Reconstruction d'haplotypes à partir de pedigrees : comparaison des approches de reconstruction par simulation Monte-Carlo

Vincent Garnier

Rapport de stage 2010
Master 2 SRO
Statistiques et recherche opérationnelle

Reconstruction d' haplotypes à partir de pedigrees : comparaison des approches de reconstruction par simulation Monte-Carlo

GARNIER Vincent

Septembre 2010

Sommaire

Liste des figures

Introduction

J’ai effectué mon stage de deuxième année de master à l’Institut National de la Recherche Agronomique (INRA) qui est le premier organisme de recherche agronomique en Europe. Ce stage s’est déroulé du 12 avril au 30 septembre 2010.

L’Institut National de la Recherche Agronomique est un établissement public à caractère scientifique et technologique. L’INRA d’Orléans se situe à Ardon, dans le Domaine de Limère à 10 km d’Orléans. Il est constitué de cinq unités dont l’unité Amélioration, Génétique et Physiologie Forestières, d’un Service Déconcentré d’Appui à la Recherche et d’un Domaine Expérimental situé à Bourges.

Mon stage s'inscrit dans le cadre du projet « NovelTree »; projet ayant pour objectif l'amélioration génétique des propriétés des arbre forestiers et des caractéristiques du bois afin de satisfaire les évolutions de la demande sur le plan qualitatif et quantitatif et d'améliorer la durabilité des forêts dans le contexte du changement climatique.

L'objectif de mon stage est de rassembler les outils disponibles à la reconstruction d'haplotypes puis de comparer leurs efficacités en utilisant des jeux de données issues de la simulation Monte-Carlo. La comparaison de ces logiciels se fera à travers une étude statistique.

Après avoir présenté l' INRA, nous exposerons les méthodes statistiques qui sont utilisées dans les différents logiciels de reconstruction d' haplotypes, puis nous comparerons ces logiciels à travers une étude statistique relativement simple.

Je tiens à remercier vivement l'ensemble des personnes qui m'ont aidé durant la totalité de ce stage et qui m'ont permis d'effectuer mon travail dans les meilleures conditions. Ces remerciements s'adressent plus particulièrement à :

– M. PILATE, Directeur de l'unité Amélioration, Génétique et Physiologie Forestière, qui a accepté de m'intégrer temporairement dans son unité.

- M. SANCHEZ, responsable du stage.

- M. EMILION, professeur encadrant du stage.

- L'ensemble du personnel pour leur sympathie et leur disponibilité.

- Cécile, Lydia, Charles-Edouard, Maxime, et bien d'autres pour avoir passé de bons moments à l' INRA.

1. Contexte et objectifs :

1.1. L'INRA :

1.1.1. Organisation générale :

L' **I**nstitut **N**ational de la **R**echerche **A**gronomique, couramment connu sous son acronyme INRA, a 20 centres engagés dans 21 pôles thématiques prioritaires, près de 150 sites de recherche et d'expérimentation dans toute la France, y compris en outre-mer, avec 14 départements scientifiques dans le domaine de l'agriculture, de l'alimentation et de l'environnement (Figure 1).

Figure 1 – Les implémentations de l' INRA en France

L' **I**nra est un organisme français de recherche en agronomie fondé en 1946 dans le contexte de la reconstruction nationale d'après-guerre et du projet de modernisation de l'agriculture française. Cet institut est placé sous la statut d' Établissement public à caractère scientifique et technologique (EPST), et sous la double tutelle du ministère chargé de la Recherche et du ministère chargé de l' Agriculture.

Les grandes missions à l' Inra sont les suivantes :

- produire et diffuser des connaissances scientifiques ;
- concevoir des innovations et des savoir-faire pour la société ;
- éclairer, par son expertise, les décisions des acteurs publics et privés ;
- développer la culture scientifique et technique et participer au débat science/société ; former à la recherche et par la recherche.

Depuis les mutations du monde agricole, l'INRA a accompagné des filières alimentaires et des territoires avec l'objectif de répondre aux attentes exprimées par la société, notamment celle de la suffisance alimentaire de la nation. Les défis scientifiques ont changé et l'INRA a profondément renouvelé ses approches pour y répondre.

Si l'**agriculture** est toujours au coeur des préoccupations de l'INRA, elle est désormais accompagnée de deux autres champs de recherche en forte interaction : l'environnement et l'alimentation.

L'**environnement** est étudié afin de mieux comprendre le fonctionnement et l'évolution des milieux naturels dans l'optique d'une gestion durable des ressources naturelles.

L'**alimentation** est recherchée pour accompagner l'évolution de toute la chaîne alimentaire vers un bien-être accru des consommateurs en termes de santé, de qualité et de plaisir.

Aujourd'hui, les recherches menées à l'INRA intègrent désormais les préoccupations complexes qui sont au fondement d'un développement durable : l'écologie, la gestion et la préservation des ressources naturelles et de la biodiversité, la viabilité économique et le bienêtre social, le respect de l'animal…

L'INRA est le premier institut de recherche agronomique européen et parmi les trois premiers mondiaux dans les domaines de l'agriculture, de l'alimentation et de l'environnement. L'INRA est aussi le deuxième institut de recherche publique française.

1.1.2. Le centre d' Orléans :

Le centre de l'Inra dans lequel j'ai effectué mon stage a été inauguré le 16 avril 1977 à Ardon.

Du point de vue de la superficie, il se compose de 63 hectares sur le site d'Ardon, 13100 m^2 bâtis, 47 hectares sur 260 parcelles de pépinière et 5000 m^2 de serres contenant 65000 plants forestiers en élevage et expérimentation (site http://www.orleans.inra.fr).

Le centre de recherche d'Orléans est composé de 5 unités dont l'unité Recherche Amélioration, Génétique et Physiologie Forestière où j'ai effectué mon stage, d'un service déconcentré d'Appui à la Recherche et d'un domaine expérimental situé à Bourges (Figure 2).

Figure 2 – Les différentes unités.

Au niveau du personnel, le centre d'Orléans accueille en outre chaque année 50 non-titulaires parmi lesquels un nombre important d'étudiants - Doctorats ou Masters - post-doctorants et chercheurs étrangers.

Quatre domaines de recherche sont traités sur le centre de recherche Inra d'Orléans :

- La sélection d'arbres forestiers sur des critères de croissance optimale, de qualité du bois, et de résistance aux parasites.
- La biologie des insectes forestiers ravageurs, leur épidémiologie, et leurs relations avec les arbres-hôtes.
- La maîtrise des érosions et des pollutions, l'évaluation de risques agroclimatiques.
- L'amélioration génétique des performances des troupeaux et la qualité de leurs produits.

Ces thématiques de recherche, au travers des unités présentes notamment sur le site d'Ardon, donnent au centre d'Orléans une orientation marquée en matière d'environnement et de développement durable.

1.1.3. L'unité Amélioration, Génétique et Physiologie Forestières :

J'ai effectué mon stage dans l' Unité de Recherche Amélioration, Génétique et Physiologie Forestières qui rassemble des compétences en génétique, génomique et physiologie appliquées à une même classe d'objets : les arbres forestiers.

Cette unité est constituée de trois équipes de recherche, tandis que ses effectifs sont de 38 personnes permanentes dont 23 chercheurs et ingénieurs. L'unité encadre en moyenne 4 doctorants et accueille 1 à 2 étudiants de master par an.

Dans le domaine de l'amélioration et de la génétique, cette unité contribue à l'innovation variétale forestière pour le reboisement à basse et moyenne altitude en vue d'une production de bois d'œuvre. De plus, elle développe une étude sur les interactions entre variétés améliorées et populations naturelles d'arbres forestiers.

Le projet de l'unité a pour principaux objectifs la valorisation raisonnée des ressources génétiques forestières et l'évaluation de l'impact écologique des populations domestiquées sur l'écosystème forestier.

Ce projet s'appuie sur trois lignes de force :

- Maintenir un effort de création variétale sur quelques espèces importantes pour les reboisements ;
- Approfondir nos connaissances sur les processus responsables de la formation du bois ;
- Centrer les études de génomique sur le Peuplier.

Au niveau européen, l'unité s'implique très activement dans la recherche en génétique forestière puisqu'elle coordonne 2 projets européens (NovelTree et TreeBreedex) et est partenaire de 2 autres projets (EVOLTREE et EnergyPoplar).

1.2. Contexte du stage :

Mon stage s'inscrit dans le cadre du projet NovelTree qui fait parti des 5 projets coordonnés par l' INRA.

Ce projet européen a été lancé le 25 juin 2008. 14 organismes de recherche, dont 2 en France, participent à ce projet de 6,3 millions d'euros dont 4,1 millions financés par l'Europe pour 4 ans (2008-2011). Ce projet, coordonné par l'INRA d'Orléans a pour objectif l'amélioration génétique des propriétés des arbre forestiers et des caractéristiques du bois afin de satisfaire les évolutions de la demande sur le plan qualitatif et quantitatif et d'améliorer la durabilité des forêts dans le contexte du changement climatique.

Le projet NovelTree se focalise sur 4 espèces majeures dans l'économie forestière européenne : le pin sylvestre, le pin maritime, l'épicéa et le **peuplier**.

Quatre unités de recherche et deux unités expérimentales de l'INRA représentant 18 chercheurs contribuent à ce projet.

Le contexte scientifique de mon stage est le suivant :

Le génotypage des individus, en l'occurrence les peupliers, en termes de lecture du polymorphisme aux gènes le long du génome, produit un double jeu de marqueurs aux gènes pour les organismes diploïdes (double jeu de génome, l'un du père et l'autre de la mère). Ainsi, la composition en marqueurs pour chaque individu analysé est connue « en vrac », mais l'arrangement des polymorphismes dans le double génome n'est pas connu automatiquement.

Par exemple, nous pouvons supposer que l'individu i possède les marqueurs 1, 2, 3, 4, 5, 6, 7, 8, 9 et 10, la figure 3 montre ainsi 3 génotypes possibles pour l'individu i avec ces 10 marqueurs. Pour ces 3 génotypes, les marqueurs sont identiques mais l'arrangement est différent.

Individu i				
1	2	3	4	5
6	7	8	9	10

Individu i				
2	5	8	1	4
7	10	3	6	9

Individu i				
3	7	1	5	9
8	6	2	10	4

Figure 3 – Plusieurs génotypes possibles pour le même individu i.

Le nombre totale de possibilité est très grand puisqu'avec 2 marqueurs (1 et 2 par exemple), il y a 4 possibilités de combinaison pour un locus : (1;1), (1;2), (2;1) ou (2;2), où un locus désigne un emplacement précis sur un chromosome.

1.3. Objectifs du stage :

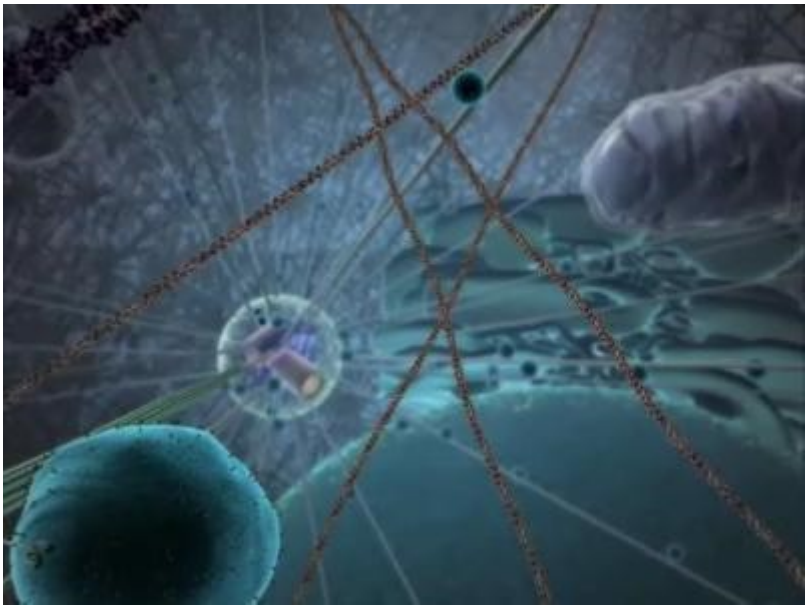
Les différents objectifs de ce stage sont :

- Mettre au point un programme générant un pedigree, générant ainsi des génotypes ;
- Rassembler les outils disponibles de reconstruction d' haplotypes ;
- Comparer l'efficacité de ces outils en utilisant des jeux de données issues de la simulation Monte-Carlo de pedigrees d'individus ;
- Identifier des faiblesses éventuelles de chaque méthode en fonction de la nature des données.

L'algorithme EM et l'approche Bayésienne seront abordés dans ce rapport.

2. Définitions et Méthode :

2.1. Notions de biologie :



2.1.1. Chromosomes, gènes, allèles , homozygotie, hétérozigotie.

Les ***chromosomes*** sont des éléments essentiels du noyau cellulaire, de forme déterminée et en nombre constant pour chaque espèce, ils constituent le **génome** d'un individu. Les chromosomes sont porteurs du patrimoine génétique d'un individu (les gènes), et permettent de distribuer de façon égalitaire ce patrimoine entre deux cellules filles lors d'une division cellulaire .

Un ***gène*** est une portion de chromosome, une séquence ordonnée de nucléotides, une portion d'une molécule d'ADN, qui occupe une position précise sur le chromosome, ou locus (loci lorsque c'est au pluriel) et qui constitue une unité d'information génétique dont la transmission est héréditaire. Le gène est l'élément qui est à l'origine des caractères congénitaux de chaque individu et est le support de leur transmission de génération en génération. Ils sont susceptibles de subir des mutations.

Chez les organismes ***diploïdes*** (lorsque les chromosomes dans les cellules sont présentes par paires, un jeux provenant de la mère et un jeux du père), un gène donné est présenté avec deux copies, pas forcément identiques. L'***allèle*** est l'une des deux copies localisées au même locus sur chaque membre d'une paire de chromosomes homologues. Les allèles représentent aussi des variantes d'un même gène, comme le produit de mutations ou de changements dans leur séquence ordonnée de nucléotides.

Un individu donné peut être caractérisé comme **homozygote** (deux allèles identiques sur chaque locus), ou **hétérozygote** (deux allèles différents).

Comme nous avons pu le constater, un gène est caractérisé par la fonction qu'il joue dans le génome. Par ailleurs, plusieurs gènes peuvent se complémenter pour assurer une même fonction. Enfin, le mot gène est couramment utilisé pour désigner un locus ou un allèle.

2.1.2. Génome, génotype, phénotype, haplotype.

Le ***génome*** désigne l'ensemble de l'information héréditaire d'un organisme. Cette information est présente en totalité dans chacune des cellules de l'organisme. Lorsqu'une cellule se divise, l'information est copiée et transmise aux cellules filles.

Le génome contient toutes les instructions nécessaires au développement, au fonctionnement, au maintien de l'intégrité et à la reproduction des cellules et de l'organisme. Ces instructions sont les gènes.

Deux êtres vivants d'espèces différentes présentent des génomes qui diffèrent par leur taille ainsi que par le nombre, l'ordre et la nature des instructions qu'ils contiennent. Deux individus de la même espèce, au contraire, possèdent le même catalogue d'instructions, même si celles-ci peuvent exister dans des versions légèrement différentes, les allèles, (d'un individu à un autre ou chez un même individu, lorsque les copies héritées du père et de la mère diffèrent).

Le ***génotype*** est l'information génétique portée par les gènes d'un individu et l'expression visible ou fonctionnel du génotype est appelé le phénotype. Le génotype est la combinaison des deux exemplaires du gène présents dans le génome d'une cellule ou d'un organisme diploïde.

Le ***phénotype*** est la manifestation apparente de la constitution du génome sous la forme d'un trait morphologique, d'un syndrome clinique, d'une variation qualitative ou quantitative du produit final d'expression d'un gène.

Un ***haplotype*** défini l'ensemble des différents gènes présents et génétiquement liés sur un même chromosome. On peut remarquer ainsi qu'un génotype diploïde se compose de 2 haplotypes.

2.1.3. Le principe de méiose, le brassage génétique.

Le ***brassage génétique*** est réalisé à 2 niveaux, au sein de chaque individu et lors de la reproduction.

Chez les espèces eucaryotes, le brassage génétique intervient au cours de la reproduction sexuée des individus. Le brassage constitue la construction de nouvelles combinaisons génotypiques à partir d'un pool génétique donné. Le processus de la méiose permet ce brassage.

La figure 4 montre la construction d'un gamète d'un individu de la génération n après une reproduction entre deux individus de la génération n-1.

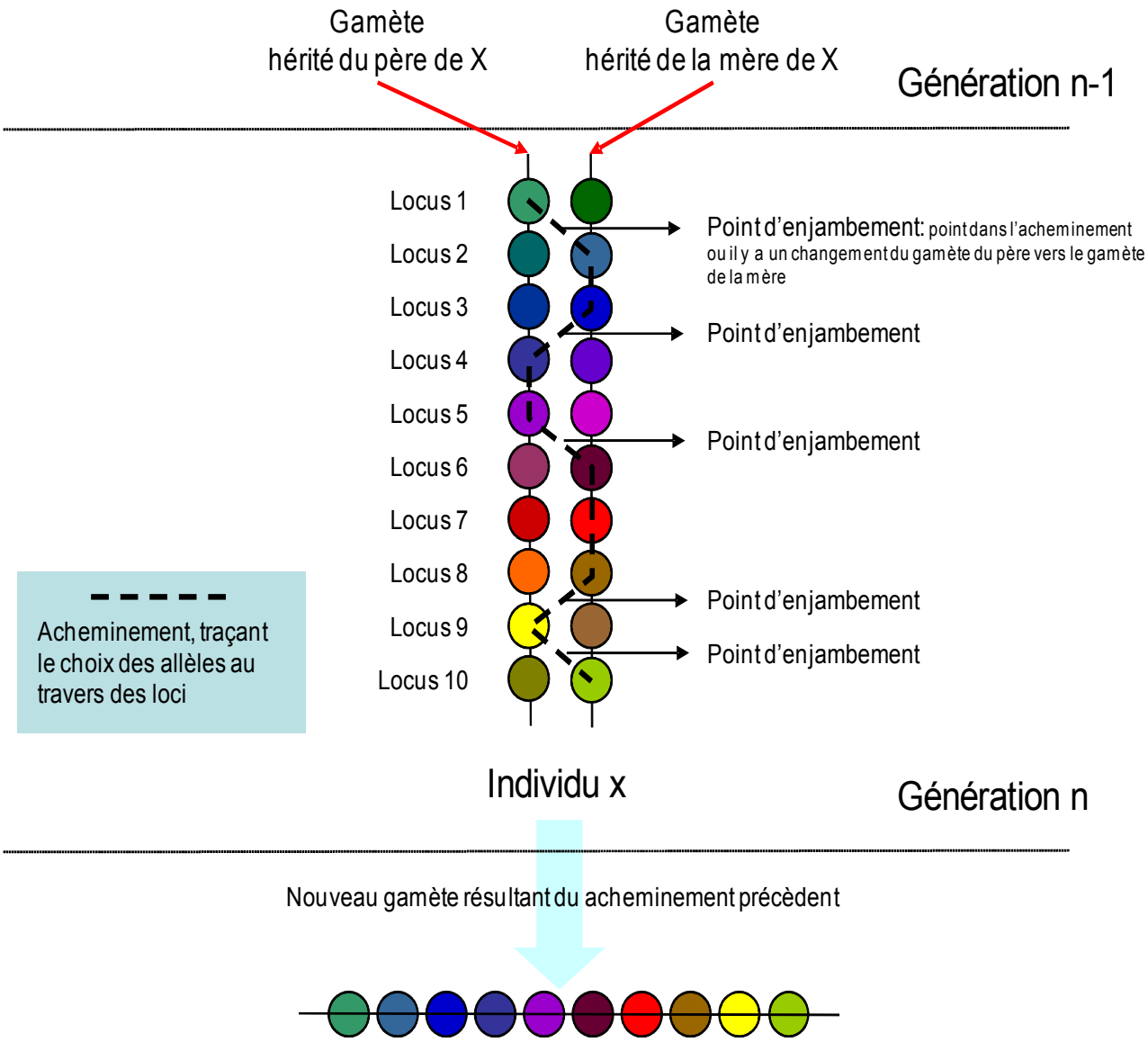


Figure 4 – Construction d'un gamète

Dans ce schéma, un individu X de la génération n contient un gamète hérité de son père et un gamète hérité de sa mère. L'acheminement, traçant le choix des allèles au travers des différents loci, est aléatoire. Ainsi, pour chaque loci, un allèle est choisit aléatoirement. Lorsqu'il y a un changement dans l'acheminement,passant du gamète hérité du père au gamète hérité de la mère, on appelle ce point **un point d'enjambement**.

