UNIVERSITÉ DU QUÉBEC À MONTRÉAL

CONSTRUCTION D'UN INTERVALLE DE CONFIANCE PAR LA MÉTHODE BOOTSTRAP ET TEST DE PERMUTATION

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN MATHÉMATIQUES

PAR

NATALIYA DRAGIEVA

FÉVRIER 2008

La vie est courte,
La vie est belle,
La vie est pleine de secrets,
Il faut les découvrir ...

REMERCIEMENTS

Une nouvelle page de ma vie était ouverte il y a 2.5 ans. Un pays différent, une langue différente, une culture différente et l'UQAM. Mes premiers pas dans la langue française. Mes premiers pas dans la statistique. J'ai commencé à marcher.

UQAM, merci pour l'accueil chaleureux. Mario, Louise, merci de m'avoir confiée des petits secrets de la langue française.

En marchant, j'ai rencontré des personnages particuliers, des personnages extraordinaires.

Fabrice, merci.
Merci de m'avoir prise en charge. Merci d'avoir partagé tes recherches. Merci pour la confiance et la patience. Merci !.

Sorana, merci.
Merci pour des bonnes discussions et des conseils. Merci pour le support moral. Merci pour la confiance.

Le chemin était partagé avec mes objets précieux.

Aleksandra, Zlatina, mes filles, mes trésors, Plamen, mon cœur, merci.
Merci pour la compréhension et le support. Merci pour la patience et votre amour. Merci à mes parents de m'avoir donnée la connaissance et la puissance de la biologie et des mathématiques.

Mes amies, je vous remercie d'exister, des petits morceaux de moi dispersés par tout.

Merci à des professeurs inoubliables pour leurs qualités professionnelles.

Merci !

TABLE DES MATIÈRES

LISTE DES TABLEAUX vii

LISTE DES FIGURES viii

RÉSUMÉ xii

INTRODUCTION 1

CHAPITRE I
LE MÉCANISME HÉRÉDITAIRE, BIOLOGIE DES POPULATIONS 3

1.1 Notions de biologie 3

1.1.1 Chromosomes, gènes, allèles, homozygotie, hétérozygotie 4

1.1.2 Génome, génotype, phénotype 6

1.1.3 Mutations, polymorphisme 8

1.1.4 Recombinaison, les principes de ségrégation et de sélection. 11

1.2 La cartographie génétique 13

1.2.1 Single Nucleotide Polymorphism 13

1.2.2 Les distances génétiques 17

CHAPITRE II
ÉVOLUTION, MODÈLES MATHÉMATIQUES 20

2.1 L'évolution 20

2.2 La théorie de l'équilibre de liaison 21

2.2.1 Loi de Hardy-Weinberg 24

2.3 Cartographie par déséquilibres de liaison (cartographie par DL) 27

2.4 Modélisation de l'histoire de la population. 29

2.4.1 Processus de coalescence, graphe de recombinaison ancestral. 32

2.4.2 Coalescence et mutations 36

2.4.3 Nombre de sites polymorphes 37

CHAPITRE III
CARTOGRAPHIE GÉNÉTIQUE FINE PAR LA MÉTHODE MAPARG 41

3.1 Suppositions et notations 41

3.2 Modélisation de l'histoire de la population (SNPs) 44

CHAPITRE IV
INTERVALLE DE CONFIANCE BOOTSTRAP 54

4.1 Inférence statistique 54

4.2 Les méthodes d'inférence statistique. 55

4.3 IC par la méthode *bootstrap* 56

4.3.1 Échantillon aléatoire initial de données de taille n 58

4.3.2 Fonction de répartition empirique (EDF), principe du *plug-in* 59

4.3.3 L'échantillon *bootstrap* 60

4.3.4 Moyenne *bootstrap*, écart type *bootstrap*, quantile d'ordre q 65

4.3.5 Intervalle de confiance *bootstrap* pour un paramètre θ arbitraire. 65

4.3.6 Distribution de θ. 66

4.3.7 Choix du B, le nombre de *bootstraps*. 67

4.3.8 L'estimateur *bootstrap* du biais. 69

4.4 Intervalle de confiance *bootstrap* 69

4.4.1 L'intervalle de confiance empirique. 71

4.4.2 Intervalle de confiance *bootstrap* accéléré avec correction de biais. 73

4.4.3 L'intervalle *bootstrap* de confiance bilatéral symétrique de base 75

4.4.4 L'intervalle *bootstrap* de confiance bilatérale symétrique de Student 77

4.4.5 Les differents types d'intervalles de confiance pour r_T 81

CHAPITRE V
TESTS DE PERMUTATION 85

5.1 Test d'hypothèse, principes généraux 85

5.1.1 Niveau effectif de signification d'un test - *probabilité critique* (*p-value*) . . 87

5.1.2 Signification de la *probabilité critique* 88

5.2 Permutation - notions de base 89

5.3 Test de permutation 90

5.3.1 Test de permutation dans le cadre de la méthode *MapArg* 93

5.3.2 La probabilité critique locale, la probabilité critique globale, la probabilité critique ponctuelle 95

5.3.3 Nombre de permutations nécessaire 98

5.3.4 Intervalle de confiance pour la *probabilité critique* 99

5.3.5 Construction d'un échantillon permuté par les méthodes Monte Carlo et échantillonnage pondéré, résultats 102

CONCLUSION 127

APPENDICE - MapArgBP (code C++) 131

BIBLIOGRAPHIE 179

LISTE DES TABLEAUX

1.1 Relations génotype/phénotype 7

2.1 Fréquences génotypiques 25

2.2 Les fréquences des quatre gamètes AB, Ab, aB et ab 28

4.1 L'échantillon initial 59

4.2 Caractéristiques des méthodes pour IC 80

5.1 Les types d'erreurs et le test d'hypothèse 86

LISTE DES FIGURES

1.1 Composantes du mécanisme héréditaire 4

1.2 Le génome humain 4

1.3 Les gènes 5

1.4 L'homozygotie et l'hétérozygotie 6

1.5 Règles générales d'écriture 8

1.6 Quelques types des mutations 9

1.7 Recombinaison 12

1.8 Reproduction diploïde 12

1.9 SNPs 14

1.10 DNA variation 16

1.11 SNPs - autisme 17

2.1 Les types d'erreues 22

2.2 Sélection naturelle 22

2.3 Population est infinitivment grande 23

2.4 Accouplement aléatoire 23

2.5 Migration 24

2.6 Deux populations a) 26

2.7 Deux populations b) 27

2.8 Modèle de Wright-Fisher 30

2.9 Modèle de base 31

2.10 Le processus de coalescence 33

2.11 Topologie 36

2.12 Une généalogie sans et avec mutations 37

2.13 Coalescence 38

2.14 Recombinaison 38

2.15 ARG-exemple 39

3.1 La distance r_m 42

3.2 L'ensemble A_i 44

3.3 L'ensemble B_i 45

3.4 L'opérateur R 46

3.5 L'opérateur C_ij 47

3.6 ARG simulé 52

3.7 La vraisemblance, *MapArg* 53

4.1 Échantillonnage et inférence statistique. 54

4.2 Échantillons bootstrap. 61

4.3 Bootstrap - séquence entière 63

4.4 Bootstrap - par marqueur 64

4.5 Distribution de $\hat{L}^b((\hat{r}_T)_b^*)$, a) 67

4.6 Distribution de $\hat{L}^b(((\hat{r}_T)_b^*), b)$ 68

4.7 Les familles d'intervalle de confiance *bootstrap* 70

4.8 IC empirique 84

5.1 Les types d'erreues 87

5.2 Permutation 89

5.3 Permutations-deux groupes 91

5.4 Re-echantillonnage Monte Carlo 103

5.5 $IC_{95\%}$, ensemble de données $A1$, fenêtres de 2 et 5 104

5.6 Les histogrammes des fréquences des estimateurs bootstraps, l'ensemble des données $A1$, nombres de bootstraps différents 105

5.7 Variabilité de l'estimateur dû à la méthode *MapArg* et à la méthode bootstrap . 106

5.8 $IC_{95\%}$ et la longueur des fenêtres 107

5.9 $IC_{95\%}$ pour les ensembles de données $B1$ et $C1$ 108

5.10 $IC_{95\%}$ pour les ensembles de données $D1$ et $E1$ 109

5.11 $IC_{95\%}$ pour les ensembles de données $G1$ et $H1$ 110

5.12 $IC_{95\%}$ pour les ensembles de données $U1$, $V1$, $W1$ et $X1$ 111

5.13 $IC_{95\%}$ pour les ensembles de données $Y1$ et $Z1$ 112

5.14 $IC_{95\%}$ pour l'ensemble de données $Z1$ 113

5.15 $IC_{95\%}$, l'histogramme des fréquences 114

5.16 $IC_{95\%}$ test 116

5.17 $IC_{95\%}$, *p-valeur* p_{r_i}, $D1$ 117

5.18 $IC_{95\%}$, *p-valeur* p_{r_i}, $U1$ 118

5.19 $IC_{95\%}$, *p-valeur* p_{r_i}, $W1$ 119

5.20 $IC_{95\%}$, *p-valeur* p_{r_i}, $X1$ 120

5.21 $IC_{95\%}$, *p-valeur* p_{r_i}, $Y1$ 121

5.22 $99^{\text{-éme}}$ quantile par position, $A1$ 122

5.23 $99^{\text{-éme}}$ quantile par position, $D1$ 123

5.24 $99^{\text{-éme}}$ quantile par position, $W1$ 124

5.25 $A1$, B=5000, P=5000 125

5.26 $E1$, B=500, P=500 126

RÉSUMÉ

Ce mémoire traite d'une application pratique de deux méthodes statistiques non paramétriques : le bootstrap et le test de permutation. La méthode du bootstrap a été proposée par Bradley Efron (1979) comme une alternative aux modèles mathématiques traditionnels dans des problèmes d'inférence complexe ; celle-ci fournit plusieurs avantages sur les méthodes d'inférence traditionnelles. L'idée du test de permutation est apparue au début du XX-éme siècle dans les travaux de Neyman, Fisher et Pitman. Le test de permutation, très intensif quant au temps de calcul, est utilisé pour construire une distribution empirique de la statistique de test sous une hypothèse afin de la comparer avec la distribution de la même statistique sous l'hypothèse alternative.

Notre objectif est de déterminer l'intervalle de confiance pour un estimateur à maximum de vraisemblance d'une méthode de cartographie génétique existante (*MapArg*, Larribe et al. 2002) et de tester la qualité de cet estimateur, c'est-à-dire d'établir des seuils de signification pour la fonction de la vraisemblance. Les deux méthodes utilisent le calcul répétitif d'une même statistique de test sur des échantillons obtenus à partir de l'échantillon initial, soit avec le «bootstrap», soit avec des permutations. Dans un test d'hypothèse, les deux méthodes sont complémentaires.

Le but de ce mémoire est de proposer différentes variantes pour la construction de l'intervalle de confiance, et de tester des hypothèses distinctes, afin de trouver la meilleure solution adaptée pour la méthode *MapArg*. Pour faciliter la compréhension des décisions prises, un rappel de ... de l'inférence statistique et des tests d'hypothèse est fait dans les chapitres 4 et 5 où la théorie du bootstrap et celle de test de permutation sont présentées. Comme les qualités d'un estimateur dépendent de la méthode utilisée pour le calculer, les chapitres 1 et 2 présentent la base biologique et la base en mathématiques sur lesquelles la méthode *MapArg* est construite, tandis qu'on trouvera dans le chapitre 3 une explication de la méthode *MapArg*.

Mots clés : mutation, recombinaison, coalescence, cartographie génétique, «bootstrap», test de permutation.

INTRODUCTION

Un des problèmes importants de la biologie, l'étude de la transmission du *matériel héréditaire*, est devenu un problème de plus en plus actuel pour les statisticiens. En effet, les scientifiques essayent depuis longtemps de modéliser la transmission du *matériel génétique* afin d'estimer la position d'une *mutation* d'intérêt, et de répondre à des questions quant aux maladies causées par un facteur génétique. Différentes méthodes pour estimer la position probable d'une *mutation* ont été proposées, et fonctionnent avec un certain succès.

Notre travail est concentré sur la méthode *MapArg* (Larribe et al., 2002), une méthode de cartographie génétique fine basée sur la reconstruction de l'histoire de la population (au sens génétique), histoire modélisée par le processus de coalescence avec recombinaison (Griffiths et Marjoram, 1996) : le graphe de recombinaison ancestral (ARG). Supposons que la *mutation* d'intérêt se trouve sur un segment génomique identifié et que l'on dispose de segments d'ADN de membres malades et sains d'une population, on peut alors, en plaçant la *mutation* en diverses positions sur le segment étudié, reconstruire retrospectivement la généalogie de la population en question, conditionnellement à la position de la *mutation*, afin d'obtenir un estimateur à maximum de vraisemblance de la position de la *mutation*. La méthode propose une estimation par intervalle, ce qui reflète l'estimateur obtenu. L'objectif principal de notre projet de recherche est de fournir une méthode pour construire un intervalle de confiance pour l'estimateur obtenu, et de tester la qualité de cet estimateur.

Le mémoire est composé de 5 chapitres. Le chapitre 1 introduira les principes biologiques de base concernant la transmission génétique héréditaire, tandis que le chapitre 2 introduira les concepts mathématiques pour modéliser des processus évolutifs. Le chapitre 3 décrit l'idée derrière la méthode *MapArg*, et les chapitres 4 et 5 discutent différents aspects des méthodes proposées pour tester la confiance et la qualité de l'estimateur de la position probable de la *mutation*. Notons que les deux méthodes, le «bootstrap» pour un intervalle de confiance, et le

«test de permutation» pour la qualité de l'estimateur, sont des méthodes non paramétriques. Des idées nouvelles sont proposées pour les seuils de signification de la fonction de vraisemblance et pour la maximisation de la fonction de vraisemblance.

CHAPITRE I

LE MÉCANISME HÉRÉDITAIRE, BIOLOGIE DES POPULATIONS

1.1 Notions de biologie

Le processus d'expression d'un gène est d'importance fondamentale pour toute la matière organique. Au dix-neuvième siècle, Gregor Mendel a découvert les principes de la transmission du matériel génétique en faisant une série d'expériences avec des pois (Mendel, 1865). Le matériel génétique a été identifié comme étant l'acide désoxyribonucléique (ADN) en 1944 (Avery et al. 1944) et la nature double hélicoïdale de l'ADN a été découverte en 1953 par Francis Crick, James Watson et Maurice Wilkins (prix Nobel de médecine en 1962).

Les erreurs produites par le système reproductif qui mènent vers des changements positifs ou négatifs chez les êtres vivants, sont la source de l'évolution d'une population d'organismes vivants. Pour prévenir les effets négatifs, on étudie aujourd'hui les mécanismes d'hérédité en utilisant différentes méthodes multidisciplinaires. Même si les découvertes scientifiques récentes montrent que l'ADN d'une personne ne contient pas juste deux ensembles de gènes, un de chaque parent, mais également occasionnellement des copies multiples d'un ou plusieurs gènes et des gènes qui sont tout-à-fait absents, on introduira ci-dessous la terminologie de la biologie classique pour les principaux éléments du système héréditaire.

Mots clés : Chromosomes, gènes, allèle, homozygotie, hétérozygotie, génotypes, phénotype, mutation, polymorphisme, recombinaison, coalescence, haplotypes, cartographie génétique, distance génétique, SNPs.

1.1.1 Chromosomes, gènes, allèles, homozygotie, hétérozygotie

Le mécanisme héréditaire d'un organisme est localisé dans la cellule. Les composants principaux sont les *protéines*, les *gènes* et les *chromosomes* (Figure 1.1) .

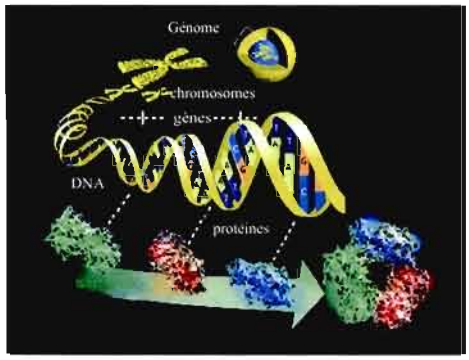


Figure 1.1: Le mécanisme héréditaire.

Les *chromosomes* sont des éléments essentiels du noyau cellulaire, de forme déterminée et en nombre constant pour chaque espèce (23 paires chez l'homme), porteurs des facteurs déterminants de l'hérédité (les *gènes*) (Figure 1.2).

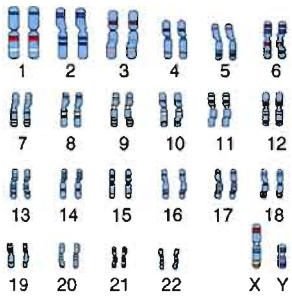


Figure 1.2: Le génome humain compte 23 paires de chromosomes qui peuvent être différenciées par leur longueur et leur motif unique. L'ensemble des chromosomes présenté ci-dessus contient le chromosome Y, il ságit donc du génome d'un homme (ce qui veut dire qu'il provient d'un homme). Les femmes portent quand à elles deux chromosomes X.

Un *gène* est une séquence ordonnée de nucléotides qui occupe une position précise sur un *chromosome* déterminé et qui constitue une information génétique dont la transmission est héréditaire (Figure 1.3). Le *gène* est l'élément qui est à l'origine des caractères congénitaux de chaque personne et est le support de leur transmission de génération en génération. Les *gènes* correspondent à une portion d'une molécule d'ADN (ou parfois d'ARN), possèdent la capacité de se répliquer et sont susceptibles de subir des *mutations*.

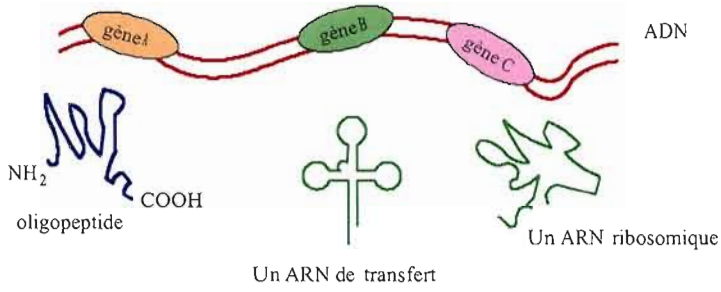


Figure 1.3: Un gène est une partie du chromosome dont la séquence sert à coder une molécule qui intervient dans la vie de la cellule, ce peut être un oligopeptide, un ARN de transfert, un ARN ribosomique, etc.

Chez les organismes possédant des paires de chromosomes, comme l'homme, deux formes sont présentes pour un gène donné. Les paires de *gènes* se situent sur les *chromosomes* à des positions particulières, ou des *loci* (singulier : locus). L'*allèle* est l'un des deux *gènes* localisés au même site sur chaque membre d'une paire de *chromosomes* homologues. Les gènes qui se situent au même locus sont alléliques entre eux, tandis que les gènes qui se situent à différents loci sont non-alléliques entre eux. On distingue deux types d'allèles : des allèles dominants (s'expriment dès qu'ils sont présents) et des allèles récessifs (muets lorsqu'ils sont associés à l'allèle dominant). Il existe parfois des allèles codominants : les deux allèles en présence s'expriment. Les caractéristiques importantes d'un organisme sont l'homozygotie (2 allèles identiques sur chaque chromosome) et l'hétérozygotie (2 allèles différents sur chaque chromosome). Les allèles récessifs ne s'expriment que lorsque la cellule est homozygote pour ce gène (elle possède deux allèles identiques). C'est-à-dire, une personne est homozygote si elle

a à un locus deux formes identiques d'un gène particulier, un hérité de chaque parent.

Exemple 1.1 *Une fille qui est un homozygote pour la fibrose cystique (CF) a reçu le gène CF de ces deux parents et elle a donc la maladie. L'opposé d'un homozygote est un hétérozygote : une personne qui possède à un locus deux formes différentes d'un gène particulier. Par exemple, le père et la mère non affectés de la maladie, mais ayant un enfant atteint sont des hétérozygotes pour la fibrose cystique : chacun porte un gène normal et un gène de CF, mais n'a aucun signe ou symptôme de la maladie (Figure 1.4). Dans ce cas on dit que l'allèle qui s'exprime est l'allèle dominant et l'allèle non exprimé est récessif.*

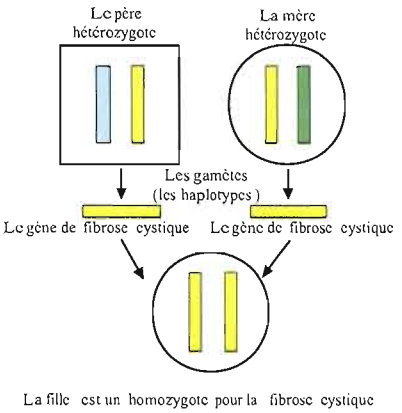


Figure 1.4: Les parents hétérozygotes avec un enfant homozygote

Comme nous l'avons vu, un gène est caractérisé par la fonction qu'il occupe dans le chromosome, ainsi que par la place qu'il occupe dans le chromosome (le locus). Par ailleurs, plusieurs gènes peuvent assurer la même fonction. Le mot gène est courament utilisé pour désigner un locus ou un allèle.

1.1.2 Génome, génotype, phénotype

Le *génome* des organismes vivants est l'ensemble de leur matériel génétique (matériel héréditaire). Le *génotype* est l'information génétique portée par les *gènes* d'un individu et l'expression du *génotype* est appelé le *phénotype*. Le *phénotype* qui représente un ensemble

de caractères apparents d'un individu, correspond à la partie exprimée du *génotype* et à des phénomènes déterminés par le milieu extérieur. Un certain *génotype* ne s'exprime pas toujours dans le même *phénotype*, et sachant le *phénotype*, on ne peut pas déterminer avec certitude le *génotype*. À chaque *phénotype* correspondent en général plusieurs *génotypes* (Tableau 1.1).

Un gène	$\Longleftrightarrow$	*Un phénotype*
Plusieurs genes	$\Longleftrightarrow$	*Un phénotype*
Une mutation d'un gène	$\Longleftrightarrow$	*Un phénotype*
Interaction de plusieurs genes	$\Longleftrightarrow$	*Un phénotype*
Un gène	$\Longleftrightarrow$	*Plusieurs phenotypes distincts*

Tableau 1.1: Rôle du génome, récapitulation des différents cas.

Quand la susceptibilité à la maladie est gouvernée par un seul gène déterminé, le phénotype de toutes personnes porteuses d'un tel gène est le même, c'est le cas de maladies simples, par exemple, la maladie de Huntington (Gusella et al. 1983). Quand plusieurs gènes sont responsables de la même maladie dans les différentes sous-populations, la maladie est hétérogène, la présence de plusieurs gènes résulte en un même phénotype (par exemple, le cancer de sein hérité, Hall et al. (1990), Wooster et al. (1995)). Dans le cas de maladies complexes, l'interaction de plusieurs gènes détermine le phénotype d'intérêt, comme par exemple pour les diabètes de type 2 (Horikawa et al. 2000).

Les génomes des organismes vivants ont des tailles considérables allant d'une centaine de millions à des milliards de nucléotides. Le génome humain, par exemple, est composé d'environ 3 milliards de paires de bases. L'étude d'un génome passe donc par des opérations de cartographie puis de séquençage, ainsi que par l'interprétation des séquences. Pour connaître les «instructions» que renferme un fragment d'ADN, on lit la succession des bases puriques et pyrimidiques : les bases puriques contenues dans l'ADN ou l'ARN sont essentiellement l'adénine (A) et la guanine (G), tandis que les bases pyrimidiques contenues dans l'ADN sont la cytosine (C) et la thymine (T). Il existe des règles générales d'écriture pour représenter des

gènes (Figure 1.5).

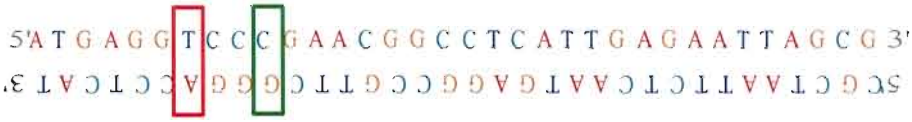


Figure 1.5: Un gène constitue une portion d'ADN double brin dont la séquence (succession des bases) code une fonction. La séquence du second brin peut être déduite de celle du premier, comme l'ordre de nucléotides suit les règles de base d'appariement de Watson-Crick, c'est-à-dire que l'adénine (A) apparaît avec la thymine (T), et que la guanine (G) apparaît avec la cytosine (C).

1.1.3 Mutations, polymorphisme

Des changements quelconques du matériel génétique sont appelés des mutations. La mutation désigne la modification d'un gène, d'un chromosome ou d'un génome par un agent physique ou chimique, entraînant l'apparition d'un ou plusieurs caractères nouveaux et génétiquement stables ; elle peut-être naturelle ou provoquée. Les mutations peuvent être bénéfiques, si elles résultent en une amélioration de la fonctionnalité de la protéine, mais elles sont le plus souvent négatives. Les mutations, survenant de façon aléatoire, créent de nouveaux allèles, c'est-à-dire de nouvelles formes d'un gène donné. Le taux de mutation pour un gène donné est toujours faible, de 10^{-5} à 10^{-11} par génération. Elles peuvent survenir à n'importe quel moment dans n'importe quelle cellule de l'organisme, mais elles ne seront transmissibles que si elles se produisent au niveau de la lignée germinale et la descendance héritera alors de la mutation. Si elles apparaissent dans une cellule somatique (non germinale), elles ne peuvent pas être transmises à la descendance. Différents types d'allèles peuvent être présents à un locus suite à une mutation, ou un changement soudain du matériel génétique. La figure 1.6 illustre quelques types des mutations possibles :

- ***suppression*** - quand une partie de la séquence originale disparaît dans la copie. Quand la suppression se produit, tous les codons s'en trouvent modifiés et la protéine perd, dans la plupart des cas, sa fonctionnalité ;

- ***translocation*** - quand une partie de la séquence originale est répétée quelquefois dans la copie ;
- ***inversion*** - quand une partie de la séquence originale est inversée dans la copie.

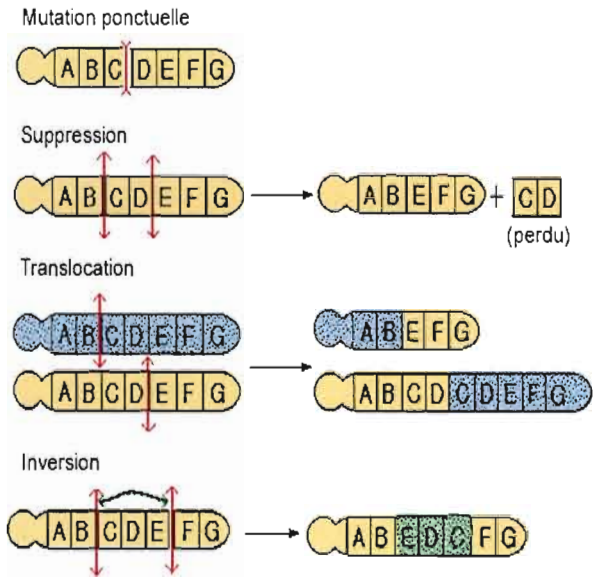


Figure 1.6: Quelques types des mutations

Ces modifications ont souvent des conséquences graves puisqu'elles affectent de nombreux gènes.

Un gène est dit polymorphe lorsqu'il existe dans une population sous plusieurs formes chez au moins 1 % des individus. Ce polymorphisme intervient à différents niveaux. C'est lui qui est responsable de l'unicité de chaque être vivant et qui permet de les distinguer les uns des autres au sein d'une même espèce. À quoi sont dues ces différences ? Les deux raisons pour lesquelles apparait cette diversité sont :

- des différences dans les milieux qui ont interagi différemment (cette hypothèse ne rend

pas compte des différences observées entre membres d'une même sous population qui ont le même mode de vie et le même habitat).

- des différences dans le génome

Le polymorphisme génétique représente la variation entre les individus dans la séquence de gènes. Ces variations qui rendent compte des différents allèles dans une population sont normales et ne sont pas pathogènes. Le polymorphisme est la possibilité d'avoir de multiples allèles d'un gène dans une population, exprimant habituellement différents phénotypes. Cependant, ce qui est important pour un locus est le degré de polymorphisme. Le degré de polymorphisme d'un marqueur dépend du nombre d'allèles différents présents dans ce marqueur et aussi de leur fréquence. L'hétérozygotie H d'un marqueur est définie comme la probabilité que deux allèles retirés indépendamment soient différents. Soit, k le nombre d'allèles dans un marqueur et $p_1, \ldots, p_k$ les fréquences de chaque allèle. La probabilité de retirer un allèle de type i est *P(allèle 1 est de type i)*$= p_i, i = 1, \ldots, k$. On tire une deuxième fois un allèle indépendamment du premier et la probabilité de retirer le même allèle sachant que le premier est de type *i* est

P(allèle 2 est de type i | allèle 1 est de type i)$= p_i^2, i = 1, \ldots, k$.

C'est-à-dire la probabilité de retirer indépendamment deux allèles différents est

H=P(allèle 2 n'est pas de type i | allèle 2 est de type i)$= 1 - \sum_{i=1}^{k} p_i^2$.

Plus la valeur de H est proche de 1, plus le marqueur est polymorphe. Un locus ayant 1000 allèles avec la même fréquence serait considéré beaucoup plus polymorphique qu'un locus auquel il y a deux allèles avec la même fréquence, $H_{1000} = 1 - 1000 * 0.001^2 = 1 - 0.001 = 0.999$, $H_2 = 1 - 2 * 0.5^2 = 1 - 0.5 = 0.5$ et $H_{1000} \geq H_2$. En général on associe le terme mutation a n'importe quel allèle rare, et le polymorphisme à n'importe quel allèle commun.

Le polymorphisme génétique est la présence dans une population d'au moins deux phénotypes alternatifs, génétiquement déterminés par différents allèles. Un locus est dit polymorphe si le ou les allèles rares y ont une fréquence au moins égale à 1 % et si par conséquent, l'hétérozygotie pour cet allèle se produit avec une fréquence d'au moins 2 %. L'homme présente

un polymorphisme phénotypique pour de nombreux caractères. Le polymorphisme est neutre si la présence ou l'absence d'un allèle donné ne signifie pas un avantage ou un désavantage sélectif. On s'accorde généralement pour dire que l'homme constitue une espèce avec un polymorphisme génétique très varié.

Exemple 1.2 *Prenons le gène ADH, responsable du métabolisme de l'alcool et situé sur le chromosome 12. Il existe plusieurs gènes de l'ADH (ADH1-7), parmi lesquels ADH2 et ADH3 sont polymorphes. Les 3 allèles (versions) ADH2*1, ADH2*2 et ADH2*3 du gène ADH2 sont traduits en protéines ne différant que par un seul acide aminé, mais avec des activités différentes, de même que les allèles ADH3*1 et ADH3*2. Par ailleurs, la fréquence des différents allèles diffère selon les ethnies. Chez les Caucasiens et les Afro-américains, on rencontre essentiellement l'allèle ADH2*1 (à faible activité enzymatique), chez les Asiatiques, ADH2*2 est le plus fréquent (et exprime une enzyme à forte activité). Les sujets qui possèdent ADH2*3 (15% des Afro-américains) ont aussi une métabolisation plus rapide. Un polymorphisme génétique a été mis en évidence au niveau du gène de l'ADH2. L'allèle ADH2*1 code pour une enzyme active, présente chez tous les caucasiens alors que l'ADH2*2 code pour une enzyme inactive présente chez 50 % des Asiatiques. Ainsi, chez les individus ayant ce déficit génétique, une consommation excessive d'alcool sera plus dommageable. (Cholé-Doc,2001)*

1.1.4 Recombinaison, les principes de ségrégation et de sélection.

La recombinaison désigne généralement l'échange de matériel génétique entre deux chromosomes homologues lors de la méiose. La recombinaison génétique assure la variabilité des espèces, ce qui fait que les enfants d'un même couple ne seront jamais identiques (à l'exception des jumeaux identiques). La recombinaison est la force génétique formant la variation dans des espèces ; le principe est que deux séquences de longueurs égales produisent une troisième séquence de même longueur, ce qui est montrée sur la figure 1.7 ; au contraire, la figure 1.8 représente un modèle de reproduction diploïde sans recombinaison.

Un haplotype est l'analogue «multilocus» d'un allèle à un locus simple. Il est composé de tous les allèles qui sont transmis ensemble d'un parent à un descendant (Figure 1.8).

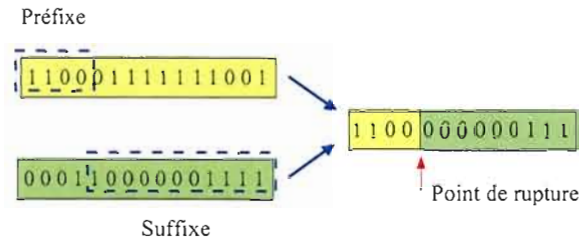


Figure 1.7: La recombinaison

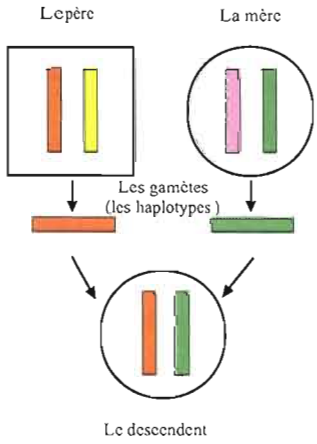


Figure 1.8: Le modèle de reproduction diploïde sans recombinaison

Le principe de ségrégation : pendant la formation des gamètes (des haplotypes), les paires de déterminants héréditaires (les chromosomes) se séparent (ségrègent) de telle manière que chaque gamète est également probable de contenir l'un ou l'autre membre de la paire. Les caractéristiques comme homozygotie et hétérozygotie ne peuvent pas s'appliquer aux gamètes, puisque les gamètes contiennent seulement un allèle de chaque gène. En effet les spermatozoïdes et les ovules ne contiennent qu'une moitié du patrimoine génétique d'un individu.

Le principe de sélection : la ségrégation des membres de n'importe quelle paire d'allèles est indépendante de la ségrégation d'autres paires dans la formation des cellules reproductrices. La détermination de la base génétique d'un caractère, résultante de différents croisements, exige l'analyse des accouplements entre les organismes et, une fois un grand nombre de descendants

obtenus, leur classification par rapport aux phénotypes. L'analyse de la ségrégation par cette méthode n'est pas possible pour les êtres humains. Cependant, le mode de transmission d'un caractère peut être déterminé en examinant l'aspect des phénotypes qui reflète la ségrégation des allèles dans plusieurs générations d'individus relatifs. Ceci est typiquement fait avec un arbre généalogique qui montre le phénotype de chacun individu ; un tel diagramme s'appelle un pédigré.

1.2 La cartographie génétique

La cartographie génétique constitue une autre façon d'étudier le génome. Compte tenu de la complexité des procédés comme cartographie physique, séquençage, interprétation des séquences, il est évident que des approches différentes peuvent se révéler intéressantes pour la connaissance des génomes. On peut, sans disposer d'un séquençage complet ou de cartes physiques très précises, étudier un caractère physiologique ou pathologique particulier. Dans ce cas on fait appel aux méthodes de cartographie génétique pour identifier les gènes qui contrôlent les caractères. Ces méthodes consistent à détecter directement au niveau de l'ADN les polymorphismes, c'est-à-dire les variations génétiques différenciant un individu d'un autre. On définit le marqueur génétique (chromosomique) comme l'élément donnant une information topographique pour un gène. Le marqueur génétique est une séquence unique dans le génome qui peut correspondre à un gène mais aussi être quelconque, c'est-à-dire le marqueur génétique est la modification structurale de l'ADN, possiblement liée à un caractère héréditaire mais sans en être la cause. Cette région sert de balise : on l'ordonne par rapport aux autres marqueurs, et l'ensemble obtenu donne une carte utile pour se situer dans le génome.

1.2.1 Single Nucleotide Polymorphism

Le SNP (Single Nucleotide Polymorphism) est une variation stable de la séquence d'ADN génomique, portant sur une seule base. Il peut y avoir quelques millions de positions de nucléotide dans le génome humain auxquels il y a un certain degré de variation naturelle.

Beaucoup de SNPs n'ont pas d'implications fonctionnelles, mais ils définissent un locus

unique dans le génome et sont polymorphes. Les SNPs qui se trouvent dans les régions codantes (SNPc) et dans les régions régulatrices des gènes seront particulièrement intéressants pour réaliser la cartographie des maladies multifactorielles (étude d'association de gènes candidats impliqués dans ces maladies). Les SNPs (marqueurs de polymorphisme nucléotidique) représentent la variabilité génétique naturelle dans le génome humain. L'expression synonyme «marqueur biallélique» réfère aux deux allèles qui peuvent différer à une position donnée dans une cellule diploïde (Figure 1.9). Les SNPs sont considérés comme la source génétique principale de la variabilité phénotypique qui distingue les individus dans les espèces données.

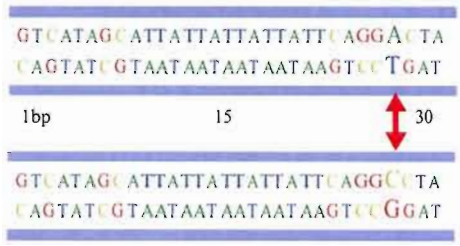


Figure 1.9: Exemple d'un SNP à la position 30 d'une séquence.

Pour comprendre comment avec l'aide de SNPs on localise des gènes de la maladie, on doit comprendre comment des gènes sont hérités. Quand on hérite un caractère ou une maladie, on n'hérite pas simplement l'ADN de ce caractère. Au lieu de cela, on obtient une longue partie de l'ADN qui peut affecter beaucoup de caractéristiques.

Exemple 1.3 *Soit un père qui transmet les yeux bleus à ses enfants, et qui transmet également les cheveux noirs. Dans cet exemple hypothétique, l'ADN des yeux bleus et l'ADN de cheveux noirs composent un bloc d'ADN qui est toujours hérité ensemble. On a hérité cette partie génétique du père, qui a hérité la même partie de son père, etc. jusqu'à l'ancêtre commun, qui a développé pour la première fois ce caractère particulier. Ainsi, même dans une grande population mélangée, les personnes avec cette partie spécifique de l'ADN seraient génétiquement reliées entre eux, parce qu'ils partagent un ancêtre commun avec les cheveux noirs et les yeux bleus observés.*

Les avantages des SNPs :

- très abondants ;
- stables ;
- cartographie avec de plus en plus de précision.

Il existe aussi des marqueurs microsatellites - un type des marqueurs chromosomiques constitué par des segments d'ADN contenant des répétitions en tandem de courts motifs di, tri, ou tetra-nucléotidiques. Ils ont la particularité d'être très nombreux, dispersés sur tout le génome, et aussi d'être le siège des variations à l'origine des polymorphismes multi-alléliques très informatifs. Puisque les SNPs sont bialléliques, ils ne sont pas aussi informatifs que les microsatellites qui peuvent avoir beaucoup d'allèles. Les fréquences des allèles d'un SNP indiqué peuvent différer dans diverses populations. Comment les SNPs peuvent-ils être employés pour identifier les gènes reliés à la maladie ? Les SNPs ont des propriétés et une densité dans le génome humain qui les rendent attrayants comme marqueurs ou outils pour l'identification des gènes dans les parties jusqu'ici non caractérisées du génome. Il y a des grands espoirs que les SNPs soient utiles pour identifier les gènes candidats qui contribuent aux maladies polygéniques (qui sont déterminés par plusieurs gènes).

Actuellement, plusieurs initiatives sont en cours pour exploiter l'information de la variabilité génétique. Leur but est :

- d'identifier les gènes qui contribuent à la maladie ;
- d'identifier le gène ciblé pour développer des nouveaux principes thérapeutiques ;
- d'identifier les gènes qui peuvent influencer des résultats de thérapie.

L'exploitation de SNPs est une procédure en plusieurs phases. Dans la première phase, on créera une carte de SNPs suffisamment dense qui couvrira éventuellement le génome entier. Le choix d'un SNP est déterminé de façon à ce que celui-ci soit le plus informatif comme marqueur génétique. Un SNP avec une fréquence $\leq 10\%$ sera moins informatif qu'un SNP avec une fréquence plus élevée ($30 - 50$ %) dans une population donnée. Dans la deuxième

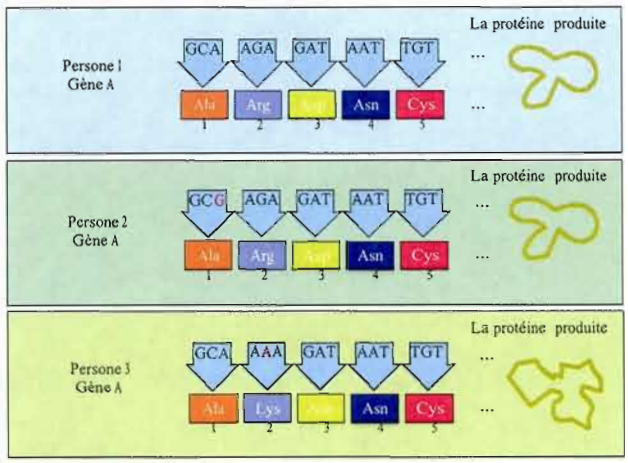


Figure 1.10: Dans un gène, la variation de la séquence ADN peut changer la protéine produite par le code génétique. Par exemple, le changement dans le codon du gène A (position 1) de la deuxième personne, ne change pas la protéine produite ; par contre, le changement dans le codon du gène A (position 2) de la troisième personne change la protéine produite.

phase, les fréquences relatives des SNPs couvrant une grande partie du génome humain seront corrélées avec les maladies spécifiques en comparant des fréquences d'allèles dans des populations en bonne santé et des populations malades. Cette information focalisera l'analyse sur une plus petite partie du génome augmentant en fait la résolution de SNPs utiles. Dans la troisième phase, afin d'étudier la variabilité génétique plus en détails, on prend typiquement, un nombre limité de gènes (10-100) probablement liés avec les maladies spécifiques pour cette étude. Dans cette phase, la variabilité génétique avec une plus basse fréquence est susceptible d'être plus informative. On s'attend à ce que des gènes contribuant à la maladie soient identifiés dans un processus itératif. L'identification de SNPs située dans les régions de codage des gènes (cSNPs) (Figure 1.10) sera particulièrement importante puisqu'elles peuvent représenter une liaison de génotype/phénotype dans les maladies spécifiques (Figure 1.11).

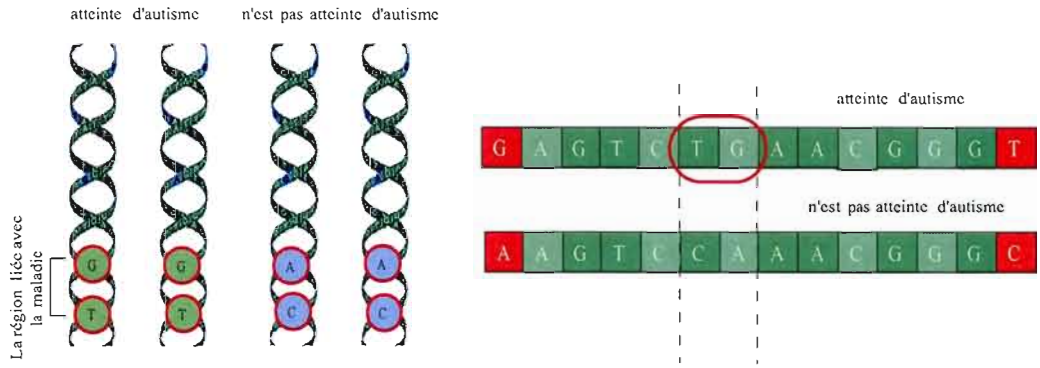


Figure 1.11: Dans les régions déjà soupçonnées d'avoir une liaison avec la maladie on trouve des SNPs qui sont communs pour les personnes atteintes d'autisme, mais sont absentes pour les personnes non atteintes par la maladie, ce qui signifie que le gène responsable pour la maladie se trouve probablement dans la région entre ces SNPs.

1.2.2 Les distances génétiques

Pour construire une carte génétique, on a besoin d'une définition de la distance génétique. L'unité de mesure qu'on utilise est nommée Morgan (M) et signifie le pourcentage de recombinaison par méiose entre deux marqueurs physiquement liés.

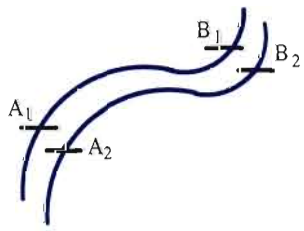
Supposons deux loci A et B, chacun avec deux allèles codominants respectivement A_1, A_2 pour le locus A et B_1, B_2 pour le locus B. Par rapport aux positions du locus A et du locus B sur les chromosomes, on distingue deux situations possibles :

1. Les deux loci A et B sont situés sur des paires de chromosomes différentes. Dans ce cas, quatre types de gamètes peuvent être produits, chacun avec probabilité $\frac{1}{4}$:

$$\begin{array}{cccc} A_1B_1 & A_1B_2 & A_2B_1 & A_2B_2 \\ P(A_1B_1)=\frac{1}{4} & P(A_1B_2)=\frac{1}{4} & P(A_2B_1)=\frac{1}{4} & P(A_2B_2)=\frac{1}{4} \end{array}$$

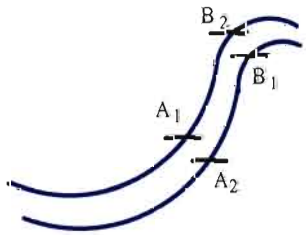
Les quatre gamètes ont la même probabilité $\frac{1}{4}$

2. Les loci A et B sont sur la même paire de chromosomes. Dans ce cas, quatre types de gamètes A_1B_1, A_1B_2, A_2B_1, A_2B_2 peuvent être produits, mais on distingue deux cas :



couplage

(a) Si les allèles A_1 et B_1 sont sur le même chromosome de la paire, on dit que A_1 et B_1 sont en *couplage*

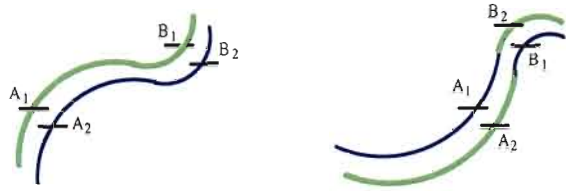


répulsion

(b) Si les allèles A_1 et B_1 sont chacun sur un chromosome différent, on dit que A_1 et B_1 sont en *répulsion*

Lorsque les locus de deux couples d'allèles sont situés sur le même chromosome, et A_1 et B_1 sont en *couplage* et qu'il n'y a pas de recombinaison entre eux, seulement deux types de gamètes peuvent se produire A_1B_1 et A_2B_2. On retrouve chez l'enfant l'allèle A_1 en *coupling* avec allèle B_1 comme chez les parents. Les gamètes A_1B_1 et A_2B_2 sont dits *parentaux*.

Lorsque les loci de deux couples d'allèles sont situés sur le même chromosome, et A_1 et B_1 sont en *coupling* mais qu'il y a recombinaison entre eux, les gamètes produits A_1B_2 et A_2B_1 sont dits *recombinés*. Au cours de la méiose, la recombinaison donne lieu à des génotypes hybrides qui différent des génotypes parentaux. Il s'est passé entre les loci A et B des enjam-



Les gamètes parentaux Les gamètes recombinés

bements ou *crossing-over* en nombre impair : une recombinaison entre deux loci se produit s'il y a un nombre impair de *crossing-over*.

On appelle *taux de recombinaison* la proportion de gamètes recombinés sur l'ensemble des gamètes transmis $\theta = \frac{R}{P+R}$, où R est le nombre de gamètes recombinés, et $P + R$ est le nombre de gamètes transmis (*parentaux* et *recombinés*)

Une distance de 100 centiMorgan (1M) entre deux marqueurs signifie que la probabilité de recombinaison entre ces deux marqueurs, lors de la transmission du chromosome du parent à l'enfant, est très faible soit de l'ordre de 1%. Le Morgan (M) a un équivalent physique, le nombre de paires de base, mais il n'a pas de relation mathématique simple entre ces deux distances. La distance mesurée devient imprécise si les gènes sont éloignés.

