Rapport de stage de DEA effectué sous la direction de Mireille Régnier
Projet ALGORITHMES INRIA Rocquencourt

Etude de Problèmes combinatoires liés à l'Analyse du Génome : Séquençage et Polymorphisme.

Frédéric Giroire

DEA Algorithmique
http://ultralix.polytechnique.fr/Labo/Jean-Marc.
Steyaert/Xpres.html
2002-2003

Contents

1 Informations générales

Mon stage de DEA a été effectué sous la direction de Mireille Régnier au sein du projet ALGORITHMES[1] de l'INRIA Rocquencourt du 20 mars au 30 septembre. Les thèmes de prédilection du projet ALGO sont la conception et l'analyse d'algorithmes, le calcul formel, l'analyse combinatoire et l'asymptotique.

Au cours de mon stage, nous nous sommes tout d'abord intéréssés à parcourir un large domaine de la recherche en bioinformatique. L'étude du travail des membres de l'équipe, au premier rang desquels Mireille Régnier, [15][20], Mathias Vandenbogaert [20] et Edouard Dolley [21] que je tiens tous à remercier pour leur grande disponibilité et leur gentillesse m'a donné un premier exemple d'application des techniques combinatoires aux problèmes biologiques. En particulier l'étude des mots sous-représentés dans l'ADN est un terrain propice d'emploi des méthodes enseignées dans mon DEA, le DEA d'algorithmique et plus spécifiquement lors des cours de la filière combinatoire que j'ai suivie. Cette imprégnation dans le domaine s'est poursuivie au cous de différentes lectures et présentations comme celles du Trimestre "Mathematics, Informatics and Genomics" de l'Institut Henri Poincaré [2] du 14 Avril au 11 Juillet 2003 ou la journée de Bioinformatique de l'Ecole de Biologie Industrielle de Cergy[3]. Une bonne partie de mon stage jusqu'à présent a été utilisée à essayer de me doter d'une bonne culture bioinformatique sur des sujets comme la statistique des motifs dans l'ADN [20], les arbres phylogéniques [22], les transferts de gènes [12] [13] [14], le polymorphisme entre individus [3][22], le séquencage [1][4][2] et à l'approfondissement de ma connaissance des théories combinatoires, comme l'utilisation des séries génératrices [19].

C'est au cours de l'étude d'articles sur des problèmes liés au séquencage [1][4] et lors de discussions, notamment avec Michael S Waterman[4] de University of Southern California[5] et Sorin Istrail de Celera Genomics[6] au sujet des problèmes de séquencage et de polymorphisme, que nous nous sommes orientés vers le sujet présenté dans ce rapport. L'analyse des méthodes de comptage statistique utilisées nous a persuadé du fait que l'utilisation de méthodologies développées dans le projet ALGO s'appuyant notamment sur l'emploi des séries génératrices pouvait grandement améliorer la précision des calculs et l'efficacité des algorithmes. D'autre part, les discussions nous ont montré que cette nouvelle méthode d'analyse pouvait amener des progrès dans des domaines cruciaux comme la détection des répétitions dans les différents génomes lors du séquencage et la détection de polymorphisme qui est une des orientations majeures de la bioinformatique pour ces prochaines années.

[1] `http://algo.inria.fr/`
[2] `http://www.ihp.jussieu.fr/`
[3] `http://www.ebi-edu.com/`
[4] `http://www-hto.usc.edu/people/Waterman.html`
[5] `http://www.usc.edu/`
[6] `http://www.celera.com/`

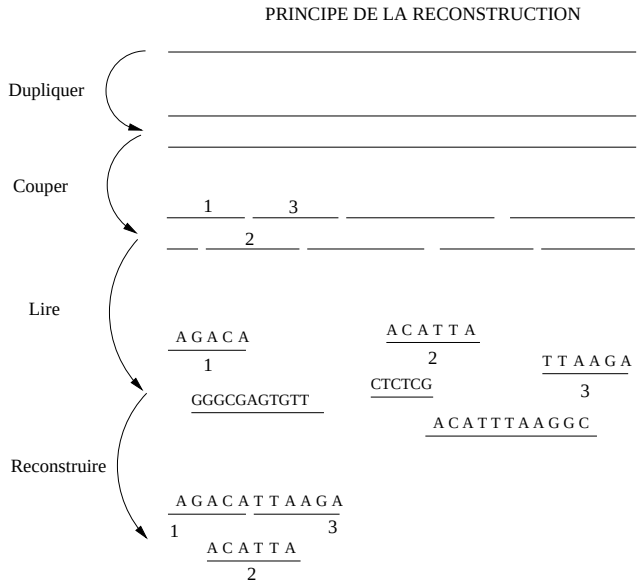

Figure 1: Principes de la reconstruction

2 Le Séquençage et Introduction des problèmes

Dans cette section, nous présentons tout d'abord les principes généraux du séquençage (section 2.1), puis les différents problèmes qu'il soulève (section 2.2). Ensuite nous nous attarderons un peu plus en particulier sur les deux problèmes que nous traitons dans ce rapport : celui des répétitions (section 2.3) et celui de la découverte du polymorphisme (section 2.4).

2.1 Le Séquençage

La découverte de l'ADN par Crick et Watson en 1953 a été une avancée décisive pour la compréhension du vivant. Mais tout aussi importante qu'elle soit, cette découverte n'a été qu'un pas en avant : l'alphabet de la vie est maintenant connu, mais quels en sont les mots et leur grammaire ? Pour pouvoir répondre à cette question, une étape nécessaire est de connaitre la séquence génétique des êtres vivants et en particulier celle, qui pour des raisons évidentes nous importe le plus : la séquence génétique de l'être humain. Ce programme, qui a aboutit au bout de 50 ans de maturation au décryptage presque complet du génome humain, c'est le séquençage.

L'ADN est formée de deux brins complementaires. Chaque brin est une succession de nucléotides ou bases, organisés en double hélice. Il y a quatre types de nucléotides : A pour adénine, T pour, C pour cytosine, G pour guanine. La complémentarité entre les deux brins s'effectue selon la règle : $A \leftrightarrow T$ et $C \leftrightarrow G$. Le but du séquençage est de déterminer la succession de bases d'un génome entier. Par exemple, l'ADN humain

est constitué d'environ 3 milliards de paires de bases. Or les techniques existantes pour lire ces séquences biologiques utilisent des machines qui ne peuvent décoder que de petits bouts de $500 - 700$ paires de bases. Il va donc falloir découper le génome que l'on veut séquencer en de tels petits bouts. Le problème qui surgit ici est qu'il n'est pas possible de couper bout par bout l'ADN dans l'ordre, puis de le séquencer dans le même ordre. Les méthodes de découpage s'apparentent plutôt à la plongée d'un plat de spaghettis dans un gigantesque mixeur. Plus précisément, quand on a séquencé un bout d'ADN, on ne sait pas quelle etait sa position dans l'ADN de départ. La solution trouvée à ce problème fut d'abandonner le séquençage. Somme toute, on a bien vécu jusqu'ici sans connaitre l'ordre des bases du génome. Ensuite des esprits un peu chagrins, aidés de la figure 1, ont suggéré de cloner le génome plusieurs fois, de couper ces copies aléatoirement en petits bouts lisibles et ensuite de les réassembler en utilisant le recouvrement des différentes séquences.

```
Ainsi ACGGTAAT et
          TAATCCTG
```

auraient de bonnes chances d'avoir des positions voisines dans le génome.

Deux grands groupes se sont lancés dans cette grande aventure biologique : l'entreprise Celera Genomics et le consortium international The Human Genome Project [7]. Plusieurs méthodes de séquençage ont été inventées. Il faut distinguer quatre phases principales : dupliquer, couper, lire, reconstruire (voir figure 1). C'est la façon de combiner et d'effectuer celles-ci qui va différencier les différentes méthodes entre elles. Celle que nous allons étudier a été la plus utilisée au cours de la longue histoire du séquençage : le séquençage du génome entier par *shotgun*. De quoi s'agit-il ?

La figure 2 nous en donne les lignes directrices. Les deux premières phases s'entremêlent. De la même manière que l'on ne peut pas lire une grande séquence d'ADN, on ne peut pas la recopier. Cependant, si le séquencage direct se fait pour une taille maximum d'environ 700 paires de base, le clonage permet, selon la technique employée, de recopier jusqu'à 50000 paires de bases. Il faut donc d'abord couper le génome en bouts de taille moyenne, disons plusieurs dizaines de milliers de paires de base, à l'aide d'enzymes de restriction. Ces bouts de taille moyenne sont ensuite clonés par différentes méthodes. Ensuite ils sont redécoupés selon d'autres méthodes, comme l'envoi d'ondes, en segments d'environ 500 paires de bases. On se retrouve avec une gigantesque soupe de dizaines de millions de morceaux d'ADN que l'ont a enfin pu lire, mais qu'il faut maintenant réassembler. La quatrième phase commence ici. Tous les bouts seront comparés deux à deux...

On sait maintenant que les bases 10830 a 10840 du 7e chromosome de la bactérie E. Coli sont AATCGCTTAAAC. Et on peut acheter un T-shirt avec la séquence d'un chromosome de notre amie la vache. Mais je vais ici vous décevoir : ce n'est que la belle partie de l'histoire. Même le séquençage du génome humain annoncé par les média généralistes n'est pas complet. Il reste certains problèmes que nous allons voir maintenant.

[7] `http://www.ornl.gov/TechResources/Human_Genome/home.html`

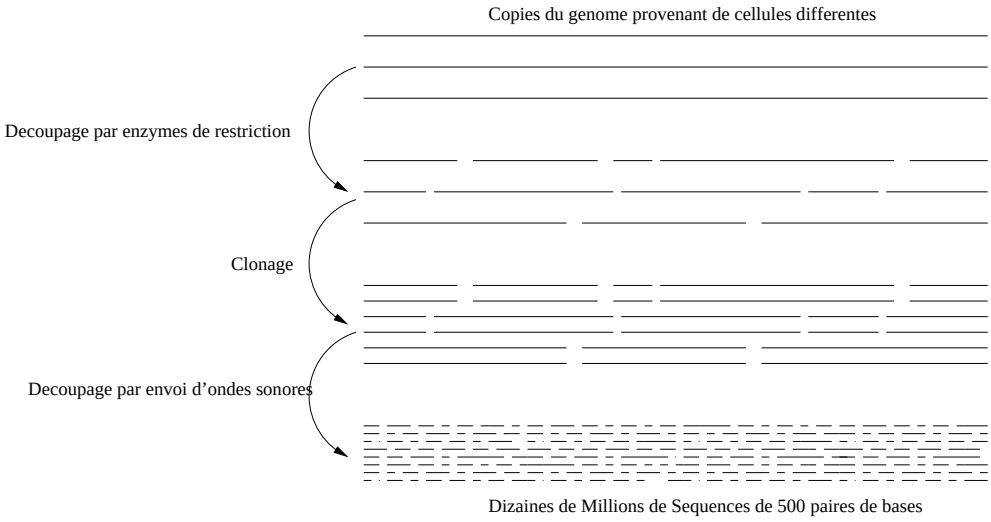

Figure 2: Séquençage par shotgun du génome entier

2.2 Les problèmes de Séquençage

La première difficulté rencontrée est, comme nous l'avons vu dans la section 2.1, l'échelle de plusieurs milliards de bases des données manipulées. Les différentes méthodes de séquençage répondent à ce problème. Mais leur lourdeur va rendre problématique l'algorithmique du séquencage en raison de l'apparition d'erreurs sur les séquences au cours du processus. Plusieurs phénomènes sont à la source de ces erreurs :

- au cours des clonages, de l'ADN étranger est introduit.
- Des mutations entre les différents ADN de départ peuvent exister.
- La machine à séquencer fait assez fréquemment des erreurs de lecture.

Dans la suite, nous oublierons les deux premiers types d'erreurs pour ne considérer que les erreurs de lecture que nous qualifierons d'erreurs de séquençage. Nos modèles considèreront souvent un taux d'erreur qui sera typiquement entre 0.1 et quelques pourcents.

En raison de ces erreurs différents types de difficultés surviennent :

- les procédures d'alignement sont moins précises.
- la précision de la séquence reconstituée n'est pas très grande.
- $\rightarrow$ la recherche de l'haplotype d'un individu par séquencage devient plus difficile. En effet il va s'agir de distinguer les SNPs (Single Nucleotid Polymor-

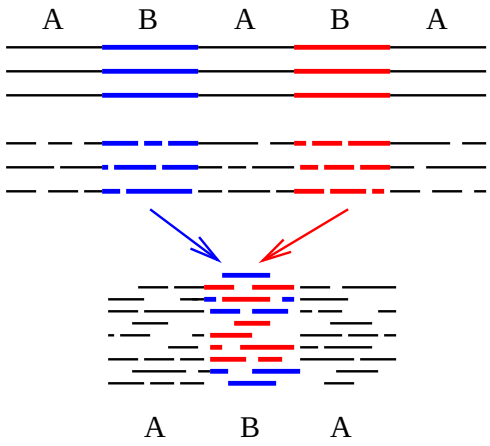

Figure 3: Répétitions et Séquençage par shotgun

phism) de simples erreurs de séquencage. Nous parlerons plus largement de ce problème en sections 2.4 et 8.