La ***méiose*** se compose de deux divisions cellulaires successives. Elle suit une phase de réplication de l' ADN. Elle aboutit à partir d'une cellule-mère diploïde à la formation de 4 cellules-filles haploïdes. La méiose assure le passage de la phase diploïde à la phase haploïde, et a pour finalité la formation des gamètes, les cellules reproductives. La figure 5 montre ainsi comment la méiose permet de passer d'une cellule mère diploïde à 4 cellules filles haploïdes.

.

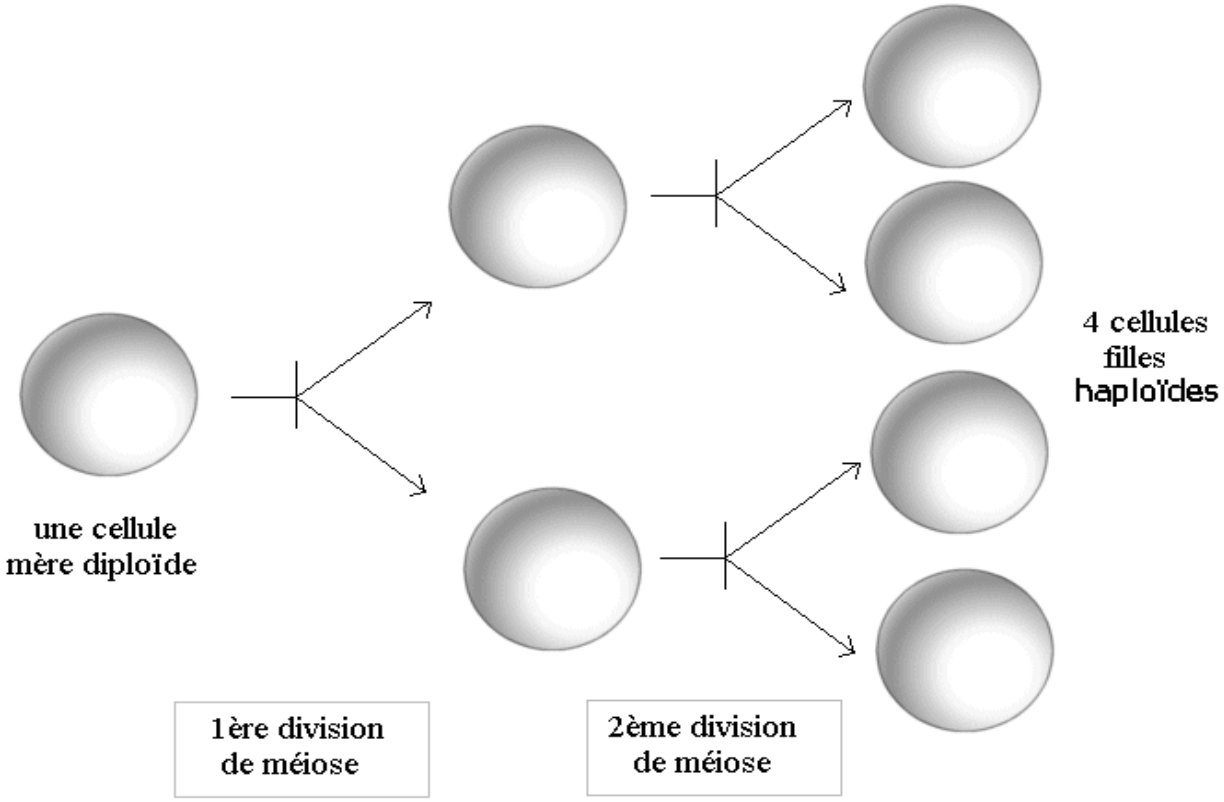


Figure 5 – Bilan succinct de la méiose

2.2. Le programme « simherit » :

2.2.1. Le principe :

La première partie de mon stage consistait à modifier un programme écrit en fortran90 par mon maître de stage Léopoldo Sanchez. Le programme initiale servait pour l'étude de la transmission de caractères quantitatives, ceux typiquement contrôlés par un grand nombre de loci, chacun ayant un petit effet sur le phénotype final. Le programme actuel conserve uniquement la partie dédié à l'hérédité, ou transmission des allèles, d'une génération à la suivante. Ce programme s'appelle « simherit ». Le but était de générer un pedigree pour ensuite l'utiliser pour comparer l'efficacité de divers logiciels permettant la reconstruction d' haplotypes. Je vais expliquer ici le principe de celui-ci.

Il faut d'abord savoir qu'un génotype est composé de deux haplotypes. Un haplotype représente l'information génétique contenue dans un des deux chromosomes homologues.

Tout d'abord, pour avoir des descendants, il faut commencer par construire les génotypes des fondateurs. On attribut pour chaque locus et pour chaque chromosome homologue un allèle (un nombre allant de 1 à 10 par exemple), lequel est échantillonné aléatoirement d'un pool génétique initial. Ce pool génétique est caractérisé par les fréquences des allèles à chaque locus. La figure 6 donne un exemple de pool génétique pour 2 locus et 10 allèles :

locus	allele	allele_freq
1	1	0.104167
1	2	0.095833
1	3	0.108333
1	4	0.079167
1	5	0.104167
1	6	0.075000
1	7	0.116667
1	8	0.075000
1	9	0.125000
1	10	0.116667
2	1	0.100000
2	2	0.116667
2	3	0.100000
2	4	0.091667
2	5	0.120833
2	6	0.125000
2	7	0.062500
2	8	0.083333
2	9	0.108333
2	10	0.091667

Figure 6 – Exemple de pool génétique.

La dernière colonne indique la fréquence d'apparition de l'allèle i pour le locus j. Par exemple, pour le locus 1, l'allèle 1 apparaît dans environ 10% des haplotypes.

Les génotypes des fondateurs sont ainsi construits aléatoirement. On obtient par exemple la figure 7.

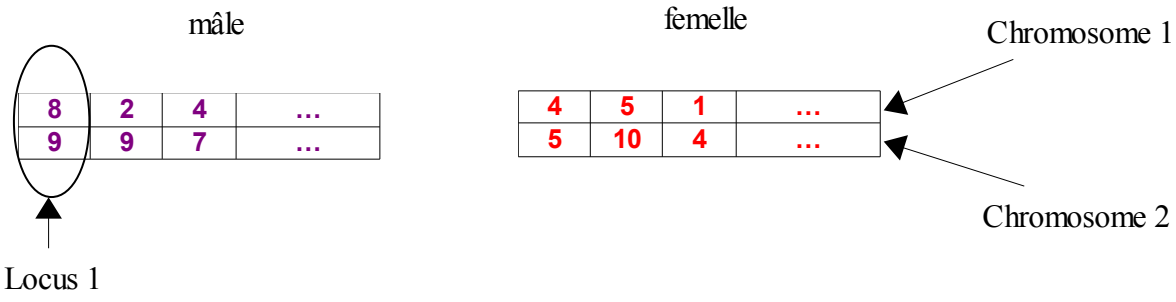


Figure 7 – Génotype des fondateurs.

Une fois que les génotypes des fondateurs sont construits, les génotypes des descendants peuvent être construits en simulant le processus de la méiose pour chaque parent fondateur. Les descendants sont ainsi le résultat de la fusion aléatoire de deux gamètes de deux parents différents (on assume une espèce dioïque, dont les fleurs unisexuées mâles et femelles sont portées par des individus différents) . Le choix des parents est aléatoire, ainsi que leurs gamètes pour la formation de chaque nouveau descendant. Ainsi, pour chaque descendant, on regarde le génotype du père et le génotype de la mère.

On parcourt chaque génotype, locus par locus, et pour chaque locus, le programme va choisir aléatoirement un des deux allèles : c'est le principe d'acheminement vu dans la figure 4. Cet allèle sera placé dans un vecteur qui, une fois remplie, représentera un gamète. On va ainsi obtenir deux gamètes (un gamète mâle et un gamète femelle) indiqués dans la figure 8.

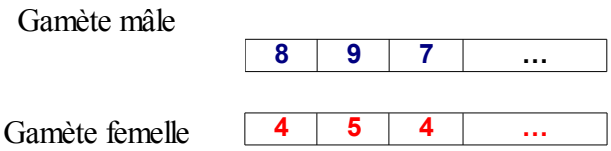


Figure 8 – Gamètes obtenus.

Ces deux gamètes vont alors être assemblés pour former le génotype du descendant.

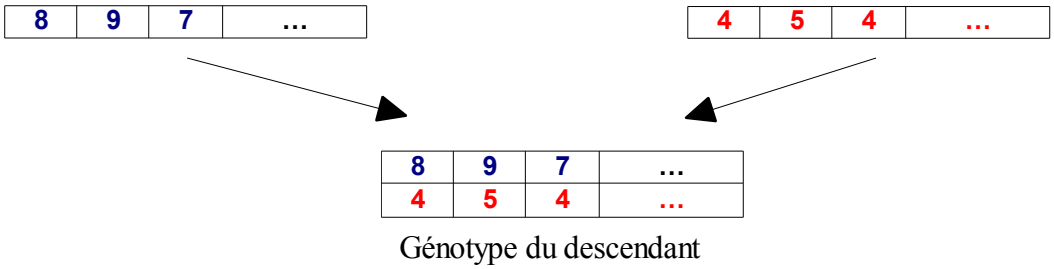


Figure 9 – Formation du génotype du descendant.

Parmi les descendants de cette première génération, le programme va aléatoirement déterminer lesquels seront des reproducteurs pour une nouvelle génération (le nombre de reproducteurs est égal au nombre de fondateurs). On génère ainsi plusieurs générations. La figure 10 montre un résumé de la simulation pour générer un pedigree.

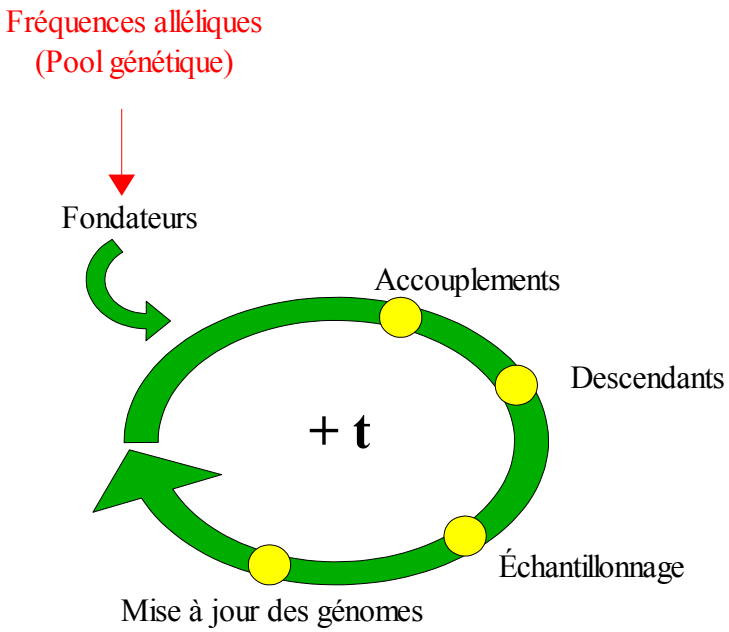


Figure 10 – Génération d'un pedigree

Ainsi, on part de n fondateurs, puis on fusionne aléatoirement deux gamètes de deux parents différents et on obtient ainsi m descendants où m > n. Puis nous faisons un échantillonnage, c'est à dire qu'on ne va garder qu'un certain nombre de descendants selon le **taux d'échantillonnage** choisit. Par exemple, si ce taux est de 0.2, on ne va garder que 20% des descendants, la sélection se faisant aléatoirement. Puis les génotypes des descendants sélectionnés seront utilisés pour former les génotypes des descendants de la génération suivante.

Pour avoir un fichier de sortie, on fait varier divers paramètres :

- le nombre de fondateurs (mâles et femelles) ;
- le nombre de descendants par génération (mâles et femelles) ;
- le nombre de génération ;
- le taux d'échantillonnage ;
- le nombre de loci (N.B. : un locus, des loci) ;
- le numéro des allèles (1 à 8 par exemple).

2.2.2. Exemple :

Le programme « simherit » produit 2 sorties. La première sortie est un tableau de la forme suivante :

self,dad,mum,gener,rank_off,rank_parsurv,sex

1	0	0	0	0	1	1	m
2	0	0	0	0	2	1	m
3	0	0	0	0	3	1	m
4	0	0	0	0	4	1	m
5	0	0	0	0	5	1	m
			...				
21	0	0	0	0	1	1	f
22	0	0	0	0	2	1	f
23	0	0	0	0	3	1	f
24	0	0	0	0	4	1	f
25	0	0	0	0	5	1	f
			...				
41	10	25	1	1	1	1	m
42	17	22	1	2	2	1	m
43	10	24	1	3	3	1	m
44	4	40	1	4	4	1	m
45	8	27	1	5	5	1	m
			...				
66	2	29	1	1	1	1	f
69	12	33	1	4	2	1	f
70	9	29	1	5	3	1	f
71	18	35	1	6	4	1	f
72	5	36	1	7	5	1	f

Figure 11 – Premier type de sortie avec le programme « simherit ».

- La première colonne est l'identifiant de l'individu ;
- La deuxième colonne est l'identifiant du père ;
- La troisième colonne est l'identifiant de la mère ;
- La quatrième colonne indique dans quelle génération on se trouve (la génération 0 indique les fondateurs) ;
- La cinquième colonne représente le rang du descendant dans la génération ;
- La sixième colonne représente le rang des fondateurs dans la génération ;
- La septième colonne indique si l'individu est un reproducteur ou non :
- La dernière colonne indique le sexe de l'individu.

La deuxième sortie que produit le programme « simherit » est montrée dans la figure 12. Ici, il n'y a que les génotypes des 5 premiers individus.

```
#1
1 1 6 2 2 1 4 5 8 4
7 2 7 5 6 6 8 6 2 6
#2
5 2 3 1 8 1 2 7 3 4
2 6 8 8 3 4 2 7 5 8
#3
7 2 5 7 3 2 6 3 4 5
8 5 4 3 8 8 1 6 2 1
#4
5 1 8 8 7 3 1 1 6 5
2 4 8 1 1 2 6 5 1 7
#5
3 4 2 6 1 5 2 3 3 3
6 1 6 6 1 7 4 8 6 1
```

Figure 12 – Deuxième type de sortie avec le programme « simherit ».

On fait par exemple une simulation avec 40 fondateurs (20 mâles et 20 femelles), on génère 3 générations de descendants et on ne considère que les 5 premiers locus. On a 187 individus au total.

On peut représenter les données sous la forme d'un arbre . La figure 13 représente une partie de ces données sur 3 générations.

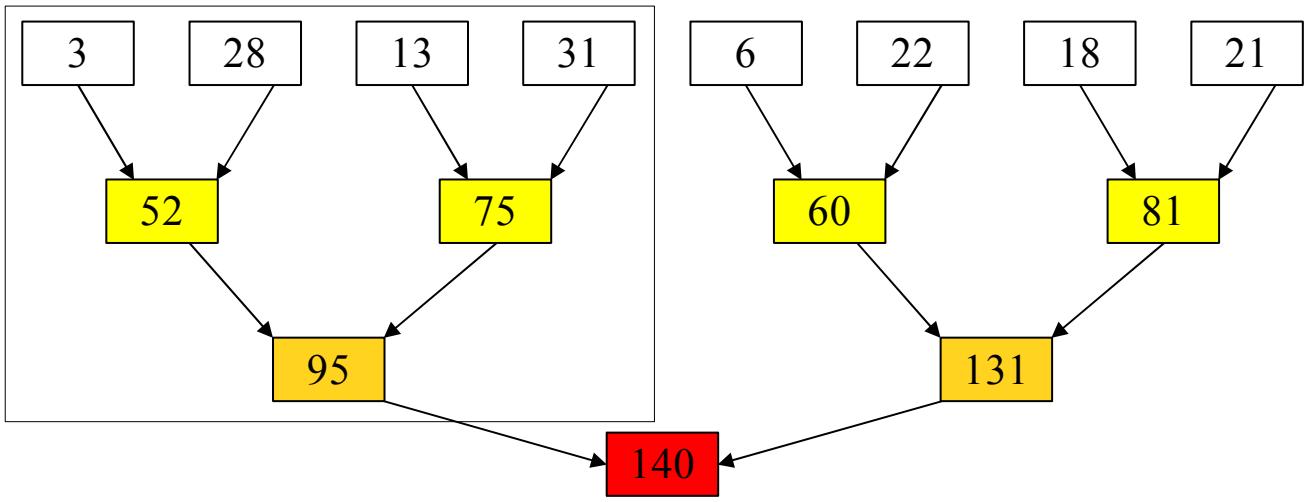


Figure 13 – Représentation d'une partie des données sur 3 générations.

Chaque ligne correspond à une génération, les blocs blancs correspondent aux numéros des fondateurs, les blocs jaunes correspondent aux numéros des individus de la première génération, les blocs oranges correspondent aux individus de la deuxième génération et les blocs rouges correspondent aux individus de la troisième génération. Dans cette figure, seulement quelques individus sont représentés dans le seul but d'atteindre un individu de la troisième génération. Ainsi, dans notre exemple, l'individu 140 est le descendant des individus 95 et 131.

On peut regarder le génotype des individus contenus dans le cadre, c'est à dire les individus 3, 28, 13, 31, 52, 75 et 95, pour vérifier qu'un individu de la génération (n+1) hérite bien de la moitié de l'information de son père et de sa mère qui sont issus de la génération n. Ces génotypes sont représentés dans les figures 14 et 15.

