RAPPORT ALGORITHMIQUE AVANCÉE

Problème de détection des doublons

Préparé par :

OUGDAL Achraf
AROUINI Youssef

Encadré par :

Mr. HAFIDI Imad

SOMMAIRE

INTRODUCTION

Problématique :

Comment gérer les doublons ? Cette question est récurrente lorsque l'on souhaite traiter et analyser des données, que ce soit pour les compter, les filtrer, les regrouper ou les supprimer.
La gestion des doublons dans une base de données est très importante. Une mauvaise identification peut nuire aux performances et à l'efficacité des actions menées au sein d'un organisme. Les doublons nuisent à la productivité des équipes commerciales, marketing, de téléprospection, etc. D'après l'étude de 2013 de Dynamics Markets, les doublons représentent 10% du budget gaspillé.

Objectif :

Dans ce projet, On propose trois algorithmes (Algorithme Naïve, Algorithme avec tri, Algorithme Hash) en utilisant les quatre distances (Jaro-Winkler, LevenShtein, N-Gramme, Soundex) afin de répondre à cette problématique.

PRESENTATION DES STRUCTURES DE DONNEES UTILISES

Tableaux dynamiques :

En informatique, un tableau est une structure de données représentant une séquence finie d'éléments auxquels on peut accéder efficacement par leur position, ou indice, dans la séquence. C'est un type de conteneur que l'on retrouve dans un grand nombre de langages de programmation.
Le temps d'accès à un élément par son index est constant, quel que soit l'élément désiré. Cela s'explique par le fait que les éléments d'un tableau sont contigus dans l'espace mémoire. Ainsi, il est possible de calculer l'adresse mémoire de l'élément auquel on veut accéder, à partir de l'adresse de base du tableau et de l'index de l'élément. L'accès est immédiat, comme il le serait pour une variable simple.
Les limites d'une telle structure viennent de son avantage. Un tableau étant représenté en mémoire sous la forme de cellules contiguës, les opérations d'insertion et de suppression d'élément sont impossibles, sauf si on crée un nouveau tableau, de taille plus grande ou plus petite. Il est alors nécessaire de copier tous les éléments du tableau original dans le nouveau tableau, puis de libérer l'espace mémoire alloué à l'ancien tableau. Cela fait donc beaucoup d'opérations et oblige certains langages fournissant de telles possibilités à implémenter leurs tableaux, non pas sous la forme traditionnelle (cellules adjacentes), mais en utilisant une liste chaînée, ou une combinaison des deux structures pour améliorer les performances.

Tas :

En informatique, un tas est une structure de données de type arbre qui permet de retrouver directement l'élément que l'on veut traiter en priorité. C'est un arbre binaire presque complet ordonné.
Un arbre binaire est dit presque complet si tous ses niveaux sont remplis, sauf éventuellement le dernier, qui doit être rempli sur la gauche. Ses feuilles sont donc à la même distance minimale de la racine, plus ou moins 1.
Le Tas est une structure de données utile pour le tri par tas, il permet de construire une file de priorité efficace.
Le temps d'exécution su Tri par Tas est O(n log n).

Tableaux de Hachage :

Une table de hachage est, en informatique, une structure de données qui permet une association clé-valeur, c'est-à-dire une implémentation du type abstrait tableau associatif ; en particulier, l'implémentation d'une table des symboles lorsque les clés sont des chaînes de caractères.
Dans une table de hachage, on calcule la position d'un élément à partir de sa propre valeur.
Le principe de hachage n'est pas parfait, il pose certains problèmes :

- Il faut nécessairement 0 < h(clé) < taille du tableau
- Problèmes de collision : Plusieurs entités ont le même indice.

Dans un tableau de hachage le temps de recherche est constant, l'accès à un élément est indépendant de la taille : Recherche O(1).