- En particulier, le problème que nous allons étudier : comment distinguer les répétitions presque identiques des erreurs.

2.3 Répétitions et Séquençage

Motivation : Bien que souvent modélisée par des modèles de Bernoulli ou de Markov, l'ADN est très loin d'avoir une composition aléatoire. En particulier, elle est très répétitive. Plus de 30 % du génome est constitué de répétitions qui peuvent prendre plusieurs formes : répétitions non dupliquées, satellites, tandem, transposons, répétitions dispersées, répétitions en segment, chromosomes dupliqués, ...

Par exemple, deux séquences dont la taille peut aller de quelques bases à plusieurs dizaines de base peuvent se répéter l'une à la suite de l'autre jusqu'à plusieurs centaines de fois (cas de la figure 3).De telles configurations sont un défi pour les differents algorithmes de séquençage. En effet des séquences venant des différentes parties répétées vont naturellement être rapprochées en raison du très fort alignement qu'elles auront les unes avec les autres. Dès lors comment savoir par exemple dans le cas d'une répétition en tandem combien de fois les séquences se répètent ?

De tels phénomènes peuvent aussi entrainer des inversions dans la séquence reconstituée : un séquence RXRYR peut devenir la séquence RYRXR. Quand les zones répétées ne sont pas trop grandes (moins de quelques dizaines de milliers de paires de bases), on utilise de grandes séquences d'ADN que l'on séquence aux deux bouts. Il s'agit de "mate pairs". Ceux-ci donnent des indications de positions relatives de bouts du génome par rapport à d'autres. Quand au contraire ces zones sont trop grandes, il est impossible d'avoir recours à cette méthode. Nous nous placerons dans ce cas pour notre étude.

2.4 Polymorphisme et Séquençage

Quand on dit que le séquençage du génome humain a été effectué à 99,9%, cela signifie que les données génétiques communes à l'espèce humaine sont maintenant à disposition. Mais les individus d'une même espèce ne sont pas tous identiques. Ces différences sont codées dans leur génome. Et en fait si la lecture du génome humain est une avancée déterminante dans la voie de la connaissance du vivant, ce n'est avant tout qu'une étape pour déterminer l'étendue du polymorphisme génétique au sein de l'espèce humaine. En effet ce sera grâce à cette connaissance que l'on pourra comprendre pourquoi certains individus résistent mieux à certaines maladies qu'à d'autres par exemple, et ainsi trouver des principes de développement de médicaments. Ainsi l'étude du polymorphisme est le problème chaud du moment et des prochaines années en bioinformatique.

Mais c'est un problème coûteux : faire le séquençage du génome d'un individu est déjà très long et très coûteux. Dans le cas de la recherche des polymorphismes, il va s'agir du séquençage de plusieurs individus. Pour diminuer les coûts, l'idée est d'utiliser des données déjà produites et d'utiliser plus efficacement celles à venir. Il va s'agir de rechercher le polymorphisme à partir des données du séquençage [3].

Le principe est le suivant : l'ADN séquencé provient des paires de chromosomes d'un individu. Un des chromosomes vient de la mère et l'autre du père. Ces deux versions diffèrent en quelques endroits. On appelle SNP (pour Single Nucleotide Polymorphisme) une position du génome d'une espce pour laquelle différents individus peuvent avoir des bases différentes. Pendant le multi-alignement des séquences, au lieu de distinguer entre erreurs de séquençage et différences réelles entre des zones répétés pour séparer les répétitions, on peut essayer, selon le même principe de distinguer entre erreurs de séquençage et SNPs pour séparer le génome venant du père de celui venant de la mére. On pourrait ainsi construire une sous-carte des emplacements des SNPs à partir de séquençages déjà produits.

3 Les méthodes statistiques pour traiter les répétitions

Les prochaines sections (3, 4, 5) traiteront spécifiquement le problème des répétitions. Dans cette section, après avoir donné des principes généraux sur les méthodes statistiques (3.1), nous allons exposer une méthode particulière (3.2) et présenter les formules et résultats qu'elle peut obtenir (3.3).

3.1 Généralité et Principes

Différentes méthodes ont été envisagées et utilisées pour traiter ce problème de répétitions, comme la cartographie du génome ou l'utilisation de "mate pairs". La classe de solutions que nous allons étudier est celle des méthodes statistiques.

Ces méthodes utilisent la constatation que les zones répétitives du génome ne sont pas en fait absolument identiques. En raison de phénomènes de mutation au cours de l'évolution, les zones répliquées se distinguent par des différences de bases qui en représentent quelques pourcents. La détection des répétitions va donc se faire en essayant de distinguer ces différences réelles entre les reads d'avec celles générées accidentellement par le processus de séquençage. Pour cela, on construit des alignements multiples de reads constitués de tous les reads qui se recoupent de façon à utiliser autant d'information que possible quand on analyse une région. L'idée essentielle ensuite est que les différences réelles entre les répétitions peuvent être distinguées des erreurs de séquençage par le fait que les erreurs sont distribuées aléatoirement contrairement aux différences réelles.

Ce principe a été utilisé dans des travaux récents comme [1] [2] [4] [6] [7] [5]. Les méthodes ensuite diffèrent par les algorithmes utilisés pour prendre ces analyses statistiques et aussi par les choix pour mener à bien celles-ci. Nous avons choisi d'analyser plus en profondeur celle employée M. Tammi, E. Arner, T Britton et B. Andersson dans leur article "Separation of nearly identical repeats in shotgun assemblies using defined nucleotide positions, DNPs" [1] car elle est actuellement la plus accomplie et donne des résultats de simulation satisfaisant.

3.2 Méthode et Algorithme de Tammi and all

Dans cette sous-section, il sera souvent utile de se référer à la figure 4 pour se faire une image de la méthode et des différentes notions introduites. La *base de consensus* d'une colonne est définie ici comme la base la plus fréquente de la colonne. Les *colonnes candidates* sont celles où l'on observe des déviations du consensus et où ces déviations contiennent au moins D_{min} bases du même type, appelé *type de base candidat*. Ces derniers sont candidats à venir d'une autre répétition du génome.

En raison du taux de couverture, du taux élévé d'erreurs de séquençage, les distributions calculées sur une seule colonne candidate ne vont pas permettre de distinguer sûrement entre erreurs et différences . Par contre cela sera possible en considérant les coincidences entre les déviations entre deux colonnes candidates (voir figure 5). Remarquez que cela implique qu'au moins deux différences soient présentes dans un read pour pouvoir être détectées avec cette méthode. Mais c'est une contrainte nécessaire et réaliste en pratique (1% de taux de mutation) et on la retrouve dans les autres travaux

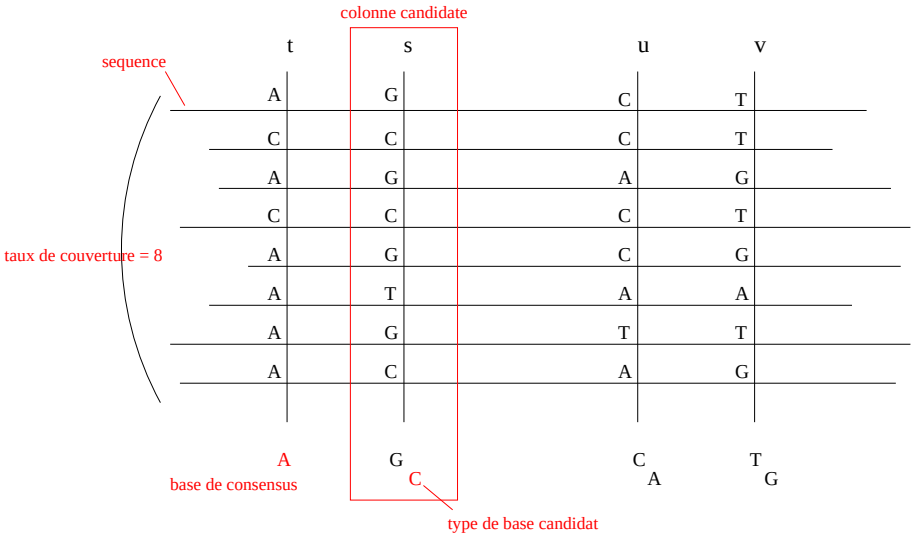

Figure 4: Mots clés : multi-alignement, séquences, couverture, base de consensus, colonnes candidates, type de base candidat.

comme, par exemple, [4]. L'algo va donc repérer pour chaque multi-alignement les colonnes candidates et quand il en trouve deux il calcule la probabilité d'observer par chance leurs coincidences. Si elle est en dessous d'en certain seuil, on considère que l'on est en présence de répétitions à séparer.

Les auteurs évaluent les performances de leur algorithme en simulant des séquençages par shotgun de différentes séquences aléatoires contenant des répétitions en tandem de longueur typiquement 2000 paires de bases. Leurs métriques sont le taux d'erreurs (faux positifs) et la sensibilité (taux de vrais positifs repérés).

3.3 Définitions et Formules

Considérons deux positions fixées, u et v, dans un multi-alignement de k séquences recouvrant à la fois u et v. Notons la base en position u (resp. v) de la j-ème séquence $a_{u,j}$ (resp. $a_{v,j}$). Soit $I_{u,j}$ la variable indicatrice de l'événement que la base en position u du read j dévie du consensus et est d'un type de base candidat spécifique. Les probabilités p_j ($I_{u,j} = 1$) et q_j ($I_{v,j} = 1$) sont calculées à partir des valeur de qualité Phred [6]. Le nombre total de déviations en position u est noté $N_u = \sum_{j=1}^{k} I_{u,j}$. $I_{v,j}$ et N_v sont définis de façon similaire. Soit $I_j = I_{u,j} I_{v,j}$ la variable indicatrice d'une coincidence pour la j-ème séquence, c'est-à-dire que les deux positions dévient du consensus pour la séquence j. Enfin le nombre total de coincidences est noté :

$$C = \sum_{j=1}^{k} I_j = \sum_{j=1}^{k} I_{u,j} I_{v,j}.$$

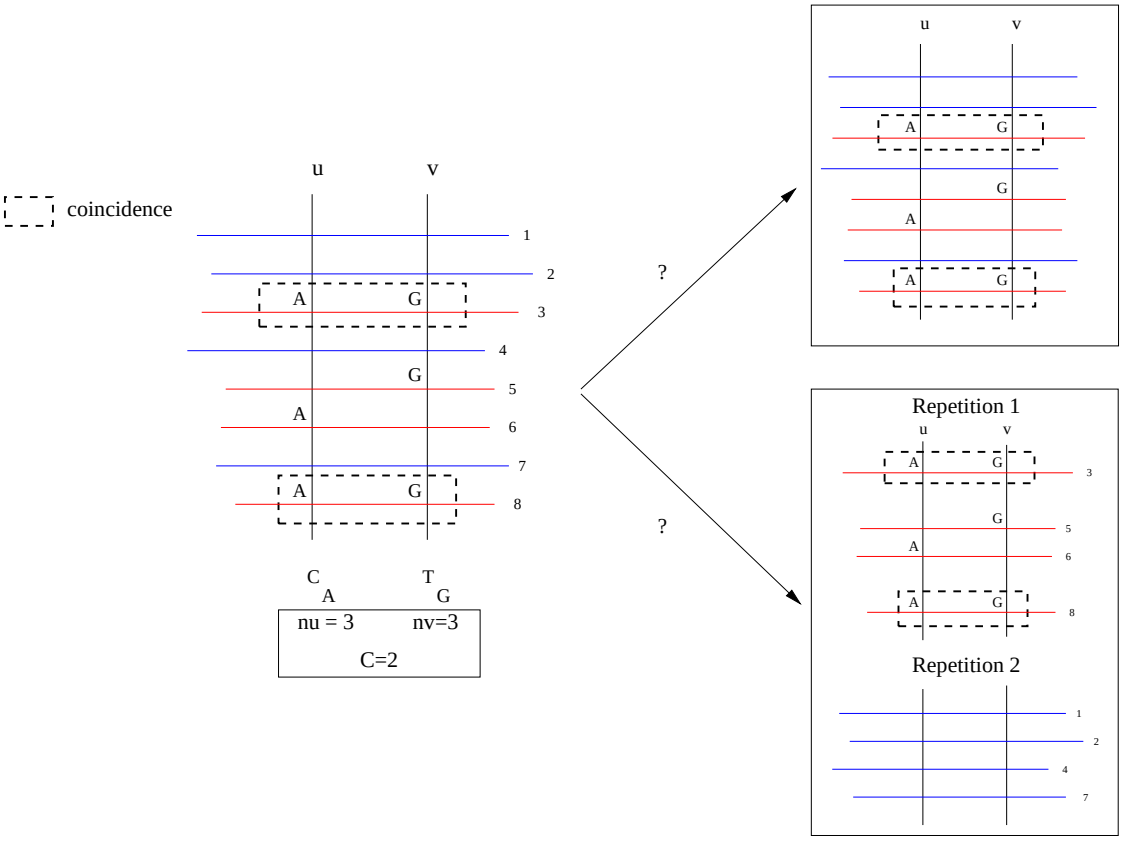

Figure 5: Étude des coincidences : erreur de séquençage ou différences réelles ?