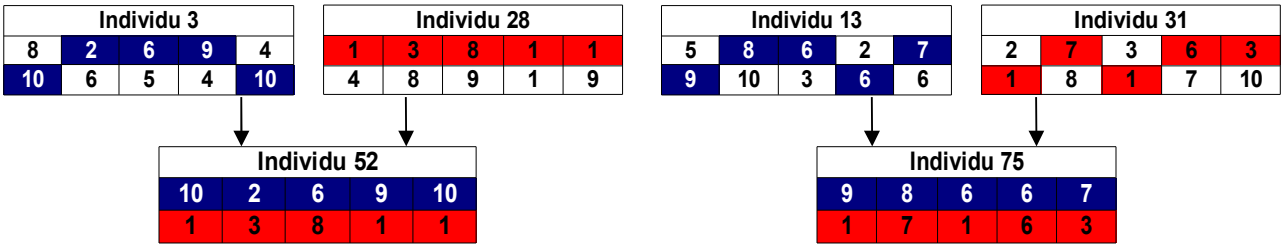


Figure 14 - Génotype pour les fondateurs et la génération 1.

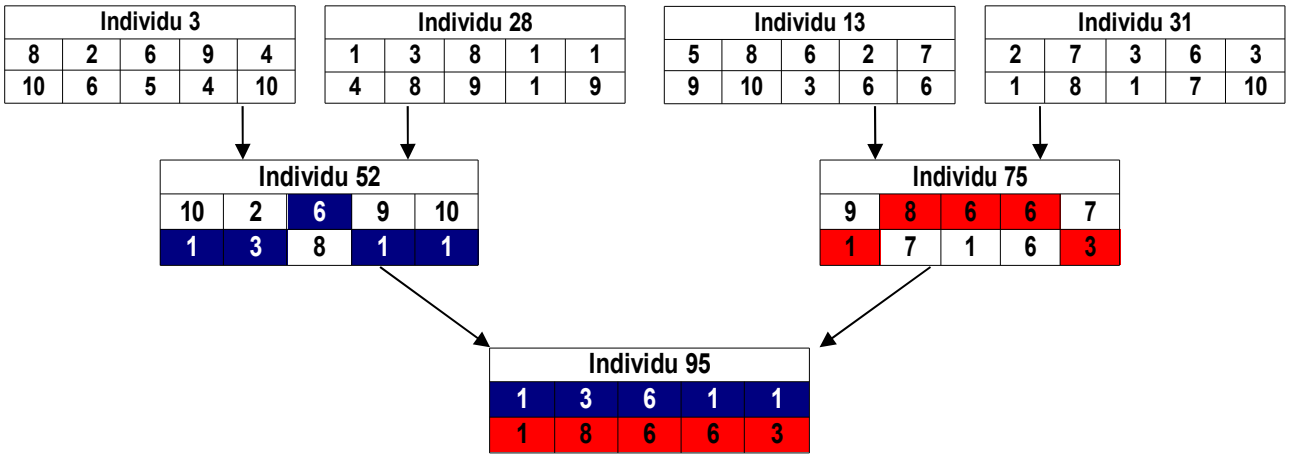


Figure 15 – Génotypes pour les générations 1 et 2.

Le gamète bleu correspond à ce qui a été hérité du père et le gamète rouge correspond à ce qui a été hérité de la mère. De même, chaque allèle sélectionné dans le génotype du père est en bleu, et chaque allèle sélectionné dans le génotype de la mère est en rouge.

Ainsi, un individu de la génération (n+1) hérite bien de la moitié de l'information de son père et de la moitié de l'information de sa mère.

2.3. Méthodes Statistiques :

Après avoir générer un pedigree, la seconde partie de mon stage consistait à utiliser cette simulation dans le but de comparer l'efficacité de plusieurs logiciels permettant la reconstruction d' haplotypes. Ces logiciels utilisent principalement les statistiques bayésiennes ou le principe de l'algorithme EM.

2.3.1. Statistiques Bayésiennes :

Le principal logiciel que j'ai utilisé pour la reconstruction des haplotypes utilise les statistiques bayésiennes. Ce logiciel s'appelle PHASE et sera explicité plus tard.

2.3.1.1. Notations et définitions :

L' ensemble des observations est noté $x = (x_1, x_2, \ldots, x_n)$.

En statistiques bayésiennes, on considère, en plus des données récoltées dans le cadre d'une expérience, un ***a priori*** sur le paramètre θ que l'on cherche à estimer. C'est le terme P(θ).
Cela peut permettre d'inclure dans les analyses des résultats précédents, formels ou non. En pratique, toute la difficulté consiste à estimer de manière correcte les a priori.

En statistiques bayésiennes, on va chercher non pas la « meilleure valeur » d'un paramètre, comme dans le maximum de vraisemblance, mais on va estimer la distribution de probabilité de ce paramètre.
On peut imaginer que l'on ne cherche pas à estimer la valeur du paramètre sur un cas, mais sur un grand ensemble de cas pour lesquelles les valeurs divergent. Si on fait suffisamment d'expériences, la distribution de probabilité du paramètre peut être tellement pointue qu'en pratique on considèrera une unique valeur.

On modélise l'information a priori au travers d'une loi de probabilité, appelée ***loi a priori*** et sa densité est notée π(θ). Grâce au théorème de Bayes, elle donne naissance à la ***loi a posteriori*** π(θ|x) qui tire toute l'information des données et de la loi a priori :

$$\pi(\theta|\mathbf{x}) = \frac{f(\mathbf{x}|\theta)\pi(\theta)}{\int_\Theta f(\mathbf{x}|\theta)\pi(\theta)d\,\theta}$$

où f(x|θ) est la loi des observations, c'est à dire la loi conditionnelle de x sachant θ :

$$f(\mathbf{x}|\theta) = \prod_{i=1}^{n} f(x_i|\theta).$$

Alors que la statistique classique repose sur la loi des observations, la statistique bayésienne repose sur la loi a posteriori. Cette loi peut s'interpréter comme un résumé de l'information disponible sur θ, une fois x observé.

Les différentes difficultés de l'approche bayésienne sont les suivante :

- Traduction des informations a priori ;
- Que faire en l'absence d'information a priori ? ;
- Difficultés numériques et algorithmiques pour calculer des approximations de cette loi a priori.

Dans le dernier cas, les approches possibles sont l'intégration numérique et les méthodes de simulation de Monte-Carlo : les **méthodes MCMC** (Monte Carlo Markov Chains)

2.3.1.2. Méthode MCMC :

Soit V = (V_1, …, V_j, …, V_k), où k > 1, un vecteur aléatoire de densité f(v). On souhaite calculer E[h(V)] où h(V) est un réel. On suppose que l'on ne sait pas simuler selon la loi de V. Le principe des méthodes de simulation de Monte-Carlo par chaîne de Markov (MCMC) est de générer une chaîne de Markov ($V^{(l)}$; l > 0) qui soit asymptotiquement distribuée selon la loi de V, puis d'approcher E[h(V)] :

$$\frac{1}{L}\sum_{l=1}^{L} h(\mathbf{V}^{(l)}) \overset{L\to+\infty}{\longrightarrow} E[h(\mathbf{V})].$$

On précise que $V^{(l)}$ est un vecteur de la même dimension que V. Les problèmes des méthodes MCMC concernent le contrôle de la convergence :

- A partir de combien d'itérations chaîne de Markov a-t-elle atteint son régime limite ?
- Combien d'itérations sont ensuite nécessaires pour obtenir une bonne approximation de la loi a posteriori visée ?

La méthode la plus populaire pour simuler une chaîne de Markov est l'**échantillonnage de Gibbs**.

2.3.1.3. L' échantillonnage de Gibbs :

L'échantillonneur de Gibbs est la technique MCMC la plus simple. Sa popularité date de l'application de Geman et Geman (1984). L'expression « échantillonneur de Gibbs » vient de l'utilisation que Geman et Geman ont fait de la distribution de Gibbs pour modéliser les images satellites, mais son applicabilité est beaucoup plus générale.

On suppose que k > 1, où k est la dimension de V. On part de $V^{(0)} = (V_1^{(0)}, \dots, V_j^{(0)}, \dots, V_k^{(0)})$ arbitraire.

A létape (l) de l'algorithme de Gibbs, on simule le vecteur $V^{(l)} = (V_j^{(l)} ; j = 1, \dots, k)$ comme suit :

- Simulation de $V_1^{(l)}$:

$$V_1^{(l)} \sim \text{Loi}(V_1 | V_2 = v_2^{(l-1)}, \dots, V_k = v_k^{(l-1)})$$

- Simulation de $V_j^{(l)}$:

$$V_j^{(l)} \sim \text{Loi}(V_j | V_1 = v_1^{(l)}, ..., V_{j-1} = v_{j-1}^{(l)}, V_{j+1} = v_{j+1}^{(l-1)}, ..., V_k = v_k^{(l-1)})$$

- Simulation de $V_k^{(l)}$:

$$V_k^{(l)} \sim \text{Loi}(V_k | V_1 = v_1^{(l)}, \ldots, V_{k-1} = v_{k-1}^{(l)})$$

Notons que cet algorithme décrit une chaîne markovienne de premier ordre, puisque la distribution conditionnelle d'un tirage dépend de la réalisation précédente.

2.3.2. Algorithme EM :

D'autres logiciels pour la reconstruction d' haplotypes utilisent le principe de l'algorithme EM. Je n'ai pas eu le temps d'exploiter ces logiciels, ils seront expliquer brièvement plus tard.

Soit G l'ensemble des génotypes et H l'ensemble des haplotypes possibles. Soit f_k la fréquence de l'haplotype h_k, k = 1..M, alors la vraisemblance s'écrit :

$$L(f_1, ..., f_M) = Pr(G \mid H) = \prod_{i=1..M} Pr(g_i \mid f_1, ... f_M)$$

Où la probabilité conditionnelle d'un génotype i à s sites hétérozygotes est défini par :

$$Pr(g_i \mid f_1, ..., f_M) = \sum_{j=1..2^{s-1}} Pr(d_{ij} \mid f_1, ..., f_M)$$

Correspondant à la probabilité d'obtenir le génotype g_i sachant qu'on connaît les fréquences des haplotypes h_1, …, h_M.

Soit $d_{ij} = (h_l, h_k)$ représentant un diplotype , et si P respecte l'équilibre de Hardy Weinberg, la probabilité d'obtenir le diplotype d_{ij} est donnée par :

$$Pr(d_{ij}) = \delta_{lk} f_l f_k \quad \text{avec } \delta_{lk} = \begin{cases} 2 \text{ si } l \neq k \\ 1 \text{ si } l = k \end{cases}$$

L'algorithme EM fait 2 étapes successives de manière itérative, l'**étape E** (*Espérance*) puis l'**étape M** (*Maximisation*).

On initialise les fréquences des haplotypes aléatoirement : $fk^{(0)}$.

L'algorithme EM procède de la manière suivante :

- Dans l'étape E, connaissant les fréquences des haplotypes h_1, …, h_M , l'algorithme va d'abord calculer la probabilité d'avoir le diplotype d_{ij} . Cette probabilité est notée $Pr(d_{ij})$.
- Dans l'étape M, l'algorithme recalcule cette probabilité en la divisant par la somme des probabilités $Pr(d_{ij})$ avec j variant de 1 à 2^{s-1}. Enfin, les fréquences des haplotypes sont recalculées avec la nouvelle valeur de $Pr(d_{ij})$ et ces fréquences seront ainsi utilisées dans la prochaine itération.

La figure 16 montre un schéma résumant cette méthode.

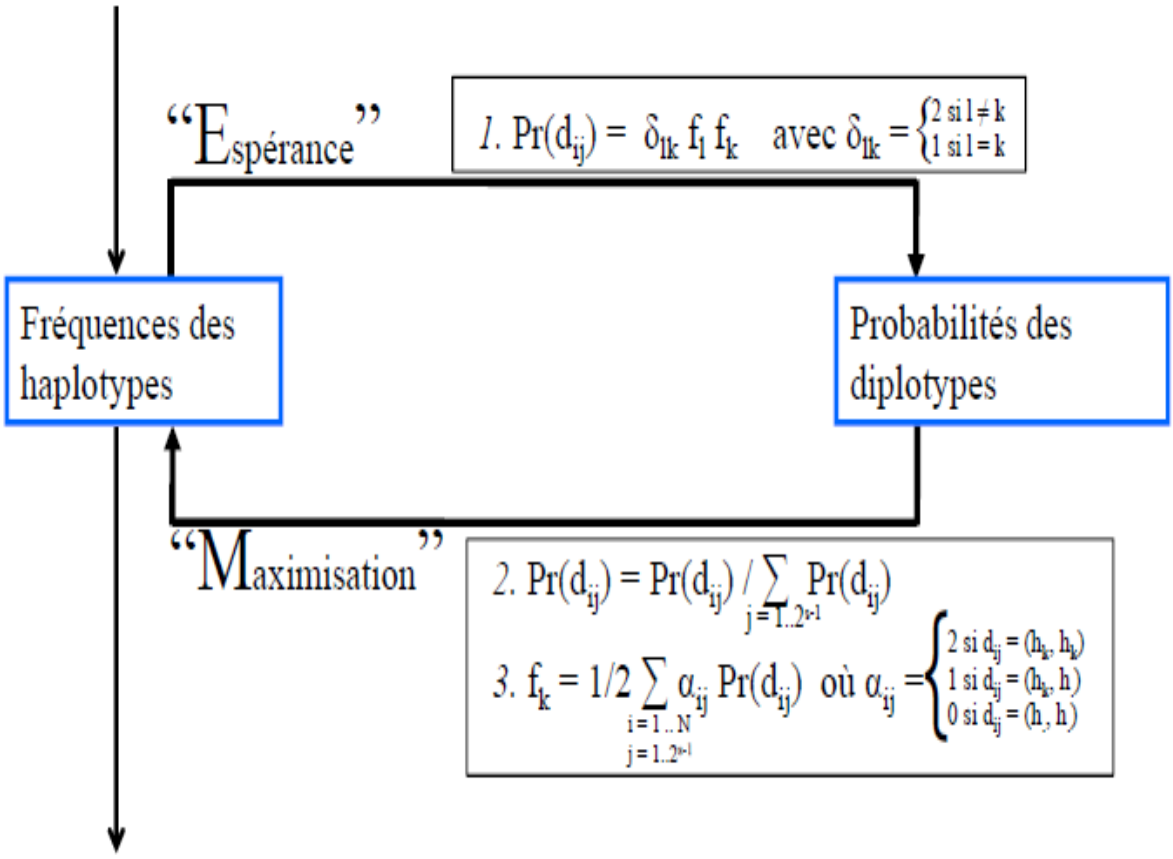


Figure 16 – Résumé de la méthode EM.

Dès que $f_k^{(t+1)} \approx f_k^{(t)}$ on aura atteint un maximum de vraisemblance.

3. Comparaison de logiciels pour la reconstruction d' haplotypes :

La deuxième partie de mon stage consistait à utiliser le pedigree simulé avec le programme écrit en FORTRAN pour pouvoir comparer plusieurs logiciels de reconstruction d' haplotypes. Cependant, par manque de temps, je n'ai pu utiliser qu'un logiciel pour le moment : PHASE. D'autres logiciels seront quand même énoncés et brièvement expliqués et je tenterais de faire une comparaison selon la documentation que j'ai pu lire durant mon stage.

3.1. Le Logiciel PHASE :

J'ai donc surtout utilisé le logiciel PHASE pour la reconstruction d'haplotypes.

3.1.1. Principe

Le programme PHASE est une implémentation de la méthode Bayésienne pour la reconstruction d' haplotypes.

On commence avec un échantillon de n individus diploïdes provenant d'une population (chacun ayant reçu un chromosome de chaque parent). On connaît les génotypes $G = \{G_1, \ldots, G_n\}$ correspondant aux paires d'haplotypes inconnus $H = \{H_1, \ldots, H_n\}$. On connaît un ensemble de fréquences haplotypiques inconnues de la population $F = \{F_1, \ldots F_M\}$ et on connaît un ensemble de fréquences haplotypiques échantillonnées $f = \{f_1, \ldots, f_M\}$ où M est le nombre d'haplotypes possibles pour l'échantillon. Les M haplotypes possibles sont notés arbitrairement de 1 à M.

Cette méthode PHASE regarde les haplotypes non résolus comme des quantités aléatoires non observées et vise à évaluer leur distribution conditionnelle donnant l'information qui peut être obtenue depuis les données génotypiques connues.
PHASE utilise un algorithme d'échantillonnage de Gibbs qui est un type d'algorithme MCMC (Markov Chain Monte Carlo) pour obtenir un échantillon approximatif de la distribution a priori de P(H|G).

On commence avec $H^{(0)} = \{H_1^{(0)}, \ldots, H_n^{(0)}\}$ choisit arbitrairement. On veut obtenir $H^{(t+1)}$ depuis $H^{(t)}$ pour $t = 0, 1, 2, \ldots$.

L'algorithme est le suivant :

1 - Un individu i est choisi aléatoirement parmi tous les individus ambigus (c'est à dire les individus qui ont plus d'un haplotype possible).

2 - Un sous-ensemble S des loci ambigus de l'individu i est choisi pour être mis à jour. Soit H(S) représentant l'information haplotypique pour l'individu i au loci ambigu S, et soit H(-S) le complémentaire de H(S), c'est à dire l'information haplotypique de tous les autres individus sans l'individu i.
On échantillonne $H^{(t+1)}(S)$ depuis $P[H(S)|G,H^{(t)}(-S)]$.

3 - $H^{(t+1)}(-S) = H^{(t)}(-S)$

A chaque itération, il est nécessaire de mettre à jour P[H(S)|G,$H^{(t)}$(-S)].

Pour H(S) consistant avec l'information génotypique G, la distribution conditionnelle est :

$$\begin{aligned} Pr[H(S)|G, H^{(t)}(-S)] &\propto Pr(H_i|H_{-i}) \\ &\propto \pi(h_{i1}|H_{-i})\pi(h_{i2}|H_{-i}, h_{i1}) \end{aligned}$$

C'est équivalent à la distribution conditionnelle pour la paire d' haplotypes $H_i = \{h_{i1}, h_{i2}\}$, consistante avec les génotypes G_i où $\pi(.|H)$ est la distribution conditionnelle d'un futur haplotype échantillonné, donnant un ensemble H d' haplotypes échantillonnés précédemment. H_{-i} est l'ensemble des haplotypes excluant l'individu i.

La clé de la logique de l'algorithme PHASE est que les haplotypes non résolus ont tendance à être semblables aux haplotypes connus, et la façon dont l'estimation a priori est calculée en utilisant notamment la théorie de coalescence. La théorie de coalescence est une approche classique rétrospectif de la génétique des populations. Sur la base d'un échantillon d'individus, cette approche reconstruit rétrospectivement le partage de polymorphismes génétiques entre lignages, et infère les fondations à la base du pedigree.

3.1.2. Exemple

Durant mon stage, j'ai utilisé la version 2.1 de PHASE. Ce logiciel peut être téléchargé à l'adresse suivante : http://www.stat.washington.edu/stephens/software.html

Pour analyser nos propres données, on doit préparer un fichier d'entrée dans un format approprié avant d'exécuter le programme. Ainsi, l'utilisateur doit spécifier le nombre d'individus à analyser, le nombre de locus, le type de locus (SNP ou microsatellite) et le génotype de chaque individu.

Pour étudier le fonctionnement de PHASE, nous pouvons prendre un exemple avec :

- 40 fondateurs (20 mâles et 20 femelles) ;
- 200 descendants par génération (100 mâles et 100 femelles) ;
- 1 seule génération ;
- 10 locus ;
- 2 possibilités d'allèles (1 ou 2).

