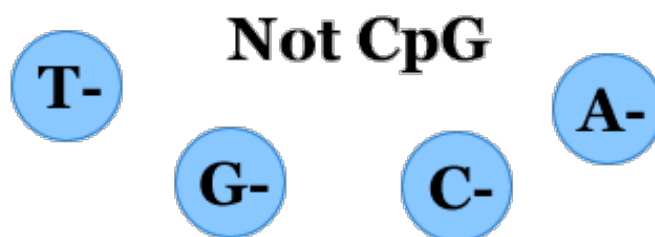
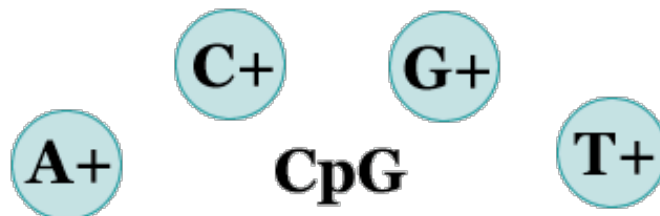


Programmation en R pour trouver des îlots CpG

Guohua XU (Augix)
M1 bioinformatique • UCBL Université Lyon1 • 17 Mai 2005



| | | | | | | | | | | | |
|-----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|-----------|-----------|
| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| O | A | C | A | G | C | T | A | G | C | T | A |
| O' | OA | AC | CA | AG | GC | CT | TA | AG | GC | CT | TA |
| H | - | - | - | - | - | - | + | + | + | + | + |

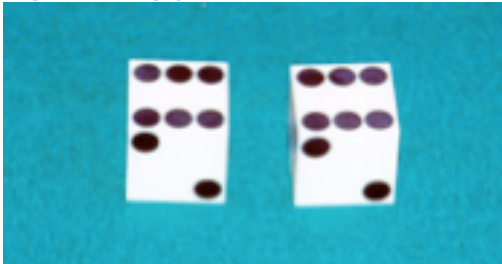
O: Observations O': Formated O H: Hidden Status

$$V_k(i) = \max_{\{H_1, \dots, H_{i-1}\}} P[o_1 \dots o_{i-1}, H_1, \dots, H_{i-1}, o_i, H_i = k]$$

1. Commence par un exemple

1.1 Le Casino malhonnête

Il y a 2 types de dés dans un casino:



- 'Fair die': le dé normal

P: Probabilité

$$P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = 1/6$$

- 'Loaded die': le dé truqué (la face du numéro 6 est plus lourde que les autres)

$$P(1) = P(2) = P(3) = P(4) = P(5) = 1/10$$

$$P(6) = 1/2$$

Par exemple:

Si on a observé une sequence de lancers de dés comme cela:

O = 14236534561216234566263646566136364656

Question:

Quand est-ce que le joueur du casino a-t-il utilisé le dé truqué?

On peut deviner comme ça:

O = 14236534561216234566263646566136364656

H = FFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLLLLL

F: Faire die L: Loaded die

1.2 La meilleure hypothèse

O = 14236534561216234566263646566136364656
 Longueur de O est 38

c'est une séquence qu'on a observé, il existe une séquence cachée qui correspond à cette séquence observée.

La séquence cachée peut être:

H = FF
 H = LFFF
 ...
 H = FFFL
 ...
 H = LL

Il y a 2^{38} possibilités.

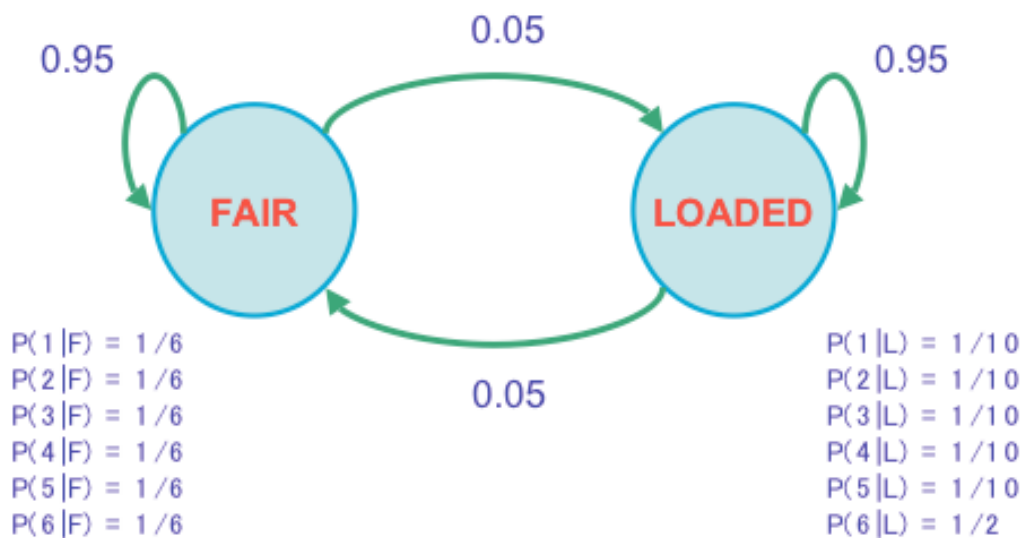
Question:

Quelle séquence est la meilleure séquence cachée?

Ici, on ajoute une condition:

soit, le joueur casino change une fois le dé quand il a fini 20 fois de jouer.

Donc, on a un modèle comme cela:



En fait, c'est un modèle de chaînes de Markov à états cachés.

- Les états observés : $b = \{ 1,2,3,4,5,6 \}$
- La séquence observée : $O = 1423653456\dots$
- Les états cachés : $a = \{ F,L \}$
- La séquence cachée : $H = ???????????...$
- Probabilités de Transitions entre les états cachés :
 $P(H_{i+1} | H_i) : T_{FF}=0.95 \ T_{FL}=0.05 \ T_{LL}=0.95 \ T_{LF}=0.05$
- Probabilités d'Émissions entre les états observés :
 $P(1 | F) = 1/6 \dots P(6 | F) = 1/6$
 $P(1 | L) = 1/10 \dots P(6 | L) = 1/2$
- Probabilités Initiales :
 $P(H_1 = a_i) = 1/2$

D'après chaque séquence cachée possible, on peut calculer la probabilité pour observer la séquence O:

$O = 14236534561216234566263646566136364656$

$H' = \text{FF}$

$$P = (1/2) \times T_{FF}^{37} \times (1/6)^{38} = 2.018347e-31$$

$H'' = \text{LFF}$

$$P = (1/2) \times T_{LF} \times T_{FF}^{36} \times (1/10) \times (1/6)^{37} = 6.373728e-33$$

$H''' = \text{FFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLLLL}$

$$P = (1/2) \times T_{FF}^{17} \times T_{FL} \times T_{LL}^{19} \times (1/6)^{18} \times (1/10)^9 \times (1/2)^{11} = 1.896432e-29$$

Donc, H''' est plus probable que H' et H'' .

Mais, est-ce que H''' est la meilleure ?

Peut-on calculer tous les H possibles?

Non, car il y a 2^{38} possibilités pour H.

En fait, on peut utiliser l'algorithme de Viterbi et la programmation dynamique en R.

2. Algorithme de Viterbi

Un programme **Casino.R** est écrit pour trouver la séquence cachée pour une séquence observée. Ce programme utilise l'Algorithme de Viterbi et la Programmation Dynamique.

Par exemple:

D'abord, on a une séquence observée O.

```
O='14236534561216234566263646566136364656...'
```

Pour chaque état observé (par exemple: $O_4='3'$), on veut trouver un état caché correspondant, soit 'F', soit 'L'. ('F' représente 'Faire die', 'L' représente 'Loaded Die'.)

Donc, il y aura une séquence cachée pour O. Par exemple:

```
H='FFFLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL...'
```

On définit $V_k(i)$ comme la probabilité de la séquence cachée la plus probable si l'on fixe le i ème état caché à valeur k, $H_i=k$.

$$V_k(i) = \max_{\{H_1, \dots, H_{i-1}\}} P[O_1 \dots O_{i-1}, H_1, \dots, H_{i-1}, O_i, H_i=k]$$





$\{H_1, \dots, H_{i-1}\}$ est les valeur on se donne pour la séquence caché.