Les auteurs donnent un test pour savoir si les coincidences arrivent par chance et non pas systématiquement, ce dernier cas indiquant que les positions u et v contiennent des informations sur des différences entre zones répétées et non pas juste des erreurs de séquençage. Ils calculent la distribution approchée de C sous l'hypothèse que les déviations par rapport au consensus apparaissent indépendamment. En effet, le principe de base de ces calculs est de se placer sous des hypothèses d'indépendance, de calculer la probabilité de se retrouver dans une situation particulière et de déterminer si elle est suffisamment faible pour rejeter l'hypothèse d'indépendance.

Les auteurs calculent la distribution de C sachant $N_u = n_u$ et $N_v = n_v$, soit sachant le nombre de déviations par rapport au consensus aux deux positions. Ils se placent dans deux cas :

1 Cas uniforme: $p_1 = p_2 = ... = p_k = q_1 = q_2 = ... = q_k$

2 Cas général biaisé où les p_i et q_i sont différents

3.3.1 Cas uniforme

Pour le cas où les p_j et q_j sont tous égaux entre eux, on peut obtenir une formule par une analyse standarde de comptage.

Proposition 3.1

$$P(C = x|N_u = n_u, N_v = n_v) = \frac{\binom{n_v}{x}\binom{k-n_u}{n_u-x}}{\binom{k}{n_u}}, 0 \leq x \leq n_v, 0 \leq n_u - x \leq k - n_v$$

Preuve :
pb : placer n_u déviations en colonne u et n_v en colonne v avec x coincidences.
nb total de placements $= (kn_u).(kn_v)$
Pour un placement des n_v déviations : $(n_v x).(k - n_v n_u - x)$
donc en tout : nb de placements avec x coincidences : $(kn_v).(n_v x).(k - n_v n_u - x)$
D'où

$$P(C = x|N_u = n_u, N_v = n_v) = \frac{(kn_v)(n_v x)(k - n_u n_u - x)}{(kn_u)(kn_v)}, 0 \leq x \leq n_v, 0 \leq n_u - x \leq k - n_v$$

•

3.3.2 Cas général biaisé

Dans le cas où les p_i et q_i sont différents, les auteurs approximent la distribution de C sachant $N_u = n_u$ et $N_v = n_v$ par une distribution de Poisson de paramètre et d'espérance :

$$E(C|N_u = n_u, N_v = n_v) \approx \sum_{j=1}^{k} \frac{n_u p_j}{n_u p_j + \lambda_u^{(j)}(1 - p_j)} \times \frac{n_v q_j}{n_v q_j + \lambda_u^{(j)}(1 - q_j)}$$

4 Analyse de l'Approximation Poissonienne

Dans cette section, nous analyserons tout d'abord les fondements théoriques de l'approximation de Poisson présentée en section 3.3 (4.1) et ensuite nous évaluerons les performances obtenues en l'utilisant (4.2).

4.1 Analyse théorique

Les auteurs présentent leur analyse statistique sans la justifier autrement que par des généralités et emploi de mots clés comme par exemple "approximation par distribution de Poisson". Il n'est ni question de recherche du domaine de validité, ni énoncé d'un quelconque théorème mais plutôt d'un appel à l'indulgence des formules mathématiques que nous ne laisserons pas passer ;o). Nous allons discuter trois points :

- approximation de $P(C = x)$
- approximation de $P(C = x|N_u = n_u N_v = n_v)$
- calcul de l'espérance

4.1.1 approximation de $P(C = x)$

L'approximation de la distribution de $P(C = x)$ par une loi de Poisson est justifiée par le résultat mathématique quúne distribution binomiale est bien approximé par une loi de Poisson sous certaines conditions.

[[mettre le théorème exact avec les conditions]]

La probabilité d'avoir un coindidence sur une séquençagej est $p_j q_j$ et celle de ne pas en avoir 1 - $p_j q_j$. En prenant pour hypothèse que les p_j et q_j ne sont pas très différents les uns des autres, on peut considérer être dans une situation modélisée par une loi binomiale de paramètres p^2 et k.

Ici les valeurs de p et q sont bien supposées petites (typiquement de l'ordre de 0,1 ou moins).

Donc [[sur certains domaines... a definir]]

On peut raisonnablement supposer que la distribution $P(C = x)$ peut être approximée par une loi de Poisson.

4.1.2 approximation de $P(C = x|N_u = n_u N_v = n_v)$

"On the other hand, the conditioning on N_u and N_v only introduces weak dependencies, wich implies that a Poisson approximation should still be satisfactory" Une telle déclaration si elle appelle des justifications mathématiques qui ne sont pas présente dans l'article, devrait au moins appeler une étude de validité de l'approximation par des résultats "expérimentals."

4.1.3 calcul de l'espérance

Dans le cas d'erreurs de séquencage, on peut raisonnablement supposer que les variables aléatoires $\{I_{u,j}\}$ sont indépendantes. En effet, l'apparition d' une erreur sur le read j ne depend pas des erreurs presentes sur les autres reads. La variables$I_{u,j}$ prend les valeurs 1 et 0 avec les probabilités p_j et $1 - p_j$.

Les auteurs définissent les variables $N_u = \sum_i I_{u,i}$ et $N_u^{(j)} = \sum_{i \neq j} I_{u,i} = N_u - I_{u,j}$. Ils approchent les distributions de N_u et $N_u^{(j)}$ par des distributions poissoniennes. Le paramètre d'une loi de Poisson étant l'espérance de la variable aléatoire, les paramètres λ_u et $\lambda_u^{(j)}$ de N_u et $N_u^{(j)}$ sont posés égaux à $\sum_{i=1}^{k} p_i$ et $\sum_{i=1}^{k} p_i - p_j$ respectivement. Nous étudions ci-dessous la validité théorique de cette hypothèse.

Nous rappelons d'abord quelques propriétés de la loi de Poisson.

Théorème 4.1 *Soit $(X_i)_{i \in I}$ une famille de variables aléatoires indépendantes et suivant des lois de Poisson de paramètre λ_i. Alors la variable aléatoire $\sum_{i \in I} X_i$ suit une loi de Poisson de paramètre $\sum_{i \in I} \lambda_i$.*

Preuve :

La preuve se base sur les deux propriétés suivantes:

1 La transformée de Laplace d'une v.a. X suivant une loi de Poisson de paramètre λ est $\Phi_X(t) = e^{\lambda(e^t - 1)}$ (et c'est caractéristique d'une loi de Poisson).

2 Si X et Y sont indépendantes, alors:

$$\Phi_{X_1 + X_2}(t) = \Phi_{X_1}.\Phi_{X_2} \quad .$$

La première propriété se prouve par :

$\Phi_X(t) = E[e^{tX}] = \sum_{k=0}^{\infty} P(X = k)e^{tk} = \sum_{k=0}^{\infty} \frac{\lambda^k}{k!} e^{-\lambda} e^{tk} = e^{-\lambda} \sum_{k=0}^{\infty} \frac{(\lambda.e^t)^k}{k!} = e^{-\lambda}.e^{\lambda.e^t} = e^{\lambda(e^t - 1)}$.

En utilisant ces deux propriétés, on obtient la série génératrice de la somme :

$$\Phi_{\sum_{i \in I} X_i}(t) = \prod_{i \in I} e^{\lambda_i(e^t - 1)} = e^{(\sum_{i \in I} \lambda_i)(e^t - 1)}$$

•

On suppose ici que $N_u^{(j)}$ suit bien notre loi de Poisson. La série génératrice de probabilité de $N_u = N_u^{(j)} + I_j$, est, du fait de l'indépendance des variables aléatoires:

$$\begin{aligned} \Phi_{N_u}(t) &= \Phi_{N_u^{(j)}}(t).\Phi_{I_j(t)} \\ &= e^{\lambda_u^{(j)}(e^t - 1)}.(p_{j,u}e^t + (1 - p_{j,u})) \\ &= e^{\lambda_u^{(j)}(e^t - 1)}.(1 + p_{j,u}(e^t - 1)) \end{aligned}$$

En utilisant le développement de Taylor de l'exponentielle : $e^x = 1+x+x^2/2+...$, on obtient:

$$e^{p_{j,u}(e^t-1)} = 1 + p_{j,u}(e^t - 1) + O(p_{j,u}^2.\frac{(e^t-1)^2}{2})$$

On a alors

$$\begin{aligned}\Phi_{N_u}(t) &= e^{\lambda_u^{(j)}(e^t-1)+p_j(e^t-1)} + O(p_j^2.\frac{(e^t-1)^2}{2}) \\ &= e^{(\lambda_u^{(j)}+p_j)(e^t-1)}.(1 + O(p_j^2.\frac{(e^t-1)^2}{2}))\end{aligned}$$

Soit:

$$\Phi_{N_u}(t) = e^{\lambda_u(e^t-1)}(1 + O(p_j^2.\frac{(e^t-1)^2}{2}))$$

On obtient donc ici une estimation de l'erreur d'une des étapes du calcul de l'espérance de $P(C = x|N_u = n_u, N_v = n_v)$.

4.2 Evaluation des performances

Principe. L'analyse théorique de l'approximation poissonienne en pointant les faiblesses de sa justification nous a amené à essayer d'en évaluer les performances. Le principe est de se placer dans le cas particulier où les p_i et q_j sont égaux entre eux (cas uniforme) pour pouvoir ainsi comparer les distributions données par la méthode combinatoire exacte 3.3.1 et par l'approximation 3.3.2. Cela nous donnera une mesure des performance de l'approximation.

Protocole. Nous allons comparer la distribution de C sachant N_u et N_v pour différentes valeurs de k, p et q, c'est-à-dire que l'on calcule la probabilité $P(C = x|N_u = n_u, N_v = n_v)$pour différentes valeurs de x, n_u, n_v. Pour chaque valeur de k (on rappelle au lecteur qu'il s'agit du nombre de séquences du multi-alignement), on obtient trois matrices avec x variant de 0 à $\frac{k}{2}$ en colonne et n_u variant de 0 à $\frac{k}{2}$ en ligne. Pour pouvoir exposer nos résultats de facon lisible sous forme de matrices, on prendra $n_u = n_v$. La première matrice donne la distribution calculée par la méthode exacte, la seconde celle calculée par l'approximation. La troisième nous donne un pourcentage de différence entre les valeurs des deux premières.

Calculs

- Pour le calcul exact, on utilise la formule donnée en 3.3.1.
- Pour l'approximation, on a vu en 3.3.2 que la distribution de C sachant $N_u = n_u$ et $N_v = n_v$ est approximée par une distribution de Poisson de paramètre et d'espérance :

$$E(C|N_u = n_u, N_v = n_v) \approx \sum_{j=1}^{k} \frac{n_u p_j}{n_u p_j + \lambda_u^{(j)}(1-p_j)} \times \frac{n_v q_j}{n_v q_j + \lambda_u^{(j)}(1-q_j)}$$

Quand on se place dans le cas où les p_i et q_j sont égaux entre eux, la formule se simplifie :

$$\begin{aligned} E(C|N_u = n_u, N_v = n_v) &\approx \frac{kn_u n_v p^2}{(n_u p + p(k-1)(1-p)) \times (n_v p + p(k-1)(1-p))} \\ &\approx \frac{kn_u n_v}{(n_u + (k-1)(1-p)) \times (n_v + (k-1)(1-p))} \end{aligned}$$

On sait que l'espérance d'une loi de Poisson est aussi son paramètre.
On a donc :

$$P(C = x|N_u = n_u, N_v = n_v) = e^{-E(C|N_u=n_u,N_v=n_v)} \times \frac{E(C|N_u = n_u, N_v = n_v)^x}{x!}.$$

- Pour le pourcentage de différence, on calcule $\frac{P_2 - P_1}{P_1}$. Ce qui explique que l'on peut obtenir des valeurs négatives.