On a ainsi 240 individus.

La figure 17 montre un fichier d'entrée avec les 5 premiers génotypes parmi les 240 génotypes :

```
240
10
P 1 2 3 4 5 6 7 8 9 10
MMMMMMMMMM
#1
 1 1 2 1 1 1 1 2 2 1
 2 1 2 2 2 2 2 2 1 2
#2
 2 1 1 1 2 1 1 2 1 1
 1 2 2 2 1 1 1 2 2 2
#3
 2 1 2 2 1 1 2 1 1 2
 2 2 1 1 2 2 1 2 1 1
#4
 2 1 2 2 2 1 1 1 2 2
 1 1 2 1 1 1 2 2 1 2
#5
 1 1 1 2 1 2 1 1 1 1
 2 1 2 2 1 2 1 2 2 1
```

Figure 17 – Exemple de fichier d'entrée PHASE.

La première ligne indique le nombre d'individus, la seconde indique le nombre de locus, la troisième indique la position des différents locus et il ne faut pas oublier de mettre le caractère 'P' à titre indicatif. La quatrième ligne ne contient que des S et des M, S si le locus correspondant est biallélique, M pour un locus multiallélique. Il faut autant de caractères M ou S que de locus. Puis on marque les génotypes des individus.

Puis, on a amélioré le programme « simherit » pour qu'il produise une sortie directement au bon format (cf l'annexe 1).

Le fichier étant au bon format, on peut lancer PHASE et on obtient un fichier de sortie donnant ces principales informations :

- Une liste des haplotypes appartenant à la « meilleure » reconstruction, avec un résumé des fréquences avec laquelle chaque haplotype apparaît dans la « meilleure » reconstruction. La figure 18 ne montre que les 10 premiers haplotypes appartenant à cette liste.

```
BEGIN LIST_SUMMARY
   1 1 1 1 1 1 1 1 1 1 1   2.000000
   2 1 1 1 1 1 1 1 2 1 2   1.000000
   3 1 1 1 1 1 1 1 2 2 2   1.000000
   4 1 1 1 1 1 1 2 1 1 1   1.000000
   5 1 1 1 1 1 1 2 1 1 2   1.000000
   6 1 1 1 1 1 1 2 1 2 2   1.000000
   7 1 1 1 1 1 2 1 1 1 1   6.000000
   8 1 1 1 1 1 2 1 1 1 2   1.000000
   9 1 1 1 1 1 2 1 1 2 2   2.000000
  10 1 1 1 1 1 2 1 2 1 1   2.000000
```

Figure 18 – Liste des haplotypes appartenant à la meilleure reconstruction.

- Un résumé des meilleures reconstructions. Ce résumé indique un couple d' haplotypes pour chaque individu. La figure 19 n'indique que les couples d' haplotypes (formant ainsi un génotype) pour les 10 premiers individus.

```
BEGIN BESTPAIRS_SUMMARY
#1: (43,164)
#2: (109,135)
#3: (177,191)
#4: (36,161)
#5: (39,145)
#6: (8,163)
#7: (69,177)
#8: (33,95)
#9: (90,133)
#10: (123,187)
```

Figure 19 – Liste des couples d' haplotypes dans la meilleure reconstruction.

- Une liste des meilleures propositions de génotype pour chaque individu, avec des () aux positions où la phase a été difficile à déduire, et des [] autour des allèles qui ont été difficiles à déduire. La figure 20 ne montre que les meilleures propositions de génotype pour les 5 premiers individus.

```
BEGIN BESTPAIRS1
0 #1
(1) 1 2 (2) (1) (2) (1) 2 (1) (1)
(2) 1 2 (1) (2) (1) (2) 2 (2) (2)
0 #2
(1) (2) (2) (2) (1) 1 1 2 (2) (2)
(2) (1) (1) (1) (2) 1 1 2 (1) (1)
0 #3
2 (1) (2) (2) (2) (1) (2) (2) 1 (2)
2 (2) (1) (1) (1) (2) (1) (1) 1 (1)
0 #4
(1) 1 2 (2) (1) 1 (1) (2) (2) 2
(2) 1 2 (1) (2) 1 (2) (1) (1) 2
0 #5
(1) 1 (2) 2 1 2 1 (1) (1) 1
(2) 1 (1) 2 1 2 1 (2) (2) 1
```

Figure 20 – Liste des meilleures propositions de génotype.

On peut également obtenir des informations complémentaires, par exemple :

- Une liste indiquant une estimation de la fréquence de chaque haplotype échantillonné ainsi qu'une estimation de l'écart type pour ces fréquences. La figure 21 montre une estimation des 10 premiers haplotypes.

index	haplotype	E(freq)	S.E
1	1 1 1 1 1 1 1 1 1 1 1 1	0.002211	0.001900
2	1 1 1 1 1 1 1 1 1 1 1 2	0.002040	0.001847
3	1 1 1 1 1 1 1 1 1 1 2 1	0.001163	0.001507
4	1 1 1 1 1 1 1 1 1 1 2 2	0.001345	0.001555
5	1 1 1 1 1 1 1 1 2 1 1 1	0.001193	0.001597
6	1 1 1 1 1 1 1 1 2 1 2 2	0.000983	0.001447
7	1 1 1 1 1 1 1 1 2 2 1 1	0.001169	0.001224
8	1 1 1 1 1 1 1 1 2 2 2 2	0.001672	0.001654
9	1 1 1 1 1 1 1 2 1 1 1 1	0.001084	0.001499
10	1 1 1 1 1 1 1 2 1 1 2 2	0.001018	0.001308

Figure 21 – Liste des estimations de la fréquence de chaque haplotypes.

- Une liste des paires d' haplotypes les plus probables pour chaque individu., ainsi que les probabilités correspondantes. La figure 22 montre une listes des 10 premières paires d' haplotypes pour l'individu 1.

index	haplotype	E(freq)	S.E
1	1 1 1 1 1 1 1 1 1 1 1 1	0.002211	0.001900
2	1 1 1 1 1 1 1 1 1 1 1 2	0.002040	0.001847
3	1 1 1 1 1 1 1 1 1 1 2 1	0.001163	0.001507
4	1 1 1 1 1 1 1 1 1 1 2 2	0.001345	0.001555
5	1 1 1 1 1 1 1 1 2 1 1 1	0.001193	0.001597
6	1 1 1 1 1 1 1 1 2 1 2 2	0.000983	0.001447
7	1 1 1 1 1 1 1 1 2 2 1 1	0.001169	0.001224
8	1 1 1 1 1 1 1 1 2 2 2 2	0.001672	0.001654
9	1 1 1 1 1 1 1 2 1 1 1 1	0.001084	0.001499
10	1 1 1 1 1 1 1 2 1 1 2 2	0.001018	0.001308

–

Figure 22 – Liste des paires d 'haplotypes les plus probables.

3.1.3 Simulation

Dans notre simulation, on a généré plusieurs pedigree en faisant varier plusieurs critères :

- Le nombre de fondateurs mâles et femelles : 20 ou 60 ;
- Le nombre de descendants mâles ou femelles : 25, 40, 100 (s'il y a 20 fondateurs), 75, 120, 300 (s'il y a 60 fondateurs). Ce nombre dépend du taux d'échantillonnage ;
- Le taux d'échantillonnage : 0.2 , 0.5 ou 0.8 ;
- Le nombre possibles d'allèles différents : 2, 4, 8 ou 16.

Le nombre de locus reste inchangé (10) ainsi que le nombre de générations (1). On a ainsi 24 cas différents qui sont résumés dans la figure 23.

cas	nb_de_générations	fondateurs_mâle	fondateur_ femelle	descendant_ mâle	descendant_femelle	taux_échantillonnage	nb_locus	nb_allèle
1	1	20	20	100	100	0,2	10	2
2	1	20	20	100	100	0,2	10	4
3	1	20	20	100	100	0,2	10	8
4	1	20	20	100	100	0,2	10	16
5	1	60	60	300	300	0,2	10	2
6	1	60	60	300	300	0,2	10	4
7	1	60	60	300	300	0,2	10	8
8	1	60	60	300	300	0,2	10	16
9	1	20	20	40	40	0,5	10	2
10	1	20	20	40	40	0,5	10	4
11	1	20	20	40	40	0,5	10	8
12	1	20	20	40	40	0,5	10	16
13	1	60	60	120	120	0,5	10	2
14	1	60	60	120	120	0,5	10	4
15	1	60	60	120	120	0,5	10	8
16	1	60	60	120	120	0,5	10	16
17	1	20	20	25	25	0,8	10	2
18	1	20	20	25	25	0,8	10	4
19	1	20	20	25	25	0,8	10	8
20	1	20	20	25	25	0,8	10	16
21	1	60	60	75	75	0,8	10	2
22	1	60	60	75	75	0,8	10	4
23	1	60	60	75	75	0,8	10	8
24	1	60	60	75	75	0,8	10	16

Figure 23 – Résumé des cas pour la simulation.

Une simulation comporte 24 cas et nous avons fait 8 simulations. Nous avons donc générer 192 pedigrees différents.

Nous voulions faire varier le nombre de locus (10 et 50) et le nombre de générations (1, 4 et 7) mais PHASE utilise déjà beaucoup de temps pour donner une sortie avec les cas les plus simples, c'est à dire avec 10 locus et une seule génération.

Pour pouvoir utiliser le logiciel PHASE, on a créé un programme de simulation qui génère un pedigree, ce programme s'appelle « simherit » et a été expliqué dans une partie précédente. Il a fallu adapter la sortie que donnait ce programme pour pouvoir avoir une sortie avec PHASE.

3.1.4 Résultats

Dans cette partie, nous allons étudier l'efficacité du logiciel PHASE en analysant 4 types d'erreur :

- **tx_erreur_phase** : On compte le nombre de locus où PHASE a eu du mal à faire une déduction, c'est à dire qu'on compte le nombre de locus dont les allèles sont entre parenthèses, c'est donc une erreur potentielle. Puis on divise ce nombre par le nombre total de locus dans le pedigree ;
- **tx_reel_erreur** : On compte le nombre d'erreurs réelles, c'est à dire qu'on compte le nombre de changement d'allèle entre le fichier d'entrée et le fichier de sortie que donne PHASE. Puis on divise ce nombre par le nombre total d'allèles dans le pedigree ;

- **tx_erreur_type1** : On compte le nombre de fois où PHASE annonçait une erreur potentielle (donnée entre parenthèses) alors qu'il n'y en a pas en réalité, puis on divise ce nombre par le nombre total d'allèles dans le pedigree ;
- **tx_erreur_type2** : On compte le nombre de fois où PHASE n'a pas détecté d'erreur alors qu'il y en a une en réalité, puis on divise ce nombre par le nombre total d'allèles dans le pedigree.

Pour calculer ces 4 types d'erreurs, j'ai créé un programme fortran qui s'appelle « comparaison ». Ce programme est décrit dans l'annexe 2. Par exemple, pour la première simulation, les résultats obtenus après exécution du programme « comparaison » sont indiqués dans la figure 24.

cas n°	tx_erreur_phase	tx_reel_erreur	tx_erreur_type1	tx_erreur_type2
1	0,52	0,27	0,25	0
2	0,43	0,4	0,19	0,17
3	0,13	0,45	0,07	0,38
4	0,16	0,68	0,08	0,6
5	0,5	0,25	0,25	0
6	0,47	0,38	0,23	0,14
7	0,22	0,44	0,11	0,33
8	0,13	0,7	0,05	0,63
9	0,51	0,26	0,26	0
10	0,47	0,38	0,25	0,16
11	0,17	0,45	0,08	0,37
12	0,16	0,68	0,07	0,59
13	0,48	0,23	0,25	0
14	0,67	0,35	0,35	0,03
15	0,19	0,43	0,1	0,33
16	0,14	0,72	0,06	0,64
17	0,49	0,28	0,22	0
18	0,44	0,39	0,23	0,18
19	0,16	0,46	0,08	0,38
20	0,14	0,75	0,07	0,7
21	0,5	0,24	0,26	0
22	0,56	0,37	0,28	0,08
23	0,13	0,42	0,07	0,36
24	0,11	0,73	0,05	0,68

Figure 24 – Les 4 types d'erreurs pour la première simulation.

Puis, on cherche à savoir quels paramètres ont une influence sur chaque taux d'erreurs pour cette ***première simulation*.** Pour cela, on trace un graphique de chaque type d'erreurs en fonction du nombre de fondateurs (20 ou 60), du nombre d'allèles (2, 4, 8 ou 16) et du taux d'échantillonnage (0.2, 0.5 ou 0.8). On utilise ainsi la fonction **coplot** de R où :

X=as.factor(nb_allèle)
Y=as.factor(fondateurs_mâle)
Z=taux_échantillonnage

- Etude de « tx_erreur_phase » :

Ce taux correspond aux erreurs potentielles, là où PHASE a eu du mal à faire une déduction. Dans R, on entre :

coplot(tx_erreur_phase~Z|X*Y,data=tableau,panel=panel.smooth)

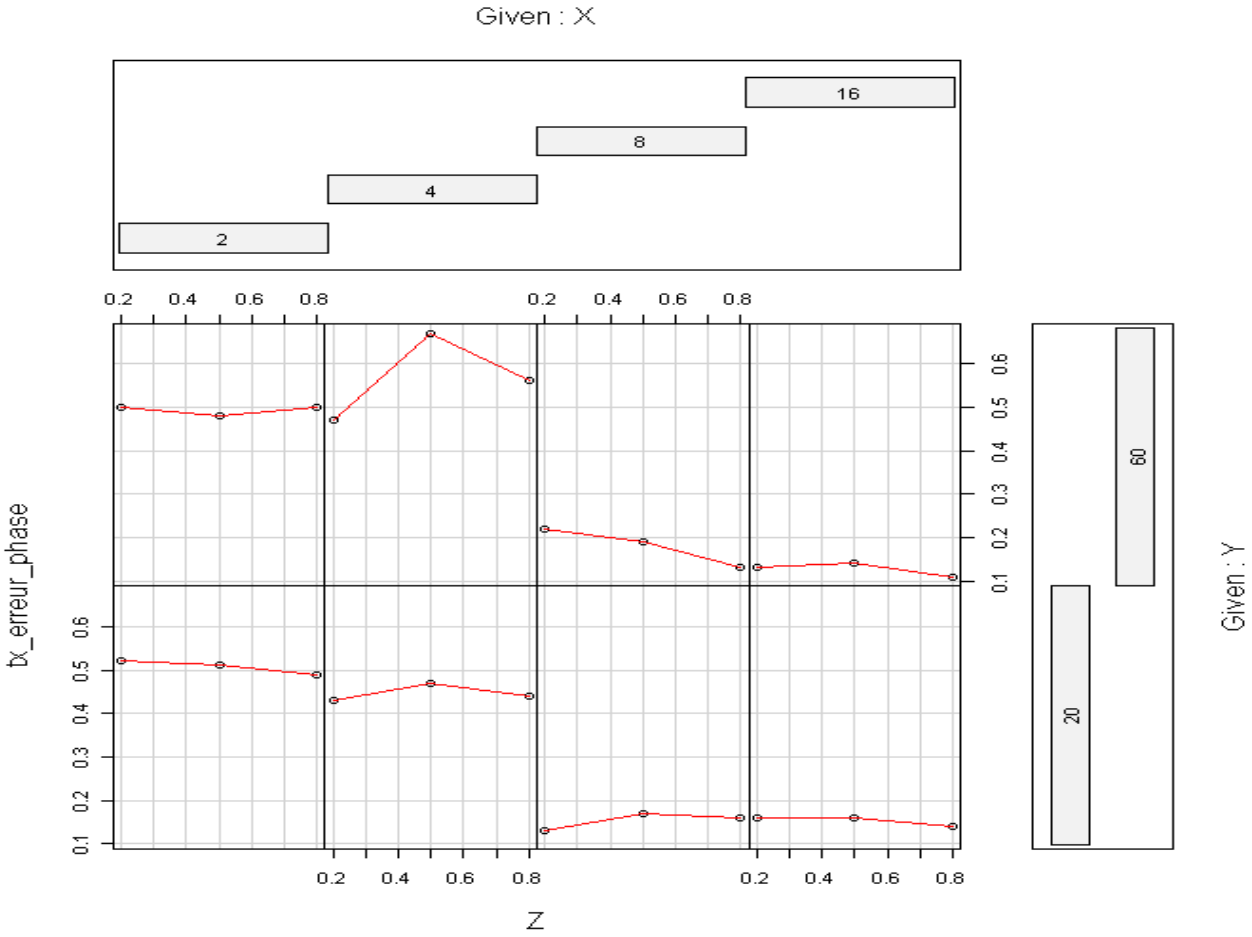


Figure 25 – Graphique « tx_erreur_phase » pour la première simulation.

Pour ce taux d'erreur, on remarque que le nombre de fondateurs et le taux d'échantillonnage ne semblent avoir aucunes incidences sur ce taux d'erreurs à l'inverse du nombre d'allèles qui semble avoir un rôle important.

Pour un polymorphisme avec 2 ou 4 allèles, le taux d'erreurs moyen est de 0.5 alors que pour un polymorphisme de 8 ou 16 allèles, le taux d'erreurs moyen est de 0.15. Ainsi, plus le nombre d'allèles est grand, et plus ce taux d'erreurs sera faible.

Enfin, nous remarquons que, pour un polymorphisme avec 2 ou 4 allèles, le taux d'erreurs est assez élevé : de l'ordre de 50 % en moyenne et avec un pic proche de 70% dans le cas de 60 fondateurs, un taux d'échantillonnage de 0.5 et un polymorphisme avec 4 allèles. Cependant, dans la littérature, il n'y a pas eu d'étude sur ce type d'erreurs, il n'est donc pas possible de savoir si ce taux est cohérent.