Par définition, on peut trouver $V_l(i+1)$ qui correspond à la condition: le i ème état caché est valeur l, $H_{i+1}=l$.

$$V_l(i+1) = \max_{\{H_1, \dots, H_{i-1}\}} P(O_{i+1}, H_{i+1}=l \mid H_i=k) P[O_1 \dots O_{i-1}, H_1, \dots, H_{i-1}, O_i, H_i=K]$$

$$V_l(i+1) = \max_k e_l(O_{i+1}) a_{kl} V_k(i)$$

max, cela veut dire l'on cherche la variable pour maximiser l'expression suivante. $e_l(O_{i+1})$ est la probabilité d'émission, c'est-à-dire la probabilité pour observer O_{i+1} d'après la condition $H_{i+1}=l$. a_{kl} est la probabilité de transition, $P[H_{i+1}=l \mid H_i=k]$.

| | O ₁ | O ₂ | ... | O _i | O _{i+1} | ... |
|-----|----------------|----------------|-----|---|---------------------|-----|
| 1 | | | |  | | |
| 2 | | | |  | | |
| ... | | | | | | |
| l | | | |  | V _{l(i+1)} | |
| ... | | | | | | |
| k | | | |  | V _{k(i)} | |
| ... | | | | | | |

On recherche le meilleur k pour l qui maximise V_{l(i+1)}.
Ici, par exemple, on trouve 2 pour l.

Un bloc de code en R est écrit pour faire cela:

```
V[1, 1] = log(1/12)      #####Probabilités Initiales
V[1, 2] = log(1/12)      #####Probabilités Initiales
Maxk = c(0, 0)
for (i in 2:len) {      #####Calculer de 2 à la longueur de séquence observée
  j = i - 1
  for (l in c(1, 2)) {  #####Fixer Hj
    for (k in c(1, 2)) { #####Fixer Hi
      tmp1 = paste(k, l, sep = "")
      Maxk[k] = V[j, k] + log(M1[l, as.numeric(ss[i])]) + log(T1[1, tmp1])
    }

    if ((Maxk[1]) > (Maxk[2])) { #####cherche le plus grand k
      V[i, l] = Maxk[1]
      P[i, l] = 1
    }
    else {
      V[i, l] = Maxk[2]
      P[i, l] = 2
    }
  }
}
}
```

Finalement, on va obtenir le tableau ci-dessous:

| | O ₁ | O ₂ | ... | O _i | O _{i+1} | ... |
|-----|----------------|----------------|-----|----------------|------------------|-----|
| 1 | ... | 3 | ... | ... | ... | ... |
| 2 | ... | ... | 6 | ... | 1 | ... |
| ... | 2 | ... | ... | ... | ... | ... |
| l | ... | ... | ... | ... | 2 | ... |
| ... | ... | 5 | ... | ... | ... | ... |
| k | ... | ... | 4 | ... | ... | ... |
| ... | ... | ... | ... | ... | ... | ... |

À la fin, on tire le plus grand $V(n)$, n est la longueur de la séquence observée O . Et puis, on remonte le chemin qui est bien la séquence cachée.

Un bloc de code en R est écrit pour faire cela:

```
if (V[ $len$ , 1] > V[ $len$ , 2]) R[ $len$ ] = 1 else R[ $len$ ] = 2
for (i in ( $len$  - 1):1) {
  #remonter le chemin
  j = i + 1
  R[i] = P[j, R[j]]
}
```

Pour utiliser le programme Casino.R, le plus simple c'est:

Télécharger le package [CpG_1.0.tar.gz](#)

Installer le package, dans le terminal (par exemple Bash):

```
$ R CMD INSTALL CpG_1.0.tar.gz
```

Lancer R

```
$ R
```

Dans R: (charger le 'library CpG')

```
> library(CpG)
```

Lancer la fonction Casino

```
> Casino()
```

3. Trouver des îlots CpG

3.1 Les îlots CpG

Dans le génome humain, la méthylation agit sur les dinucléotides CG, en méthylant la cytosine qui par la suite sera mutée en thymine. Pour cette raison, le taux des dinucléotides CpG est anormalement bas (On met le “p” pour différencier ce dinucléotide de la paire C-G entre les deux brins de l’ADN). Néanmoins, il existe des régions dans les quelles ce processus n’agit pas, et où le taux de CpG est plus élevé qu’alentours, régions nommées “îlots CpG”. Ces régions ont des longueurs de quelques centaines à quelques milliers de bases.

