Kenyon Alexander

Gibbs Sampling :

L'idée et l'analyse de l'utilisation de Gibbs Sampling pour l'alignement de plusieurs séquences était publié en 1993 par Charles Lawrence et al., en *Science*. Dans leur papier, intitulé « Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment », les auteurs présentent le problème de l'alignement de plusieurs séquences. Ils écrivent, « A wealth of protein and DNA sequence data is being generated by genome projects and other sequencing efforts. A crucial barrier to deciphering these sequences and understanding the relations among them is the difficulty of detecting subtle local residue patterns common to multiple sequences. Such patterns frequently reflect similar molecular structures and biological properties »(Lawrence et al. 208). Autrement dit, le but du papier est de deviner comment trouver "subtle local residue patterns" parmi plusieurs séquences, pour qu'on puisse quantifier leurs similarités et éventuellement former des hypothèses sur leur fonctionne commune.

Description des Fichiers et des Fonctionnes

L'algorithme, écrit individuellement d'après Lawrence et al. 1993, a une structure comme la suivante :

Fichiers :

1. gibbs_main.c
   Un fichier qui contient la fonctionne `main()` de l'algorithme. Les plusieurs parties de cette fonctionne seraient décrites plus dans la section « Fonctionne Main »

2. gibbs_functions.h
   Un fichier qui contient toutes les fonctionnes secondaires de l'algorithme. Le fichier gibbs_main.c importe ce fichier.

Fonctionne Main :

1. Première partie : lignes 18-71 :
   - Importe le fichier Fasta par utiliser la fonctionne `readFasta()`
   - Initialiser le struct `unMotif` et le remplit avec les données du fichier Fasta.
   - Randomiser les motif débuts originaux de chaque séquence
   - Initialiser les chiffres pour la maximisation, comme le tableau `resultats`.

2. Deuxième partie : lignes 74-272 :

- Itérer pour un nombre d'itérations égal au constant `NUM_ITERATIONS`
- Chaque `x` itérations, soit `x = PHASE_SHIFT_FREQUENCY`, on utilise les résultats de la fonctionne `phaseShift()` au lieu de toutes les autres fonctionnes. On met à jour `F`, `F_global`, et `meilleur_F` comme nécessaire.
- Pour toute autre itération, on utilise le processus décrit dans Lawrence et al. 1993 pour décaler le début du motif dans une des séquences, choisie aléatoirement. Après avoir décalé, on calcule le score `F` du nouveau alignement et s'il est mieux que `F_global`, on change `F_global` d'être `F`. On utilise aussi un critère de Metropolis, pour qu'on puisse éviter d'être coincé dans les maximales locales de `F`. De plus, si `F` est plus que `meilleur_F`, on change `meilleur_F` d'être `F`, mais il n'y a pas de critère de Metropolis, donc `meilleur_F` est le vrai meilleur `F`. On garde les `tDebuts` de `meilleur_F` pour les résultats plus tard.

3. Troisième Partie : lignes 268-319
    - Les positions du meilleur alignement sont gardés dans le tableau `resultats`, à l'index `index_du_meilleur_F`.
    - Afficher toutes les informations importantes au terminal.
    - Grapher la progression de `F_global` par utiliser gnuPlot.

<u>Liste de Fonctionnes Secondaires :</u>

`tgSeq* readFasta :`

Cette fonctionne prends un fichier fasta et retourne un tableau du struct `tgSeq`, chaque index dans ce tableau représentant une séquence dans le fichier fasta.

`int** remplir_C:`

Cette fonctionne construit, remplit et retourne le tableau C, donné les tDebuts actuels de toutes les séquences.

`int* remplir_b:`

Cette fonctionne construit, remplit et retourne le tableau b (ou c(j)), donné les tDebuts actuels de toutes les séquences.

`float* remplir_rho :`

Cette fonctionne construit, remplit et retourne le tableau rho, donné les tDebuts actuels de toutes les séquences.

`float** remplir_q:`

Cette fonctionne construit, remplit et retourne le tableau q, donné les tDebuts actuels de toutes les séquences.