Exemple 1.3 *Soit A_1 et B_1 deux marqueurs génétiques et $R = 18$ le nombre d'haplotypes recombinés entre eux par méiose et $P = 2$ le nombre d'haplotypes non recombinés, dans ce cas la distance génétique entre A_1 et B_1 est $D_{A_1,B_1} = \frac{R}{P+R} = \frac{18}{20} = 0.90$, le pourcentage d'haplotypes recombinés entre les marqueurs A_1 et B_1.*

CHAPITRE II

ÉVOLUTION, MODÈLES MATHÉMATIQUES

2.1 L'évolution

Les sciences biologiques définissent généralement l'évolution comme l'ensemble des changements dans la constitution génétique d'un groupe biologique au cours des générations ou l'ensemble des changements graduels qui interviennent au cours des générations, dans les attributs caractéristiques des organismes vivants. Il est clair que les effets de l'évolution se produisent chez les individus d'une population, mais c'est la population dans l'ensemble qui évolue réellement. L'évolution se traduit d'un point de vue génétique par un changement des fréquences des allèles dans le patrimoine héréditaire d'une population.

Exemple 2.1 Supposons qu'un caractère est déterminé par la transmission d'un gène ayant deux allèles - B et b. Si dans la génération des parents la frequence de l'allèle B est $f(B) = 92\%$ et la fréquence de l'allèle b est $f(b) = 8\%$, mais pour la génération de leurs descendants les frequences d'allèles B et b deviennent respectivement $f(B) = 90\%$ et $f(b) = 10\%$, l'évolution s'est produite entre les générations. Les individus ont hérité l'allèle b mais ils ne sont pas évolués. Au contraire, le patrimoine héréditaire de la population entière a évolué dans la direction d'une fréquence plus élevée de l'allèle b. Il s'est produit un changement des fréquences d'allèles, et il y a donc une evolution.

L'évolution représente des changements du patrimoine héréditaire ayant comme résultat l'adaptation progressive des populations à leur environnement. L'évolution se produit parce que la variation génétique existe dans les populations et parce qu'il y a une sélection favorisant

les organismes qui sont mieux adaptés à l'environnement. Comme la variation génétique et la sélection sont des phénomènes de population, elles sont discutées en termes des fréquences des allèles. Quatre processus expliquent la plupart des changements des fréquences d'allèles dans les populations. Ils forment la base des changements cumulatifs des caractéristiques génétiques dans les populations. Ces quatre processus sont :

1. *mutation* - l'origine des nouveaux caractères génétiques dans la population se traduisant par des changements soudains et transmissibles dans les gènes ;
2. *migration* - mouvements des individus de sous populations dans une population plus large ;
3. *sélection naturelle* - habillité des individus à survivre et à se reproduire dans leurs environnements ;
4. *dérive génétique aléatoire* - les changements aléatoires indirects dans les fréquences alléliques, produits dans chaque population par hasard, (en particulier dans les plus petites).

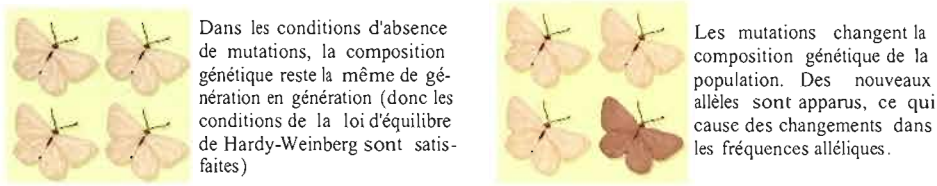
2.2 La théorie de l'équilibre de liaison

La théorie de l'équilibre de liaison a été développée au début du 20-ème siècle par Godfrey Hardy, un mathématicien anglais, et Wilhelm Weinberg, un médecin allemand. Ils ont montré qu'à partir d'une population idéalisée, c'est-à-dire d'une population d'accouplements aléatoires, en absence des forces de l'évolution, les fréquences d'allèle et les fréquences de génotype demeureront constantes de générations en générations. La loi de Hardy-Weinberg de l'équilibre génétique fournit un modèle mathématique permettant d'analyser des changements de fréquence d'allèle durant l'évolution dans une population, en examinant les relations entre des fréquences relatives de deux ou plusieurs allèles.

Les sept conditions que doivent être satisfaites pour que l'évolution ne se produise pas dans une population (une population idéalisée dans l'état d'équilibre de Hardy-Weinberg) sont :

1. *pas de mutation (pas de perte/gain d'allèle).*- pour qu'une population soit à l'équilibre

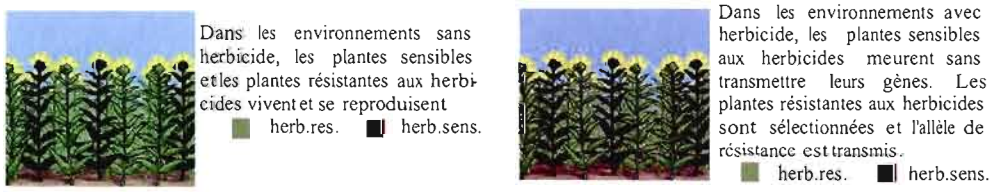
de Hardy-Weinberg, il ne doit y avoir aucun changement des fréquences alléliques dû à la mutation (Figure 2.1).



(c) pas de mutation (d) avec mutation

Figure 2.1: N'importe quelle mutation dans un gène particulier changerait l'équilibre des allèles dans le patrimoine héréditaire. Les mutations peuvent demeurer cachées dans de grandes populations pour un certain nombre de générations, mais peuvent être décelées plus rapidement dans une petite population ;

2. *pas de sélection naturelle (pas de perte/gain d'allèle)*-pour une population à l'état d'équilibre, la sélection ne se produit pas. Aucun allèle ne devrait être sélectionné par rapport aux autres. Si la sélection se produit, les allèles sélectionnés deviennent plus communs ;



(a) pas de sélection (b) avec sélection

Figure 2.2: Par exemple, si la résistance à un herbicide particulier permet aux herbes de survivre dans un environnement pulvérisé avec cet herbicide, l'allèle pour la résistance deviendrait plus fréquent dans la population

3. *la population est infiniment grande* - donc, la population n'est pas sujette à une variation de la fréquence d'un allèle (une taille grande de la population minimise les variations d'échantillonnage) ;

Sur la figure est montrée la distribution de la population de la Californie avant l'année 1500. Un tremblement de terre qui cause la mort de trois sur 20 personnes dans les régions moins peuplées aura un effet significatif sur la composition génétique de la population.

(a) petite population

Sur la figure est montrée la distribution de la population de la Californie pendant l'année 1997. Un tremblement de terre qui cause la mort de trois personnes dans une population de 10 millions n'aura pas un effet significatif sur la composition génétique de cette population.

(b) grande population

Figure 2.3: Une grande population reproductrice aide à s'assurer que le hasard ne pourrait pas perturber l'équilibre génétique. Dans une petite population peuvent exister seulement quelques copies d'un certain allèle, et si pour une raison fortuite les organismes avec cet allèle ne se reproduisent pas avec succès, cet allèle pourrait être perdu. Ce changement aléatoire et non sélectif est ce qui se produit dans la dérive génétique.

4. *Tous les membres de la population se reproduisent ;*
5. *Accouplement totalement aléatoire* - pour une population dans l'état d'équilibre, les accouplements doivent être aléatoires.

Les coraux dispersent leur sperme dans l'océan. Le contact avec les œufs des autres coraux est complètement dû au hasard.

(a) Accouplement aléatoire

Les grands coléoptères tendent à choisir des compagnons de grande taille et les petits coléoptères tendent à choisir des petits compagnons.

(b) Accouplement selectif

Figure 2.4: Si les accouplements sont sélectifs, les individus tendent à choisir des compagnons semblables à eux-mêmes. Bien que ceci ne change pas les fréquences alléliques, cela aura une conséquence sur le taux d'hétérozygocité.

Les accouplements aléatoires réfèrent aux accouplements, qui se produisent dans une population proportionnellement à leurs fréquences génotypiques. Par exemple, si $f(AA) = 0.68$, $f(Aa) = 0.29$ et $f(aa) = 0.03$ sont les fréquences des génotypes AA, Aa et aa

dans une population, alors on s'attend que 46.24 % ($0.68 * 0.68 * 100 = 46.24$) des accouplements se produiraient entre les individus AA. Si une différence significative de cette valeur prévue est rapportée, alors les accouplements ne se fait pas de façon aléatoire dans cette population ;

6. *chacun des membres de la population produit le même nombre de descendants ;*
7. *il n'a aucune migration dans ou hors de la population (pas de perte/gain d'allèle) ;*

(a) pas de migration (b) migration

Figure 2.5: Pour que la fréquence allélique reste constante dans une population à l'équilibre, aucun nouvel allèle ne peut être hérité par la population, et aucun allèle ne peut être perdu. La migration et l'émigration pouvaient changer la fréquence allélique.

Les conditions ci-dessus décrivent une population idéale (l'absence de forces évolutives). Quand les mécanismes de l'évolution n'agissent pas sur une population, l'évolution ne se produira pas, les fréquences de patrimoine héréditaire demeureront sans changement (les fréquences d'allèles demeurent stables en traversant les générations). Pourtant, puisqu'il est peu probable, que ces sept conditions soient satisfaites dans la réalité, l'évolution est un résultat inévitable, et souhaitable pour la survie de l'espèce.

2.2.1 Loi de Hardy-Weinberg

Loi de Hardy-Weinberg - Dans une population diploïde idéale, les fréquences alléliques d'un gène s'exprimant sous la forme de deux allèles A et a sont constantes au fil des générations. Plus précisément, si $p \in [0, 1]$ est la proportion d'allèles A à la génération n, (la proportion de l'allèle a étant $1 - p$) alors c'est encore le cas à chaque génération suivante. De plus, si $f(AA)$,

$f(Aa)$, $f(aa)$ sont les proportions respectives des génotypes AA, Aa et aa dans la population initiale (avec $f(AA) + f(Aa) + f(aa) = 1$) alors les fréquences respectives pour toutes les générations suivantes sont égales à :

$$f(AA)_{n+i} = p^2 = \left(f(AA)_n + \frac{f(Aa)_n}{2}\right)^2,$$

$$f(Aa)_{n+i} = pq = 2\left(f(AA)_n + \frac{f(Aa)_n}{2})(f(aa)_n + \frac{f(Aa)_n}{2}\right)$$

et

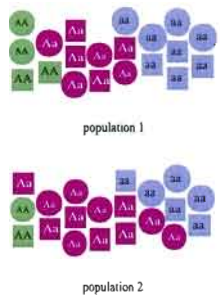
$$f(aa)_{n+i} = q^2 = \left(f(aa)_n + \frac{f(Aa)_n}{2}\right)^2$$

$$pour \quad i = 1, 2, 3, ...$$

Type d'allèle	A	a	Fréquences génotypiques
A	p^2	pq	$f(AA) = p^2$
a	pq	q^2	$f(Aa) = 2pq = 2p(1-p)$
			$f(aa) = q^2$

Tableau 2.1: Fréquences génotypiques dans l'état d'équilibre. (Loi de Hardy-Weinberg.).

Exemple 2.1 Regardons maintenant un gène avec deux allèles A et a, A-dominant, a-récessif (Figure 2.6) dans deux populations différentes, chacune constituée de 20 *individus. Les fréquences alléliques sont calculées comme suit : la fréquence de l'allèle A est* $f(A) = p = 0.4$ *dans les deux populations, la fréquence d'allèle a est* $f(a) = q = 0.6$, *dans les deux populations. On voit que même si les fréquences génotypiques dans les deux populations sont différentes, les fréquences d'allèles A et a sont identiques. Sous les conditions* 1 *à* 7 *ci-dessus, la loi de Hardy-Weinberg décrit comment la reproduction affecte les fréquences d'allèles et la fréquence génotypiques. D'après la loi de Hardy-Weinberg on s'attend que les fréquences*



(a) Deux populations a).

$$f(AA) = \frac{4}{20} = 0.2$$
$$f(Aa) = \frac{8}{20} = 0.4$$
$$f(aa) = \frac{8}{20} = 0.4$$

population 1

$$f(AA) = \frac{2}{20} = 0.1$$
$$f(Aa) = \frac{12}{20} = 0.6$$
$$f(aa) = \frac{6}{20} = 0.3$$

population 2

(b) Les fréquences génotypiques.

$$f(A) = f(AA) + \frac{1}{2} f(Aa) = 0.2 + \frac{1}{2} 0.4 = 0.4$$
$$f(a) = f(aa) + \frac{1}{2} f(Aa) = 0.4 + \frac{1}{2} 0.4 = 0.6$$

population 1

$$f(A) = f(AA) + \frac{1}{2} f(Aa) = 0.1 + \frac{1}{2} 0.6 = 0.4$$
$$f(a) = f(aa) + \frac{1}{2} f(Aa) = 0.3 + \frac{1}{2} 0.6 = 0.6$$

population 2

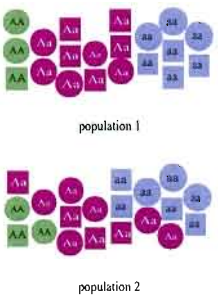
(c) Les fréquences alléliques.

Figure 2.6: Dans les deux populations les individus homozygotes par rapport à l'allèle A sont de couleur verte, les homozygotes par rapport à l'allèle a sont en bleu et les individus hétérozygotes (Aa) sont en violet.

des allèles restent inchangées, mais les fréquences des génotypes se stabiliseront après une génération.

Prenons maintenant les mêmes populations, mais après une génération (Figure 2.6). On remarque que les fréquences génotypiques dans les deux populations deviennent les même, bien qu'elles ont étaient différentes dans la génération précédente. Selon la loi de Hardy-Weinberg, les fréquences alléliques dans les deux populations restent inchangées par rapport à celle de la génération précédente. C'est-à-dire la proportion de génotypes peut-être prédite par la loi de Hardy-Weinberg comme suit $f(AA) = p^2, f(Aa) = 2pq, f(aa) = q^2$. *Comme les deux popula-*

tions sont maintenant à l'état d'équilibre de Hardy-Weinberg, la proportion de chaque génotype restera la même dans les générations qui suivent et ces proportions dépendent seulement des fréquences alléliques.



(a) Après une génération.

$$f(AA) = \frac{3}{20} = 0.15 \approx 0.16 = 0.4^2 = p^2$$
$$f(Aa) = \frac{10}{20} = 0.5 \approx 0.48 = 2 \times 0.4 \times 0.6 = 2pq$$
$$f(aa) = \frac{7}{20} = 0.35 \approx 0.36 = 0.6^2 = q^2$$

population 1

$$f(AA) = \frac{3}{20} = 0.15 \approx 0.16 = 0.4^2 = p^2$$
$$f(Aa) = \frac{10}{20} = 0.5 \approx 0.48 = 2 \times 0.4 \times 0.6 = 2pq$$
$$f(aa) = \frac{7}{20} = 0.35 \approx 0.36 = 0.6^2 = q^2$$

population 2

(b) Les fréquences des génotypes.

Figure 2.7: Après une génération, les proportions des différentes génotype ont changées dans chacune des populations.

On appelle déséquilibre de liaison (noté D), le paramètre qui mesure l'écart à l'association aléatoire de deux allèles situés à des loci différents sur les gamètes. Il mesure le degré de dépendance statistique entre deux caractères, qui sont ici les allèles à deux loci différents. Remarquons que l'existence d'un déséquilibre de liaison entre deux loci n'implique pas une liaison génétique entre les deux loci.

2.3 Cartographie par déséquilibres de liaison (cartographie par DL)

Le déséquilibre de liaison (DL) défini l'association statistique d'allèles. La cartographie par DL se fonde sur l'hypothèse que les régions voisines à un gène d'intérêt sont transmises d'une génération à l'autre en même temps que ce gène. Ces régions peuvent être identifiées par l'ensemble spécifique de marqueurs (haplotypes) qu'ils contiennent. Ainsi, si l'on trouve des haplotypes qui sont partagés plus fréquemment entre des groupes d'individus qui présentent une maladie ou un caractère génétique donné, on peut utiliser ces haplotypes pour localiser les gènes pathologiques. La cartographie par DL diffère de la cartographie classique par analyse de liaison

génotype	A	a	*fréquence*
B	$f_{AB} = p_A p_B + D$	$f_{aB} = q_a p_B - D$	p_B
b	$f_{Ab} = p_A q_b - D$	$f_{ab} = q_a q_b + D$	q_b
fréquence	p_A	q_a	1

Tableau 2.2: Considérons uniquement ce qui se passe à deux loci biallélіques Aa et Bb. On appellera p_A (q_a) la fréquence de l'allèle A (a), et p_B (q_b) la fréquence de l'allèle B (b). Les fréquences des quatre gamètes AB, Ab, aB et ab sont décrites complètement à l'aide du tableau.

familiale qui a été employée pour trouver des gènes associés avec des maladies génétiques simples (par exemple : l'anémie falciforme et la fibrose kystique). En effet, la cartographie par DL est effectuée sur des échantillons obtenus dans la population plutôt que sur des échantillons obtenus dans des familles. La cartographie par DL utilise l'information provenant de l'histoire des événements génétiques dans la population, qui est beaucoup plus riche comparativement au faible nombre d'événements génétiques survenus dans l'histoire de la famille utilisée dans la cartographie par analyse de liaison génétique familiale. Comme conséquence, la cartographie par DL possède une puissance statistique supérieure pour détecter des gènes associés à des maladies courantes. Voici les deux principaux inconvénients de la cartographie par DL :

1. le DL ne concerne qu'une faible distance chromosomique, ce qui exige une cartographie avec des marqueurs très denses, ce qui est plus coûteux ;
2. le nombre d'haplotypes présents dans une population mixte multiethnique est très grand, ce qui fait que de nombreux gènes seront statistiquement impossibles à distinguer du *bruit de fond* génétique.

Les études de caractère sont basés sur les principes différents comme suit :

- elaborer différents modèles correspondant aux différentes hypothèses génétiques ;
- tester les différentes hypothèses en comparant leur vraisemblance.

La cartographie par DL a été employée sous une forme ou une autre dans la plupart des découvertes de gènes de maladies courantes qui ont été annoncées à ce jour (seuls environ 300 gènes ont été découverts sur un total d'environ 2 000 à 4 000 gènes de maladies).

2.4 Modélisation de l'histoire de la population.

L'application de la théorie mathématique et des méthodes statistiques tentant de ré- soudre des problèmes concernant la localisation des gènes qui causent la maladie, exige un modèle mathématique de l'histoire de la population sur lequel on étudie le comportement d'hérédité dans les générations. Un modèle de base dans la théorie génétique des populations est le modèle de Wright-Fisher. Ce modèle a des hypothèses similaires à des celles utilisées pour obtenir l'équilibre de Hardy-Weinberg, à l'exception importante de la taille finie N de la population. Voici les hypothèses du modèle :

- la taille N de la population est finie et constante ;
- les individus sont haploïdes (haploïde - cellule ou organisme contenant un chromosome de chaque paire de chromosomes homologues trouvés dans la cellule diploïde normale) ;
- accouplement aléatoire par rapport au gène étudié ;
- les générations ne se chevauchent pas ;
- pas de mutation ;
- pas de sélection.

On étudiera un locus ayant deux allèles A et a. Considérons N individus diploïdes ($2N$ allèles), donc la taille de la population est $2N$ et $n = 0, 1, 2, ...$ les générations successives. Les phénomènes de mutation et de sélection sont négligés dans un premier temps.

Soit X_n la variable aléatoire qui compte le nombre d'allèles A à la génération n. La population à la génération $n+1$ est déduite de celle de la génération n par un tirage binomial avec remise de $2N$ gènes disposés dans une urne dont la proportion d'allèles A est $\frac{X_n}{2N} = \frac{i}{2N}$. La probabilité de copier un allèle A dans la prochaine génération $n+1$ est $\pi_i = \frac{i}{2N}$. De plus,

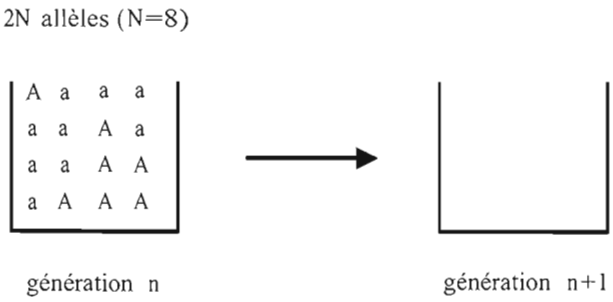


Figure 2.8: L'état de la population dans la génération n peut-être représenté par une *urne* qui contient $2N$ boules : i de type A et $2N - i$ de type a. Puis, pour construire la $(n + 1)$- ème génération, on choisit au hasard de l'urne $2N$ boules avec remise.

les adultes produisent un nombre infini de gamètes ayant la même fréquence d'allèle. De cet ensemble de gènes, $2N$ gamètes sont tirés au hasard pour construire N individus diploïdes pour la prochaine génération. Évidemment la probabilité de choisir j allèles de type A dans la prochaine génération suit la loi binomiale, c'est à dire

$$p_{ij}(X_{n+1} = j | X_n = i) = \binom{2N}{j} \times \pi_i^j \times (1 - \pi_i)^{2N-j}, \tag{2.1}$$

$$i = 0, 1, 2, ..., 2N, \quad j = 0, 1, 2, ..., 2N$$

Le processus $\{X_n, n = 0, 1, ...\}$ est une chaîne de Markov avec la matrice de transition $P = (p_{ij})$ dans l'espace $S = \{0, 1, \ldots, 2N\}$.

La moyenne et la variance d'une variable aléatoire binomiale avec les paramètres $2N$ et π_i sont respectivement $2N\pi_i$ et $2N\pi_i(1-\pi_i)$. Ainsi, la moyenne et la variance de $\pi_j = \frac{j}{2N}$ par rapport à $\pi_i = \frac{i}{2N}$ sont

$$E(\pi_j) = E\left(\frac{j}{2N}\right) = \frac{E(j)}{2N} = \frac{2N\pi_i}{2N} = \pi_i \tag{2.2}$$

et

$$Var\left(\pi_j\right) = Var\left(\frac{j}{2N}\right) = \frac{Var(j)}{4N^2} = \frac{2N\pi_i(1-\pi_i)}{4N^2} = \frac{\pi_i(1-\pi_i)}{2N}, 0 \leq i, j \leq N \quad (2.3)$$

Comme on peut le voir $E(\pi_j) = \pi_i$, c'est-à-dire $E(X_{n+1}|X_n) = X_n$, $n = 1, 2, 3, ...$, la fréquence moyenne d'allèle demeure la même, ce qui signifie que les changements de la fréquence d'allèle sont non dirigés et démontre que le passé et le futur sont indépendants conditionnellement au présent. D'intérêt particulier est la probabilité de la fixation d'une mutation c'est-à-dire le remplacement d'un type d'allèle par exemple A avec un autre sur l'échelle de population. En plus d'être une chaîne de Markov, la suite $\{X_n, n = 0, 1, ...\}$ est également une martingale. Ce qui implique que l'espérance de X_n est constante au cours du temps et égale à $E(X_0)$. Cette propriété de $\{X_n, n = 0, 1, ...\}$ permet de déterminer la probabilité de fixation (et la probabilité de disparition) de l'allèle A (a). La probabilité, sachant que $X_0 = i$, que X atteigne $2N$ est égale à $\frac{i}{2N}$ (Une bonne explication pour la probabilité de fixation d'un allèle peut être trouvée dans le livre de Tavaré " Ancestral Inference in Molecular Biology ", 2001.).

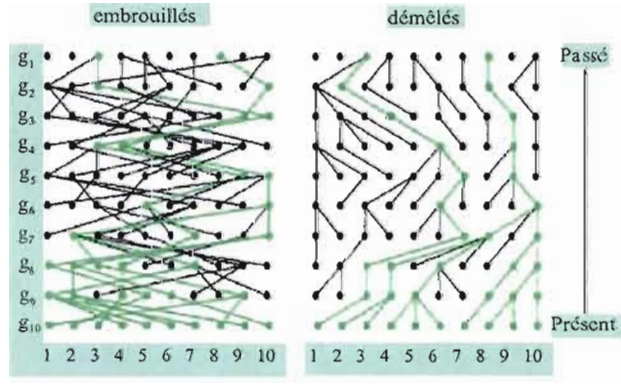


Figure 2.9: On voit une population haploïde de 10 individus pendant 10 générations. Chaque individu a un parent, mais un parent peut avoir plus d'une descendance. Les mutations peuvent se produire (représentées par des individus avec une couleur différente) et disparaître encore. Il est impossible de prédire quelle population va être obtenue en particulier. Selon le modèle, les populations évoluent indépendamment les unes des autres.

Une représentation graphique du modèle de Wright-Fisher est donnée sur la figure 2.9. Prenons en compte maintenant la mutation. Supposons en plus des hypothèses du modèle de base de Wright-Fisher que l'allèle A mute en allèle a avec probabilité u, et a en A avec probabilité v. Il convient alors dans la définition de la chaîne de remplacer π_i par $\pi_i = \frac{i(1-u)+(2N-i)v}{2N}$. Le nouveau processus est aussi une chaîne de Markov mais cette fois ci, tous les états sont récurrents et apériodiques (le nouveau processus n'est plus une martingale).

La dérive génétique est le processus cumulatif qui implique la perte de quelques gènes dû au hasard et la réplication des autres sur des générations successives dans une petite population. Le processus de dérive génétique est un processus stochastique (ou aléatoire), à la différence des processus évolutifs modélisés dans le cadre de populations supposées infinies (comme ceux de la sélection ou de la mutation en grandes populations (processus déterministes)). La dérive génétique est le seul processus qui régit le destin des mutations. Des changements résultant de la dérive génétique peuvent avoir comme conséquence des pertes d'allèles et une diminution de la variabilité génétique de la population, de sorte que la population soit poussée vers l'homozygotie et loin de l'hétérozygotie.

2.4.1 Processus de coalescence, graphe de recombinaison ancestral.

Deux individus dans une population finie qui évolue selon le modèle de Wright-Fisher, ont toujours un ancêtre commun dans le passé (Figure 2.10). Ainsi, toute population finie possède un arbre généalogique dont la racine est un unique ancêtre ; le processus de convergence de l'arbre généalogique vers une racine est nommé le processus de coalescence.

Le processus de coalescence (modèle aléatoire de généalogie décrivant les relations de parenté entre les individus) a été présenté par Kingman (1982) comme description mathématique de l'évolution généalogique d'une population des gènes, et il est devenu un outil standard pour l'analyse des données moléculaires. Plusieurs méthodes existantes ont démontré l'efficacité des analyses évolutionnaires basées sur le processus de coalescence (par exemple Griffiths et Tavaré, 1994, Kuhner et al. 1995, Donnelly et Stephens, 2000). En ajoutant l'influence de la recombinaison sur le processus de coalescence, on obtient un modèle plus probable de l'histoire

de la population. À l'origine, le processus de coalescence avec recombinaison a été proposé par Hudson (1983, 1991). Le modèle a été encore développée par Griffiths et Marjoram (1996), et on le connaît sous le nom de «graphe de recombinaison ancestral» (ARG, pour Ancestral Recombination graph). La caractéristique importante de ARG est le temps aléatoire nécessaire pour monter vers l'ancêtre commun, mesuré en unités de N générations.

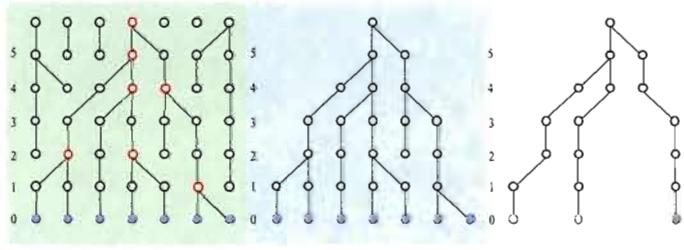


Figure 2.10: En remontant dans le passé, on voit que les lignées vont progressivement fusionner les unes avec les autres par une série de coalescence (marqués en rouge), jusqu'à un seul ancêtre commun, à la génération six. On ne s'intéresse qu'aux lignées qui laisseront des descendants à la génération actuelle. On ne considère qu'un échantillon tiré au hasard de la population, et pas la généalogie de la population entière.

Kingman a décrit formellement le processus de coalescence en 1982 pour un échantillon de taille n tiré d'une population diploïde de taille N (ou d'une population haploïde de taille $2N$). Il s'agit d'une marche aléatoire dans le passé ou l'on va passer par des états successifs avec n lignées, $n-1$ lignées, $n-2$ lignées, etc., jusqu'à l'ancêtre commun où on n'aura plus qu'une seule lignée. Bien entendu, le passage d'un état avec j lignées à un état avec $j-1$ lignées correspond à un évènement de coalescence. Au cours du processus de coalescence, on va donc séjourner pendant un certain temps T_n à un état avec n lignées, puis un temps T_{n-1} à un état avec $n-1$ lignées, etc., pour finir par un temps T_2 pendant lequel on n'aura plus que 2 lignées avant l'ultime évènement de coalescence.

Kingman a dérivé la distribution de probabilité de ces temps T_j en faisant les hypothèses suivantes :

1. Le modèle démographique sous-jacent correspond au modèle de Wright-Fisher.

2. La taille de l'échantillon est beaucoup plus petite que la taille de la population ($n << N$), de telle sorte qu'il ne peut pas y avoir qu'un seul évènement de coalescence par génération.

On s'intéresse à la probabilité $P(j)$ qu'un évènement de coalescence se produise parmi les j lignées de la génération précédente. Pour une paire de lignées quelconque, c'est la probabilité que ces deux lignées soient dérivées d'une même copie d'un individu de la génération précédente, c'est-à-dire qu'ils sont identiques par ascendance à la génération précédente. Cette probabilité est égale à $\frac{1}{2N}$. De plus, un lignée peut coalescer avec n'importe quelle autre lignée, et ceci avec la même probabilité. Si l'on a j lignées, on peut former $\frac{j(j-1)}{2}$ paires différentes, ce qui représente le nombre de combinaisons possible de deux lignées parmi j et $P(j) = \frac{j(j-1)}{4N}$. De plus, la probabilité qu'il n'y ait aucun évènement de coalescence est $1 - P(j)$. Le temps de coalescence T_j peut être mesuré comme le nombre de générations écoulées jusqu'à ce que l'on ait un évènement de coalescence. C'est donc une variable aléatoire qui est le nombre d'essais nécessaires pour observer un succès de probabilité $P(j)$. Une telle variable aléatoire suit une loi géométrique qui a la distribution suivante : $P(T_j = t) = [1 - P(j)]^{t-1} P(j)$. C'est-à-dire, pendant $t - 1$ générations il n'y a pas eu de coalescence et il y en a eu une à la t-ième. L'espérance et la variance d'une telle loi géométrique sont connues et égales à

$$E(T_j) = 1 + \frac{1 - P(j)}{P(j)} = \frac{1}{P(j)} = \frac{4N}{j(j-1)}, \tag{2.4}$$

et

$$V(T_j) = \frac{1 - P(j)}{P(j)^2} = \frac{1 - \frac{j(j-1)}{4N}}{\left(\frac{j(1-j)}{4N}\right)^2} = 4N\frac{4N - j(j-1)}{j^2(j-1)^2}. \tag{2.5}$$

Si la taille de la population est grande, le temps d'une génération est presque négligeable par rapport au temps total de la généalogie. Dans ce cas, le temps jusqu'à l'ancêtre commun est une variable aléatoire continue et on utilise la loi exponentielle avec le paramètre $\lambda = \frac{j(j-1)}{4N}$ et

la fonction de densité $f_j(t) = \lambda e^{-\lambda t} = \frac{j(j-1)}{4N} e^{-t\frac{j(j-1)}{4N}}$ pour calculer l'espérance et la variance de temps de coalescence $T_j, j = 2, 3, ...$ comme suit :

$$E(T_j) = \int_{t=0}^{\infty} f_j(t) t dt = \int_{t=0}^{\infty} \frac{j(j-1)}{4N} e^{-t\frac{j(j-1)}{4N}} t dt = \frac{4N}{j(j-1)} = \frac{1}{\lambda}, \tag{2.6}$$

et

$$V(T_j) = \frac{1}{\lambda^2} = \frac{16N^2}{j^2(j-1)^2}. \tag{2.7}$$

Si l'unité de mesure du temps de coalescence est $2N$ générations (la taille de la population) on a que $E(T_j) = \frac{2}{j(j-1)}$. On voit que les temps de coalescence moyens augmentent exponentiellement quand on remonte dans le passé. Donc dans une population de taille constante, on s'attend à ce que la majorité des évènements de coalescence surviennent relativement tôt et que les derniers soient très espacés. Notamment, le temps moyen pour la dernière coalescence est égal à $2N$ générations, avec toutefois une variance égale à $2N(2N-1)$, soit près du carré de la moyenne. Le processus généalogique a donc une très forte variabilité. Ceci implique que les généalogies de locus indépendants pourront être très différentes. Par exemple, on peut représenter les généalogies tirées de 3 échantillons de 10 gènes simulés pour la même population stationnaire, mais pour 3 locus différents.

On peut également dériver la taille totale T_n de la généalogie, c'est-à-dire le temps jusqu'à l'ancêtre commun le plus récent (MRCA, pour Most Recent Common Ancestor) de tout l'échantillon. On a bien évidement $T = \sum_{j=2}^{n} T_j = 4N(1 - \frac{1}{n})$. Lorsque n est grand, on a donc $T \approx 4N$, ce qui correspond au temps moyen de fixation d'un nouveau mutant de fréquence initiale $\frac{1}{2N}$ dans une population. On voit donc de nouveau la relation entre processus de dérive et processus de coalescence. On notera aussi que comme la probabilité de coalescence de n'importe quelle paire de lignées est identique, toutes les topologies de généalogies ayant les mêmes temps de coalescence sont équiprobables (Des exemples des topologies généalogiques probables pour un échantillon sont montrés sur la figure 2.11).

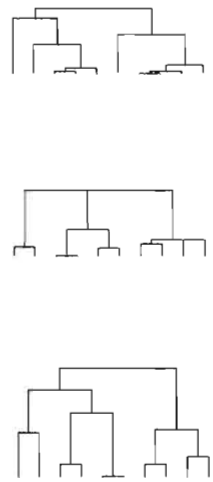


Figure 2.11: Les arbres générés par le logiciel TreeToy

2.4.2 Coalescence et mutations

Jusqu'à ici, nous n'avons pas parlé de mutations dans le processus de coalescence, car pour des gènes neutres, le processus de coalescence ne dépend pas du processus de mutation. On peut considérer les deux processus comme totalement indépendants car la longueur des branches d'une généalogie dépendra uniquement du processus démographique et pas du processus de mutation. L'addition de mutations au processus de coalescence s'effectue donc d'une manière très simple. On suppose que, pour une généalogie donnée, les mutations se produisent aléatoirement sur la longueur des branches (Figure 2.11).

On fait habituellement l'hypothèse que les mutations se produisent suivant une loi de Poisson de paramètre $\lambda = ut$, où u est le taux de mutation par unité de temps, et t la longueur d'un segment de branche quelconque. Dans la version continue du processus de coalescence, où le temps est mesuré en unité de $2N$ générations, le paramètre λ devient $\lambda = uT = \frac{\theta\tau}{2}$, avec $\theta = 4Nu$ et $\tau = \frac{T}{2N}$.

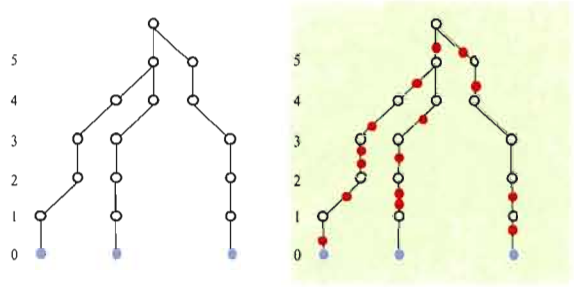


Figure 2.12: Une généalogie sans mutation et une généalogie avec mutation.

2.4.3 Nombre de sites polymorphes

On s'intéresse aussi au nombre de sites polymorphes S dans un échantillon. Le modèle de mutation le plus simple est le modèle nommé «modèle des sites infinis». Ce modèle a été introduit par Kimura en 1968. Selon ce modèle, toute nouvelle mutation se produit à un nouveau site qui n'a encore jamais été touché par une mutation, donc chaque nouvelle mutation produit un nouvel allèle. Sous le modèle de sites infinis, le nombre de sites polymorphes S d'un échantillon est simplement le nombre de mutations s'étant produites dans la généalogie des gènes de l'échantillon. L'espérance de cette variable aléatoire est une fonction de la longueur totale de la généalogie T et du taux de mutation u :

$$E(S|\theta,n) = E\left(\frac{\theta T}{2}\right) = \frac{\theta}{2}E(T) = \frac{\theta}{2}\sum_{j=2}^{n} jT_j = \frac{\theta}{2}\sum_{j=2}^{n}\frac{2j}{j(j-1)} = \theta\sum_{i=1}^{n-1}\frac{1}{i} \tag{2.8}$$

Ce qui a été obtenu de manière sensiblement plus compliquée par Watterson en 1975. Un bon estimateur de θ basé sur le nombre observé de sites polymorphes est donc :

$$\hat{\theta}_S = \frac{S}{\sum_{i=1}^{n-1}\frac{1}{i}}.$$

Pour compléter la modélisation de l'histoire de la population par le processus de coalescence, regardons maintenant les événements possibles en remontant dans le temps :

- *Coalescence* - on a vu que dans le cas d'une population idéale, totalement isolée, tous les gènes présents à un locus finiront par provenir d'un seul gène ancêtre. Ce phénomène est appelé la coalescence des gènes d'une population. Le processus de coalescence est le processus qui caractérise la fusion de deux lignées ancestrales en une seule dans une représentation de la généalogie d'un ensemble de gènes. Les événements de coalescence réduisent la dimension de l'échantillon d'une unité (Figure 2.13).

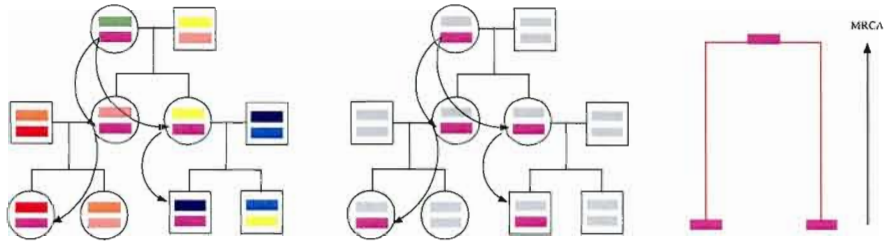


Figure 2.13: Coalescence.

- *Recombinaison* - le processus par lequel les gènes se combinent entre eux de façon différente à chaque génération. La recombinaison redistribuera une séquence à deux séquences, où une séquence portera le matériel à la gauche du point de recombinaison et l'autre le matériel à la droite de ce point (Figure 2.14).

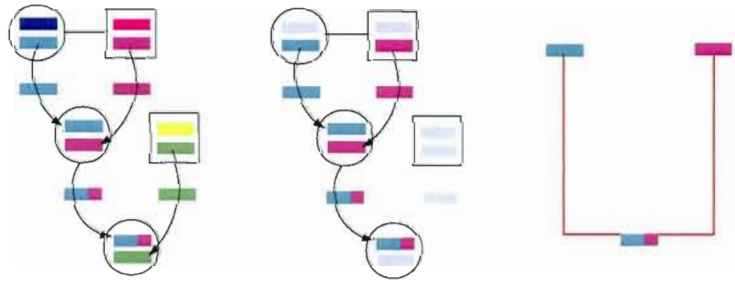
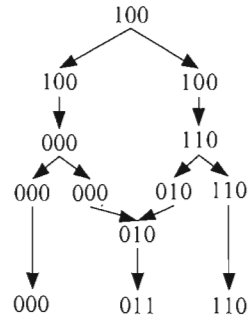


Figure 2.14: Recombinaison.

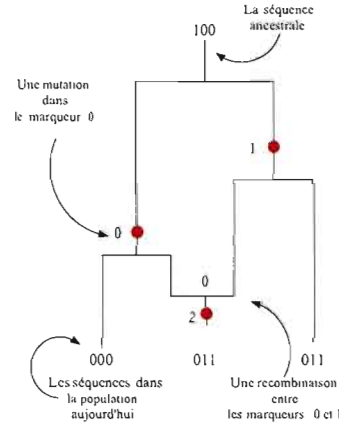
- *Mutation* - la mutation (en reculant dans le temps) change une séquence à une certaine position de l'état muté à l'état ancestral.

Supposons que nous avons aléatoirement tirés des individus de la population et que la

population est de taille constante. Supposons également que toutes mutations sont neutres (une mutation neutre est une mutation qui n'a aucun effet sur la forme physique darwinienne de ses porteurs ou une mutation qui n'a aucun effet phénotypique.). On regarde alors dans le passé le processus stochastique qui décrit l'ascendance, ou la généalogie, de l'échantillon. C'est le graphique connu sous l'appellation ARG. Les branches de ce graphique désignent des lignes d'ascendance.



(a) representation numerique



(b) ARG

Figure 2.15: Un ARG possible pour trois séquences. En déplaçant dans le temps (vers le haut de l'ARG), les points représentent une mutation à la position donnée - 0 pour le premier marqueur, 1 pour le deuxième et 2 pour le troisième ($100 \xrightarrow{0\bullet} 000$, $011 \xrightarrow{2\bullet} 010$, $100 \xrightarrow{1\bullet} 110$), deux lignées jointes représentent une coalescence et la division d'une lignée est un événement de recombinaison : les marqueurs à gauche (respectivement la droite) du point de recombinaison sont hérités de la gauche (respectivement de droite).

Les marqueurs ne sont pas indépendants. Les marqueurs proches tentent de se transmettre ensemble. La connaissance d'un marqueur c'est la connaissance partielle des autres marqueurs et la dépendance entre eux diminue avec la distance.

L'ARG est une généalogie complète pour les séquences échantillonnées. Si en partant du

bas d'un ARG, on suit la généalogie à un unique marqueur, on obtient un arbre local, qui correspond en fait à un arbre de coalescence, car pour un seul locus, on n'a plus de recombinaison.

CHAPITRE III

CARTOGRAPHIE GÉNÉTIQUE FINE PAR LA MÉTHODE MAPARG

La méthode *MapArg* (Larribe *et al.*, 2002) décrite dans ce chapitre est basée sur des suppositions idéalisées. Il est cependant possible, en ajoutant de nouvelles hypothèses, d'approcher la complexité d'un modèle de transmission génétique (de génération en génération) plus réaliste, afin d'obtenir une bonne estimation de la position de la mutation.

3.1 Suppositions et notations

Prenons une population d'individus qui se comportent selon le modèle de Wright-Fisher. La taille $N(t)$ de la population antécédente au temps t est donnée par la formule $N(t) = N(0)exp(-kt)$, où la constante k est positive si la population augmente dans le temps et négative si la population diminue. Pour $k = 0$, la taille de la population est constante ($N(t) = N, \forall t$). On utilisera le ratio $\lambda(t) = N(0)/N(t)$ dans la suite.

Chaque individu est représenté par une séquence de L marqueurs (la phase d'haplotypes est connue). Les marqueurs sont codés par 0, 1 et -1 (0 pour le matériel génétique transmis inchangé du MRCA, 1 pour le matériel génétique transmis muté du MRCA et -1 pour le matériel génétique non ancestral).

Exemple 3.1 : Exemple d'une séquence de 5 marqueurs est la suite "1 1 0 0 0".

Les marqueurs sont ordonnés par leur distance par rapport au marqueur le plus à gauche de la séquence et étiquetés (marqueur 0 est le marqueur qui débute la séquence à gauche, il est

suivi par le marqueur 1 etc.). La distance entre deux marqueurs consécutifs m et $m+1$ est notée r_m (r_m est le taux de recombinaison par méiose entre les marqueurs m et $m+1$), les distances entre les marqueurs d'une séquence sont supposées additives (une hypothèse raisonnable quand la séquence génétique considérée est assez petite). La longueur d'une séquence de L marqueurs est $r = \sum_m r_m$ (Figure 3.1).

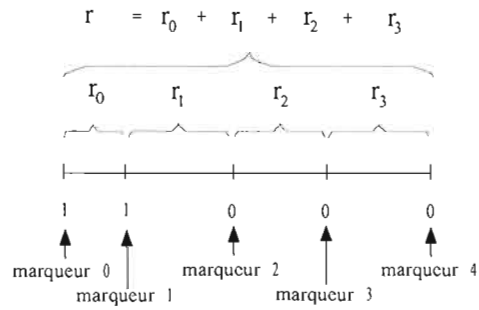


Figure 3.1: La distance r_m.

Le taux de *recombinaison* ajusté par génération entre deux marqueurs consécutifs m et $m+1$ est $\rho_m = 4N(0)r_m$, c'est-à-dire pour une séquence de L marqueurs, le taux de recombinaison par génération est $\rho = \sum_m \rho_m$.

Le taux de *mutation* ajusté pour un marqueur, disons m, est $\theta_m = 4N(0)u_m$, où u_m est le taux de mutation spécifique par génération pour le marqueur m, cela revient à dire que pour une séquence de L marqueurs le taux de recombinaison par génération est $\theta = \sum_m \theta_m$.

La *mutation* est supposée assez rare de telle sorte qu'on la considère non récurrente : chaque site polymorphique (marqueur) ne peut muter qu'une seule fois dans l'arbre de coalescence d'un échantillon de la population en question. Les séquences sont divisées en deux sous-ensembles, le sous-ensemble de cas et le sous-ensemble de témoins, par rapport au phénotype exprimé : malade et non malade, respectivement. Dans chaque sous-ensemble peuvent exister des séquences de même type, c'est-à-dire des séquences ayant le même code ; le même

nombre (multiplicité) de séquences d'un même type i est notée par n_i. La notation s_i signifie une séquence de type i.

La *pénétrance*, c'est-à-dire la probabilité qu'un individu présente le phénotype pathologique sachant qu'il a le génotype à risque, est supposé complète. La pénétrance est dite complète si cette probabilité est égale à 1, et incomplète si elle est inférieure à 1. Lorsqu'un gène dominant n'entraîne aucune manifestation, il est dit non pénétrant. La pénétrance incomplète peut alors expliquer des anomalies de transmission ou de distribution dans des maladies dominantes.

Remarque : Une pénétrance complète fait que que la maladie sera exprimée chez tous les individus porteurs de l'anomalie. Une pénétrance à 70% signifie que 7 porteurs sur 10 exprimeront la maladie.

Sous l'hypothèse que la *pénétrance* est complète, les individus de même phénotype portent le gène de la maladie avec une probabilité égale à 1. Cela signifie que les séquences qui appartiennent à un sous-ensemble de cas portent la mutation avec certitude et que les séquences du sous-ensemble de contrôles ne sont pas mutantes. Pour éviter les faux positifs et créer un modèle idéalisé, on suppose de plus qu'il n'y a pas de *phénocopies*. Dans une analyse de liaison génétique, les *phénocopies* seront des faux positifs, puisqu'ils ne sont pas porteurs des gènes causant la maladie, mais sont malades.

Pour chaque type i de séquence, on définit l'ensemble A_i, qui contient la position des marqueurs ancestraux (du MRCA, codés par 0), et l'ensemble B_i, qui contient les segments ancestraux (Figure 3.2 et Figure 3.3).

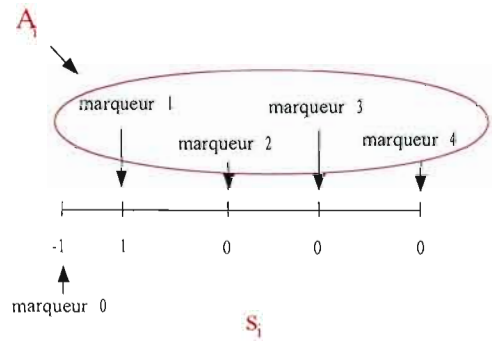


Figure 3.2: L'ensemble A_i pour la séquence s_i contient les marqueurs 1, 2, 3 et 4.

3.2 Modélisation de l'histoire de la population (SNPs)

Les événements qui sont utilisés pour décrire la transmission génétique au fils des années sont la *recombinaison*, la *coalescence* et la *mutation*. On définit respectivement trois opérateurs R, C et M. L'opérateur R représente les événements de recombinaison dans l'histoire de la population, l'opérateur C est responsable des événements de coalescence antérieurs et l'opérateur M est l'opérateur qui gère le chemin de la mutation vers le MRCA.

1. L'opérateur R est une fonction d'une seule séquence s_i et d'un intervalle $I_{m,m+1}$ entre deux marqueurs consécutifs m, $m+1$ qui contiennent le point de recombinaison. Une propriété importante de l'opérateur R est qu'il affecte l'histoire de l'échantillon si et seulement si le point de recombinaison appartient au segment ancestral de la séquence (Figure 3.4).

   Comme conséquence, pour un échantillon de taille $n = \sum_i n_i$ séquences, le taux de changement par l'opérateur R en reculant en arrière dans le temps vers le moment t sans avoir de changement est $\frac{n\beta\rho}{2}$, où $\beta = \sum_i \frac{n_i}{n} \sum_{m \in B_i} \frac{\rho_m}{\rho}$.