3.2 Proba.R

Le but 1 est de trouver un modèle (les probabilités émissions) qui peut discriminer des morceaux de séquence, selon qu’ils possèdent ou non un îlot CpG. L’idée est la suivante : à partir d’un jeu de séquences (dans le fichier [mus_tem_app.fa](#)) qui ne contiennent pas d’îlot CpG, on construit un processus de Markov, noté P-. À partir d’un autre jeu de séquences (dans le fichier [mus_cpg_app.fa](#)) qui ne contient que des îlot CpG, on calcule un autre modèle Markov, noté P+.

Un programme [Proba.R](#) est écrit pour faire cela.

Après avoir installé le package CpG et chargé la librairie CpG, on peut lancer la fonction [Proba](#).

```
> Proba(OutputFile='model.test')
```

Le résultat est sorti dans le fichier 'model.test'. Pour plus d'informations, tapez:

```
> help(Proba)
```

Le résultat dans le fichier 'model.test':

```
"proba_plus" "proba_moins" "V3" "V4"  
"AA" 0.283931134765473 0.290615758887686 0.25 0.25  
"AC" 0.216260668335329 0.198788457655444 0.25 0.25  
"AG" 0.294286627821019 0.282990663998871 0.25 0.25  
"AT" 0.205152325889693 0.227455242290023 0.25 0.25  
"CA" 0.281656967794952 0.349511525554683 0.25 0.25  
"CC" 0.274279830015769 0.263346702709620 0.25 0.25  
"CG" 0.158029647329547 0.0423133775621264 0.25 0.25  
"CT" 0.285998508636238 0.344477402514756 0.25 0.25  
"GA" 0.253217833900574 0.284090347972204 0.25 0.25  
"GC" 0.253594942272825 0.208112915084803 0.25 0.25  
"GG" 0.276745878447146 0.263147339859246 0.25 0.25  
"GT" 0.216135227597656 0.244240673660091 0.25 0.25  
"TA" 0.184366999046627 0.203852894478115 0.25 0.25  
"TC" 0.255584345576128 0.229434998542093 0.25 0.25  
"TG" 0.281323238660285 0.280375204353454 0.25 0.25  
"TT" 0.278372381084242 0.286077161645975 0.25 0.25
```

3.3 CpG.R

Le but 2 est de trouver les îlots CpG dans les Chromosomes (dans les fichiers **mus1.fa mus2.fa mus3.fa**). C'est comme trouver les dés truqués ('loaded die') dans l'exemple Casino.

- Les états observés : $b = \{ A, C, G, T \}$ $b' = \{ AA, AC, \dots, TT \}$
- La séquence observée : $O = ACACGTTACGATCG \dots$
 $O' = 0A AC CA AC CG \dots$
- Les états cachés : $a = \{ +, - \}$ $a' = \{ ++, +-, -+, -- \}$
+ représente l'îlot CpG - représente non-îlot CpG
- La séquence cachée : $H = ???????????...$

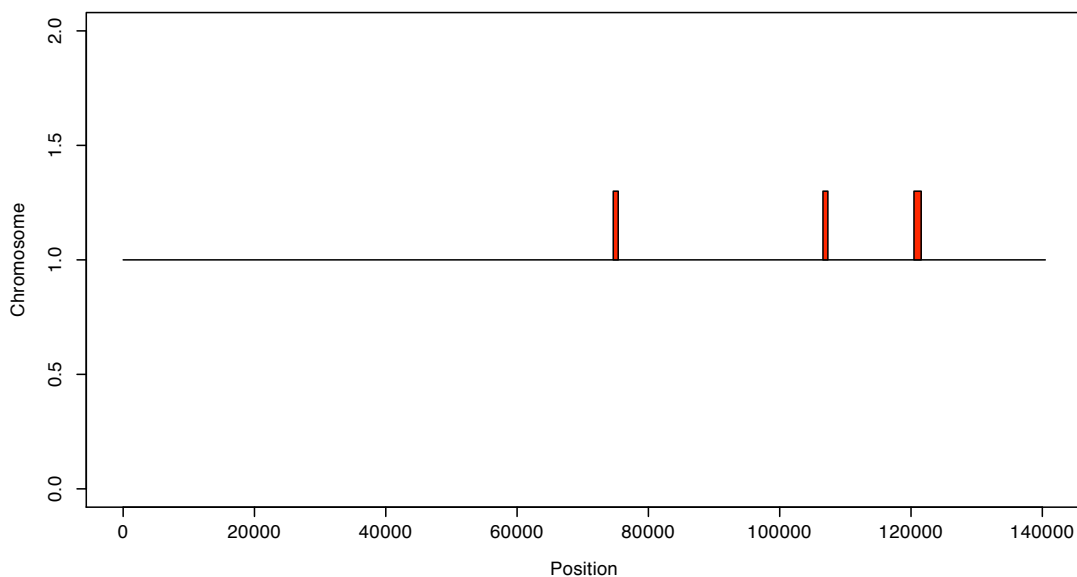
- Probabilités de Transitions entre les états cachés :
 $P(H_{i+1} | H_i) : T_{++} T_{+-} T_{--} T_{-+}$
- Probabilités d'Émissions entre les états observés :
 $P('AA' | H'_i=++) \dots P('CG' | H'_i=++) \dots P('TT' | H'_i=++)$
 \dots
 $P('AA' | H'_i=--) \dots P('CG' | H'_i=--) \dots P('TT' | H'_i=--)$
- Probabilités Initiales :
 $P(H_1 = a_i) = 1/2$

