21232.cel"
[31] "GSM21233.cel"
```

Eseguiamo quindi la normalizzazione RMA ($\log_2$) nella sottodirectory "data" ed esaminiamo i dati normalizzati per le prime cinque sonde e i primi cinque campioni:

```
> library(affy)
> library(hgu133a.db)
> library(hgu133acdf)
> setwd(paste(getwd(),"/data",sep=""))
> raw.data = ReadAffy(verbose = FALSE, filenames = cels, cdfname = "hgu133acdf")
> data.rma.norm = rma(raw.data)
Background correcting
Normalizing
Calculating Expression
> rma = exprs(data.rma.norm)
> rma[1:5, 1:5]
          GSM21203.cel GSM21204.cel GSM21205.cel GSM21206.cel GSM21207.cel
1007_s_at    10.606195    10.338992    10.390181    10.523067    10.790810
1053_at       4.586310     4.462708     4.571781     4.530547     4.545586
117_at        5.936847     6.016915     6.234214    10.113877     6.253083
121_at        8.762880     8.951345     9.026180     8.803613     9.130337
1255_g_at     4.361086     4.439068     4.597530     4.019156     4.353772
```

Registriamo infine i dati normalizzati in un file:

```
> write.table(rma, file = "rma.txt", quote = FALSE, sep = "\t")
```

Aggiungiamo le sonde come prima colonna dei dati elaborati:

```
> tt = cbind(row.names(rma), rma)
> colnames(tt) = c("ProbID", sub(".cel", "", colnames(rma), ignore.case = TRUE))
> rownames(tt) = NULL
> tt[1:5, 1:5]
     ProbID      GSM21203           GSM21204           GSM21205           GSM21206
[1,] "1007_s_at" "10.6061947450577" "10.3389916272064" "10.390181163724"  "10.5230672101482"
[2,] "1053_at"   "4.58630979543013" "4.46270777736086" "4.57178078332995" "4.53054651020166"
[3,] "117_at"    "5.93684658044002" "6.01691505298677" "6.23421446338666" "10.1138768429987"
[4,] "121_at"    "8.76288033305696" "8.9513454849938"  "9.02618022186855" "8.80361343714862"
[5,] "1255_g_at" "4.3610859654913"  "4.43906808970735" "4.59752976632791" "4.01915640899688"
```

Scarichiamo nella directory di lavoro il file completo delle annotazioni (formato CSV separato da virgola) dal sito Affymetrix (indirizzo http://www.affymetrix.com/Auth/analysis/downloads/na36/ivt/HG-U133A.na36.annot.csv.zip); eliminiamo tutte le colonne tranne Probe.Set.ID e Entrez.Gene e leggiamo il file nel dataframe *annot*:

```
> annot <- read.table("HG-U133A.na35.annot.csv", header = TRUE, sep = “,")
> head(annot)
  Probe.Set.ID Entrez.Gene
1    1007_s_at         780
2      1053_at        5982
3       117_at        3310
4       121_at        7849
5    1255_g_at        2978
6      1294_at        7318
```

Combiniamo quindi i dati e le annotazioni nel dataframe *comb* e salviamo il file annotato "comb2.txt":

```
> comb = merge(annot, tt, by.x = “Probe.Set.ID", by.y = "ProbID")
> comb[1:5, 1:5]
  Probe.Set.ID Entrez.Gene         GSM21203         GSM21204         GSM21205
1    1007_s_at         780 10.6061947450577 10.3389916272064  10.390181163724
2      1053_at        5982 4.58630979543013 4.46270777736086 4.57178078332995
3       117_at        3310 5.93684658044002 6.01691505298677 6.23421446338666
4       121_at        7849 8.76288033305696  8.9513454849938 9.02618022186855
5    1255_g_at        2978  4.3610859654913 4.43906808970735 4.59752976632791
> write.table(comb, file = "comb2.txt", quote = FALSE, sep = "\t", row.names = FALSE)
```

Eseguiamo la media dei valori (con cinque cifre decimali) per le sonde ripetute:

```
> comb2 <- subset(comb, select = -c(Probe.Set.ID))
> comb2 <- data.frame(lapply(comb2, as.character), stringsAsFactors = FALSE)
> comb2 <- data.frame(lapply(comb2, as.numeric), stringsAsFactors = FALSE)
> out <- aggregate(. ~ Entrez.Gene, data = comb2, mean)
> out = format(out, digits = 5)
> out[1:5, 1:5]
  Entrez.Gene GSM21203 GSM21204 GSM21205 GSM21206
1           2   9.0928   9.7931   9.0962   9.8969
2           9   4.5077   4.3635   4.5445   4.3781
3          10   6.2881   6.2111   7.0938   5.7314
4          12  11.5590   8.0948   8.2142  12.3144
5          13   4.5742   4.6783   5.0740   4.5521
```

Salviamo infine il risultato finale nel file di testo "GSE1297.RMA.txt" in formato tab-separated:

```
> write.table(out, file = "GSE1297.RMA.txt", quote = FALSE, sep = "\t",
      row.names = FALSE)
```

Appendice

Funzione R `generateSeqWithMultinomialModel()`

```
generateSeqWithMultinomialModel <- function(n, probabilities) {

# Define the letters in the alphabet
letters <- c("A", "C", "G", "T")

# Make a random sequence of length n letters, using the multinomial model with probabilities "probabilities"
seq <- sample(letters, n, rep=TRUE, prob=probabilities) # sample with replacement

# Return the sequence
return(seq)
}
```

Funzione R `makeDotPlot1()`

```
makeDotPlot1 <- function(seq1,seq2,dotsize=1) {

length1 <- length(seq1)
length2 <- length(seq2)

# Make a plot:
x <- 1
y <- 1
plot(x, y, ylim=c(1,length2), xlim=c(1,length1), col="white") # make an empty plot

# Now plot dots at every position where the two sequences have the same letter:
for (i in 1:length1) {
      letter1 <- seq1[i]
      for (j in 1:length2) {
            letter2 <- seq2[j]
            if (letter1 == letter2) {
                  # add a point to the plot
                  points(x=i, y=j, cex=dotsize, col="blue", pch=7)
            }
      }
}

}
```

Funzione R `multinomialprob()`

```
multinomialprob <- function(mysequence, probabilities) {

nucleotides <- c("A", "C", "G", "T") # Define the alphabet of nucleotides
names(probabilities) <- nucleotides
mysequence <- toupper(mysequence) # Convert the sequence to uppercase letters
seqlength <- length(mysequence)
seqprob <- numeric() # Make a variable to hold to probability of the whole sequence

for (i in 1:seqlength) { # For each letter in the input sequence
      nucleotide <- mysequence[i] # Find the i-th nucleotide in the sequence
      # Calculate the probability of the i-th nucleotide in the sequence
      nucleotideprob <- probabilities[nucleotide]
      # The probability of the whole sequence is calculated by multiplying together
      # the probabilities of the nucleotides at each sequence position
      if (i == 1) { seqprob <- nucleotideprob[[1]] }
      else { seqprob <- seqprob * nucleotideprob[[1]] }
}

# Return the value of the probability of the whole sequence
return(seqprob)
}
```

Funzione R `unrootedMEtree()`

```
##
# Function to calculate an unrooted phylogenetic tree with bootstraps, using the minimum evolution method.
##

unrootedMEtree <- function(alignment,type) {

# Load the ape and seqinR packages:
require("ape")
require("seqinr")

# Define a function for making a tree:
makemytree <- function(alignmentmat) {
      alignment <- ape::as.alignment(alignmentmat)
      if (type == "protein") {
            mydist <- dist.alignment(alignment)
    }
      else if (type == "DNA") {
      alignmentbin <- as.DNAbin(alignment)
        mydist <- dist.dna(alignmentbin)
    }
      mytree <- fastme.bal(mydist)
      mytree <- makeLabel(mytree, space="") # get rid of spaces in tip names
      return(mytree)
}

# Infer a tree:
mymat <- as.matrix.alignment(alignment)
mytree <- makemytree(mymat)

# Bootstrap the tree;
myboot <- boot.phylo(mytree, mymat, makemytree)

# Plot the tree:
plot.phylo(mytree,type="u")
nodelabels(myboot,cex=0.7)
mytree$node.label <- myboot

return(mytree)
}
```