PROJET

L3 B2MCP - Année 2013 - 2014
Programmation en BioInformatique

Vincent Rocher

Correspondence:
vincent.rocher@etu.univ-rouen.fr
Full list of author information is available at the end of the article

Introduction

La mutation dans l'ADN est un processus d'évolution natural : les erreurs de réplication de l'ADN causent des substitutions, des insertions et des supressions de nucléotides, aboutissant à la modification de l'ADN. La similitude entre des séquences peut donc être l'indice d'une origine commune.

La génomique comparative a pour but de comparer une séquence inconnue avec une base de données de séquence connues afin d'en extraire de l'information. Cette méthode permet d'obtenir une similitude entre séquences, et d'en quantifier le degré de similitude.
On peut comparer les séquences entre elles de plusieurs façon, tel que l'alignement de portions de génomes ou de génomes complet, la comparaison de la composition des séquences ...

L'alignement de séquence est la principale méthode de comparaison des séquences, elle permet d'identifier les régions conservées et d'en déduire l'homologie.
Deux gènes ayant la même fonction dans leurs organismes respectifs ne signifie pas pour autant qu'ils sont homologues, de plus que deux gènes homologues peuvent ne pas avoir la même fonction. Il est donc essentiel de pouvoir analyser ces deux gènes, les comparer entres eux, pour savoir si ils sont bien homologues ou non.
Le but de ce projet est de construire par le biais de plusieurs langages (Python et R) un programme composé de plusieurs scripts, dans le but de comparer deux séquences, d'effectuer un alignement local (Smith & Waterman) et d'en déterminer l'homologie, par étude statistique.

Matériels et méthodes

Sélection des séquences

Dans ce rapport, nous utiliserons des séquences issues d'une **pectine esterase** de deux différents organismes :
Salmonella bongori lien NCBI
Et *Solanum lycopersicum* lien NCBI
Le pourcentage d'identité obtenu lors de l'alignement local sur le site d'EMBOSS[1] est le suivant (output):

```
#=======================================
#
# Aligned_sequences: 2
# 1: 63653960-63656206
# 2: c759315-758032
# Matrix: EDNAFULL
# Gap_penalty: 10.0
# Extend_penalty: 0.5
#
# Length: 1996
# Identity:     802/1996 (40.2%)
# Similarity:   802/1996 (40.2%)
# Gaps:         915/1996 (45.8%)
# Score: 1068.5
#
#
#=======================================
```

Pourquoi choisir ces deux séquences?

En choisissant deux séquences issues d'une enzyme ayant la même fonction, on veut vérifier que la proximité de leur rôle laisse supposer qu'elles peuvent dériver l'une de l'autre, mais peuvent aussi être issues d'une convergence évolutive et donc qu'il parait avisé de vérifier si deux organismes aussi éloignés phylogénétiquement peuvent avoir conserver cette sequence aussi longtemps auquel cas on pourrait conclure qu'elle est indispensable car très conservée, ou si au contraire c'est une convergence évolutive et que leur similarité de rôle est un hasard et qu'elles n'ont rien de plus en commun.
Rq : On enregistre ces deux séquences dans deux fichiers :
sequenceheinz.fasta pour la pectine esterase issue de *Solanum lycopersicum* et **sequencesalmo.fasta** pour la pectine esterase issue de *Salmonella bongori*.

Alignement des séquences

Choix de l'algorithme :

Dans le cas ou l'on cherche a trouver les ressemblances entres deux séquences assez similaires, l'utilisation d'un alignement global (c'est à dire comparer l'ensemble de la première séquence avec la deuxième) est recommandé.
Dans notre cas, le but visé est plutôt de trouver dans deux séquences relativement

différentes des régions similaires, séparées par des régions de faible similitude.
Dans ce cas, la méthode d'alignement global décrite plus haut ne donne en général pas de bons résultats puisqu'elle étire les deux séquences. L'algorithme de programmation dynamique de Smith et Waterman a été développé pour répondre au besoin d'alignement local. C'est une adaptation de la méthode de Needleman et Wunsch (alignement global), modifiée pour favoriser l'identification de régions communes.

Description de l'algorithme de Smith & Waterman :

Lorsque l'on exécute un alignement local, on va rechercher les identités en priorités, on doit aussi admettre l'insertion de "trous".
Afin de privilégier les identités par rapport aux insertions/délétions (indels), on doit choisir une valeur non nulle lorsque les deux lettres des deux séquences sont identiques, soit une identité (match). On doit également choisir une valeur (qui peut être nulle) lorsqu'il y a remplacement d'une lettre par une autre, soit substitution (mismatch).
Chaque cellule de la matrice contient donc un score qui peut être positif ou négatif suivant l'opération faite pour y arriver.
Ensuite, après le calcul dynamique des scores, toutes les cases de la matrice qui contiennent un score négatif sont mises à zéro.
La case de la matrice qui contient la valeur de score la plus élevée et qui est le plus en bas à droite correspond alors à l'alignement optimal, et on construit donc l'alignement à partir de cette case, en remontant jusqu'à ce qu'une case contienne un zéro.

Pseudo-code :

Algorithm 0.1: Algorithme de Smith & Waterman(x, y, M, N)

Deux séquences x et y de taille M et N, matrice de substitution S et cout d'indel D.
Déclaration :
$M \leftarrow length(x)$
$N \leftarrow length(y)$
$T \leftarrow [0] * m$
for $i \leftarrow 0$ **to** M
  do $T \leftarrow [0] * N$
Récursion :
for $i \leftarrow 1$ **to** M
  do for $j \leftarrow 1$ **to** N
  do $T(i,j) = max \begin{cases} 0, \\ T(i-1, j-1) + S, \\ T(i-1, j) + D, \\ T(i, j-1) + D, \end{cases}$
return (T)

Choix des langages utilisés

Pour réaliser ce projet, nous utiliserons 2 langages que sont Python[2] et R[3]. Le premier servira principalement à effectuer l'alignement local, le second servira à simuler des séquences aléatoires ($2 * 100$ ici) issues des deux séquences choisies à partir de leur distribution en nucléotides.
L'ensemble des scripts nécessaires à l'alignement(que ce soit des 2 séquences ou des $2 * 100$ séquences simulées) sont situés dans un seul programme : **alignement.py**
Pour R, les scripts sont divisés en deux programmes : **simulationseq.R** et **stat.R**
Le premier simule 100 séquences aléatoires issues d'une séquence et l'enregistre dans un fichier Multifasta, le second lui est nécessaire à la réalisation des tests statistique en rapport avec l'alignement.
Nous montrerons ci-dessous les fonctions utilisées en Python et en R pour la réalisation du projet.

Liste des fonctions Python :

Il y a 4 fonctions utilisées dans le programme **alignement.py**, essentielles à la réalisation de l'alignement :

Multifasta (lireFastaMul) : La première fonction a pour rôle d'ouvrir un fichier Multifasta et d'en extraire les séquences (fonctionne aussi avec du simple fasta).