Un programme **CpG.R** est écrit pour faire cela.

Après avoir installé le package CpG et chargé la librairie CpG, on peut lancer la fonction **CpG**.

```
> CpG()
```

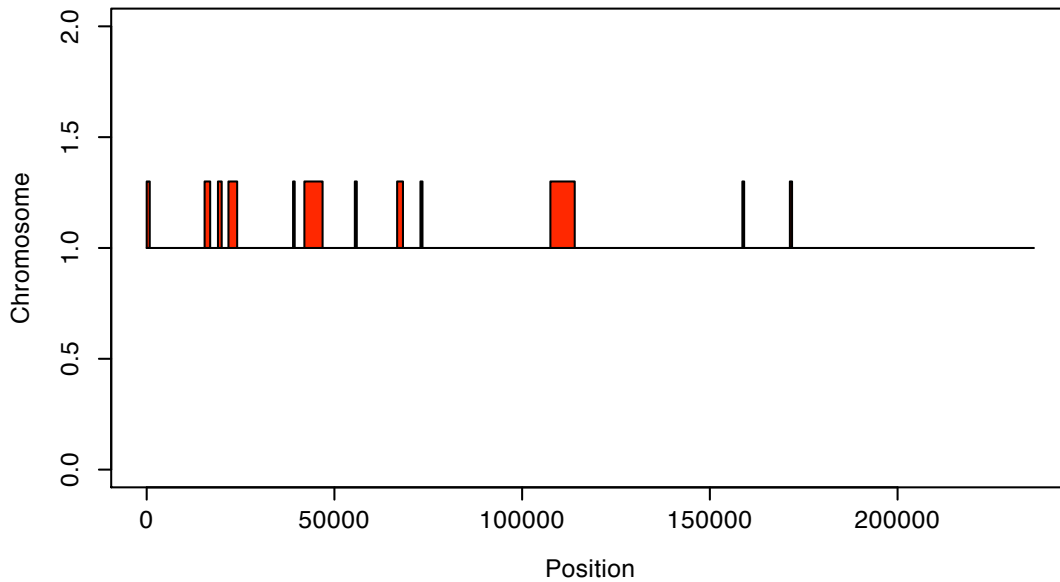
Le résultat présente les îlots CpG dans le Chromosome mus3.fa



Ici, on trouve 3 îlots CpG, on peut aussi voir ses positions sur le Chromosome.