Résultats. Nous ne présentons ici que les résultats obtenus pour $k = 8$. En annexe, nous fournissons des résultats complémentaires, les matrices pour des valeurs de k de 4 et 16. Ici, $p = q = 0,1$. Voici un exemple de lecture des matrices : pour k = 4, $P(C = 1|N_u = 2, N_v = 2) = 6.6667\,10^{-1}$.
Méthode de comptage :

$$\begin{pmatrix} 1. & 0. & 0. & 0. & 0. \\ 8.7500\,10^{-1} & 1.2500\,10^{-1} & 0. & 0. & 0. \\ 5.3571\,10^{-1} & 4.2857\,10^{-1} & 3.5714\,10^{-2} & 0. & 0. \\ 1.7857\,10^{-1} & 5.3571\,10^{-1} & 2.6786\,10^{-1} & 1.7857\,10^{-2} & 0. \\ 1.4286\,10^{-2} & 2.2857\,10^{-1} & 5.1429\,10^{-1} & 2.2857\,10^{-1} & 1.4286\,10^{-2} \end{pmatrix}$$

Approximation de Poisson :

$$\begin{pmatrix} 1. & 0. & 0. & 0. & 0. \\ 8.6060\,10^{-1} & 1.2920\,10^{-1} & 9.6975\,10^{-3} & 4.8527\,10^{-4} & 1.8212\,10^{-5} \\ 6.2844\,10^{-1} & 2.9192\,10^{-1} & 6.7799\,10^{-2} & 1.0498\,10^{-2} & 1.2191\,10^{-3} \\ 4.3498\,10^{-1} & 3.6210\,10^{-1} & 1.5072\,10^{-1} & 4.1823\,10^{-2} & 8.7040\,10^{-3} \\ 2.9924\,10^{-1} & 3.6104\,10^{-1} & 2.1780\,10^{-1} & 8.7593\,10^{-2} & 2.6421\,10^{-2} \end{pmatrix}$$

Pourcentage de différence :

$$\begin{pmatrix} 0.0 & 0 & 0 & 0 & 0 \\ -1.645371577 & 3.356336880 & 0 & 0 & 0 \\ 17.30951370 & -31.88589712 & 89.83753462 & 0 & 0 \\ 143.5861314 & -32.40759445 & -43.73160829 & 134.2076659 & 0 \\ 1994.651906 & 57.95282543 & -57.65029344 & -61.67808622 & 84.94504519 \end{pmatrix}$$

[[commenter les matrices]]
[[domaine de validites]]
[[matrices de differences en pourcentage]]

5 Notre méthode : utilisation des séries génératrices

Comme le démontre l'analyse de la section 4.1 et l'évaluation des performances de la section 4.2, l'approximation poissonienne ne donne pas de résultats précis dans de nombreux cas. Nous allons donc introduire dans cette section une nouvelle méthode de calcul fondée sur l'utilisation des séries génératrices. Il s'agit non plus d'une approximation mais d'un calcul exact comme dans le cas de la formule combinatoire simple, mais qui, contrairement à cette dernière, permet de prendre en compte le cas non uniforme (c'est-à-dire avec des p_i pas tous identiques).

Nous allons exposer la méthode (5.1), analyser sa complexité pour savoir si elle est applicable à notre problème (5.2), et montrer comment en avoir une implémentation algorithmique efficace en Maple (5.3).

5.1 Principe

Notre but est de déterminer la distribution des coincidences entre les déviations du consensus des colonnes u et v sachant le nombre de ces déviations. Nous allons tout d'abord exprimer cette distribution en fonction de coefficients de séries génératrices, puis donner une forme pour calculer ces dernières.

Definition 5.1 *Soit $s(u, v, z)$ la série génératrice de probabilités comptant les déviations du consensus sur la colonne u, les déviations du consensus sur la colonne v et les coincidences:*

$$s(u, v, z) = \sum_{x, n_u, n_v} P(C = x, N_u = n_u, N_v = n_v) u^{n_u} v^{n_v} z^x \quad . \tag{1}$$

On note $s(u, v)$ la série $s(u, v, 1)$.

De

$$P(C = x | N_u = n_u, N_v = n_v) = \frac{P(C = x, N_u = n_u, N_v = n_v)}{P(N_u = n_u, N_v = n_v)}$$

et

$$[u^{n_u} v^{n_v}] s(u, v, 1) = P(N_u = n_u, N_v = n_v) \ ,$$

on déduit

$$P(C = x | N_u = n_u, N_v = n_v) = \frac{[z^x u^{n_u} v^{n_v}] s(u, v, z)}{[u^{n_u} v^{n_v}] s(u, v, 1)} \quad .$$

Proposition 5.1 *La série génératrice $s(u, v, z)$ satisfait:*

$$s(u, v, z) = \prod_{i=1}^{k} (p_i q_i uvz + (1 - p_i)(1 - q_i) + p_i(1 - q_i)u + (1 - p_i)q_i v) \tag{2}$$

On obtient $s(u, v)$ en prenant $z = 1$ dans cette formule.

preuve La variable u (resp. v) marque une déviation sur la colonne u (resp. v) et la variable z marque une coincidence. Pour la séquence i on a donc

- une coincidence avec probabilité $p_i q_i$ marquée par uvz
- une déviation en u et pas en v avec probabilité $p_i(1-q_i)$ marquée par u
- une déviation en v et pas en u avec probabilité $(1-p_i)q_i$ marquée par v
- pas de déviation avec probabilité $(1-p_i)(1-q_i)$

Ensuite prendre l'union de ces événements pour la séquence i revient à faire la somme de ces monômes et prendre l'intersection sur les séquences à prendre le produit des séries génératrices. •

Remarque 5.1 *On obtient facilement la formule dans le cas uniforme :*

$$s(u,v,z) = (p^2uvz + (1-p)^2 + p(1-p)(u+v))^k$$

5.2 Complexité

[[le nombre de termes est exponentiel...]]

5.3 Algorithmique et Maple

Comme souligné en section 5.2, le nombre de termes de notre calcul est exponentiel en k, le nombre de séquences. Il est impossible d'obtenir des résultats dès que k dépasse 10 si on essaie pas d'optimiser l'implémentation en Maple. Il existe différentes méthodes pour diminuer les temps de calculs, comme

- éviter les calculs redondants grâce à
  - la mise en mémoire des sous-formules sous forme formelle.
  - la mise en mémoire des sous-calculs sous forme numérique.
  - l'emploi de mtéthodes de substitutions.
- éviter les calculs inutiles grâce
  - au troncage des calculs
  - au choix des bornes
- jouer sur l'ordre des calculs
- étudier les termes négligeables.

. Dans cette sous-section nous présentons, l'idée de base de notre algorithme qui permet d'éviter de nombreux calculs redondants par l'utilisation des notations et du lemme suivants :

Monôme	Coefficient
g_0	1
g_1	σ_1
g_2	$-1/2\sigma_2 + 1/2\sigma_1^2$
g_3	$1/3\sigma_3 - 1/2\sigma_2\sigma_1 + 1/6\sigma_1^3$
g_4	$-1/4\sigma_4 + 1/3\sigma_3\sigma_1 + 1/8\sigma_2^2 - 1/4\sigma_2\sigma_1^2 + 1/24\sigma_1^4$
g_5	$1/5\sigma_5 - 1/4\sigma_4\sigma_1 - 1/6\sigma_3\sigma_2 + 1/6\sigma_3\sigma_1 + 1/8\sigma_2^2\sigma_1 - 1/12\sigma_2\sigma_1^3 + 1/120\sigma_1^5$

Table 1: Coefficients de P(t)

Notation: Soient a et b deux entiers, et $(p_i)_{1\leq i\leq k}$ et $(q_i)_{1\leq i\leq k}$. On note:

$$\sigma_{a,b}(\{p_i\},\{q_i\}) = \sum_{i=1}^{k} p_i^a q_i^b \ ,$$

et

$$\sigma_a(\{p_i\}) = \sum_{i=1}^{k} p_i^a \ .$$

Lemme 5.1 *La fonction $P(t) = \prod_{i=1}^{k}(1+p_i t)$ se réécrit $P(t) = \sum_{j\geq 0} g_j t^j$ où g_j est une fonction des $(\sigma_1(\{p_i\}, ..., \sigma_j(\{p_i\})$. Les premiers coefficients sont donnés dans la table 1.*

Preuve : Pour établir ce résultat, on transforme P(t) en l'exponentielle d'une somme $exp(\sum_{i=1}^{k} ln(1+p_i t))$. On utilise le développement en série $ln(1+x) = x - x^2/2 + ... + (-1)^{n+1}/nx^n + \cdots$. En inversant l'ordre des sommations, on obtient:

$$P(t) = exp(\sum_{i=1}^{k}\sum_{j\geq 1}\frac{(-1)^{j+1}}{j}(p_i t)^j) = exp(\sum_{j\geq 1}\sigma_j(\{p_i\})\frac{(-1)^{j+1}t^j}{j})$$

La formule de Taylor permet d'obtenir le développement en série de t, avec des coefficients qui dépendent des probabilités (p_i) par les fonctions σ_j. •

Proposition 5.2

- *La série génératrice $s(u,v)$ se réécrit sous la forme :*

$$s(u,v) = C \cdot \sum_{a\geq 0} g_a u^a \cdot \sum_{b\geq 0} g_b v^b$$

où les g_a et g_b sont des séries dont les coefficients dépendent respectivement des familles $(\sigma_c(\{\frac{p_i}{1-p_i}\}))_{c\geq 0}$ et $(\sigma_c(\{\frac{q_i}{1-q_i}\}))_{c\geq 0}$.

- *La série génératrice $s(u,v,z)$ se réécrit sous la forme :*

$$s(u,v,z) = \sum_{x=0}^{k} z^x f_x(u,v)$$

où les $f_x(u,v)$ sont des séries dont les coefficients dépendent de la famille $\sigma_{a,b}(\{\frac{p_i}{1-p_i}\},\{\frac{q_i}{1-q_i}\})$.

Preuve :

Pour prouver notre résultat nous allons utiliser le lemme 5.1 après avoir réécrit nos séries génératrices.

- **Calcul de** $s(u,v)$ D'après la proposition 5.1, on a

$$\begin{aligned} s(u,v) &= \prod_{i=1}^{k}(p_iq_iuv + (1-p_i)(1-q_i) + p_i(1-q_i)u + (1-p_i)q_iv) \\ &= \prod_{i=1}^{k}(p_iu + (1-p_i))(q_iv + (1-q_i)) \\ &= A \cdot \prod_{i=1}^{k}(1 + \frac{p_i}{1-p_i}u) \prod_{i=1}^{k}(1 + \frac{q_i}{1-q_i}v) \end{aligned}$$

avec

$$A = \prod_{i=1}^{k}(1-p_i)(1-q_i) \ . \tag{3}$$

Les deuxième et troisième facteurs se réécrivent comme des produits $P(z)$ avec les substitutions $p_i \rightarrow \frac{p_i}{1-p_i}$ et $p_i \rightarrow \frac{q_i}{1-q_i}$. D'après le lemme 5.1, $\prod_{i=1}^{k}(1 + \frac{p_i}{1-p_i}u)$ se réécrit comme une somme $\sum_{j\geq 0} g_j u^j$ où g_j est une fonction des $(\sigma_1(\{\frac{p_i}{1-p_i}\}, ..., \sigma_j(\{\frac{p_i}{1-p_i}\})$ (de même $\prod_{i=1}^{k}(1 + \frac{q_i}{1-q_i}v)$ se réécrit comme une somme $\sum_{j\geq 0} g'_j v^j$ où g'_j est une fonction des $(\sigma_1(\{\frac{q_i}{1-q_i}\}, ..., \sigma_j(\{\frac{q_i}{1-q_i}\})$).

- **Calcul de** $s(u,v,z)$ On pose $\phi_i(u,v) = (1-p_i)(1-q_i) + p_i(1-q_i)u + q_i(1-p_i)v)$ On réécrit:

$$\begin{aligned} s(u,v,z) &= \prod_{i=1}^{k}(p_iq_iuvz + (1-p_i)(1-q_i) + p_i(1-q_i)u + (1-p_i)q_iv \\ &= \prod_{i=1}^{k}(p_iq_iuvz + \phi_i(u,v)) \end{aligned}$$

En factorisant $\phi_i(u,v)$, on obtient :

$$s(u,v,z) = B \cdot \prod_{i=1}^{k}(1 + \frac{p_iq_iuv}{\phi_i(u,v)}z)$$

en posant :

$$B = \prod_{i=1}^{k}\phi_i(u,v)$$

Le second facteur se réécrit comme un produit $P(z)$ avec la substitution $p_i \rightarrow \frac{p_iq_iuv}{\phi_i(u,v)}$. D'après le lemme 5.1, $\prod_{i=1}^{k}(1 + \frac{p_iq_iuv}{\phi_i(u,v)}z)$ se réécrit comme une somme

•

La proposition 5.2 permet donc d'éviter un triple calcul, mais au contraire de calculer une fois pour toute les coefficients du développement de P, de les sauvegarder, et de les utiliser à chaque fois en effectuant la substitution appropriée.

5.4 Résultats

[[peut etre (ou plutot) une section entiere a completer par la suite]]

[[pcorr ptot matrice choix de Dmin opt....]]

[[dire que l'o expose les resultats que pour p=q mais travaux a venir des differnts autres matrices de resultats]]

6 Maple

introduction et remarques generales aux calculs de s(u,v) et de s(u,v,z).

- parler des variables et fonctions maple
- Nous avons joint en annexe...
- En plus de ces différentes parties, nous avons deux phases principales lors de notre calcul de s(u,v) et de s(u,v,z). La distinction a l'origine de ces deux phases est de separer ce qui depend de k de ce qui n'en depend pas. Ainsi on ne fera qu'une seule fois les calculs de formules intermediaires qui n'en dependent pas.
-

```
# LE PROJET

# L'objectif :
# On veut effectuer une comparaison de methodes qui calculent la probabilite

# Les fichiers maple de ce projet
#  - proc_num.mpl
#  - proc_den.mpl
#  - comparaison.mpl

# "proc" est l'abreviation pour procedures, "num" pour numerateur, "den" pou
# Dans le fichier 'proc_num' (resp. 'proc_den') on trouve un ensemble de pro
# Dans le fichier 'comparaison.mpl', on desire implementer une evaluation co
#Cela s'effectue en trois etapes :
# - 1) On calcule les formules generales ('g','s1prim','s2prim') que l'on st
# - 2) on fixe une valeur pour k. Et on calcule les formules correspondantes
# - 3) ensuite, on extrait les coefficients necessaires correspondant a diff
```

6.1 Xmax

Le but final de nos calculs est d'extraire un coefficient a partir de nos fonctions en u,v,z. Les degres que nous considererons correspondent aux valeurs de n_u, n_v, x que l'on etudiera. Or n_u, n_v, x sont inferieurs a k/2 et les k consideres pour une couverture du genome de l'ordre de 6 a 8 ne depassent pas 20.