`float* remplir_p:`

Cette fonction construit, remplit et retourn le tableau p, donné les tDebuts actuels de toutes les séquences.

`float calculer_F :`

Cette fonctionne calcule le score F, donné tous les tableau `C`, `q`, et `p`.

`float* remplir_Q :`

Cette fonctionne calcule le tableau Qx, donné le tableau q.

`float* remplir_P:`

Cette fonctionne calcule le tableau Px, donné le tableau p.

`float* remplir_A:`

Cette fonctionne calcule et normalise le tableau Ax, donné les tableaux Qx et Px.

`float calculer_F_pour_phaseShift :`

Cette fonction calcule le score F pour un phase shift. Utilisée dans la fonctionne `phaseShift`.

`float* phaseShift:`

Cette fonction calcule les scores des nouvelles Ax pour chaque décalage de taille x, selon le constant `MAX_SHIFT`. Puis, elle choisit un nombre de décalage selon la distribution de probabilités de chaque Ax adjacent.

Résultats :

Résultats de TestSeqFasta2 :

TestSeqFasta2 :

```
>1
AAAAAAAAAAGWTSGTAAAAAAAAAAA
>2
LLLLLLLLLLGWTSGTLLLLLLLLLLL
>3
CCCCCCCCCCGWTSGTCCCCCCCCCCC
```

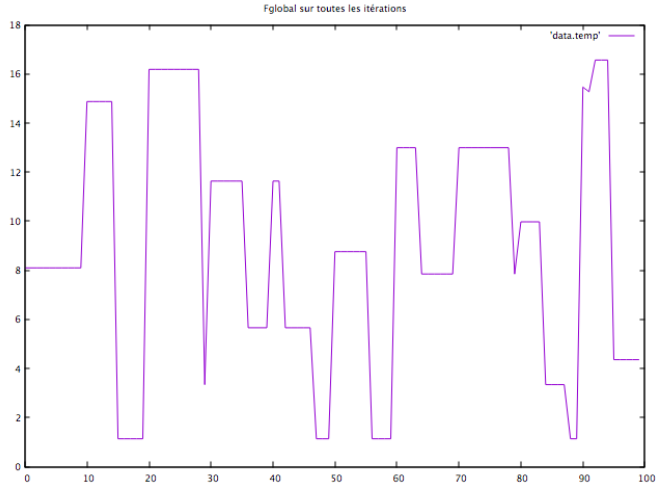
Résultats après quelque paramètres différents :

Paramètres constants :

- `LG_MOTIF = 6`
- `MAX_SHIFT = 5`

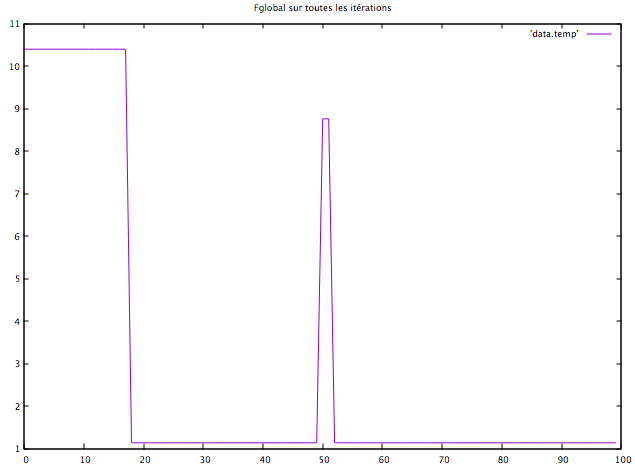
`NUM_ITERATIONS : 100, PHASE_SHIFT_FREQUENCY : 10`

```
Originals:
Seq1: 8, Seq2: 18, Seq3: 16
--------------------
Meilleur_F: 16.568964
F_global: 4.360360
index_de_meilleur_F: 92
Sequence 0 motif result: 6
AAAAGW
Sequence 1 motif result: 6
LLLLGW
Sequence 2 motif result: 7
CCCGWT
Runtime: 0.002232 seconds
```