PRESENTATION DES QUATRE DISTANCES

Livenshtein :

La distance de Levenshtein est une distance, au sens mathématique du terme, donnant une mesure de la différence entre deux chaînes de caractères. Elle est égale au nombre minimal de caractères qu'il faut supprimer, insérer ou remplacer pour passer d'une chaîne à l'autre.
Elle a été proposée par Vladimir Levenshtein en 1965. Elle est également connue sous les noms de distance d'édition ou de déformation dynamique temporelle, notamment en reconnaissance de formes et particulièrement en reconnaissance vocale.
Cette distance est d'autant plus grande que le nombre de différences entre les deux chaînes est grand. La distance de Levenshtein peut être considérée comme une généralisation de la distance de Hamming. On peut montrer en particulier que la distance de Hamming est un majorant de la distance de Levenshtein.
L'algorithme ci-dessous, dû à Wagner et Fischer (1974), permet de calculer la distance de Levenshtein entre deux chaînes de caractères courtes.

```
entier DistanceDeLevenshtein(caractère chaine1[1..longueurChaine1], caractère
chaine2[1..longueurChaine2])
  // d est un tableau de longueurChaine1+1 rangées et longueurChaine2+1 colonnes
  // d est indexé à partir de 0, les chaînes à partir de 1
  déclarer entier d[0..longueurChaine1, 0..longueurChaine2]
  // i et j itèrent sur chaine1 et chaine2
  déclarer entier i, j, coûtSubstitution

  pour i de 0 à longueurChaine1
     d[i, 0] := i
  pour j de 0 à longueurChaine2
     d[0, j] := j

  pour i de 1 à longueurChaine1
    pour j de 1 à longueurChaine2
       si chaine1[i-1] = chaine2[j-1] alors coûtSubstitution := 0
                          sinon coûtSubstitution := 1
        d[i, j] := minimum(
                        d[i-1, j  ] + 1,              // effacement du nouveau caractère de chaine1
                        d[i,   j-1] + 1,              // insertion dans chaine2 du nouveau caractère de chaine1
                        d[i-1, j-1] + coûtSubstitution   // substitution
                      )
renvoyer d[longueurChaine1, longueurChaine2]
```

Soundex :

Soundex est un algorithme phonétique d'indexation de noms par leur prononciation en anglais britannique. L'objectif de base est que les noms ayant la même prononciation soient codés avec la même chaîne de manière à pouvoir trouver une correspondance entre eux malgré des différences mineures d'écriture. Soundex est le plus largement connu des algorithmes phonétiques et est souvent utilisé incorrectement comme synonyme de « algorithme phonétique ».
L'algorithme soundex est très utilisé en informatique pour corriger les erreurs orthographiques et est disponible dans la plupart des SGBD.

N-gramme :

Un n-gramme est une sous-séquence de n éléments construite à partir d'une séquence donnée. L'idée semble provenir des travaux de Claude Shannon en théorie de l'information. Son idée était que, à partir d'une séquence de lettres donnée, il est possible d'obtenir la fonction de vraisemblance de l'apparition de la lettre suivante. À partir d'un corpus d'apprentissage, il est facile de construire une distribution de probabilité pour la prochaine lettre avec un historique de taille n. Cette modélisation correspond en fait à un modèle de Markov d'ordre n où seules les n dernières observations sont utilisées pour la prédiction de la lettre suivante. Ainsi un bigramme est un modèle de Markov d'ordre 2.

Jaro-Winkler :

La distance de Jaro-Winkler mesure la similarité entre deux chaînes de caractères. Il s'agit d'une variante proposée en 1999 par William

E. Winkler, découlant de la distance de Jaro (1989, Matthew A. Jaro) qui est principalement utilisée dans la détection de doublons.
Le résultat est normalisé de façon à avoir une mesure entre 0 et 1, donc zéro représente l'absence de similarité et 1, l'égalité des chaines comparées.
Cette mesure est particulièrement adaptée au traitement de chaînes courtes comme des noms ou des mots de passe.

ALGORITHME DE RECHERCHE

En informatique, un algorithme de recherche est un type d'algorithme qui, pour un domaine, un problème de ce domaine et des critères donnés, retourne en résultat un ensemble de solutions répondant au problème.