   Un événement de recombinaison augmente le nombre d'ancêtres avec 1.

2. L'opérateur C est une fonction de deux séquences, soit de même type i (C_i), soit de deux types différents i et j ($C_{i,j}$). En employant l'opérateur C sur deux séquences du

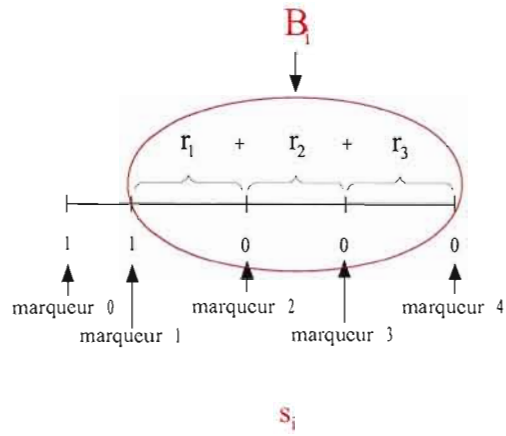


Figure 3.3: L'ensemble B_i pour la séquence s_i contient les marqueurs 1, 2, 3 et 4.

même type, on obtient une séquence de même type : $C_i = C(s_i, s_i) = s_i$. En employant l'opérateur C sur deux séquences de types différents, on obtient une séquence soit de type qui existe déjà, soit d'un type qui n'existe pas encore : $C_{i,j} = C(s_i, s_j) = s_l$ (Figure 3.5).

C'est-à-dire que l'opérateur C diminue le nombre de séquences de 1 chaque fois qu'il est utilisé. Pour un échantillon de taille $n = \sum_i n_i$ séquences, le taux de changement par l'opérateur C en allant en arrière dans le temps vers le moment t sans avoir des changements est $\frac{n(n-1)\lambda(t)}{2}$.

3. L'opérateur M est une fonction d'un marqueur m muté d'une séquence s_i et change son état muté vers l'état ancestral ($M(s_i, m) = 0$), il ne change pas le nombre total de séquences. L'opérateur M est employé sur le marqueur m d'une séquence s_i si et seulement si la séquence s_i est la seule et unique séquence avec un marqueur muté à la position m. Comme conséquence, pour un échantillon de $n = \sum_i n_i$ séquences, le taux de changement par l'opérateur M en retournant dans le temps vers le moment t sans avoir des changements est $\frac{n\alpha\theta}{2}$, où $\alpha = \sum_i \frac{n_i}{n} \sum_{m \in A_i} \frac{\theta_m}{\theta}$.

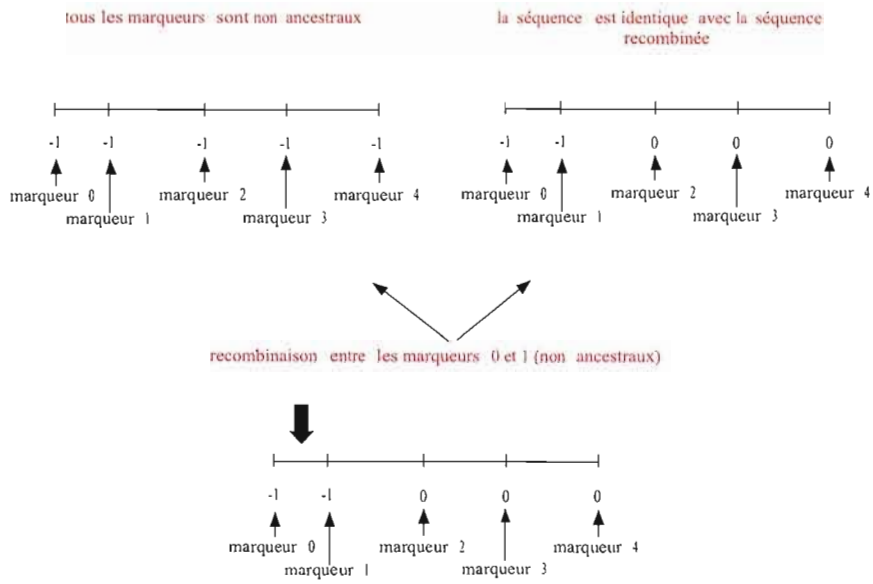


Figure 3.4: L'opérateur R.

Le taux total de changement du matériel ancestral est $\frac{nD(t)}{2}$, où $D(t) = (n-1)\lambda(t) + \alpha\theta + \beta\rho$. Soit t_τ et $t_{\tau+1}$ les temps des événements consécutifs τ et $\tau+1$ qui changent le matériel ancestral. Les probabilités que le changement au temps $t_{\tau+1}$ soit causé par les opérateurs R, C ou M sont respectivement :

$$P_\tau(R) = \int_{t_\tau}^{\infty} \left[\frac{\beta\rho}{D(t_{\tau+1})}\right] g(t_{\tau+1}|t_\tau)dt_{\tau+1}, \tag{3.1}$$

$$P_\tau(C) = \int_{t_\tau}^{\infty} \left[\frac{(n-1)\lambda(t_{\tau+1})}{D(t_{\tau+1})}\right] g(t_{\tau+1}|t_\tau)dt_{\tau+1}, \tag{3.2}$$

$$P_\tau(M) = \int_{t_\tau}^{\infty} \left[\frac{\alpha\theta}{D(t_{\tau+1})}\right] g(t_{\tau+1}|t_\tau)dt_{\tau+1}, \tag{3.3}$$

ou $g(t_{\tau+1}|t_\tau)dt_{\tau+1}$ est la fonction de densité de $t_{\tau+1}|t_\tau$, déterminée par le taux total de changement et qui peut être simulé par des variables indépendantes suivant la loi uniforme (Larribe *et al.*, 2002).

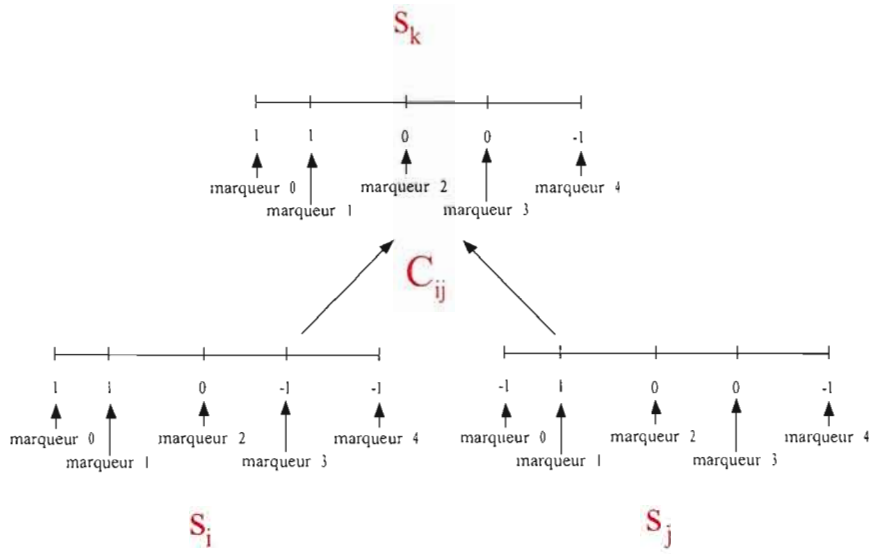


Figure 3.5: Illustration d'une coalescence de deux séquences i et j vers une séquence de type $k \neq i, j$.

Pour $\lambda(t) = 1$, c'est-à-dire quand la population est de taille constante, les probabilités $P_\tau(R)$, $P_\tau(C)$ et $P_\tau(M)$ deviennent beaucoup plus simples :

$$P_\tau(R) = \frac{\beta\rho}{n - 1 + \alpha\theta + \beta\rho}, \tag{3.4}$$

$$P_\tau(C) = \frac{n - 1}{n - 1 + \alpha\theta + \beta\rho}, \tag{3.5}$$

$$P_\tau(M) = \frac{\alpha\theta}{n - 1 + \alpha\theta + \beta\rho}. \tag{3.6}$$

À chaque étape de la construction d'un ARG de l'échantillon des séquences, les probabilités de chaque événement sont calculées et un des opérateurs R, C et M est choisi, ce qui nous donne un système d'équations récurrentes :

$$
\begin{aligned}
Q(H_\tau) = \quad & P_\tau(C) \sum_{n_i>1} \tfrac{(n_i-1)}{n-1} Q(H_\tau + C_i) + \\
& + P_\tau(C) \sum_{i\neq j, compatible} \tfrac{2(n_k+1-\delta_{ik}-\delta_{jk})}{n-1} Q(H_\tau + C_{ij}^k) + \\
& + P_\tau(M) \sum_{m\in A_i, unique} \tfrac{\theta_m}{\alpha\theta} \tfrac{(n_j+1)}{n} Q(H_\tau + M_i^j(m)) + \\
& + P_\tau(R) \sum_{m\in B_i} \tfrac{\rho_m}{\beta\rho} \tfrac{(n_j+1)(n_k+1)}{n(n+1)} Q(H_\tau + R_i^{jk}(m)).
\end{aligned}
$$

Pour simplifier la lecture et éviter des formules compliquées, un exemple est montré en détails.

Exemple 3.2 : Prenons les données suivantes :

s_0	0 0 0 0	$n_0 = 2$	cas
s_1	0 0 0 1	$n_1 = 1$	cas
s_2	0 0 1 0	$n_2 = 1$	cas
s_3	0 0 0 1	$n_3 = 1$	témoin
s_4	1 1 0 1	$n_4 = 1$	témoin
$r_0 = 7.382e^{-006}$	$r_1 = 9.6382e^{-005}$	$r_2 = 2.1236e^{-005}$	$r = 0.000125$
$N = 10000$	$\rho = 4Nr = 5$	$u_m = 5e^{-005}$	

Bien que la mutation cherchée est située entre les marqueurs 0 et 3 par hypothèse (c'est-à-dire que l'on a 3 intervalles qui peuvent contenir la mutation), sa position est inconnue ; l'ARG est construit premièrement sous la condition que la vraie position de la mutation est localisée dans l'intervalle $I_{0,1}$, deuxièmement sous la condition que la vraie position de la mutation est dans l'intervalle $I_{1,2}$ et enfin sous la condition que la vraie position de la mutation se trouve dans l'intervalle $I_{2,3}$. Pour l'ARG—$mutation \in I_{0,1}$ un marqueur muté est inséré au milieu de l'intervalle $I_{0,1}$ de toutes les séquences des cas, et un marqueur non muté est inséré au milieu de l'intervalle $I_{0,1}$ de toutes les séquences des témoin, ce qui modifie les données de façon

suivante :

s_0	0 1 0 0 0	$n_0 = 2$	cas
s_1	0 1 0 0 1	$n_1 = 1$	cas
s_2	0 1 0 1 0	$n_2 = 1$	cas
s_3	0 0 0 0 1	$n_3 = 1$	témoin
s_4	1 0 1 0 1	$n_4 = 1$	témoin
$r_0 = 3.691e-006$	$r_1 = 3.691e-006$	$r_2 = 9.6382e-005$	$r_3 = 2.1236e-005$
$r = 0.000125$	$N = 10000$	$\rho = 4Nr = 5$	$u_m = 5e-005$
$\rho_0 = 0.14764$	$\rho_1 = 0.14764$	$\rho_2 = 3.85528$	$\rho_3 = 0.84944$
$\alpha = 1$	$\beta = 1$		

Le calcul est montré pour la première itération de la construction de l'ARG. Commençons avec l'ensemble initial de séquences. Les événements possibles sont :

- Coalescence de deux séquences du même type.

  C_0 coalescence de deux séquences de type s_0

  La probabilité partielle de cet événement est calculée par le premier terme de l'équation de récurrence comme suit : $P(C_0) = P_\tau(C)\frac{n_0-1}{n-1} = \frac{1}{20}$

- Mutation.

  $M_{s_3}^{m=3}$ marqueur 3 mute dans la séquence s_3

  $M_{s_4}^{m=0}$ marqueur 0 mute dans la séquence s_4

  $M_{s_4}^{m=2}$ marqueur 2 mute dans la séquence s_4

  La probabilité partielle de cet événement est calculée par le troisième terme $P_\tau(M) \sum_{m \in A_i, unique} \frac{\theta_m}{\alpha\theta} \frac{(n_j+1)}{n} Q(H_\tau + M_i^j(m))$ de l'équation de récurrence comme suit :

- Recombinaison. La recombinaison est possible dans chaque intervalle et pour chaque type

$$M_{s_3}^{m=3} = P_\tau(M)\frac{\theta_3}{\alpha\theta}\frac{n_0+1}{n} = \frac{1}{20}$$
$$M_{s_4}^{m=0} = P_\tau(M)\frac{\theta_0}{\alpha\theta}\frac{n_j+1}{n} = \frac{1}{60}, \quad n_j = 0$$
$$M_{s_4}^{m=2} = P_\tau(M)\frac{\theta_2}{\alpha\theta}\frac{n_j+1}{n} = \frac{1}{60}, \quad n_j = 0$$

de séquence. La probabilité partielle de ces événements est calculée par le quatrième terme $P_\tau(R)\sum_{m\in B_i}\frac{\rho_m}{\beta\rho}\frac{(n_j+1)(n_k+1)}{n(n+1)}Q(H_\tau + R_i^{jk}(m))$ de l'équation de récurrence comme suit :

$R_{s_0}^{I_{0,1}}$	la séquence s_0 recombine dans intervalle $I_{0,1}$	$P(R_{s_0}^{I_{0,1}}) = 1.75762e^{-4}$
$R_{s_0}^{I_{1,2}}$	la séquence s_0 recombine dans intervalle $I_{1,2}$	$P(R_{s_0}^{I_{1,2}}) = 1.75762e^{-4}$
$R_{s_0}^{I_{2,3}}$	la séquence s_0 recombine dans intervalle $I_{2,3}$	$P(R_{s_0}^{I_{2,3}}) = 0.00458962$
$R_{s_0}^{I_{3,4}}$	la séquence s_0 recombine dans intervalle $I_{3,4}$	$P(R_{s_0}^{I_{3,4}}) = 0.001011242$
$R_{s_1}^{I_{0,1}}$	la séquence s_1 recombine dans intervalle $I_{0,1}$	$P(R_{s_1}^{I_{0,1}}) = 1.75762e^{-4}$
$R_{s_1}^{I_{1,2}}$	la séquence s_1 recombine dans intervalle $I_{1,2}$	$P(R_{s_1}^{I_{1,2}}) = 1.75762e^{-4}$
$R_{s_1}^{I_{2,3}}$	la séquence s_1 recombine dans intervalle $I_{2,3}$	$P(R_{s_1}^{I_{2,3}}) = 0.00458962$
$R_{s_1}^{I_{3,4}}$	la séquence s_1 recombine dans intervalle $I_{3,4}$	$P(R_{s_1}^{I_{3,4}}) = 0.001011242$

$R_{s_2}^{I_{0,1}}$	la séquence s_2 recombine dans intervalle $I_{0,1}$	$P(R_{s_2}^{I_{0,1}}) = 1.75762e^{-4}$
$R_{s_2}^{I_{1,2}}$	la séquence s_2 recombine dans intervalle $I_{1,2}$	$P(R_{s_2}^{I_{1,2}}) = 1.75762e^{-4}$
$R_{s_2}^{I_{2,3}}$	la séquence s_2 recombine dans intervalle $I_{2,3}$	$P(R_{s_2}^{I_{2,3}}) = 0.00458962$
$R_{s_2}^{I_{3,4}}$	la séquence s_2 recombine dans intervalle $I_{3,4}$	$P(R_{s_2}^{I_{3,4}}) = 0.001011242$

$R_{s3}^{I_{0,1}}$	la séquence s_3 recombine dans intervalle $I_{0,1}$	$P(R_{s3}^{I_{0,1}}) = 1.75762e^{-4}$
$R_{s3}^{I_{1,2}}$	la séquence s_3 recombine dans intervalle $I_{1,2}$	$P(R_{s3}^{I_{1,2}}) = 1.75762e^{-4}$
$R_{s3}^{I_{2,3}}$	la séquence s_3 recombine dans intervalle $I_{2,3}$	$P(R_{s3}^{I_{2,3}}) = 0.00458962$
$R_{s3}^{I_{3,4}}$	la séquence s_3 recombine dans intervalle $I_{3,4}$	$P(R_{s3}^{I_{3,4}}) = 0.001011242$
$R_{s4}^{I_{0,1}}$	la séquence s_4 recombine dans intervalle $I_{0,1}$	$P(R_{s4}^{I_{0,1}}) = 1.75762e^{-4}$
$R_{s4}^{I_{1,2}}$	la séquence s_4 recombine dans intervalle $I_{1,2}$	$P(R_{s4}^{I_{1,2}}) = 1.75762e^{-4}$
$R_{s4}^{I_{2,3}}$	la séquence s_4 recombine dans intervalle $I_{2,3}$	$P(R_{s4}^{I_{2,3}}) = 0.00458962$
$R_{s4}^{I_{3,4}}$	la séquence s_4 recombine dans intervalle $I_{3,4}$	$P(R_{s4}^{I_{3,4}}) = 0.001011242$

La somme $Q(H_0)$ de toutes les probabilités est calculée : $Q(H_0) = 0.163095$. Un événement est choisi au hasard, par exemple $M_{s4}^{m=2}$, et on passe à l'étape suivante en arrière dans le temps. La vraisemblance de l'échantillon au point supposé d'être la position de la mutation est le produit des probabilités totales à chaque pas en arrière du temps jusqu'au MRCA. La figure 3.1 illustre un ARG simulé par le programme *MapArg* sur les données de l'exemple 3.2. Un graphique de la vraisemblance de ces mêmes données est montré sur la figure 3.7.

Notons pour finir ce chapitre que la méthode *MapArg* est beaucoup plus complexe que la présentation qui vient d'être faite, mais qu'une présentation détaillée de la méthode serait trop longue dans le cadre de ce travail. Nous avons présenté la base sur laquelle la méthode est construite. La principale omission est le fait que la méthode utilise des vraisemblances composites ; bien que le processus d'estimation change, le concept de la méthode et le calcul des probabilités illustrées ci-dessus sont identiques.

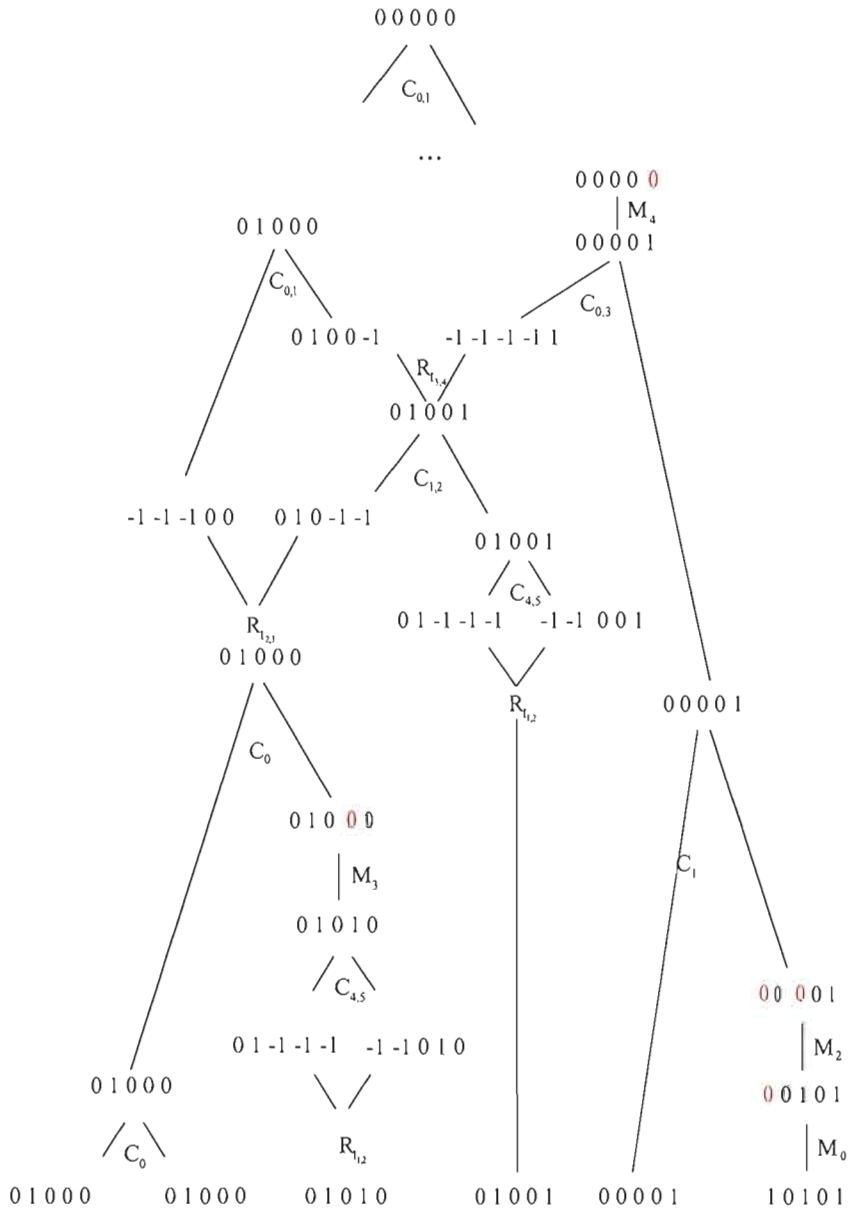


Figure 3.6: L'ARG simulé par la méthode *MapArg* pour des données montrées dans l'exemple 3.2. Après 24 itérations, toutes les séquences de l'échantillon initial coalescent vers le MRCA.

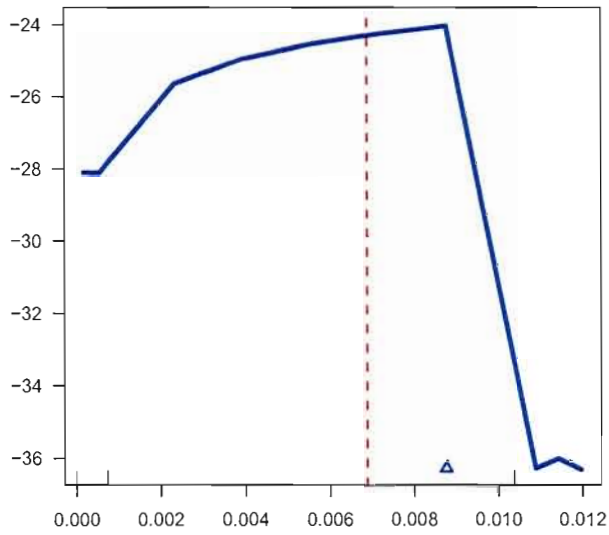


Figure 3.7: La ligne bleue représente les valeurs de la fonction de la vraisemblance aux points d'évaluation, calculée pour des données de l'exemple 3.2, la ligne rouge montre la vraie position de la mutation et le triangle bleu - la position estimée par la méthode *MapArg*. Même si le nombre de séquences et des marqueurs n'est pas significatif, l'estimateur est bon (situé dans le même intervalle tant comme la vraie position de la mutation).

CHAPITRE IV

INTERVALLE DE CONFIANCE BOOTSTRAP

4.1 Inférence statistique

Notre objectif, comme l'objectif de beaucoup d'études scientifiques, est d'établir des règles générales à partir d'observations particulières. Lorsqu'on dit que la mutation est localisée, c'est parce que les observations réalisées sur un échantillon d'haplotypes permettent d'affirmer avec suffisamment de confiance que l'existence de la mutation à un certain locus aura un effet déterminé sur les individus de la population. Les méthodes d'inférence statistique permettent de faire certaines affirmations concernant les caractéristiques d'une population à partir d'observations réalisées sur un échantillon (Figure 4.1), mais la nature aléatoire des variables étudiées provoque une incertitude dans l'inférence, et les conclusions tirées au sujet de la population peuvent être fausses.

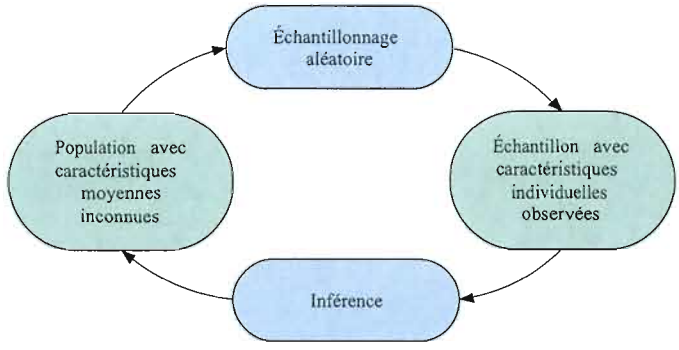


Figure 4.1: Échantillonnage et l'inférence statistique.

Pour que les conclusions tirées soient correctes, les dépendances inévitables introduites dans le plan expérimental doivent être contrôlées, et prises en compte dans l'analyse statistique. Dans la plupart des tests statistiques, l'indépendance des observations est une condition essentielle quant à la validité des tests. Si des dépendances existent, l'analyse peut être rendue impossible. Le but de l'intervalle de confiance est d'indiquer la confiance que l'on peut avoir dans les résultats et le but d'un test d'hypothèse est de déterminer s'il est vraisemblable ou non que le paramètre d'un modèle prenne des valeurs différentes dans des populations distinctes. La différence fondamentale entre l'intervalle de confiance et le test d'hypothèse est que le premier est centré sur une statistique connue (mesurée dans un échantillon) tandis que le second est centré sur le paramètre d'une population.

4.2 Les méthodes d'inférence statistique.

Selon les suppositions d'une méthode statistique, on distingue trois grandes classes des méthodes comme suit :

1. ***Les méthodes paramétriques standards*** basées sur la connaissance de la distribution a priori et des paramètres du modèle. On suppose que la loi de probabilité fait partie d'une famille paramétrique de lois et on essaie d'estimer les paramètres (par exemple la moyenne et l'écart type pour la distribution normale).
2. ***Les méthodes non paramétriques standards*** qui ont été développées pour être employées dans les cas où l'on ne connaît pas les paramètres de la loi de la variable d'intérêt (par conséquent nommées non paramétriques). En termes plus techniques, les métho-des non paramétriques ne se fondent pas sur l'évaluation des paramètres (tels que l'écart type ou la moyenne) décrivant la distribution de la variable de l'intérêt pour une population. Par conséquent, ces méthodes sont également appelées des méthodes à paramètre libres ou des méthodes de distributions libres.
3. ***Les méthodes de ré échantillonnage***, quand on a des hypothèses faibles, ou aucune hypothèse n'est fait au sujet de la distribution fondamentale de la population.

4.3 IC par la méthode *bootstrap*

La méthode du *bootstrap* a été proposée par Bradley Efron (1979) comme une alternative aux modèles mathématiques traditionnels dans des problèmes d'inférence compliqués où une modélisation mathématique de la distribution des erreurs est difficile. Ce dernier a écrit beaucoup au sujet de la méthode et de ses généralisations (Efron, 1982, Efron, 1985, etc.). Des milliers de papiers ont été rédigés sur le *bootstrap* dans les dernières décennies. La méthode et ses modifications ont trouvé une utilisation très large dans des problèmes appliqués en remplaçant les méthodes asymptotiques usuelles. Le *bootstrap* est une méthode de type *Monte Carlo* basée sur les données observées (Efron et Tibshirani 1993, Mooney et Duval 1993). Dépendant de l'information disponible au sujet de la loi de la population ou du paramètre d'intérêt, il existe la méthode de *bootstrap* paramétrique et la méthode de *bootstrap* non paramétrique. Le nom *bootstrap* provient de l'expression *«to pull oneself up by one's bootstrap»* (*«Les aventures du baron Munchausen»*, par Rudolph Erich Raspe). Dans ce livre, le baron est tombé au fond d'un lac profond. Quand tout semblait perdu, une idée géniale lui est venue à l'esprit : se soulever en tirant sur les languettes de ses bottes. Le *bootstrap* fournit plusieurs avantages par rapport à l'approche paramétrique traditionnelle. Il est facile de décrire la méthode et celle-ci s'applique aux situations compliquées, car des suppositions quant à la distribution des données, telles que la normalité ou d'autres, ne sont pas nécessaires (la distribution a priori est inconnue).

Le *bootstrap* non paramétrique appartient à un sous-ensemble de méthodes non paramétriques qui est défini par Dudewicz (1976) comme un sous-ensemble de méthodes qui fournissent des procédures statistiques d'inférence basées sur des hypothèses faibles (ou aucune hypothèse) sur la distribution fondamentale de la population. On devrait employer des procédures non paramétriques seulement quand les suppositions au sujet de la loi fondamentale sont sérieusement douteuses quant à leur validité. Quand les hypothèses ne sont pas violées, les procédures non paramétriques auront habituellement une plus grande *variance* (l'évaluation ponctuelle), moins de *puissance* (test d'hypothèse), un *intervalle de confiance* plus large (dans l'évaluation d'intervalle de confiance), et un plus gros *risque* (dans la théorie de décision) en comparaison avec l'approche paramétrique correspondante.

L'idée dominante dans le *bootstrap* s'appelle le principe de substitution ou principe *plug-in*. L'inférence fréquentiste implique le remplacement d'une distribution F inconnue par une estimation F^* (le ré échantillonnage d'un échantillon est une approximation de l'échantillonnage d'une population finie). La deuxième idée est de remplacer le calcul analytique des propriétés d'un estimateur $\hat{\theta}$ d'un paramètre inconnu $\theta = \theta(F)$ par un calcul empirique sur F^*. Disons qu'on a au plus un échantillon réel des données et aucune information supplémentaire au sujet de la population d'où il provient. La méthode de *bootstrap* consiste à construire un nombre B (B entier) d'*échantillons bootstrap* (images de l'échantillon initial), afin de les utiliser pour faire des inférences. Le *bootstrap* s'applique par ré échantillonnage de données avec remise en obtenant un nouvel ensemble de données à chaque fois à partir de l'échantillon initial (ré échantillonnage avec remise signifie qu'après avoir tiré aléatoirement un élément de l'échantillon original on le remet avant de retirer le prochain élément. Par conséquent, chaque élément peut être retiré plus qu'une fois, ou pas du tout). Les propriétés des *échantillons bootstrap* sont inférées en analysant chaque *échantillon bootstrap* exactement comme si on avait analysé le vrai échantillon de données, cela veut dire que, pour chaque nouvel échantillon, on calcule de la même façon un nouvel estimateur (image simulée de l'estimateur initial). L'ensemble des images simulées de l'estimateur initial est considéré comme un modèle de sa distribution sur la population de l'échantillon initial. Soit, $X = (X_1, ..., X_n)$ l'échantillon initial et $X^*(1), ..., X^*(B)$ les *échantillons bootstrap* simulés sur X. L'estimation à partir d'un échantillon aléatoire amène toujours vers une incertitude, puisque fondé sur un échantillon aléatoire, l'estimateur $\theta(X^*(1), ..., X^*(B))$ est aussi (une variable) aléatoire. La procédure de *bootstrap* consiste en un certain nombre de répétitions, et plus le nombre d'images simulées est grand, plus la statistique est précise, mais un nombre de simulations entre 50 et 200 est généralement suffisant (Efron et Tibshirani, 1993) pour des problèmes simples ; mais on croit qu'au moins 1000 duplications de *bootstrap* sont nécessaires, dans la plupart des applications. On montrera qu'un nombre B beaucoup plus grand pourrait être nécessaire dans les cas où la distribution d'échantillonnage est fortement asymétrique, si la taille de l'échantillon de départ est petite. D'autre part, bien qu'un B infini corresponde à l'intervalle *bootstrap* théoriquement exact, l'augmentation de B vers l'infini ne peut pas être toujours efficace par rapport au coût informatique. La méthode de *bootstrap* est exigeante au point de vue informatique, il fonc-

tionne bien quand la taille de l'échantillon est grande, mais peut être peu faible quand la taille de l'échantillon est petite (disons 5, 10 ou même 20), indépendamment du nombre (B) d'échantillons de *bootstrap* employés.

Le *bootstrap* est une technique générale pour estimer les quantités inconnues associées à des modèles statistiques :

1. les écarts type d'estimateurs ;
2. les intervalles de confiance pour des paramètres inconnus ;
3. les *p-values* pour des statistiques de test sous une hypothèse nulle.

Il faut se rappeler qu'un modèle statistique est essentiellement un ensemble de loi de probabilité qui essaie de décrire l'état réel de la nature à partir de données aléatoires disponibles.

Dans le cas général, nous voulons évaluer l'exactitude, la précision, la dispersion (donc le biais, l'écart type, etc.) d'un estimateur arbitraire $\widehat{\theta}$ en connaissant seulement un échantillon des données $X = (X_1, ..., X_n)$ tiré d'une population avec la fonction de répartition F inconnue. Le *bootstrap* fournit une méthodologie alternative pour la construction des intervalles de confiance en utilisant l'échantillon initial des données pour calculer la statistique du test et estimer sa distribution, sans avoir d'hypothèses au sujet de la distribution fondamentale de la population. On n'a pas besoin d'un calcul théorique de l'écart type, ainsi on ne se préoccupe pas de la complexité mathématique de l'estimateur $\widehat{\theta}$.

4.3.1 Échantillon aléatoire initial de données de taille n

L'échantillon aléatoire initial de données de taille n est noté par $X = (X_1, ..., X_n)$. On le considère comme un ensemble des variables aléatoires $X_1, ..., X_n$ indépendantes et identiquement distribuées (i.i.d.) où la fonction de densité et la fonction de répartition sontt notées par f et F respectivement. L'échantillon est utilisé pour l'inférence concernant une caractéristique de la population, généralement notée par θ, en utilisant une statistique $T(X_1, ..., X_n)$ dont la valeur pour l'échantillon est t. Si on suppose que t est un estimé de θ, l'attention est concentrée sur la

loi de t, afin de répondre à des questions liées, par exemple, à l'écart type, au biais ou aux quantiles de cette distribution. Les quantiles sont nécessaires pour la détermination des intervalles de confiance *bootstrap* pour θ (l'intervalle de confiance *bootstrap* est basé sur la probabilité de type $P(a_1 \leq S_n(X_1, \ldots, X_n) \leq a_2)$).

Dans le contexte du problème étudié, l'échantillon initial est caractérisé par sa taille N - le nombre total d'haplotypes au départ, et par la taille des deux sous-échantillons déterminés par le statut de la maladie, respectivement n pour le sous-échantillon de cas et m pour le sous-échantillon de témoins ($N = n + m$). De plus, une caractéristique importante est la multiplicité de différents haplotypes, soit d le nombre de différents types d'haplotypes de cas, chacun avec multiplicité n_i ($\sum_{i=1}^{d} n_i = n, i = 1, ..., d$) et respectivement l le nombre de différents types d'haplotypes de témoins, chacun avec multiplicité m_j ($\sum_{j=1}^{l} m_j = m, i = 1, ..., l$). Prenons comme exemple les données présentées au tableau 4.1 :

Haplotypes	Multiplicité	Code de statut	Paramètres d'échantillon
0 0 0 0	$n_1 = 5$	1 - cas	$n = n_1 + n_2 + n_3 + n_4 = 8$
0 0 0 1	$n_2 = 1$	1 - cas	$m = m_1 + m_2 = 2$
0 0 1 0	$n_3 = 1$	1 - cas	$d = 4$
1 0 0 0	$n_4 = 1$	1 - cas	$l = 2$
0 0 0 1	$m_1 = 1$	0 - contrôle	$N = n + m = 10$
1 1 0 1	$m_2 = 1$	0 - contrôle	

Tableau 4.1: L'échantillon initial de données de taille $N = 10$.

4.3.2 Fonction de répartition empirique (EDF), principe du *plug-in*

Soit $X = (X_1, ..., X_n)$ un échantillon aléatoire de taille n tiré d'une distribution F, et soit $\hat{F}_n$ la fonction de répartition empirique qui se définit comme la loi discrète qui donne la

probabilité $\frac{1}{n}$ à chaque valeur X_i, $i = 1, ..., n$ et

$$\hat{F}_n(x) = \frac{\#X_i \leq x}{n} = \frac{1}{n}\sum_{i=1}^{n} I\{X_i \leq x\}, \tag{4.1}$$

$$I\{X_i \leq x\} = \begin{cases} 1, & \text{si } X_i \leq x, \\ 0, & \text{sinon} \end{cases}. \tag{4.2}$$

Le concept du *plug-in*, discuté par Efron et Tibshirani (1993), est une méthode simple pour estimer des paramètres d'échantillons et un outil maniable pour l'application de la méthode. C'est-à-dire, si une certaine caractéristique d'une distribution F doit être déterminée à partir d'un échantillon aléatoire tiré de F et si, par exemple, la fonction de distribution empirique $\hat{F} = F_n$ est utilisée pour estimer F, n'importe quelle caractéristique de F telle que sa moyenne ou sa médiane est estimée en employant la caractéristique correspondante de F_n. Par exemple, l'estimateur *plug-in* d'un paramètre $\theta = s(F)$ est défini comme $\hat{\theta} = s(\hat{F})$. Alors, le *bootstrap* peut être utilisé d'une manière automatique dans les études de l'écart type et du biais.

4.3.3 L'échantillon *bootstrap*

Un échantillon *bootstrap* est défini comme un échantillon aléatoire d'observations indépendantes et identiquement distribuées (i.i.d.) de taille m tiré avec remise de la fonction de distribution empirique $\widehat{F} = F_n$, d'une population de n objets $X = (X_1, \ldots, X_n)$, composé d'éléments de l'échantillon initial $X_1, \ldots, X_n$. On dénote l'échantillon *bootstrap* par $X^* = (X_1^*, ..., X_m^*)$. L'astérisque dans la notation indique que $X^* = (X_1^*, ..., X_m^*)$ n'est pas l'échantillon initial $X = (X_1, \ldots, X_n)$. Notons que dans le *bootstrap* classique $m = n$. Cependant, Swanepoel (1986) a défini le procédé de *bootstrap modifié* ($m \neq n$) et recommande cette méthode dans les cas où le *bootstrap classique* échoue.

Soit θ le paramètre d'intérêt et $\hat{\theta} = s(X)$ l'estimateur de θ basé sur l'échantillon $X = (X_1, \ldots, X_n)$. Un échantillon *bootstrap* $X^* = (X_1^*, X_2^*, \ldots, X_n^*)$ est obtenu par ré échantillonnage avec remise n-fois des données initiales $X = (X_1, X_2, \ldots, X_n)$. Par exemple, considé-

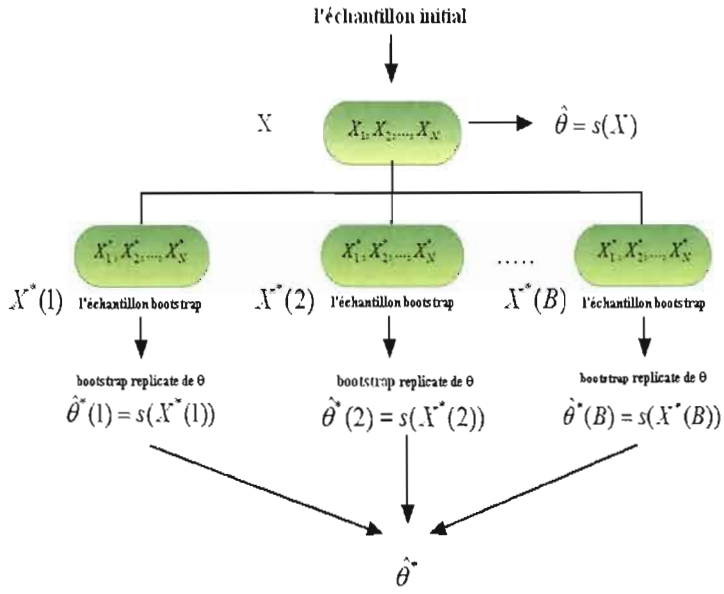


Figure 4.2: Les étapes à suivre pour construire des échantillons *bootstrap* $X^* = (X_1^*, ..., X_m^*)$ et les estimateurs *bootstrap* $\hat{\theta}^*(b), b = 1, 2, 3, ..., B$ de $\hat{\theta}$.

rons l'échantillon $X = (X_1, X_2, X_3, X_4, X_5)$, les échantillons *bootstrap* peuvent être :

$$X^*(1) = (X_2, X_4, X_1, X_5, X_1),$$
$$X^*(2) = (X_3, X_2, X_4, X_4, X_1),$$
$$X^*(3) = (X_4, X_3, X_4, X_4, X_2),$$
$$...$$

Avec chaque échantillon *bootstrap* $X^*(1), X^*(2), X^*(3), ..., X^*(B)$, nous pouvons calculer une réplication *bootstrap* de $\hat{\theta}$, $\hat{\theta}^*(b) = s(X^*(b))$ en utilisant le principe *plug-in*. Les problèmes de l'inférence statistique impliquent souvent l'estimation d'un certain aspect d'une distribution F basée sur un échantillon aléatoire tiré de F. La fonction de distribution empirique $\hat{F}$ est une évaluation simple de la distribution F. La méthode de *bootstrap* est une application directe du principe *plug-in*.

En pratique, on construit un ensemble de B échantillons *bootstrap* en répétant B fois la

procédure suivante :

1. Un générateur de nombres aléatoires choisit les nombres entiers $i_1, i_2, i_3, ..., i_n, i_j \in [1; n]$, $j = 1, 2, ..., n$, avec probabilité $\frac{1}{n}$;
2. On calcule alors l'échantillon $X^* = (X_{i_1}, X_{i_2}, X_{i_3}, ..., X_{i_n}) = (X_1^*, X_2^*, ..., X_n^*)$.

Dans ce cas, pour un échantillon $X = (X_1, ..., X_n)$ en supposant que chacun des X_i est différent, la probabilité qu'une valeur particulière X_i de l'échantillon initial n'apparaisse pas dans un échantillon *bootstrap* $X^* = (X_1^*, X_2^*, ..., X_n^*)$ est :

$$P(X_j^* \neq X_i; 1 \leq j \leq n) = \left(1 - \frac{1}{n}\right)^n \tag{4.3}$$

car

$$P(X_j^* = X_i) = \frac{1}{n} \tag{4.4}$$

Quand n est grand, la probabilité $P(X_j^* \neq X_i; 1 \leq j \leq n) = \left(1 - \frac{1}{n}\right)^n$ converge vers $e^{-1} \approx 0.37$.

Une fois l'estimateur fourni par le programme *MapArg*, certaines évaluations doivent être faites pour évaluer la fidélité ou la robustesse des résultats. L'estimateur obtenu par la méthode de Larribe et al (2002) est une évaluation ponctuelle de la position de la mutation, et ceci soulève le dilemme : est-ce que le résultat obtenu sur l'échantillon initial est également probable d'être vrai par rapport à d'autres échantillons semblables d'haplotypes ? Les échantillons semblables d'haplotypes peuvent être construits par la méthode du *bootstrap* de manières différentes et l'intervalle de confiance dépend de la manière choisie. Par exemple, construire un *échantillon bootstrap* en tenant compte de l'unicité de différents haplotypes ou construire un *échantillon bootstrap* compte tenu de l'effet des marqueurs. Si on choisit l'unicité des haplotypes, à chaque etape les séquences entières sont tirées avec remise pour construire un nouvel échantillon (*échantillon bootstrap*) de même taille, avec lequel la position probable de la mutation est estimée. Si on choisit l'effet des marqueurs, les états de chaque marqueur sont rassemblés et avec le

bootstrap sont construits de nouveaux haplotypes en conservant les paramètres N, n et m de l'échantillon initial.

Exemple 4.1 Reprenons les données du tableau 4.1. Les paramètres $N = 10$, $n = 8$ *et* $m = 2$ *sont pris en compte. De l'ensemble des haplotypes cas sont tirées avec remise* 8 *séquences entières, et de la même façon* 2 *séquences entières sont tirées de l'ensemble des haplotypes témoins pour compléter l'échantillon bootstrap.*

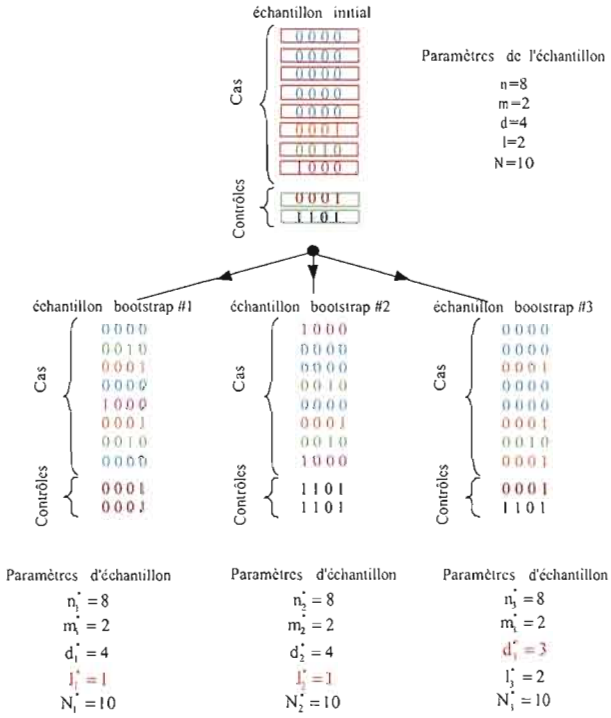


Figure 4.3: Les étapes à suivre pour construire les *échantillons bootstrap* sur l'ensemble de données *peter.dat*. Le *bootstrap* est réalisé sur les haplotypes entiers. Les paramètres N, n et m sont inchangés, mais on voit que d et l peuvent varier. Dans chaque *échantillon bootstrap* le nombre de types différents de séquences présents ne dépasse pas les nombres initiaux d pour les cas et l pour les contrôles, et il n'y a pas de nouveaux types de séquences.

On a vu la procédure bootstrap basée sur les différents types de séquences. Une autre approche est de reconstruire les séquences par bootstrap de marqueurs. Dans ce cas, par ordre

des marqueurs les états de chaque marqueur sont rassemblés indépendamment par rapport au phénotype exprimé. Un haplotype cas est construit en tirant dans l'ordre approprié un marqueur à la fois avec remise dans chaque groupe de marqueurs représentant le phénotype attribué.

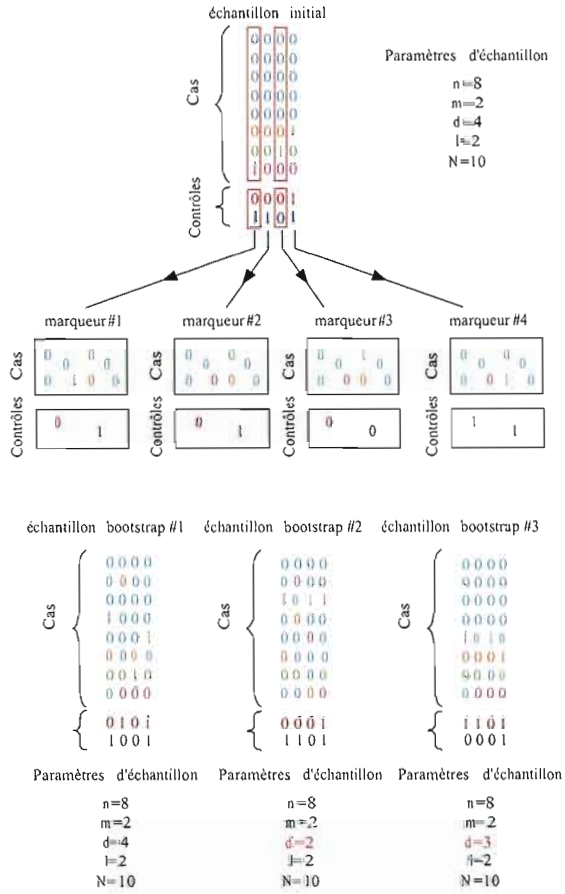


Figure 4.4: Les étapes à suivre pour construire les *échantillons bootstrap* par marqueur pour l'ensemble des données *peter.dat*. Les paramètres N, n et m demeurent inchangés, d et l peuvent varier. Dans chaque échantillon bootstrap le nombre de différents types de séquences participant peut dépasser les nombres initiaux d pour les cas et l pour les contrôles, et cette fois on observe de nouveaux types de séquences.

Dans l'extension du programme *MapArg*, le *bootstrap* réalisé est basé sur les séquences entières. En utilisant le *bootstrap* de marqueurs, on perd une partie de l'information portée par la liaison entre les différents marqueurs. Les études faites sur des petits ensembles de marqueurs

génétiques où l'information est concentrée donnent souvent de bons résultats, ce qui détermine le choix d'un *ré échantillonnage bootstrap* par séquences entières.