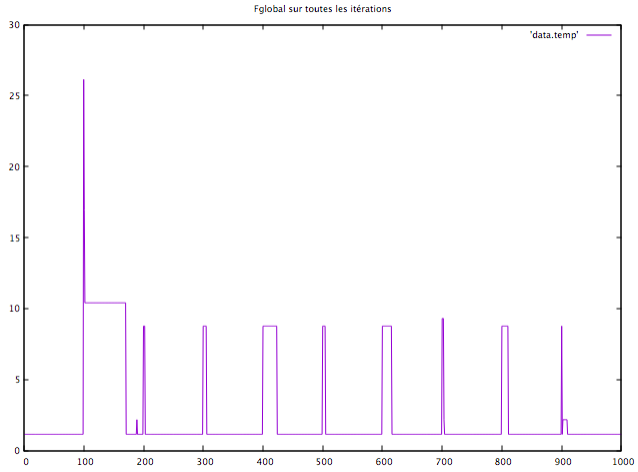
NUM_ITERATIONS : 100, PHASE_SHIFT_FREQUENCY : 50

```
Originals:
Seq1: 4, Seq2: 11, Seq3: 6
--------------------
Meilleur_F: 10.397618
F_global: 1.133810
index_de_meilleur_F: 0
Sequence 0 motif result: 4
AAAAAA
Sequence 1 motif result: 11
WTSGTL
Sequence 2 motif result: 6
CCCCGW
Runtime: 0.001846 seconds
```



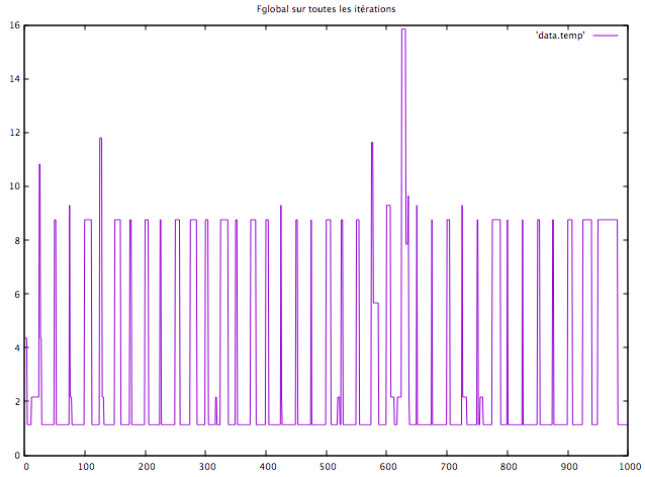
NUM_ITERATIONS : 1000, PHASE_SHIFT_FREQUENCY : 100

```
Originals:
Seq1: 3, Seq2: 17, Seq3: 4
--------------------
Meilleur_F: 26.135427
F_global: 1.133810
index_de_meilleur_F: 100
Sequence 0 motif result: 11
WTSGTA
Sequence 1 motif result: 13
SGTLLL
Sequence 2 motif result: 14
GTCCCC
Runtime: 0.012714 seconds
```



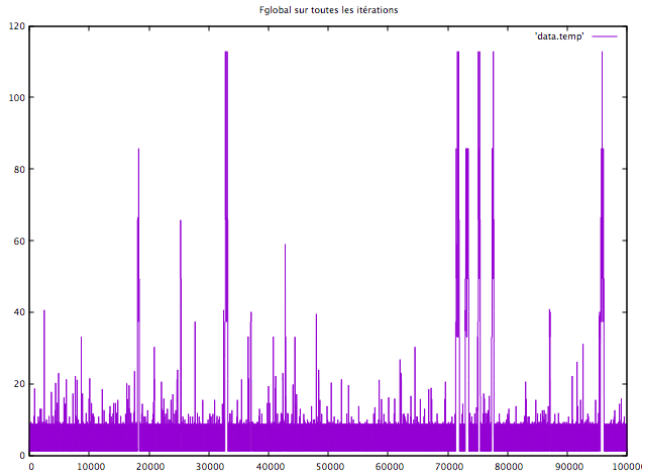
NUM_ITERATIONS : 1000, PHASE_SHIFT_FREQUENCY : 25

```
Originals:
Seq1: 14, Seq2: 3, Seq3: 6
--------------------
Meilleur_F: 15.852244
F_global: 1.133810
index_de_meilleur_F: 625
Sequence 0 motif result: 12
TSGTAA
Sequence 1 motif result: 15
TLLLLL
Sequence 2 motif result: 0
CCCCCC
Runtime: 0.014871 seconds
```