- Etude de « tx_reel_erreur »

Ce taux correspond au nombre d'erreurs réelles, c'est à dire au nombre de changement d'allèle entre le fichier d'entrée et le fichier de sortie de PHASE. Dans R, on entre :

```
coplot(tx_reel_erreur~Z|X*Y,data=tableau,panel=panel.smooth)
```

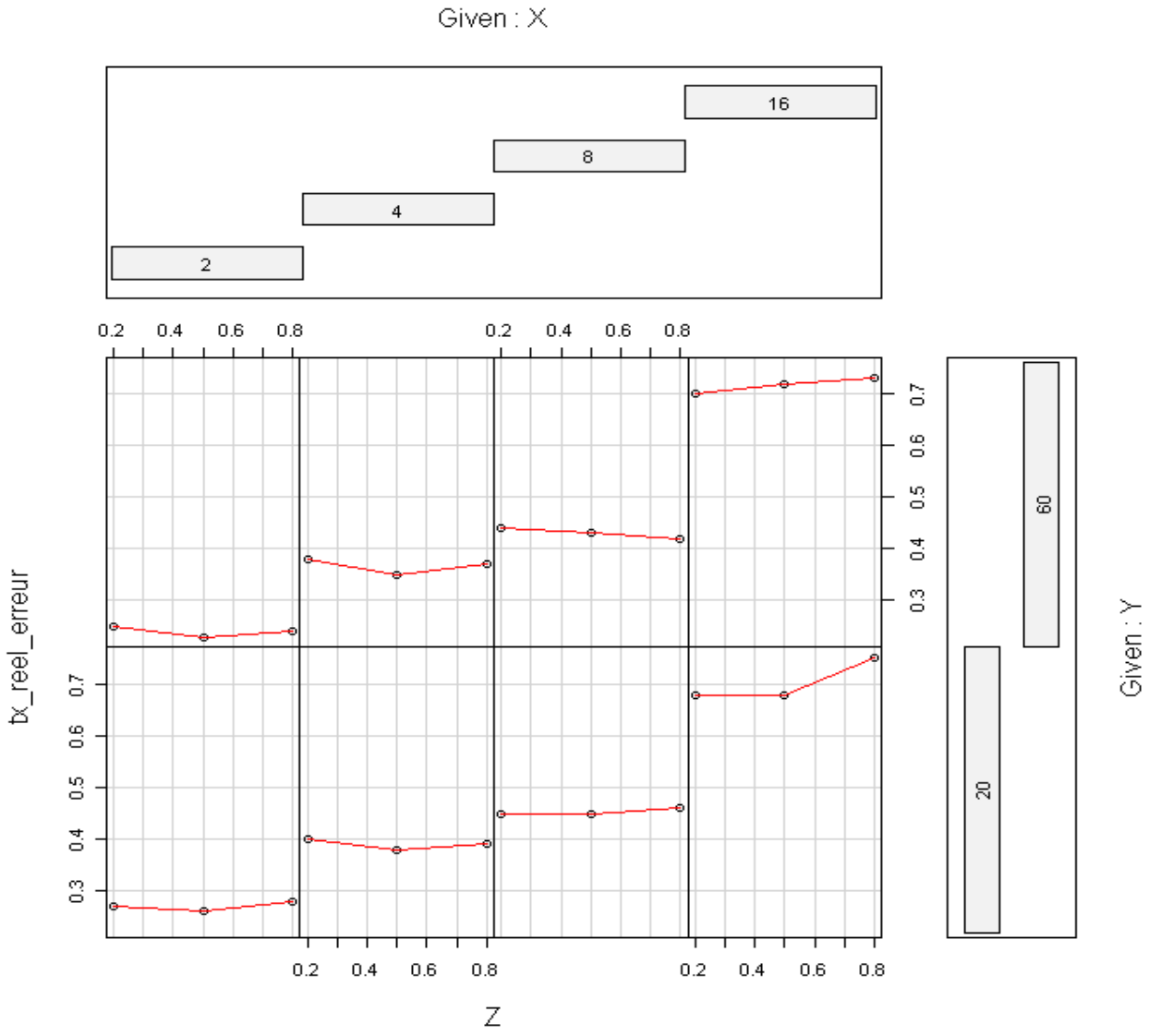


Figure 26 – Graphique « tx_reel_erreur » pour la première simulation.

Pour ce taux d'erreurs, on remarque encore une fois que le nombre de fondateurs et le taux d'échantillonnage ne semblent avoir aucunes incidences sur ce taux d'erreurs à l'inverse du nombre d'allèles qui semble encore avoir un rôle important.

En effet, pour un polymorphisme avec 2 allèles, ce taux d'erreurs est d'environ 0.25 alors qu'il est de 0.7 pour un polymorphisme avec 16 allèles. Ainsi, plus le nombre d'allèles est grand, et plus ce taux d'erreurs sera élevé.

Enfin, nous remarquons que, pour un polymorphisme avec16 allèles, le taux d'erreurs est assez élevé : de l'ordre de 70 % dans le cas de 60 fondateurs. En effet, plus il y aura un nombre important d'allèles et plus il y aura de combinaisons possibles de 2 allèles pour un locus, donc le risque de faire une erreur sera plus grand.

Dans la vie réelle, on est souvent confronté à des polymorphismes avec un faible nombre d'allèles. Ici, pour un polymorphisme avec 2 allèles, le taux d'erreurs est de 25% environ, et pour un polymorphisme avec 4 allèles, le taux d'erreurs est de 37% environ. Il serait donc intéressant de savoir si ce taux d'erreurs est cohérent avec les problèmes réels de génétique.

Dans un article intitulé : « *A Comparison of Bayesian Methods for Haplotype Reconstrution from Pomulation genotype Data* » par Matthew Stephens et Peter Donnelly, une étude a été faite sur ce type d'erreurs pour le logiciel PHASE. Ainsi, ils ont permuté aléatoirement un ensemble de 57 haplotypes avec aucunes données manquantes et cela 100 fois. Ils ont obtenu 100 jeux de données contenant chacun 28 individus possibles. Pour cette simulation, ils ont obtenu un taux d'erreurs de **36%**, ce qui est très similaire à notre étude. On peut donc en déduire que notre taux est cohérent.

- Etude de « tx_erreur_type1 »

Ce taux correspond aux erreurs probables annoncées par PHASE, mais il n'y en avait aucune en réalité. Dans R, on entre :

```
coplot(tx_erreur_type1~Z|X*Y,data=tableau,panel=panel.smooth)
```

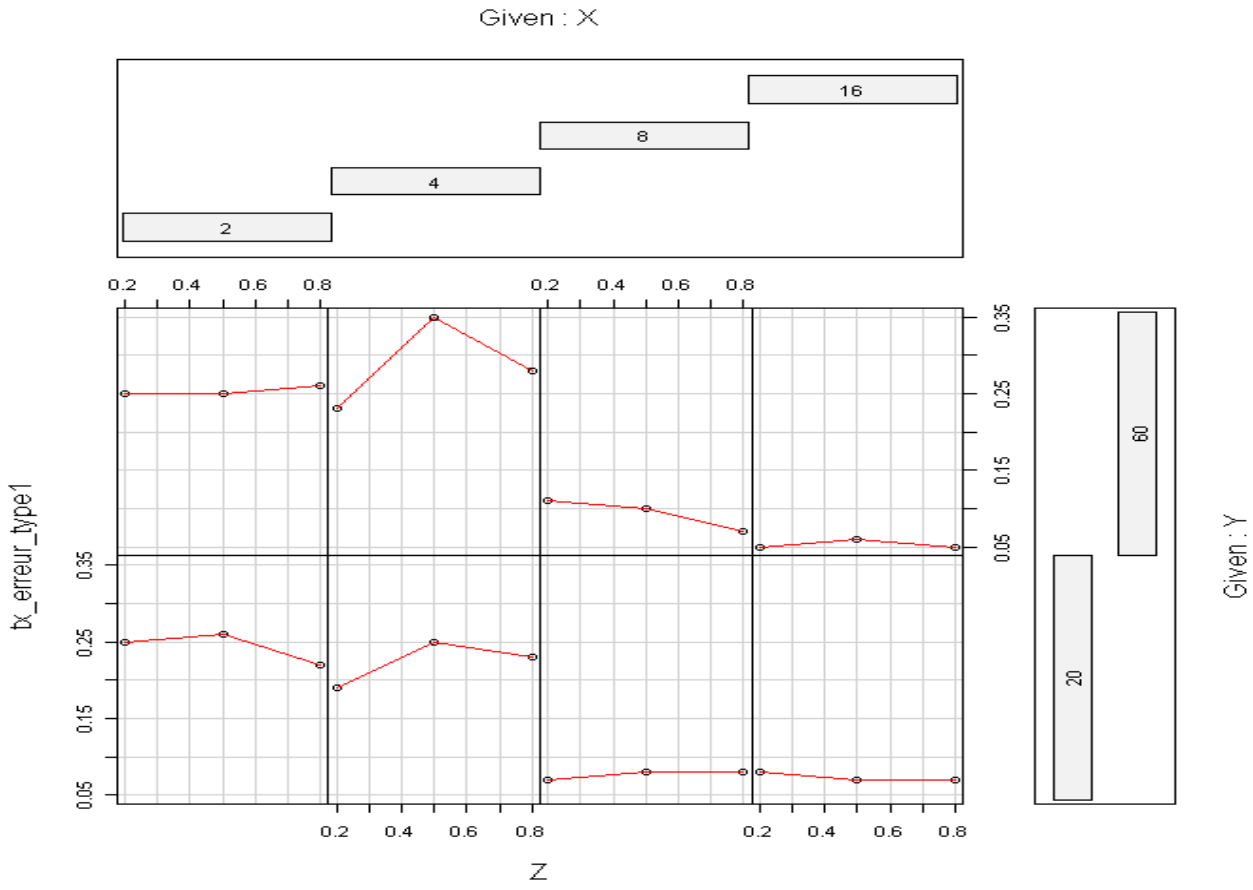


Figure 27 – Graphique « tx_erreur_type1 » pour la première simulation.

Pour ce taux d'erreurs, on peut faire les mêmes remarques que pour le « taux d'erreurs probables », c'est à dire que le nombre de fondateurs et le taux d'échantillonnage ne semblent avoir aucunes incidences sur ce taux d'erreurs à l'inverse du nombre d'allèles qui semble jouer un rôle majeur.

Pour un polymorphisme avec 2 ou 4 allèles, le taux d'erreurs moyen est de 0.25 alors que pour un polymorphisme de 8 ou 16 allèles, le taux d'erreurs moyen est de 0.07. Ainsi, plus le nombre d'allèle sera grand et plus ce taux d'erreurs sera faible.

Enfin, nous remarquons que, pour un polymorphisme avec 2 ou 4 allèles, le taux d'erreurs est assez élevé par rapport aux cas avec 8 et 16 allèles : de l'ordre de 25 % en moyenne et avec un pic à 35% dans le cas de 60 fondateurs, un taux d'échantillonnage de 0,5 et un polymorphisme avec 4 allèles, alors que ce taux est à 7% pour un polymorphisme avec 8 et 16 allèles. Cependant, dans la littérature, il n'y a pas eu d'étude sur ce type d'erreurs, il n'est donc pas possible de savoir si ce taux est cohérent.

- Etude de « tx_erreur_type2 »

Ce taux correspond aux erreurs que PHASE n'a pas détecté alors qu'il y en avait une en réalité. Dans R, on entre :

```
coplot(tx_erreur_type2~Z|X*Y,data=tableau,panel=panel.smooth)
```

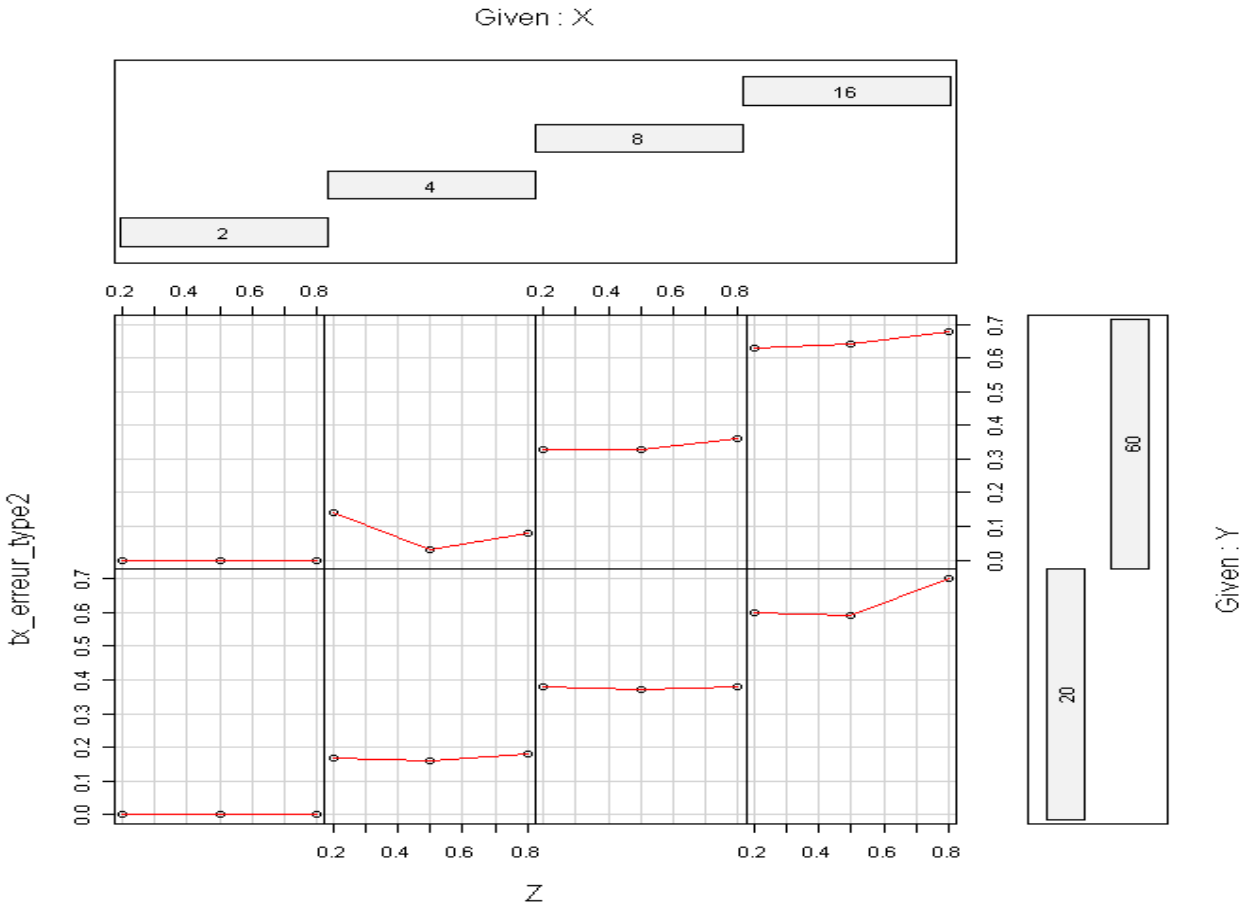


Figure 28 – Graphique « tx_erreur_type2 » pour la première simulation.

Pour ce taux d'erreurs, on peut faire les mêmes remarques que pour le « taux d'erreurs réelles », c'est à dire que le nombre de fondateurs et le taux d'échantillonnage ne semblent avoir aucunes incidences sur ce taux d'erreurs à l'inverse du nombre d'allèles qui semble jouer un rôle majeur.

Pour un polymorphisme avec 2 allèles, ce taux d'erreurs est très intéressant puisqu'il est proche de 0, alors qu'il est de 0.65 pour un polymorphisme avec 16 allèles. Ainsi, plus le nombre d'allèles est grand, et plus ce taux d'erreurs sera élevé.

Enfin, nous remarquons que, pour un polymorphisme avec16 allèles, le taux d'erreurs est assez élevé : de l'ordre de 65 % dans le cas de 60 fondateurs. En revanche, pour un polymorphisme avec 2 allèles, ce taux d'erreurs est très bas (très proche de 0). Ainsi, PHASE est très performant pour les cas avec un faible nombre d'allèles, il a très bien su repérer les difficultés (là où la phase était difficile à déduire). C'est rassurant puisque dans les problèmes réels de génétique, on est souvent confronté à des polymorphismes avec un faible nombre d'allèles. Cependant, dans la littérature, il n'y a pas eu d'étude sur ce type d'erreurs, il n'est donc pas possible de savoir si ce taux est cohérent.

Une seule simulation ne suffit pas pour faire une étude statistique détaillée. Nous allons donc utiliser nos 8 simulations pour faire une analyse plus détaillée des différents taux d'erreurs.

Grâce à la première simulation, nous avons pu constater que les différents taux d'erreurs semblaient dépendre du nombre d'allèles donc du polymorphisme. On peut affirmer ou non cette observation en regardant s'il existe des associations entre les différentes variables. On peut regarder pour cela la **matrice des corrélations** :

```
tab=data.frame(nb_allèle,fondateurs_mâle,taux_échantillonnage,tx_erreur_phase,tx_reel_erreur,
tx_erreur_type1,tx_erreur_type2)
cor(tab)
```

On obtient les valeurs suivantes :

	nb_allèle	fondateurs_mâle	taux_échantillonnage
nb_allèle	1.00000000	-0.02269636	0.07337185
fondateurs_mâle	-0.02269636	1.00000000	0.02260312
taux_échantillonnage	0.07337185	0.02260312	1.00000000
tx_erreur_phase	**-0.53653074**	-0.08476116	0.27626389
tx_reel_erreur	**0.91576707**	-0.08037828	0.09236565
tx_erreur_type1	**-0.47612509**	-0.06661444	0.28266106
tx_erreur_type2	**0.92628716**	-0.02127707	-0.01816201

On remarque que le nombre d'allèles est fortement corrélé aux taux d'erreurs du type « tx_reel_erreur » (corrélé à 92 %)) et « tx_erreur_type2 » (corrélé à 93 %). Le polymorphisme étant corrélé positivement à ces 2 types d'erreurs, plus le polymorphisme sera important et plus ces deux taux seront grands.

On remarque également que le nombre d'allèles est corrélé aux taux d'erreurs du type « tx_erreur_phase » (corrélé à 54 %) et « tx_erreur_type1 » (corrélé à 48 %). Le polymorphisme étant corrélé négativement à ces 2 types d'erreurs, plus le polymorphisme sera important et plus ces taux seront faibles.

Enfin, le nombre de fondateurs et le taux d'échantillonnage ne semblent pas jouer un rôle prépondérant.

Cette première analyse est en cohérence avec les observations faites pour les 4 types d'erreurs de la première simulation.

Puis, on peut représenter graphiquement ces corrélations grâce à une ***analyse en composantes principales***. Pour cela, on ne doit pas avoir de données manquantes.

Pour utiliser les fonctions dans R qui font des analyses en composantes principales, on importe la librairie **ade4**.