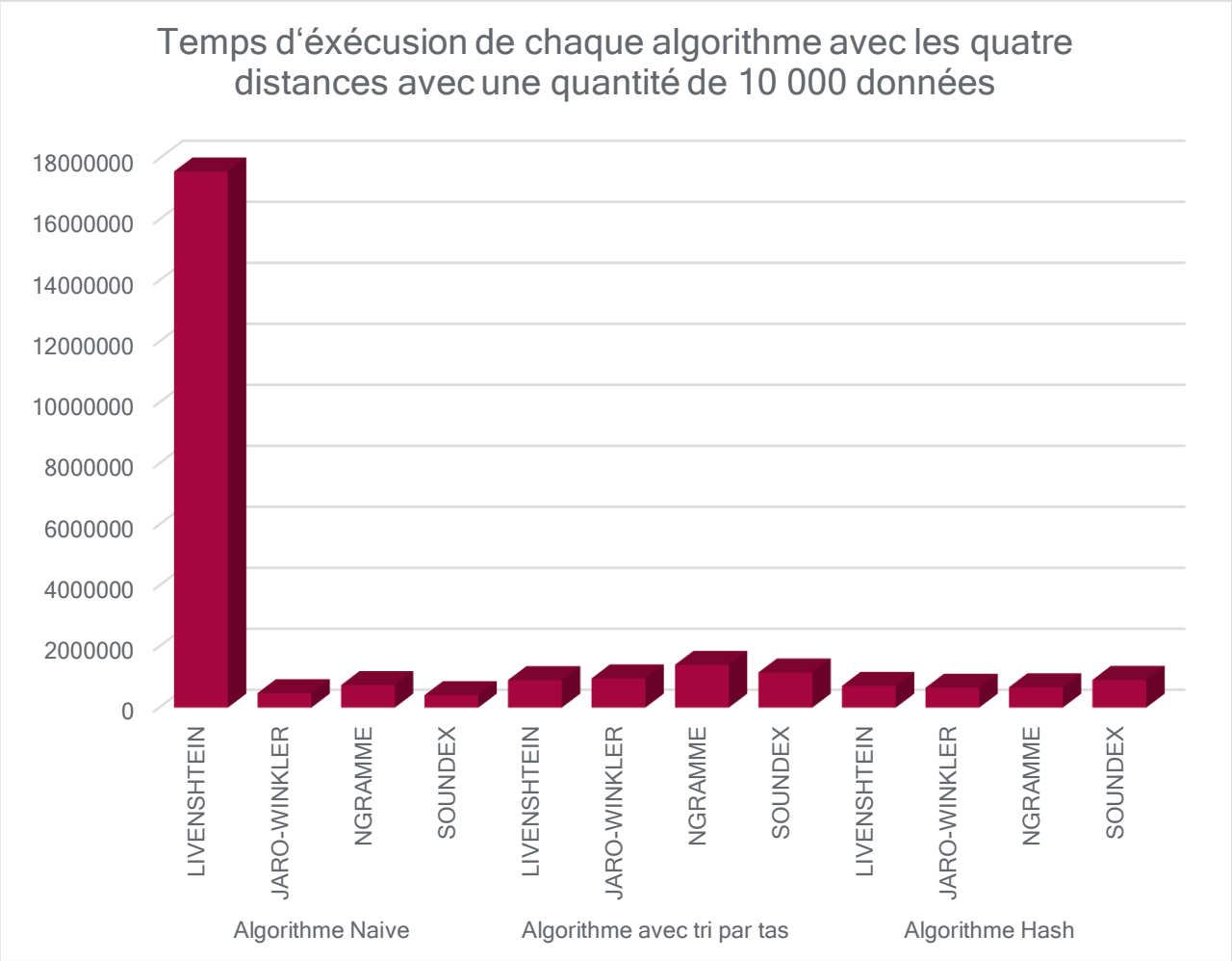
Supposons que l'ensemble de ses entrées soit divisible en sous-ensemble, par rapport à un critère donné, qui peut être, par exemple, une relation d'ordre. De façon générale, un tel algorithme vérifie un certain nombre de ces entrées et retourne en sortie une ou plusieurs des entrées visées.

L'ensemble de toutes les solutions potentielles dans le domaine est appelé espace de recherche.