NUM_ITERATIONS : 100000, PHASE_SHIFT_FREQUENCY : 100

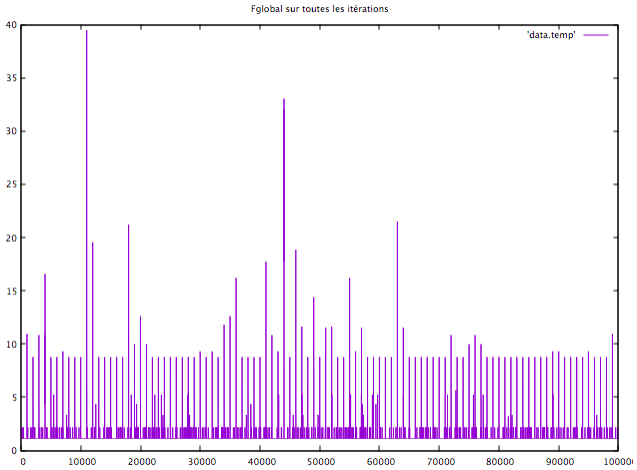
```
Originals:
Seq1: 6, Seq2: 18, Seq3: 5
--------------------
Meilleur_F: 112.883667
F_global: 1.133810
index_de_meilleur_F: 32800
Sequence 0 motif result: 10
GWTSGT
Sequence 1 motif result: 10
GWTSGT
Sequence 2 motif result: 10
GWTSGT
Runtime: 1.116344 seconds
```



NUM_ITERATIONS : 100000, PHASE_SHIFT_FREQUENCY : 1000

```
Originals:
Seq1: 15, Seq2: 2, Seq3: 9
--------------------
Meilleur_F: 39.477043
F_global: 1.133810
index_de_meilleur_F: 11000
Sequence 0 motif result: 13
SGTAAA
Sequence 1 motif result: 11
WTSGTL
Sequence 2 motif result: 13
SGTCCC
```

```
Runtime: 1.086291 seconds
```



Discussion:

Le fichier TestSeq2.fasta a présenté un problème pour le Gibbs Sampler jusqu'a le phaseShift était installée. Pour quelque raison, les extrémités de chaque séquence, avec les mêmes caractères, ont voulu dire que l'algorithme ne trouverait que les extrémités et pas le milieu, où est le motif cible. Les résultats de ces tests sur ce fichier montrent qu'on a envie de plus de phaseShifts—le seul test qui a trouvé le vrai motif était (100,000 ; 100). Ce qui est encourageant est que le test (100,000 ; 100) a trouvé le vrai motif presque tout le temps.

Resultats de TestSeqFasta3 :

TestSeqFasta3 :

```
>1
ACALCLCACCGWTSGTLLALCCLAAC
>2
LACCAAAACLGWTSGTALCCCCCAAA
>3
CALLAAAAACGWTSGTCCLLALCLLLLL
```

Résultats après quelque paramètres différents :

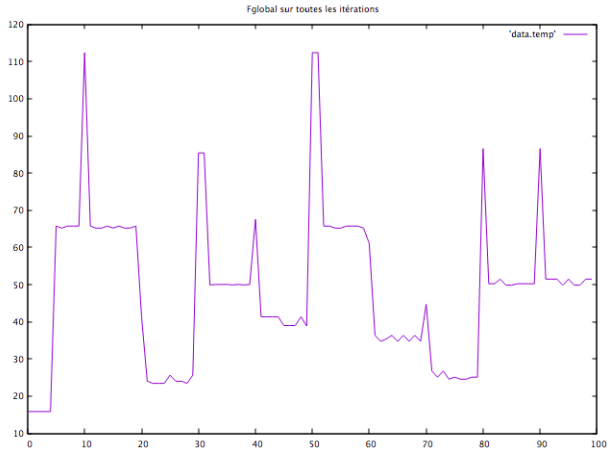
Paramètres constants :