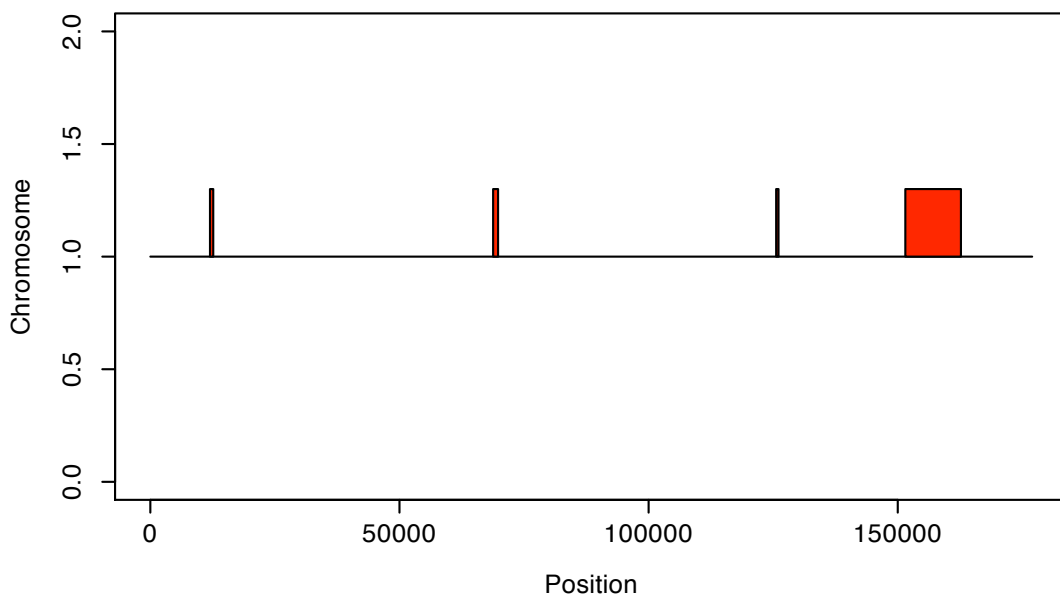
```
> CpG(File=system.file("mus1.fa",package="CpG"))
```

mus1.fa



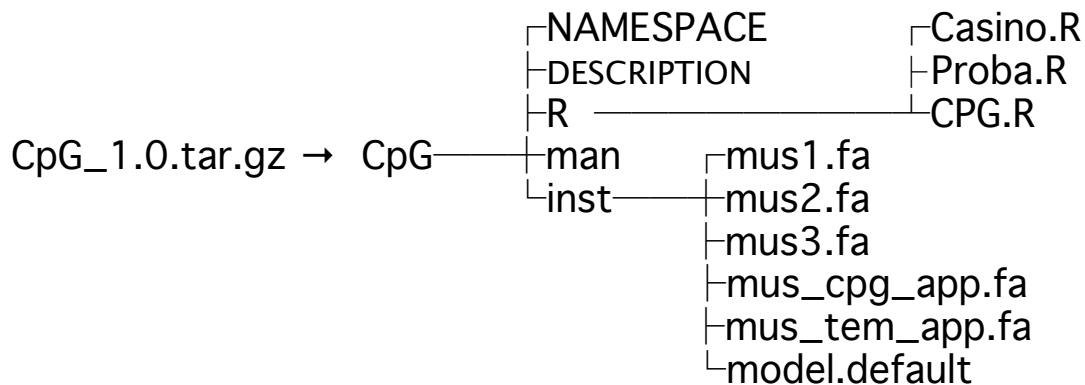
```
> CpG(File=system.file("mus2.fa",package="CpG"))
```

mus2.fa



4. Package CpG_1.0.tar.gz

4.1 Les fichiers



Pour utiliser CpG_1.0.tar.gz, le plus simple c'est:

Installer le package, dans le terminal (par exemple Bash):

```
$ R CMD INSTALL CpG_1.0.tar.gz
```

Les fichiers dans 'inst' sont aussi copiés automatiquement. L'expression `system.file("mus1.fa", package="CpG")` peut appeler le fichier mus1.fa qui a installé.

Lancer R

```
$ R
```

Dans R: (charger le 'library CpG')

```
> library(CpG)
```

Il y a 3 fonctions: **Casino**, **CpG**, **Proba**.

CpG package just for fun.



Documentation for package 'CpG' version 1.0

Help Pages

| | |
|------------------------|--|
| Casino | Find out the Loaded Die sequences |
| CpG | Find out the CpG islands in a Chromosome (or a large-scale Sequence) |
| Proba | Calculate the Probabilities |

```
> help(CpG)
```

CpG {CpG}

R Documentation

Find out the CpG islands in a Chromosome (or a large-scale Sequence)

Description

Find out the CpG islands in a Chromosome (or a large-scale Sequence)

Usage

```
CpG(File = system.file("mus3.fa",package="CpG"), Model =  
system.file("model.default", package="CpG") )
```

Arguments

File

The sequence file which formatted by FASTA.