Struct/opération	Rechercher
SET	O(n)
TAS	O(n)
HASHTABLE	O(n)

SIMULATION ET ANALYSE DES RESULTATS NUMERIQUES

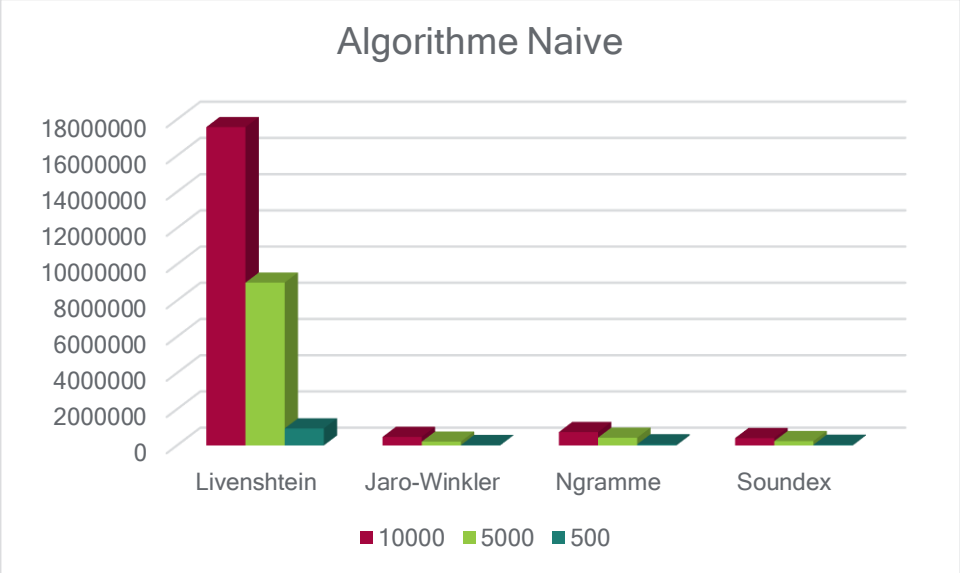
Graphe représente l'évolution du temps d'exécution de chaque algorithme en fonction de la taille des données.



Le graphe ci-dessus décrit le temps d'exécution de chaque distance pour les 3 algorithmes.
Pour l'algorithme naïve, on constate que la distance de Levenshtein a pris un temps considérablement grand pour faire la comparaison puis la suppression des mots doublons. Cependant, les autres distances étaient rapides lors de l'exécution.
Pour l'algorithme du tri par tas et du Hashage, les résultats sont très proche, sauf que, pour l'algorithme du tri par tas, la distance N-Gram était la plus lente, et pour l'algorithme de Hashage, c'était la distance de Soundex.

Le point commun entre les 3 Graphes est la rapidité de la distance de Jaro Winkler par rapport aux autres distances, c'était la distance la plus rapide lors de l'exécution d'un algorithme lent et couteux.

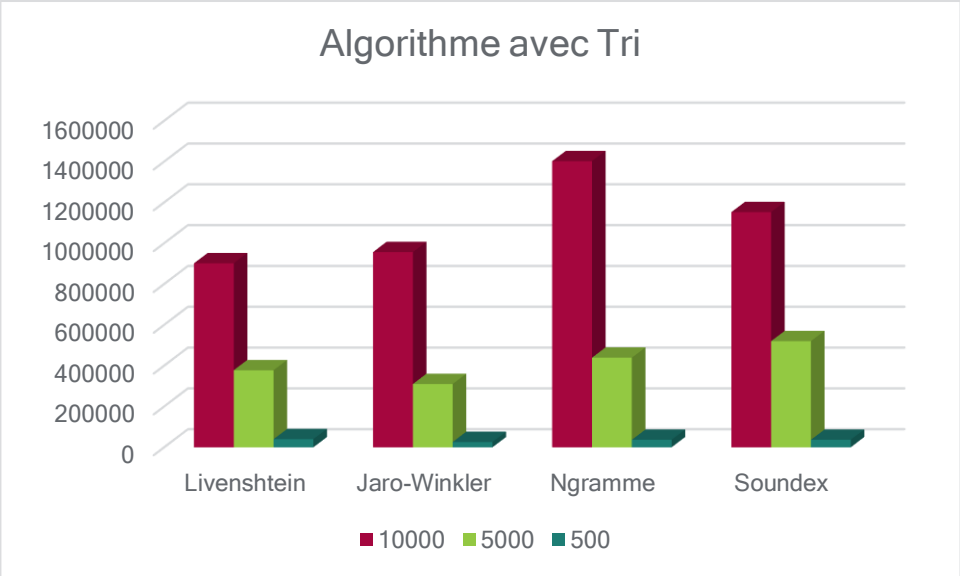
Graphe représente l'évolution du temps d'exécution de l'algorithme naïve en fonction de la taille des données :



Le graphe ci-dessus montre l'évolution du temps d'exécution de l'algorithme naïve en fonction de la taille des données avec les distances.