- `LG_MOTIF = 6`
- `MAX_SHIFT = 5`

`NUM_ITERATIONS : 100, PHASE_SHIFT_FREQUENCY : 10`

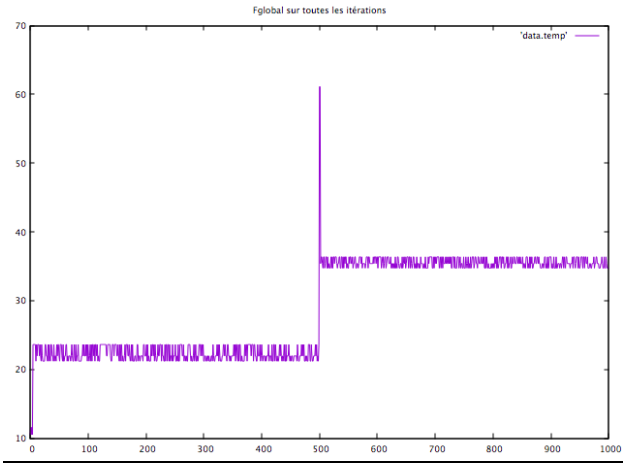
```
Originals:
Seq1: 17, Seq2: 18, Seq3: 10
--------------------
Meilleur_F: 112.387543
F_global: 51.477676
index_de_meilleur_F: 10
Sequence 0 motif result: 10
GWTSGT
Sequence 1 motif result: 10
GWTSGT
```

```
Sequence 2 motif result: 10
GWTSGT
Runtime: 0.003177 seconds
```



NUM_ITERATIONS : 1000, PHASE_SHIFT_FREQUENCY : 500

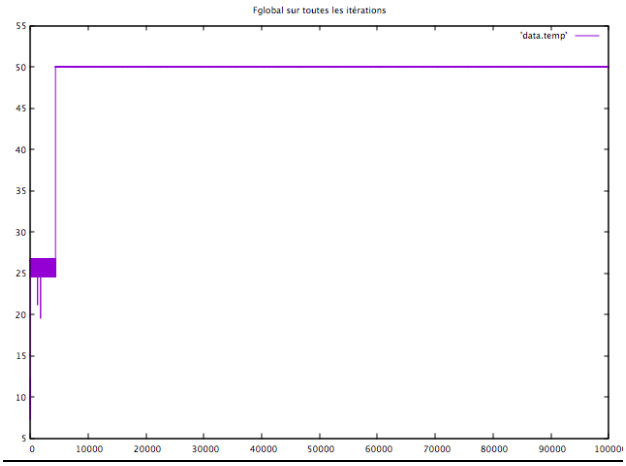
```
Originals:
Seq1: 17, Seq2: 19, Seq3: 6
--------------------
Meilleur_F: 61.111691
F_global: 34.719276
index_de_meilleur_F: 500
Sequence 0 motif result: 8
CCGWTS
Sequence 1 motif result: 8
CLGWTS
Sequence 2 motif result: 8
ACGWTS
Runtime: 0.013362 seconds
```



NUM_ITERATIONS : 100000, PHASE_SHIFT_FREQUENCY : 100001

```
Originals:
Seq1: 3, Seq2: 0, Seq3: 2
--------------------
Meilleur_F: 50.061409
F_global: 50.020618
index_de_meilleur_F: 4394
Sequence 0 motif result: 11
```

```
WTSGTL
Sequence 1 motif result: 11
WTSGTA
Sequence 2 motif result: 11
WTSGTC
Runtime: 1.085689 seconds
```



Discussion:

L'importance du phaseShift est evident dans l'analyse des résultats de TestSeq3.fasta. Le test (100 ; 10) était beaucoup plus précis que le test (100,000 ; 100,001), juste à cause des phaseShifts. Dans le graphe de (100,000 ; 100,001), on peut voir exactement le même problème que celui décrit dans Lawrence et al. 1993—l'algorithme devient coincé dans un maximum local.