Lors de nos calculs, il ne sera donc pas necessaire de calculer nos coefficients au-dela d'un certain ordre que l'on appellera x_{max}. Typiquement $x_{max} = 10$. En effet avec une valeur de 10, on peut effectuer tous nos calculs pour des valeurs de k allant jusqu'a 20.

Ainsi par exemple, nos sommes $\sum_{j\geq 1} a_j u^j$ seront remplacees par $\sum_{j=1}^{x_{max}} a_j u^j$ et nos calculs seront regulierement tronques pour ne garder que des monomes en u,v,z de puissances inferieures a x_{max}.

6.2 Principe du calcul en Maple de $s(u,v)$

[[!!! mettre un exemple de formule que nous obtenons - tables ...]] [[!!! parler des variables maple, du programme des commandes...]]

On a vu que

$$s(u,v) = A.exp(\sum_{j\geq 1}(\sum_{i=1}^{k}(\frac{p_i}{1-p_i})^j)\frac{(-1)^{j+1}u^j}{j}).exp(\sum_{j\geq 1}(\sum_{i=1}^{k}(\frac{q_i}{1-q_i})^j)\frac{(-1)^{j+1}v^j}{j})$$

On pose

$$\sigma_j = (\sum_{i=1}^{k}(\frac{p_i}{1-p_i})^j$$

et

$$\tau_j = (\sum_{i=1}^{k}(\frac{q_i}{1-q_i})^j$$

En effet ces quantites se retrouveront a de nombreux endroits dans nos calculs et nous voulons eviter de les recalculer a chaque fois pour des considerations d'efficacite des calculs.

Ce qui nous donne

$$s(u,v) = A.exp(\sum_{j\geq 1}\sigma_j\frac{(-1)^{j+1}u^j}{j}).exp(\sum_{j\geq 1}\tau_j\frac{(-1)^{j+1}v^j}{j})$$

On pose

$$s1 = exp(\sum_{j\geq 1}\sigma_j\frac{(-1)^{j+1}u^j}{j})$$

et

$$s2 = exp(\sum_{j\geq 1}\tau_j\frac{(-1)^{j+1}v^j}{j})$$

En effet on voit que dans le developpement de Taylor en u et v que nous devons effectuer, les coefficients en u vont venir de s_1 et ceux en v de s_2. On peut donc effectuer des calculs separes de A, s_1 et s_2.

Pour diminuer le nombre de variables lors des developpements, on pose :

$$r_j = \sigma_j\frac{(-1)^{j+1}u^j}{j}$$

On obtient

$$s_1 = exp(\sum_{j\geq 1}r_j u^j)$$

On effectuera le calcul que pour (see expl xmax)

$$s_1 = exp(\sum_{j=1}^{x_{max}}r_j u^j)$$

On effectue alors un developpement de Taylor jusqu'au degre x_{max}. On obtient

$$s_1 = \sum_{j=1}^{x_{max}} f(\{r_i\}_{i \in 1..x_{max}}) u^j$$

On garde cette formule en memoire pour s_1 et on fait de meme pour s_2. Il s'agit de formules intermediaires de base qui nous permettront de calculer les formules finales pour les differentes valeurs de k que nous allons tester.

Maintenant k est fixé Jusqu'ici nous n'avions pas besoin de la valeur de k. Pour la suite nous fixons k.

On calcule maintenant indépendamment du reste les σ_j (τ_j) et r_j. Pour une extraction du coefficient en $[z^x u^{n_u} v^{n_v}]$, nous avons besoin des σ_j (τ_j) pour $j = 1..n_u$ ($j = 1..n_v$). Pour etre tranquille pour tous nos calculs nous gardons en memoire les resultats pour $j = 1..xmax$.[[on pourrait faire nos calculs pour un max qui depend de k pour optimiser mais cela ne changerait pas grandement l'efficacite vu que c'est de petits calculs]]

On effectue ensuite une substitution (dans notre formule intermediaire de s_1 (de s_2)) de r_j en $\sigma_j \frac{(-1)^{j+1} u^j}{j}$ (en $\tau_j \frac{(-1)^{j+1} v^j}{j}$).

On obtient ainsi pour chaque valeur de k que nous voulons considérer les formules finales de s_1, s_2 et A à partir desquelles il sera immédiat d'extraire les coefficients de s(u,v,z).

6.3 Principe du calcul en Maple de s(u,v,z)

On a vu que

$$s(u,v,z) = B \cdot exp(\sum_{i=1}^{k}\sum_{j\geq 1}(-1)^{j+1}/jz^j(\frac{p_iq_iuv}{\phi_i(u,v)})^j)$$

avec

$$B = \prod_{i=1}^{k}\phi_i(u,v)$$

et

$$\phi_i(u,v) = (1-p_i)(1-q_i) + p_i(1-q_i)u + q_i(1-p_i)v$$

.

On nomme s_0 la partie exponentielle de s(u,v,z)

$$s_0 = exp(\sum_{i=1}^{k}\sum_{j\geq 1}(-1)^{j+1}/jz^j(\frac{p_iq_iuv}{\phi_i(u,v)})^j)$$

6.3.1 Considerations generales et plan du calcul de s(u,v,z) :

Dans cette formule, B est une fonction en u,v. La partie exponentielle est une fonction en u, v, z. Les coefficients en z viendront donc de celle-ci. Nous allons donc d'une part effectuer un calcul de B de facon a le m,ettre sous la forme $\sum_{a,b} c_{a,b}u^av^b$. D'autre part, nous effectuerons un developpement de la partie exponentielle et nous garderons une table de fonctions en u et v de la meme forme qui correspondront aux coefficients des differentes puissances de z. Ensuite il suffira de multiplier chacunes de ces fonctions par B pour obtenir la table des polynomes en u et v a partir desquels on pourra extraire les coefficients correspondant aux probabilites que l'on voudra etudier pour divers n_u et n_v

6.3.2 Traitement de la partie exponentielle

$$s_0(u,v,z) = exp(\sum_{i=1}^{k}\sum_{j\geq 1}(-1)^{j+1}/jz^j(\frac{p_iq_iuv}{\phi_i(u,v)})^j)$$

D'ou

$$s_0(u,v,z) = exp(\sum_{j\geq 1}(-1)^{j+1}/jz^j\sum_{i=1}^{k}(\frac{p_iq_iuv}{\phi_i(u,v)})^j)$$

On a maintenant à développer cette partie exponentielle. On veut éviter de calculer un grand nombre de fois les mêmes quantités

On pose donc

$$\xi_j = \sum_{i=1}^{k}(\frac{p_iq_iuv}{\phi_i(u,v)})^j$$

En effet ces quantités se retrouveront à de nombreux endroits dans notre développement et nous voulons éviter de les recalculer à chaque fois pour des considérations d'efficacité des calculs.

Ce qui nous donne :

$$s_0(u,v,z) = exp(\sum_{j\geq 1}(-1)^(j+1)/j\xi_j z^j)$$

On ne considère que les termes correspondant à j allant de 1 à x_{max} (cf section x_{max}).

$$s_0(u,v,z) = exp(\sum_{j=1}^{x_{max}}(-1)^(j+1)/j\xi_j z^j)$$

On extrait alors le coefficient de Taylor en z^x pour x allant de 1 à x_{max}.

Indépendamment on effectue le calcul des ξ_j pour j allant de 1 à x_{max}.

$$\xi_j = \sum_{i=1}^{k}(\frac{p_i q_i uv}{(1-p_i)(1-q_i)+p_i(1-q_i)u+q_i(1-p_i)v})^j$$

En divisant et en multipliant par $(1-p_i)(1-q_i)$, on a

$$\xi_j = \sum_{i=1}^{k}(\frac{(\frac{p_i}{1-p_i}u)(\frac{q_i}{1-q_i}v)}{(\frac{p_i}{1-p_i}u)+(\frac{q_i}{1-q_i}v)+1})^j$$

On pose t = $(\frac{p_i}{1-p_i}u)$ et $s = (\frac{q_i}{1-q_i}v)$

$$\xi_j = \sum_{i=1}^{k}(\frac{ts}{t+s+1})^j$$

Il suffit maintenant de s'intéresser au développement de $(\frac{ts}{t+s+1})^j$ sous la forme $\sum_{a,b} e_{j,a,b}u^a v^b$.

Ensuite il n'y a plus quà effectuer les substitutions

$$s \rightarrow (\frac{p_i}{1-p_i}u)$$

et

$$t \rightarrow (\frac{q_i}{1-q_i}v)$$

et de faire la somme sur i allant de 1 à k pour obtenir :

$$\xi_j = \sum_{a,b} e_{j,a,b}(\sum_{i=1}^{k}(\frac{p_i}{1-p_i})^a(\frac{q_i}{1-q_i})^b)u^a v^b$$

On note

$$\sigma_{a,b} = \sum_{i=1}^{k}(\frac{p_i}{1-p_i})^a(\frac{q_i}{1-q_i})^b$$

Les étapes de notre calcul seront :

- calcul de $e_{j,a,b}$
- calcul de $\sigma_{a,b}$
- donner

$$\xi_j = \sum_{a,b} e_{j,a,b}\sigma_{a,b}u^a v^b$$

Ensuite on substitue dans nos fonctions f_x les ξ_j par leur formule. On avait obtenu x_{max} polynomes en u et v de degré x_{max}. Les formules de ξ_j sont elles aussi des polynomes en u et v de degré x_{max}. Par substitution, on obtient un de gré de 2*x_{max}. On va donc tronquer nos nouveaux polynomes en réappliquant la formule de Taylor.

[[definir f]]

6.3.3 Le calcul de B

Indépendamment du calcul de la partie exponentielle, on calcule B sous la forme du polynome en u et v voulu.

On a vu que

$$B = \prod_{i=1}^{k}(1-p_i)(1-q_i) + p_i(1-q_i)u + q_i(1-p_i)v$$

.

En factorisant par $(1-p_i)(1-q_i)$,

$$B = \prod_{i=1}^{k}(1-p_i)(1-q_i)(1 + \frac{p_i}{1-p_i}u + \frac{q_i}{1-q_i}v)$$

.

$$B = \prod_{i=1}^{k}(1-p_i)(1-q_i).\prod_{i=1}^{k}(1 + \frac{p_i}{1-p_i}u + \frac{q_i}{1-q_i}v$$

.

On pose

$$A = \prod_{i=1}^{k}(1-p_i)(1-q_i)$$

$$B = A.\prod_{i=1}^{k}(1 + \frac{p_i}{1-p_i}u + \frac{q_i}{1-q_i}v)$$

$$B = exp(\sum_{i=1}^{k} ln(1 + \frac{p_i}{1-p_i}u + \frac{q_i}{1-q_i}v))$$

$$B = exp(\sum_{i=1}^{k}\sum_{j\geq 1}(-1)^{j+1}/j(\frac{p_i}{1-p_i}u + \frac{q_i}{1-q_i}v)^j)$$

$$B = exp(\sum_{i=1}^{k}\sum_{j\geq 1}(-1)^{j+1}/j\sum_{l=0}^{j}\binom{j}{l}(\frac{p_i}{1-p_i})^l u^l + (\frac{q_i}{1-q_i})^{j-l}v^{j-l})$$

$$B = exp(\sum_{j\geq 1}(-1)^{j+1}/j\sum_{l=0}^{j}\binom{j}{l}u^l v^{j-l}\sum_{i=1}^{k}(\frac{p_i}{1-p_i})^l + (\frac{q_i}{1-q_i})^{j-l})$$

On retrouve ici nos

$$\sigma_{a,b} = \sum_{i=1}^{k}(\frac{p_i}{1-p_i})^a(\frac{q_i}{1-q_i})^b$$

Les étapes du calcul de B :

- calcul de A
- calcul des $\sigma_{a,b}$ (en commun avec l'etape de la partie exponentielle)
- développement de l'exponentielle par Taylor

Quand on effectuera le développement, au lieu de se trimbaler tous ces termes, on utilisera la formule

$$exp(\sum_{j=1}^{x_{max}}\sum_{l=0}^{j}a_{j,l}u^l v^{j-l})$$

Et ensuite on fera la substitution :

$$a_{j,l} \rightarrow (-1)^{j+1}/j\binom{j}{l}\sigma_{l,j-l}$$

6.3.4 Multiplication de B et de la partie exponentielle

On multiplie maintenant nos polynomes f_x par le polynome B et on tronque pour avoir un degré x_{max}.

A la fin de ce processus, on se retrouve avec :

- x_{max} polynomes en u et v $f_x(u,v)$ qui correspondent aux différentes puissances de x. Ils dépendent des $\sigma_{a,b}$ et de A.
- La table des $\sigma_{a,b}$.
- La valeur de A.