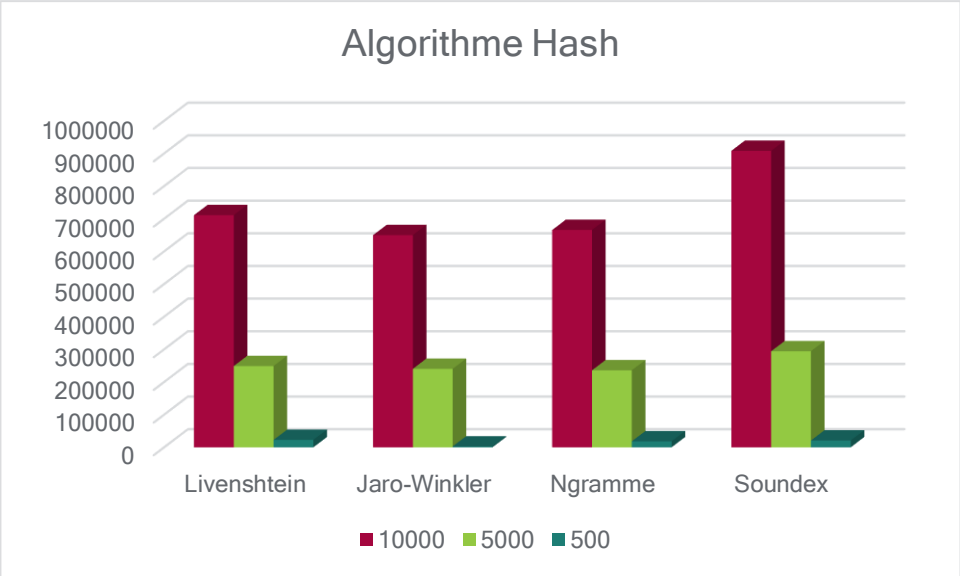
Pour les distances : Ngramme, Jaro Winkler et Soundex, le temps d'exécution évolue d'une manière proportionnelle par rapport à la taille des données, par contre pour la distance de Levenshtein, le temps d'exécution évolue d'une manière vaste et illogique.
Donc on peut dire que pour l'algorithme naïve, la distance de Levenshtein est la moins stable.
De plus, la distance de Jaro Winkler reste la plus rapide.

Graphe représente l'évolution du temps d'exécution de l'algorithme avec Tri en fonction de la taille des données :