Pour effectuer une analyse en composantes principales, on utilise la fonction **dudi.pca.**

```
acp<-dudi.pca(tab)
```

Un histogramme s'affiche et on nous demande de choisir le nombre d'axes. Dans un premier temps, on sélectionne tous les axes, ici 7.

```
Select the number of axes: 7
```

Puis, on affiche les valeurs propres.

```
acp$eig
```

```
[1] 3.8364637840 1.4216904746 1.0148267371 0.6175540119 0.0907522872
[6] 0.0180390020 0.0006737032
```

Enfin, on affiche les pourcentages de variance dus aux axes.

```
pve<-100*acp$eig/sum(acp$eig)
pve
```

```
[1] 54.806625486 20.309863922 14.497524816  8.822200169  1.296461246
[6]  0.257700029  0.009624332
```

Les pourcentages de variance dus aux deux premiers axes sont de 54.81% pour l'axe 1 et de 20.31% pour l'axe 2.A eux seuls, ces 2 axes contribuent à 75.12% de la variance du nuage. On peut donc retenir deux axes.

```
acp<-dudi.pca(tab)
```

```
Select the number of axes: 2
```

Le cercle des corrélations sur les axes 1 et 2 correspond à une projection des variables initiales sur un plan à deux dimensions constitué par les deux premiers facteurs. Pour afficher le cercle des corrélations, on utilise la fonction **s.corcircle** .

```
s.corcircle(acp$co,xax=1,yax=2)
```

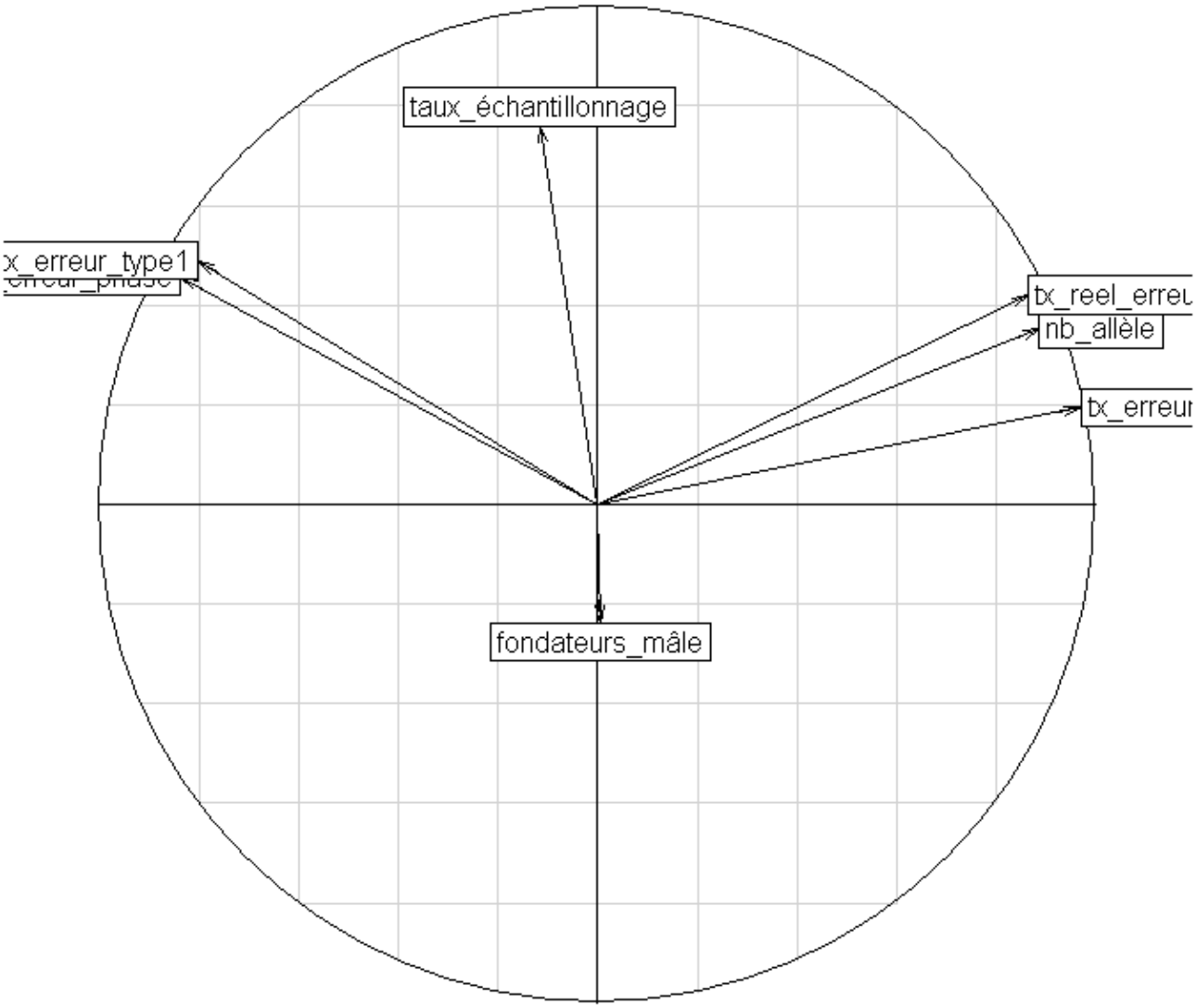


Figure 29 - Cercle des corrélations sur les axes 1 et 2

D'après le cercle des corrélations, l'axe 1 oppose les 4 types d'erreurs. Ainsi, si le taux d'erreurs probables de PHASE et le taux d'erreurs de type 1diminuent, à l'inverse, le taux d'erreurs réelles et le taux d'erreurs de type 2 augmentent

De plus, le nombre d'allèles est fortement corrélé avec le taux d'erreurs de type 2 et le taux d'erreurs réelles, et le nombre d'allèles est corrélé négativement aux deux autres types d'erreurs. Le polymorphisme joue donc un rôle important.

Le taux d'échantillonnage et le nombre de fondateurs, quant à eux, ne sont pas corrélés à un taux d'erreurs.

Le cercle des corrélations confirme donc que le polymorphisme joue un rôle important à l'inverse du nombre de fondateurs et du taux d'échantillonnage qui n'ont pas d'impact sur les différents types d'erreur.

Maintenant qu'on a une idée plus précise des facteurs qui jouent un rôle dans les différents taux d'erreurs, nous pouvons recommencer à tracer un graphique de chaque type d'erreurs en fonction du nombre de fondateurs, du nombre d'allèles, et du taux d'échantillonnage. Cette fois-ci, ce sera avec nos 8 simulations. On rappelle que :

X=as.factor(nb_allèle)
Y=as.factor(fondateurs_mâle)
Z=taux_échantillonnage

- Etude de « tx_erreur_phase » :

On rappelle que ce taux correspond aux erreurs potentielles, là où PHASE a eu du mal à faire une déduction.

coplot(tx_erreur_phase~Z|X*Y,data=tableau,panel=panel.smooth)

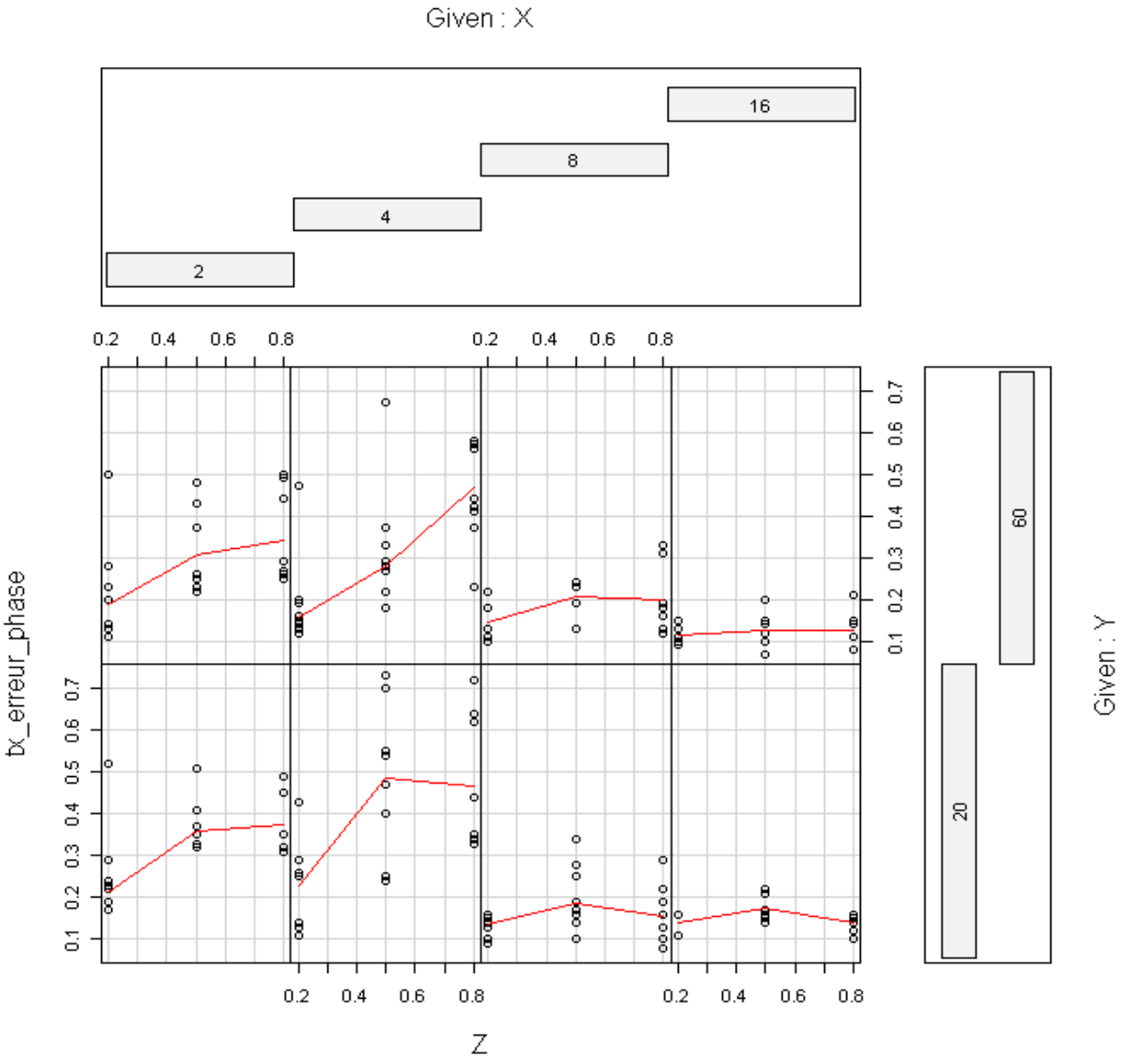


Figure 30 - Graphique « tx_erreur_phase » pour les 8 simulations.

On remarque que pour un polymorphisme avec 2 ou 4 allèles, les 8 simulations ont donné des taux d'erreurs très variés (par exemple le taux varie de 0,1 à 0,5 pour un polymorphisme avec 2 allèles, 60 fondateurs et un taux d'échantillonnage de 0.2). Cela peut être dû à une erreur de notre part. Nous avons peut-être fait une erreur en manipulant les fichiers d'entrées et de sorties de PHASE ou en entrant des mauvais paramètres, et ainsi les taux d'erreurs auraient pu être faussés.

En revanche, pour un polymorphisme avec 8 ou 16 allèles, les résultats sont plus cohérents car les 8 taux d'erreurs calculés sont groupés. Ainsi, ce taux est proche de 0.15 en moyenne.

Les principales observations sont les mêmes que pour la première simulation, c'est à dire que le nombre de fondateurs et le taux d'échantillonnage ne semblent avoir aucun impact sur ce taux d'erreurs à l'inverse du nombre d'allèles qui semble avoir un rôle important. Plus le nombre d'allèles est grand, et plus ce taux d'erreurs sera faible.

Il serait intéressant de tester si les différents facteurs sont significatifs. Pour cela, il faudrait faire une analyse de la variance (ANOVA). Cependant, pour les cas à 2 allèles et à 4 allèles, les résultats que nous avons obtenus sont trop variés, il ne serait donc pas judicieux de faire une ANOVA ici.

- Etude de « tx_reel_erreur »

On rappelle que ce taux correspond au nombre d'erreurs réelles, c'est à dire au nombre de changement d'allèle entre le fichier d'entrée et le fichier de sortie de PHASE.

```
coplot(tx_reel_erreur~Z|X*Y,data=tableau,panel=panel.smooth)
```

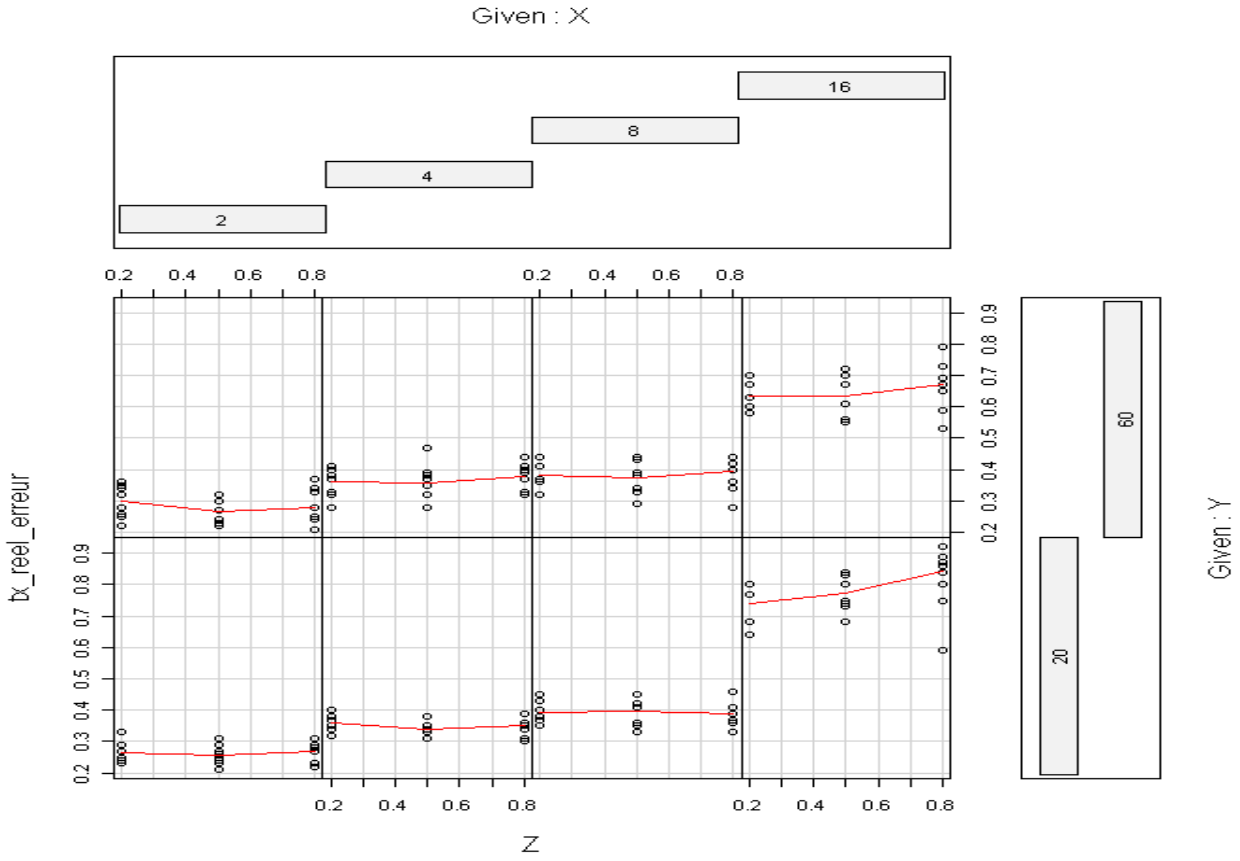


Figure 31 – Graphique « tx_erreur_phase » pour la première simulation.

Ici, les résultats sont cohérents car les 8 taux d'erreurs calculés sont groupés.

Les principales observations sont les mêmes que pour la première simulation. Pour ce taux d'erreurs, on remarque encore une fois que le nombre de fondateurs et le taux d'échantillonnage ne semblent avoir aucunes incidences sur ce taux d'erreurs à l'inverse du nombre d'allèles qui semble encore avoir un rôle important.

En effet, pour un polymorphisme avec 2 allèles, ce taux d'erreurs est d'environ de 28% en moyenne, ce qui est cohérent avec d'autres études de PHASE trouvées dans la littérature, alors qu'il est d'environ de 70% pour un polymorphisme avec 16 allèles. Comme je l'ai expliqué précédemment, plus il y aura un nombre important d'allèles et plus il y aura de combinaisons possibles de 2 allèles pour un locus, donc le risque de faire une erreur sera plus grand. Ainsi, plus le nombre d'allèles est grand, et plus ce taux d'erreurs sera élevé.

Les résultats étant cohérents, on peut ici faire une ANOVA pour tester si les facteurs sont significatifs. Cette analyse consiste à tester si les différences de variation dans chaque groupe défini par les modalités des variables explicatives s'écartent de manière significative de 0.

Nous pouvons faire une ANOVA sur chaque facteur. Commençons par le nombre d'allèles :

Soit H_0 l'hypothèse : Le nombre d'allèles n'a pas d'impact sur le taux d'erreurs réelles.

```
res1=aov(tx_reel_erreur~nb_allèle)
summary(res1)
```

```
             Df  Sum Sq  Mean Sq  F value  Pr(>F)
nb_allèle     1  4.5616  4.5616   925.05  < 2.2e-16 ***
Residuals     1  78 0.8778  0.0049
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

La probabilité de se tromper en rejetant H_0 est très faible ($<2.2*10^{-16}$), on rejette donc H_0. Ainsi, le nombre d'allèles a un effet sur le taux d'erreurs réelles, ce qui confirme nos observations précédentes.

Nous pouvons maintenant tester si le nombre de fondateurs a un impact sur ce taux d'erreurs réelles :

Soit H_0 l'hypothèse : Le nombre de fondateurs n'a pas d'impact sur le taux d'erreurs réelles.

```
res2=aov(tx_reel_erreur~fondateurs_mâle)
summary(res2)
```

```
                Df   Sum Sq   Mean Sq   F value  Pr(>F)
fondateurs_mâle  1   0.0351  0.035142   1.1575  0.2834
Residuals       178  5.4042  0.030361
```

La probabilité de se tromper en rejetant H_0 est grande (0.2834), on ne rejette pas H_0. Ainsi, le nombre de fondateurs n'influe pas sur le taux d'erreurs réelles, ce qui confirme encore une fois nos observations précédentes.

Enfin, nous pouvons tester si le taux d'échantillonnage a un impact sur le taux d'erreurs réelles :

Soit H_0 l'hypothèse : Le taux d'échantillonnage n'a pas d'impact sur le taux d'erreurs réelles.

```
res3=aov(tx_reel_erreur~taux_échantillonnage)
summary(res3)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
taux_échantillonnage	1	0.0464	0.046405	1.5317	0.2175
Residuals	178	5.3930	0.030298		

La probabilité de se tromper en rejetant H_0 est grande (0.2175), on ne rejette pas H_0. Ainsi, le taux d'échantillonnage n'a aucune incidence sur le taux d'erreurs réelles, ce qui confirme encore une fois nos observations précédentes.

Pour conclure, en ce qui concerne le taux d'erreurs réelles, seul le polymorphisme de l'individu joue un rôle important. Plus le polymorphisme sera important, et plus le taux d'erreurs sera grand.