Matrice de substitution (msubst) : La seconde fonction consiste a renvoyer un score en fonction des lettres présentes dans les deux séquences.

Alignement (Align) : La troisième fonction est l'alignement à proprement parlé, qui va comparer chaque nucleotides des deux séquences en fonction de l'algorithme de Smith & Waterman.

Tracé arrière (backtrac) : La dernière fonction sert à réaliser le tracé arrière par rapport au tableau de score obtenu par l'alignement.

Rq : Chaque fonction est détaillé dans le programme par des commentaires.

Lecture des fichiers Multifasta

```
def lireFastaMul(nomFichier):
    fichier=open(nomFichier,"r")
    lignes=fichier.readlines()
    fichier.close()
    sequence,nom="",""
    sequences,noms=[],[]
        for ligne in lignes:
        if ligne[0]==">":
            if sequence != "":
                noms.append(nom)
                sequences.append(sequence)
            nom=ligne[1:-1]
            sequence=""
        else:
            sequence=sequence+ligne[:-1]
    if sequence !="":
        noms.append(nom)
        sequences.append(sequence)
    return (noms,sequences)
```

Explication :
La fonction **LireFastaMul** va récupérer une valeur **nomfichier** contenant une destination de fichier, par exemple *"/Users/.../Documents/sequenceheinz.fasta"*, le fichier va ensuite être ouvert en mode lecture sous le nom **fichier**, l'ensemble du contenu de **fichier** va être ensuite enregistré dans la variable **lignes**, puis le fichier est fermé.
Ensuite pour chaque **ligne** de l'ensemble des **lignes**, la boucle va vérifier si le premier caractère est un **'>'**, si c'est le cas, la ligne va être enregistré dans la variable **nom**, puis ajouté à la liste **noms**, si ce n'est pas le cas, la ligne sera enregistrée dans la variable **sequence**, à la suite des autres lignes, sans le dernier caractère (qui est un saut de ligne).
A chaque fois que la boucle croise un **'>'**, la variable **sequence** sera enregistrée dans la liste **sequences**, qui contiendra l'ensemble des séquences du fichier Multifasta.

Matrice de substitution

Pour assimiler un score à une identité ou à une substitution entre deux nucléotides, il est nécessaire de construire une matrice de substitution sur mesure, répondant aux critères du programme et des nucléotides présents dans les séquences fasta.

Définition : Une matrice de substitution permet, pour chaque acide aminé, de connaître sa capacité à être substitué par chaque autre acide aminé, y compris lui-même.
Lorsque le programme d'alignement Smith & Waterman calculera le score maximum

pour chaque case, il va également appeler une fonction **msubst**, qui contiendra la matrice de substitution, avec le score d'identité substitution défini par l'utilisateur. On peut illustrer cette matrice de la façon suivante :

	A	**C**	**G**	**T**	**W**	**Y**	**R**	**N**
A	Y	N	M	N	Y	N	Y	Y
C	N	Y	N	M	N	Y	N	Y
G	M	N	Y	N	N	N	Y	Y
T	N	M	N	Y	Y	Y	N	Y
W	Y	N	N	Y	Y	N	N	Y
Y	N	Y	N	Y	N	Y	N	Y
R	Y	N	Y	N	N	N	N	Y
N	Y	Y	Y	Y	Y	Y	Y	Y

Où :
Y : Score Match.
N : Score Mismatch.
M : Score Mismatch mais même type de base (purique ou pyrimidique).
On remarque également la présence des lettres W,Y,R et N, qui ne sont pas des nucléotides, mais qui sont parfois présentes dans les séquences, et qui possèdent des propriétés particulières :
W : Est soit un A soit un T, donc renverra une identité dans le cas ou les deux séquences ont un A,T ou W en commun.
Y : Est soit un C soit un T, donc renverra une identité dans le cas ou les deux séquences ont un C,T ou Y en commun.
R : Est soit un A soit un G, donc renverra une identité dans le cas ou les deux séquences ont un A,G ou R en commun.
N : N peut être tout les nucléotides à la fois, donc on suppose qu'il y aura identité pour n'importe quelle lettre.

Voici le rendu en code python de la fonction **msubst**:

```
def msubst(L,C):
    global Y,N,M
    sub=[
        #A C G T W Y R N
        #0 1 2 3 4 5 6 7
        [Y,N,M,N,Y,N,Y,Y],#A/0
        [N,Y,N,M,N,Y,N,Y],#C/1
        [M,N,Y,N,N,N,Y,Y],#G/2
        [N,M,N,Y,Y,Y,N,Y],#T/3
        [Y,N,N,Y,Y,N,N,Y],#W/4
        [N,Y,N,Y,N,Y,N,Y],#Y/5
        [Y,N,Y,N,N,N,N,Y],#R/6
        [Y,Y,Y,Y,Y,Y,Y,Y]]#N/7
    eq = {'A':0,'C':1,'G':2,'T':3,'W':4,'Y':5,'R':6,'N':7}
    score=sub[eq[L]][eq[C]]
    return(float(score))
```

la variable **sub** est un tableau(liste de liste) représentant en python la matrice de substitution vu plus haut, il permet de renvoyer un score déterminé par l'utilisateur en fonction du numéro de la ligne et de la colonne.

`eq = {'A':0,'C':1,'G':2,'T':3,'W':4,'Y':5,'R':6,'N':7}`
Correspond à la création d'un dictionnaire, ou chaque lettre renvoi un numéro, ainsi, il suffira au programme d'aller chercher l'élément colonne 0 si c'est un 'A' , ligne 0 si il compare avec un autre 'A', et renvoyer la valeur de score correspondant, ici **Y**.

`score=sub[eq[L]][eq[C]]`
Le code ci-dessus renverra donc un score, en fonction du numéro reçu par le dictionnaire, selon les lettres des deux séquences.