Ordre du calcul en maple :

- le calcul s'effectue dans la fonction principale calcul_formules_num
- $$s_0(u,v,z) = exp(\sum_{j\geq 0} \frac{(-1)^{j+1}}{j} \xi_j z^j)$$

  Par Taylor on se ramene a :

  $$s_0(u,v,z) = \sum_{x\geq 0} f_x(\xi_j) z^x)$$

  ```
  fonction g_0 = calcul_f
  avec g_0 table
  ```

- avec

  $$\xi_j = \sum_{a,b} e_{j,a,b} (\sum_{i=1}^{k} (\frac{p_i}{1-p_i})^a (\frac{q_i}{1-q_i})^b) u^a v^b$$

  ```
  fonction calcul_xi
  et calcul_e
  On effectue la substitution dans la formule g0 de la valeur des xi.
  On obtient donc :
  g_0 comme fonction des sigma
  ```

- on calcule B
- on multiplie g_0 et B et on tronque.

6.4 toplevel

```
# On prend une valeur de xmax, p et q.
xmax := 9;
seq(assign('p[i]',.1),i=1..20);
seq(assign('q[i]',.1),i=1..20);

# On effectue le calcul de s(u,v) et s(u,v,z).
g := calcul_formules_num(xmax,sigma):

sprimtmp := calcul_formules_den(xmax):
s1prim := sprimtmp[1]:
s2prim := sprimtmp[2]:

# On effectue des calculs pour differentes valeurs de k grace a la
# fonction calcul_matrice_k_unique
```

```
# Le principe est de calculer les sigma tau_u et tau_v (fonctions
#calcul_sigma, calcul_tau) et d’effectuer une substitution
#Les resultats rendus sont nos matrices.
#
calcul_matrice_k_unique(s1prim,s2prim,xmax,k,n);
```

Cas x_{max} = 2 :

$f_{x=0} = A.(\%0 + v.(\%10 + u.\%11 + u^2.\%12) + v^2.(\%20 + u.\%21 + u^2.\%22) + O(v^3))$

avec

$\%0 := 1 + \sigma_{1,0}u - 1/2\sigma_{2,0} + 1/2\sigma_{1,0}^2u^2 + O(u^3)$

$\%10 := \sigma_{0,1}$

$\%11 := -\sigma_{1,1} + \sigma_{0,1}\sigma_{1,0}$

$\%12 := -\sigma_{1,0}\sigma_{1,1} + \sigma_{0,1}(-1/2\sigma_{2,0} + 1/2\sigma_{1,0}^2)$

$\%20 := -1/2\sigma_{0,2} + 1/2\sigma_{0,1}^2$

$\%21 := -\sigma_{0,1}\sigma_{1,1} + (-1/2\sigma_{0,2} + 1/2\sigma_{0,1}^2)\sigma_{1,0}$

$\%22 := 1/2\sigma_{1,1}^2 - \sigma_{0,1}\sigma_{1,0}\sigma_{1,1} + (-1/2\sigma_{0,2} + 1/2\sigma_{0,1}^2)(-1/2\sigma_{2,0} + 1/2\sigma_{1,0}^2)$

$\xi_1 = uv\sigma_{1,1} - u^2v\sigma_{2,1} - uv^2\sigma_{1,2} + 2u^2v^2\sigma_{2,2}$

$\xi_2 = u^2v^2\sigma 2, 2$

les f en fonctions des ξ_j :

1

ξ_1

$-1/2\xi_2 + 1/2\xi_1^2$

$1/3\xi_3 - 1/2\xi_2\xi_1 + 1/6\xi_1^3$

$-1/4\xi_4 + 1/3\xi_3\xi_1 + 1/8xi_2^2 - 1/4\xi_2\xi_1^2 + 1/24\xi_1^4$

$1/5\xi_5 - 1/4\xi_4\xi_1 - 1/6\xi_3\xi_2 + 1/6\xi_3\xi_1 + 1/8\xi_2^2\xi_1 - 1/12\xi_2\xi_1^3 + 1/120\xi_1^5$

7 Analyse de l'approximation / comparaison de methode

[[a faire]]

k = 4

$$\begin{pmatrix} 1. & 0. & 0. \\ 0.7500000000 & 0.2500000000 & 0. \\ 0.1666666667 & 0.6666666667 & 0.1666666667 \end{pmatrix}$$

$$\begin{pmatrix} 1. & 0. & 0. \\ 0.7500000002 & 0.2499999999 & 0. \\ 0.1666666675 & 0.6666666667 & 0.1666666665 \end{pmatrix}$$

$$\begin{pmatrix} 1. & 0. & 0. \\ 0.7466310865 & 0.2181537140 & 0.03187052067 \\ 0.4846590416 & 0.3510432171 & 0.1271319935 \end{pmatrix}$$

k = 8

$$\begin{pmatrix} 1. & 0. & 0. & 0. & 0. \\ 0.8750000000 & 0.1250000000 & 0. & 0. & 0. \\ 0.5357142857 & 0.4285714286 & 0.03571428571 & 0. & 0. \\ 0.1785714286 & 0.5357142857 & 0.2678571429 & 0.01785714286 & 0. \\ 0.01428571429 & 0.2285714286 & 0.5142857143 & 0.2285714286 & 0.01428571429 \end{pmatrix}$$

$$\begin{pmatrix} 1. & 0. & 0. & 0. & 0. \\ 0.8749999997 & 0.1250000000 & 0. & 0. & 0. \\ 0.5357142858 & 0.4285714283 & 0.03571428570 & 0. & 0. \\ 0.1785714292 & 0.5357142849 & 0.2678571424 & 0.01785714287 & 0. \\ 0.01428571801 & 0.2285714308 & 0.5142857109 & 0.2285714296 & 0.01428571432 \end{pmatrix}$$

$$\begin{pmatrix} 1. & 0. & 0. & 0. & 0. \\ 0.8606029987 & 0.1291954211 & 0.009697535830 & 0.0004852710742 & 0.00001821246290 \\ 0.6284438234 & 0.2919175838 & 0.06779911950 & 0.01049775886 & 0.001219074915 \\ 0.4349752348 & 0.3621021726 & 0.1507189064 & 0.04182279748 & 0.008704016125 \\ 0.2992359866 & 0.3610350296 & 0.2177984909 & 0.08759294575 & 0.02642072075 \end{pmatrix}$$

8 Polymorphisme et Séquençage

[[...]]

9 conclusion

9.1 Résumé

[[Nous avons développée une nouvelle méthode combinatoire...]]

9.2 Travaux en cours et travaux futurs

Comme mon stage finit le 30 septembre (il reste environ 3 mois), cette version du rapport n'est pas complète et ce qui se situe aujourd'hui dans cette section est appelé à déménager dans des sections de l'avant conclusion.

Produire plus de résultats pour la comparaison de méthode : construire les matrices de différence entre notre méthode et l'approximation de Poisson pour différents p_i et q_i

Tester notre nouvelle méthode d'analyse statistique par simulation : analyse du taux d'erreurs (faux positifs) et de la sensibilité (taux de vrais positifs repérés)

Préciser les principes et l'algorithme de recherche des SNPs

References

[1] M. Tammi, E. Arner, T Britton and B. Andersson “Separation of nearly identical repeats in shotgun assemblies using defined nucleotide positions, DNPs”, *Bioinformatics* pp. 379-388, Vol. 18 no. 3 2002

[2] M. Tammi, E. Arner, B. Andersson “TRAP: Tandem Repeat Assembly Program, produces improved shotgun assemblies of repetitive sequences” *Computer Methods and Programs in Biomedicine* 2001

[3] L. Li, J. Kim et M. Waterman “Haplotype Reconstruction from SNP Alignment” *Recomb* 2003

[4] J. Kececioglu, J. Yu “Separating repeats in DNS sequence assembly” *Proceedings of the 5th Annual International Confernece on Computational Molecular Biology* April 2001

[5] P. Pevzner, H. Tang and M. Waterman “A new approach to fragment assembly in DNA sequencing.” *Proceedings of the 5th Annual International Confernece on Computational Molecular Biology* April 2001

[6] B. Ewing and P. Green “Base-calling of automated sequencer traces using Phred II. Error probabilities.” *Genome Res.* 186-194

[7] P. Green “http://www.phrap.org/phrap.docs/phrap.html” 1996

[8] A. Apostolico, Z. Galil “Combinatorial Algorithms on Words”

[9] M. Waterman “Introduction to computational Biology”

[10] Cl. Gomez, B. Salvy, P. Zimmermann “Calcul formel : Mode d’Emploi”

[11] P. Dumas, X. Gourdon “Maple : Son bon usage en mathématiques”

[12] J.F. Lefebvre, N. El-Mabrouk, E. Tillier et D. Sankoff “Detection and validation of single gene inversions” *Proceedings of the 5th Annual International Confernece on Computational Molecular Biology* Vol. 1 no.1 page 1-7 2003

[13] N. El-Mabrouk “Reconstructing an ancestral genome using minimum segmetns duplications and reversals”

[14] N. El-Mabrouk “Genome rearrangement by reversals and insertions/deletions of contiguous segments”

[15] Mireille Régnier “A unified approach to word occurence probabilities” *Discrete applied mathematics* 104 (2000) p 259-280

[16] B. derrida et T. Fink “Sequence Determination from Overlapping Fragments: A Simple Model of Whole-Genome Shotgun Sequencing *Physical Review Letters* 2001

[17] W. Kent et D. Haussler “GigAssembler: An algorithm for the Initial Assembly of the Human Genome Working Draft 2000

[18] S. Schbath, N. Bossard et Simon Tavaré “The Effect of Nonhomogeneous Clone Length Distribution on the Progress of an STS Mapping Project *Journal of Computational Biology* Vol. 7 2000

[19] P. Flajolet et R. Sedgewick “The average case analysis of algorithms, Complex Asymptotics and Generating Functions”

[20] A. Denise, M. Regnier, et M. Vandenbogaert “Assessing the statistical significance of overrepresented oligonucleotides” *WABI* 2001