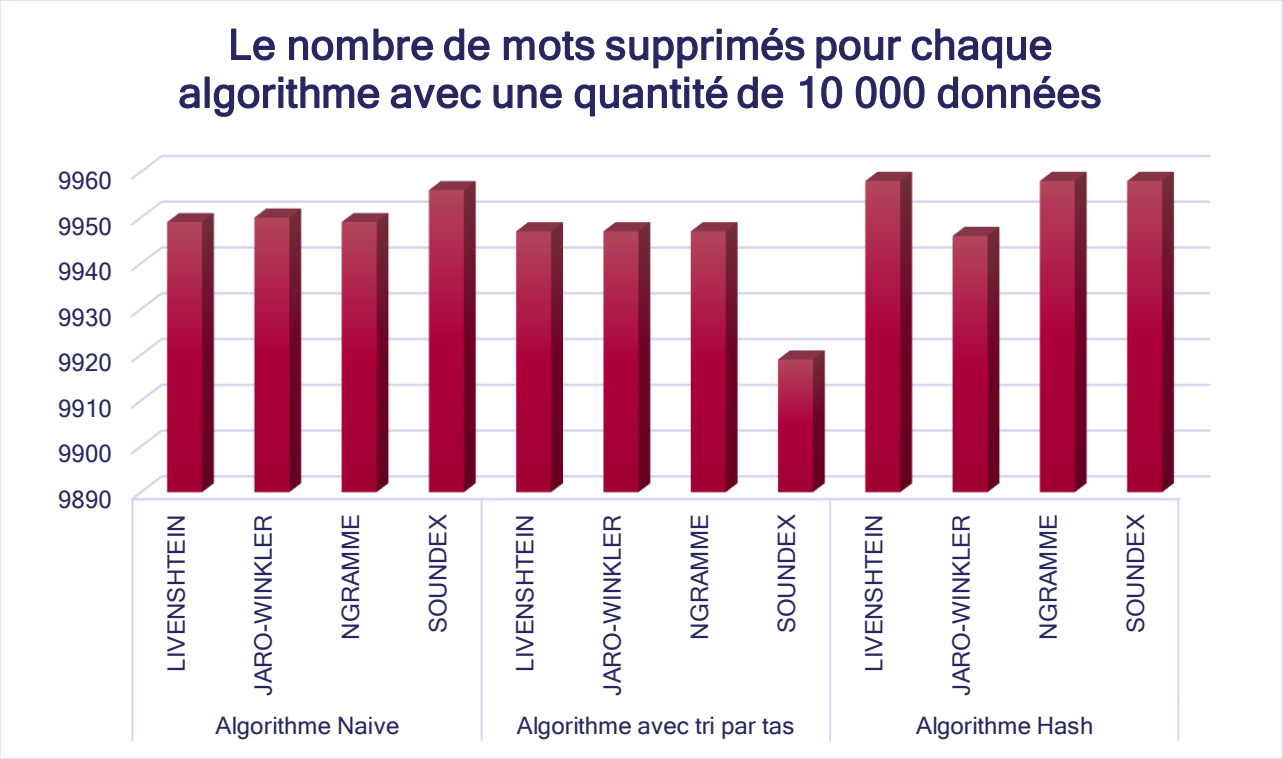
Pour l'algorithme du Tri par Tas, on constate que le temps d'exécution de toutes les distances évolue d'une manière plus ou moins similaire, sauf Pour la distance de N-gramme qui, à un fichier de 10 000 mots, a pris un temps plus grand par rapport aux autres distances.

Graphe représente l'évolution du temps d'exécution de l'algorithme Hash en fonction de la taille des données.



Enfin, pour l'algorithme de Hashage, on constate que la distance de Soundex a pris un temps considérablement grand par rapport aux autres distances lors de l'exécution de l'algorithme avec un fichier de 10 000 données.

Graphe représente le nombre de mots supprimés pour chaque Algorithme :



Le graphe ci-dessus décrit le nombre des mots supprimés pour chaque Algorithme avec une quantité de 10 000 données. On constate que, les résultats sont un peu près proches sauf pour la distance de Soundex pour l'algorithme avec tri par tas qui a supprimé moins de données par rapport aux autres distances dans tous les algorithmes. Ainsi on constate la stabilité de la distance de Jaro Winkler qui supprime pour chaque algorithme le même nombre de mots.

D‘après les graphes qui représentent l‘évolution du temps d‘exécution et le nombre de mots supprimés pour chaque algorithme, il est bien évident que l‘algorithme Hash est le plus efficace soit au niveau du temps d‘exécution soit au niveau du quantité de données supprimées. Et surtout si les distances utilisées soient N-Gramme ou Levenshtein.
Ce résultat dû à l‘implémentation de la structure utilisée qui permet une association clé-valeur, ce qui facilite l‘accès et la suppression des données.

Conclusion

L‘objectif de ce projet était de pouvoir faire une étude et une comparaison entre les structures des données (les Tableaux dynamiques, Tas, les Tables de hachage) au niveau de la complexité, temps d‘exécution et l‘efficacité. Et cela en programmant trois algorithmes qui sert à répondre aux problèmes de gestion des doublons à l‘aide des quatre distances (Jaro-Winkler, Levenshtein, N-Gramme et Soundex).

En conclusion, et d’après les résultats obtenus lors des tests, on peut dire que la distance de Jaro Winkler est la plus efficace grâce à sa rapidité lors des 3 algorithme, et sa stabilité.

Cela ne dit pas que les autres distances sont inutiles, chaque distance peut être la meilleure si utilisée dans un domaine qui la convient. Par exemple, dans les applications Web, la distance de levenshtein peut être utile pour les recherches où l'utilisateur tape un mot de manière incorrecte, et vous devrez rechercher des correspondances proches au lieu de correspondances exactes.