Alignement des séquences

```
def Align(seq1,seq2):
    m=len(seq1)+1
    n=len(seq2)+1
    T=[0]*m
    for i in range(m):
        T[i]=[0]*n
    i=j=1
    maxi=0
    while i<m:
        while j<n:
            p1=i-1
            p2=j-1
            score1=T[i-1][j-1]+msubst(seq1[p1],seq2[p2])
            score2=T[i][j-1]+d
            score3=T[i-1][j]+d
            T[i][j]=max(score1,score2,score3,0)
            if T[i][j]>=maxi:
                maxi=T[i][j]
                imax=i
                jmax=j
            j=j+1
        i=i+1
        j=1
    return (T,maxi,imax,jmax)
```

La fonction d'alignement est basée sur l'algorithme de Smith & Waterman vu plus haut, il est donc nécessaire de construire une matrice de taille M et N issue des séquences $(x+1)$ et $(y+1)$, illustré par :

```
    T=[0]*m
    for i in range(m):
        T[i]=[0]*n
```

Ce code va construire un tableau contenant m fois 0, puis par une boucle for, qui pour chaque élément i de la liste T, les remplaces par n fois 0, permettant de construire une table des scores, pour l'instant rempli uniquement de zéros, mais faisant la taille+1 des séquences.
La suite du code consiste en une boucle qui pour chaque élément i de m, va pour chaque élément j de n, va calculer le score maximum entre :

$$\begin{gathered} 0, \\ T(i-1,j-1)+S, \\ T(i-1,j)+D, \\ T(i,j-1)+D, \end{gathered}$$

Tracé arrière

Le fonction permettant d'effectuer le tracé arrière n'est pas indispensable au score, elle permet cependant de montrer le meilleur alignement possible entre les deux séquences sélectionnées, par rapport aux paramètres et à l'algorithme utilisé.

```
def backtrac(seq1,seq2,T,imax,jmax):
    Align1=""
    Align2=""
    i=len(seq1[0:imax])
    j=len(seq2[0:jmax])
    while (i>0) or (j>0):
        if (T[i][j]==0):
            i=0
        elif (i>0) and (j>0) and (T[i][j]==T[i-1][j-1]+msubst(seq1[i-1],seq2[j-1])):
            Align1=seq1[i-1]+Align1
            Align2=seq2[j-1]+Align2
            i=i-1
            j=j-1
        elif (i>0) and (T[i][j]==T[i-1][j]+d):
            Align1=seq1[i-1]+Align1
            Align2="-"+Align2
            i=i-1
        elif (j>0) and (T[i][j]==T[i][j-1]+d):
            Align1="-"+Align1
            Align2=seq2[j-1]+Align2
            j=j-1
    return(Align1,Align2)
```

Le tracé arrière se base plus ou moins sur la même logique que l'alignement sauf qu'on part de la fin, c'est à dire ou le score est le plus élevé(imax, jmax), puis on remonte jusqu'à trouver un 0. On peut le représenter en pseudo-code de la façon suivante :

Algorithm 0.2: Algorithme de Smith & Waterman: Tracé arrière$(x, y, T, imax, jmax)$

Deux séquences x et y de taille M et N, matrice de substitution S et cout d'indel D.
Déclaration :
$AlignmentA \leftarrow$ ""
$AlignmentB \leftarrow$ ""
$i \leftarrow M[0 : imax]$
$j \leftarrow N[0 : jmax]$
while $(i > 0)$**or**$(j > 0)$
do
if $(T[i][j] == 0)$

then $i = 0$
comment: Pour break la boucle si on rencontre un 0
else if $(i > 0)$ **and** $(j > 0)$ **and** $T(i, j) == T(i - 1, j - 1) + S(Ai, Bj)$

$$\textbf{then} \begin{cases} AlignmentA \leftarrow Ai + AlignmentA \\ AlignmentB \leftarrow Bj + AlignmentB \\ i \leftarrow i - 1 \\ j \leftarrow j - 1 \end{cases}$$

else if $(i > 0)$ **and** $T(i, j) == T(i - 1, j) + D$

$$\textbf{then} \begin{cases} AlignmentA \leftarrow Ai + AlignmentA \\ AlignmentB \leftarrow \text{" } - \text{ "} + AlignmentB \\ i \leftarrow i - 1 \end{cases}$$

else $(j > 0)$ **and** $T(i, j) == T(i, j - 1) + D$

$$\textbf{then} \begin{cases} AlignmentA \leftarrow \text{" } - \text{ "} + AlignmentA \\ AlignmentB \leftarrow Bj + AlignmentB \\ j \leftarrow j - 1 \end{cases}$$

return $(AlignmentA, AlignmentB)$

alignement.py (cf Annexe)

Le programme **alignement.py** regroupe toutes les fonctions et le code Python nécessaire à l'alignement dans ce rapport, il est structuré de la manière suivante :

On commence par déclarer les variable(ici des listes) qui vont récupérer le(s) séquences et leurs noms :

```
noms1=[]
noms2=[]
sequences1=[]
sequences2=[]
```

Ensuite on interagit avec l'utilisateur via des input, on obtient ainsi :
noms(1, 2),séquences(1,2) : va envoyer le contenu de l'input (soit la destination

du fichier) à la fonction **LireFastaMul** qui va renvoyer deux listes, qu'on enregistrera dans les deux variables.
Ensuite suit les variables qui vont demander la valeur de score attribuée en cas d'indel (**d**),le code va vérifier si cette valeur est négative, si non, elle renvoi une valeur négative.**Y**,**N** et **M** contiendront respectivement les valeurs de score d'une identité, d'une substitution et d'une substitution dans le cas ou ce sont les mêmes types de bases (purique, pyrimidique).
back contiendra l'information si l'utilisateur veut ou non effectuer le tracé arrière (déconseillé dans le cas ou on aligne plusieurs séquences).

```
noms1,sequences1=lireFastaMul(input('Destination du fichier séquence(s) 1: \n'))
noms2,sequences2=lireFastaMul(input('Destination du fichier séquence(s) 2: \n'))
d=float(input('Score d\'indel : '))
if d>0:
    d=-d
Y=float(input('Score match : '))
N=float(input('Score mismatch : '))
M=float(input('Score mismatch même type de base purique/pyrimidique : '))
back=input('Voulez vous effectuer le tracé arrière ?(y/n) :')
```

Ce code va enregistrer le nombre de séquences contenu dans les fichiers fasta, et dans le cas ou le nombre de séquence n'est pas égal dans les deux fichiers, il prend la valeur du plus petit nombre (afin de ne pas répéter la boucle plus de fois qu'on ne peut aligner de séquence).
Ce compteur va servir à répéter la boucle qui contiendra la fonction d'alignement.

```
if len(sequences1)<len(sequences2):
    nombreseq=len(sequences1)
else:
    nombreseq=len(sequences2)
```

Boucle faisant appel x fois aux fonctions **Align** et **backtrac**, x étant le nombre de séquences à comparer :

```
i=0
scores=[]
T=[]
while i<nombreseq:
    T,score,imax,jmax=Align(sequences1[i],sequences2[i])
    scores.append(score)
    if back=='y':
        align1,align2=backtrac(sequences1[i],sequences2[i],T,imax,jmax)
    T=[]
    i=i+1
```

La suite du programme est la partie la plus importante, le code contient la boucle qui va effectuer l'alignement tant que i n'est pas égal aux nombres de séquences (donc si il y en a 100, la boucle s'exécutera 100 fois).
La boucle appelle la fonction Align (cf plus haut), et va lui envoyer l'élément **i** des

deux listes contenant les séquences à comparer.
L'intérêt d'enregistrer les séquences issues des fichiers dans des listes, même lorsqu'il n'y en a qu'une par fichier est démontré ici, car du coup on sait combien de séquences on doit comparer, c'est à dire combien de fois la boucle doit s'exécuter.
Dans le cas ou la variable **back** correspond à **y**, la boucle va également effectuer le tracé arrière de l'alignement grâce à la fonction **backtrac**, et va récupérer le résultat dans deux variables, qu'on affichera ensuite.
On remet T=[] afin qu'il puisse contenir une nouvelle table, issue de l'alignement des séquences suivantes, sans que T contienne encore la table de score des séquences que nous venons de comparer.