[21] http://algo.inria.fr/dolley/QuickScore/

[22] Recomb proceedings 2003

A Le code Maple

```
#--------------------------
# 'comparaison.mpl'
#--------------------------

# LE PROJET

# L'objectif :
# On veut effectuer une comparaison de methodes qui calculent la probabilite P[C=x|Nu=nu,Nv=nv].

# Les fichiers maple de ce projet
#  - proc_num.mpl
#  - proc_den.mpl
#  - comparaison.mpl

# "proc" est l'abreviation pour procedures, "num" pour numerateur, "den" pour denominateur.
# Dans le fichier 'proc_num' (resp. 'proc_den') on trouve un ensemble de procedures qui servent au calcul du numera
# Dans le fichier 'comparaison.mpl', on desire implementer une evaluation comparative des differentes methodes de c
#Cela s'effectue en trois etapes :
# - 1) On calcule les formules generales ('g','s1prim','s2prim') que l'on stocke en memoire et qui sont des formule
# - 2) on fixe une valeur pour k. Et on calcule les formules correspondantes a partir de nos formules intermediaire
# - 3) ensuite, on extrait les coefficients necessaires correspondant a differentes valeurs de x,nu,nv pour faire


# Etape 1 :

xmax := 10;

g := calcul_formules_num(xmax);
savelib('g'):
sprimtmp := calcul_formules_den(xmax);
s1prim := sprimtmp[1];
s2prim := sprimtmp[2];
savelib('s1prim'):
savelib('s2prim'):


# Etape 2 :

k := 8;
seq(assign(p[i],.1),i=1..k);
seq(assign(q[i],.1),i=1..k);

calcul_k_donne_num(xmax,k);
s1,s2 := op(calcul_k_donne_den(s1prim,s2prim,xmax,k));


# Etape 3 :

donner_coeff_num(g,4,4,3);
donner_coeff_den(s1,s2,4,4);
donner_proba(g,s1,s2,4,4,3);

donner_proba(g,s1,s2,2,2,2);
donner_coeff_num(g,2,2,2);
donner_coeff_den(s1,s2,2,2);

donner_proba_1(2,2,2);

donner_proba_3 := proc(g,s1,s2,nu,nv,x)
local result;

result := donner_coeff_num(g,nu,nv,x)/donner_coeff_den(s1,s2,nu,nv);
result
end;
```

```
donner_proba_1 := proc(nu,nv,x)
local result;

result :=  evalf((binomial(nv,x)*binomial(k-nv,nu-x))/binomial(k,nu)) ;
result
end;

calcul_esperance_2 := proc(nu,nv,x)
global k,p;
local E,pp;

pp := p[1];
E := (k*nu*nv)/((nu + (k-1)*(1-pp))*(nv + (k-1)*(1-pp)));
E
end:

donner_proba_2 := proc(nu,nv,x)
local result,E;

E := calcul_esperance_2(nu,nv);
result := exp(-E)*E^x/x!;
result
end:


# Principe du calcul
# Boucle for : essayer differentes valeurs de k
# Matrice : essayer differentes valeurs de nu et nv (nu=nv=i) et de cobs (=j)
# (les index des matrices vont de 1 a n)
# calculs necessaire pour le calcul de la methode 3

kmax := xmax*2;
seq(assign(p[i],.1),i=1..kmax);
seq(assign(q[i],.1),i=1..kmax);

T_n := [2,4,8] ;
T_k := [4,8,16] ;

for y from 1 to 3 do

n := T_n[y] ;
k := T_k[y] ;

# calculs necessaire pour le calcul de la methode 3
#seq(assign(p[i],.1),i=1..k);
#seq(assign(q[i],.1),i=1..k);

calcul_k_donne_num(xmax,k);
s1,s2 := op(calcul_k_donne_den(s1prim,s2prim,xmax,k));

# x=j-1
# nu=i-1
# nv=i-1

M_test_1 := matrix(n+1,n+1,
    proc(i,j)
    donner_proba_1(i-1,i-1,j-1)
    end) ;

M_test_3 := matrix(n+1,n+1,
    proc(i,j)
    evalf(donner_proba_3(g,s1,s2,i-1,i-1,j-1));
    end) ;

M_test_2 := matrix(n+1,n+1,
    proc(i,j)
    donner_proba_2(i-1,i-1,j-1)
    end) ;
```

```
od;


#--------------------------
# 'proc_num.mpl'
#--------------------------

#------------------------------
# La Methode 1 n'est pas utilisee.
#
# calcul des xi[methode 1]
#calcul_xi := proc(p,q,k)
#global u,v;
#    local i,j,xi_tmp,xi;
#
#    for j to 5 do
# xi_tmp[j] := sum(((p[i]*q[i]*u*v)/((1-p[i])*(1-q[i]) + p[i]*(1-q[i])*u + q[i]*(1-p[i])*v))^j,i=1..k);
#
#xi[j] := taylor(taylor(xi_tmp[j],u=0,6),v=0,6);
#
#od;
#
#xi;
#end:
#savelib('calcul_xi'):

#------------------------------
# table xi [methode 2]
# calcul des xi
# On les donne directement sous la forme voulue (contrairement a la methode 1)
# car pb pour les avoir par maple et simplification
# A FAIRE
#  - essayer 'simplify' ou 'combine' pour les calculer par maple
#  - faire un calcul a la main des e[j][a][b]

calcul_xi := proc(xmax,sigma,e)
global u,v;
local a,b,j,xi;

for j from 1 to xmax do
xi[j] := sum(sum(u^a*v^b*sigma[a][b]*e[j][a][b],a=1..xmax),b=1..xmax);
od;

xi;
end:

savelib('calcul_xi'):

#------------------------------
# calcul des xi [methode 2] (suite)
# calcul de e en utilisant le developpement de taylor de
# (uv/(u+v+1))^j
# expliquer plus par ecrit comment on enleve les pi...
# la somme en i
# ....

calcul_e := proc(xmax)
local i,j,a,b,e,bip;
global u,v;

for i from 1 to xmax do
bip[i] := taylor(taylor((u*v/(u + v + 1))^i,u=0,xmax+1),v=0,xmax+1);
od;

for j from 1 to xmax do
for a from 1 to xmax do
for b from 1 to xmax do
e[j][a][b] := coeftayl(bip[j],[u,v]=[0,0],[a,b]);
```

```
od;
od;
od;
e
end:

savelib('calcul_e'):

#------------------------------
# table f
# calcul de la partie exponentielle de s
# on utilise la formule de taylor pour obtenir une table de polynomes en en u et v pour les differentes valeurs pos

calcul_f := proc(xi,xmax)
global z;
local j,expo,x,f;

expo := exp(sum((-1)^(j+1)/j*z^j*xi[j],j=1..xmax)) ;

for x from 0 to xmax do
f[x] := coeftayl(expo,z=0,x);
od;
f;
end:

savelib('calcul_f'):

#---------------------------
# tronquer
# Avec la procedure 'calcul_f', on aura les xmax polynomes.
# En utilisant la procedure 'calcul_xi', on obtient les formules des xi
# On effectue une substitution (par dagerisation ou explicitement)
# On obtient des polynomes en u et v de degre 2*xmax
# La procedure 'tronquer' va donc tronquer ces polynomes en utilisant la formule de taylor pour obtenir des polynor

#calcul_ff := proc(sigma,e,u,v)
#    local f,ff,x;
#    f := calcul_f(calcul_xi(sigma,e,u,v));
#
#for x to 5 do
#ff[x] := taylor(taylor(f[x],u=0,6),v=0,6);
#od;
#ff;
#end:
#
#savelib('calcul_ff'):

tronquer := proc(f,xmax)
global u,v;
local x,ff;

for x from 0 to xmax do
ff[x] := taylor(taylor(f[x],u=0,xmax+1),v=0,xmax+1);
od;
ff
end:

savelib('tronquer'):

#---------------------------
# calcul des sigmas[i][j]
# intervient dans le calcul de B et de la partie exponentielle

calcul_sigma := proc(p,q,k,xmax)
    local i,a,b,sigma;

for a from 0 to xmax do
for b from 0 to xmax do
```

```
sigma[a][b] := add((p[i]/(1-p[i]))^a*(q[i]/(1-q[i]))^b,i=1..k);
od;
od;
sigma
end:

savelib('calcul_sigma'):

#--------------------------
# Le calcul de B

#--------------------------
# Le calcul de A
# Sert dans le calcul de B et dans le calcul de s(u,v)

calcul_A := proc(p,q,k)
local A,i;

A := mul((1-p[i])(1-q[i]),i=1..k);
A
end:

savelib('calcul_A'):

#--------------------------
# Le developpement de taylor de B'

calcul_B := proc(a,xmax)
global u,v;
local Bprim,Bdev,j,l;

Bprim := exp(sum(sum(a[j][l]*u^l*v^(j-l),l=0..j),j=1..xmax));
Bdev := taylor(taylor(Bprim,u=0,xmax+1),v=0,xmax+1);
Bdev
end;

savelib('calcul_B'):

#--------------------------
# Multiplier les ff venant de l'expo par le dev de B
# et tronquer
# et multiplier par A

calcul_s := proc(f,B,A,xmax)
global u,v;
local x,fff;

for x from 0 to xmax do
fff[x] := A*taylor(taylor(f[x]*B,u=0,xmax+1),v=0,xmax+1);
od;
fff
end;

savelib('calcul_s'):

#--------------------------
# Calcul (formule) des a[j][l]
# On a utilise dans notre calcul formel les a[j][l] pour alleger la notation et reduire le nombre de variables dans
# On revient a la fin des calculs aux expressions exactes par substitution.

calcul_a := proc(sigma,xmax)
local j,l,a;

for j from 1 to xmax do
for l from 0 to j do
a[j][l] := (-1)^(j+1)*j!*sigma[l][j-l]/(j*(j-l)!*l!);
od;
od;
```

```
a
end;

savelib('calcul_a'):

#--------------------------
# 'calcul_formules' utilise les procedures definies precedemment pour calculer les polynomes en u et v pour les di

calcul_formules_num := proc(xmax)
global sigma,A,p,q;
local g0,xi,g1,g2,g,a,y,e;

g0 := calcul_f(xi,xmax);

e := calcul_e(xmax); # [[reflechir ou mettre ce calcul (si entier ok ici) ]]
xi := calcul_xi(xmax,sigma,e);

g1 := tronquer(g0,xmax);
g2 := calcul_B(a,xmax);

g := calcul_s(g1,g2,A,xmax);


a := calcul_a(sigma,xmax);

# simplify et combine ????
#simplify(g);
g
end:

savelib('calcul_formules_num'):

#--------------------------
#
# Tous les calculs precedents ont ete faits pour un k quelconque.
# Par contre
# - le calcul de A
# - le calcul des sigmas
# ne peuvent se faire sans assigner de valeur a k.
# On a donc separer en deux parties le calcul de formules :
#   1) on trouve la formule pour les polynomes qui ne depend pas de k. On n'aura besoin que de la calculer une seu
#   2) quand on aura besoin d'un k donne, on invoque la fonction 'calcul_k_donne_num' suivante qui a fait les calcu
#
#
#

calcul_k_donne_num := proc(xmax,k)
global A,p,q,sigma;

A := calcul_A(p,q,k);

sigma := calcul_sigma(p,q,k,xmax);

end:

savelib('calcul_k_donne_num'):

#--------------------------
# 'donner_coeff_num'
# Cette procedure donne le coefficient de s(z,u,v) en [z^x,u^nu,v^nv]
# Cela donnera le numerateur de notre fraction.

#--------------------------
# extraire les coeffs a partir des data memorisees

donner_coeff_num := proc(fff,nu,nv,x)
global u,v;
local tmp;
```

```
tmp := coeftayl(fff[x],[u,v]=[0,0],[nu,nv]);
tmp
end;

savelib('donner_coeff_num'):

#--------------------------
# 'proc_den.mpl'
#--------------------------

calcul_tau := proc(p,xmax,k)
local tau,i;

for j from 1 to xmax do
tau[j] := sum((p[i]/(1-p[i]))^j,i=1..k);
od;
tau
end;

savelib('calcul_tau'):

calcul_r := proc(tau,xmax)
local j,r;

for j from 1 to xmax do
r[j] := tau[j]*(-1)^(j+1)/j;
od;
r
end;

savelib('calcul_r'):

calcul_suv := proc(r,u,xmax)
local sprim,s1;

sprim := exp(sum(r[j]*u^j,j=1..xmax));
s1 := taylor(sprim,u=0,xmax+1);
s1
end;

savelib('calcul_suv'):

#--------------------------
#
calcul_formules_den := proc(xmax)
global tau_u,tau_v,u,v;
local r_u,r_v,s1prim,s2prim;

s1prim := calcul_suv(r_u,u,xmax);
s2prim := calcul_suv(r_v,v,xmax);

r_u := calcul_r(tau_u,xmax);
r_v := calcul_r(tau_v,xmax);
[s1prim,s2prim]
end;

savelib('calcul_formules_den'):

#--------------------------
# on a besoin ici de connaitre k

calcul_k_donne_den := proc(s1prim,s2prim,xmax,k)
global p,q,tau_u,tau_v;
local s1,s2;

tau_u := calcul_tau(p,xmax,k);
tau_v := calcul_tau(q,xmax,k);
```

```
#s1prim := subs(tau_u=tau_u,s1prim);
#s2prim := subs(tau_v=tau_v,s2prim);

s1 := simplify(s1prim);
s2 := simplify(s2prim);

[s1,s2];
end;

savelib('calcul_k_donne_den'):

#-------------------------
# 'donner_coeff_den'
# Cette procedure donne le coefficient de s(u,v) en [u^nu,v^nv]
# Cela donnera le denominateur de notre fraction.

donner_coeff_den := proc(s1,s2,nu,nv)
global u,v,A;
local tmp;

tmp := A*coeftayl(s1,u=0,nu)*coeftayl(s2,v=0,nv);
tmp
end;

savelib('donner_coeff_den'):
```