- Etude de « tx_erreur_type1 »

On rappelle que ce taux correspond aux erreurs probables annoncées par PHASE, mais il n'y en avait aucune en réalité.

```
coplot(tx_erreur_type1~Z|X*Y,data=tableau,panel=panel.smooth)
```

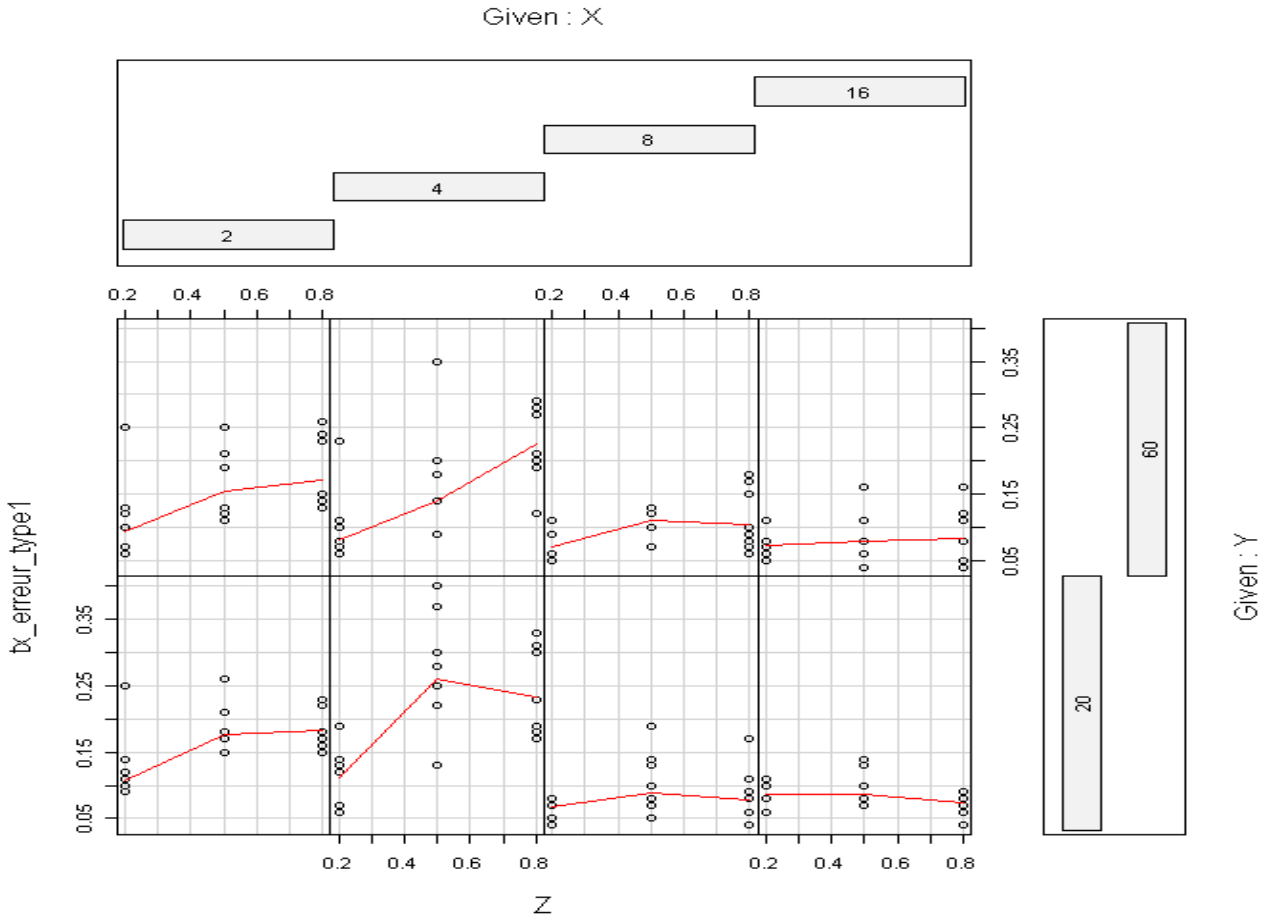


Figure 32 – Graphique « tx_erreur_type1 » pour les 8 simulations.

Nous pouvons d'abord faire la même observation que pour le taux d'erreurs de PHASE, c'est à dire que pour un polymorphisme avec 2 ou 4 allèles, les 8 simulations ont donné des taux d'erreurs très variés (par exemple le taux varie de 0,08 à 0,35 pour un polymorphisme avec 4 allèles, 60 fondateurs et un taux d'échantillonnage de 0.5). Encore une fois, cela peut être dû à une erreur de notre part : nous avons peut-être fait une erreur en manipulant les fichiers d'entrées et de sorties de PHASE ou en entrant des mauvais paramètres, et ainsi les taux d'erreurs auraient pu être faussés.

Cependant, pour un polymorphisme avec 8 ou 16 allèles, les résultats sont plus cohérents car les 8 taux d'erreurs calculés sont groupés. Par exemple, ce taux varie de 0.05 à 0.10 pour un polymorphisme avec 16 allèles, un taux d'échantillonnage de 0.8 et 20 fondateurs.

Enfin, les principales observations qui ressortent de ces graphiques sont les mêmes que pour le graphique représentant le taux d'erreurs PHASE, c'est à dire que le nombre de fondateurs et le taux d'échantillonnage ne semblent avoir aucun impact sur ce taux d'erreurs à l'inverse du nombre d'allèles qui semble avoir un rôle important. Plus le nombre d'allèles est grand, et plus ce taux d'erreurs sera faible.

Il serait également intéressant de faire une ANOVA mais, pour les cas à 2 allèles et à 4 allèles, les résultats que nous avons obtenus sont trop variés, il ne serait donc pas judicieux de faire une ANOVA ici.

- Etude de « tx_erreur_type2 »

On rappelle que ce taux correspond aux erreurs que PHASE n'a pas détecté alors qu'il y en avait une en réalité.

```
coplot(tx_erreur_type2~Z|X*Y,data=tableau,panel=panel.smooth)
```

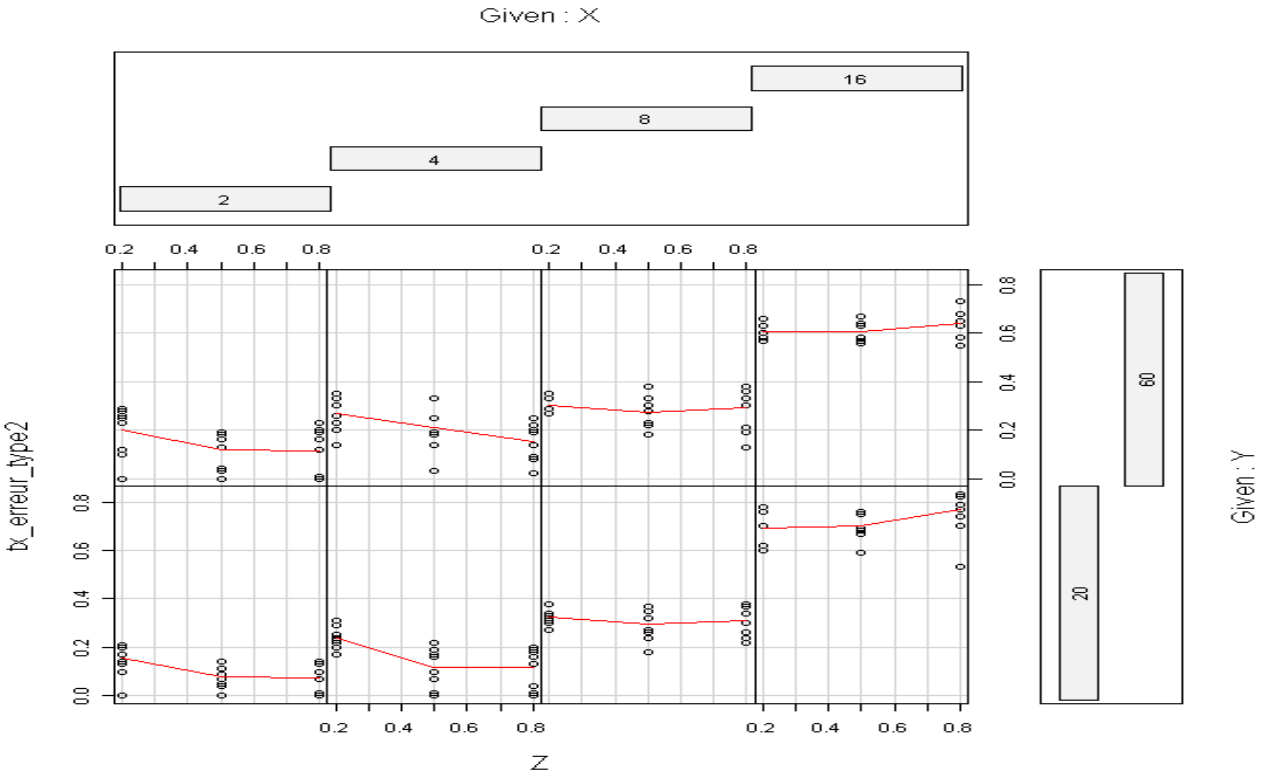


Figure 33 – Graphique « tx_erreur_type2 » pour les 8 simulations.

Ici, les résultats sont cohérents car les 8 taux d'erreurs calculés sont groupés.

Les principales observations sont les mêmes que pour le taux d'erreurs réelles. Pour ce taux d'erreurs, on remarque encore une fois que le nombre de fondateurs et le taux d'échantillonnage ne semblent avoir aucunes incidences sur ce taux d'erreurs à l'inverse du nombre d'allèles qui semble encore avoir un rôle important.

En effet, pour un polymorphisme avec 2 allèles, ce taux d'erreurs est d'environ de 18% en moyenne quelque soit le nombre de fondateurs, alors qu'il est d'environ de 70% en moyenne pour un polymorphisme avec 16 allèles. Plus il y aura un nombre important d'allèles et plus il y aura de combinaisons possibles de 2 allèles pour un locus, donc on augmente le risque de faire une erreur de type 2. Ainsi, plus le nombre d'allèles est grand, et plus ce taux d'erreurs sera élevé.

Les résultats étant cohérents, on peut ici aussi faire une ANOVA sur chaque facteur pour tester si ces facteurs sont significatifs. Commençons par le nombre d'allèles :

Soit H_0 l'hypothèse : Le nombre d'allèles n'a pas d'impact sur le taux d'erreurs de type 2.

```
res1=aov(tx_erreur_type2~nb_allèle)
summary(res1)
```

```
             Df    Sum Sq   Mean Sq  F value     Pr(>F)
nb_allèle     1    7.6637    7.6637   1075.6  < 2.2e-16 ***
Residuals   178    1.2683    0.0071
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

La probabilité de se tromper en rejetant H_0 est très faible ($<2.2*10^{-16}$), on rejette donc H_0. Ainsi, le nombre d'allèles a un effet sur le taux d'erreurs de type 2, ce qui confirme nos observations précédentes.

Nous pouvons maintenant tester si le nombre de fondateurs a un impact sur ce taux d'erreurs de type 2:

Soit H_0 l'hypothèse : Le nombre de fondateurs n'a pas d'impact sur ce taux d'erreurs .

```
res2=aov(tx_erreur_type2~fondateurs_mâle)
summary(res2)
```

```
                Df   Sum Sq   Mean Sq  F value  Pr(>F)
fondateurs_mâle  1   0.0040  0.004044   0.0806  0.7768
Residuals      178   8.9279  0.050157
```

La probabilité de se tromper en rejetant H_0 est grande (0.7768), on ne rejette pas H_0. Ainsi, le nombre de fondateurs n'influe pas sur le taux d'erreurs de type 2, ce qui confirme encore une fois nos observations précédentes.

Enfin, nous pouvons tester si le taux d'échantillonnage a un impact sur le taux d'erreurs de type 2 :

Soit H_0 l'hypothèse : Le taux d'échantillonnage n'a pas d'impact sur ce taux d'erreurs.

```
res3=aov(tx_erreur_type2~taux_échantillonnage)
summary(res3)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
taux_échantillonnage	1	0.0029	0.002946	0.0587	0.8088
Residuals	178	8.9290	0.050163		

La probabilité de se tromper en rejetant H_0 est grande (0.8088), on ne rejette pas H_0. Ainsi, le taux d'échantillonnage n'a aucune incidence sur le taux d'erreurs de type 2, ce qui confirme encore une fois nos observations précédentes.

Pour conclure, en ce qui concerne le taux d'erreurs de type 2, seul le polymorphisme de l'individu joue un rôle important. Plus le polymorphisme sera important, et plus le taux d'erreurs sera grand.

Pour résumé les études qui ont été faites sur les 4 types d'erreurs, nous pouvons dire que nous avons observé deux comportements : tandis que le taux d'erreurs réelles et le taux d'erreurs de types 2 tendent à croître suivant l'importance croissante du polymorphisme, le taux d'erreurs PHASE et le taux d'erreurs de type 1 ont une tendance inverse.

De plus, nous avons pu remarquer et démontrer que le taux d'échantillonnage et le nombre de fondateurs n'ont pas d'impact sur les différents types d'erreurs, à l'inverse du nombre d'allèles donc du polymorphisme de l'individu qui joue un rôle majeur dans l'étude des taux d'erreurs.

Enfin, nous avons également pu constater que les différents taux d'erreurs pouvaient paraître assez élevés, cependant, pour le taux d'erreurs réelles, nous avons pu voir dans la littérature que des études ont montrés que ce taux était du même ordre que le taux d'erreurs réelles dans notre étude. La valeur de ce taux d'erreurs dépend bien sûr de la simulation, des exemples sont donnés dans plusieurs documents :

- *« A Comparison of Bayesian Methods for Haplotype Reconstruction from Population Genotype Data »* par Matthew Stephens et Peter Donnelly ;
- *« Accounting for Decay of Linkage Disequilibrium in Haplotype Inference and Missing-Data Imputation »* par Matthew Stephens et Paul Scheet ;
- « *A Fast and Flexible Statistical Model for Large-Scale Population Genotype Data : Applications to Inferring Missing Genotypes and Haplotypic Phase* » par Paul Scheet et Matthew Stephens.

3.2. Autres logiciels :

Dans la littérature, il existe beaucoup de logiciels permettant la reconstruction d' haplotypes. Je ne parlerais ici que des logiciels ou packages R les plus cités dans les différentes publications.

3.2.1. Packages R

3.2.1.1. Package « haplo.stats »

« Haplo.stats » fait une reconstruction d' haplotypes en estimant de façon itérative les fréquences haplotypiques sachant les données observées et l'estimation actuelle des paramètres de régression.

L'estimation du maximum de vraisemblance des paramètres de régression est calculée avec l'**algorithme EM**.

On note $z = (x_e, x_f, x_g)$ le vecteur des covariances, où x_e représente les covariances non génétiques, x_f représente les covariances génétiques et x_g représente les covariances d'interaction en terme de covariance génétique et non génétique. On note ainsi $\beta = (\beta_e, \beta_f, \beta_g)$ le vecteur des coefficients de régression associés aux covariances.

La vraisemblance pour les données génétiques est une fonction de fréquences haplotypiques $\theta = (\theta_1, \ldots, \theta_J)$ où θ_J est la fréquence du $j^{ème}$ haplotype avec $j = \{1, \ldots, J\}$

Soit h un vecteur des fréquences haplotypiques pour un individu, avec la $j^{ème}$ composante étant égale au nombre d' haplotypes h_j que l'individu possède.

En supposant l'équilibre de Hardy-Weinberg, la probabilité du génotype g (ou h) est :

$$Pr_{\varphi}(h) = \binom{2}{h_J} \prod_{j=1}^{J-1} \binom{2}{h_j} \varphi_J^{h_j} \left(1 + \sum_{j=1}^{j-1} \varphi_j\right)^{-2}$$

où : $$\varphi_j = \frac{\theta_j}{1 - \sum_{j=1}^{J-1} \theta_j}$$

L'**étape E** de l'algorithme EM consiste à prendre l'estimation conditionnelle sur les données complètes de la log-vraisemblance compte tenu des données observées. C'est une fonction de probabilité conditionnelle sur le comptage d' haplotypes. Sa forme générale sous l'indépendance de x_g et de x_e est :

$$Pr(h_{ij}, h_{ij'} | d_i^{(obs)}) = \frac{f_\beta(y_i|z_i) Pr_\varphi(g_i)}{\sum_{g \epsilon G} f_\beta(y_i|z_i) Pr_\varphi(g)}$$

où :

- i = 1, …, N
- j,j' = 1, …, J
- $d_i^{(obs)}$ représente les données observées
- G est l'ensemble des paires d' haplotypes consistantes avec le génotype observé k
- $f_\beta(y_i \mid z_i)$ est la densité de y_i

L'**étape M** de l'algorithme EM consiste en la maximisation de l'estimation conditionnelle sur les données complètes de la log-vraisemblance. Les paramètres de régression peuvent être facilement estimés avec une régression pondérée où les poids sont les probabilités conditionnelles :

$$w_{gi} = Pr(h_{ij}, h_{ij'} | d_i^{(obs)})$$

Les fonctions les plus intéressantes contenues dans ce package et qui ont un rapport direct avec notre contexte sont : ***haplo.em*, *haplo.em.fitter*, *haplo.group*.**

3.2.1.2. Package « Hapassoc »

« Hapassoc » utilise un **algorithme EM** par la méthode des poids pour estimer les paramètres de régression.

L'étape d'estimation consiste à calculer la log-vraisemblance conditionnelle estimée des données complètes (x, y) compte tenu des données observées (x_{obs}, y) et des estimateurs du paramètre actuel. L'étape de maximisation maximise la fonction résultante.

« Hapassoc » commence par générer des « pseudo-individus » pour tous les individus qui ont des génotypes qui entraînent plusieurs haplotypes possibles. Ces « pseudo-individus » représentent toutes les configurations d' haplotypes possibles pour le génotype ambigu et ont un poids associé qui peut être calculé en utilisant les lois de Bayes.

La log-vraisemblance conditionnelle estimée est une fonction de comptage d' haplotypes :

$$\begin{aligned} Q(\theta|\theta^{(t)}) &= \sum_{i=1}^{n} E[l_{y|x}(\theta|x_i, y_i)|x_{obs,i}, y_i, \theta^{(t)}] \\ &= \sum_{i=1}^{n}\sum_{j=1}^{|S|} w_{ij}(\theta^{(t)}) l_{y|x}(\theta|x^{(j)}, y_i) + \sum_{i=1}^{n}\sum_{j=1}^{|S|} w_{ij}(\theta^{(t)}) l_x(\theta, x^{(j)}) \end{aligned}$$

où :

- $L_{y|x}$ est la log-vraisemblance pour le modèle de régression ;
- l_x est la log-vraisemblance pour les paramètres du modèle de covariance ;
- $\theta^{(t)} = (\beta^{(t)}, \gamma^{(t)})$ correspond aux estimateurs du paramètre actuel, β est le paramètre de régression et γ est le paramètre du modèle de covariance ;
- Les poids w_{ij} pour les pseudo-individus, avec $x^{(j)}$ étant le $j^{ème}$ vecteur de covariance, sont calculés en utilisant les lois de Bayes :

$$\begin{aligned} w_{ij}^{(t)} &= Pr\{x_i = x^{(j)} | x_{obs,i}, y_i, \theta^{(t)}\} \\ &= \begin{cases} 0, & \text{si x(j) n'est pas compatible avec xobs,i} \\ \frac{Pr(y_i|x^{(j)},\theta^{(t)}) Pr(x^{(j)}|\theta^{(t)})}{\sum Pr(y_i|x^{(k)},\theta^{(t)}) Pr(x^{(k)}|\theta^{(t)})}, & \text{si x(j) est compatible avec xobs,i} \end{cases} \\ &= \begin{cases} 0, & \text{si x(j) n'est pas compatible avec xobs,i} \\ \frac{Pr(y_i|x^{(j)},\theta^{(t)}) Pr(x_g^{(j)}|\theta^{(t)}) Pr(x_c^{(j)}|\theta^{(t)})}{\sum Pr(y_i|x^{(k)},\theta^{(t)}) Pr(x_g^{(k)}|\theta^{(t)}) Pr(x_c^{(k)}|\theta^{(t)})}, & \text{si x(j) est compatible avec xobs,i} \end{cases} \end{aligned}$$

3.2.1.3. Autres packages :

Il existe beaucoup de packages R dans la littérature permettant la reconstruction d' haplotypes. Les packages qui ont un rapport direct avec notre contexte sont répertoriés dans la figure 34. Ce tableau indique le package avec ses fonctions les plus intéressantes ainsi que les méthodes mathématiques qui sont utilisées.

Package	Méthode(s) utilisée(s)	Fonction(s) intéressante(s)
MasterBayes	Maximum de vraisemblance, méthode MCMC	MasterBayes, MCMCped, MLE.ped
SHARE	Algorithme EM	haplo
hapsim	aucune méthode particulière	haplosim
HaploSim	aucune méthode particulière	SampleHaplotype, SamplePedigree
pegas	Monte-Carlo	hw.test
haplo.ccs	Algorithme EM	haplo.hpp

Figure 34 – Tableau des packages R les plus intéressants.