Liste des fonctions R :

Pour le langage R nous n'utiliserons que deux fonctions, ainsi que la librairie **seqinr**[4].

Calcul du pourcentage de lettres présent dans une séquence

Puisque nous travaillons sur deux séquences, il est nécessaire d'utiliser une fonction qui va calculer le pourcentage de chaque nucléotide, puis renvoyer le résultat à l'utilisateur, ainsi nous n'auront pas besoin de répéter le code deux fois :

```
prob<-function(masequence){
  compo<-table(masequence)
  probtot<-table(masequence)/length(masequence)
  return (probtot)
}
```

Le code suivant consiste à récupérer une séquence extraite d'un fichier fasta (via **seqinr**), d'afficher via la fonction **table** le nombre de chaque type de nucléotide présent dans la séquence, et de diviser chaque nombre par le nombre total de nucléotide de la séquence.
Ainsi nous obtiendrons un résultat sous la forme :

```
        a         c         g         t
0.2986204 0.1744548 0.2216288 0.3052960
```

Ce pourcentage va nous servir pour générer 100 séquences issue d'une seule séquence, grâce à la fonction qui va suivre.

Simulation et écriture de 100 séquences dans un fichier multifasta

Une fois le pourcentage de chaque nucléotide obtenu, nous pouvons écrire dans un fichier multifasta 100 séquences issues du pourcentage de la séquence d'origine :

```
ecriture<-function(taille,proba,filename) {
  for (i in 1:100){
    nom<-paste(i,"_seq_1",sep="")
    echantillon<-sample(c("A","C","G","T"),taille,replace<-T,prob<-proba)
    write.fasta(sequences<-echantillon, name<-nom,file.out<-filename,open<-'a')
  }
}
```

La fonction créer une boucle qui, 100 fois va écrire une séquence issue de la séquence d'origine (et de même taille) dans un même fichier.
Il est important d'ajouter replace=T afin que R ne répète pas 'A,C,G,T' en boucle mais qu'il se servent de ces 4 lettres comme d'un alphabet pour distribuer, en fonction du pourcentage voulu, les lettres dans la séquence. La fonction
`write.fasta(sequences<-echantillon, name<-nom,file.out<-filename,open<-'a')`
est issue de **seqinr** et permet d'écrire, en mode écriture sans réécrire par dessus le fichier ('a'), au format fasta, voir multifasta dans ce cas là.

simulationseq.R

Voici le premier programme que nous utiliserons, il permet de simuler 2*100 séquences aléatoires de même pourcentage que deux séquences "mères" et d'écrire ces séquences simulées dans deux fichier multifasta différents.
Rq : Il est nécessaire de lancer ce fichier dans R de la manière suivante :
`source('simulationseq.R')`
Et ce afin de lancer l'interaction utilisateur (input).

Le programme démarre, on demande à l'utilisateur d'envoyer le chemin d'accès aux fichiers contenant les deux séquences (équivalent input) :

```
fasta1<-readline(prompt<-"Indiquez le chemin du premier fichier : ")
fasta2<-readline(prompt<-"Indiquez le chemin du premier fichier : ")
```

Ensuite, on déclare qu'on utilise la librairie **seqinr**, pour pouvoir lire correctement un fichier fasta, et de pouvoir écrire au format fasta dans un fichier, on déclare ensuite les vecteurs qui contiendront les séquences 1 et 2 :

```
library("seqinr")
tseq1<-read.fasta(file<-fasta1)
seq1<-tseq1[[1]]
tseq2<-read.fasta(file<-fasta2)
seq2<-tseq2[[1]]
```

On détermine la taille des deux vecteurs seq1 et seq2, afin de simuler des séquences de même taille :

```
taille1<-length(seq1)
taille2<-length(seq2)
```

On appelle la fonction **prob** (cf plus haut) qui va renvoyer le pourcentage de chaque nucléotide dans les deux séquences :

```
proba1<-prob(seq1)
proba2<-prob(seq2)
```

On détermine un nom pour les deux fichiers, grâce à un nombre aléatoire, ainsi nous n'écrirons jamais dans le même fichier (si on relance le script), et les deux fichiers auront le même numéro :

```
nombre<-sample(100:10000,1)
filename1<-paste(nombre, "_echantillon1.txt", sep="")
filename2<-paste(nombre, "_echantillon2.txt", sep="")
```

Ensuite on appelle la fonction **ecriture**, qui va récupérer la taille, le pourcentage en nucléotide, et le nom que doit prendre le fichier pour chaque séquence et simuler 100 séquences dans un même fichier multifasta (cf plus haut) :

```
ecriture(taille1,proba1,filename1)
ecriture(taille2,proba2,filename2)
```

Enfin, on indique à l'utilisateur le nom des deux fichiers produits, pour qu'il puisse les utiliser dans python pour réaliser les 100 alignements :

```
cat(filename1,' et ',filename2,'produits',sep<-"")
```

stat.R

Le second et dernier programme utilisé sous R consiste à récupérer les scores des séquences simulées, le score obtenu lors de l'alignement des séquences principales, et affiche la p-value et un graphique représentant les scores simulées :

Le programme démarre, on demande à l'utilisateur d'envoyer le résultat du score de l'alignement et du fichier score contenant les 100 scores (équivalent input) :

```
fichierscore<-readline(prompt<-"Indiquez le chemin du fichier 'scores.txt' : ")
mainscore<-readline(prompt<-"Indiquez le score obtenu lors de l'alignement
 des deux séquences principales : ")
```

On récupère les valeurs de scores dans une variable,
on calcule le nombre de scores supérieurs au score de l'alignement principal, et on calcul le nombre de scores dans le fichier (normalement 100) :

```
scores<-scan(fichierscore)
scoresup<-sum(scores>as.numeric(mainscore))
nombrescore<-length(scores)
```

Enfin on calcule la p-value grâce au calcul suivant : Estimation p-value= $\frac{(S>P)}{TotalS}$
Ou $S > P$ correspond aux scores supérieurs à P (score principal) et $TotalS$ le nombre de scores (100)?
Enfin on affiche un graphique qui correspond aux différentes valeurs de score des 100 alignements :

```
pvalue<-scoresup/nombrescore
cat('La p-value observée du score est de',pvalue,'\n')
plot(density(scores), xlab="Valeur des scores", ylab="densité")
```