B Le code C

```
Programme : ce qu'il faut arranger

gestion des arguments
une aide (no arguments)

commentaire dans le code + aide de structures et de nom des fonctions



#include <list>
#include <map>
#include <set>
#include <vector.h>
#include <string>
#include <iostream.h>
#include <cstdlib>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>


vector<double> calcul_PQ(vector<double> p) {
  int taille = p.size();
  vector<double> P(taille);

  for(int i=0;i<taille;i++)
    P[i] = p[i]/(1.-p[i]);

  return P;
}

void etape_rec_suvz(vector<vector<vector<double> > > &p1, vector<vector<vector<double> > > &p2,double P, double Q,
void etape_rec_suv(vector<vector<double> > &p1, vector<vector<double> > &p2, double P, double Q, int xmax);


void print_mat_suvz(vector<vector<vector<double> > > m);
void print_mat_suv(vector<vector<double> > m);

vector<vector<vector<double> > > calculer_suvz(int k, int xmax, vector<double> p, vector<double> q);

vector<vector<double> > calculer_suv(int k, int xmax, vector<double> p, vector<double> q);

void construire_matrice(vector<vector<vector<double> > > s_uvz, vector<vector<double> > s_uv, int xmax);

int affichage;

int main(int argc, char ** argv){

  cout << endl << endl;
  cout << "*********************************************" << endl;
  cout << "***    PROGRAMME CALCUL SERIE GENERATRICE    ***" << endl;
  cout << "*********************************************" << endl;

  int k;
  int xmax;


  k = atoi(argv[1]);
  xmax = atoi(argv[2]);

  affichage = atoi(argv[3]);

  vector<vector<vector<double> > > s_uvz;
  vector<vector<double> > s_uv;
```

```
  vector<double> p(k,.1);
  vector<double> q(k,.1);


  s_uvz = calculer_suvz(k,xmax,p,q);
  s_uv = calculer_suv(k,xmax,p,q);

  if(affichage)
    cout << "polynomes finaux : " << endl;

  if(affichage) {
    print_mat_suvz(s_uvz);
    print_mat_suv(s_uv);
  }


  cout << endl;
  cout << "k " << k << endl;
  cout << endl;

  construire_matrice(s_uvz,s_uv,xmax);



}

//*************************************************************************
void construire_matrice(vector<vector<vector<double> > > s_uvz, vector<vector<double> > s_uv, int xmax) {

  if(affichage)
    cout << "fonction : construire_matrice" << endl;

  double element;

  for(int i=0;i<xmax+1;i++) {
    for(int j=0;j<xmax+1;j++) {
element = s_uvz[j][i][i] / s_uv[i][i];
cout << element << "  ";
    }
    cout << endl;
  }
}






//*************************************************************************

vector<vector<vector<double> > > calculer_suvz(int k, int xmax, vector<double> p, vector<double> q){

  if(affichage)
    cout << "fonction : calculer_coef_suvz" << endl;

  vector<vector<vector<double> > > pol(xmax+1);
  vector<vector<vector<double> > > pol_tmp(xmax+1);

  for(int i=0;i<xmax+1;i++) {
    pol[i].resize(xmax+1);
    for(int j=0;j<xmax+1;j++)
      pol[i][j].resize(xmax+1);
  }
  for(int i=0;i<xmax+1;i++) {
    pol_tmp[i].resize(xmax+1);
    for(int j=0;j<xmax+1;j++)
      pol_tmp[i][j].resize(xmax+1);
```

```
  }

  vector<double> P;
  vector<double> Q;

  P = calcul_PQ(p);
  Q = calcul_PQ(q);

  pol[1][1][1] = P[1]*Q[1];
  pol[0][1][0] = P[1];
  pol[0][0][1] = Q[1];
  pol[0][0][0] = 1;

  if(affichage)
    print_mat_suvz(pol);

  for(int i=0;i<(k-1);i++) {
    if(affichage)
      cout << "etape : " << i << endl;
    etape_rec_suvz(pol,pol_tmp,P[i],Q[i],xmax);
    if(affichage)
      print_mat_suvz(pol);
  }


  double A = 1;

  for(int i=0;i<k;i++)
    A *= (1-p[i])*(1-q[i]);

  for(int i=0;i<xmax+1;i++)
    for(int j=0;j<xmax+1;j++)
      for(int k=0;k<xmax+1;k++)
pol[i][j][k] *= A;


  return pol;

}

//*************************************************************************

vector<vector<double> > calculer_suv(int k, int xmax, vector<double> p, vector<double> q){

  if(affichage)
    cout << "fonction : calculer_coef_suv" << endl;

  vector<vector<double> > pol(xmax+1);
  vector<vector<double> > pol_tmp(xmax+1);

  for(int i=0;i<xmax+1;i++)
    pol[i].resize(xmax+1);
  for(int i=0;i<xmax+1;i++)
    pol_tmp[i].resize(xmax+1);

  vector<double> P;
  vector<double> Q;

  P = calcul_PQ(p);
  Q = calcul_PQ(q);

  pol[1][1] = P[1]*Q[1];
  pol[1][0] = P[1];
  pol[0][1] = Q[1];
  pol[0][0] = 1;

  if(affichage)
    print_mat_suv(pol);
```

```
  for(int i=0;i<(k-1);i++) {
    if(affichage)
      cout << "etape : " << i << endl;
    etape_rec_suv(pol,pol_tmp,P[i],Q[i],xmax);
    if(affichage)
      print_mat_suv(pol);
  }


  double A = 1;

  for(int i=0;i<k;i++)
    A *= (1-p[i])*(1-q[i]);

  for(int i=0;i<xmax+1;i++)
    for(int j=0;j<xmax+1;j++)
pol[i][j] *= A;

  if(affichage) {
    cout << "apres multiplication par A : " << A << endl;
    print_mat_suv(pol);
  }

  return pol;

}




//***************************************************************************

void etape_rec_suvz(vector<vector<vector<double> > > &p1, vector<vector<vector<double> > > &p2,double P, double Q,

  if(affichage)
    cout << "fonction : etape_rec_suvz" << endl;

  for(int i=0;i<xmax+1;i++) {
    for(int j=0;j<xmax+1;j++) {
      for(int k=0;k<xmax+1;k++) {
p2[i][j][k] = ((((i>0)&&(j>0)&&(k>0))?P*Q*p1[i-1][j-1][k-1]:0) + ((j>0)?P*p1[i][j-1][k]:0) + ((k>0)?Q*p1[i][j][k-1]
      }
    }
  }
  p1 = p2;
}

void etape_rec_suv(vector<vector<double> > &p1, vector<vector<double> > &p2, double P, double Q, int xmax) {

  if(affichage)
    cout << "fonction : etape_rec_suv" << endl;

  for(int i=0;i<xmax+1;i++) {
    for(int j=0;j<xmax+1;j++) {
p2[i][j] = ((((i>0)&&(j>0))?P*Q*p1[i-1][j-1]:0) + ((i>0)?P*p1[i-1][j]:0) + ((j>0)?Q*p1[i][j-1]:0) + p1[i][j];
    }
  }

  p1 = p2;
}




//***************************************************************************
```

```
void print_mat_suvz(vector<vector<vector<double> > > m) {

  int taille = m.size();

  for(int i=0;i<taille;i++) {
    cout << "i = " << i << endl;
    for(int j=0;j<taille;j++) {
      for(int k=0;k<taille;k++) {
cout << m[i][j][k] << "  ";
      }
      cout << endl;
    }
  }



}

void print_mat_suv(vector<vector<double> > m) {

  int taille = m.size();

  for(int i=0;i<taille;i++) {
    for(int j=0;j<taille;j++)
cout << m[i][j] << "  ";
    cout << endl;
  }



}
```

C Résutats complémentaires de lévaluation des performances.

k = 4

Méthode de comptage :

$$\begin{pmatrix} 1. & 0. & 0. \\ 7.5000\,10^{-1} & 2.5000\,10^{-1} & 0. \\ 1.6667\,10^{-1} & 6.6667\,10^{-1} & 1.6667\,10^{-1} \end{pmatrix}$$

Approximation de Poisson :

$$\begin{pmatrix} 1. & 0. & 0. \\ 7.4663\,10^{-1} & 2.1815\,10^{-1} & 3.1871\,10^{-2} \\ 4.8466\,10^{-1} & 3.5104\,10^{-1} & 1.2713\,10^{-1} \end{pmatrix}$$

Pourcentage de différence :

$$\begin{pmatrix} 0.0 & 0 & 0 \\ -0.4491884667 & -12.73851440 & 0 \\ 190.7954249 & -47.34351744 & -23.72080392 \end{pmatrix}$$

k = 8

Méthode de comptage :

$$\begin{pmatrix} 1. & 0. & 0. & 0. & 0. \\ 8.7500\,10^{-1} & 1.2500\,10^{-1} & 0. & 0. & 0. \\ 5.3571\,10^{-1} & 4.2857\,10^{-1} & 3.5714\,10^{-2} & 0. & 0. \\ 1.7857\,10^{-1} & 5.3571\,10^{-1} & 2.6786\,10^{-1} & 1.7857\,10^{-2} & 0. \\ 1.4286\,10^{-2} & 2.2857\,10^{-1} & 5.1429\,10^{-1} & 2.2857\,10^{-1} & 1.4286\,10^{-2} \end{pmatrix}$$

Approximation de Poisson :

$$\begin{pmatrix} 1. & 0. & 0. & 0. & 0. \\ 8.6060\,10^{-1} & 1.2920\,10^{-1} & 9.6975\,10^{-3} & 4.8527\,10^{-4} & 1.8212\,10^{-5} \\ 6.2844\,10^{-1} & 2.9192\,10^{-1} & 6.7799\,10^{-2} & 1.0498\,10^{-2} & 1.2191\,10^{-3} \\ 4.3498\,10^{-1} & 3.6210\,10^{-1} & 1.5072\,10^{-1} & 4.1823\,10^{-2} & 8.7040\,10^{-3} \\ 2.9924\,10^{-1} & 3.6104\,10^{-1} & 2.1780\,10^{-1} & 8.7593\,10^{-2} & 2.6421\,10^{-2} \end{pmatrix}$$

Pourcentage de différence :

$$\begin{pmatrix} 0.0 & 0 & 0 & 0 & 0 \\ -1.645371577 & 3.356336880 & 0 & 0 & 0 \\ 17.30951370 & -31.88589712 & 89.83753462 & 0 & 0 \\ 143.5861314 & -32.40759445 & -43.73160829 & 134.2076659 & 0 \\ 1994.651906 & 57.95282543 & -57.65029344 & -61.67808622 & 84.94504519 \end{pmatrix}$$

k = 16

Méthode de comptage :

$$\begin{pmatrix} 1. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 9.38\,10^{-1} & 6.25\,10^{-2} & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 7.58\,10^{-1} & 2.33\,10^{-1} & 8.33\,10^{-3} & 0. & 0. & 0. & 0. & 0. & 0. \\ 5.11\,10^{-1} & 4.18\,10^{-1} & 6.96\,10^{-2} & 1.79\,10^{-3} & 0. & 0. & 0. & 0. & 0. \\ 2.72\,10^{-1} & 4.84\,10^{-1} & 2.18\,10^{-1} & 2.64\,10^{-2} & 5.49\,10^{-4} & 0. & 0. & 0. & 0. \\ 1.06\,10^{-1} & 3.78\,10^{-1} & 3.78\,10^{-1} & 1.26\,10^{-1} & 1.26\,10^{-2} & 2.29\,10^{-4} & 0. & 0. & 0. \\ 2.62\,10^{-2} & 1.89\,10^{-1} & 3.93\,10^{-1} & 3.00\,10^{-1} & 8.43\,10^{-2} & 7.49\,10^{-3} & 1.25\,10^{-4} & 0. & 0. \\ 3.15\,10^{-3} & 5.14\,10^{-2} & 2.31\,10^{-1} & 3.85\,10^{-1} & 2.57\,10^{-1} & 6.61\,10^{-2} & 5.51\,10^{-3} & 8.74\,10^{-5} & 0. \\ 7.77\,10^{-5} & 4.97\,10^{-3} & 6.09\,10^{-2} & 2.44\,10^{-1} & 3.81\,10^{-1} & 2.44\,10^{-1} & 6.09\,10^{-2} & 4.97\,10^{-3} & 7.77\,10^{-5} \end{pmatrix}$$

Approximation de Poisson :

$$\begin{pmatrix} 1. & 0. & 0. & 0. & 0. & 0. & 0. & 0. & 0. \\ 9.27\,10^{-1} & 7.05\,10^{-2} & 2.68\,10^{-3} & 6.81\,10^{-5} & 1.30\,10^{-6} & 1.97\,10^{-8} & 2.50\,10^{-10} & 2.72\,10^{-12} & 2.59\,10^{-14} \\ 7.66\,10^{-1} & 2.04\,10^{-1} & 2.72\,10^{-2} & 2.41\,10^{-3} & 1.61\,10^{-4} & 8.56\,10^{-6} & 3.80\,10^{-7} & 1.45\,10^{-8} & 4.82\,10^{-10} \\ 5.89\,10^{-1} & 3.12\,10^{-1} & 8.24\,10^{-2} & 1.45\,10^{-2} & 1.92\,10^{-3} & 2.03\,10^{-4} & 1.79\,10^{-5} & 1.35\,10^{-6} & 8.95\,10^{-8} \\ 4.33\,10^{-1} & 3.62\,10^{-1} & 1.51\,10^{-1} & 4.22\,10^{-2} & 8.82\,10^{-3} & 1.47\,10^{-3} & 2.05\,10^{-4} & 2.45\,10^{-5} & 2.56\,10^{-6} \\ 3.11\,10^{-1} & 3.63\,10^{-1} & 2.12\,10^{-1} & 8.27\,10^{-2} & 2.42\,10^{-2} & 5.65\,10^{-3} & 1.10\,10^{-3} & 1.84\,10^{-4} & 2.68\,10^{-5} \\ 2.20\,10^{-1} & 3.33\,10^{-1} & 2.52\,10^{-1} & 1.27\,10^{-1} & 4.82\,10^{-2} & 1.46\,10^{-2} & 3.69\,10^{-3} & 7.98\,10^{-4} & 1.51\,10^{-4} \\ 1.55\,10^{-1} & 2.89\,10^{-1} & 2.69\,10^{-1} & 1.68\,10^{-1} & 7.81\,10^{-2} & 2.92\,10^{-2} & 9.06\,10^{-3} & 2.42\,10^{-3} & 5.63\,10^{-4} \\ 1.09\,10^{-1} & 2.42\,10^{-1} & 2.68\,10^{-1} & 1.98\,10^{-1} & 1.09\,10^{-1} & 4.85\,10^{-2} & 1.79\,10^{-2} & 5.67\,10^{-3} & 1.57\,10^{-3} \end{pmatrix}$$

Pourcentage de différence :

$$\begin{pmatrix}
0.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1.14947 & 12.8376 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1.02956 & -12.5321 & 226.206 & 0 & 0 & 0 & 0 & 0 & 0 \\
15.3752 & -25.4140 & 18.3513 & 713.787 & 0 & 0 & 0 & 0 & 0 \\
59.3791 & -25.0592 & -30.3951 & 60.0058 & 1505.02 & 0 & 0 & 0 & 0 \\
193.808 & -3.85216 & -43.8142 & -34.3336 & 91.8664 & 2366.65 & 0 & 0 & 0 \\
738.375 & 76.3840 & -35.8755 & -57.5033 & -42.7789 & 95.0256 & 2854.23 & 0 & 0 \\
4819.52 & 461.896 & 16.4721 & -56.5429 & -69.5981 & -55.8872 & 64.5895 & 2663.45 & 0 \\
140345 & 4761.27 & 339.548 & -18.8574 & -71.2398 & -80.0902 & -70.5966 & 13.9876 & 1920.09
\end{pmatrix}$$