Resultats de lipocalin.fst :

lipocalin.fst :

```
>ICYA_MANSE|uniprot|P00305|ICYA_MANSE Insecticyanin-A;
GDIFYPGYCPDVKPVNDFDLSAFAGAWHEIAKLPLENENQGKCTIAEYKYDGKKASVY
NSFVSNGVKEYMEGDLEIAPDAKYTKQGKYVMTFKFGQRVVNLVPWVLATDYKNYAIN
YNCDYHPDKKAHSIHAWILSKSKVLEGNTKEVVDNVLKTFSHLIDASKFISNDFSEAA
CQYSTTYSLTGPDRH
>LACB_BOVIN|uniprot|P02754|LACB_BOVIN Beta-lactoglobulin;
MKCLLLALALTCGAQALIVTQTMKGLDIQKVAGTWYSLAMAASDISLLDAQSAPLRVY
VEELKPTPEGDLEILLQKWENGECAQKKIIAEKTKIPAVFKIDALNENKVLVLDTDYK
KYLLFCMENSAEPEQSLACQCLVRTPEVDDEALEKFDKALKALPMHIRLSFNPTQLEE
QCHI
>BBP_PIEBR|uniprot|P09464|BBP_PIEBR Bilin-binding protein;
MQYLIVLALVAAASANVYHDGACPEVKPVDNFDWSNYHGKWWEVAKYPNSVEKYGKCG
WAEYTPEGKSVKVSNYHVIHGKEYFIEGTAYPVGDSKIGKIYHKLTYGGVTKENVFNV
LSTDNKNYIIGYYCKYDEDKKGHQDFVWVLSRSKVLTGEAKTAVENYLIGSPVVDSQK
LVYSDFSEAACKVNN
>RET4_BOVIN|uniprot|P18902|RET4_BOVIN Retinol-binding
protein 4;
ERDCRVSSFRVKENFDKARFAGTWYAMAKKDPEGLFLQDNIVAEFSVDENGHMSATAK
GRVRLLNNWDVCADMVGTFTDTEDPAKFKMKYWGVASFLQKGNDDHWIIDTDYETFAV
QYSCRLLNLDGTCADSYSFVFARDPSGFSPEVQKIVRQRQEELCLARQYRLIPHNGYC
DGKSERNIL
```

>MUP2_MOUSE|uniprot|P11589|MUP2_MOUSE Major urinary protein 2;
MKMLLLLCLGLTLVCVHAEEASSTGRNFNVEKINGEWHTIILASDKREKIEDNGNFRL
FLEQIHVLEKSLVLKFHTVRDEECSELSMVADKTEKAGEYSVTYDGFNTFTIPKTDYD
NFLMAHLINEKDGETFQLMGLYGREPDLSSDIKERFAKLCEEHGILRENIIDLSNANR
CLQARE

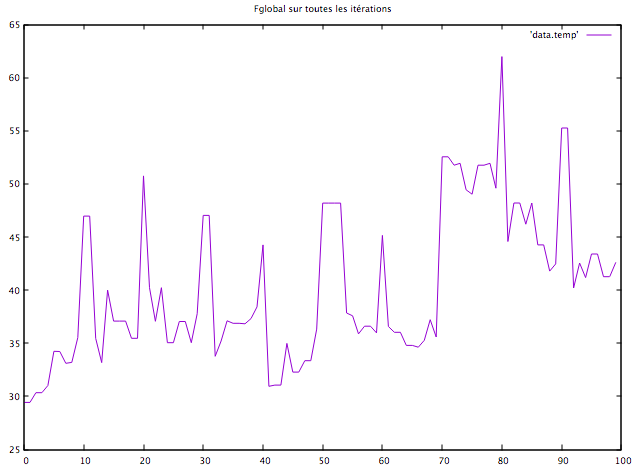
Résultats après quelque paramètres différents :

Paramètres constants :