Résultats

Réponses aux questions

Puisque nous avons l'ensemble des outils nécessaire au bon fonctionnement des programmes, pourquoi ne pas les exécuter ?
Lançons le script **alignement.py** dans une fenêtre terminale :

```
$ python3 alignement.py
```

Le programme va nous poser plusieurs questions (on peut choisir ou non d'effectuer le tracé arrière, dans notre exemple, nous cherchons juste le score):

```
Last login: Fri Apr 25 18:57:21 on ttys000
MacBook-Pro-de-Vincent:~ Vincent$ python3 /Users/Vincent/Documents/Python/Projet-Prog/alignement.py
##################################################
#                                                #
#  Programme d'alignement local Smith & Waterman   #
#  Réalisé dans le cadre du projet Programmation   #
#                En Bio-Informatique               #
#                Par ROCHER Vincent                #
#           Version de python :  3.3.2             #
#                                                #
##################################################
Destination du fichier de la ou des séquence(s) 1:
/Users/Vincent/Documents/sequenceheinz.fasta
Destination du fichier de la ou des séquence(s) 2:
/Users/Vincent/Documents/sequencesalmo.fasta
Score d'indel : -2
Score match : 1
Score mismatch : 0
Score mismatch même type de base purique/pyrimidique (possibilité de mettre même score que mismatch) : 0
Voulez vous effectuer le tracé arrière ?(y/n) :n
Début de l'alignement ... Veuillez patienter.
Comparaison de :
gi|459212741:63653960-63656206 Solanum lycopersicum cultivar Heinz 1706 chromosome 7, SL2.40, whole genome shotgun sequence
Avec :
gi|339998036:c759315-758032 Salmonella bongori NCTC 12419, complete genome
En cours...
Sequences (1_1:2_1) Aligned. Score : 452.0
MacBook-Pro-de-Vincent:~ Vincent$
```

On obtient l'ouput suivant :

`Sequences (1_1:2_1) Aligned. Score : 452.0`
L'alignement local du programme nous propose un score de 452, par rapport aux pénalités/récompenses que nous avons choisis.

Sous R, nous utilisons le programme **simulationseq.R**, toujours dans une fenêtre terminale :

```
MacBook-Pro-de-Vincent:~ Vincent$ R

R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin10.8.0 (64-bit)

R est un logiciel libre livré sans AUCUNE GARANTIE.
Vous pouvez le redistribuer sous certaines conditions.
Tapez 'license()' ou 'licence()' pour plus de détails.

R est un projet collaboratif avec de nombreux contributeurs.
Tapez 'contributors()' pour plus d'information et
'citation()' pour la façon de le citer dans les publications.

Tapez 'demo()' pour des démonstrations, 'help()' pour l'aide
en ligne ou 'help.start()' pour obtenir l'aide au format HTML.
Tapez 'q()' pour quitter R.

[Sauvegarde de la session précédente restaurée]

> source('/Users/Vincent/Documents/Python/Projet-Prog/simulationseq.R')
Indiquez le chemin du premier fichier : /Users/Vincent/Documents/sequenceheinz.fasta
Indiquez le chemin du premier fichier : /Users/Vincent/Documents/sequencesalmo.fasta
7058_echantillon1.txt  et  7058_echantillon2.txt produits >
```

On lance le programme avec :
`source('/Users/Vincent/Documents/Python/Projet-Prog/simulationseq.R')`

Ce qui nous produit l'output :

`7058_echantillon1.txt  et  7058_echantillon2.txt produits`

Dans ces deux fichiers, il y'a 100 séquences simulées, produites par R, et issues des deux séquences principales, le numéro du fichier est aléatoire, mais les deux fichiers portent le même numéro (pour qu'on puisse les reconnaitre).

On relance à nouveau le programme **alignement.py** sous python, mais avec cette fois les deux fichier produits par R, cette fois, le programme va aligner chaque séquence des deux fichiers une par une, et écrire l'ensemble des 100 scores obtenues dans un fichier **scores.txt** :

```
Sequences (1_100:2_100) Aligned. Score : 454.0
fichier scores.txt produit.
```

Une fois le score des séquences principales et les scores des 100 alignements des séquences simulées obtenus (**scores.txt**), il est important de réaliser un test statistique qui permettra de quantifier la probabilité que le score obtenu soit le fruit du hasard ou s'il est représentatif d'homologie de séquences.
On peut alors tenter de répondre à la question sur le rôle du hasard en construisant un test d'hypothèse de la manière suivante :

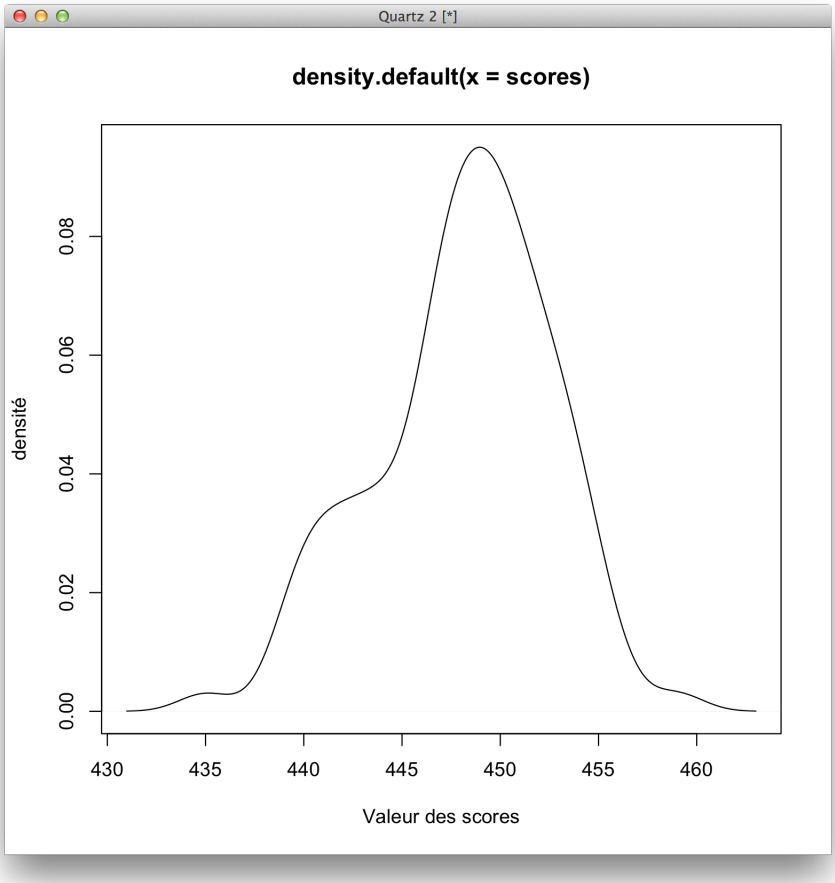
H0 : " Le score observé aurait pu être obtenu a partir de séquences aléatoires. " (D'où l'importance de simuler 2*100 séquences aléatoires).
H1 : " Le score observé n'est pas le fait du hasard. "

On peut aussi reformuler :

H0 : " Les deux séquences ont des distributions de nucléotides indépendantes. " contre l'alternative
H1 : " Les distributions des deux séquences sont liées."