4.3.4 Moyenne *bootstrap*, écart type *bootstrap*, quantile d'ordre q

On définira maintenant la moyenne bootstrap. Pour un ensemble d'estimateurs $\widehat{\theta}^{*b}, b = 1, 2, 3, ..., B$, la moyenne est :

$$\widehat{\theta}^*(.) = \frac{\sum_{b=1}^{B} \widehat{\theta}^{*b}}{B}. \tag{4.5}$$

L'écart type est aussi une caractéristique importante de chaque distribution. Pour un ensemble d'estimateurs $\widehat{\theta}^{*b}, b = 1, 2, 3, ..., B$, l'écart type estimé est calculé par la formule :

$$\widehat{se}_B = \sqrt{\frac{\sum_{b=1}^{B} \left(\widehat{\theta}^{*b} - \widehat{\theta}^*(.)\right)^2}{B-1}}, \tag{4.6}$$

et B est le nombre total d'échantillons *bootstrap* (chacun de taille N).

Les deux définitions sont une application directe du principe *plug-in*, la moyenne et l'écart type d'un échantillon sont approchés par la moyenne et l'écart type de *l'échantillon bootstrap*.

Soit X une variable aléatoire. Pour $q \in [0, 1]$, on dit que x_q est un *quantile* d'ordre q de X si $P(X \leq x_q) \geq q$ et $P(X \geq x_q) \leq 1 - q$. Pour $q = \frac{1}{2}$, on parle de la *médiane*. Pour $q = \frac{1}{4}$ et $q = \frac{3}{4}$on parle de *premier* et *troisième quartiles*.

4.3.5 Intervalle de confiance *bootstrap* pour un paramètre θ arbitraire.

Soit, $X_1, X_2, \ldots, X_n$ l'échantillon aléatoire prélevé d'une distribution F dont le paramètre θ est inconnu. Supposons qu'on peut trouver deux statistiques, disons $A(X_1, X_2, ..., X_n)$ et $B(X_1, X_2..., X_n)$, telles que $\Pr[A(X_1, ..., X_n) < \theta < B(X_1, ..., X_n)] = \gamma$, ou γ est une probabilité fixée ($0 < \gamma < 1$). Les valeurs spécifiques des statistiques $A(X_1, ..., X_n) = a$ et

$B(X_1, ..., X_n) = b$ qui assurent la probabilité fixée γ forment un intervalle de confiance pour le parametre θ.

Il faut souligner que les valeurs des statistiques $A(X_1, \ldots, X_n)$ et $B(X_1, \ldots, X_n)$ avant d'être observées sont des variables aléatoires, c'est-à-dire le paramètre θ appartiendra à l'intervalle aléatoire avec les points terminaux $A(X_1, \ldots, X_n)$ et $B(X_1, \ldots, X_n)$ avec probabilité γ. Une fois les valeurs spécifiques de $A(X_1, \ldots, X_n) = a$ et $B(X_1, \ldots, X_n) = b$ observées, il devient impossible d'attribuer une probabilité à l'événement que θ appartienne à un intervalle spécifique $(a; b)$ sans considérer θ comme une variable aléatoire avec une distribution appropriée.

4.3.6 Distribution de θ.

Quand suffisamment d'échantillons *bootstrap* sont (ont été) produits, non seulement l'écart type, mais n'importe quel aspect de la distribution de l'estimateur $\hat{\theta} = t(F)$ pourrait être estimé. Les distributions *bootstrap*s imitent la forme et la variance de la vraie distribution de l'estimateur sur la population.

La méthode de Larribe et al (2002) par le programme *MapArg* fournit un estimateur $\hat{r}_T$ de la position probable de la mutation pour un ensemble d'haplotypes de cas et de contrôles tel que la fonction de vraisemblance $L(r_T)$ atteint son maximum à $\hat{r}_T$ ($\hat{r}_T : \hat{L}(\hat{r}_T) = \max_{r_T} \hat{L}(r_T)$). C'est-à-dire, pour chaque lancement du programme on a une paire $(\hat{r}_T; \hat{L}(\hat{r}_T))$ de résultats. Ensuite, pour chaque échantillon *bootstrap*, le programme *MapArg* fournit une paire $(\hat{r}_T^b; \hat{L}^b(\hat{r}_T^b))$, $b = 1, 2, ..., B$ de résultats *bootstrap*. Regardons maintenant les distributions de $\hat{r}_T^b$ et $\hat{L}^b(\hat{r}_T^b)$, un exemple graphique est montré sur les figures 4.5 et 4.6.

Example 4.1 Pour illustrer les definitions précédentes, prenons les données A1 (Appendice A). Les valeurs de $(\hat{r}_T)^b$ *et* $\hat{L}^b((\hat{r}_T)^b)$, $b = 1, 2, ..., B$ *sont analysées pour un nombre* $B = 500$. *de bootstrap*

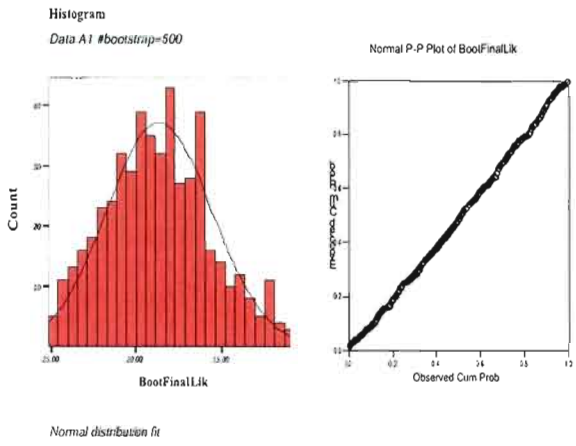


Figure 4.5: Les données du fichier *A1.dat*, avec les paramètres $N = 100$, $n = 50$, $m = 50$, $d = 14$, $l = 26$ sont ré échantillonnées. Un nombre $B = 100$ d'échantillons *bootstrap* sont produits. Par chaque échantillonne *bootstrap* l'estimateur $\hat{r}_T^b : \hat{L}^b(\hat{r}_T^b) = \max_{r_t} \hat{L}^b(r_t)$, $b = 1, 2, 3, ..., B$ est calculé et les paires $((\hat{r}_T)^b ; \hat{L}^b((\hat{r}_T)^b))$ sont conservées. Une analyse sur les valeurs de $\hat{L}^b((\hat{r}_T)^b)$ est effectuée. Le premier graphique représente l'histogramme de $\hat{L}^b((\hat{r}_T)^b)$ et la comparaison avec la distribution normale. Comme l'histogramme favorise l'hypothèse de normalité, une deuxième évaluation est faite par «$p - pplot$». Le deuxième graphique représente la distribution cumulative observée de $\hat{L}^b((\hat{r}_T)^b)$ vers la distribution cumulative théorétique.

4.3.7 Choix du B, le nombre de *bootstraps*.

La construction d'un intervalle de confiance *bootstrap* nécessite de calculer, de façon répétitive, B fois la même statistique du test pour obtenir les estimateurs *bootstrap*. Comment déterminer le nombre de répétitions nécessaires, pour que le rapport (confiance dans les résultats)/(temps à calculer) soit acceptable ? Pour répondre à cette question, rappelons-nous les deux définitions de la convergence :

Convergence en distribution : Une suite $\{F_n\}$ de fonctions de la distribution cumulative converge en distribution vers F si $F_n(x) \rightarrow F(x)$ en tout points communs de F c'est-à-dire

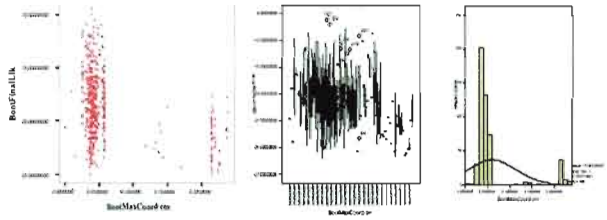


Figure 4.6: Le premier graphique est un graphique pour les paires $((\hat{r}_T)_b^*; \hat{L}^b((\hat{r}_T)_b^*))$, le troisième graphique représente l'histogramme de $(\hat{r}_T)_b^*$.

si $\{X_n\}$ sont des variables aléatoires avec fonctions de répartition F_n, et X est une variable aléatoire avec fonction de répartition F.

$$X_n \xrightarrow{Loi} X$$

si $\forall X : -\infty < X < \infty, F_n(X) \to F(X)$ quand $n \to \infty$.

Convergence en probabilité : X_n converge en probabilité vers X si

$P(|X_n - X| < \varepsilon) \to 1, \quad \forall \varepsilon > 0$, quand $n \to \infty$.

En utilisant le théorème limite central, on obtient :

$$\sqrt{n}(\hat{F}_n(a) - F(a)) \xrightarrow{L} N(0, F(a)(1 - F(a)) \tag{4.7}$$

d'où

$$\hat{F}_n(a) \xrightarrow{P} F(a) \tag{4.8}$$

Maintenant, en utilisant le résultat final, nous avons que $se_{\hat{F}}(\theta^*)$ est un estimateur consistant de $se_F(\theta^*)$. Selon la loi des grands nombres, l'estimateur *bootstrap* idéal pour $se_{\hat{F}}(\theta^*)$ est défini comme :

$$\lim_{B\to\infty} \widehat{se}_B = se_{\hat{F}}(\theta^*), \tag{4.9}$$

et il s'appelle l'estimateur *bootstrap* non paramétrique de l'écart type. Ainsi, le choix de B peut être fait par une procédure itérative jusqu'a ce que le nombre de *bootstraps* ne se reflète pas dans l'estimateur *bootstrap* $\widehat{se}_B$. (En pratique on s'interesse plutôt à savoir comment la moyenne des échantillons *bootstrap* se situe par rapport à la moyenne de la population.)

4.3.8 L'estimateur *bootstrap* du biais.

Pour construire un intervalle de confiance, on a besoin de l'écart type et aussi du biais. L'estimateur *bootstrap* du biais est défini comme suit :

$$Biais_{\hat{F}}(\hat{\theta}) = \mathrm{E}_{\hat{F}}\left[s(x^*)\right] - t(\hat{F}) = \hat{\theta}^*(.) - \hat{\theta}, \tag{4.10}$$

où

$$\hat{\theta}^*(.) = \frac{\sum_{b=1}^{B} \hat{\theta}^*(b)}{B}.$$

L'estimateur *bootstrap* du biais est la différence entre la moyenne de la distribution *bootstrap* et la statistique pour les données originales. Un petit biais signifie que la distribution *bootstrap* est centrée sur la statistique de l'échantillon original et suggère que la distribution de la statistique d'échantillonnage est centrée sur le paramètre de la population.

4.4 Intervalle de confiance *bootstrap*

Nous sommes intéressés à trouver l'intervalle de confiance *bootstrap* pour un paramètre θ arbitraire où $\hat{\theta}$ et l'estimateur de θ basé sur l'échantillon initial. La figure 4.7 ci-dessous représente les familles d'intervalles de confiance *bootstrap*. Pour faciliter la compréhension de la terminologie utilisée, une partie des noms de différentes familles et sous-familles est laissée non traduite de la langue anglaise.

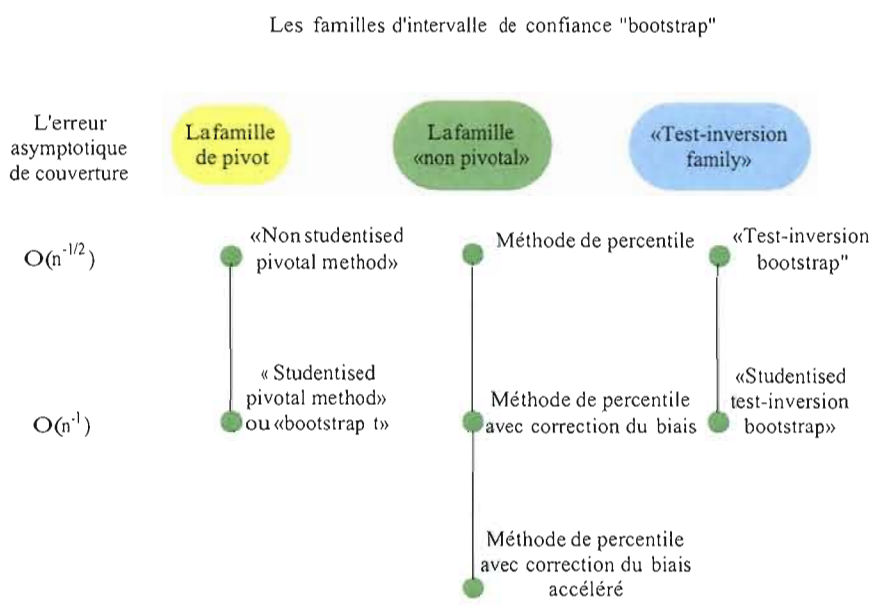


Figure 4.7: Les intervalles de confiance *bootstrap* sont divisés en trois familles, par rapport à la méthode utilisée pour les construire, et chaque famille est subdivisée par rapport à la précision de l'intervalle fourni.

Chaque famille fournit des intervalles de confiance avec différentes propriétés. On étudiera quatre types d'intervalle de confiance *bootstrap* :

1. intervalle de confiance *bootstrap* empirique (intervalle de confiance *bootstrap* bilatéral basé sur les percentiles) ;
2. intervalle de confiance *bootstrap* accéléré avec correction du biais ;
3. intervalle de confiance *bootstrap* bilatéral symétrique de base ;
4. intervalle de confiance *bootstrap* bilatéral symétrique de Student (*bootstrap t*) ;

En comparant le comportement et les propriétés de ces intervalles on décidera lequel sera plus approprié dans nos études.

4.4.1 L'intervalle de confiance empirique.

L'intervalle $\left\{\hat{\theta}^*_{(\alpha)}; \hat{\theta}^*_{(1-\alpha)}\right\}$ entre les α-ème et $1-\alpha$-ème percentile de la distribution *bootstrap* d'une statistique est un intervalle de confiance au niveau $1-\alpha$ pour le paramètre correspondant. Si le biais est petit et la distribution est à peu près normale, l'intervalle de confiance basé sur les percentiles et l'intervalle de confiance t sont semblables. Si ce n'est pas le cas, il est évident que la normalité et les conditions quant au biais ne sont pas satisfaites. Si $\hat{\theta}_{lo} = \hat{\theta}^*_{(\alpha)}$ est la borne inferieure et $\hat{\theta}_{up} = \hat{\theta}^*_{(1-\alpha)}$ est la borne supérieure d'un tel intervalle de confiance *bootstrap*, on peut obtenir la probabilité que la vraie valeur du paramètre soit hors de l'intervalle trouvé comme suit :

$$\Pr(\widehat{\theta}_{lo} > \theta) = \frac{\alpha}{2} + O(n^{-1/2}), \tag{4.11}$$

et

$$\Pr(\widehat{\theta}_{up} < \theta) = \frac{\alpha}{2} + O(n^{-1/2}), \tag{4.12}$$

ce qui nous donne un intervalle de confiance précis de premier ordre.

Les avantages de l'intervalle de confiance empirique (l'intervalle *bootstrap* de confiance bilatéral basé sur les percentiles) sont :

- c'est un intervalle calculé *automatiquement*, intuitif et utilisable avec chaque transformation vers la normalité ;
- on peut l'employer même si la distribution de $\theta^*(b)$ est asymétrique ;
- il conserve le domaine de θ (une fois $\hat{\theta}$ fourni) ;
- les transformations sont respectées (intervalle de confiance cohérent par exemple pour θ ou $\log(\theta)$).

L'intervalle de confiance empirique n'est pas cependant l'intervalle de confiance idéal. Il

faut souligner quelques désavantages de ce type d'intervalle de confiance :

- une précision seulement de premier ordre ;
- la construction d'un tel intervalle de confiance exige un grand nombre B d'échantillons *bootstrap*s (1000, 2000 et plus) ;
- il ignore le biais de $\hat{\theta}$;
- il est inutile quand $se(\widehat{\theta}) = se_\theta(\widehat{\theta})$, c'est-à-dire quand l'écart type de $\hat{\theta}$ dépend de θ.

La correction du biais pour ce type d'intervalle de confiance est effectuée de la manière suivante :

1. On détermine la proportion p de valeurs $\hat{\theta}_b^*$ inférieure à $\hat{\theta}$: $p = \frac{\#\{\widehat{\theta}_b^* < \widehat{\theta}\}}{B}$ (p coefficient d'asymétrie).
2. On calcule le percentile $z_0 = \Phi^{-1}(p)$ relatif à la distribution normale réduite.
3. Soit $z_1 = 2z_0 + z_{\frac{\alpha}{2}}$ et $z_1 = 2z_0 + z_{1-\frac{\alpha}{2}}$. On calcule $\alpha_1 = \Phi(z_1)$ et $\alpha_2 = \Phi(z_2)$ les valeurs de la fonction de répartition de la distribution normale réduite aux points z_1 et z_2.
4. Les limites de confiance déterminées par la méthode des percentiles corrigés pour le biais (*bias corrected percentile confidence interval*) sont alors les percentiles $\widehat{\theta}^*_{(\alpha_1)}$ et $\widehat{\theta}^*_{(\alpha_2)}$ de la distribution des $\widehat{\theta}_b^*$ (Efron et Tibshirani (1993), Chernick (1999)). C'est-à-dire ($\widehat{\theta}^*_{(\alpha_1)}$; $\widehat{\theta}^*_{(\alpha_2)}$) est un intervalle de confiance empirique pour θ corrigé pour le biais.

On remarque que si $p = 0.5$, c'est-à-dire si $\hat{\theta}$ est la médiane de la distribution des $\hat{\theta}_b^*$, il n'y a pas de correction pour le biais, puisque $z_0 = 0$, et on retrouve la méthode précédente. Les limites de confiance seront toutes légèrement plus grandes que dans le cas de la méthode des percentiles simples.

Examinons maintenant, le problème de construction d'un intervalle de confiance *bootstrap* empirique pour l'estimateur $\hat{r}_T$ de la position probable r_T de la mutation obtenue par la méthode *MapArg*. La méthode fournit un estimateur $\hat{r}_T$: $\hat{L}(\hat{r}_T) = \max_{r_T} \hat{L}(r_T)$. Comme on le voit, $\hat{r}_T$ et $\hat{L}(\hat{r}_T)$ sont mutuellement dépendants : la valeur de $\hat{L}(\hat{r}_T)$ est calculée conditionnellement à $\hat{r}_T$ et le choix de $\hat{r}_T$ est déterminé par la valeur de $\hat{L}(\hat{r}_T)$, c'est-à-dire le compor-

tement de la paire $(\hat{r}_T\,;\,\hat{L}(\hat{r}_T))$ sur l'ensemble d'échantillons *bootstrap* nous sera d'un intérêt particulier. Premièrement, l'ensemble d'haplotypes initial est ré échantillonné B fois. Chaque échantillon *bootstrap* a la même taille N tout comme l'échantillon initial, et les tailles, n du sous-échantillon *cas* et m du sous-échantillon *témoins* sont préservées. Pour chaque échantillon *bootstrap* par la méthode *MapArg* est calculé un estimateur *bootstrap* $\hat{r}_T^b : \hat{L}^b(\hat{r}_T^b) = \max_{r_T} \hat{L}^b(r_T)$, $b = 1, 2, 3, ..., B$ de la position probable de la mutation. Les séries d'expériences sont effectuées sur différents ensembles des données avec fenêtres de longueurs diverses en augmentant le nombre B de *bootstrap*s. La distribution de $\hat{L}^b(\hat{r}_T^b)$, $b = 1, 2, ..., B$ est examinée pour chaque série indépendante. Le nombre B de *bootstrap* à effectuer est déterminé par le critère défini auparavant à la sou-section 4.3.7.

4.4.2 Intervalle de confiance *bootstrap* accéléré avec correction de biais.

La méthode précédente, qui prend en compte le biais, peut être généralisée de manière à tenir compte d'un éventuel changement de l'erreur standard de $\hat{\theta}$ lorsque θ varie. Elle porte alors le nom de méthode des percentiles avec correction pour le biais et accélération (*bias corrected and accelerated confidence interval*). Une justification de cette méthode est donnée par Efron et Tibshirani (1993). Les limites de confiance sont les percentiles $\widehat{\theta}^*_{(\alpha_1)}$ et $\widehat{\theta}^*_{(\alpha_2)}$ de la distribution des $\hat{\theta}^*_b$, où α_1 et α_2 sont les valeurs de la fonction de répartition de la variable normale réduite aux points z_1 et z_2 définis de la manière suivante :

$$z_1 = z_0 + \frac{z_0 + z_{\frac{\alpha}{2}}}{1 - a(z_0 + z_{\frac{\alpha}{2}})}, \tag{4.13}$$

et

$$z_2 = z_0 + \frac{z_0 + z_{1-\frac{\alpha}{2}}}{1 - a(z_0 + z_{1-\frac{\alpha}{2}})}. \tag{4.14}$$

Dans ces relations, z_0 est défini comme précédemment, et la constante a est appelée *accélération*, car elle est liée au taux de variation de l'erreur-standard de $\hat{\theta}$ lorsque le paramètre varie. Cette constante peut être estimée de différentes manières. Une solution consiste à utiliser

la technique du *jackknife*. On obtient alors le paramètre a par la relation suivante :

$$a = \frac{\sum_{i=1}^{B} (\widehat{\theta}_i^* - \theta(.))^3}{6(\sum_{i=1}^{B} (\widehat{\theta}_i^* - \theta(.))^2)^{\frac{3}{2}}}. \tag{4.15}$$

Ce modèle prend en compte trois problèmes inhérents de beaucoup d'estimateurs (échantillon fini) :

1. une transformation inconnue vers la normalité ;
2. le biais ;
3. l'accélération ;

La procédure pour construire un tel intervalle de confiance est la suivante :

1. on génère l'échantillon *bootstrap* $X^*(b)$, pour $b = 1, 2, \ldots, B$, ou B est grand (au moins 1000 ou 2000, mais souvent 10000) ;
2. on estime z_0 et a ;
3. on calcule z_1, z_2 et $\alpha_1 = \Phi(z_1)$ et $\alpha_2 = \Phi(z_2)$;
4. on calcule $\widehat{\theta}^*_{(\alpha_1)}$ et $\widehat{\theta}^*_{(\alpha_2)}$ respectivement $100\alpha_1$- ème et $100\alpha_2$-ème percentile de $\hat{\theta}^*(b)$.

Pour ce qui est des avantages de cet intervalle on mentionne :

1. une vraie précision de deuxième ordre quand $\hat{\theta}$ est de loi asymptotique normale ;
2. correction de biais ;
3. il respecte les transformations (approximativement) car l'estimation de a dépend faiblement de la transformation choisie pour θ ;
4. il conserve le domaine ;
5. il a une bonne performance lorsque $se(\widehat{\theta}) = se_\theta(\widehat{\theta})$, c'est-à-dire lorsque l'écart type de $\hat{\theta}$ dépend de θ.

6. On peut toujours utiliser l'intervalle de confiance *bootstrap* accéléré avec correction de biais.

Les désavantages de ce type d'intervalle de confiance sont :

1. Nécessité d'un grand nombre B de *bootstrap*s ;
2. la complexité des calculs (en particulier de a).

4.4.3 L'intervalle *bootstrap* de confiance bilatéral symétrique de base

Supposons que $\hat{\theta}$ est l'estimateur de maximum de vraisemblance (l'EMV de θ- la valeur de $\hat{\theta}$ pour laquelle la fonction de vraisemblance évaluée dans $X = (X_1, ..., X_n)$ atteint son maximum). Selon la théorie asymptotique, la statistique $\hat{\theta}$ - θ est approximativement normalement distribuée quand la taille de l'échantillon est suffisamment grande, pour l'intervalle de confiance on peut utiliser l'approximation suivante :

$$\sigma_n^2(\theta) = \frac{1}{I_n(\theta)}, \tag{4.16}$$

où

$$I(\theta) = \mathrm{E}_\theta\{[\lambda'(X|\theta)]^2\}, \tag{4.17}$$

$$\lambda(x|\theta) = \log f(x|\theta), \tag{4.18}$$

$$\lambda'(x|\theta) = \frac{\partial}{\partial\theta}\lambda(x|\theta). \tag{4.19}$$

S'il est difficile de calculer $\sigma_n^2(\theta) = \frac{1}{I_n(\theta)}$ où que l'on ne connaît pas la distribution, on pourrait utiliser l'estimateur *bootstrap* de la variance asymptotique. Même lorsque $\hat{\theta}$ n'est

pas l'EMV, la normalité asymptotique est souvent vérifiée, mais la variance est habituellement inconnue. En utilisant l'estimateur *bootstrap* de l'écart type, l'intervalle de confiance à $100(1-2\alpha)\%$ est :

$$\theta \in \left\{\hat{\theta} - z_{(1-\alpha)} * \widehat{se}_B; \hat{\theta} + z_{(1-\alpha)} * \widehat{se}_B\right\}, \tag{4.20}$$

et si le biais est différent de zéro l'intervalle de confiance avec correction de biais est :

$$\theta \in \left\{\hat{\theta} - \widehat{Biais}_B - z_{(1-\alpha)} * \widehat{se}_B; \hat{\theta} - \widehat{Biais}_B + z_{(1-\alpha)} * \widehat{se}_B\right\}, \tag{4.21}$$

où $z_{(1-\alpha)}$ est tel que

$$\Phi(z_{(1-\alpha)}) = 1 - \alpha.$$

Si $\widehat{\theta} \sim N(\theta + biais, \widehat{se}^2)$, $\widehat{\theta} \xrightarrow{p} \theta$, dans ce cas pour $\hat{\theta}_{lo}$ la borne gauche et $\hat{\theta}_{up}$ la borne droite de l'intervalle de confiance on a :

$$\Pr(\widehat{\theta}_{lo} > \theta) = \frac{\alpha}{2} + O(n^{-1}), \tag{4.22}$$

et

$$\Pr(\widehat{\theta}_{up} < \theta) = \frac{\alpha}{2} + O(n^{-1}). \tag{4.23}$$

Ce qui nous donne un intervalle de confiance précis de deuxième ordre. Si $\hat{\theta}$ ne suit pas une loi normale, mais converge en distribution vers une loi normale, on a $\Pr(\widehat{\theta}_{lo} > \theta) = \alpha + O(n^{-1/2})$ et $\Pr(\widehat{\theta}_{up} < \theta) = \alpha + O(n^{-1/2})$, et dans ce cas la précision de l'intervalle de confiance est de premier ordre.

Les avantages de l'intervalle *bootstrap* de confiance bilatéral symétrique de base sont :

- facile à calculer et interpréter ;
- le biais peut être corrigé ;
- pour $\hat{\theta}$ normalement distribué, il montre une précision de deuxième ordre ;
- le nombre de *bootstrap*s nécessaires est petit (B=200 est habituellement suffisant).

Par contre, l'intervalle *bootstrap* de confiance bilatérale symétrique de base montre quelques désavantages :

- en général, la précision est de premier ordre ;
- il ne conserve pas nécessairement le domaine ;
- il ne respecte pas la transformation ;
- il performe bien que dans les cas où $se(\widehat{\theta}) = se_{\theta}(\widehat{\theta})$, c'est-à-dire quand l'écart type de $\hat{\theta}$ dépend de θ.

Quand une distribution *bootstrap* est approximativement normale et l'écart-type est petit, nous pouvons utiliser la recette avec l'écart type *bootstrap* pour obtenir un intervalle de confiance pour n'importe quel paramètre. On ne doit pas employer cet intervalle si la distribution *bootstrap* n'est pas normale ou ne montre pas un biais substantiel. Car l'exactitude de cet intervalle dépend de la normalité asymptotique de $\hat{\theta} - \theta$, et cette exigence est difficilement vérifiable avec peu d'observations. Par conséquent, on veut construire un intervalle de confiance qui ne dépend pas de cette exigence.

4.4.4 L'intervalle *bootstrap* de confiance bilatérale symétrique de Student

Un pivot standard pour construire l'intervalle de confiance est : $t = (\widehat{\theta} - \theta)/\widehat{se(\widehat{\theta})}$. Sous les hypothèses de normalité de la population, $t = (\widehat{\theta} - \theta)/\widehat{se(\widehat{\theta})}$ suit une distribution t et dans ce cas il est facile de trouver l'intervalle de confiance :

$$\theta \in \left\{ \hat{\theta} - t^*_{(1-\alpha)} * \widehat{se}_B ; \hat{\theta} + t^*_{(\alpha)} * \widehat{se}_B \right\}, \tag{4.24}$$

où $t^* = (\hat{\theta}^* - \hat{\theta})\big/\widehat{se}_B$ et $t^*_{(\alpha)}$ dénote α-quantile de la distribution de t^*.

Pour calculer les intervalles traditionnels, nous avons dû utiliser les tables de la distribution de Student. Gosset à l'origine a construit cette distribution avec l'exigence qu'on ré échantillonne d'une population normale. L'idée mise en application dans le *bootstrap t* est de définir une statistique dont la distribution ne soit pas une fonction de la valeur réelle et inconnue du paramètre θ. La statistique T : $T = \frac{\widehat{\theta}-\theta}{\sigma_\theta}$ peut remplir ce rôle. Il s'agit alors d'approcher la distribution théorique de T par ré échantillonnage. Dans ce but, on calcule : $t^*_b = \frac{\widehat{\theta}^*_b - \widehat{\theta}}{\widehat{\sigma}(\widehat{\theta}^*_b)}$, $\widehat{\sigma}(\widehat{\theta}^*_b)$ étant l'écart type de $\widehat{\theta}^*_b$, qui dépend donc de l'échantillon b. Elle peut être calculée par une formule théorique, lorsqu'une telle formule est disponible, ou à partir du ré échantillonnage de l'échantillon $x^*(b)$ utilisé pour calculer $\widehat{\theta}^*_b$. Il s'agit alors d'un *bootstrap* à deux niveaux, puisque chaque échantillon $x^*(b)$ fait lui-même l'objet d'un ré échantillonnage, permettant de calculer l'écart-type.

Algorithme de *bootstrap t* :

1. Échantillon initial : $X = (X_1, X_2, \ldots, X_n) \rightarrow \theta$
2. Échantillon *bootstrap* (1) :$X^*(1) = (X^*_1, X^*_2, \ldots, X^*_n) \rightarrow \theta^*(1)$
   - 2.1 Échantillon *bootstrap* $1.1 : X^{**}(1.1) = (X^{**}_1, X^{**}_2, ..., X^{**}_n) \rightarrow \theta^{**}(1.1)$
   - 2.2 Échantillon *bootstrap* $1.2 : X^{**}(1.2) = (X^{**}_1, X^{**}_2, ..., X^{**}_n) \rightarrow \theta^{**}(1.2)$
   - 2.3 Échantillon *bootstrap* $1.3 : X^{**}(1.3) = (X^{**}_1, X^{**}_2, ..., X^{**}_n) \rightarrow \theta^{**}(1.3)$

     .....
   - 2.B Échantillon *bootstrap* $1.B : X^{**}(1.B) = (X^{**}_1, X^{**}_2, ..., X^{**}_n) \rightarrow \theta^{**}(1.B)$
3. On calcule t^*_1 et l'écart type de $\theta^*(1)$.
4. Échantillon *bootstrap* (2) :$X^*(1) = (X^*_1, X^*_2, \ldots, X^*_n) \rightarrow \theta^*(2)$
   - 4.1 Échantillon *bootstrap* $2.1 : X^{**}(2.1) = (X^{**}_1, X^{**}_2, ..., X^{**}_n) \rightarrow \theta^{**}(2.1)$
   - 4.2 Échantillon *bootstrap* $2.2 : X^{**}(2.2) = (X^{**}_1, X^{**}_2, ..., X^{**}_n) \rightarrow \theta^{**}(2.2)$
   - 4.3 Échantillon *bootstrap* $2.3 : X^{**}(2.3) = (X^{**}_1, X^{**}_2, ..., X^{**}_n) \rightarrow \theta^{**}(2.3)$

     .....

$4.B$ Échantillon *bootstrap* $2.B : X^{**}(2.B) = (X_1^{**}, X_2^{**}, ..., X_n^{**}) \rightarrow \theta^{**}(2.B)$

5. On calcule t_2^* et l'écart type de $\theta^*(2)$.

   etc.

Le paramètre étudié est θ, déduit de l'échantillon initial à partir duquel on construira l'intervalle de confiance

$\theta^*(1)$, $\theta^*(2)$, etc. sont les estimateurs *bootstrap* pour calculer l'écart-type de θ.

$\theta^{**}(1.1)$, $\theta^{**}(1.2)$, $\theta^{**}(1.3)$ etc. est l'ensemble des estimateurs *bootstrap* pour calculer t_1^* et l'écart-type de $\theta^*(1)$.

$\theta^{**}(2.1)$, $\theta^{**}(2.2)$, $\theta^{**}(2.3)$ etc. l'ensemble des estimateurs *bootstrap* pour calculer t_2^* et l'écart-type de $\theta^*(2)$, etc.

Disposant de la distribution des $t_b^* = \frac{\widehat{\theta}_b^* - \widehat{\theta}}{\widehat{\sigma}(\widehat{\theta}_b^*)}$, on en détermine les percentiles $\frac{\alpha}{2}$ et $1 - \frac{\alpha}{2}$ notés $t_{(\frac{\alpha}{2})}^*$ et $t_{(1-\frac{\alpha}{2})}^*$, et on obtient les limites de confiance du paramètre θ par les relations :

$$\hat{\theta}_{lo} = \widehat{\theta} - t_{(1-\frac{\alpha}{2})}^* \widehat{\sigma}_{\widehat{\theta}^*}, \tag{4.25}$$

et

$$\hat{\theta}_{up} = \widehat{\theta} - t_{(\frac{\alpha}{2})}^* \widehat{\sigma}_{\widehat{\theta}^*}. \tag{4.26}$$

À-propos des avantages remarquons que :

- la précision est de premier ordre ;
- il exige un grand nombre de *bootstrap*s puisqu'on a des *bootstraps* a deux niveaux ;
- il ne conserve pas le domaine ;
- il ne respecte pas la transformation ;

Le tableau 4.2 présente les caractéristiques des méthodes de calcul de l'intervalle de

méthodes	la méthode de l'écart type	méthode du *bootstrap* - t	méthode des percentiles simples	corrigés du biais et accélération
Nombre de ré échantillonnages	100	1000x100	1000	1000
Influence d'une transformation	oui	oui	non	non
Respect du domaine	non	non	oui	oui
Ordre de précision	$n^{-\frac{1}{2}}$	n^{-1}	$n^{-\frac{1}{2}}$	n^{-1}

Tableau 4.2: Caractéristiques des méthodes de calcul de l'intervalle de confiance d'un paramètre (1- la méthode de l'erreur-standard ; 2 - méthode du *bootstrap* - t, 3 - méthode des percentiles simples ; 4 - méthode des percentiles corrigés pour le biais et accélération.)

confiance d'un paramètre. La méthode de l'écart type est la plus rapide ($B = 100$, par exemple) alors que la méthode du *bootstrap* t est la plus coûteuse, puisqu'elle fait appel au *bootstrap* à deux niveaux : par exemple, B = 1000 répétitions au premier niveau, chacune de celles-ci faisant l'objet de $B^{'} = 100$ répétitions pour l'estimation de l'écart-type. La deuxième ligne du tableau concerne l'influence d'une transformation. La méthode de l'écart type et la méthode du *bootstrap* t sont influencées par une transformation alors que les méthodes basées sur les percentiles ne le sont pas. Aucune transformation n'est nécessaire d'être précisée pour les dernières méthodes citées. La troisième ligne du tableau indique si l'intervalle de confiance respecte le domaine dans lequel doit se trouver le paramètre θ. Enfin, la quatrième ligne a trait à l'ordre de précision.

Dans les études ultérieures, la moyenne, la médiane et le mode sont également analysés (quand les données sont biaisées le test de la médiane peut être plus utile que le test de la moyenne pour l'estimation du paramètre d'intérêt) Le *bootstrap* fournit des réponses convenables dans des circonstances défavorables.

L'analogue bayésien du *bootstrap* fréquentiste pour les variables aléatoires i.i.d a été proposé par Rubin (1981). L'idée est la suivante : sachant que les variables aléatoires $X = (X_1, \ldots, X_n)$ i.i.d ont une fonction de distribution F inconnue, $\theta(F, X)$ une fonction de la distribution F inconnue, et les données initiales sont X, on cherche la distribution à postériori de $\theta(F, X|X = x)$. L'approche bayésienne à ce problème est de conduite à une distribution à priori sur F, puis à utiliser la distribution à postériori du $\theta(F, X|X = x)$ pour inférer sur $\theta(F, X|X = x)$. Récemment, Lo (1987) a montré que le *bootstrap* bayésien et le *bootstrap* fréquentiste sont asymptotiquement équivalents. Weng (1989) a considéré les propriétés d'efficacité de second ordre ; Hjort (1991) a aussi développé le *bootstrap* bayésien pour des données censurées.

4.4.5 Les differents types d'intervalles de confiance pour r_T

Comment utiliser la théorie du *bootstrap* pour trouver l'intervalle de confiance de l'estimateur de la position de la mutation ? On s'intéresse à la distribution d'une statistique S (dans notre cas la position de la mutation) calculée à partir d'un échantillon de taille N qui contient n haplotypes de *cas* et m haplotypes de *témoins* ($N = n + m$). De plus, on a d différents types d'haplotypes de *cas*, chacun avec multiplicité n_i et respectivement l différents types d'haplotypes de *témoins*, chacun avec multiplicité m_j. Nous voulons construire des intervalles de confiance pour la position de la mutation qui est une variable aléatoire continue, mais mesurée par des valeurs discrètes (les points d'évaluation de la fonction de maximum de vraisemblance) selon la méthode *MapArg*. La position de la mutation est estimée par la fonction de maximum de vraisemblance composite. Premièrement, nous allons utiliser des échantillons de *bootstrap* pour construire des intervalles de confiance par la méthode de percentile, prouver la validité de ces intervalles et illustrer les résultats. Dans nos études, on construit les échantillons de *bootstrap* de deux façons différentes :

1. On tire au hasard les haplotypes de *cas* avec remise en produisant un nouvel échantillon d'haplotypes de *cas* ; ici on considère que les haplotypes des *cas* sont indépendants, et leur multiplicité n'est pas la conséquence d'une dépendance entre eux. De la même façon on construit l'échantillon *bootstrap* d'haplotypes de *témoins*.

2. Cette fois on considère qu'il y a une dépendance entre les haplotypes de même type et pour construire l'échantillon *bootstrap* d'haplotypes de *cas*, les haplotypes sont tirés par blocs (par exemple, l'ensemble d'haplotypes de même type est un bloc, si on a d différents types d'haplotypes on a d blocs). Les échantillons *bootstrap* d'haplotypes de *cas* et les échantillons *bootstrap* d'haplotypes de *témoins* sont obtenus par échantillonnage par blocs.

Les intervalles de confiance sont construits pour chaque type d'échantillonnage et leur comportements (exactitude, couverture, etc.) sont comparés. De plus, pour examiner l'influence de la taille de l'échantillon, nous avons essayé des tests par le *bootstrap* modifié, c'est-à-dire les échantillons *bootstrap* simulés ont une taille supérieure à la taille de l'échantillon initial. Soit B le nombre d'échantillons *bootstrap*. Notons par r_T la position exacte de la mutation et par $\hat{r}_T$ la maximum de la fonction de vraisemblance pour l'échantillon initial. Soit $\hat{r}^*_{T_i}$ $(i = 1, 2, \ldots, B)$ des solutions *bootstrap* de la fonction de maximum de vraisemblance pour les échantillons *bootstrap*. Comme on ne fait pas d'hypothèse au sujet de la loi des $\hat{r}^*_{T_i}$ $(i = 1, 2, \ldots, B)$, commençons avec la construction d'un intervalle de confiance empirique (l'intervalle de confiance basé sur les percentiles) Soit α le niveau désiré de confiance et supposons que les valeurs *bootstraps* $\hat{r}^*_{T_i}$ $(i = 1, 2, \ldots, B)$ sont ordonnées. Les limites inférieures et supérieures de l'intervalle de confiance seront $(\hat{r}^*_{T_i})_{\text{bas}}$ et $(\hat{r}^*_{T_i})_{\text{haut}}$ avec bas $= (\frac{\alpha}{2}) * B$ et haut $= (1 - \frac{\alpha}{2}) * B$. L'intervalle est choisi tel qu'il contient une proportion $(1 - \alpha)$ des valeurs *bootstrap*, et les valeurs aux extrémités sont coupées d'une façon symétrique. Si bas $= (\frac{\alpha}{2}) * B$ ou haut $= (1 - \frac{\alpha}{2}) * B$ n'est pas un nombre entier puis on utilise l'interpolation entre les deux quantiles. Pour illustrer ceci nous avons fait un histogramme de la distribution obtenue de $\hat{r}^*_{T_i}$ pour l'ensemble des données $A1$ (Figure 4.8).

Les autres types des intervalles de confiance ont également été construits, sauf l'intervalle bootstrap de confiance bilatérale symétrique de Student par la raison que on était limité par rapport du temps.

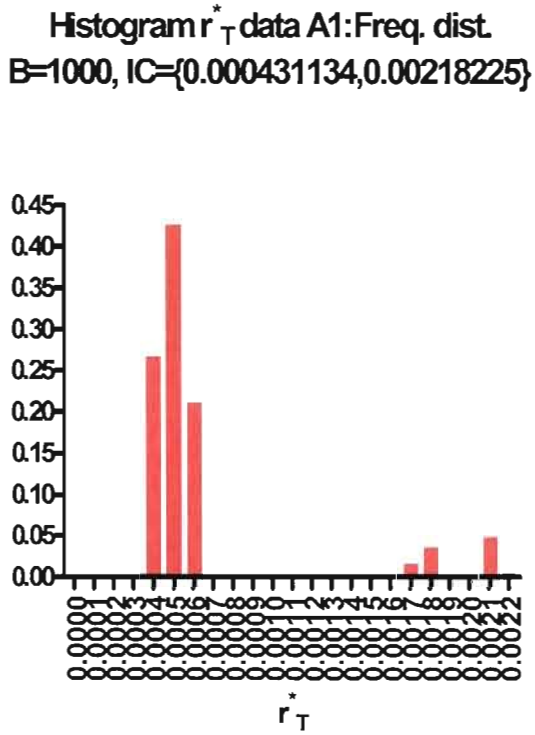
Selon la théorie, pour examiner l'exactitude d'intervalle de confiance, nous devrons faire l'expérience suivante : d'abord, on connaît la vraie position de la mutation dans les données

initiales simulées. Alors, nous produisons 100 fois 20, 50, 100, 200, 500 échantillons *bootstrap* de données en gardant le nombre initial de cas et de témoins. Pour chacun de ces échantillons on calcule l'intervalle de confiance de niveau 0.10, 0.05, 0.025, 0.01. Dans la réalité, une étude sur les diagrammes des fréquences des estimateurs nous a suggéré à chercher une façon différente afin d'obtenir des intervalles plus sûrs, ce qu'on verra dans le chapitre suivant.

Quelques mots pour les erreurs suivies d'une estimation bootstrap. Il y a deux sources d'erreur dans l'inférence de *bootstrap* :

1. l'erreur induite en employant un échantillon particulier X pour représenter la population ;
2. l'erreur produite par l'impossibilité d'énumérer tous les échantillons de *bootstrap*.

Cette source d'erreur peut être contrôlée en rendant le nombre B des ré échantillons *bootstrap* suffisamment grand ou en utilisant *double bootstrap*. Beran (1987) a montré que, pour l'échantillonnage iid, si certaines conditions de régularité sont satisfaites, les erreurs de *doubles bootstrap* intervalles de confiance convergente vers zéro plus rapidement que pour les *bootstrap* intervalles simples.



$$\mu_{r^*_{T_i}} = 0.00159425$$

$$r_T = 0.001784$$

$$\widehat{r_T} = 0.000431134$$

Figure 4.8: On remarque que l'intervalle rapporté contient des parties où l'estimateur n'a jamais pris des valeurs, de plus la moyenne de $\hat{r}^*_{T_i}$ tombe dans une telle partie.

CHAPITRE V

TESTS DE PERMUTATION

Les premières idées du test de permutation peuvent être trouvées dans les travaux de Neyman (1923), Fisher (1935a), et Pitman (1937), mais c'est dans le livre *«Design of Experiments»* (Fisher, 1935b) que l'on trouve la première description claire d'un test statistique basé sur les permutations des observations originales. Le test de permutation a été introduit comme une alternative non paramétrique du test t unidimensionnel et il a été employé pour remplacer la distribution de Student quand la normalité de la distribution des données ne pouvait pas être assurée. Un tel test est très intensif concernant le temps du calcul et avec l'augmentation de la puissance des ordinateurs, les tests de permutation ont été redécouverts (Edgington 1995 ; Manly 1997 ; Good 1994) et sont maintenant utilisés pour construire une distribution empirique de la statistique de test sous une hypothèse «nulle» afin de la comparer avec la distribution de la même statistique sous une hypothèse «alternative». Le test de permutation est une méthode non paramétrique qui nous permet de tester une hypothèse de façon empirique et de l'accepter ou de la rejeter quand les méthodes standards sont impuissantes. La méthode est convenable pour n'importe quelle statistique de test. L'application principale du test de permutation se retrouve dans les problèmes à deux échantillons.

5.1 Test d'hypothèse, principes généraux

La majorité des analyses statistiques font appel à des tests d'hypothèse. En statistique, un test d'hypothèse est utilisé comme une règle de décision entre deux scénarios. On définit une première hypothèse appelée l'hypothèse nulle (notée H_0), et en complément on définit une

seconde hypothèse appelée l'hypothèse alternative que l'on veut démontrer (notée H_1 ou H_A). Choisir entre les hypothèses H_0 et H_1 signifie de prendre le risque de se tromper par rapport à la réalité, et ce risque est mesuré par deux probabilités :

- *le risque de première espèce*, α, qui est la probabilité de rejeter l'hypothèse H_0 alors que l'hypothèse H_0 est vraie : $\alpha = P(\textit{Rejeter } H_0 | H_0 \textit{ est vraie})$ (tableau 5.1) ;
- *le risque de deuxième espèce*, β, qui est la probabilité de ne pas rejeter l'hypothèse H_0 alors que l'hypothèse H_A est vraie : $\beta = P(\textit{Ne pas rejeter } H_0 | H_1 \textit{ est vraie})$ (tableau 5.1).

	réalité	réalité
décision	l'hypothèse H_0 est vraie	l'hypothèse H_A est vraie
ne pas rejeter l'hypothèse H_0	$1-\alpha$	β l'erreur de 2ème espèce
rejeter l'hypothèse H_0	α l' erreur de 1ère espèce	$1-\beta$ puissance

Tableau 5.1: Les types d'erreurs et le test d'hypothèse

En pratique, il est aisé de connaître α-le risque d'erreur de 1-ère espèce, tandis que β-celui de 2-ème espèce, est difficile à calculer et souvent négligé. La probabilité que le test soit significatif (ou la probabilité de rejeter l'hypothèse nulle) quand l'hypothèse alternative est vraie est nommée la puissance du test. Elle se déduit du risque β par : $P(\textit{Rejeter } H_0 | H_1 \textit{ est vraie}) = 1 - \beta$ (un test est significatif quand la valeur observée de la statistique du test dépasse une certaine limite, appelée valeur critique, dont la valeur dépend du seuil de signification α choisi). La puissance n'est pas une valeur simple : elle change selon la valeur de la variable étudiée. Par exemple, la probabilité de rejeter l'hypothèse de l'égalité des moyennes dans deux populations dépend de la différence réelle entre les deux moyennes. La probabilité de rejeter l'hypothèse nulle augmente avec la différence entre les moyennes. Le raisonnement employé dans un test d'hypothèse est l'analogue statistique de celui connu en mathématique sous le nom de *raisonnement par l'absurde* : cherchant à montrer la présence d'un effet, on teste la

possibilité qu'il n'y ait pas d'effet. Une interprétation graphique de α, β et $1-\beta$ est donnée sur la Figure 5.1.