- LG_MOTIF = 6
- MAX_SHIFT = 5

NUM_ITERATIONS : 100, PHASE_SHIFT_FREQUENCY : 10

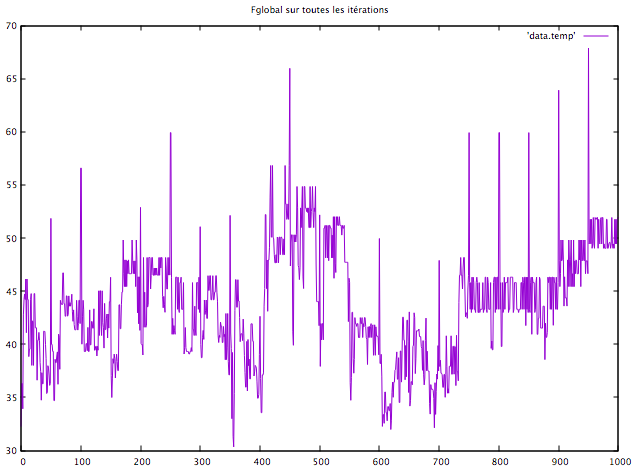
```
Originals:
Seq1: 141, Seq2: 32, Seq3: 17
--------------------
Meilleur_F: 61.989418
F_global: 42.572655
index_de_meilleur_F: 80
Sequence 0 motif result: 103
WVLATD
Sequence 1 motif result: 108
LVLDTD
Sequence 2 motif result: 114
NVLSTD
Sequence 3 motif result: 104
WIIDTD
Sequence 4 motif result: 108
TIPKTD
Runtime: 0.005324 seconds
```



NUM_ITERATIONS : 1000, PHASE_SHIFT_FREQUENCY : 50

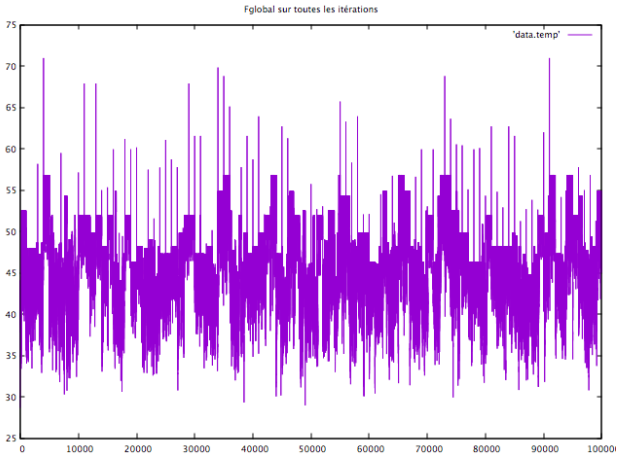
```
Originals:
Seq1: 147, Seq2: 98, Seq3: 89
--------------------
Meilleur_F: 67.887985
F_global: 51.938465
index_de_meilleur_F: 950
Sequence 0 motif result: 107
TDYKNY
Sequence 1 motif result: 112
```

```
TDYKKY
Sequence 2 motif result: 118
TDNKNY
Sequence 3 motif result: 108
TDYETF
Sequence 4 motif result: 112
TDYDNF
Runtime: 0.036540 seconds
```



NUM_ITERATIONS : 100,000, PHASE_SHIFT_FREQUENCY : 1000

```
Originals:
Seq1: 96, Seq2: 73, Seq3: 160
--------------------
Meilleur_F: 70.966370
F_global: 52.397366
index_de_meilleur_F: 4000
Sequence 0 motif result: 22
FAGAWH
Sequence 1 motif result: 30
VAGTWY
Sequence 2 motif result: 36
YHGKWW
Sequence 3 motif result: 19
FAGTWY
Sequence 4 motif result: 32
INGEWH
Runtime: 3.167069 seconds
```



Discussion:

Le fichier lipocalin.fst est beaucoup plus compliqué que les autres, et en fait il a deux motifs qui peuvent être trouvés. Pourtant, il est intéressant de voir le meilleur_F, qui croisse avec la croissance du nombre d'itérations. Les résultats de l'analyse sur lipocalin.fst nous montrent qu'en réalité, il n'y a pas de solution absolue.