Afin d'effectuer ce test statistique, on se sert du dernier programme R **stat.R**, celui-ci, à partir des scores obtenus lors des alignement, va calculer la p-value relative à l'alignement entre les séquences de *Salmonella bongori* et *Solanum lycopersicum* :
En choisissant une région critique de 20% pour laquelle on ne rejette pas H0, on obtient le résultat suivant :

```
> source('/Users/Vincent/Documents/Python/Projet-Prog/stat.R')
Indiquez le chemin du fichier 'scores.txt' : /Users/Vincent/scores.txt
Indiquez le score obtenu lors de l'alignement des deux séquences principales : 452
Indiquez une valeur pour alpha (ex : 0.1) : 0.20
Read 100 items
La p-value observée du score est de 0.16
On rejette H0>
```



Graphique représentant les scores obtenues lors des alignements

Dans notre cas, on peut dire qu'au risque de 20% (ce qui est élevé), nos séquences ont des distributions liées, c'est à dire qu'on peut les considérer comme homologue, cependant, au vue de la p-value obtenue, on peut affirmer qu'elles sont faiblement homologues.
La p-value n'est pas une valeur parfaite, mais elle permet de donner une bonne idée de l'homologie entres deux séquences.

Conclusion de l'analyse statistique :
Si la p-value est faible, on rejette l'hypothèse H0. Plus la p-value est faible, plus l'alignement est statistiquement significatif.
La p-value permet donc d'avoir une idée sur l'homologie entre deux séquences, car le score donné n'est pas très indicatif, surtout qu'il dépend des valeurs de scores attribuées et de la longueur des séquences.
Il est donc nécessaire de simuler des séquences de même taille et de même distribution que les deux séquences à aligner afin d'avoir une idée du score "palier", c'est à dire le score obtenu alors que les séquences sont totalement aléatoires, on peut alors analyser la différence entre cette valeur là (le mieux est d'en simuler plusieurs, comme nous avons fait) et la valeur de score obtenue.

Un alignement sans étude statistique derrière n'a aucune valeur, puisqu'il n'apporte pas d'information qualitative sur l'homologie des deux séquences.[5]
On pourrait aller plus loin en analysant un alignement entre deux pectines esterase de la même espèce, en sachant leur pourcentage d'identité, et voir si la p-value observée est différente avec nos résultat.

References

1. European Bioinformatics Institute
2. Python Software Foundation: Imp – Access the Import Internals. http://www.python.org (2005). Python Software Foundation
3. R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2013). R Foundation for Statistical Computing. http://www.R-project.org/
4. Charif, D., Lobry, J.R.: SeqinR 1.0-2: a contributed package to the R project for statistical computing devoted to biological sequences retrieval and analysis. In: Bastolla, U., Porto, M., Roman, H.E., Vendruscolo, M. (eds.) Structural Approaches to Sequence Evolution: Molecules, Networks, Populations. Biological and Medical Physics, Biomedical Engineering, pp. 207–232. Springer, New York (2007). ISBN : 978-3-540-35305-8
5. Rost, B.: Twilight zone of protein sequence alignments. Protein engineering (1999)

Annexe : Ensemble des scripts R et Pyhton
simulationseq.R

```
#!/usr/bin/env Rscript
#Permet de remove l'environnement pour ne pas
#que les vecteurs gardent ceux du programme précédent.
rm(list = ls(all = TRUE))
##############FONCTIONS##############
#Fonction qui va écrire les 100 séquences dans un fichier texte au format fasta
ecriture<-function(taille,proba,filename) {
  for (i in 1:100){
    nom<-paste(i,"_seq_1",sep="")
    echantillon<-sample(c("A","C","G","T"),taille,replace<-T,prob<-proba)
    write.fasta(sequences<-echantillon, name<-nom,file.out<-filename,open<-'a')
  }
}
#Fonction qui calcule la distribution en nucléotide d'une séquence, et renvoi 4 valeurs :
prob<-function(masequence){
  compo<-table(masequence)
  probtot<-table(masequence)/length(masequence)
  return (probtot)
}
#####################################
###INTERACTION UTILISATEUR (input)###
#vecteur qui prend la valeur de la destination du fichier 1 entré par l'utilisateur
fasta1<-readline(prompt<-"Indiquez le chemin du premier fichier : ")
#vecteur qui prend la valeur de la destination du fichier 2 entré par l'utilisateur
fasta2<-readline(prompt<-"Indiquez le chemin du premier fichier : ")
#####################################

library("seqinr")#On déclare qu'on utilise seqinr pour lire les fichiers fasta

#Puis on lis les fichiers fasta, on les enregistre dans un vecteur
#et on selectionne uniquement la séquence avec [[1]] :
#Séquence 1
tseq1<-read.fasta(file<-fasta1)
seq1<-tseq1[[1]]
#Séquence 2
tseq2<-read.fasta(file<-fasta2)
seq2<-tseq2[[1]]

#On détermine la taille des séquence pour créer des echantillons de la même taille :
taille1<-length(seq1)#Nombre de nucléotides dans la séquence 1
taille2<-length(seq2)#Nombre de nucléotides dans la séquence 2

#On utilise la fonction prob pour calculer la distribution en nucleotide de chaque séquence :
proba1<-prob(seq1)#proba1 prend les valeur de probabilitées de la séquence 1
proba2<-prob(seq2)#proba2 prend les valeur de probabilitées de la séquence 2
#On génère ensuite un nombre aléatoire entre 100 et 10000
#afin d'obtenir un nom de fichier sortant unique :
nombre<-sample(100:10000,1)
#On additione ce nombre avec "_echantillon1.txt" et "_echantillon2.txt"
#pour obtenir le nom des deux fichiers :
#Nom du fichier contenant les 100 seq issues de seq1 :
filename1<-paste(nombre, "_echantillon1.txt", sep="")
filename2<-paste(nombre, "_echantillon2.txt", sep="")
#Nom du fichier contenant les 100 seq issues de seq2 :
#Enfin on appelle la fonction ecriture
#qui va écrire 2*100 fois une séquence issue de seq1 et seq2, dans deux fichiers séparées :
ecriture(taille1,proba1,filename1)#On écris dans un premier fichier, 100 seq de seq1
ecriture(taille2,proba2,filename2)#On écris dans un second fichier, 100 seq de seq2

#On print le nom des deux fichiers à l'utilisateur  :
cat(filename1,' et ',filename2,'produits',sep<-"")
```