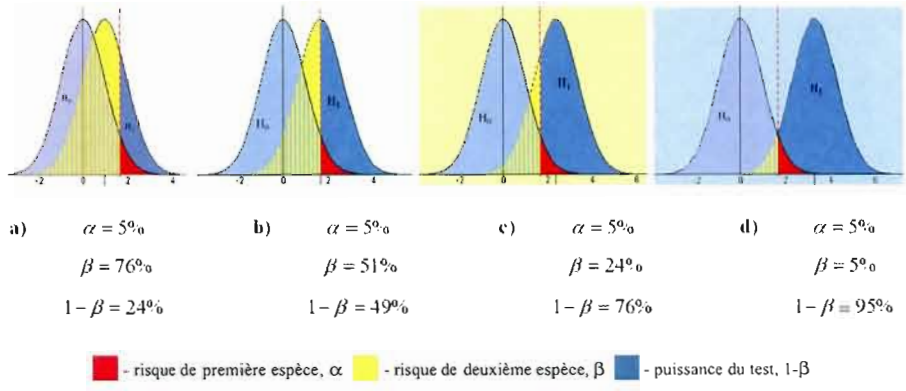


Figure 5.1: Supposons que nous voulons tester l'hypothèse nulle que les moyennes μ_1 et μ_2 de deux distributions sont égales ($\mu_1 = \mu_2$) contre l'hypothèse alternative qu'elles sont différentes ($\mu_1 \neq \mu_2$). Le risque de première espèce établi est 5%. Si on accepte l'hypothèse alternative c'est-à-dire $\mu_1 \neq \mu_2$, on prend le risque β de deuxième espèce. Dans le cas a) comme l'ilustre le graphique, pour 76% des valeurs de la statistique du test on ne rejette pas H_0. C'est-à-dire que le risque β de deuxième espèce est très élevé, donc la puissance du test, la probabilité de rejeter H_0 quand H_1 est vraie, est très petite. Dans les graphiques b), c) et d) la distance entre les moyennes augmente, ce que reflètent les probabilités β et $1-\beta$.

5.1.1 Niveau effectif de signification d'un test - *probabilité critique* (*p-value*)

Soit $\hat{\theta}$ l'estimateur initial du paramètre θ d'intérêt (θ arbitraire) et $\hat{\theta}_0$ la valeur hypothétique de θ générée en accord avec H_0. Le niveau effectif de signification d'un test, ou la *probabilité critique*, est défini comme suit :

$$p = P\left(\hat{\theta}_0 \geq \hat{\theta} | H_0\right). \tag{5.1}$$

Plus le niveau effectif de signification est petit, plus l'évidence que l'hypothèse H_0 est fausse est forte. C'est-à-dire, dans un problème de comparaison, la *probabilité critique* est définie comme le seuil auquel on observe une différence statistiquement significative, une différence avec une faible chance d'être due au hasard, la *probabilité critique* est associée avec la probabilité d'obtenir un effet au moins aussi grand que celui qui a été observée sous H_1, en supposant que l'hypothèse H_0 est vraie $p = P\left(\hat{\theta}^* \geq \hat{\theta}|H_0\right)$. Lorsque la *probabilité critique* est inférieure ou égale au risque accepté de première espèce α (souvent 5%), on rejette l'hypothèse H_0 et l'hypothèse H_1 est acceptée. On notera qu'il existe une différence entre la *probabilité critique* et α : α est la limite supérieure de *probabilité critique* que l'on accepte pour rejeter l'hypothèse nulle. Il existe plusieurs conceptions d'un test statistique ; par exemple, le test d'hypothèse selon la théorie de Neyman et Pearson et le test de signification selon l'approche de Fisher. Dans le premier cas, α est fixé et si la *probabilité critique* est inférieure à α, alors l'hypothèse H_1 est considérée comme plus probable ; α est défini a priori et p est calculé a posteriori à partir des résultats obtenus. La *probabilité critique* n'est qu'un intermédiaire de calcul pour choisir entre H_0 et H_1. Dans le deuxième cas, la *probabilité critique* est calculée et détermine le choix entre H_0 et H_1. En utilisant l'approche de Fisher, la *probabilité critique* est interprétée comme une mesure d'évidence d'une expérience simple et elle est supposée être combinée avec d'autres sources d'information. Ainsi, il n'y a aucun seuil pour la *signification* (Fisher, 1973). Ces deux approches sont différentes de part leur conception. Dans le présent travail on adoptera l'approche de Fisher.

Dans le cas de problèmes avec des comparaisons multiples, en augmentant le nombre de comparaisons il faut revoir une *probabilité critique* à la baisse (par exemple, descendre à 0.01 ou 0.005 selon le nombre de comparaisons) sinon on risque d'observer par simple hasard trop de tests *positifs*.

5.1.2 Signification de la *probabilité critique*

La *probabilité critique* est souvent considérée comme la probabilité que l'hypothèse nulle soit vraie, mais cette opinion est inexacte. Comme on a vu, la *probabilité critique* est la proba-

bilité d'observer des résultats égaux ou plus extrêmes que les résultats réels quand l'hypothèse nulle est vraie. Une petite *probabilité critique* signifie que des résultats aussi extrêmes que les résultats obtenus sont peu probables sous l'hypothèse nulle, et entraîne son rejet (la probabilité qu'un événement se soit produit est très petite si H_0 était vrai). La *probabilité critique* donne aussi une mesure du poids de l'évidence d'un test statistique, par exemple si on compare deux groupes A et B, plus la *probabilité critique* est petite, plus la différence entre eux est grande.

5.2 Permutation - notions de base

En mathématique, la notion de permutation exprime l'idée que des objets distincts peuvent être arrangés dans des ordres différents. Dans un cas simple, quand on a seulement un ensemble de n éléments, le nombre des réarrangements possibles entre eux est donné par la formule $n!$. Par exemple, si nous voulons énumérer tous les rangements (permutations) possibles de trois chiffres distincts 1, 2, 3 le nombre total des rangements est $3! = 6$ (Figure 5.2).

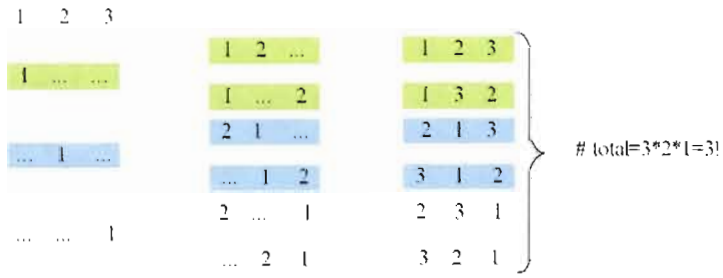


Figure 5.2: Prenons le premier élément, le chiffre 1, il y a trois positions possibles pour celui-ci, soit il occupe la première place, soit la deuxième, soit la troisième. Prenons maintenant le deuxième élément, le chiffre 2, pour chaque position de 1, il y a deux possibilités de placer 2. Une fois deux des trois places occupées, la position du troisième élément est déterminée de façon univoque. Ce qui donne le nombre total des permutations possibles $3 * 2 * 1 = 3!$.

Supposons maintenant que nous avons deux ensembles, un avec n éléments et un autre avec m éléments, et que nous voulons permuter les éléments de deux groupes. Le nombre de toutes les permutations possibles entre les éléments de deux groupes est donné par la formule 5.2 $C_{n!m!}^{(n+m)!}()$. De façon plus simple, une permutation peut-être interprétée comme l'échange d'un

certain nombre d'éléments du premier groupe avec le même nombre d'éléments du deux-ième groupe. Le rang de la permutation est le nombre d'éléments échangés à la fois. Par exemple, l'état initial de deux groupes est une permutation du rang 0 (chaque élément reste à sa place). Si un élément du deuxième groupe est échangé avec un élément quelconque du premier groupe, la permutation est de rang 1, etc. Le nombre de toutes les permutations possibles est la somme des nombres de permutations possibles pour chaque rang :

$$C_{n!m!}^{(n+m)!} = \binom{n}{0}\binom{m}{0} + \binom{n}{1}\binom{m}{1} + ... + \binom{n}{min(n,m)}\binom{m}{min(n,m)} = \sum_{i=0}^{\min(n,m)} \binom{n}{i}\binom{m}{i} \tag{5.2}$$

Le nombre des permutations possibles d'un rang définit les poids de ce rang par rapport au nombre total des permutations, C'est-à-dire qu'à chaque permutation, selon son rang, on peut assigner une probabilité qu'elle soit retirée de l'ensemble de toutes les permutations possibles. Ceci nous permet de définir une distribution des permutations de différents rangs. Cette distribution est utilisée pour désigner un sous-échantillon de permutations de l'ensemble de toutes les permutations possibles quand leur énumération est impossible en utilisant la méthode de l'échantillonnage pondéré (importance sampling).

Prenons l'exemple suivant (Figure 5.3) : un premier ensemble composé de trois éléments indexés par les chiffres 1, 2, 3 et un deuxième ensemble de deux éléments indexés par les chiffres 4 et 5. Le nombre d'éléments (2) du second ensemble est plus petit que le nombre d'éléments (3) de premier ensemble, c'est-à-dire que les permutations possibles peuvent être seulement de rangs 0, 1 et 2. Le nombre total de permutations possibles est $C_{3!2!}^{5} = 10$. Les poids de chaque rang sont comme suit $R_0 = \frac{1}{10}$, $R_1 = \frac{6}{10}$ et $R_2 = \frac{3}{10}$. On voit que ce sont les permutations de rang 1 qui contribueront le plus dans une analyse de variation entre les éléments de deux groupes.

5.3 Test de permutation

Dans le contexte de problèmes à deux échantillons, l'idée centrale du test de permutation est très élégante et simple. Si $A(a_1, ..., a_n)$ et $B(b_1, ..., b_n)$ sont deux échantillons aléatoirement

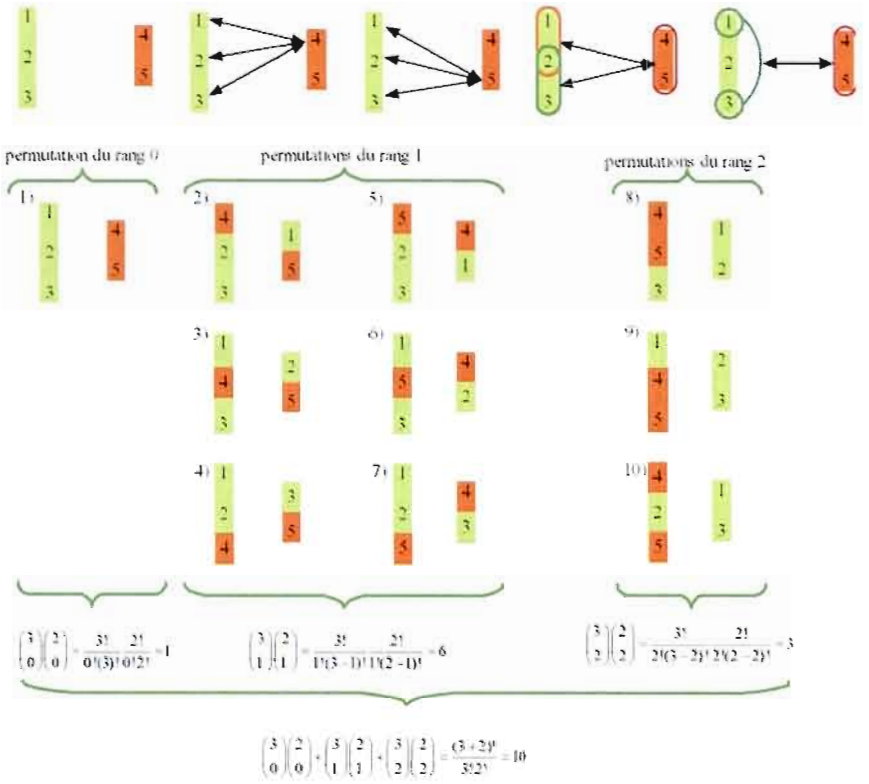


Figure 5.3: Nous voulons énumérer tous les rangements (permutations) possibles pour les éléments de deux groupes. Dans l'exemple présenté, le premier groupe contient les trois chiffres distincts 1, 2 et 3, le second 4 et 5, $n = 3$, $m = 2$, $min(n, m) = min(3, 2) = 2$. Les rangs possibles des permutations sont 0, 1 et 2. Le nombre de permutations possibles (des poids) du rang 0 est 1, du rang 1 est 6, et du rang 2 est 3. Donc au total 10.

tirés de la même population, ils constituent un ensemble simple d'individus auxquels les étiquettes *A* et *B* ont été assignées au hasard, c'est-à-dire les éléments des deux échantillons sont échangeables. N'importe quelle permutation entre eux aurait remanié les relations individus/étiquettes, mais n'aurait pas affecté les chances d'obtenir le même résultat du test (Bross, 1964). Par conséquent, la distribution de la statistique du test sous l'hypothèse nulle peut être obtenue simplement en calculant la statistique de test pour chaque permutation possible et en ordonnant les résultats. On peut employer le test de permutation exclusivement quand les données sont permutées d'une manière conforme à l'hypothèse nulle. Sous l'hy-

pothèse nulle, on suppose typiquement que les deux distributions conditionnelles sont identiques ($P(X = x|A) = P(X = x|B)$), ou d'une manière équivalente, que les données et les étiquettes sont indépendantes. Le but du test d'hypothèses est de rejeter l'hypothèse nulle à un certain niveau α de signification qui détermine la probabilité acceptable maximale que l'hypothèse nulle soit fausse (déclarant que les groupes sont différents quand l'hypothèse nulle est vraie). Comme on l'a vu, pour n'importe quelle valeur de la statistique du test, la *probabilité critique* correspondante est le plus petit seuil à partir duquel l'hypothèse nulle peut être rejetée c'est-à-dire la *probabilité critique* associée à l'observation d'une statistique de test est le seuil, auquel on rejetterait l'hypothèse nulle compte tenu de cette observation. Sous l'hypothèse nulle, les échantillons proviennent de la même loi de probabilité, les données peuvent être permutées et assignées au hasard à l'un ou l'autre des deux groupes, et on s'attend à ce que la distribution de l'estimateur sur les échantillons permutés soit la même que à la distribution de l'estimateur sur les échantillons initiaux.

Pour effectuer un test de permutation basé sur une statistique qui mesure l'effet d'intérêt, dans le cas général on suit les étapes suivantes :

- calculer la statistique de test pour les données initiales afin d'obtenir l'estimateur initial $\hat{\theta}$;
- choisir le nombre P de permutations à effectuer ;
- choisir les échantillons des données permutées d'une manière conforme à l'hypothèse nulle et à la conception d'étude ;
- calculer la statistique de test pour les données permutées pour obtenir l'estimateur $\hat{\theta_p}^*$ sous l'hypothèse nulle ;
- répéter la procédure en gardant les valeurs de $\hat{\theta_p}^*$, $p = 1, ..., P$ pour chaque échantillon permuté (Construire la distribution des valeurs de la statistique du test pour un grand nombre de re échantillons.) ;
- trouver la *probabilité critique* par identification de la position de la valeur de $\hat{\theta}$, l'estimateur initial sur la distribution de $\hat{\theta_p}^*$, $p = 1, ..., P$, (hypothèse nulle). On vérifie la probabilité d'obtenir une valeur aussi élevée sous H_0 ;
- rejeter l'hypothèse nulle pour une *probabilité critique* très petite.

5.3.1 Test de permutation dans le cadre de la méthode *MapArg*

Pour clarifier les idées proposées dans cette sous-section, une brève référence vers la méthode du maximum de vraisemblance est nécessaire. La méthode de maximum de vraisemblance part de l'hypothèse que l'échantillon observé suit une distribution dont un paramètre θ est à estimer. On fait l'hypothèse que la distribution est connue et seulement le paramètre θ ne l'est pas. Pour θ donné, la probabilité d'observer l'échantillon A est donnée par $P(A) = f(A, \theta)$, donc, la probabilité d'observer A dépend du paramètre θ. Le principe de la méthode est de chercher quel θ fournit la probabilité maximale d'observer A. On définit la fonction de vraisemblance, une fonction qui dépend de l'échantillon observé, dont le seul paramètre est θ, on la note $L(\theta) = f(A, \theta)$. Il s'agit alors simplement de rechercher la valeur $\hat{\theta}$ qui maximise $L(.)$ et on obtient l'estimateur $\hat{\theta} = \max_{\theta} L(\theta)$.

En pratique, on a juste un échantillon des données prélevées de la population d'intérêt. Avec la méthode du bootstrap on a tanté de modéliser des échantillons ayant la même probabilité d'être prélevés de la même population (des échantillons avec des histoires semblables) pour confirmer l'effet d'une mutation à la position $\hat{r}_T$ sur la population entière. Comme on l'a vu, les estimateurs bootstrap $\hat{r}_T^b$, $b = 1, 2, \ldots, B$ décrivent bien les régions ciblées sur la longueur de la séquence, les régions qui montrent une vraisemblance maximale sous les hypothèses de la méthode d'estimation. Comme on cherche une association maximale, le percentile d'ordre 5% de la distribution de $\hat{L}^b(\hat{r}_T^b)$, $b = 1, 2, \ldots, B$ est utilisée pour établir un seuil de signification pour les valeurs de la fonction de vraisemblance quand les conditions de l'hypothèse de pénétrance complète et l'absence des phénocopies sont satisfaites (données simulées, avec des caractéristiques connues). Cependant, la question de la conformité de l'estimateur $\hat{r}_T$ de la position probable de la mutation reste sans réponse quand les caractéristiques (vraie position de la mutation, penetrance, phénocopies) d'échantillon sont inconnues. Avec l'aide du test de permutation, on pourra répondre à cette question.

Ainsi, en suivant la définition de la méthode du maximum de vraisemblance, le résultat de la méthode de Larribe et al (2002) est traduit comme suit : l'ensemble initial d'haplotypes avec des paramètres N, n, m, d, l est plus probable d'être observé (selon la distribution proposée) si

une mutation est insérée à la position estimée $\hat{r}_T$. Les estimateurs de maximum de vraisemblance $\hat{r}_T^b$, $b = 1, \ldots B$ obtenus par la méthode *bootstrap* suivent un mélange des différentes distributions normales, ce qui est l'effet de l'estimation par intervalle. Ensuite, l'intervalle de confiance empirique de niveau 95% pour r_T est construit par la méthode de *bootstrap* :

$$IC_{95\%}(r_T) = (\hat{r}^*_{T_i,bas}; \hat{r}^*_{T_i,haut}), \tag{5.3}$$

$$bas = \frac{\alpha}{2} * B,$$
$$haut = (1 - \frac{\alpha}{2}) * B.$$

Par conséquent, on veut tester la signification de ce résultat, déterminer si l'estimateur et l'intervalle de confiance sont raisonnables.

Dans le problème d'estimation de la position probable de la mutation par la méthode *MapArg*, les tests utilisés ne sont pas indépendants (comme ils sont mélangés), ce qui rend le calcul d'un seuil de signification pour la fonction de vraisemblance impossible par les méthodes standards. À cause de cette difficulté, le test de permutation est développé pour construire une distribution empirique afin de déterminer le seuil critique pour les valeurs de la fonction de vraisemblance. Les valeurs - seuil calculées par ce test s'appliquent seulement aux données à partir desquelles elles ont été calculées. En particulier, les valeurs - seuil seront différentes pour les applications de la méthode sur les même données mais, avec des longueurs différentes de fenêtres.

Supposons qu'on a un ensemble d'haplotypes prélevés d'une population, et que chaque haplotype est classifié d'après son phénotype, soit dans le sous-ensemble des cas, soit dans le sous-ensemble des témoins. Si l'expression phénotypique n'est pas une conséquence d'une mutation commune pour les haplotypes du sous-ensemble des cas, et absente dans les haplotypes du sous-ensemble des témoins, chaque permutation entre les haplotypes de deux sous-ensembles mélangera les relationñ haplotype/phénotype mais n'affectera pas l'estimateur de la position de la mutation (l'estimation par la fonction de maximum de vraisemblance n'aurait pas donné une

préférence à une disposition particuliée des haplotypes). Donc, on s'attend à ce que les valeurs maximales de la fonction de vraisemblance pour chaque permutation ne varient pas beaucoup, et qu'elles soient équiprobables dans n'importe quelle position hypothétique de la mutation sur la longueur de la séquence. Quand les sous-ensembles sont permutés, la méthode est exécutée pour chaque permutation. Les paires $(\hat{r}_T^p, \hat{L}^p(\hat{r}_T^p))$ sont stockées et ordonnées par rapport aux valeurs de $\hat{L}^p(\hat{r}_T^p)$.

5.3.2 La probabilité critique locale, la probabilité critique globale, la probabilité critique ponctuelle

La probabilité critique locale - On définit la *probabilité critique locale* de notre statistique de test comme la probabilité de dépasser la valeur observée $\hat{L}(\hat{r}_T)$ de la statistique du test pour les données initiales à une position r_i indiquée sur la longueur d'haplotype, assumant l'hypothèse nulle. C'est-à-dire *la probabilité critique locale* notée p^l est un vecteur de probabilités notées $p^l_{r_i}$, où :

$$p^l_{r_i} = P(\hat{L}^p(r_i) \geq \hat{L}(\hat{r}_T)|H_0) = \frac{\sum_{p=1}^{P} I(\hat{L}^p(r_i) \geq \hat{L}(\hat{r}_T))}{P}, \tag{5.4}$$

$$\text{où} \quad I(\hat{L}^p(r_i) \geq \hat{L}(\hat{r}_T)) = \begin{cases} 1, & \text{si} \quad \hat{L}^p(r_i) \geq \hat{L}(\hat{r}_T) \\ 0, & \text{sinon,} \end{cases} \tag{5.5}$$

$$pour \quad i = 1, 2, ..., \text{nombre des points d'evaluation.}$$

La *probabilité critique locale* nous donne une idée sur la position dans l'haplotype qui présente une plus grande probabilité d'être associée avec la maladie sous H_0.

La probabilité critique globale - *La probabilité critique globale* est la probabilité que la valeur observée $\hat{L}(\hat{r}_T)$ de la statistique du test soit dépassée n'importe où sur la longueur d'haplotype, sous l'hypothèse nulle. C'est-à-dire *la probabilité critique globale* notée p^g est :

$$p^g = P(\hat{L}^p(\hat{r}_T^p) \geq \hat{L}(\hat{r}_T)|H_0) = \frac{\sum_{p=1}^{P} I(\hat{L}^p(\hat{r}_T^p) \geq \hat{L}(\hat{r}_T))}{P}. \quad (5.6)$$

Une troisième *probabilité critique* notée p_{r_i} est calculée par position r_i comme suit :

$$p_{r_i} = P(\hat{L}^p(r_i) \geq \hat{L}(r_i)|H_0) = \frac{\sum_{p=1}^{P} I(\hat{L}^p(r_i) \geq \hat{L}(r_i))}{P}. \quad (5.7)$$

Selon la nature de la méthode de Larribe et al (2002), la valeur de $\hat{L}(r_i)$ dépend du matériel génétique plus proche (entourant) la position r_i. Donc la p_{r_i} apporte de l'information sur les effets des fenêtres des marqueurs qui affectant la valeur de $\hat{L}(r_i)$.

Les hypothèses qu'on examinera par le test de permutation sont les suivantes :

- *L'hypothèse nulle composée* : l'association haplotype/phénotype n'est pas due à une mutation. On en déduit deux hypothèses comme suit :
  1. *L'hypothèse (H_{00})* : il n'existe pas une mutation influençant la maladie dans l'ensemble d'haplotypes donné.
  2. *L'hypothèse (H_{01})* : il existe une mutation, mais elle n'est pas liée à l'expresion phénotypique des haplotypes (c'est-à-dire la pénétrance est incomplète et il y a des phénocopies).
- *L'hypothèse alternative (H_A)* : L'association haplotype/phénotype est due à une mutation à la position estimée $\hat{r}_T$.

Les trois hypothèses sont les hypothèses globales et une traduction en termes statistiques est nécessaire.

Sous l'hypothèse H_{00} les haplotypes sont supposés échangeables et on s'attend à ce que les permutations entre les haplotypes de cas et de témoins ne favorisent pas une position particulière par rapport aux autres ; de plus comme la mutation est supposée inexistante, les valeurs $\hat{L}(\hat{r}_T)$ et $\hat{L}^p(\hat{r}_T^p)$ devraient être similaires.

Sous l'hypothèse H_{01}, les haplotypes sont supposés aussi échangeables, mais comme la pénétrance est incomplète et qu'il y a des phénocopies (les haplotypes sont mal classifiés), on s'attend à ce qu'ils existent des échantillons permutés qui favoriseront une position particulière (peut-être différente de $\hat{r}_T$) par rapport aux autres ; comme conséquence, les valeurs $\hat{L}^p(\hat{r}_T^p)$ dépasseront la valeur initiale $\hat{L}(\hat{r}_T)$ à certaines positions avec une probabilité significative.

Il est clair que si l'hypothèse H_A est vraie, la valeur initiale $\hat{L}(\hat{r}_T)$ dépasse les valeurs de $\hat{L}^p(\hat{r}_T^p)$. De plus, le test de permutation nous permet de vérifier si l'intervalle de confiance bootstrap est pertinent.

Un bref algorithme de test de permutation pour la méthode *MapArg* est le suivant. Soit n le nombre d'haplotypes *cas* et m le nombre d'haplotypes *témoins*. Si l'hypothèse nulle H_0 est vraie, les haplotypes de deux sous-ensembles sont échangeables :

1. On calcule la statistique du test en gardant le statut initial des haplotypes pour obtenir l'estimateur principal $\hat{L}(\hat{r}_T)$ de la probabilité conditionnelle maximale et la position $\hat{r}_T$ relative ($\hat{L}(\hat{r}_T) = \max_{r_i, i=1,..,k} L(r_i)$, $k =$ nombre des points evaluées). Les $L(r_i)$ sont aussi stockés. On conserve les valeurs de la fonction de vraisemblance pour chacun des points d'évaluation.
2. Un nombre P de permutations à effectuer est choisi. Le choix de P est expliqué dans la sous-section 5.3.3.
3. Sous l'hypothèse nulle que les données proviennent de la même population et qu'il y a absence de liaison entre la position de la mutation et le statut d'haplotypes (les données sont alors échangeables), on peut rassembler les deux sous-échantillons de cas et de contrôles dans un ensemble de taille $n + m = N$.
4. Une fois les sous-échantillons rassemblés on effectue les permutations entre les haplotypes *cas* et les haplotypes *témoins* d'une façon standard. On tire au hasard, sans remise, un nombre n d'haplotypes, ce qui constitue l'échantillon de cas et les données restantes (m) forment, l'échantillon de témoins. Pour chaque permutation, on vérifie si l'échantillon obtenu est identique avec les données initiales ou s'il a été déjà énuméré parmi les permutations précédentes, si oui on ne calcule pas la statistique du test pour

cet échantillon, et on lance la permutation suivante. La procédure est répétée jusqu'à au moment où on atteint le nombre des permutations désiré.

5. Pour chaque permutation p ($p = 1, 2, ..., P$), on calcule la statistique du test $\hat{L}^p(r_i)$ ($i = 1, ..., k$) et on garde les valeurs obtenues pour chacune des positions.
6. Pour chaque permutation on calcule $\hat{L}^p(\hat{r}_T^p) = \max_{r_i, i=1,...,k} L^p(r_i)$, l'estimateur de l'association maximale avec la mutation et la position relative $\hat{r}_T^p$.
7. Pour chaque position on calcule la *probabilité critique locale* :

$$p_{r_i}^l = P(\hat{L}^p(r_i) \geq \hat{L}(\hat{r}_T)|H_0) = \frac{\sum_{p=1}^{P} I(\hat{L}^p(r_i) \geq \hat{L}(\hat{r}_T))}{P}. \tag{5.8}$$

8. Pour $\hat{L}(\hat{r}_T)$ et $\hat{L}^p(\hat{r}_T^p)$ ($p = 1, 2, ..., P$), on calcule la *probabilité critique globale* :

$$p^g = P(\hat{L}^p(\hat{r}_T^p) \geq \hat{L}(\hat{r}_T)|H_0) = \frac{\sum_{p=1}^{P} I(\hat{L}^p(\hat{r}_T^p) \geq \hat{L}(\hat{r}_T))}{P}. \tag{5.9}$$

9. On analyse les résultats.

Les valeurs critiques pour établir le seuil de signification du test de permutation, qui sont classifiées suggestives, significatives, et fortement significatives, sont prises des travaux de Lander et de Kruglyak (1995), et correspondent aux trente-septième, quatre-vingt-quinzième, et 99.9-éme percentiles, de la distribution de $\hat{L}^p(\hat{r}_T^p)$ (Par exemple, le quatre-vingt-quinzième percentile est celui correspondant au critère habituel $\alpha = 0.05$).

5.3.3 Nombre de permutations nécessaire

On a vu que le nombre de combinaisons possibles entre les éléments de deux groupes avec m et n éléments respectivement est

$$C_{mn}^{m+n} = \frac{(m+n)!}{m!n!}, \tag{5.10}$$

ce qui nous donne par exemple pour $m = 29$ et $n = 29$ un nombre de permutations possibles de $C_{mn}^{m+n} = 30\ 067\ 266\ 499\ 541\ 040$, soit plus de 30 billiards. Si on avait besoin d'une seconde seulement pour calculer la statistique de test pour une permutation, ce qui est totalement irréaliste, le temps nécessaire pour obtenir la *probabilité critique* exacte dans cet exemple serait d'environ 956 076 070 ans, ou d'environ 1 milliard d'années (l'âge de la Terre est actuellement estimé à 4.5 milliards d'années). Par conséquent, il y a une contrainte évidente de temps de calcul pour un test de permutation. Pour éviter cette contrainte, on utilise la méthode Monte Carlo, en employant un échantillon aléatoire des permutations plutôt que toutes les permutations dans le calcul de *probabilité critique*, (la probabilité de sélectionner n'importe quel échantillon particulier soit égale à $\frac{1}{C_{mn}^{m+n}}$ ou dans le choix d'échantillon permuté, chaque permutation entre les éléments de deux groupes soit équiprobable). Une autre solution est la méthode de l'échantillonnage pondéré (importance sampling), une méthode basée sur les poids de différents rangs de permutations. Le résultat de ceci est que la *probabilité critique* calculée est elle-même une variable aléatoire, et donc on a besoin d'information sur sa précision.

5.3.4 Intervalle de confiance pour la *probabilité critique*

La *probabilité critique (p-valeur)* unilatérale du test de permutation est le nombre de statistiques de test plus grandes ou égales que la statistique de test basée sur l'échantillon initial, donc la *probabilité critique* est simplement une proportion estimée distribuée selon la loi binomiale. L'approximation normale (théoreme de de Moivre-Laplace) de la distribution binomiale nous permet facilement d'obtenir un niveau de précision désiré pour la *probabilité critique* estimée comme fonction de *l'écart type* (se_p) ou du *coefficient de variation* (cv_p) (Brown et al., 2001) :

$$\hat{se}_p \approx \sqrt{\frac{p(1-p)}{P}}, \tag{5.11}$$

où p =*p-valeur* et P est le nombre de permutations effectuées, et *l'intervalle de confiance* symétrique de niveau α est donné par :

$$IC_{1-\alpha} \approx \left\{ p - valeur \pm (z_{\frac{\alpha}{2}} \times se_p) \right\}, \tag{5.12}$$

$$z_{\frac{\alpha}{2}} : P(Z \geq z_{\frac{\alpha}{2}}) = \frac{\alpha}{2}, \tag{5.13}$$

pour $z_\alpha \approx N(0;1)$. Enfin, le *coefficient de variation* de la *probabilité critique* est :

$$cv_p = \frac{se_p}{p - valeur}. \tag{5.14}$$

Exemple 5.1 : a) Un nombre de 1 000 *permutations est effectué, la statistique de test est calculée pour chaque ensemble d'haplotypes permutés. Les résultats sont ordonnés et comparés avec la statistique du test calculée sur l'échantillon initial. La probabilité critique obtenue est* $p = 0.05$. *Dans ce cas, l'écart type, l'intervalle de confiance et le coefficient de variation de la probabilité critique sont calculés comme suit :*

1. $se_p \approx \sqrt{\frac{0.05(1-0.5)}{1000}} \approx 0.0069$
2. $IC_{95\%} \approx p - valeur \pm (1.96 \times se) \Leftrightarrow IC_{95\%} = (0.036476; 0.063524)$
3. $cv = \frac{se}{p-valeur} = \frac{0.0069}{0.05} = 0.138$

L'écart type et le coefficient de variation peuvent être contrôlées par le nombre de permutations. Plus le nombre de permutations est grand, plus l'écart type est petit et plus le coefficient de variation diminue.

La *marge d'erreur* d est un estimateur de l'étendue que les résultats peuvent avoir si l'on recommence l'expérience. Elle donne la precision recherchée ou l'intervalle de confiance. Elle dépend de l'écart type et du seuil de signification α choisie, donc elle depend uniquement de la taille de l'échantillon et en aucun cas de la taille de la population mère. On a :

$$d = z_{\frac{\alpha}{2}} \sqrt{\frac{p - valeur(1 - (p - valeur))}{P}},$$

où $z_{\frac{\alpha}{2}}$ est tel que $P(Z \geq z_{\frac{\alpha}{2}}) = \alpha$ pour $Z \approx N(0;1)$. La *marge d'erreur* d prend uniquement en compte l'erreur de l'échantillon. Plus la marge d'erreur est petite, plus la taille P d'échantillon est grande :

$$P = \frac{\textit{probabilité critique}(1 - \textit{probabilité critique})}{(\frac{d}{t})^2}.$$

Quand le nombre de permutations effectuées (la taille de l'échantillon utilisé) est de 10% ou plus du nombre total des permutations possibles (de celle de la population), un facteur de correction est appliqué selon la formule suivante :

$$P' = \frac{P}{1 + \frac{P}{C_{mn}^{m+n}}}.$$

Exemple 5.1 b) Suite de l'exemple 5.1 a). On suppose qu'il s'agit d'une première étude et donc la $p - valeur$ *est inconnu. Au début de chaque expérience, on désire avoir une marge d'erreur acceptable et une certitude que les résultats obtenus ne sont pas dûs au hasard. On a besoin de s'assurer qu'une fois l'expérience répétée, le résultat ne différera pas beaucoup de celui déjà calculé. On fixe une marge d'erreur raisonnable par exemple* $d = 5\%$ *et un niveau de confiance de* $95\% \Leftrightarrow z_{\frac{\alpha}{2}} = 1.96$. *Comme la proportion à estimer est inconnue, on prend* $p - valeur = 50\% = 0.5$ *(l'état d'équilibre). Maintenant on peut avoir une première idée au sujet de la taille d'échantillon à prélever :*

$$d = 5\% \Rightarrow P = \frac{0.50^2}{(\frac{0.05}{1.96})^2} = 400,$$

si

$$d = 3\% \Rightarrow P = \frac{0.50^2}{(\frac{0.03}{1.96})^2} \approx 1111,$$

si

$$d = 1\% \Rightarrow P = \frac{0.50^2}{(\frac{0.01}{1.96})^2} = 10000.$$

Ensuite la taille optimale d'échantillon est ajustée en contrôlant l'écart type de la probabilité critique. Si on connaît déjà la probabilité critique on peut calculer la marge d'erreur pour chaque niveau de confiance.

5.3.5 Construction d'un échantillon permuté par les méthodes Monte Carlo et échantillonnage pondéré, résultats

Le premier choix d'algorithme pour construire les permutations entre les deux sous - ensembles était la méthode Monte Carlo (Figure5.4). Par la méthode Monte Carlo, on examine un ensemble de permutations aléatoirement tirées de l'ensemble de toutes les permutations possibles. Les tailles n et m de deux sous-ensembles sont comparées et le minimum de n et m est calculé ($c = min(n; m)$), les sous-ensembles de cas et de témoins sont réunis en un seul ensemble, un nombre $c = min(n; m)$ d'haplotypes est tiré au hasard. Alors, si $c = n$, le statut d'haplotypes tirés est converti a 1 (*cas*), le statut d'haplotypes restants est converti à 0 (*témoins*), si $c = m$, le statut d'haplotypes tirés est converti a 0-*témoins*, le statut d'haplotypes restants est converti a 1-*cas*. Par la suite, on a employé l'échantillonnage pondéré, une méthode qui prend en considération des poids de différents rangs des permutations. Dans le programme *MapArgBP* aucune *règle d'arrêt* n'a été employée, le nombre de permutations désirées est entré par la ligne de commande.

Revenons sur la méthode *MapArg*, la construction d'un intervalle de confiance pour $\hat{r}_T$ et le seuil de signification pour les valeurs de la fonction de la vraisemblance. La méthode *MapArg* donne une très bonne estimation pour la vraie position de la mutation quand la longueur des fenêtres utilisées est grande, mais il arrive souvent que pour des petites fenêtres, ou quand la vraie position est très proche d'un marqueur connu, que l'estimateur obtenu soit fortement biaisé (en particulier pour certains types de données comme $A1$, $Z1$, $H1$ etc.). L'intervalle de confiance empirique était choisi parmi tous les types d'intervalles de confiance construits par la méthode de bootstrap pour les raisons suivantes : premièrement, on n'a aucune idée de la distribution de $\hat{r}_T^b$, ce qui rend inutile l'intervalle de confiance basé sur une distribution particulière ; deuxièmement, la construction d'un intervalle de confiance par la méthode de bootstrap «t» est très coûteuse par rapport au temps de calcul, car on a un bootstrap à deux niveaux. La contrainte imposée par le temps du calcul ne nous permet pas d'analyser d'une façon exhaustive tous les ensembles de données simulées fournis au début de cette recherche (ensembles *A1*, *B1*, *C1*, *D1*, *E1*, *F1*, *G1*, *H1*, *S1*, *T1*, *U1*,*V1*, *W1*, *X1*, *Y1*, *Z1*). La performance de l'intervalle de confiance

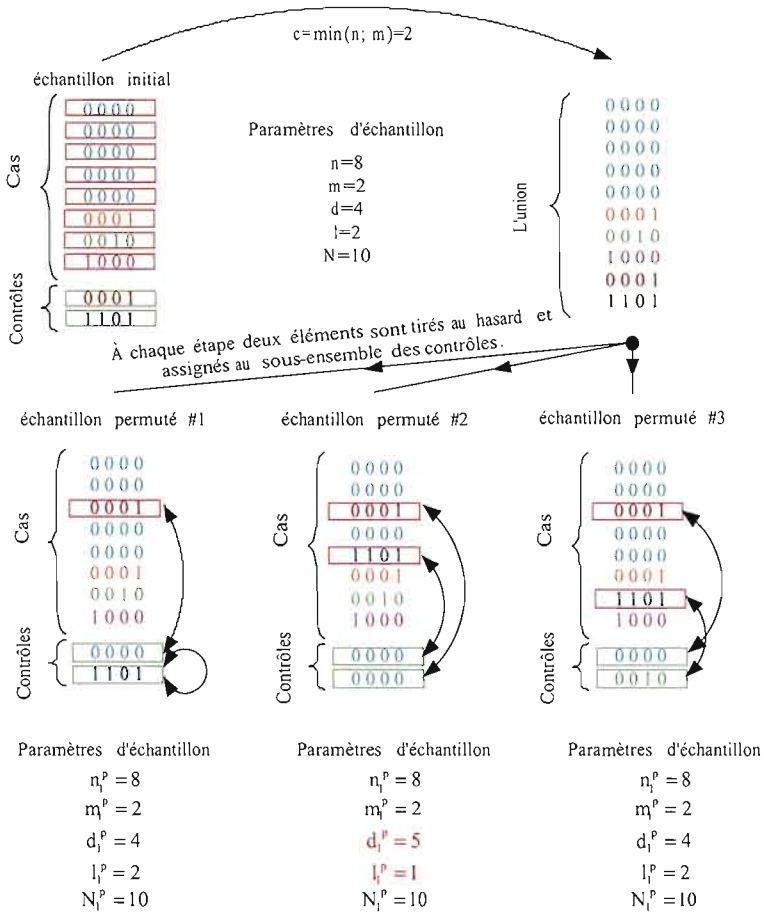
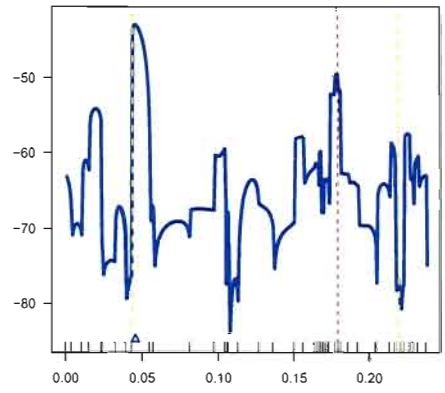
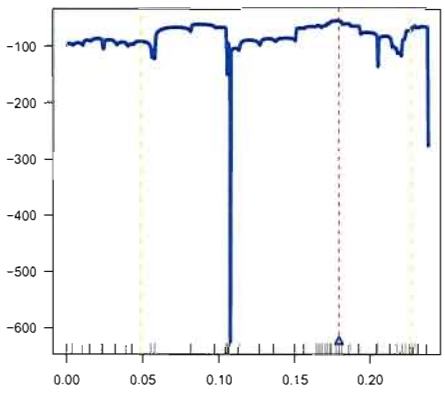


Figure 5.4: Ré échantillonnage Monte Carlo

empirique est satisfaisante pour tous les ensembles de données simulées sur lesquels l'analyse était faite. Par contre, on propose une variante de maximisation de la fonction de vraisemblance, ce qui change l'estimateur obtenu par la méthode *MapArg* et la méthode de la construction de l'intervalle de confiance. Nous n'avions pas à notre disposition un ensemble de données, pour lequel on pourrait énumérer toutes les permutations possibles ; pour cette raison, les analyses sont faites avec des échantillons représentatifs de permutations de taille 500, 1 000, 3 000, 5 000 et 10 000. Les graphiques montrés sur la figure 5.5 ((a) A1.dat, L=2, B=5 000, b) A1.dat, L=5, B=500), représentent les intervalles de confiance bootstrap empirique de niveau 95% ($IC_{95\%}$) pour l'ensemble des données $A1$ avec des fenêtres de longueur 2 et 5.



(a) A1.dat, L=2, B=5000

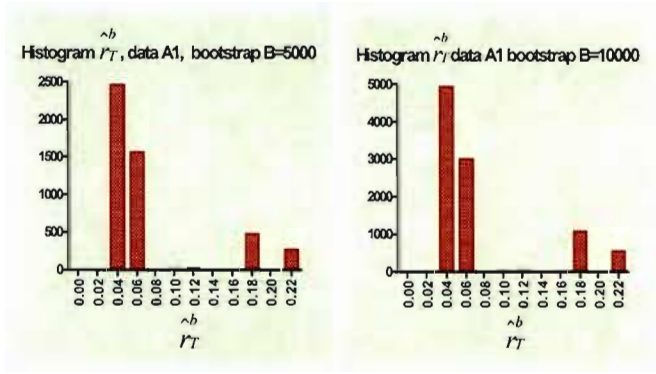


(b) A1.dat, L=5, B=500

Figure 5.5: La fonction de vraisemblance est calculée pour l'ensemble de données $A1$ (la longueur des fenêtres utilisées est a) 2, b) 5, B est le nombre de bootstraps), une représentation graphique est donnée par la ligne bleue (les paires $(r_i, log(L(r_i)))$). Les lignes jaunes déterminent les limites de l'intervalle de confiance bootstrap empirique de niveau 95% ($IC_{95\%}$).

On observe un intervalle de confiance très large indépendamment de la longueur des fenêtres utilisées.

Une fois l'intervalle de confiance construit, on peut naturellement se demander comment les différentes valeurs de $\hat{r_T}^b$, $b = 1, 2, \ldots, B$ sont distribuées à l'intérieur de $IC_{95\%}$, et si elles suivent une distribution particulière ; comment le nombre de bootstraps change cette distribution ? La figures 5.6 représentent les histogrammes des fréquences des valeurs de ${r_T}^b$, $b = 1, 2, \ldots, B$ pour pour l'ensemble des données $A1$ avec des fenêtres de longueur 2 et respectivement $B = 5\,000$, $B = 10\,000$.

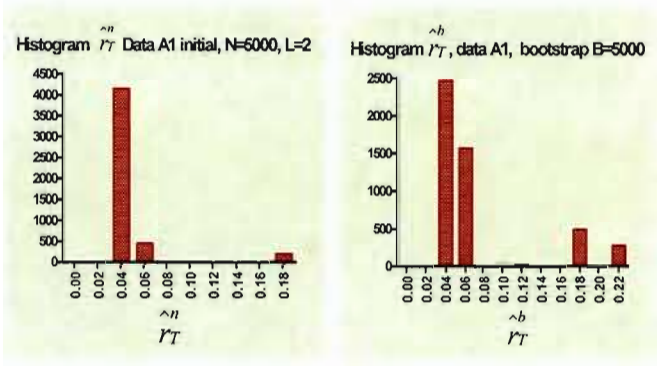


(a) A1.dat, L=2, B=5 000 (b) A1.dat, L=2, B=10 000

Figure 5.6: Les histogrammes des fréquences des estimateurs bootstraps sont construits pour deux analyses indépendantes avec des nombres de bootstraps différents, $B_1 = 5000$, $B_2 = 10000$.

On peut dire qu'un plus grand nombre de bootstraps ne reflète pas la distribution des estimateurs bootstrap et qu'un nombre de bootstraps $B = 5000$ est suffisant pour donner une idée sur la variabilité de l'estimateur.

Pour donner une idée au sujet de variabilité de l'estimateur dû à la méthode *MapArg* et à la méthode bootstrap on a fait l'expérience suivante : le programme *MapArg* était lancé 5 000 fois avec les données initiales, les estimateurs était stockés et l'histogramme des fréquences était construit et comparé avec l'histogramme des fréquences des estimateurs bootstraps (B=5 000). Les résultats sont présentés à la figure 5.7.

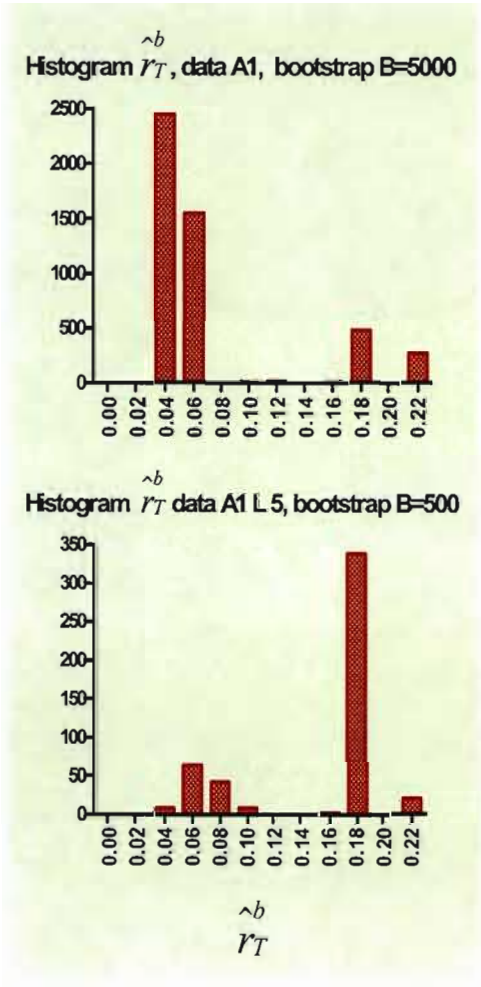


(a) A1.dat, L=2, N=5000 (b) A1.dat, L=2, B=1000

Figure 5.7: On voit que la méthode *MapArg* et la méthode bootstrap varient de même façon, mais la méthode bootstrap fournit plus d'estimateurs proches de la vraie position $r_T = 0.1784cM$ de la mutation.

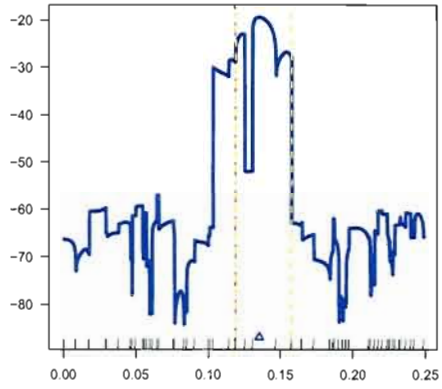
On voulait savoir comment la longueur de la fenêtre influence la distribution des estimateurs bootstrap. Les histogrammes de deux distributions de $\hat{r_T}^b$ obtenus pour des fenêtres de longueur 2 et 5, (le nombre de bootstrap est $B = 5000$ pour la fenêtre de longueur 2 et $B = 500$ pour la fenêtre de longueur 5) pour l'ensemble de données $A1$ sont montrés sur la figure 5.8.

On remarque que les valeurs bootstrap de r_T sont groupées et l'intervalle de confiance est divisé en sous-intervalles où l'estimateur bootstrap apparait avec une certitude et en sous-intervalles où il n'est jamais apparu. Est-ce qu'une telle dispersion est une caractéristique de l'intervalle de confiance bootstrap et ne dépend pas de l'ensemble de données ? Regardons les intervalles de confiance pour les ensembles des données restants (figures 5.9, 5.10, 5.11, 5.12 et 5.13).