Il peut être intéressant de comparer ces packages entre eux, voir leur efficacité suivant la méthode employée.

3.2.2. PL-EM

Ce logiciel utilise une stratégie PL (Partition-Ligation) ainsi que l'algorithme EM pour reconstruire les haplotypes. Il produit également des estimateurs sur toutes les fréquences haplotypiques. Nous allons étudier la sortie que donne PL-EM à travers un exemple.

Le fichier d'entrée est assez simple puisqu'il ne contient que les génotypes, prenons par exemple :

45001
15221
11111

avec 0 lorsqu'on a 'Aa' (homozygotie), 1 lorsqu'on a 'AA' (hétérozygotie), 2 lorsqu'on a 'aa', 3 lorsqu'on a '??', 4 lorsqu'on a 'A?' et 5 lorsqu'on a 'a?'.

Dans notre exemple, on a 3 individus. Pour pouvoir exécuter PL-EM sous Windows, on ouvre une invite de commande et on doit entrer une commande de la forme :

plem genotype.int genotype.out 0 2 a b

où :

- genotype.int est le fichier d'entré ci-dessus :
- genotype.out sera le fichier de sortie contenant toutes les informations ;
- a est le nombre d'individus (a vaut donc 3 pour notre exemple) ;
- b est le nombre d'itérations (On recommande 20 itérations) ;
- 0 et 2 sont là à titre indicatif.

On obtient une sortie qui se compose de deux principales parties. La première partie est la suivante :

```
Individual #1: top=1
*1  1, 2    0.99993
00000
01110

Individual #2: top=2
*1  2, 2    0.66665
01110
01110
*2  3, 2    0.33335
00110
01110

Individual #3: top=1
*1  1, 1    1.00000
00000
00000
```

Dans cette partie, pour chaque individu, les paires d' haplotypes prédites par le logiciel sont listées et sont accompagnées par une probabilité. Chaque individu a également deux identifiants correspondant aux deux haplotypes contenus dans la paire prédite. Ainsi, l'individu 1 a les haplotypes 1 et 2, l'haplotype 1 étant '00000' et l'haplotype 2 étant '01110'.

La deuxième partie est la suivante :

ID	theta	std error	haplotype
1	0.49998	0.20413	00000 (0)
2	0.33332	0.19245	01110 (14)
3	0.16667	0.15215	00110 (6)

Tous les haplotypes apparaissant dans chaque individu sont listés. Cette sortie donne un résumé de toutes les fréquences haplotypiques ainsi que l'erreur standard de déviation.

Il serait intéressant de comparer ce logiciel avec le logiciel PHASE car PHASE utilise une approche Bayésienne alors que PL-EM utilise l'algorithme EM. Une comparaison permettrait de comparer l'efficacité des 2 méthodes statistiques employées.

3.2.3. Logiciels supplémentaires

Dans la littérature, il existe beaucoup de logiciels permettant la reconstruction d'haplotypes. Ces logiciels peuvent se répertoriés en deux catégories : ceux qui utilisent une approche Bayésienne et ceux qui utilisent l'algorithme EM. Ces logiciels sont classés dans le tableau ci-dessous :

Approche Bayésienne	Algorithme EM
PHASE	**PL-EM**
HAPLOTYPER	HAPLO
HAPLOREC	HAPLORE
HAP	TripleM
HAP2	
Arlequin	

Figure 35 – Logiciels trouvés dans la littérature.

3.3. Comparaison :

Par un soucis de temps, je n'ai pu utiliser qu'un logiciel pour le moment. Ainsi, je n'ai pas encore pu faire de comparaison avec mes pedigrees simulés. Dans cette partie, je ferais donc une courte comparaison suivant tout ce que j'ai pu lire dans la littérature.

Dans beaucoup de publications, j'ai pu lire que PHASE était considéré comme le logiciel le plus rapide et le plus précis. Pour vérifier cette affirmation, je comparerais donc PHASE avec d'autres logiciels suivant 2 critères : le temps d'exécution et le taux d'erreur.

L'article « *A Comparison of Phasing Algorithms for Trios and Unrelated Individuals* », écrit entre autres par Matthew Stephens, fait une bonne comparaison entre plusieurs logiciels énoncés dans la figure 35.

3.3.1. Temps d'exécution

Ils ont fait 2 simulations :

- ST4 : 100 jeux de données de 100 trios simulés avec un taux de recombinaison constant, la taille de la population constante et les accouplements sont générés aléatoirement. On a également 2% de valeurs manquantes (ces valeurs manquantes sont aléatoires) ;
- SU4 : 100 jeux de données de 90 individus indépendants simulés avec un taux de recombinaison constant, la taille de la population constante et les accouplements sont générés aléatoirement.

On fait ces 2 simulations avec les logiciels PHASE, HAP, HAP2, tripleM et PL-EM et on calcule le temps d'exécution. On obtient le tableau suivant :

Temps d'exécution		
Algorithme	**ST4**	**SU4**
PHASE	3 h 32 min	10 h 52 min
HAP	22.3 s	35.1 s
HAP2	18.4 s	2 min 6 s
tripleM / PL-EM	1.5 s	4 min 22 s

Figure 36 – Comparaison des temps d'exécution.

Les temps d'exécution les plus courts étant en rouge, on remarque que, pour ces 2 simulations, PHASE n'est pas du tout le plus rapide, bien au contraire. Pour la simulation ST4, il met 3 h 32 min là où PL-EM met seulement 1.5 s, l'écart est donc énorme.

Cela peut s'expliquer par le fait que PHASE calcule plus de chose que PL-EM. Le logiciel PL-EM ne calcule que la meilleure estimation alors que PHASE calcule la meilleure estimation, le meilleur échantillon et les meilleurs estimateurs d'incertitude.

Cependant, HAP2 calcule la même chose que PHASE mais HAP2 ne met que 18.4 s pour la simulation ST4, ce qui est très rapide comparé à PHASE. Cela peut quand même s'expliquer par le fait que PHASE est plus complexe que HAP2, PHASE prend plus de choses en compte pour faire les calculs que HAP2 ne fait pas.

3.3.2. Taux d'erreur

On fait 6 autres simulations en plus de ST4 et SU4 :

- ST1 : 100 jeux de données de trios simulés avec un taux de recombinaison constant, la taille de la population constante et les accouplements sont générés aléatoirement ;
- ST2 : même chose que ST1 mais avec l'ajout d'un taux de combinaison variable ;
- ST3 : même chose que ST2 excepté qu'un modèle de démographie avec des américains blancs a été utilisé ;
- SU1 : 100 jeux de données de 90 individus indépendants simulés avec un taux de recombinaison constant, la taille de la population constante et les accouplements générés aléatoirement ;
- SU2 : même chose que SU1 mais avec l'ajout d'un taux de combinaison variable ;
- SU3 : même chose que SU2 excepté qu'un modèle de démographie avec des américains blancs a été utilisé.

On nomme « switch error » le pourcentage d'échanges possibles dans la reconstruction d'haplotypes. La figure 37 compare l'efficacité des logiciels grâce à ce taux d'erreur :

Taux « Switch error » (en %)				
Simulation	PHASE	HAP	HAP2	tripleM
ST1	0,74	2,14	2,58	3,02
ST2	0,22	1,51	5,97	2,87
ST3	1,36	2,4	2,95	3,81
ST4	1,48	2,62	3,17	4,12
SU1	2,4	6,5	6,9	9
SU2	2,2	9,8	15,1	13,1
SU3	4,8	7,1	8,2	11
SU4	5,3	7,8	9,2	11,4

Figure 37 – Comparaison du taux d'erreur du type « switch error ».

Le taux d'erreur le plus faible étant en rouge, on remarque que, pour ces 8 simulations, PHASE est celui où le taux est le plus faible (variant de 0,22 % à 5,3 %). Pour la simulation SU2, PHASE est nettement plus précis que les autres logiciels : 2,2 % pour PHASE alors que le deuxième taux le plus faible est de 9,8 % pour HAP.

PHASE est donc nettement plus précis que HAP, HAP2 ou encore tripleM. Pour appuyer cette affirmation, on étudie un autre type d'erreur. On nomme « IGP » le pourcentage de génotypes qui ont leur phase déduite incorrectement. Les résultats sont dans le tableau ci-dessous :

Taux « IGP » (en %)				
Simulation	PHASE	HAP	HAP2	tripleM
ST1	0,05	0,17	0,23	0,24
ST2	0,02	0,11	0,43	0,2
ST3	0,12	0,21	0,27	0,33
ST4	0,12	0,2	0,29	0,34
SU1	2,5	7,9	7,1	5,8
SU2	2,4	9,5	11	8
SU3	5,1	8,5	8,6	8,2
SU4	5,2	8,4	8,7	8

Figure 38 – Comparaison du taux d'erreur du type « IGP ».

Le taux d'erreur le plus faible étant en rouge, on peut confirmer ce qui a été dit auparavant. PHASE est donc plus précis que HAP, HAP2 ou encore tripleM.

En résumé, si on se concentre seulement sur la simulation ST4, on obtient les informations suivantes :

Simulation ST4			
Logiciel	temps d'exécution	taux erreur du type «switch error»	taux erreur du type «IGP»
PHASE	3 h 32 min	1,48%	0,12%
HAP	22,3 s	2,62%	0,20%
HAP2	18,4 s	3,17%	0,29%
tripleM	1,5 s	4,12%	0,34%

Figure 39 – Résumé pour la simulation ST4.

En résumé, pour la simulation ST4, PHASE est le plus lent mais le plus précis. A l'inverse, tripleM est le plus rapide mais le moins précis. Ces observations sont les mêmes pour les autres simulations.

PHASE est bien le logiciel le plus précis mais pas le plus rapide même si on peut considérer qu'il est rapide par rapport à tous les calculs qu'il fait.

Problèmes rencontrés

Conclusion

Références bibliographiques

Annexes

Annexe 1 : *Préparation du fichier d'entrée pour PHASE*

```
!=====================================
        SUBROUTINE write_2                                      !
====================AJOUTE
!=====================================
 IMPLICIT NONE

 INTEGER :: i,k,j

 DO i = 1,num_tot_ind

!==============================
! Pour PHASE
!==============================

            IF (indv(i)%sex=='m' .AND. indv(i)%surv==1 .AND. indv(i)%gener==time) THEN

                  IF(i<10) THEN
                        WRITE(21,'(A1,I1)') '#',i
                  ELSE IF(i<100) THEN
                        WRITE(21,'(A1,I2)') '#',i
                  ELSE IF(i<1000) THEN
                        WRITE(21,'(A1,I3)') '#',i
                  ELSE IF(i<10000) THEN
                        WRITE(21,'(A1,I4)') '#',i
                  END IF

                        DO k=1,2
                              WRITE(21,'(100I3)') (mal_off_gnome(k,j,indv(i)%rank_off),
j=1,num_neu_loci)
                        END DO

            ELSE IF (indv(i)%sex=='f' .AND. indv(i)%surv==1 .AND. indv(i)%gener==time)
THEN

                  IF(i<10) THEN
                        WRITE(21,'(A1,I1)') '#',i
                  ELSE IF(i<100) THEN
                        WRITE(21,'(A1,I2)') '#',i
                  ELSE IF(i<1000) THEN
                        WRITE(21,'(A1,I3)') '#',i
                  ELSE IF(i<10000) THEN
                        WRITE(21,'(A1,I4)') '#',i
                  END IF

                        DO k=1,2
```

```
                                        WRITE(21,'(100I3)') (fem_off_gnome(k,j,indv(i)%rank_off),
j=1,num_neu_loci)
                                END DO
                        END IF

!==============================



  END DO
!==================================
        END SUBROUTINE write_2
!==================================


!==================================
        SUBROUTINE write_3                                      !
=====================AJOUTE
!==================================
  IMPLICIT NONE

  INTEGER :: i,j,k

  DO i = 1,num_tot_ind

!==============================
! Pour PHASE
!==============================

        IF (indv(i)%gener==0) THEN
                IF (indv(i)%sex=='m') THEN

                        IF(i<10) THEN
                                WRITE(21,'(A1,I1)') '#',i
                        ELSE IF(i<100) THEN
                                WRITE(21,'(A1,I2)') '#',i
                        ELSE IF(i<1000) THEN
                                WRITE(21,'(A1,I3)') '#',i
                        ELSE IF(i<10000) THEN
                                WRITE(21,'(A1,I4)') '#',i
                        END IF

                                DO k=1,2
                                        WRITE(21,'(100I3)') (sir_gnome(k,j,indv(i)%rank_par),
j=1,num_neu_loci)
                                END DO
                ELSE IF (indv(i)%sex=='f') THEN

                        IF(i<10) THEN
                                WRITE(21,'(A1,I1)') '#',i
```

```
                        ELSE IF(i<100) THEN
                                WRITE(21,'(A1,I2)') '#',i
                        ELSE IF(i<1000) THEN
                                WRITE(21,'(A1,I3)') '#',i
                        ELSE IF(i<10000) THEN
                                WRITE(21,'(A1,I4)') '#',i
                        END IF

                                DO k=1,2
                                        WRITE(21,'(100I3)') (dam_gnome(k,j,indv(i)%rank_par),
j=1,num_neu_loci)
                                END DO
                END IF
        END IF

!==============================


!       IF (indv(i)%gener==0 .AND. indv(i)%sex=='m') THEN
!               WRITE(21,'(3I4,100I4)') indv(i)%self, indv(i)%dad, indv(i)%mum,
(sir_gnome(1,j,i),j=1,num_neu_loci), (sir_gnome(2,j,i),j=1,num_neu_loci)
!       END IF



  END DO

!==================================
        END SUBROUTINE write_3
!==================================
```

Annexe 2 : *Script « comparaison» (programme fortran)*

Ce programme permet de calculer 4 types d'erreur :
- Les erreurs probables (PHASE n'est pas sûr d'avoir fait la bonne estimation) ;
- Les erreurs réelles ;
- erreur de type 1 : erreur probable annoncée par PHASE mais il n' y a aucune erreur ;
- erreur de type 2 : PHASE n'a pas détecté d'erreur alors qu'il y en a une en réalité.

```
!======================
PROGRAM comparaison
!======================
IMPLICIT NONE
```

```
CHARACTER(len=50) :: heading
CHARACTER(len=70):: ligne1,ligne2
CHARACTER (len=30) :: var1,var2,var3

INTEGER, PARAMETER :: dp=SELECTED_REAL_KIND(14)    ! precision up to 14 decimal digits
INTEGER i, j, j2, k, pos1, pos2

REAL(KIND=dp) :: taux_erreur1, taux_erreur2, taux_erreur3, taux_erreur4, compteur, compteur2
REAL(KIND=dp) :: compteur3, compteur4, num_loci, num_ind

compteur=0
compteur2=0
compteur3=0
compteur4=0
pos1=0
pos2=0

!================================================================================
                    On regarde les erreurs probables et les erreurs réelles :
!================================================================================

OPEN(UNIT=1, FILE='gen3-phase.txt',STATUS='old')

READ(1,*) num_loci
READ(1,*) num_ind

DO i=1,num_ind
        READ(1,'(A)') heading
        DO k=1,2
                READ(1,'(A)') ligne1
                DO j=1,len(ligne1)-2
                        var1=ligne1(j:j)
                        var2=ligne1(j+1:j+1)
                        IF(var1=="(") THEN
                                compteur=compteur+1
                                ligne1(j:j)=var1
                                ligne1(j+1:j+1)=var2
                        END IF
                END DO
        END DO
END DO

taux_erreur1=(compteur/2)/(num_loci*num_ind)


write(*,*) "nombre probable d'erreur :", compteur/2
write(*,*) "tx probable d'erreur :",taux_erreur1

CLOSE(1)

OPEN(UNIT=1, FILE='gen3-phase.txt',STATUS='old')
OPEN(UNIT=2, FILE='gen3.inp',STATUS='old')

READ(2,'(A)') heading
READ(2,'(A)') heading
READ(2,'(A)') heading
READ(2,'(A)') heading
READ(1,'(A)') heading
READ(1,'(A)') heading
```

```
DO i=1,num_ind
        READ(1,'(A)') heading
        READ(2,'(A)') heading
        DO k=1,2
                READ(2,'(A)') ligne2
                READ(1,'(A)') ligne1
                DO j2=1,len(ligne2)
                        var2=ligne2(j2:j2)
                        IF (var2 /=" ") THEN
                                pos2= pos2 + 1
                                DO j=1,len(ligne1)
                                var1=ligne1(j:j)
                                        IF (var1 /= " " .AND. var1 /= "(" .AND. var1 /= ")") THEN
                                                pos1=pos1 + 1
                                                IF(var2 /= var1 .AND. pos1==pos2) THEN
                                                        compteur2 = compteur2 + 1
                                                END IF
                                        END IF
                                END DO
                                pos1=0
                        END IF
                END DO
                pos2=0
        END DO
END DO

pos1=0
pos2=0

taux_erreur2=(compteur2/2)/(num_loci*num_ind)

write(*,*) "nombre réel d'erreur :",compteur2/2
write(*,*) "tx réel d'erreur :",taux_erreur2

CLOSE(1)
CLOSE(2)

!================================================================================
                        On regarde les erreurs de type 1 et de type 2:
!================================================================================

OPEN(UNIT=1, FILE='gen3-phase.txt',STATUS='old')
OPEN(UNIT=2, FILE='gen3.inp',STATUS='old')


READ(2,'(A)') heading
READ(2,'(A)') heading
READ(2,'(A)') heading
READ(2,'(A)') heading
READ(1,'(A)') heading
READ(1,'(A)') heading

DO i=1,num_ind

    READ(1,'(A)') heading
    READ(2,'(A)') heading

    DO k=1,2

        READ(2,'(A)') ligne2
```

```
        READ(1,'(A)') ligne1

        DO j2=1,len(ligne2)

            var2=ligne2(j2:j2)

            IF (var2 /=" ") THEN

                pos2= pos2 + 1

                DO j=1,len(ligne1)-1

                        var1=ligne1(j:j)
                        var3=ligne1(j+1:j+1)

                        IF(var1 /= "(" .AND. var1 /= " " .AND. var1 /= ")" .AND. var3 /= ")") THEN
                                pos1= pos1 + 1
                                IF (var1/=var2 .AND. pos1==pos2) THEN
                                        compteur4=compteur4+1
                                END IF
                        END IF

                        IF (var1 == "(" .AND. var3 /= " ") THEN
                                pos1=pos1 + 1
                                IF(var2 == var3 .AND. pos1==pos2) THEN
                                        compteur3 = compteur3 + 1
                                END IF
                       END IF

                 END DO
                pos1=0
            END IF
        END DO
        pos2=0
    END DO
END DO

taux_erreur3 = (compteur3/2)/(num_loci*num_ind)
taux_erreur4 = (compteur4/2)/(num_loci*num_ind)

write(*,*) " nombre d'erreur de type 1 :", compteur3/2
write(*,*) " taux d'erreur du type 1 :", taux_erreur3
write(*,*) " nombre d'erreur de type 2 :", compteur4/2
write(*,*) " taux d'erreur du type 2 :", taux_erreur4

CLOSE(1)
CLOSE(2)
!================================
        END PROGRAM comparaison
!================================
```