stat.R

```
#!/usr/bin/env Rscript
###INTERACTION UTILISATEUR (input)###
#vecteur qui prend la valeur de la destination du fichier scores.txt
fichierscore<-readline(prompt<-"Indiquez le chemin du fichier 'scores.txt' : ")
#Enregistre le score des deux séquences d'origine
mainscore<-readline(prompt<-"Indiquez le score obtenu lors de l'alignement des deux séquences principales : ")
alpha<-readline(prompt<-"Indiquez une valeur pour alpha (ex : 0.1) : ")
#######################################
#######################################
#########TEST D'INDEPENDANCE#########
###CALCUL DE LA P-VALUE DE L'ALIGNEMENT###
#Scan le fichier et enregistre les scores dans un vecteur :
scores<-scan(fichierscore)
#Calcule le nombre de scores parmis les scores des 100 séquences
#simulées lesquels sont supérieur au score des séquences d'origine :
scoresup<-sum(scores>as.numeric(mainscore))
nombrescore<-length(scores)#Calcule le nombre de scores dans le fichier (normalement 100)

pvalue<-scoresup/nombrescore#Va calculer la p-value entre le score principal et ceux générés par les séquences aléatoires
#Donne le résultat de la p-value du score et le résultat
cat('La p-value observée du score est de',pvalue,'\n')
plot(density(scores), xlab="Valeur des scores", ylab="densité")
if (pvalue>alpha){
  cat('On ne rejette pas HO')
}else{
  cat('On rejette HO')
}
```