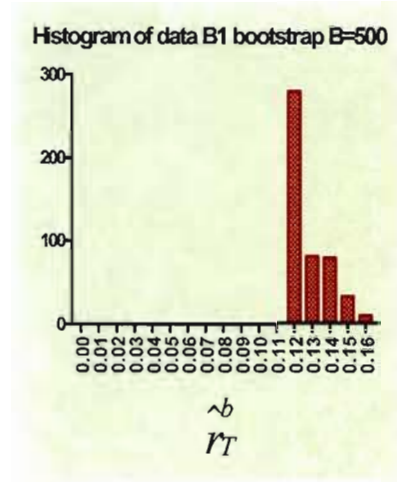


(a) A1.dat, L=2, B=5000 ; A1.dat, L=5, B=500

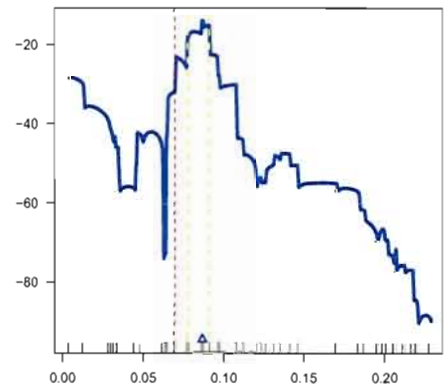
Figure 5.8: Évidemment, la longueur des fenêtres change considérablement la distribution de l'estimateur $\hat{r_T}^b$ à l'intérieur de l'intervalle de confiance. Au contraire, les limites de l'intervalle de confiance restent inchangées indépendamment de la longueur des fenêtres.



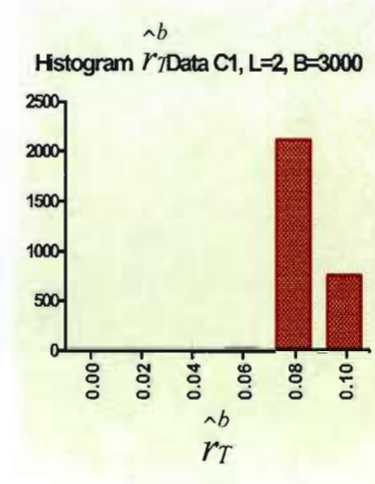
(a) B1.dat, L=2, B=1000, $r_T = 0.11855$



(b) B1.dat, L=2, B=1000, $r_T = 0.11855$

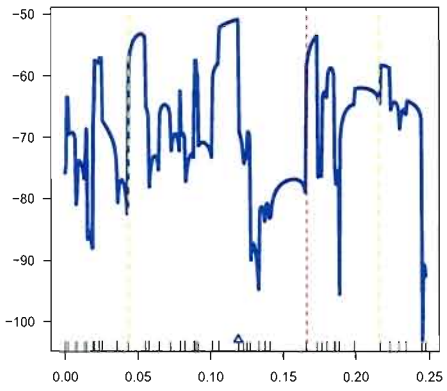


(c) C1.dat, L=5, B=1000, $r_T = 0.06895$

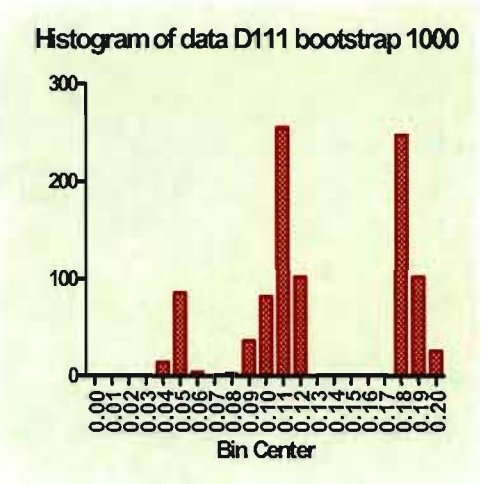


(d) C1.dat, L=5, B=1000, $r_T = 0.06895$

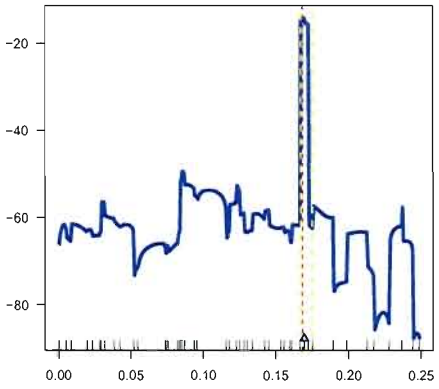
Figure 5.9: Pour les ensembles de données $B1$ et $C1$ on remarque que les estimateurs ne sont pas dispersés a l'intérieur de l'intervalle de confiance. On voit que pour l'ensemble des données B1 30% des estimateurs bootstraps sont très proches de la vraie position de la mutation. Au contraire, pour l'ensemble des données $C1$, l'intervalle de confiance bootstrap ne contient pas la vraie position de la mutation et on a besoin d'un nombre de bootstraps beaucoup plus grand pour s'assurer que la vraie position de la mutation se situe dans l'intervalle de confiance.



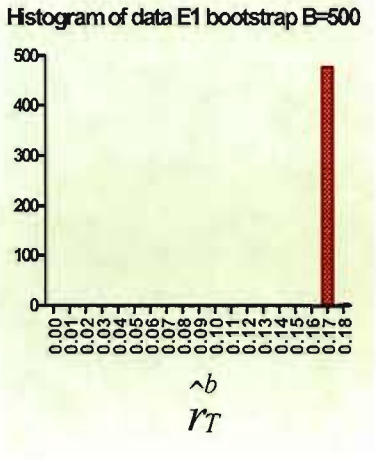
(a) D1.dat, L=2, B=5000, $r_T = 0.1661$



(b) D1.dat, L=2, B=5000, $r_T = 0.1661$

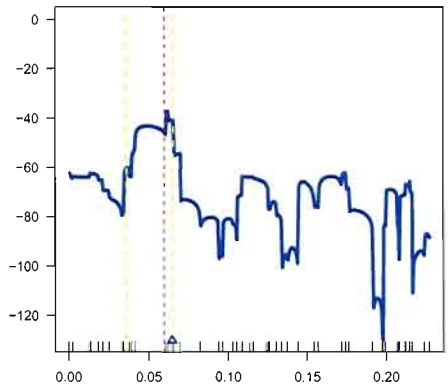


(c) E1.dat, L=2, B=500, $r_T = 0.16865$

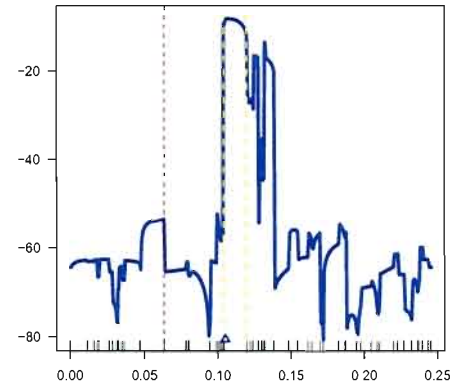


(d) E1.dat, L=2, B=500, $r_T = 0.16865$

Figure 5.10: On voit que pour l'ensemble de données $D1$, les estimateurs bootstraps sont dispersés à l'intérieur de l'intervalle de confiance et que même si la vraie position de la mutation est dans l'intervalle de confiance bootstrap, on n'a jamais obtenu un estimateur près de celle ci. Au contraire, pour l'ensemble des données $E1$, tous les estimateurs bootstrap sont concentrés autour de la vraie position de la mutation et l'intervalle de confiance est bon.

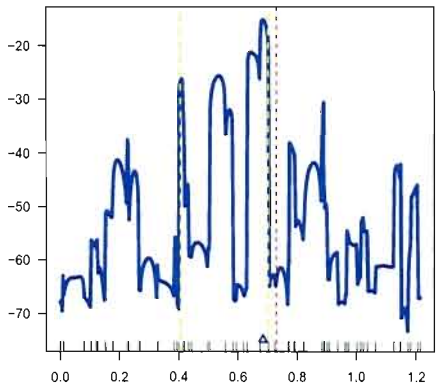


(a) G1.dat, L=3, B=1000, $r_t = 0.059625$

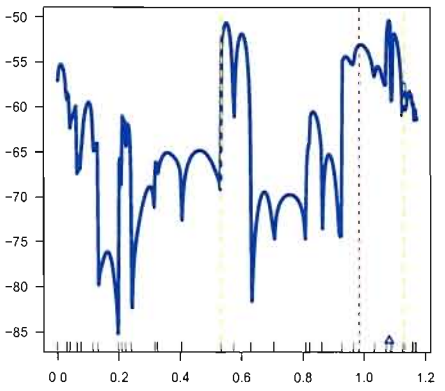


(b) H1.dat, L=2, B=1000, $r_t = 0.0636$

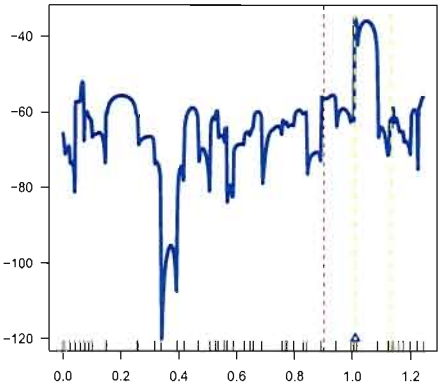
Figure 5.11: Pour l'ensemble de données $G1$, l'histogramme des fréquences des estimateurs bootstraps n'est pas montré, les distances entre les couples de marqueurs (m47 ; m48) et (m51 ; m52) sont 0, ce qui fait que l'estimateur *MapArg* et respectivement les estimateurs bootstraps sont fortement biaisés. Si on enlève les marqueurs appropriés, on obtient un bon intervalle de confiance. Pour l'ensemble des données $H1$, même si l'intervalle de confiance bootstrap semble bon, la vraie position de la mutation est toujours à l'extérieur.



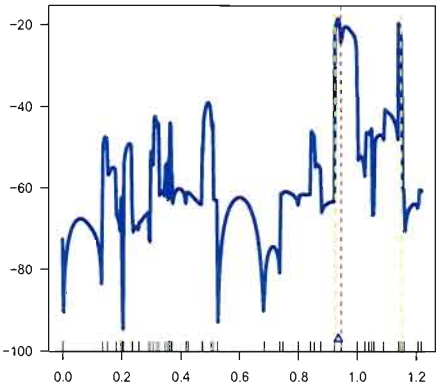
(a) U1.dat, L=2, B=1000



(b) V1.dat, L=2, B=3000

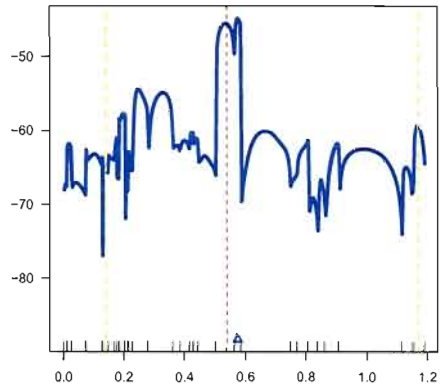


(c) W1.dat, L=2, B=5000

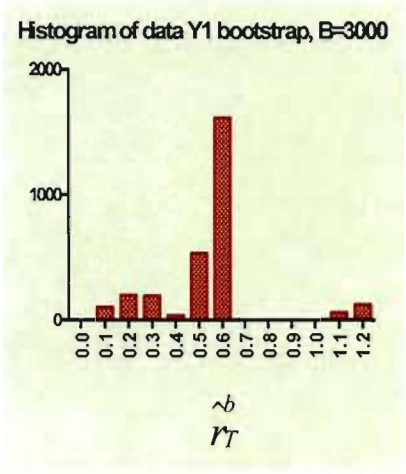


(d) X1.dat, L=2, B=1000

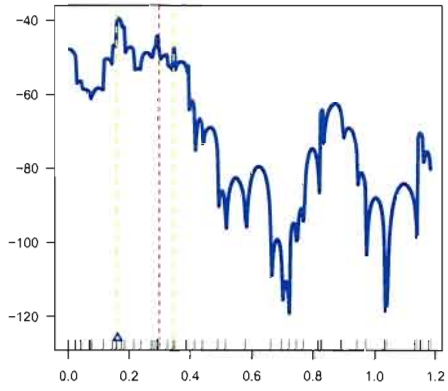
Figure 5.12: Les intervalles de confiance bootstrap pour les ensembles de données $U1$, $V1$, $W1$ et $X1$.



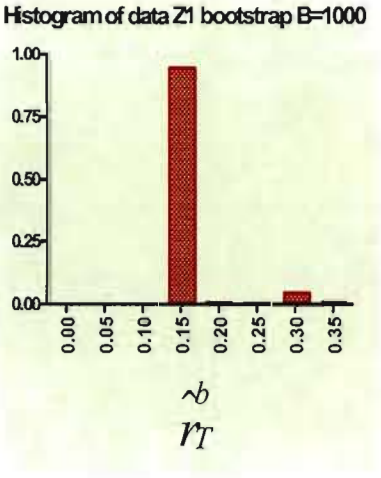
(a) Y1.dat, L=2, B=1000



(b) Y1.dat, L=2, B=1000



(c) Z1.dat, L=5, B=1000



(d) Z1.dat, L=10, B=2000

Figure 5.13: Pour l'ensemble de données $Y1$, une grande partie des estimateurs bootstrap est concentrée autour de la vraie position de la mutation, mais l'intervalle de confiance est très large. Pour l'ensemble de données $Z1$ la vraie position de la mutation est à l'intérieur de l'intervalle de confiance, mais très peu d'estimateurs bootstraps sont prés de la vraie position.

Comme on l'a vu, l'intervalle de confiance bootstrap pour l'estimateur obtenu par la méthode *MapArg* n'est pas un intervalle de confiance typique, ce qui nous poussait à chercher une façon de tester l'intervalle de confiance et déterminer un sous-intervalle de confiance. Un test de permutation est développé pour établir le seuil de signification de la fonction de vraisemblance.

Avant de procéder a test de permutation, nous ferons une remarque au sujet des valeurs de la fonction de vraisemblance pour les données initiales. Pour chaque position d'évaluation r_i de la fonction de la vraisemblance, la moyenne bootstrap de $\hat{L}^b(r_i), b = 1, \ldots, B$ est calculée ; regardons, par exemple, le graphique pour l'ensemble de données $Z1$ (figure 5.14). La ligne bleue représente le graphique de la fonction de vraisemblance pour les données initiales et la ligne noire représente la moyenne bootstrap calculée par point.

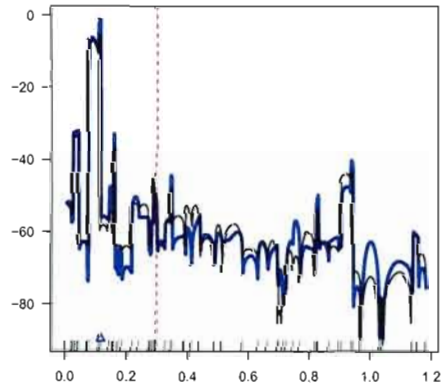
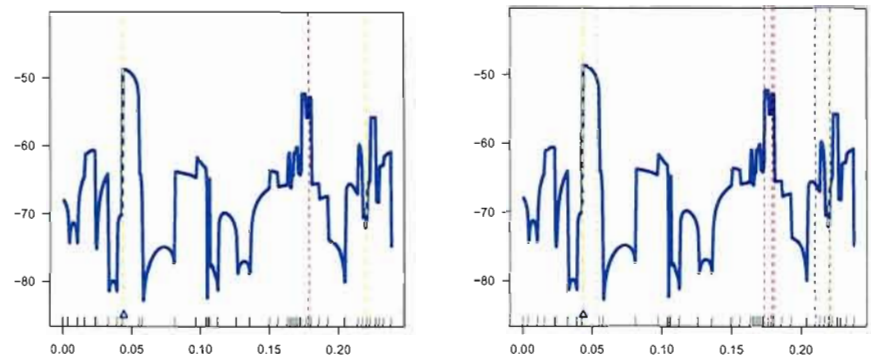


Figure 5.14: On remarque que la moyenne ne diffère pas beaucoup par rapport aux valeurs initiales de la fonction de la vraisemblance, ce qui nous donne une stabilité de $\hat{L}(r_i)$, toutes les comparaisons qui suivent sont fait par rapport aux valeurs initiales de la fonction de vraisemblance.

Montrons maintenant comment tester l'intervalle de confiance avec l'aide du test de permutation. Dans l'exemple qui suit, le nombre de bootstraps et le nombre de permutations sont maintenus égaux ce qui nous donne le même nombre d'estimateurs sous les deux hypothèses

testées (la méthode de bootstrap fournit des estimateurs sous H_1 ; au contraire, le test de permutation fournit des estimateurs sous H_0). Soit le nombre de bootstrap $B = 1\ 000$ et le nombre de permutations $P = 1\ 000$. La figure 5.15 représente : a) le graphique de la fonction de vraisemblance, l'estimateur initial $\hat{r_T}$ de r_T et l'intervalle de confiance bootstrap pour $\hat{r_T}$; b) sous-intervalles $IC_{95\%}{}^i$, $i = 1, 2, 3$.



(a) $\hat{r_T}$ et $IC_{95\%}{}^{\hat{r_T}}$ A1.dat

(b) sous-intervalles $IC_{95\%}{}^{\hat{r_T}}$

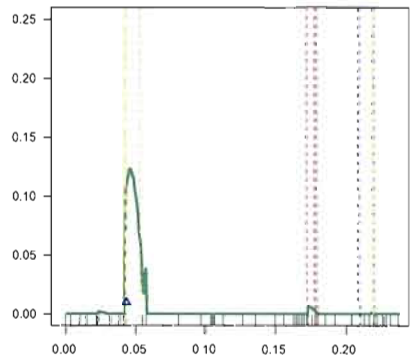
Figure 5.15: D'après l'histogramme des fréquences de ${r_T}^b$, l'intervalle de confiance bootstrap $IC_{95\%}$ se divise en trois sous-intervalles $IC_{95\%}{}^1(0.043; 0.053)$ (en orange), $IC_{95\%}{}^2(0.173; 0.18)$ (en brun), $IC_{95\%}{}^3(0.21; 0.22)$ (en bleu).

Pour tester l'intervalle de confiance bootstrap empirique et les sous-intervalles, les trois probabilités suivantes sont calculées : la probabilité critique globale p^g, pour l'analyse présentée $p^g = 0.18 > 0.05$, ce qui est le premier signe que l'estimateur initial $\hat{r_T}$ n'est pas bon. Une grande probabilité critique globale ne nous donne que pour l'estimateur initial $\hat{r_T}$ l'histoire de la population qui se répète de la même façon, indépendamment de l'état du marqueur à cette position (muté/non muté). C'est-à-dire que l'association haplotype/phénotype n'est pas due à une mutation à la position estimée $\hat{r_T}$ (on accepte H_0). Bien qu'avec la probabilité globale on teste l'estimateur initial, p^g ne nous donne aucune information au sujet de la qualité des estimateurs bootstrap ${\hat{r_T}}^b$. Regardons maintenant le graphique de *la probabilité critique locale* p^l=($p^l_{r_0}$, $p^l_{r_1}$, ..., $p^l_{r_n}$) ; le graphique de la *probabilité critique* notée p_{r_i} et le graphique des fréquences relatives

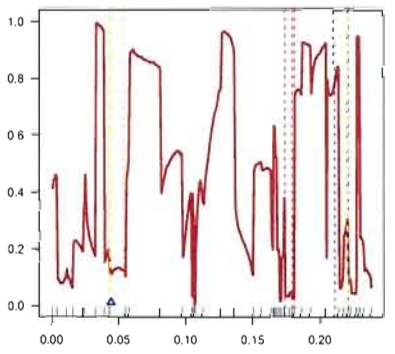
de l'estimateurs $\hat{r_T}^p$ de r_T obtenus en permutant les données initiales (figure 5.16).

La probabilité critique notée p_{r_i} nous donne un test pour les valeurs de la fonction de vraisemblance sous H_0 et H_1 par position, c'est-à-dire la probabilité d'obtenir une valeur plus élevée pour $\hat{L}(r_i)$ quand H_0 est vraie. Comme l'intervalle de confiance est choisi l'intervalle ou $\hat{L}(r_i)_{H_1}$ est un maximum absolu par rapport de $\hat{L}(r_i)_{H_0}$. Quelques graphiques sont illustrés sur les figures 5.17, 5.18, 5.19, 5.20, 5.21.

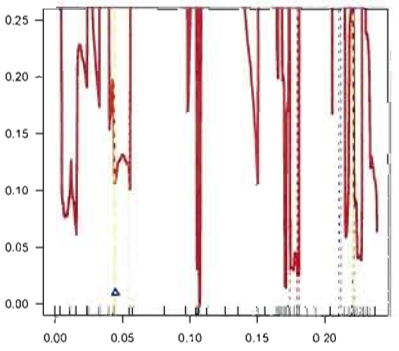
Une façon différente de présenter la probabilité critique notée p_{r_i} est de calculer le 99-éme quantile de la distribution de $\hat{L}(r_i)_{H_0}$, $i = 1, \ldots, k =$ nombre des positions d'évaluation et comparer le graphique avec le graphique pour les données initiales. Quelques exemples sont montrés sur les figures 5.22, 5.23, 5.24, (la ligne verte représente le graphique du 99-éme quantile par position).



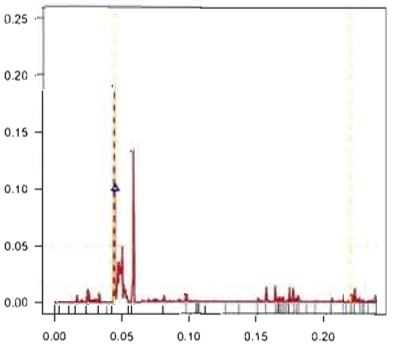
(a) la probabilité critique locale p^l=($p^l_{r_0}$, ..., $p^l_{r_n}$)



(b) la *probabilité critique* p_{r_i}



(c) la *probabilité critique* p_{r_i}, zoom 400%



(d) les fréquences relatives de $r_T{}^p$

Figure 5.16: On voit que dans le premier sous-intervalle (en rouge) les valeurs de la fonction de vraisemblance sous H_0 (permutations) dépassent la valeur initiale maximale de $\hat{L}(r_i)$ avec une grande probabilité ; de plus, on remarque que dans le troisième sous-intervalle, les valeurs initiales $\hat{L}(r_i)$ ne sont pas un maximum absolu. On peut exclure les deux sous-intervalles de l'intervalle de confiance bootstrap pour l'ensemble de données $A1$.

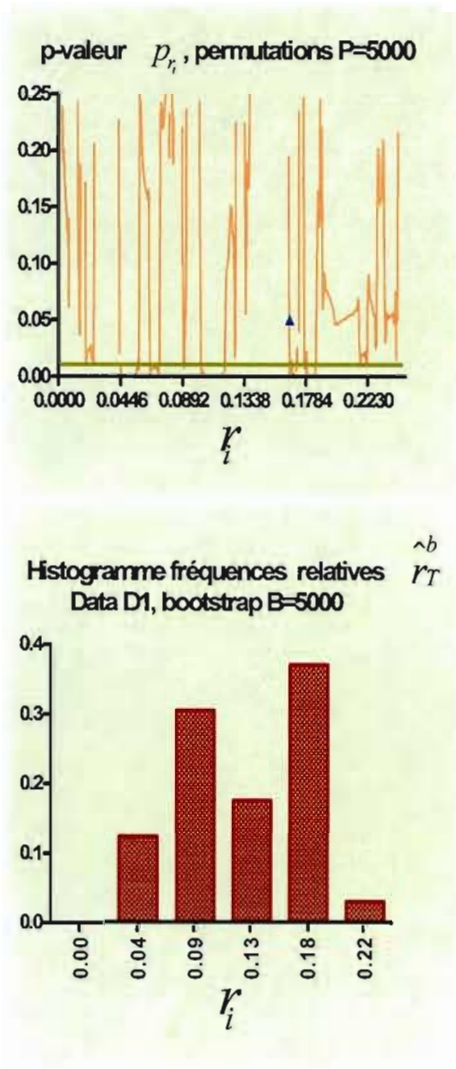


Figure 5.17: On voit que pour un petit intervalle autour de la vraie position de la mutation (le triangle bleu) la p-valeur p_{r_i} est significative. On remarque qu'il existe d'autres positions avec une p-valeur p_{r_i} significative, mais en augmentant le nombre de permutations on obtient un intervalle de confiance bien centré autour de la vraie position de la mutation (le triangle bleu).

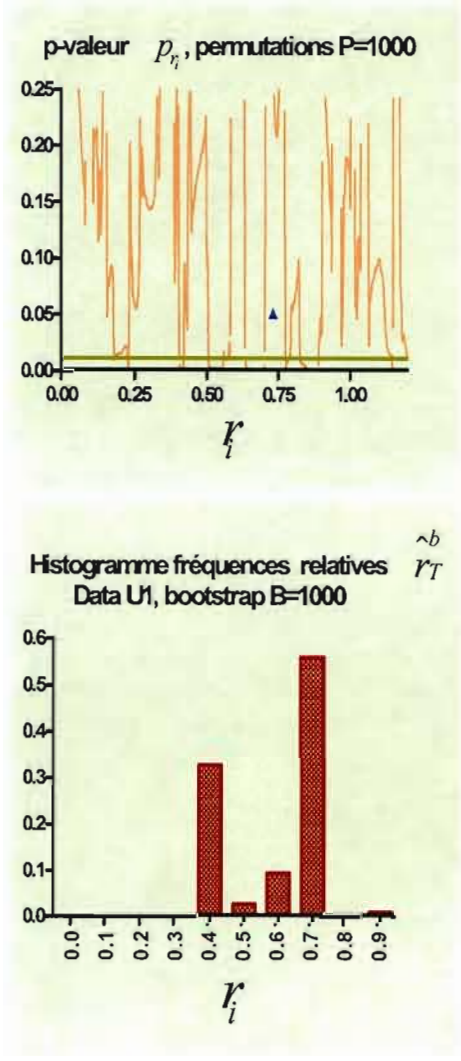


Figure 5.18: On voit que pout un petit intervalle autour de la vraie position de la mutation la p-valeur p_{r_i} est significative. On remarque qu'il existe d'autres positions avec une p-valeur p_{r_i} significative, mais en augmentant le nombre de permutations on obtient un intervalle de confiance bien centré autour de la vraie position de la mutation.

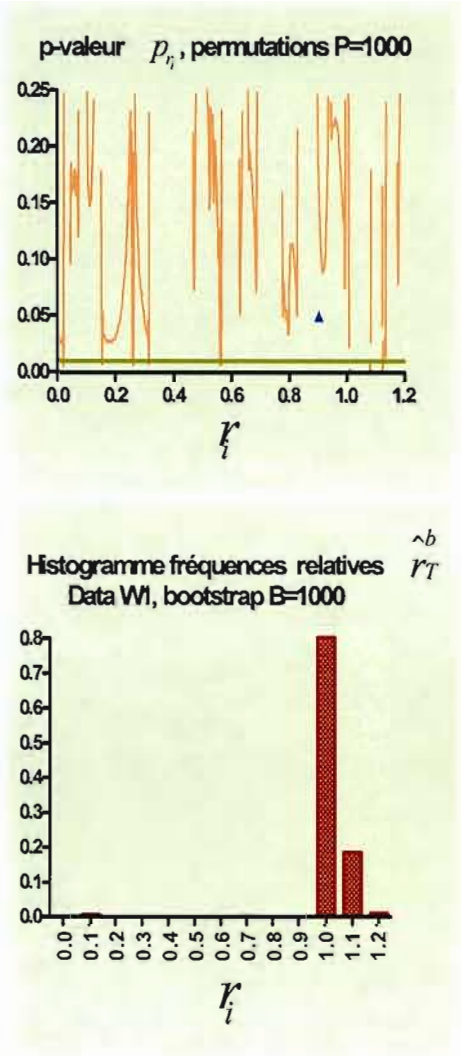


Figure 5.19: On voit que pour un petit intervalle autour de la vraie position de la mutation la *p-valeur* p_{r_i} est significative. On remarque qu'il existe d'autres positions avec une *p-valeur* p_{r_i} significative, mais en augmentant le nombre de permutations on obtient un intervalle de confiance bien centré autour de la vraie position de la mutation.

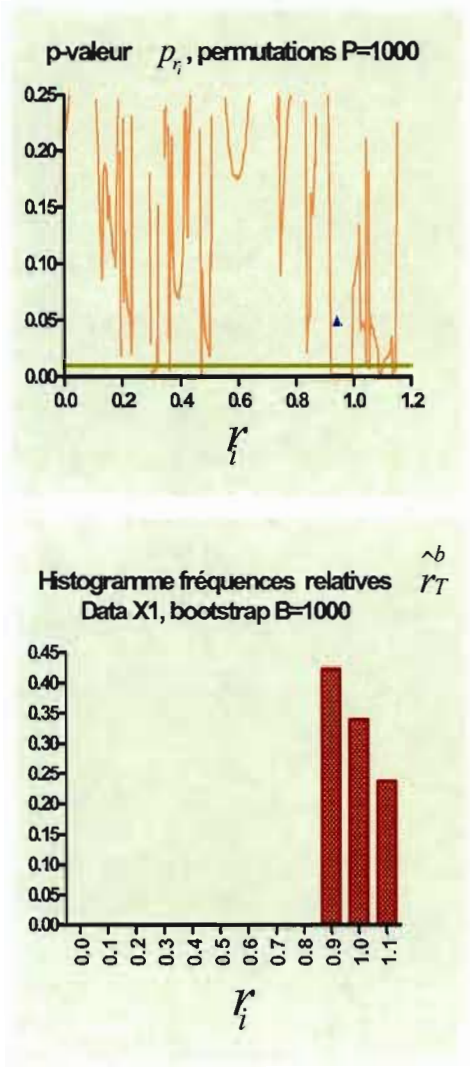


Figure 5.20: On voit que pour un petit intervalle autour de la vraie position de la mutation la p-valeur p_{r_i} est significative. On remarque qu'il existe d'autres positions avec une p-valeur p_{r_i} significative, mais en augmentant le nombre de permutations on obtient un intervalle de confiance bien centré autour de la vraie position de la mutation.

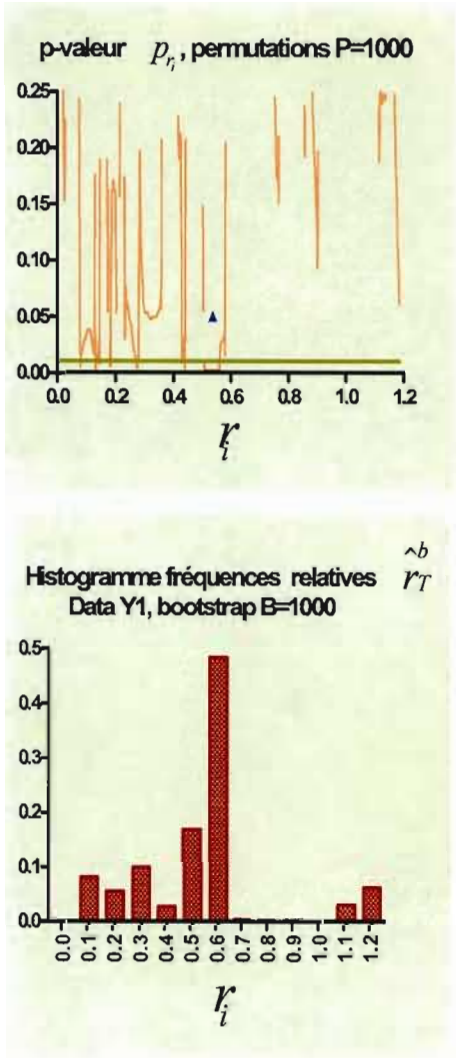


Figure 5.21: On voit que pour un petit intervalle autour de la vraie position de la mutation la p-valeur p_{r_i} est significative. On remarque qu'il existe d'autres positions avec une p-valeur p_{r_i} significative, mais en augmentant le nombre de permutations on obtient un intervalle de confiance bien centré autour de la vraie position de la mutation.

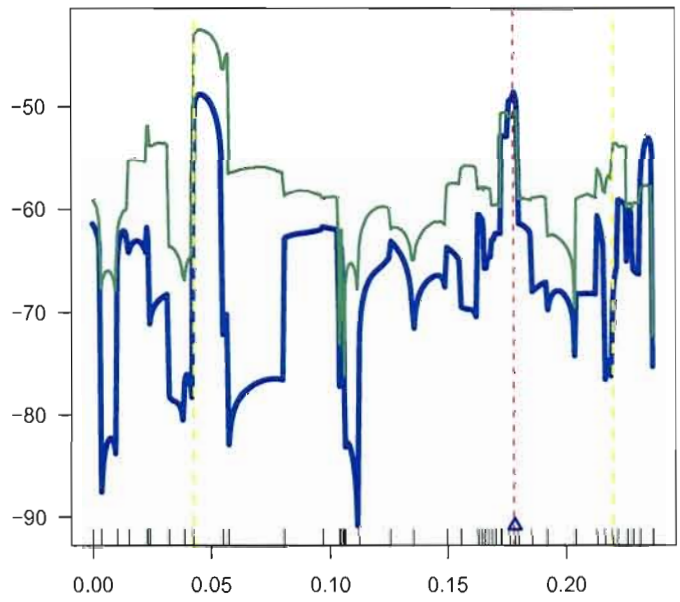


Figure 5.22: Pour l'ensemble de données $A1$, le maximum absolu est atteint pour un intervalle bien centré autour de la vraie position de la mutation (la ligne verte represent le graphique du $99^{\text{-éme}}$ quantile par position).

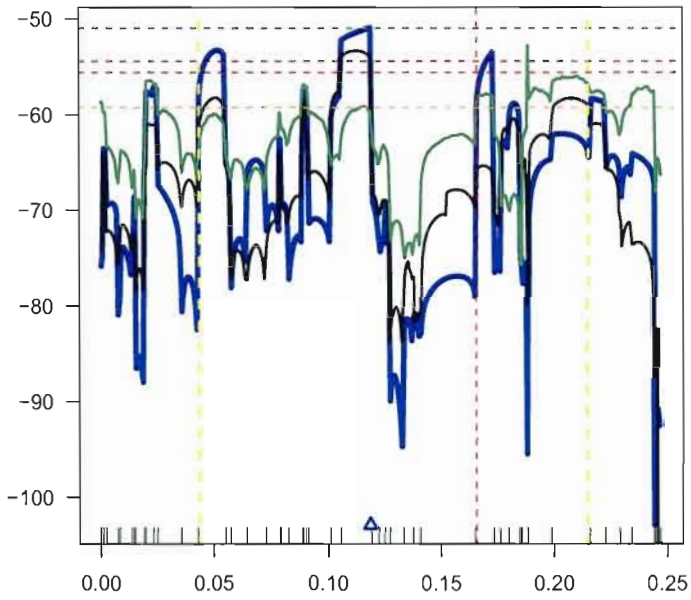


Figure 5.23: Pour l'ensemble de données $D1$, le maximum absolu est atteint pour trois intervalles et l'un d'eux contient la vraie position de la mutation (la ligne verte represent le graphique du $99^{\text{-éme}}$ quantile par position).

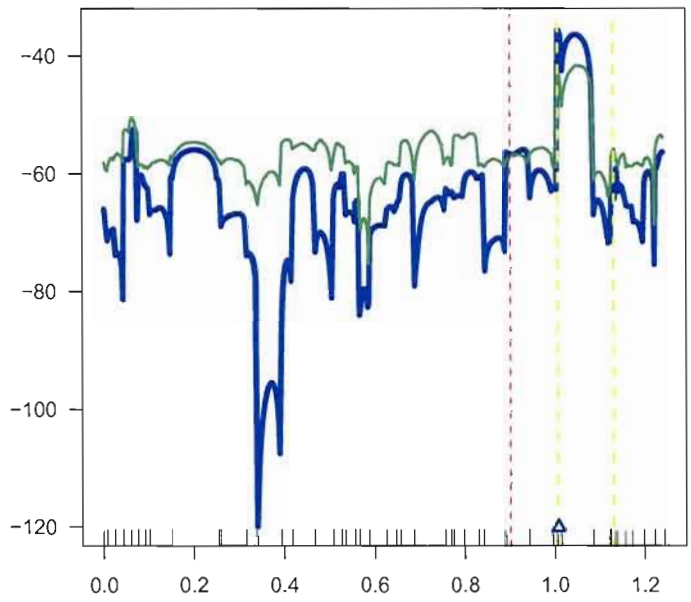


Figure 5.24: Pour l'ensemble de données $W1$, le maximum absolu est atteint pour deux intervalles et l'un d'eux contient la vraie position de la mutation (la ligne verte represent le graphique du $99^{\text{-éme}}$ quantile par position).

Data A1
K100 L 2 bootstrap 5000 permutations 5000

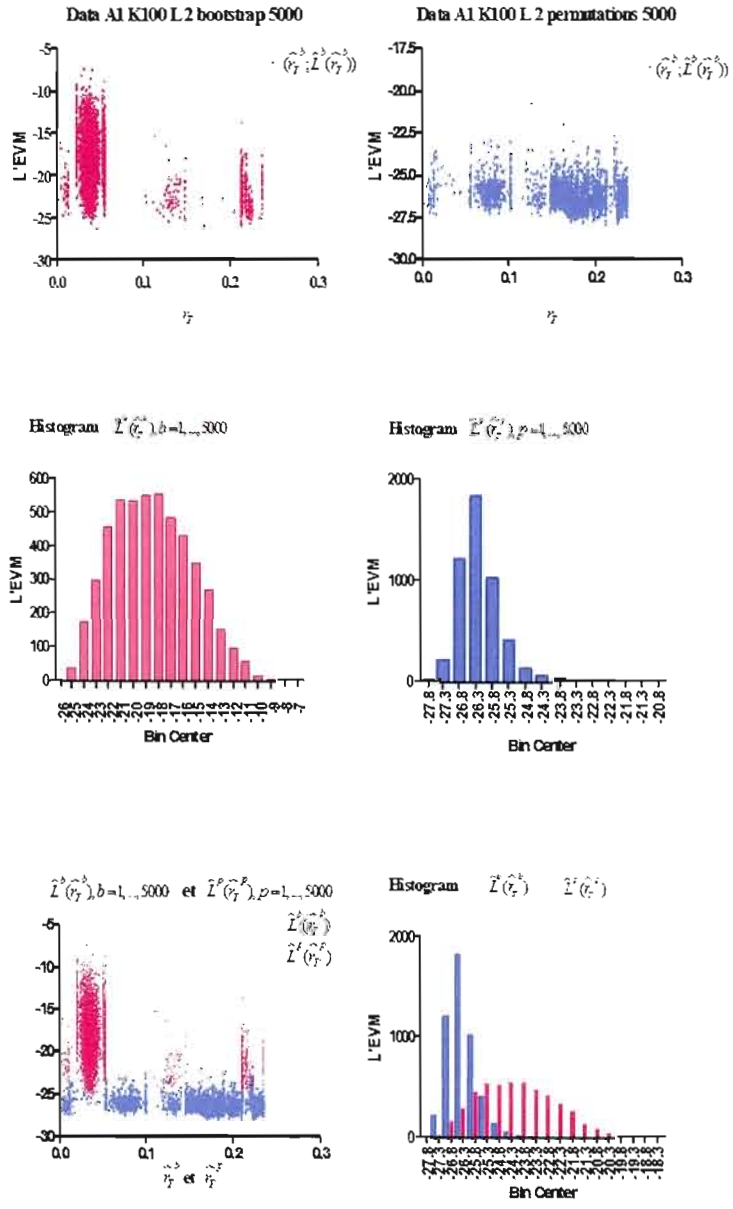


Figure 5.25: *A*1, B=5000, P=5000

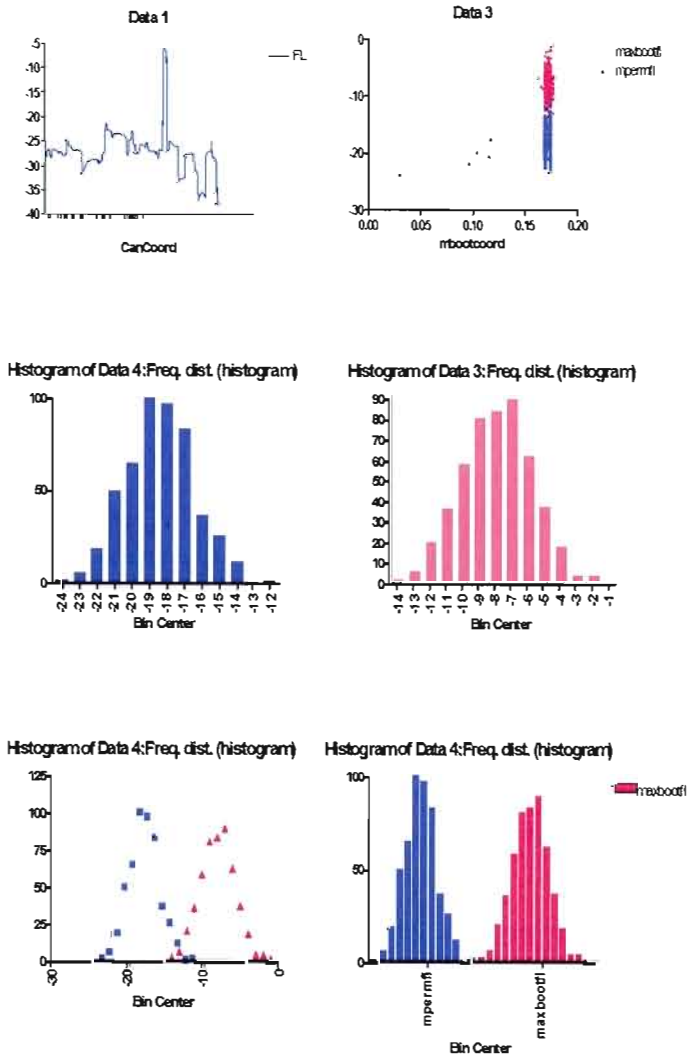


Figure 5.26: $E1$, B=500, P=500

CONCLUSION

L'objectif principal de ce projet était de développer des méthodes pertinentes pour obtenir un intervalle de confiance et tester l'estimateur de la position d'une mutation d'intérêt fourni par la méthode MapArg. Le travail a été divisé en deux parties : une partie théorique et une partie pratique. La partie pratique consiste essentiellement à un travail d'algorithmique et de programmation (C++), et fait suite à ce qui a été développé dans la partie théorique.

La construction d'un intervalle de confiance par la méthode du bootstrap, et le test de permutation nécessitent en général une répétition de la méthode MapArg sur des données «bootstrapées» ou permutées. On a utilisé le code original du programme MapArg comme une base de son extension MapArgBP. Le défi principal était le temps du calcul nécessaire pour un nombre de bootstraps ou de permutations significatif. Par exemple, pour certaines données, le temps du calcul pour 3000 permutations était de 3 mois. D'autre part, les fichiers des résultats numériques des tests performés pendant les 9 derniers mois ont un volume considérable : 25 GB sur l'un d'ordinateurs utilisés et 12 GB sur l'autre. L'estimation par la méthode MapArg pour les petits intervalles semblait instable. L'analyse des graphiques et des résultats numériques nous permettait de trouver la façon correcte de tester l'estimateur obtenu.

Pour réduire le temps du calcul, la méthode *MapArg* utilise des sous-ensembles de séquences génomiques nommées des fenêtres. D'un point de vu mathématique, pour des petites fenêtres, la comparaison des valeurs de la fonction de la vraisemblance (le problème de maximisation de la fonction de la vraisemblance sur la longueur totale de la séquence) n'est pas correcte. Pour cette raison, un test de la distribution des valeurs de la fonction de la vraisemblance par intervalle a été introduit, ce que nous a permis de corriger l'estimateur. Pour des données simulées, nous avons obtenu une très bonne estimation de la position probable de la mutation.

Nous pouvons dire que les problèmes discutés dans cet ouvrage sont complexes et qu'il y

a encore beaucoup de travail à faire afin de donner une réponse positive pour des vraies données. Pour améliorer le temps du calcul, une version parallèle du programme est suggérée.