Model

The Model file. Default Model file is `system.file("model.default",package="CpG")`

Author(s)

Guohua XU (Augix) augix.com@gmail.com

References

<http://www.augix.com/bioinfo/R/CpG.html>

See Also

[Casino](#), [Proba](#)

Examples

```
CpG(File = system.file("mus1.fa",package="CpG"), Model =  
system.file("model.default",package="CpG") )  
CpG(File = system.file("mus2.fa",package="CpG"), Model =  
system.file("model.default",package="CpG") )  
CpG(File = system.file("mus3.fa",package="CpG"), Model =  
system.file("model.default",package="CpG") )
```

[Package *CpG* version 1.0 [Index](#)]

4.1 L'Explication du code source de CpG.R

Le code source de **CpG.R**

```
CpG <- function( File =  
system.file("mus3.fa",package="CpG"),Model=system.file("  
model.default",package="CpG") ) {  
  start <- Sys.time()  
  print('Please wait for a minute...')  
  
  ### Charger Sequence  
  ### File représente le fichier chromosome mus3.fa  
  seq <- readLines(File)  
  ### On n'a pas besoin de la ligne commence par >  
  ind <- grep(">",seq)  
  ### acgtacgt -> ACGTACGT  
  sequence <- toupper(paste(seq[-ind],collapse=""))  
  ### 'ACGTACGT' -> 'A' 'C' 'G' 'T' 'A' 'C' 'G' 'T'  
  s <- unlist(strsplit(sequence,""))  
  len <- length(s)  
  ### s = 'A' 'C' 'G' 'T' 'A' 'C' 'G' 'T'  
  ### ss = '0A' 'AC' 'CG' 'GT' 'TA' 'AC' 'CG' 'GT'  
  ss <- paste(c(0,s)[1:len],s,sep="")
```

```

#### Charger Model
#### Model représente model.default
M1 <- read.table(Model,header=T)
#### Les Probabilités de Transitions
T1 <-
matrix(c(0.999,0.999992,0.001,0.000008),nrow=16,ncol
=4,byrow=TRUE)
MT <- log(M1)+log(T1)
colnames(MT) <- c('++','--','+-','-+')
v <- cbind(MT[ss,'++'],MT[ss,'--'],MT[ss,'+-'],MT[ss,'-+'])
colnames(v) <- c('++','--','+-','-+')
V <- cbind(1:len,0)
P <- cbind(1:len,0)
colnames(V) <- c('+','-')
colnames(P) <- c('+','-')
#### Les Probabilités Initiales
V[1,] <- c(log(1/8),log(1/8))

#### Calculer V et P
#### Utiliser max et which.max pour remplir 'for'
#### Mais il toujours coûte beaucoup de temps
for (i in 2:len) {
  j <- i-1
  #### Si Hi='+' quelle Hj= P[i,'+'] maximise V[i,'+']
  temp <- c(v[i,'++']+V[j,'+'],v[i,'-+']+V[j,'-'])
  V[i,'+'] <- max(temp)
  P[i,'+'] <- c('+','-')[which.max(temp)]

  #### Si Hi='-' quelle Hj= P[i,'-'] maximise V[i,'-']
  temp <- c(v[i,'+-']+V[j,'+'],v[i,'--']+V[j,'-'])
  V[i,'-'] <- max(temp)
  P[i,'-'] <- c('+','-')[which.max(temp)]
}

```

```

#### Remonter pour trouver la sequence cachée
HS <- c()
#### Dernière état caché de la séquence cachée
HS[len] <- c('+','-')[which.max(c(V[len,+'],V[len,-']))]
for (i in (len-1):1) {
#### Par exemple, P[i, '+'] représente Hi-1
  HS[i] <- P[i+1,HS[i+1]]
}
#### '-' '-' '+' '+' '+' -> '--++++'
HS2 <- paste(c('-',HS), c(HS,-'), sep='')
#### Un îlot commence par deb, finie par fin
deb <- which(HS2=='-+')
fin <- which(HS2=='+-')

#### Faire dessin
plot( c(0:2), xlim=c(0,len), type='n', xlab='Position',
ylab='Chromosome' )
segments(0,1,len,1)
rect(deb,1.0,fin,1.3,col='red')
print(Sys.time()-start)
print('Job finished!')
}

```