alignement.py

```
#!/usr/bin/python3
# -*- coding: utf8 -*-
#Programme : alignement.py
#Auteur : Vincent ROCHER
#But du programme : Alignement local(Smith et Waterman), avec ou sans tracé arrière.
#
##############FONCTIONS################
import sys #Module qui permet la vérification de la version de Python de l'utilisateur
if sys.version_info < (3, 0): #Si Version de python inférieur à 3 :
    exit() #Quitter python

else:
    ######MULTIFASTA###################
#Lecture du fichier fasta, récupère le nom et le contenu de la séquence
#de chaque séquence dans deux listes, puis renvoi les deux listes :
    def lireFastaMul(nomFichier):
        fichier=open(nomFichier,"r")#Ouverture en mode lecture
        lignes=fichier.readlines()#Récupère toutes les lignes du fichier
        fichier.close()#Fermeture du fichier
    #Declaration des variables et des listes :
        sequence=""
        nom=""
        sequences=[]
        noms=[]
    #Boucle de lecture
        for ligne in lignes:#Parcours chaque ligne du fichier
            if ligne[0]==">":#Si le premier élément de la ligne contient un '>'
                if sequence != "":#Enregistre seulement si sequence contient quelque chose, évite d'enregistrer le sequence="" du débu
                    noms.append(nom)#Enregistre le nom dans la liste des noms
                    sequences.append(sequence)#Enregistre la séquence dans la liste des séquences
                nom=ligne[1:-1]#Enregistre le nom de la séquence sans le '>' et sans le saut de ligne
                sequence=""#Remet la valeur de la variable à 0 pour ne pas enregistrer deux séquences en même temps
            else:#Enregistre ligne dans la variable séquence et sans saut de ligne
                sequence=sequence+ligne[:-1]
        if sequence !="":#Permet d'enregistrer la dernière séquence car n'enregistre qu'après avoir vu un '>'
            noms.append(nom)
            sequences.append(sequence)
        return (noms,sequences)#Renvoi des deux listes contenant toutes les séquences

    ######TABLEAU DE SUBSTITUTION######
    def msubst(L,C):
#L=Nucléotide qui correspondra à la ligne
#C=Nucléotide qui correspondra à la colonne
#Y=score match déterminé par l'utilisateur
#N=score mismatch déterminé par l'utilisateur
#M=score mismatch mais même type de base
        global Y,N,M
        sub=[
            #A C G T W Y R N
            #0 1 2 3 4 5 6 7
            [Y,N,M,N,Y,N,Y,Y],#A/0
            [N,Y,N,M,N,Y,N,Y],#C/1
            [M,N,Y,N,N,N,Y,Y],#G/2
            [N,M,N,Y,Y,Y,N,Y],#T/3
            [Y,N,N,Y,Y,N,N,Y],#W/4
            [N,Y,N,Y,N,Y,N,Y],#Y/5
            [Y,N,Y,N,N,N,N,Y],#R/6
            [Y,Y,Y,Y,Y,Y,Y,Y]]#N/7
        eq = {'A':0,'C':1,'G':2,'T':3,'W':4,'Y':5,'R':6,'N':7}#Dictionnaire qui remplace les lettres par leur numéro correspondant dan
        score=sub[eq[L]][eq[C]]
        return(float(score))

    ######ALIGNEMENT LOCAL##############
    def Align(seq1,seq2):
    ###Construction Matrice#####
        m=len(seq1)+1 #m = nombre de lignes
        n=len(seq2)+1 #n = nombre de colonnes
        T=[0]*m #Construction de m lignes
        for i in range(m):
```

```
        T[i]=[0]*n #Construction de n colonnes
#########Alignement#########
    i=j=1
    maxi=0
    while i<m:
        while j<n:
        #Nécessité d'enlever -1 pour obtenir la bonne position de chaque nucleotides
            p1=i-1#position du nucleotide de la séquence 1
            p2=j-1#position du nucleotide de la séquence 2
            score1=T[i-1][j-1]+msubst(seq1[p1],seq2[p2])#Score d'un match/mismatch
            score2=T[i][j-1]+d#Insertion dans la séquence 1, délétion dans séquence 2
            score3=T[i-1][j]+d#Délétion dans la séquence 1, insertion dans séquence 1
            T[i][j]=max(score1,score2,score3,0)#Calcul du score maximum et insertion dans Table
            if T[i][j]>=maxi:
                maxi=T[i][j]
                imax=i#Coordonnées du score max au niveau des lignes (i)
                jmax=j#Coordonnées du score max au niveau des colonnes (j)
            j=j+1
        i=i+1
        j=1#Remet j à 1 pour faire recommencer la boucle a la première colonne
    return (T,maxi,imax,jmax)
######TRACE ARRIERE################
def backtrac(seq1,seq2,T,imax,jmax):
    Align1="" #Contiendra le premier alignement
    Align2="" #Contiendra le second alignement
    Align3="" #Va servir a illustrer l'alignement dans le html par des . ou des |
    i=len(seq1[0:imax])#i va prendre comme valeur imax
    j=len(seq2[0:jmax])#j va prendre comme valeur jmax
    while (i>0) or (j>0): #Continu tant que i et j ne valent pas 0
        if (T[i][j]==0): #Sert a arrêter la boucle si on croise un score nul (0)
            i=0
        elif (i>0) and (j>0) and (T[i][j]==T[i-1][j-1]+msubst(seq1[i-1],seq2[j-1])): #Vérifie si le score est égal au score de la
            Align1=seq1[i-1]+Align1
            Align2=seq2[j-1]+Align2
            if (seq1[i-1]==seq2[j-1]):
                Align3="|"+Align3
            else:
                Align3="."+Align3
            i=i-1
            j=j-1
        elif (i>0) and (T[i][j]==T[i-1][j]+d): #Vérifie si le score est égal au score de la case de gauche
            Align1=seq1[i-1]+Align1
            Align2="-"+Align2
            Align3=" "+Align3
            i=i-1
        elif (j>0) and (T[i][j]==T[i][j-1]+d): #Vérifie si le score est égal au score de la case du haut
            Align1="-"+Align1
            Align2=seq2[j-1]+Align2
            Align3=" "+Align3
            j=j-1
    return(Align1,Align2,Align3)
#######Print le résultat du tracé arrière dans une page result.html####
#Pas eu le temps de détailler le code, désolé (résultat visible dans result.html)#
def printhtml(align1,align2,align3,nom1,nom2,score):
    global Y,N,d,nombreseq
    taille=len(align1)
    html_strdeb='''
    <!doctype html>
    <html>
    <head>
    <meta charset="UTF-8">
    <title>Local Alignment</title>
    <body>'''
    html_strdeb2='''
    # Program: Smith & Waterman
    # Autor : Rocher Vincent
    # Align_format: pair
    # Report_file: stdout
    ########################################

    #========================================
    #
```

```
            # Aligned_sequences: '''
            html_strdeb3='''
            # 1: '''
            html_strdeb4='''
            # 2: '''
            html_strdeb5='''
            # Gap_penalty: '''
            html_strdeb6='''
            # Match: '''
            html_strdeb7='''
            # Mismatch:'''
            html_strdeb8='''
            # Length: '''
            html_strdeb9='''
            # Score: '''
            html_strdeb10='''
            #
            #
            #====================================='''
            html_strfin='''
            </body>
            </html>'''
            with open("result.html","w") as html: #Ecrit dans le fichier result.html
                n=1
                html.write(html_strdeb)
                html.write('''<pre id="alignmentContent" xml:space="preserve">''')
                html.write(html_strdeb2)
                html.write(str(nombreseq))
                html.write(html_strdeb3)
                html.write(nom1)
                html.write(html_strdeb4)
                html.write(nom2)
                html.write(html_strdeb5)
                html.write(str(d))
                html.write(html_strdeb6)
                html.write(str(Y))
                html.write(html_strdeb7)
                html.write(str(N))
                html.write(html_strdeb8)
                html.write(str(taille))
                html.write(html_strdeb9)
                html.write(str(score))
                html.write(html_strdeb10)
                html.write('<br />')
                while (n<=len(align1)):
                    html.write('1: ')
                    html.write(align1[n:n+30].replace('','' '))
                    html.write('<br />')
                    html.write('   ')
                    html.write(align3[n:n+30].replace('','' '))
                    html.write('<br />')
                    html.write('2: ')
                    html.write(align2[n:n+30].replace('','' '))
                    html.write('<br />')
                    html.write('<br />')
                    n=n+31
                html.write('''</pre>''')
                html.write(html_strfin)
                print("le tracé arrière est visible dans le fichier result.html")
    #####Declaration variables et interaction utilisateur######
        noms1=[]#Liste qui récupère les noms des séquences issue du fichier 1
        noms2=[]#Liste qui récupère les noms des séquences issue du fichier 2
        sequences1=[]#Liste qui va contenir les séquences du premier fichier
        sequences2=[]#Liste qui va contenir les séquences du second fichier
```

```
####################INTERACTION UTILISATEUR#################
    print('''#########################################################''')
    print('''#                                                       #''')
    print('''#  Programme d'alignement local Smith & Waterman        #''')
    print('''#  Réalisé dans le cadre du projet Programmation        #''')
    print('''#                En Bio-Informatique                    #''')
    print('''#                Par ROCHER Vincent                     #''')
    print('#           Version de python : ',sys.version[0:5],'            #''')
    print('''#                                                       #''')
    print('''#########################################################''')
#Récupère la destination des deux fichiers et renvoi les séquences dans deux variables différentes :
    noms1,sequences1=lireFastaMul(input('Destination du fichier de la ou des séquence(s) 1: \n'))
    noms2,sequences2=lireFastaMul(input('Destination du fichier de la ou des séquence(s) 2: \n'))
    d=float(input('Score d\'indel : ')) #valeur du score en cas d'indel
    if d>0:
        d=-d
    Y=float(input('Score match : ')) #valeur du score en cas d'identité
    N=float(input('Score mismatch : ')) #valeur du score en cas de substitution
    M=float(input('Score mismatch même type de base purique/pyrimidique (possibilité de mettre même score que mismatch) : '))
    back=input('Voulez vous effectuer le tracé arrière ?(y/n) :')
#########Alignement de chaque séquence du fichier 1 avec chaque séquence du fichier 2#########
#Vérifie le nombre de séquences dans chaque fichier fasta, prend la plus petite série pour comparer
    if len(sequences1)<len(sequences2):
        nombreseq=len(sequences1)
    else:
        nombreseq=len(sequences2)


########LANCEMENT DES FONCTIONS############


    print('''Début de l'alignement ... Veuillez patienter.''')
    i=0
    scores=[]
    T=[]#Tableau qui va contenir le tableau de score
    while i<nombreseq: #Boucle qui va continuer tant qu'i ly a des sequences a aligner
        print('Comparaison de :\n',noms1[i],'\nAvec :\n',noms2[i],'\nEn cours...',sep='')
        T,score,imax,jmax=Align(sequences1[i],sequences2[i]) #Renvoi le score d'alignement
        print('Sequences (1_',i+1,':2_',i+1,') Aligned. Score : ',score,sep='')
        scores.append(score) #Ajoute le score dans une liste
        if back=='y':
            print('Alignement de :\n',noms1[i],'\nAvec :\n',noms2[i],'\nEn cours...',sep='')
            align1,align2,align3=backtrac(sequences1[i],sequences2[i],T,imax,jmax) #Effectue le tracé arrière si 'y'
            printhtml(align1,align2,align3,noms1[i],noms2[i],scores[i]) #Affiche le résultat dans une page web
        T=[]
        i=i+1

    if nombreseq>1: #Ecrit dans un fichier texte les scores si il y en a plus d'un
        with open("scores.txt", "w") as fichier:
            for element in scores:
                fichier.write(str(element))
                fichier.write(' ')
            print('fichier scores.txt produit.')
```