Appendices

***MapArgBP* (code C++)**

```
// File nat_main.cpp
//
#include "declarations.h"

using namespace std;

int DVMax = 0;
int DVNum = 0;
VecI DVCan;


// Nataliya's parameters
class NatParams
{
public:

    long NumOfBootstraps;
    long NumOfPermutations;
    bool MultiDV;

    string BootResultsFile;             // Bootstrap    File with saved
    results to load
    string PermResultsFile;             // Permutations File with saved
    results to load

    // Clear all values to its default values
    void Clear()
    {
        NumOfBootstraps     = 0;
        NumOfPermutations   = 0;
        BootResultsFile     = "";
        PermResultsFile     = "";
        MultiDV = false;
    }

    // Default Constructor
    NatParams()
    {
        Clear();
    }

};


// Fabrice's Parameters
class FabParams
{
protected:
    int     argc;
    char**  argv;

public:
    VecS    ParList;

    string  ParFileName;
    VecS    ParFileLines;

    string  DatFileName;
```

```
VecS    DatFileLines;

VVecI   HapData;
VVecI   HapDataPerm;

string  ResFileName;

int     BackgroundL;

// Clear all values to its default values
void Clear()
{
    argc = 0;
    argv = NULL;
    ParList.clear();
    BackgroundL = 1;
    ResFileName = "MapArg.txt";
}

//////// main() command line arguments ////////////////////////////////
//////////////
void ResetMainArgs()
{
    if (argv)
    {
        delete argv;
        argv = NULL;
    }
    argc = 0;
}

void BuildMainArgs()
{
    if (ParList.size()!=argc || argv==NULL)
    {
        if (argv) delete argv;
        argc = ParList.size();
        if (argc)
        {
            argv = new char* [argc+2];
            for(VecSIter i = ParList.begin(); i!=ParList.end(); i++)
                argv[i-ParList.begin()] = (char*)i->c_str();
        }
        argv[argc] = "-o";
        argv[argc+1] = (char*)ResFileName.c_str();
        argc += 2;
    }
}

// Get parameters of fabrice's main()
void GetMainArgs(int &ArgC, char** &ArgV)
{
    BuildMainArgs();
    ArgC = argc;
    ArgV = argv;
}

// Adds a command line parameter
```

```
    void AddCmdLineParam(string Str)
    {
        ParList.push_back(Str);
        ResetMainArgs();
    }

    //
    void SetResFileName(string Str)
    {
        ResFileName = Str;
    }

    // Adds a command line parameter
    void AddCmdLineParams(VecS Par, int index = 0)
    {
        for(int i=index; i<Par.size(); i++) ParList.push_back(Par[i]);
        ResetMainArgs();
    }

    // Default Constructor
    FabParams()
    {
        Clear();
    }

    ~FabParams()
    {
        if (argv) delete argv;
    }

};


// Fabrice's Parameters
class FabResults
{
public:

    VVecE OrigMatLik;               // Original Data - MatLik
    VecE  OrigFinalLik;             // Original Data - FinalLik
    VecE  AvgFinalLik;              // Average of all FinalLiks (Bootstrap↙
     only)
    VecE  PermAvgFinalLik;          // Average of all FinalLiks          ↙
    (Permutations only)
    VecE  CorrOrigFinalLik;         // Original Data - FinalLik
    VecE  LastCorrFinalLik;         // Last Corrected - FinalLik
    VecD  OrigCoord;                // Original Data - Coord
    VecD  OrigCanCoord;             // Original Data - CanCoord
    VecI  OrigCanInterval;          // Original Data - CanInterval
    double OrigMaxCoord;            //
    double AvgMaxCoord;             //
    double PermAvgMaxCoord;         //
    double CorrOrigMaxCoord;        //
    int   OrigMaxInterval;          //
    int   CorrOrigMaxInterval;      //
    EXTNUM OrigMaxFLik;             //

    VecD  PermDistCoord;            // Permutations - Distribution
```

```
VecD  BootCoord;                  // Bootstrap - one item per call - the↙
 coord of the max value in FinalLik
VecE  BootMaxFL;                  // Bootstrap - the max value in       ↙
FinalLik of the coresponding BootCoord
VecI  BMFLInd;                    //
VecI  BMFLCoord;                  //
EXTNUM BootValue5;                // Bootstrap - the 5% level - value of↙
 BootMaxFL
EXTNUM PermValue63;               // Permutations - the 63% level of all↙
 values in all FinalLik
EXTNUM PermValue95;               // Permutations - the 95% level of all↙
 values in all FinalLik
EXTNUM PermValue99;               // Permutations - the 99% level of all↙
 values in all FinalLik
VecD  BootConfInterval;           // Bootstrap -
EXTNUM ErrorType;                 // Bootstrap - ErrorType
EXTNUM Average;                   // Bootstrap - Average
EXTNUM Biais;                     // Bootstrap - Biais
EXTNUM ICVMin;                    // Bootstrap - ICVMin
EXTNUM ICVMax;                    // Bootstrap - ICVMin
EXTNUM AChapeau;                  // Bootstrap - Â (A chapeau)
int    LTONum;                    // Bootstrap - Less Than Original -   ↙
number
double LTOCoef;                   // Bootstrap - Less Than Original -   ↙
Coef in [0,1]
VecD  PermCoord;                  //
VecE  PermMaxFL;                  //


VecD  PermPValue;                 // Permutations - PValue
VecD  PermPValue2;                // Permutations - PValue

VecD  MutDist;                    // Mutation Distribution

VVecE Levels;
VecE Level99, Level95, Level63;

VVecE DVFinalLik;                 //
VecE  DVAverage;                  //
VecE  DVBest;                     //
VecI  DVCount;                    //

// Clear all values to its default values
void Clear()
{
    OrigMatLik.clear();
    OrigFinalLik.clear();
    OrigCoord.clear();
    OrigCanCoord.clear();
    OrigMaxCoord = 0;
    BootCoord.clear();
    PermPValue.clear();
    BootConfInterval.clear();
    LastCorrFinalLik.clear();
    OrigMaxInterval = 0;
    CorrOrigMaxInterval = 0;
    MutDist.clear();
```

```
    }

    // Default Constructor
    FabResults()
    {
        Clear();
    }

};

// Fabrice's Runtime Data
class FabRuntime
{
public:

    VecS    ParFileLines;
    VecS    DatFileLines;

    int     ParPos;
    int     DatPos;

    // Clear all values to its default values
    void Clear()
    {
        ParPos = 0;
        DatPos = 0;
    }

    //
    void Reset()
    {
        ParPos = 0;
        DatPos = 0;
    }

    // Default Constructor
    FabRuntime()
    {
        Clear();
    }

    bool GetParLine(string &Line)
    {
        if (ParPos>=0 && ParPos<ParFileLines.size())
        {
            Line = ParFileLines[ParPos++];
            return true;
        }
        return false;
    }

    bool GetDatLine(string &Line)
    {
        if (DatPos>=0 && DatPos<DatFileLines.size())
        {
            Line = DatFileLines[DatPos++];
            return true;
        }
```

```
        return false;
    }

};


static NatParams  * NPar = NULL;
static FabParams  * FPar = NULL;
static FabRuntime * RT   = NULL;
static FabResults   FRes;

void WriteGnuPlotNat(NatParams &NPar, FabParams &FPar, FabResults &FRes);
void OutputTeXNat(NatParams &NPar, FabParams &FPar, FabResults &FRes);

bool GetParLine(string &Line)
{
    bool res = RT? RT->GetParLine(Line) : false;
    // cout << "Par: " << Line << endl;
    return res;
}


bool GetDatLine(string &Line)
{
    return RT? RT->GetDatLine(Line) : false;
}


long ParseNum(string s)
{
    return atoi(s.c_str());
}


bool IsParam(char *par)
{
    return par && *par=='-';
}


char GetNatParam(const char *par)
{
    if (par)
    {
        int len = strlen(par);
        if (len==3 && par[0]=='-' && par[1]=='-') return par[2];
    }
    return 0;
}


char GetFabParam(const char *par)
{
    if (par)
    {
        int len = strlen(par);
        if (len>1 && par[0]=='-') return par[1];
    }
```

```
    return 0;
}


int  GetParamLen(char *par)
{
    int len = 1000;
    char Ch = GetNatParam(par);

    if (Ch) len = 2;

    return len;
}

// bool FindMaxPosDump=0;

int  FindMaxPos(VecE V)
{
    int     Pos = -1;
    EXTNUM  Max = 0;

    // if ( FindMaxPosDump ) cout << "Pos: " << Pos << ", Max: " << Max <<↙
     endl;
    for(int k=0; k<V.size();k++)
    {
        // if ( FindMaxPosDump ) cout << k << " --- Pos: " << Pos << ",    ↙
    Max: " << Max << " V[k]: " << V[k] << endl;
        if ( (Max - V[k] < (EXTNUM)0.0) || (Pos<0))
        {
            Pos = k;
            Max = V[k];
        }
    }
    // if ( FindMaxPosDump ) cout << "Pos: " << Pos << ", Max: " << Max <<↙
     endl;
    return Pos;
}


bool HandleParam(NatParams &NPar, FabParams &FPar, VecS &Par)
{
    if (Par.size()<1) return true;

    char Ch = GetNatParam(Par[0].c_str());

    // --b <num> or --B <num> - number of bootstraps
    // --p <num> or --P <num> - number of permutations
    // --m or --M - Multiple Driving Values Method (by Nataliya Dragieva)

    switch (Ch)
    {
    case 'm':
    case 'M':
        NPar.MultiDV = true;
        break;

    case 'b':
    case 'B':
```

```
    if (Par.size()>1)
    {
        NPar.NumOfBootstraps = ParseNum(Par[1]);
        FPar.AddCmdLineParams(Par, 2);
        if (NPar.NumOfBootstraps>0) break;
    }
    cerr << "Bad Command Line Syntax:  --" << Ch << " must be followed↙
 by a positive number" << endl;
    return false;

case 'p':
case 'P':
    if (Par.size()>1)
    {
        NPar.NumOfPermutations = ParseNum(Par[1]);
        FPar.AddCmdLineParams(Par, 2);
        if (NPar.NumOfPermutations>0) break;
    }
    cerr << "Bad Command Line Syntax:  --" << Ch << " must be followed↙
 by a positive number" << endl;
    return false;

default:
    if ( Ch )
    {
        cerr << "Bad Command Line Syntax:  --" << Ch << " is not a    ↙
valid parameter. It's been ignored!" << endl;
        FPar.AddCmdLineParams(Par, 1);
    }
    else
    {
        Ch = GetFabParam(Par[0].c_str());
        switch (Ch)
        {
        case 'p': // Parameter File Name
            if (Par.size()>1)
                FPar.ParFileName = Par[1];
            break;
        case 'y': // Data File Name
            if (Par.size()>1)
                FPar.DatFileName = Par[1];
            break;
        case 'o': // Output File Name
            if (Par.size()>1)
                FPar.ResFileName = Par[1];
            break;
        case 'B': // BackgroundL
            if (Par.size()>1)
                FPar.BackgroundL = ParseNum(Par[1]);
            break;
        }
        FPar.AddCmdLineParams(Par);
    }
    break;
}

return true;
```

```
}

bool ReadParams(NatParams &NPar, FabParams &FPar, int argc, char *argv[])
{

    VecS Par;
    int  Len = 0;

    for(int i=0; i<argc; i++)
    {
        if (IsParam(argv[i]))
        {
            if (Par.size()>0)
            {
                if (! HandleParam(NPar, FPar, Par) ) return false;
                Par.clear();
            }
            Len = GetParamLen(argv[i]);
        } // if
        Par.push_back(argv[i]);
        if (Len>0)
        {
            if (--Len==0 && Par.size()>0)
            {
                if (! HandleParam(NPar, FPar, Par) ) return false;
                Par.clear();
            }
        } // if
        else Len = 0;
    } // for

    if (Par.size()>0)
    {
        if (! HandleParam(NPar, FPar, Par) ) return false;
    }

    // Read The Par File
    if ( ! FPar.ParFileName.empty())
    {
        string Line;
        VecS   Arr;

        ifstream In(FPar.ParFileName.c_str());
        if (!In )
        {
            cerr << "ERROR: Can't open \'" << FPar.ParFileName <<"\' for ↙
    reading." << endl;
            return false;
        }
        while (getline(In, Line)) FPar.ParFileLines.push_back(Line);
        In.close();

        if (FPar.DatFileName.empty())
        {
            for (VecSIter i=FPar.ParFileLines.begin(); i!=FPar.          ↙
    ParFileLines.end(); i++)
            {
```

```
            Split(*i, Arr);
            if (Arr.size()>1 && Arr[0]=="DataFile")
            {
                FPar.DatFileName = Arr[1];
                break;
            }
        }
    }
}


// Read The Dat File
if ( ! FPar.DatFileName.empty())
{
    string Line;
    VecS   Arr;

    ifstream In(FPar.DatFileName.c_str());
    if (!In )
    {
        cerr << "ERROR: Can't open \'" << FPar.DatFileName <<"\' for ↙
reading." << endl;
        return false;
    }
    while (getline(In, Line)) FPar.DatFileLines.push_back(Trim(Line));
    In.close();

    FPar.HapData.clear();
    VecI Row;

    for (VecSIter i=FPar.DatFileLines.begin(); i!=FPar.DatFileLines. ↙
end(); i++)
    {
        cout << "Reading: " << *i << endl;
        Split(*i, Arr);
        Row.clear();
        for (VecSIter j=Arr.begin(); j!=Arr.end(); j++)
        {
            if (*j=="0") Row.push_back(0);
            else if (*j=="1") Row.push_back(1);
            else if (*j=="-") Row.push_back(-1);
            else if ( ! j->empty()) Row.push_back(ParseNum(*j));
        }
        if (Row.size()>1)
        {
            VecI RCopy;
            RCopy.assign(Row.begin(), Row.end());
            FPar.HapDataPerm.push_back(RCopy);
            if (Row[0]>1)
            {
                for(int k=0; k<Row[0]; k++)
                {
                    VecI RowCopy;
                    RowCopy.assign(Row.begin(), Row.end());
                    RowCopy[0] = 1;
                    FPar.HapData.push_back(RowCopy);
                }
            }
```

```
                else
                    FPar.HapData.push_back(Row);
            }
        }
    }

    return true;
}


bool Init(NatParams &NPar, FabParams &FPar, FabRuntime &RT, FabResults &
    FRes)
{
    // test print
    cout << "NumOfBootstraps: " << NPar.NumOfBootstraps << endl;
    cout << "NumOfPermutations: " << NPar.NumOfPermutations << endl;
    cout << "Par File Name: " << FPar.ParFileName << endl;
    cout << "Dat File Name: " << FPar.DatFileName << endl;
    cout << "Res File Name: " << FPar.ResFileName << endl;
    cout << "Fabrice Command Line: " << Join(FPar.ParList) << endl;

    cout << endl << endl;
    cout << "Par File Content: " << endl;
    cout << Join(FPar.ParFileLines, "\n") << endl;

    cout << endl << endl;
    cout << "Dat File Content: " << endl;
    cout << Join(FPar.DatFileLines, "\n") << endl;

    cout << endl << endl;
    cout << "HapData: " << endl;
    for (VVecI::iterator i=FPar.HapData.begin(); i!=FPar.HapData.end(); i+
    +)
    {
        cout << Join(*i, " ") << endl;
    }

    return true;
}

string OFName;

bool CallMapArg(NatParams &NPar, FabParams &FPar, FabRuntime &RT,
    FabResults &FRes)
{
    int argc = 0;
    char** argv = NULL;


    FPar.SetResFileName(OFName + ".txt");
    FPar.GetMainArgs(argc, argv);
    ClearMapArg();
    MapArgMain(argc, argv);
    if (L_I == Lw_I)
        FPar.BackgroundL = 0;

    int k, N;
```

```
    if (FPar.BackgroundL==0)
    {
        N = MatLik.size();
        if (N>1)
        {
            MTRand MTR;
            VecI   I1;          // holds indexes in HapData
            VecI   I2;          // holds indexes in HapData
            VecI   I3;          // holds indexes in HapData

            for(k=0; k<N; k++) I1.push_back(MTR.randInt(N-1));
            for(k=0; k<N; k++) I2.push_back(I1[MTR.randInt(N-1)]);
            for(k=0; k<N; k++) I3.push_back(I2[MTR.randInt(N-1)]);

cout << endl;
cout << "I1: " << endl;
cout << Join(I1, " ") << endl;
cout << endl;
cout << "I2: " << endl;
cout << Join(I2, " ") << endl;
cout << endl;
cout << "I3: " << endl;
cout << Join(I3, " ") << endl;

            int L = FinalLik.size();
            cout << "FinalLik Correction:" << endl;

            FRes.LastCorrFinalLik.clear();

            EXTNUM t0, t1, t2, t3;
            for(k=0; k<L; k++)
            {
                cout << k << ": " << FinalLik[k] << "\t\t=>\t";

                t1 = 0.0;
                t2 = 0.0;
                t3 = 0.0;
                for(int i=0; i<N; i++)
                {
                    t1 += MatLik[I1[i]][k];
                    t2 += MatLik[I2[i]][k];
                    t3 += MatLik[I3[i]][k];
                }
                t1 /= N;
                t2 /= N;
                t3 /= N;
                FRes.LastCorrFinalLik.push_back(abs(4.0*FinalLik[k] - 6.0*↙
    t1 + 4.0*t2 - t3));
                cout << FRes.LastCorrFinalLik[k] << endl;
            }

        }
        else
        {
            FRes.LastCorrFinalLik.assign(FinalLik.begin(), FinalLik.end())↙
    ;
        }
    }
```

```
    else // FPar.BackgroundL == 1
    {
        N = MatLikC.size();
        if (N>1)
        {
            MTRand MTR;
            VecI   I1;          // holds indexes in HapData
            VecI   I2;          // holds indexes in HapData
            VecI   I3;          // holds indexes in HapData

            for(k=0; k<N; k++) I1.push_back(MTR.randInt(N-1));
            for(k=0; k<N; k++) I2.push_back(I1[MTR.randInt(N-1)]);
            for(k=0; k<N; k++) I3.push_back(I2[MTR.randInt(N-1)]);

cout << endl;
cout << "I1: " << endl;
cout << Join(I1, " ") << endl;
cout << endl;
cout << "I2: " << endl;
cout << Join(I2, " ") << endl;
cout << endl;
cout << "I3: " << endl;
cout << Join(I3, " ") << endl;

            int L = FinalLikC.size();
            cout << "FinalLikC Correction:" << endl;

            FRes.LastCorrFinalLik.clear();

            EXTNUM t0, t1, t2, t3;
            for(k=0; k<L; k++)
            {
                cout << k << ": " << FinalLikC[k] << "\t\t=>\t";

                t1 = 0.0;
                t2 = 0.0;
                t3 = 0.0;
                for(int i=0; i<N; i++)
                {
                    t1 += MatLikC[I1[i]][k];
                    t2 += MatLikC[I2[i]][k];
                    t3 += MatLikC[I3[i]][k];
                }
                t1 /= N;
                t2 /= N;
                t3 /= N;
                FRes.LastCorrFinalLik.push_back(abs(4.0*FinalLikC[k] - 6.0↙
    *t1 + 4.0*t2 - t3));
                cout << FRes.LastCorrFinalLik[k] << endl;
            }

        }
        else
        {
            FRes.LastCorrFinalLik.assign(FinalLikC.begin(), FinalLikC.end ↙
    ());
        }
    }
```

```
    return true;
}


bool CallMapArgWithOrigData(NatParams &NPar, FabParams &FPar, FabRuntime &↙
    RT, FabResults &FRes)
{

    RT.ParFileLines.assign(FPar.ParFileLines.begin(), FPar.ParFileLines.  ↙
    end());
    RT.DatFileLines.assign(FPar.DatFileLines.begin(), FPar.DatFileLines.  ↙
    end());

cout << endl << endl;
cout << "Calling Fabrice with the original data: " << endl;

    OFName = "OrigData";

    CallMapArg(NPar, FPar, RT, FRes);

//cerr << "FinalLik.size(): " << FinalLik.size() << endl;
//cerr << "FinalLikC.size(): " << FinalLikC.size() << endl;

    if (FPar.BackgroundL==0)
    {
        FRes.OrigMatLik.assign(MatLik.begin(), MatLik.end());
        FRes.OrigFinalLik.assign(FinalLik.begin(), FinalLik.end());
    }
    else
    {
        FRes.OrigMatLik.assign(MatLikC.begin(), MatLikC.end());
        FRes.OrigFinalLik.assign(FinalLikC.begin(), FinalLikC.end());
    }
//  int t;
//  for (t=0; t<Coord.size(); t++) Coord[t] *= AdjustScale;
//  for (t=0; t<CanCoord.size(); t++) CanCoord[t] *= AdjustScale;

    FRes.OrigCoord.assign(Coord.begin(), Coord.end());
    FRes.OrigCanCoord.assign(CanCoord.begin(), CanCoord.end());
    FRes.OrigCanInterval.assign(CanInterval.begin(), CanInterval.end());
    FRes.CorrOrigFinalLik.assign(FRes.LastCorrFinalLik.begin(), FRes.     ↙
    LastCorrFinalLik.end());

    int Pos = FindMaxPos(FRes.OrigFinalLik);
    if (Pos>0) FRes.OrigMaxFLik = FRes.OrigFinalLik[Pos];
    if (Pos>=0 && Pos<FRes.OrigCanCoord.size())
    {
        FRes.OrigMaxCoord = FRes.OrigCanCoord[Pos];
        FRes.OrigMaxInterval = FRes.OrigCanInterval[Pos];
    }

    int CorrPos = FindMaxPos(FRes.CorrOrigFinalLik);
    if (CorrPos>=0 && CorrPos<FRes.OrigCanCoord.size())
    {
        FRes.CorrOrigMaxCoord = FRes.OrigCanCoord[CorrPos];
        FRes.CorrOrigMaxInterval = FRes.OrigCanInterval[CorrPos];
```

```
    }

cout << endl << endl;
cout << "FinalLik (Original Data): " << endl;
cout << Join(FRes.OrigFinalLik, " ") << endl;
cout << endl << endl;
if (FinalLik.size()>0)
{
    cout << "FinalLik Maximum coord: " << FRes.OrigMaxCoord << ", Maximum ↙
    FL: " << FRes.OrigMaxFLik << ",   @index: " << Pos << ", Interval: " <↙
    < FRes.OrigMaxInterval << endl;
    cout << endl << endl;
}
cout << "Corrected FinalLik (Original Data): " << endl;
cout << Join(FRes.LastCorrFinalLik, " ") << endl;
cout << endl << endl;
if (FRes.LastCorrFinalLik.size()>0)
{
    cout << "Corrected FinalLik Maximum coord: " << FRes.CorrOrigMaxCoord ↙
    << ", @index: " << CorrPos << ", Interval: " << FRes.                ↙
    CorrOrigMaxInterval << endl;
    cout << endl;
}
cout << "Coord (Original Data): " << endl;
cout << Join(Coord, " ") << endl;
cout << endl;
cout << "CanCoord (Original Data): " << endl;
cout << Join(CanCoord, " ") << endl;

    return true;
}



bool DoMultiDV(NatParams &NPar, FabParams &FPar, FabRuntime &RT,          ↙
    FabResults &FRes)
{
    int i;

    DVNum = 1;
    DVMax = 1;

    do
    {

        int argc = 0;
        char** argv = NULL;

        RT.Reset();
        RT.ParFileLines.assign(FPar.ParFileLines.begin(), FPar.           ↙
    ParFileLines.end());
        RT.DatFileLines.assign(FPar.DatFileLines.begin(), FPar.           ↙
    DatFileLines.end());

        FPar.GetMainArgs(argc, argv);
        ClearMapArg();
        MapArgMain(argc, argv);
        if (L_I == Lw_I)
```

```
        FPar.BackgroundL = 0;

    if (FPar.BackgroundL==0)
    {
        if (FRes.DVBest.size()==0)
        {
            for(i=0; i<FinalLik.size(); i++) FRes.DVBest.push_back(0);
        }
        if (FRes.DVCount.size()==0)
        {
            for(i=0; i<FinalLik.size(); i++) FRes.DVCount.push_back(0)↙
;
        }

        FRes.DVFinalLik.push_back(FinalLik);

        for(i=0; i<DVCan.size(); i++)
        {
            FRes.DVCount[DVCan[i]-1]++;
            FRes.DVBest [DVCan[i]-1] += FinalLik[DVCan[i]-1];
        }
    }
    else
    {
        if (FRes.DVBest.size()==0)
        {
            for(i=0; i<FinalLikC.size(); i++) FRes.DVBest.push_back(0)↙
;
        }
        if (FRes.DVCount.size()==0)
        {
            for(i=0; i<FinalLikC.size(); i++) FRes.DVCount.push_back ↙
(0);
        }

        FRes.DVFinalLik.push_back(FinalLikC);

        for(i=0; i<DVCan.size(); i++)
        {
            FRes.DVCount[DVCan[i]-1]++;
            FRes.DVBest [DVCan[i]-1] += FinalLikC[DVCan[i]-1];
        }

    }


} while(++DVNum<=DVMax);

if (debug)
{
    PrintV(FRes.DVBest, "DVBest - SUM");
    PrintV(FRes.DVCount, "DVCount");
}

for(i=0; i<FRes.DVBest.size(); i++)
{
    if (FRes.DVCount[i]!=0) FRes.DVBest[i] /= FRes.DVCount[i];
}
```

```
    int j;
    for(i=0; i<FRes.DVFinalLik.size(); i++)
    {
        if (i==0)
            for(j=0; j<FRes.DVFinalLik[0].size(); j++) FRes.DVAverage.
    push_back(FRes.DVFinalLik[0][j]);
        else
            for(j=0; j<FRes.DVFinalLik[i].size(); j++) FRes.DVAverage[j] +
    = FRes.DVFinalLik[i][j];

    }
    for(j=0; j<FRes.DVAverage.size(); j++) FRes.DVAverage[j] /= i;  // i
    is = FRes.DVFinalLik.size()

    if (debug)
    {
        PrintV(FRes.DVBest, "DVBest");
        PrintV(FRes.DVAverage, "DVAverage");
    }

    return true;
}


bool SortHelper ( const int i1, const int i2 )
{
    return FRes.BootMaxFL[i1] < FRes.BootMaxFL[i2];
}


bool DoBootstrap(NatParams &NPar, FabParams &FPar, FabRuntime &RT,
    FabResults &FRes)
{
    int N = FPar.HapData.size();
    int p;
    int FLCount = 1;

    FRes.AvgFinalLik.assign(FRes.OrigFinalLik.begin(), FRes.OrigFinalLik.
    end());

    for(p=0; p<FRes.AvgFinalLik.size(); p++)
    {
        FRes.AvgFinalLik[p] = log(FRes.AvgFinalLik[p]);
    }


    if (NPar.NumOfBootstraps && N>1)
    {
        MTRand MTR;
        VecI Sick;          // holds indexes in HapData
        VecI Healthy;       // holds indexes in HapData

        VVecI::iterator begin = FPar.HapData.begin(), end = FPar.HapData.
    end(), it;

        for(it=begin; it!=end; it++)
        {
```

```
            if(it->size()>1)
            {
                if ((*it)[1]==1)
                    Sick.push_back(it-begin);
                else if ((*it)[1]==0)
                    Healthy.push_back(it-begin);
            }
        }
        int SNum = Sick.size();      // number of sick
        int HNum = Healthy.size();   // number of helthy

        if (SNum<1 || HNum<1 || N<3)
        {
            cerr << "Found " << SNum << " sick and " << HNum << " healthy ↙
    of total " << N << endl;
            cerr << "Sorry, Can't do bootstrap test!" << endl;
        }
        else
            cout << "Found " << SNum << " sick and " << HNum << " healthy ↙
    of total " << N << endl;


        RT.ParFileLines.assign(FPar.ParFileLines.begin(), FPar.          ↙
    ParFileLines.end());
        for(long i=0; i<NPar.NumOfBootstraps; i++)
        {
            RT.Reset();
            RT.DatFileLines.clear();
            for(int k=0; k<N; k++)
            {
                int rnd = FPar.HapData[k%N][1]? Healthy[MTR.randInt(HNum- ↙
    1)] : Sick[MTR.randInt(SNum-1)];
                string Line = Join(FPar.HapData[rnd]);
                RT.DatFileLines.push_back(Line);
            }

cout << endl << endl;
cout << "Selected Bootstrap (#" << (i+1) << "): " << endl;
cout << Join(RT.DatFileLines, "\n") << endl;

            char print_buf[100];
            sprintf(print_buf, "Boot%03ld", i+1);

            OFName = print_buf;

            CallMapArg(NPar, FPar, RT, FRes);

            int Pos = FindMaxPos(FRes.LastCorrFinalLik);
            if (Pos>=0 && Pos<FRes.OrigCanCoord.size())
            {
                FRes.BootCoord.push_back(FRes.OrigCanCoord[Pos]);
                FRes.BootMaxFL.push_back(FRes.LastCorrFinalLik[Pos]);
                FRes.BMFLInd.push_back((int)FRes.BMFLInd.size());
            }

            for(p=0; p<FRes.AvgFinalLik.size(); p++)
            {
                FRes.AvgFinalLik[p] += log(FRes.LastCorrFinalLik[p]);
```

```
            }
            FLCount++;
        }

cout << endl << endl;
cout << "BootCoord: " << endl;
cout << Join(FRes.BootCoord, " ") << endl;

cout << "BootMaxFL: " << endl;
cout << Join(FRes.BootMaxFL, " ") << endl;
cout << endl << endl;

        sort(FRes.BMFLInd.begin(), FRes.BMFLInd.end(), SortHelper);

        int ind5 = (int)(0.05*FRes.BootMaxFL.size());
        FRes.BootValue5 = FRes.BootMaxFL[FRes.BMFLInd[ind5]];

cout << "Level FL 95%, index=" << ind5 << ": " << FRes.BootValue5 << endl;
int x;

cout << endl << endl;
cout << "BootMaxFL (sorted): " << endl;
for(x=0; x<FRes.BMFLInd.size(); x++)
{
    if (x) cout << " ";
    cout << FRes.BootMaxFL[FRes.BMFLInd[x]];
}
cout << endl;


cout << "BootMaxFL Coords (sorted): " << endl;
for(x=0; x<FRes.BMFLInd.size(); x++)
{
    if (x) cout << " ";
    cout << FRes.BootCoord[FRes.BMFLInd[x]];
}
cout << endl << endl << endl;


        sort(FRes.BootCoord.begin(), FRes.BootCoord.end());


//cout << "BMFLInd after: " << endl;
//cout << Join(FRes.BMFLInd, " ") << endl;

        cout  << endl << endl << "Bootstrap Result:" << endl;
        if (FRes.BootCoord.size()>0)
        {

cout << "Interval 100%:" << endl;
cout << Join(FRes.BootCoord) << endl;

            if (FRes.OrigMaxCoord>FRes.BootCoord.front() && FRes.
    OrigMaxCoord<FRes.BootCoord.back())
            {
                int cut = (int)(0.025*FRes.BootCoord.size());
                FRes.BootConfInterval.assign(FRes.BootCoord.begin()+cut,
    FRes.BootCoord.end()-cut);
```

```
                for(p=0; p<FRes.BootConfInterval.size(); p++)
                {
                    if (FRes.BootConfInterval[p]>FRes.OrigMaxCoord) break;
                }
                FRes.BootConfInterval.insert(FRes.BootConfInterval.begin()↙
    +p, FRes.OrigMaxCoord);

cout << "Interval 95%: " << endl;
cout << Join(FRes.BootConfInterval, " ") << endl;

            }

            if (NPar.NumOfBootstraps>1)
            {
                // ErrorType
                EXTNUM Sum  = 0;
                EXTNUM Sum3  = 0;
                EXTNUM Avg  = 0;
                EXTNUM Diff  = 0;
                FRes.LTONum = 0;
                int i;
                for(i=0; i<FRes.BootMaxFL.size(); i++)
                {
                    Avg += FRes.BootMaxFL[i];
                }
                FRes.Average = Avg / (EXTNUM)NPar.NumOfBootstraps;

                for(i=0; i<FRes.BootMaxFL.size(); i++)
                {
                    Diff = FRes.Average - FRes.BootMaxFL[i];
                    Sum += Diff*Diff;
                    Sum3 += Diff*Diff*Diff;
                    if (Diff<(EXTNUM)0) FRes.LTONum++;
                }
                EXTNUM ATmp = sqrt(Sum);
                Sum /= (EXTNUM) (NPar.NumOfBootstraps-1);
                FRes.ErrorType = sqrt(Sum);
                FRes.Biais = FRes.Average - FRes.OrigMaxFLik;
                FRes.ICVMin = FRes.OrigMaxFLik - 1.96 * FRes.ErrorType;
                FRes.ICVMax = FRes.OrigMaxFLik + 1.96 * FRes.ErrorType;

                FRes.LTOCoef = ((double)FRes.LTONum / (double)NPar.       ↙
    NumOfBootstraps);
                ATmp = ATmp * ATmp * ATmp;
                ATmp *= 6;

                double Z0 = ltqnorm(FRes.LTOCoef);

                if (ATmp!=(EXTNUM)0)
                    FRes.AChapeau = Sum3 / ATmp;
                else
                    FRes.AChapeau = 0;

                double alpha1 = Z0 + (Z0-1.96)/(1-((double)FRes.AChapeau)*↙
    (Z0-1.96));
                double alpha2 = Z0 + (Z0+1.96)/(1-((double)FRes.AChapeau)*↙
    (Z0+1.96));
```

```
cout << "Alpha 1: " << alpha1 << endl;
cout << "Alpha 2: " << alpha2 << endl;
cout << endl;
            }
cout << "ErrorType: " << FRes.ErrorType << endl;
cout << "Average:  " << FRes.Average  << endl;
cout << "Biais:  "   << FRes.Biais    << endl;
cout << "IC-ErrorType:  [" << FRes.ICVMin  << ", " << FRes.ICVMax << "]" <↙
    < endl;
cout << "IC-ErrorType-Biais:  [" << (FRes.ICVMin-FRes.Biais)  << ", " <<  ↙
    (FRes.ICVMax-FRes.Biais) << "]" << endl;

cout << "Less-Than-Orig -> count:" << FRes.LTONum << ", coef:" << FRes.   ↙
    LTOCoef << endl;
cout << "A Chapeau: " << FRes.AChapeau << endl;

        }
        else
        {
            cout << "    IC does not exist !!!" << endl;
        }
    }

// cout << Join(FRes.OrigCanCoord, " ") << endl;

    for(p=0; p<FRes.AvgFinalLik.size(); p++)
    {
        FRes.AvgFinalLik[p] /= FLCount;
    }
    // FindMaxPosDump = 1;
    int AvgPos = FindMaxPos(FRes.AvgFinalLik);
    FRes.AvgMaxCoord = FRes.OrigCanCoord[AvgPos];
    // FindMaxPosDump = 0;

cout << "AvgMaxCoord:  " << FRes.AvgMaxCoord << endl;
cout << "Max(AvgFinalLik):  " << FRes.AvgFinalLik[AvgPos] << endl;
cout << "Average Boot FinalLik: " << endl;
cout << Join(FRes.AvgFinalLik, " ") << endl;

    return true;
}

//              ( S )   ( H )
// Calculates   (   ) * (   )
//              ( k )   ( k )
double CalcWeight(int k, int S, int H)
{
    double R = 1;

    if (k>S || k>H) return 0;

    for(int i=1; i<=k; i++)
    {
        R *= S--; R *= H--; R /= i*i;
    }
    return R;
}
```

```
bool DoPermutations(NatParams &NPar, FabParams &FPar, FabRuntime &RT,
    FabResults &FRes)
{
    int N = FPar.HapDataPerm.size();
    int p;
    int FLCount = 1;

    FRes.PermAvgFinalLik.assign(FRes.OrigFinalLik.begin(), FRes.
    OrigFinalLik.end());

    for(p=0; p<FRes.PermAvgFinalLik.size(); p++)
    {
        FRes.PermAvgFinalLik[p] = 0; // log(FRes.PermAvgFinalLik[p]);
    }


    if (NPar.NumOfPermutations && N>1)
    {
        MTRand MTR;
        VecI Sick;          // holds indexes in HapDataPerm
        VecI Healthy;       // holds indexes in HapDataPerm

        VVecI::iterator begin = FPar.HapDataPerm.begin(), end = FPar.
    HapDataPerm.end(), it;

        for(it=begin; it!=end; it++)
        {
            if(it->size()>1)
            {
                if ((*it)[1]==1)
                    Sick.push_back(it-begin);
                else if ((*it)[1]==0)
                    Healthy.push_back(it-begin);
            }
        }
        int SNum = Sick.size();     // number of sick
        int HNum = Healthy.size();  // number of helthy

        if (SNum<1 || HNum<1 || N<3)
        {
            cerr << "Found " << SNum << " sick and " << HNum << " healthy
    of total " << N << endl;
            cerr << "Sorry, Can't do permutations test!" << endl;
        }
        else
            cout << "Found " << SNum << " sick and " << HNum << " healthy
    of total " << N << endl;

        // Calculating Weights
        int Num = min(SNum, HNum);

        // if (Num>1) Num /= 2;

        VecD Weight;
        Weight.push_back(0);        // This will become the sum of all w.

        int k;
        for(k=1; k<=Num; k++) Weight.push_back(CalcWeight(k, SNum, HNum));
```

```
        for(k=1; k<=Num; k++) Weight[0] += Weight[k];

cout << endl << endl;
cout << "Weight: " << endl;
cout << Join(Weight, " ") << endl;

        int Count, ri;
        double Rnd;
        VecI Draw;
        VecI Stack;

        RT.ParFileLines.assign(FPar.ParFileLines.begin(), FPar.
    ParFileLines.end());
        FRes.PermPValue.clear();
        FRes.PermPValue2.clear();
        for(k=0; k < FRes.OrigFinalLik.size(); k++) FRes.PermPValue.
    push_back(0);
        for(k=0; k < FRes.OrigFinalLik.size(); k++) FRes.PermPValue2.
    push_back(0);
        VecE Empty;
        for(k=0; k < FRes.OrigFinalLik.size(); k++) FRes.Levels.push_back
    (Empty);
        for(long i=0; i<NPar.NumOfPermutations; i++)
        {
            RT.Reset();

            Rnd = MTR.randExc(Weight[0]);
            for(k=1; k<=Num; k++)
            {
                if (Rnd<Weight[k]) break;
                Rnd -= Weight[k];
            }
            Count = k;
            for(k=0; k<Count; k++)
            {
                ri = SNum>1? MTR.randInt(SNum-1) : 0;
                Stack.push_back(ri);
                Draw.push_back(Sick[ri]);
                Sick.erase(Sick.begin()+ri);
                SNum--;

                ri = HNum>1? MTR.randInt(HNum-1) : 0;
                Stack.push_back(ri);
                Draw.push_back(Healthy[ri]);
                Healthy.erase(Healthy.begin()+ri);
                HNum--;
            }

cout << endl << endl;
cout << "Draw (#" << (i+1) << "): " << endl;
cout << Join(Draw, " ") << endl;

            RT.DatFileLines.clear();
            for(k=0; k<N; k++)
            {
                bool flag = find(Draw.begin(), Draw.end(), k) != Draw.end
    ();
                string Line;
```

```
                if (flag && FPar.HapDataPerm[k].size()>1)
                {
                    // Count > 1
                    if ( FPar.HapDataPerm[k][0]>1 )
                    {
                        int HC = FPar.HapDataPerm[k][0];
                        FPar.HapDataPerm[k][0]--;
                        Line = Join(FPar.HapDataPerm[k]);
                        RT.DatFileLines.push_back(Line);
                        FPar.HapDataPerm[k][1] = ! FPar.HapDataPerm[k][1];
                        FPar.HapDataPerm[k][0] = 1;
                        Line = Join(FPar.HapDataPerm[k]);
                        FPar.HapDataPerm[k][1] = ! FPar.HapDataPerm[k][1];
                        FPar.HapDataPerm[k][0] = HC;
                    }
                    else
                    {
                        FPar.HapDataPerm[k][1] = ! FPar.HapDataPerm[k][1];
                        Line = Join(FPar.HapDataPerm[k]);
                        FPar.HapDataPerm[k][1] = ! FPar.HapDataPerm[k][1];
                    }
                }
                else
                    Line = Join(FPar.HapDataPerm[k]);
                RT.DatFileLines.push_back(Line);
            }

            for(k=0; k<Count; k++)
            {
                int ind = Stack.back();
                ri = Draw.back();
                Stack.pop_back();
                Draw.pop_back();
                Healthy.insert(Healthy.begin()+ind, ri);
                HNum++;

                ind = Stack.back();
                ri = Draw.back();
                Stack.pop_back();
                Draw.pop_back();
                Sick.insert(Sick.begin()+ind, ri);
                SNum++;
            }

cout << endl << endl;
cout << "Permutations: " << endl;
cout << Join(RT.DatFileLines, "\n") << endl;
            char print_buf[100];
            sprintf(print_buf, "Perm%03ld", i+1);

            OFName = print_buf;


            CallMapArg(NPar, FPar, RT, FRes);

            int MPos = FindMaxPos(FRes.LastCorrFinalLik);
            if (MPos>=0 && MPos<FRes.OrigCanCoord.size())
            {
```

```
            FRes.PermCoord.push_back(FRes.OrigCanCoord[MPos]);
            FRes.PermMaxFL.push_back(FRes.LastCorrFinalLik[MPos]);
            //FRes.BMFLInd.push_back((int)FRes.BMFLInd.size());
        }

        for(p=0; p<FRes.PermAvgFinalLik.size(); p++)
        {
            FRes.PermAvgFinalLik[p] += log(FRes.LastCorrFinalLik[p]);
        }
        FLCount++;


        if (FRes.MutDist.size()<l)
        {
            for (int ti=CanCoord.size(); ti>0; ti--) FRes.MutDist.
push_back(0);
        }

        int MaxPos = f_MaxE(FPar.BackgroundL? FinalLikC : FinalLik);

        FRes.MutDist[MaxPos] += l;

        FRes.PermDistCoord.push_back(FRes.OrigCanCoord[MaxPos]);

        if (FPar.BackgroundL)
        {

            // Save Levels
            for(k=0; k<FinalLikC.size(); k++)
            {
                FRes.Levels[k].push_back(FinalLikC[k]);
            }

            Count = min(FRes.OrigFinalLik.size(), FinalLikC.size());

            for(k=0; k<Count; k++)
            {
                if (FinalLikC[k] < FRes.OrigFinalLik[k]) FRes.
PermPValue[k] += l;
            }
            for(k=0; k<Count; k++)
            {
                if (FinalLikC[k] >= FRes.OrigMaxFLik) FRes.PermPValue2
[k] += l;
            }
        }
        else
        {
            // Save Levels
            for(k=0; k<FinalLikC.size(); k++)
            {
                FRes.Levels[k].push_back(FinalLikC[k]);
            }

            Count = min(FRes.OrigFinalLik.size(), FinalLik.size());

            for(k=0; k<Count; k++)
            {
```

```
                    if (FinalLik[k] < FRes.OrigFinalLik[k]) FRes.
    PermPValue[k] += 1;
                }
                for(k=0; k<Count; k++)
                {
                    if (FinalLik[k] >= FRes.OrigMaxFLik) FRes.PermPValue2
    [k] += 1;
                }
            }
        }

cout << "PermCoord: " << endl;
cout << Join(FRes.PermCoord, " ") << endl;

cout << "PermMaxFL: " << endl;
cout << Join(FRes.PermMaxFL, " ") << endl;


        for(k=0; k<FRes.PermPValue.size(); k++)
        {
            FRes.PermPValue[k] /= NPar.NumOfPermutations;
        }

        for(k=0; k<FRes.PermPValue2.size(); k++)
        {
            FRes.PermPValue2[k] /= NPar.NumOfPermutations;
        }

        for(k=0; k<FRes.MutDist.size(); k++)
        {
            FRes.MutDist[k] /= NPar.NumOfPermutations;
        }

cout << "Levels:" << endl;
for(k=0; k<FRes.Levels.size(); k++)
{
cout << Join(FRes.Levels[k], " ") << endl;
}

cout << "log(Levels):" << endl;
for(k=0; k<FRes.Levels.size(); k++)
{
for(int k1=0; k1<FRes.Levels[k].size(); k1++)
{
cout << log(FRes.Levels[k][k1]) << " ";
}
cout << endl;
}


        for(k=0; k<FRes.Levels.size(); k++)
        {
            sort(FRes.Levels[k].begin(), FRes.Levels[k].end() );
        }

        int ind99, ind95, ind63;
        // VecE Level99, Level95, Level63;
```

```
    ind99 = (int)((99.0*NPar.NumOfPermutations)/100.0); //  + 0.499);
    ind95 = (int)((95.0*NPar.NumOfPermutations)/100.0); //  + 0.499);
    ind63 = (int)((63.0*NPar.NumOfPermutations)/100.0); //  + 0.499);

    for(k=0; k<FRes.Levels.size(); k++)
    {
        FRes.Level99.push_back(FRes.Levels[k][ind99]);
        FRes.Level95.push_back(FRes.Levels[k][ind95]);
        FRes.Level63.push_back(FRes.Levels[k][ind63]);
    }

    sort(FRes.PermDistCoord.begin(), FRes.PermDistCoord.end());

    int m, indCount = NPar.NumOfPermutations*FRes.Levels.size();

    ind99 = (int)((99.0*indCount)/100.0); //  + 0.499);
    ind95 = (int)((95.0*indCount)/100.0); //  + 0.499);
    ind63 = (int)((63.0*indCount)/100.0); //  + 0.499);

    VecI I;
    for(m=0; m<FRes.Levels.size(); m++) I.push_back(FRes.Levels[m].
size()-1);
    EXTNUM Val;
    do
    {
        int p, pi=0;
        Val = FRes.Levels[0][I[0]];
        for(p=1; p<FRes.Levels.size(); p++)
        {
            if (I[p]>0 && FRes.Levels[p][I[p]]>Val)
            {
                Val = FRes.Levels[p][I[p]];
                pi = p;
            }
        }
        I[pi]--;
        if (indCount==ind99)
            FRes.PermValue99 = Val;
        if (indCount==ind95)
            FRes.PermValue95 = Val;
        if (indCount==ind63)
            FRes.PermValue63 = Val;
    } while (indCount-->ind63);


    VecE Tmp;
    Tmp.assign(FRes.PermMaxFL.begin(), FRes.PermMaxFL.end());
    sort(Tmp.begin(), Tmp.end());

    ind99 = (int)((99.0*Tmp.size())/100.0); // + 0.499);
    ind95 = (int)((95.0*Tmp.size())/100.0); // + 0.499);
    ind63 = (int)((63.0*Tmp.size())/100.0); // + 0.499);

    FRes.PermValue99 = Tmp[ind99];
    FRes.PermValue95 = Tmp[ind95];
    FRes.PermValue63 = Tmp[ind63];

    int ZCnt = 0;
```

```
        for (int z=0; z<FRes.PermMaxFL.size(); z++)
        {
            if (FRes.PermMaxFL[z] >= FRes.OrigMaxFLik) ZCnt++;
        }

    for(p=0; p<FRes.PermAvgFinalLik.size(); p++)
    {
        FRes.PermAvgFinalLik[p] /= FLCount;
    }
    int AvgPos = FindMaxPos(FRes.PermAvgFinalLik);
    FRes.PermAvgMaxCoord = FRes.OrigCanCoord[AvgPos];

cout << "PermAvgMaxCoord:  " << FRes.PermAvgMaxCoord << endl;
cout << "Average Perm FinalLik: " << endl;
cout << Join(FRes.PermAvgFinalLik, " ") << endl;


cout << endl << endl;
cout << "PValue -- count: " << ZCnt << ", coef: " << ((double)ZCnt/
    (double)FRes.PermMaxFL.size()) << endl;
cout << endl << endl;


cout << endl << endl;
cout << "PValues: " << endl;
cout << Join(FRes.PermPValue, " ") << endl;

cout << endl << endl;
cout << "PValues2: " << endl;
cout << Join(FRes.PermPValue2, " ") << endl;

cout << endl << endl;
cout << "Mutation Distribution: " << endl;
cout << Join(FRes.MutDist, " ") << endl;


cout << endl << endl;
cout << "Level 99%: " << endl;
cout << Join(FRes.Level99, " ") << endl;

cout << endl << endl;
cout << "Level 95%: " << endl;
cout << Join(FRes.Level95, " ") << endl;

cout << endl << endl;
cout << "Level 63%: " << endl;
cout << Join(FRes.Level63, " ") << endl;

cout << endl << endl;
cout << "PermDistCoord: " << endl;
cout << Join(FRes.PermDistCoord, " ") << endl;


cout << endl << endl;
cout << "Value 99%: " << FRes.PermValue99 << ",    Value 95%: " << FRes.
    PermValue95 << ",    Value 63%: " << FRes.PermValue63 << endl;
cout << Join(FRes.PermDistCoord, " ") << endl;
```

```
    }

    return true;
}


bool ProcessResults(NatParams &NPar, FabParams &FPar, FabResults &FRes)
{

    return true;
}


void WriteRPlot(NatParams &NPar, FabParams &FPar, FabResults &FRes);
void WriteRPlotDV(NatParams &NPar, FabParams &FPar, FabResults &FRes);


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

    NPar = new NatParams();
    FPar = new FabParams();
    RT   = new FabRuntime();

    if (ReadParams(*NPar, *FPar, argc, argv))
    {
        if (Init(*NPar, *FPar, *RT, FRes))
        {

            if (NPar->MultiDV)
            {
                DoMultiDV(*NPar, *FPar, *RT, FRes);

                WriteRPlotDV(*NPar, *FPar, FRes);
            }
            else
            {
                CallMapArgWithOrigData(*NPar, *FPar, *RT, FRes);

                DoBootstrap(*NPar, *FPar, *RT, FRes);

                DoPermutations(*NPar, *FPar, *RT, FRes);

cout << "AdjustScale: " << AdjustScale << endl;


                WriteRPlot(*NPar, *FPar, FRes);
                WriteGnuPlotNat(*NPar, *FPar, FRes);
                OutputTeXNat(*NPar, *FPar, FRes);

            }

        }
    }

    if ( RT   ) delete RT;
    if ( FPar ) delete FPar;
```

```
    if ( NPar ) delete NPar;

    return 0;
}


void WriteRPlot(NatParams &NPar, FabParams &FPar, FabResults &FRes)
{

    int AvgInd = 0;
    int PermAvgInd = 0;

    Time T;
    char KLBuf[300];
    sprintf(KLBuf, "--p%d--b%d-K%d-L%d %d-%02d-%02d %02d.%02d", NPar.      ↙
    NumOfPermutations, NPar.NumOfBootstraps, K_I, Lw_I, T.since1900()+1900↙
    , T.month(), T.dayOfMonth(), T.hour(), T.minute());

    string RFile    = FPar.ParFileName;
    int pos = RFile.find_last_of('.');
    if (pos!=string::npos)
        RFile.replace(RFile.begin()+pos, RFile.end(), "");
    string PdfFile  = "MapArg-" + RFile + KLBuf + ".pdf";
    RFile = "MapArg-" + RFile + KLBuf + ".r";

    ofstream outd(RFile.c_str(), ios::out);

    if (!outd)
    {
        cerr << "\'" << RFile << "\' can not be opened for writing.\n";
        return;
    }

    unsigned int index;
    outd << "# Command file for R" << endl;

    // X values
    outd <<"x<-c(";
    for (index=0; index<FRes.OrigCanCoord.size(); ++index)
    {
        if (index>0) outd <<",";
        if ( index%50==49 && index+1<FRes.OrigCanCoord.size() )
            outd <<"\"<<endl;
        outd << (FRes.OrigCanCoord[index] / AdjustScale) * 100 ;
    }
    outd << ")" << endl;


    extnum MinimumG, MaximumG;

    MinimumG = FRes.OrigFinalLik[0];
    MaximumG = MinimumG;

    outd <<"y<-c(";
    for ( index=0; index<FRes.OrigFinalLik.size(); ++index )
    {
        if (index>0) outd <<",";
        if ( index%50==49 && index+1<FRes.OrigFinalLik.size() )
```

```
            outd <<"+"<<endl;
        outd << log(FRes.OrigFinalLik[index]);
        if( FRes.OrigFinalLik[index] > MaximumG)
            MaximumG = FRes.OrigFinalLik[index];
        if( FRes.OrigFinalLik[index] < MinimumG)
            MinimumG = FRes.OrigFinalLik[index];
    }
    outd << ")" << endl;

    int line0 = 1;

    if ( NPar.NumOfPermutations>0 )
    {
        outd <<"y1<-c(";
        for ( index=0; index<FRes.Level99.size(); ++index )
        {
            if (index>0) outd <<",";
            if ( index%50==49 && index+1<FRes.Level99.size() )
                outd <<"+"<<endl;
            outd << log(FRes.Level99[index]);
            if( FRes.Level99[index] > MaximumG)
                MaximumG = FRes.Level99[index];
            if( FRes.Level99[index] < MinimumG)
                MinimumG = FRes.Level99[index];
        }
        outd << ")" << endl;

        outd <<"y2<-c(";
        for ( index=0; index<FRes.Level95.size(); ++index )
        {
            if (index>0) outd <<",";
            if ( index%50==49 && index+1<FRes.Level95.size() )
                outd <<"+"<<endl;
            outd << log(FRes.Level95[index]);
            if( FRes.Level95[index] > MaximumG)
                MaximumG = FRes.Level95[index];
            if( FRes.Level95[index] < MinimumG)
                MinimumG = FRes.Level95[index];
        }
        outd << ")" << endl;

        outd <<"y3<-c(";
        for ( index=0; index<FRes.Level63.size(); ++index )
        {
            if (index>0) outd <<",";
            if ( index%50==49 && index+1<FRes.Level63.size() )
                outd <<"+"<<endl;
            outd << log(FRes.Level63[index]);
            if( FRes.Level63[index] > MaximumG)
                MaximumG = FRes.Level63[index];
            if( FRes.Level63[index] < MinimumG)
                MinimumG = FRes.Level63[index];
        }
        outd << ")" << endl;

        line0 = 4;
    }
```

```
    if ( NPar.NumOfBootstraps>0 )
    {
        outd <<"y"<< line0 << "<-c(";
        for ( index=0; index<FRes.AvgFinalLik.size(); ++index )
        {
            if (index>0) outd <<",";
            if ( index%50==49 && index+1<FRes.AvgFinalLik.size() )
                outd <<"+"<<endl;
            outd << FRes.AvgFinalLik[index];
/*
            if( FRes.AvgFinalLik[index] > MaximumG)
                MaximumG = FRes.AvgFinalLik[index];
            if( FRes.AvgFinalLik[index] < MinimumG)
                MinimumG = FRes.AvgFinalLik[index];
*/
        }
        outd << ")" << endl;
        AvgInd = line0;
        line0++;
    }

    if ( NPar.NumOfPermutations>0 )
    {
        outd <<"y"<< line0 << "<-c(";
        for ( index=0; index<FRes.PermAvgFinalLik.size(); ++index )
        {
            if (index>0) outd <<",";
            if ( index%50==49 && index+1<FRes.PermAvgFinalLik.size() )
                outd <<"+"<<endl;
            outd << FRes.PermAvgFinalLik[index];
        }
        outd << ")" << endl;
        PermAvgInd = line0;
        line0++;
    }

    if (FRes.OrigMatLik.size()>1)
    {
        for (int row=0; row<FRes.OrigMatLik.size(); row++)
        {
            outd <<"y"<< row+line0 << "<-c(";
            for ( index=0; index<FRes.OrigMatLik[row].size(); ++index )
            {
                if (index>0) outd <<",";
                if ( index%50==49 && index+1<FRes.OrigMatLik[row].size() )↙

                    outd <<"+"<<endl;
                outd << log(FRes.OrigMatLik[row][index]);
                if( FRes.OrigMatLik[row][index] > MaximumG)
                    MaximumG = FRes.OrigMatLik[row][index];
                if( FRes.OrigMatLik[row][index] < MinimumG)
                    MinimumG = FRes.OrigMatLik[row][index];
            }
            outd <<")"<<endl;
        }
    }

    outd << "pdf(\"" << PdfFile << "\")" << endl;
```

```
outd << "# Min: " << MinimumG << ", Max: " << MaximumG << endl;
outd << "# log(Min): " << log(MinimumG) << ", log(Max): " << log
(MaximumG) << endl;
outd << "plot(x,y,type=\"n\",xlab=\"\",ylab=\"\",ylim=c(" << log
(MinimumG) << "," << log(MaximumG) << "),las=1)" << endl;
outd << "lines(x,y,col=\"blue\", lwd=4)" << endl;

if ( NPar.NumOfBootstraps>0 )
{
    outd << "lines(x,y"<< AvgInd << ",col=\"black\", lwd=2)" << endl;
}

if ( NPar.NumOfPermutations>0 )
{
    outd << "lines(x,y1,col=\"green\", lwd=2)" << endl;
    outd << "lines(x,y2,col=\"brown\", lwd=2)" << endl;
    outd << "lines(x,y3,col=\"orange\", lwd=2)" << endl;
    outd << "lines(x,y"<< PermAvgInd << ",col=\"black\", lwd=2)" <<
endl;
}

if (FRes.OrigMatLik.size()>1)
{
    for (int row=0; row<FRes.OrigMatLik.size(); row++)
    {
        outd << "lines(x,y" << row+line0 << ")" << endl;
    }
}


if (RealP >-1)
{
   outd<<"abline(v="<< (RealP/AdjustScale) <<",col=\"red\",lty=2,lwd=
1)"<<endl;
}

if (NPar.NumOfBootstraps)
{
   outd<<"abline(h="<< (log(FRes.BootValue5))  <<",col=\"red\",lty=2,
lwd=1)"<<endl;
}

if (NPar.NumOfPermutations)
{
   outd<<"abline(h="<< (log(FRes.PermValue63)) <<",col=\"orange\",lty=
2,lwd=1)"<<endl;
   outd<<"abline(h="<< (log(FRes.PermValue95)) <<",col=\"brown\",lty=2
,lwd=1)"<<endl;
   outd<<"abline(h="<< (log(FRes.PermValue99)) <<",col=\"green\",lty=2
,lwd=1)"<<endl;
}

if (NPar.NumOfBootstraps && FPar.HapDataPerm.size()>1 && FRes.
BootConfInterval.size()>1)
{
   outd<<"abline(v="<< (FRes.BootConfInterval[0]/AdjustScale) * 100 <
<",col=\"yellow\",lty=2,lwd=2)"<<endl;
```

```
    outd<<"abline(v="<< (FRes.BootConfInterval[FRes.BootConfInterval.
  size()-1]/AdjustScale) * 100 <<",col=\"yellow\",lty=2,lwd=2)"<<endl;
  }

  outd <<"MarkCoord<-c(";
  for (index=0; index<FRes.OrigCoord.size(); ++index)
  {
      if (index>0) outd <<",";
      if ( index%50==49 && index+1<FRes.OrigCoord.size() )
          outd << "+" << endl;
      outd << (FRes.OrigCoord[index] / AdjustScale) * 100 ;
  }
  outd <<")"<<endl;
  outd << "rug(MarkCoord)" << endl;

  outd <<"points("<< ( (FRes.OrigMaxCoord/AdjustScale) * 100) <<","<<
  log(MinimumG) <<",pch=24,col=\"blue\",lwd=2)"<<endl;

  if (NPar.NumOfBootstraps)
  {
      outd <<"points("<< ( (FRes.AvgMaxCoord/AdjustScale) * 100) <<","<<
   log(MinimumG) <<",pch=24,col=\"black\",lwd=2)"<<endl;
  }

  if (NPar.NumOfPermutations)
  {
      outd <<"points("<< ( (FRes.PermAvgMaxCoord/AdjustScale) * 100) <<"
  ,"<< log(MinimumG) <<",pch=24,col=\"red\",lwd=2)"<<endl;
  }

  outd <<"dev.off()"<<endl;
  outd << "\n";
  outd.close();

}


void WriteRPlotDV(NatParams &NPar, FabParams &FPar, FabResults &FRes)
{

  char KLBuf[100];
  sprintf(KLBuf, "-K%d-L%d", K_I, Lw_I);

  string RFile    = FPar.ParFileName;
  int pos = RFile.find_last_of('.');
  if (pos!=string::npos)
      RFile.replace(RFile.begin()+pos, RFile.end(), "");
  string PdfFile = "MapArgDV-" + RFile + KLBuf + ".pdf";
  RFile = "MapArgDV-" + RFile + KLBuf + ".r";

  ofstream outd(RFile.c_str(), ios::out);

  if (!outd)
  {
      cerr << "\'" << RFile << "\' can not be opened for writing.\n";
      return;
  }
```

```
unsigned int index;
outd << "# Command file for R" << endl;

// X values
outd <<"x<-c(";
for (index=0; index<CanCoord.size(); ++index)
{
    if (index>0) outd <<",";
    if ( index%50==49 && index+1<CanCoord.size() )
        outd <<"+"<<endl;
    outd << (CanCoord[index] / AdjustScale) * 100 ;
}
outd << ")" << endl;


extnum MinimumG, MaximumG;

MinimumG = FRes.DVBest[0];
MaximumG = MinimumG;

int row;


outd <<"y<-c(";
for ( index=0; index<FRes.DVBest.size(); ++index )
{
    if (index>0) outd <<",";
    if ( index%50==49 && index+1<FRes.DVBest.size() )
        outd <<"+"<<endl;
    outd << log(FRes.DVBest[index]);
    if( FRes.DVBest[index] > MaximumG)
        MaximumG = FRes.DVBest[index];
    if( FRes.DVBest[index] < MinimumG)
        MinimumG = FRes.DVBest[index];
}
outd << ")" << endl;

for (row=0; row<FRes.DVFinalLik.size(); row++)
{
    outd <<"y"<< row+1 << "<-c(";
    for ( index=0; index<FRes.DVFinalLik[row].size(); ++index )
    {
        if (index>0) outd <<",";
        if ( index%50==49 && index+1<FRes.DVFinalLik[row].size() )
            outd <<"+"<<endl;
        outd << log(FRes.DVFinalLik[row][index]);
        if( FRes.DVFinalLik[row][index] > MaximumG)
            MaximumG = FRes.DVFinalLik[row][index];
        if( FRes.DVFinalLik[row][index] < MinimumG)
            MinimumG = FRes.DVFinalLik[row][index];
    }
    outd <<")"<<endl;
}

int AvgInd = row + 1;
outd <<"y"<< AvgInd << "<-c(";
for ( index=0; index<FRes.DVAverage.size(); ++index )
{
```

```
        if (index>0) outd <<",";
        if ( index%50==49 && index+1<FRes.DVAverage.size() )
            outd <<"+"<<endl;
        outd << log(FRes.DVAverage[index]);
        if( FRes.DVAverage[index] > MaximumG)
            MaximumG = FRes.DVAverage[index];
        if( FRes.DVAverage[index] < MinimumG)
            MinimumG = FRes.DVAverage[index];
    }
    outd << ")" << endl;


    outd << "pdf(\"" << PdfFile << "\")" << endl;

    outd << "plot(x,y,type=\"n\",xlab=\"\",ylab=\"\",ylim=c(" << log
    (MinimumG) << "," << log(MaximumG) << "),las=1)" << endl;

    for (row=0; row<FRes.DVFinalLik.size(); row++)
    {
        outd << "lines(x,y" << row+1 << ")" << endl;
    }
    outd << "lines(x,y" << AvgInd << ",col=\"blue\", lwd=2)" << endl;
    outd << "lines(x,y,col=\"red\", lwd=2)" << endl;

    outd <<"MarkCoord<-c(";
    for (index=0; index<Coord.size(); ++index)
    {
        if (index>0) outd <<",";
        if ( index%50==49 && index+1<Coord.size() )
            outd << "+" << endl;
        outd << (Coord[index] / AdjustScale) * 100 ;
    }
    outd <<")"<<endl;
    outd << "rug(MarkCoord)" << endl;

    if (RealP >-1)
    {
       outd<<"abline(v="<< (RealP/AdjustScale) <<",col=\"red\",lty=2,lwd=
    1)"<<endl;
    }

    int Pos = FindMaxPos(FRes.DVBest);
    if (Pos>-1)
    {
       outd<<"abline(v="<< (CanCoord[Pos]/AdjustScale) * 100 <<",col=\
    "green\",lty=2,lwd=1)"<<endl;
    }

    outd <<"points("<< ( (FRes.OrigMaxCoord/AdjustScale) * 100) <<","<<
    log(MinimumG) <<",pch=24,col=\"blue\",lwd=2)"<<endl;

    outd <<"dev.off()"<<endl;
    outd << "\n";
    outd.close();

}
```

```
// +-------------------------------------------------------------------+
// | void WriteGnuPlotNat()                                            |
     |
// +-------------------------------------------------------------------+
void WriteGnuPlotNat(NatParams &NPar, FabParams &FPar, FabResults &FRes)
{

    Time T;
    char KLBuf[300];
    sprintf(KLBuf, "--p%d--b%d-K%d-L%d %d-%02d-%02d %02d.%02d", NPar.
    NumOfPermutations, NPar.NumOfBootstraps, K_I, Lw_I, T.since1900()+1900
    , T.month(), T.dayOfMonth(), T.hour(), T.minute());

    string RFile    = FPar.ParFileName;
    int pos = RFile.find_last_of('.');
    if (pos!=string::npos)
        RFile.replace(RFile.begin()+pos, RFile.end(), "");
    string CmdFile  = "MapArg-" + RFile + KLBuf + ".plt";
    string EpsFile  = "MapArg-" + RFile + KLBuf + ".eps.plt";
    string DonFile  = "MapArg-" + RFile + KLBuf + ".don";

    ofstream outc(CmdFile.c_str(), ios::out);
    ofstream oute(EpsFile.c_str(), ios::out);
    ofstream outd(DonFile.c_str(), ios::out);

    if (!outc)
    {
        cerr << "\'" << CmdFile << "\' can not be opened for writing.\n";
        return;
    }

    if (!oute)
    {
        cerr << "\'" << EpsFile << "\' can not be opened for writing.\n";
        outc.close();
        return;
    }

    if (!outd)
    {
        cerr << "\'" << DonFile << "\' can not be opened for writing.\n";
        outc.close();
        oute.close();
        return;
    }


    outc <<"# Command file for GNU PLOT"<< endl;
    outc <<"set grid"<< endl;
    outc <<"set nokey"<< endl;
    outc <<"set ylabel 'ln(L)'"<< endl;
    outc <<"set xlabel 'r_T'"<< endl;
    if ( RealP>-1 && RealI>-1 )
    {
      outc << "set arrow from " << RealP / AdjustScale << ", graph 0 to "
    << RealP / AdjustScale << ", graph 1 nohead"<< endl;
    }
```

```
outc << "plot '" << DonFile << "' using ($3*100):5 with lines ";

//outc <<"pause -1"<< endl;
/*if(BackgroundL == 1){
  outc <<"plot '"<<don_file<<"' using ($3*100):6 with lines"<< endl;
  outc <<"pause -1"<< endl;
}*/

if( FRes.OrigMatLik.size()>1)
{
   for(unsigned int j=0 ; j!= FRes.OrigMatLik.size() ; ++j){
       outc <<", '"<< DonFile << "' using ($3*100):" << 6+j << " with lines ";
   }
}
outc << endl << "pause -1" << endl;
outc.close();

// EPS file ;
oute << "# Command file for GNU PLOT to produce EPS file" << endl;
oute << "# EPS Section" << endl;
oute << "  set size 3.5/5.0 ,  3.0/3.0" << endl;
oute << "  set term postscript eps enhanced color blacktext solid
linewidth 2 \"cmr10\" 14 " << endl;
oute << "  # Syntax: lt->linetype, lw->linewidth, pt->pointtype, ps->
pointsize " << endl;
oute << "  set style line 1 lt 1 lw 2" << endl;
oute << "  set style line 2 lt 3 lw 3  " << endl;
oute << "  set style line 3 lt 7 lw 1 " << endl;
oute << "  set output \"" << EpsFile << "\"" << endl;
oute << "  unset key " << endl;


if ( RealP>-1 && RealI>-1 )
{
  oute << "set arrow from " << RealP / AdjustScale << ", graph 0 to "
<< RealP / AdjustScale << ", graph 1 nohead" << endl;
}
oute << "plot '" << DonFile << "' using ($3*100):5 with lines ls 2 ";
if ( FRes.OrigMatLik.size()>1 )
{
   for(unsigned int j=0 ; j!= FRes.OrigMatLik.size() ; ++j)
   {
       oute << ", '" << DonFile << "' using ($3*100):" << 6+j << "
with lines ls 3";
   }
}
oute << endl << "pause -1" << endl;
oute.close();
// End EPS File


outd << "# Can Int Coordinate       LikE           LLE    "<< endl;

for ( unsigned int index=0; index != (unsigned)FRes.OrigFinalLik.size
(); ++index)
{
    outd.width(3);  outd << index+1 ;
```

```
        outd.width(3);  outd << FRes.OrigCanInterval[index] ;
        outd.width(14); outd << FRes.OrigCanCoord[index] / AdjustScale ;
        outd.width(12); outd << FRes.OrigFinalLik[index] ;
        outd.width(12); outd << log( FRes.OrigFinalLik[index] );
        if ( FRes.OrigMatLik.size()>1 )
        {
           for(unsigned int ind2=0 ; ind2 != FRes.OrigMatLik.size(); ++
    ind2)
           {
               outd.width(12); outd << log( FRes.OrigMatLik[ind2][index] )
    ;
           }
        }
       outd << endl;
    }
    outd << endl;
    outd.close();

}

// +--------------------------------------------------------------------+
// | void SmallGraphNat()                                               |
// +--------------------------------------------------------------------+
void SmallGraphNat( NatParams &NPar, FabParams &FPar, FabResults &FRes,
    VecD VY , VecD VX , int GD , int GM , int Nx , int Ny , string label )
{

    // Make a graph of simple statistics like, r2, d'...
    // Min, Max and range ;
    int    IMinY = f_MinD(VY);
    double  MinY = VY[IMinY];
    int    IMaxY = f_MaxD(VY);
    double  MaxY = VY[IMaxY];
    int    IMinX = 0 ;
    double  MinX = VX[IMinX];
    int    IMaxX = VX.size()-1;
    double  MaxX = VX[IMaxX];
    double   EtX = MaxX - MinX ;
    double   EtY = MaxY - MinY ;
    for (unsigned int i=0 ; i!= VY.size(); ++ i)
    {
        //cerr<<"VY["<<i<<"]="<<  (VY[i]  - MinY)/ EtY;
        if( ( VY[i]  - MinY)/ EtY < 1e-5) VY[i]=0;
        //cerr<<"-> VY["<<i<<"]="<<  (VY[i]  - MinY)/ EtY<<endl;
    }
//cerr<<"** MinY="<<MinY<<", MaxY="<<MaxY<<", EtY="<<EtY<< endl;
    int GraphDim = GD ;
    int GraphMargin = GM ;
    int NbLabelX = Nx;
    int NbLabelY = Ny;
    out <<"% Main Graph "<< endl;
    out <<"  "<< endl;
    out <<"  \\Draw"<< endl;
    out <<"  \\Scale(1,1)\\Ragged(1) "<< endl;
    out <<"  \\Define\\LineChart(1){"<< endl;
    out <<"       \\NextTable{"<< endl;
    out <<"       \\PenSize(#1pt) \\data(0,0){\\MoveTo}\\data(0,999){\\
    LineTo} }"<< endl;
```

```
    out <<"       \\Table\\data }" << endl;
    out <<"  \\MoveTo(0,-" << GraphMargin<< ")\\MarkLoc(a) \\MoveTo(" <<
    GraphDim << ",-" << GraphMargin-.01 << ")\\MarkLoc(b)" << endl;
    out <<"  \\MoveTo(-" << GraphMargin << ",0)\\MarkLoc(c) \\MoveTo(-" <<
     GraphMargin-0.1 << "," << GraphDim << ")\\MarkLoc(d)" << endl;
    out <<"  \\DrawRectAt(-" << GraphMargin << ",-" << GraphMargin << "," 
    << GraphDim+GraphMargin << "," << GraphDim+GraphMargin << ")" << endl;
    // Tracing the main curve ;
    out << " {\\color{NavyBlue} \\LineChart{1}{  ";
    int index;
    for (index=0 ; index != FRes.OrigCoord.size(); ++ index)
    {
        out << " " << GraphDim*( (VX[index]/AdjustScale) - MinX)/ EtX ;
        out << "," << GraphDim*((  VY[index] ) - MinY)/ EtY ;
        if ( index!=FRes.OrigCoord.size() -1 ) out <<"&";
    }
    out << "} }" << endl;
    // X labels ;
    out << "{\\label \\Axis(a,b){S1 ,";
    double Increment = MinX;
    for (index = 0 ; index != NbLabelX ; ++index)
    {
        out << Increment*100 ;
        if ( index!=NbLabelX -1) out << " & ";
        Increment += EtX / (NbLabelX - 1);
    }
    out << ")" << endl;
    // Y labels ;
    out << "\\label \\Axis(c,d){W1 ,";
    Increment = MinY;
    for(index = 0; index != NbLabelY; ++index)
    {
        out.width(3); out << Increment ;
        if (index!=NbLabelY -1) out<<" & ";
        Increment += EtY / (NbLabelY - 1);
    }
    out << ") }" << endl;
    //out <<"\\MoveTo(150,-40)\\Text(--$r_T$--)"<< endl;
    out << "\\MoveTo(-40," << int(GraphDim/3*2) << ")\\Text(--" << label <
    < "--)" << endl;
    // Show TIM position if known, or any reference point you want to show
     on graph.
    if ( RealP >-1 && RealI >-1 )
    {
       out << "\\MoveTo(" << GraphDim*( ((RealP/100)/AdjustScale) - MinX)/
     EtX <<",0)\\Text(--$\\color{RedOrange}\\star$--)" << endl;
       //out <<"\\draw[style=TIMline]  plot coordinates {("<<xcoord<<",0) 
    ("<<xcoord<<",10)};";
    }
    out << "\\EndDraw " << endl;

    return;
}

// +-------------------------------------------------------------------+
// | void OutPutTeX()                                                  |
// +-------------------------------------------------------------------+
void OutputTeXNat( NatParams &NPar, FabParams &FPar, FabResults &FRes )
```

```
{

    Time T;
    char KLBuf[300];
    sprintf(KLBuf, "--p%d--b%d-K%d-L%d %d-%02d-%02d %02d.%02d", NPar.      ↙
    NumOfPermutations, NPar.NumOfBootstraps, K_I, Lw_I, T.since1900()+1900↙
    , T.month(), T.dayOfMonth(), T.hour(), T.minute());

    string RFile    = FPar.ParFileName;
    int pos = RFile.find_last_of('.');
    if (pos!=string::npos)
        RFile.replace(RFile.begin()+pos, RFile.end(), "");
    string TexFile  = "MapArg-" + RFile + KLBuf + ".tex";

    ofstream out(TexFile.c_str(), ios::out);

    if (!out)
    {
        cerr << "\'" << TexFile << "\' can not be opened for writing.\n";
        return;
    }

    VecD::iterator sjr ;

    char *prefix = "";

    float K2 = K;
    if(K2>=1000000){
        K2 /= 1000000;
        prefix = "M";
    }
    else if(K2>=1000){
        K2 /= 1000;
        prefix = "m";
    }
    char *prefix2 = "";
    float KT = TotalNbGraphEval;
    if(KT>=1000000){
        KT /= 1000000;
        prefix2 = "M";
    }
    else if(KT>=1000){
        KT /= 1000;
        prefix2 = "m";
    }
    int NbIntX = 5 , NbIntY = 5;

    //out << "\\newcount\\landscape \\landscape=0"<< endl;
    //out << "\\newcount\\LikBetweenInterval \\LikBetweenInterval=1 %(0,1)↙
    "<< endl;
    //out << "\\newcount\\ConfidenceInterval \\ConfidenceInterval=1 %(0,1)↙
    "<< endl;
    //out << "\\newcount\\LineAtTim \\LineAtTim=0 %(0,1)"<< endl;
    //out << "\\newcount\\MarkerSeparation \\MarkerSeparation=0 % ( 0,1,2)↙
    "<< endl;
    //out << "\\newcount\\TicsLines \\TicsLines=0 %(0,1,2)"<< endl;

    /*if(RP>-1 && RI>-1){
```

```
    out << "\\headline{{\\bf\\sc MapArg}\\copyright\\ ({\\tt "<
<program_name<<" / v. "
    <<version_programm<<"})\\hfill($r_T="<<RP<<"$ in interval "<<RI<
<")\\hfill{\\tt "
    <<FileR<<"}} "<< endl;
}*/

out << "%% This file is to be processed by TeX or pdfTeX" << endl;
out << "%% ----------------------------------------" << endl;
out << "\\input color"<< endl;
out << "\\input DraTex.sty \\input AlDraTex.sty" << endl;
out << "\\definecolor{NavyBlue}{cmyk}{0.94,0.54,0,0}" << endl;
out << "\\definecolor{RedOrange}{cmyk}{0,0.77,0.87,0}" << endl;

out << "\\headline{{\\bf\\sc MapArg}\\copyright\\hfill " <<  "--:--" <
< " }" << endl;
out << "\\bigskip" << endl;
out << "\\footline{\\hfill [{\\tt"<<"--:--"<<"}]\\hfill}" << endl;

out << "\\long\\def\\boxit#1#2{\\vbox{\\hrule\\hbox{\\vrule\\kern#1" <
< endl;
out << "         \\vbox{\\kern#1\\vbox{#2}\\kern#1}\\vrule}\\hrule}}" <
<< endl;
out << "\\centerline{\\bf\\boxit{.5em} {\\hfil Parameters\\hfil} }\n\\
medskip" << endl;
out << "\\medskip" << endl;

string ReTypeS;
if( ReType == 0) ReTypeS = "Normal";
else  ReTypeS = "Approx";

string MuTypeS;
if( MuType == 0) MuTypeS = "Normal";
else  MuTypeS = "Approx";
if( MarkerT == 0) MuTypeS = "."; // Snps ;
string MakerTypeS;
if( MarkerT == 0) MakerTypeS = "Snp  "; // Snps ;
if( MarkerT == 1) MakerTypeS = "Micro"; // Snps ;


string tmp = "-";
out << "\\centerline{\\vbox{\\halign{\\hfil # & \\hskip.3cm {\\tt #}\\
hfil \\hskip0.6cm&"
<<   "\\hfil# & \\hskip.3cm {\\tt #}\\hfil \\hskip0.6cm&"
<<   "\\hfil# & \\hskip.3cm {\\tt #}\\hfil \\cr" << endl;
out << "Data File & " << DataFile <<" & $ d^0$ & "  << d_I <<" & $n$ &
"<< N_0 <<"\\cr"<< endl;
out << "Par. File & " << ParFile  <<" & $K$ & "  << K2 << prefix   <<"
 & Marker & "<<MakerTypeS<<" \\cr"<< endl;
out << " Seed & "      << RandomSeed <<" & $K_T$ & "<< KT << prefix2
   <<" & $\\kappa$ & "<<kappa<<" \\cr"<< endl;
out << "$\\rho$ & "    << 4*Ne*rglobal      <<" & $r$ & "  << rho / (4*
Ne)<<" & Re$_{type}$& "<< ReTypeS <<" \\cr"<< endl;
out << "$\\theta$ & " << theta      <<" & $u_l$ & "<< mul <<" & Mu$_
{type}$& "<< MuTypeS <<" \\cr"<< endl;
//out << "Simple & "<<Simple<<" & Mu Type & "<<MuType<<" &Re Type & "<
<ReType<<" \\cr"<< endl;
out << " Nb Rep& "     << NbRep  <<" & MuOrder & "<< tmp << " &
```

```
AdjustMMM & "<< tmp    <<" \\cr"<< endl;
out << "$H$ & "<< H <<" & $u_{TIM}$ & "<< MuTim <<" & TIM Rec. & "<<
tmp <<" \\cr"<< endl;
out << " $L_w$ &"<< Lw <<  " & Lag &"  << LagW <<" & G(d) & " << NbW <
<" \\cr}}}"<< endl;

out << "\\font\\small=cmtt10 scaled 700"<< endl;
out << "\\font\\sc=cmcsc10"<< endl;
out << "\\font\\label=cmr10 scaled 800"<< endl;
out << "\\overfullrule=0pt"<< endl;

//PrintV(Use,"Use");
//PrintV(Coord_I,"Coord_I");
//PrintV(Coord,"Coord");

double TotDistance = *(Dist_I.end()-1);

out <<"\\bigskip"<< endl;
out <<"\\Draw"<< endl;
out <<"  \\Ragged(1)\\Scale(1,1)"<< endl;

// Info on used and unused markers ;
int LL = 200 ;  // Line length ;
// LL = ((double)70/(double)3) * (double)L -(double(50)/double(3));
LL = int(23.33333 * (double)(Coord_I.size()) - 16.6666) ;
if(Coord_I.size() < 5 ) LL = 100 ;
if(Coord_I.size() > 20 ) LL = 450 ;
//cerr <<"LL="<<LL<<endl;
int multiple = 5 ; // Default value from which we prin the marker
number ;
if(Coord_I.size() <= 5) multiple = 2;
if(Coord_I.size() > 5 && Coord_I.size() <= 10) multiple = 3;
out <<"  \\MoveTo(0,0)\\Line("<<LL<<",0) % Ligne ;"<< endl;
int index;
for(index=0 ; index != Coord_I.size(); ++ index){ // All markers ;
    out <<" \\MoveTo("<<LL*Coord_I[index]/ *(Coord_I.end()-1)<<",3)\\
Text(--$\\top$--) " ;
    out <<" \\MoveTo("<<LL*Coord_I[index]/ *(Coord_I.end()-1)<<",-3)\\
Text(--$\\bot$--) " ;
    if( index == 0 || (index+1) % multiple == 0)
    out <<"  \\MoveTo("<<LL*Coord_I[index]/ *(Coord_I.end()-1)<<",-12)
\\Text(--\\small "<<index+1<<"--) " ;
}
out << endl;
multiple = 5 ; // Default value from which we prin the marker number ;
if(Coord.size() <= 5) multiple = 1;
if(Coord.size() > 5 && Coord.size() <= 10) multiple = 2;
for(index=0 ; index != Coord.size(); ++ index){ // Used markers ;
    out <<" \\MoveTo("<<LL*Coord[index]/ *(Coord.end()-1)<<",0)\\Text
(--$\\bullet$--) " ;
    if( index == 0 || (index+1) % multiple == 0)
    out <<"  \\MoveTo("<<LL*Coord[index]/ *(Coord.end()-1)<<",12)\\
Text(--\\small "<<index+1<<"--) " ;
}
if(RealP >-1 && RealI >-1){
   out <<"\\MoveTo("<< LL*( ((RealP/100)/AdjustScale) / *(Coord.end()-
1) ) <<",-12}\\Text(--$\\color{RedOrange}\\star$--)"<< endl;
   //out <<"\\draw[style=TIMline]  plot coordinates {("<<xcoord<<",0)
```

```
("<<xcoord<<",10)};";
}

out << endl;
out <<"\\EndDraw"<< endl;

out << "\\bigskip"<< endl;
out << "\\centerline{\\bf\\boxit{.5em} {\\hfil Results\\hfil} }\n\\
medskip"<< endl;
out << "\\medskip"<< endl;
// out << "\\centerline{ "<< "-:-" <<" / "<< "-:-" <<" ["<<"-:-"<<"]}"
<<endl;
out << "\\centerline{\\vbox{\\halign{\\hfil # & \\hskip.3cm {\\tt #}\\
hfil \\hskip0.6cm&"
<<   "\\hfil# & \\hskip.3cm {\\tt #}\\hfil \\hskip0.6cm&"
<<   "\\hfil# & \\hskip.3cm {\\tt #}\\hfil \\cr" << endl;
out << " $\\hat{r_T}$ & "<< mlrt *100.0 <<" & Interval & "<<
mlinterval << " ";
out << "& 95\\%CI & (x:"<< tmp <<"x)  \\cr}}}" << endl;
out << "\\bigskip" << endl;

VecD VX;
VecD VY;
for(index=0 ; index != FRes.OrigFinalLik.size(); ++ index)
{
    VX.push_back( FRes.OrigCanCoord[index]/AdjustScale);
    VY.push_back( log( FRes.OrigFinalLik[index]) );
    if (FRes.OrigMatLik.size() > 1)
    {
       for ( int ind2=0; ind2 != FRes.OrigMatLik.size(); ++ind2)
       {
           VY.push_back( log( FRes.OrigMatLik[ind2][index] ));
       }
    }
}
//PrintV(VX,"VX");
//PrintV(VY,"VY");

// Min, Max and range ;
int     IMinY = f_MinD(VY);
double   MinY = VY[IMinY];
int     IMaxY = f_MaxD(VY);
double   MaxY = VY[IMaxY];
int     IMinX = 0;
double   MinX = VX[IMinX];
int     IMaxX = VX.size()-1;
double   MaxX = VX[IMaxX];
double    EtX = MaxX - MinX ;
double    EtY = MaxY - MinY ;
// copy Lik into Y vector and matrices to control units, eg to avoid
xe-45 which
VY.clear();

// cerr<<"MinY="<<MinY<<", MaxY="<<MaxY<<", EtY="<<EtY<< endl;
int GraphDim = 300;
int GraphMargin = 10;
int NbLabelX = 5;
int NbLabelY = 5;
```

```
out << "% Main Graph "<< endl;
out << "  "<< endl;
out << "  \\Draw"<< endl;
out << "  \\Scale(1,1)\\Ragged(1) "<< endl;
out << "  \\Define\\LineChart(1){"<< endl;
out << "       \\NextTable{"<< endl;
out << "       \\PenSize(#1pt) \\data(0,0){\\MoveTo}\\data(0,999){\\
LineTo} }"<< endl;
out << "       \\Table\\data }"<< endl;
out << "  \\MoveTo(0,-"<<GraphMargin<<")\\MarkLoc(a) \\MoveTo("<
<GraphDim<<",-"<<GraphMargin-.01<<")\\MarkLoc(b)"<< endl;
out << "  \\MoveTo(-"<<GraphMargin<<",0)\\MarkLoc(c) \\MoveTo(-"<
<GraphMargin-0.1<<","<<GraphDim<<")\\MarkLoc(d)"<< endl;
out << "  \\DrawRectAt(-"<<GraphMargin<<",-"<<GraphMargin<<","<
<GraphDim+GraphMargin<<","<<GraphDim+GraphMargin<<")"<< endl;
// Tracing the main curve ;

out.setf(ios_base::fixed,ios_base::floatfield);
out << " {\\color{NavyBlue} \\LineChart(3){  ";

for(index=0 ; index != (unsigned)NbCan; ++ index){
    out <<" "<< GraphDim*( (CanCoord[index]/AdjustScale) - MinX)/ EtX
;
    if(EBackgroundL == 1) out <<","<< GraphDim*(( (double)log(
FinalLikC[index]) ) - MinY)/ EtY ;
    else                  out <<","<< GraphDim*(( (double)log(
FinalLik[index]) ) - MinY)/ EtY ;
    if(index !=(unsigned)NbCan -1 ) out<<"&";
}
out <<"} }"<< endl;
// Tracing other curves if experience has been repeated ;
if(MatLik.size() > 1){
   for(unsigned int ind2=0 ; ind2 != MatLikC.size(); ++ind2){
       out <<"  \\LineChart(1){  ";
       for(unsigned int index=0 ; index != (unsigned)NbCan; ++ index){
           out <<" "<< GraphDim*( (CanCoord[index]/AdjustScale) -
MinX)/ EtX ;
           if(EBackgroundL == 1) out <<","<< GraphDim*(( (double)log(
 MatLikC[ind2][index] ) ) - MinY)/ EtY ;
           else                  out <<","<< GraphDim*(( (double)log(
 MatLik[ind2][index] ) ) - MinY)/ EtY ;
           if(index !=(unsigned)NbCan -1 ) out<<"&";
       }
       out <<"}"<< endl;
   }
}
//out.setf(0,ios_base::floatfield);
// X labels ;
out <<"{\\label \\Axis(a,b)(S1 ,";
double Increment = MinX;
for(index = 0 ; index != NbLabelX ; ++index){
    out<< Increment*100 ;
    if(index != NbLabelX -1) out<<" & ";
    Increment += EtX / (NbLabelX - 1);
}
out <<")"<< endl;
// Y labels ;
out <<"\\label \\Axis(c,d)(W1 ,";
```

```
Increment = MinY;
for(index = 0 ; index != NbLabelY ; ++index){
    out.width(3) ; out<< Increment ;
    if(index != NbLabelY -1) out<<" & ";
    Increment += EtY / (NbLabelY - 1);
}
out <<") }"<< endl;
out <<"\\MoveTo(150,-40)\\Text(--$r_T$--)"<< endl;
out <<"\\MoveTo(-40,260)\\Text(--$\\ln(L)$--)"<< endl;
// Show TIM position if known, or any reference point you want to show
 on graph.
if(RealP >-1 && RealI >-1){
   out <<"\\MoveTo("<< GraphDim*( ((RealP/100)/AdjustScale) - MinX)/
EtX <<",0)\\Text(--$\\color{RedOrange}\\star$--)"<< endl;
   //out <<"\\draw[style=TIMline]  plot coordinates {("<<xcoord<<",0)
("<<xcoord<<",10)};";
}
// Show Maximum Likelihood position.
out <<"\\MoveTo("<< GraphDim*( ( mlrt ) - MinX)/ EtX <<",0)\\Text(--$\
\color{NavyBlue}\\bigtriangleup$--)"<< endl;
// Affiche la position des marqueurs // Show markers position at the
bottom of the graph.
for(unsigned int idx=0 ; idx != Coord.size(); ++idx){
   out <<"\\MoveTo("<< GraphDim*( ( Coord[idx]/AdjustScale) - MinX)/
EtX <<",0)\\Text(--$\\vdots$--)"<< endl;
}
out <<"\\EndDraw "<< endl;

// Descriptives Statistics ;
if(MarkerT == 0 && StatDesc == 1){
    out << "\\vfill\\break"<< endl;
    // Graphe de |D'|, r^2
    // ********************************************************************
**********
    out <<"\\line(\\hfill"<< endl;
    SmallGraphNat(NPar, FPar, FRes, VDp , Coord , 100 , 10 , 4 , 5 , "$
|D'|$") ; //VecD VX , int GD , int GM , int Nx , int Ny , string label
  )
    out <<"\\hfill"<< endl;
    SmallGraphNat(NPar, FPar, FRes, Vr2 , Coord , 100 , 10 , 4 , 5 , "$
r^2$") ; //VecD VX , int GD , int GM , int Nx , int Ny , string label
 )
    out <<"\\hfill)"<< endl;
    // Graphe de d^2, OR
    // ********************************************************************
**********
    out <<"\\bigskip\\bigskip\\line(\\hfill"<< endl;
    SmallGraphNat(NPar, FPar, FRes, Vd2 , Coord , 100 , 10 , 4 , 5 , "$
d^2$") ; //VecD VX , int GD , int GM , int Nx , int Ny , string label
 )
    out <<"\\hfill"<< endl;
    SmallGraphNat(NPar, FPar, FRes, VOR , Coord , 100 , 10 , 4 , 5 , "$
OR$") ; //VecD VX , int GD , int GM , int Nx , int Ny , string label
  )
    out <<"\\hfill)"<< endl;

    // Matrice D' and r^2
    // ********************************************************************
```

```
*********************
   L = Coord.size();
   GraphDim = 100 ;
   if(L > 10) GraphDim = 200 ;
   if(L > 15) GraphDim = 300 ;
   double UnitS = (double)GraphDim / (double)L ;
   //cerr<<"UnitS="<<UnitS<<endl;
   out <<" \\font\\vsmall=cmtt10 scaled 400"<<endl;
   out <<"  \\Draw"<< endl;
   out <<"  \\Scale(1,1)\\Ragged(1) "<< endl;
   unsigned int i;
   for(i=0 ; i!= L+1 ; ++i){
       out <<"\\LineAt(0,"<<i*UnitS<<","<<GraphDim<<","<<i*UnitS<<")"<↙
<endl;
       out <<"\\LineAt("<<i*UnitS<<",0,"<<i*UnitS<<","<<GraphDim<<")"<↙
<endl;
       if(i != L){
          out <<"\\MoveTo(-15,"<<i*UnitS + UnitS/2<<")\\Text(--\\small↙
"<<L-i<<"--)"<<endl;
          out <<"\\MoveTo("<<i*UnitS + UnitS/2<<","<<GraphDim+15<<")\\↙
Text(--\\small"<< i+1 <<"--)"<<endl;
        }
   }
   for(i=0 ; i!= L ; ++i){
       for(unsigned int j=0 ; j!= L ; ++j){
           //\definecolor{Gray}{cmyk}{0,0,0,0.2}
           //\MoveTo(50,50)\color{Gray}\PaintRect(30,30)
           //cerr <<"[i="<<i<<" | j="<<j<<"] [L-i="<< L-i <<" | j+1=" ↙
<< j+1 <<"] ("<< L-i-1<<","<<j<<")" ;
           //if(j <= L-i-1) cerr <<"-> r2:"<< int(VVr2[L-i-1][j]*     ↙
10000)<< endl;
           //if(j > L-i-1)  cerr <<"-> d':"<< int(VVDp[j][L-i-1]*     ↙
10000)<< endl;

           if(j <= L-i-1){
              if(VVr2[L-i-1][j] < 1e-4) VVr2[L-i-1][j] = 0; // to     ↙
avoid wrting xe-5 or less in the Tex code ;
              out <<"\\definecolor{Gray}{cmyk}{0,0,0,"<<VVr2[L-i-1][j]↙
<<"}"<<endl;
              out <<"\\MoveTo("<<j*UnitS<<","<<i*UnitS<<")\\color      ↙
{Gray}\\PaintRect("<<UnitS<<","<<UnitS<<")"<<endl;
              out <<"\\MoveTo("<<j*UnitS+UnitS/2<<","<<i*UnitS + UnitS↙
/2<<")";
              if(VVr2[L-i-1][j]>0.5)  out<<"\\Text(--\\vsmall\\color  ↙
{white}"<< int(VVr2[L-i-1][j]*1000)<<"--)"<<endl;
              if(VVr2[L-i-1][j]<=0.5) out<<"\\Text(--\\vsmall\\color  ↙
{black}"<< int(VVr2[L-i-1][j]*1000)<<"--)"<<endl;
           }
           if(j > L-i-1){
              if(VVDp[j][L-i-1] < 1e-4) VVDp[j][L-i-1] = 0; // to     ↙
avoid wrting xe-5 or less in the Tex code ;
              out <<"\\definecolor{Gray}{cmyk}{0,0,0,"<<VVDp[j][L-i-1]↙
<<"}"<<endl;
              out <<"\\MoveTo("<<j*UnitS<<","<<i*UnitS<<")\\color      ↙
{Gray}\\PaintRect("<<UnitS<<","<<UnitS<<")"<<endl;
              out <<"\\MoveTo("<<j*UnitS+UnitS/2<<","<<i*UnitS + UnitS↙
/2<<")";
              //out <<"\\MoveTo("<<j*UnitS+UnitS/2<<","<<i*UnitS +     ↙
```

```
    UnitS/2<<")\\Text(--\\small"<<int(VVDp[j][L-i-1]*1000)<<"--)"<<endl;
                    if(VVDp[j][L-i-1]>0.5)  out<<"\\Text(--\\vsmall\\color
    {white}"<< int( VVDp[j][L-i-1]*1000)<<"--)"<<endl;
                    if(VVDp[j][L-i-1]<=0.5) out<<"\\Text(--\\vsmall\\color
    {black}"<< int( VVDp[j][L-i-1]*1000)<<"--)"<<endl;
                }
            }
        }
        out <<"  \\MoveTo("<<GraphDim+15<<","<<GraphDim/2<<")\\Text(--\\
    color{black}$|D'|$--)"<<endl;
        out <<"  \\MoveTo("<<GraphDim/2<<",-15)\\Text(--\\color{black}$r^2$
    --)"<<endl;
        out <<"  \\EndDraw"<< endl;
    }

    out <<"\\bye  "<< endl;

    //PrintV(Coord,"Coord");
    //PrintV(VDp,"VDp");
}
```

Bibliographie

Avery, O. T., MacLeod, C. M., McCarty, M., 1944 « Studies on the chemical nature of the substance inducing transformation of pneumococcal types. Induction of transformation by a deoxyribonucleic acid fraction isolated from pneumococcus type III », *J. Exp. Med.*, 1944, 79(2) :137-157.

Bross, I. D. J., « Taking a covariable into account. », 1964, *Journal of the American Statistical Association*, 59, 725-736.

Dudewicz, E.J., « Introduction to statistics and probability. », *Holt, Rinehart and Winston, New York.*, 1976.

Edgington, E. S., « Randomization tests. », 1995, *New York : Marcel Deckker, Inc.*.

Efron, B., « Bootstrap Methods : Another Look at the Jackknife », *The Annals of Statistics*, 1979, 7 (1) : 1–26.

Efron, B., « The jackknife, the bootstrap, and other resampling plans. », *Society of Industrial and Applied Mathematics CBMS-NSF Monographs*, 1982, 38.

Efron, B., « Bootstrap confidence intervals for a class of parametric problems. », *Biometrika*, 1985.

Efron, B., Tibshirani, R. J., « An introduction to the bootstrap. », *New York : Chapman & Hall*, 1993.

Fisher, R. A., « The Logic of Inductive Inference, with discussion », *Journal of the Royal Statistical Society*, 1935, 98, 39-82.

Fisher, R. A., « Design of Experiments. », 1935, *New York : Hafner*.

Fisher, R.A., « Statistical Methods and Scientific Inference. », 1993, *Macmillan, New York, third edition.*, Reprinted in Fisher (1990).

Fumeron, F., « Polymorphismes des enzymes du métabolisme de l'alcool », Cholé-Doc, *Le bulletin de liaison des banques de données NUTRIPID et CERINUT*, numero 67, Septembre/Octobre 2001.

Good, P., « Permutation tests : A practical guide to resampliing methods for testing hypotheses. », *New York : Springer-Verlag.*, 1994.

Griffiths, R., Marjoram, P., « Ancestral inference from samples of DNA sequences with recombination. », *Journal Comp Biol*, 1996, 3 :479-502.

Griffiths, R. C., Tavaré, S., « Simulating Probability Distributions in the Coalescent », *Theoret. Population Biol.*, Volume 46, Issue 2, October 1994, pp. 131-159.

Gusella, J. F.,Wexler, N. S., Conneally, P. M. et al., « A polymorphic DNA marker genetically linked to Huntington's disease », *Nature* , 1983, 306 : 234-238.

Hall, M. A., Norman, P. J., Thiel, B., Tiwari, H., Peiffer, A., Vaughan, R. W., Prescott, S., Leppert, M., Schork, N. J., Lanchbury, J. S., Wooster et al., 1995 « Quantitative-Trait Loci on Chromosomes 1, 2, 3, 4, 8, 9, 11, 12, and 18. Control Variation in Levels of T and B Lymphocyte Subpopulations », *Am J Hum Genet.*, 2002 May ; 70(5) : 1172–1182.

Horikawa, Y. et al., « Genetic variation in the gene encoding calpain-10 is associated with type 2 diabetes mellitus. », *Nat. Genet.*, 200, 26 : 163 - 75.

Hudson, R. R., « Properties of a neutral allele model with intragenic recombination. », *Theoret. Popul.Biol.*, 1983, 23 :183-201.

Hudson, R. R., « Gene genealogies and the coalescent process. », *In Oxford Surveys in Evolutionary Biology, ed. Futuyma D, Antonovics J, Oxford University Press*, 1991, 7 :1-44.

Kimura, M., « Evolutionary rate at the molecular level », *Nature*, 1963, 217 : 624-625.

Kingman, J. F. C., « The coalescent. », *Stochastic Processes and their Applications*, 1982, 13 :235–248.

Kuhner, M., Yamato, J., Felsenstein, J., « Estimating effective population size and mutation rate from sequence data using Metropolis-Hastings sampling. », *Genetics*, 1995, 140 :1421-1430.

Lander, E.S., Kruglyak, L., « Genetic dissection of complex traits : Guidelines for interpreting and reporting linkage results. », 1995, *Nat. Genet.*, 11 :241-247.

Larribe, F., Lessard S., Schork, N.J., « Gene mapping via the ancestral recombination graph », *Theoret. Popul.Biol.*, 2002, 62.2.215-229.

Manly, B. F. J., « Randomization, Bootstrap, and Monte Carlo methods in biology », 1997, *(2nd ed.). Boca Raton, Fl : Chapman & Hall.*

Mendel, G., 1865 « Versuch über Plfanzenhybriden », *Verhandlungen des naturforschenden Vereines in Brünn*, Bd. IV für das Jahr 1865, pp. 3-47.

Mooney, C.Z., Duval, R., « Bootstrapping : A nonparametric approach to statistical inference. », *Sage University paper series on REFERENCES quantitative application in the social sciences*, 1993, 95. Sage University, London.

Pitman, J., « Significance tests which may be applied to samples from any populations. », *Journal of the Royal Statistical Society*, 1937, 4, 119-130.

Stephens, M. and Donnelly, P., « Inference in Molecular Population Genetics. », *Journal of the Royal Statistical Society*, 2000, Series B, 62, 605–655.

Swanepoel, J.W.H., « A note on proving that the (modified) bootstrap works. », *Community Statistics*, 1986, 12, 2059-2083.

Wooster et al., « Identification of the breast cancer susceptibility gene BRCA2 », *Nature* , 28 December 1995, 378, 789 - 792.