

MATH0495: Projet : Wilcoxon

Julien GUSTIN

Table des matières

1	Analyse théorique	2
2	Utilisation du programme	3
3	Limites du programme	4
3.1	Limites avec une durée infinis	4
3.2	Limites avec une durée finie (15 secondes)	5

1 Analyse théorique

Le test de Wilcoxon proposé par Frank Wilcoxon en 1945 est un test statistique non paramétriques qui permet de tester l'hypothèse selon laquelle les médianes de chacun de deux groupes de données sont proches. Ce test calcul le score de deux "population" ou deux variables différentes ayant un

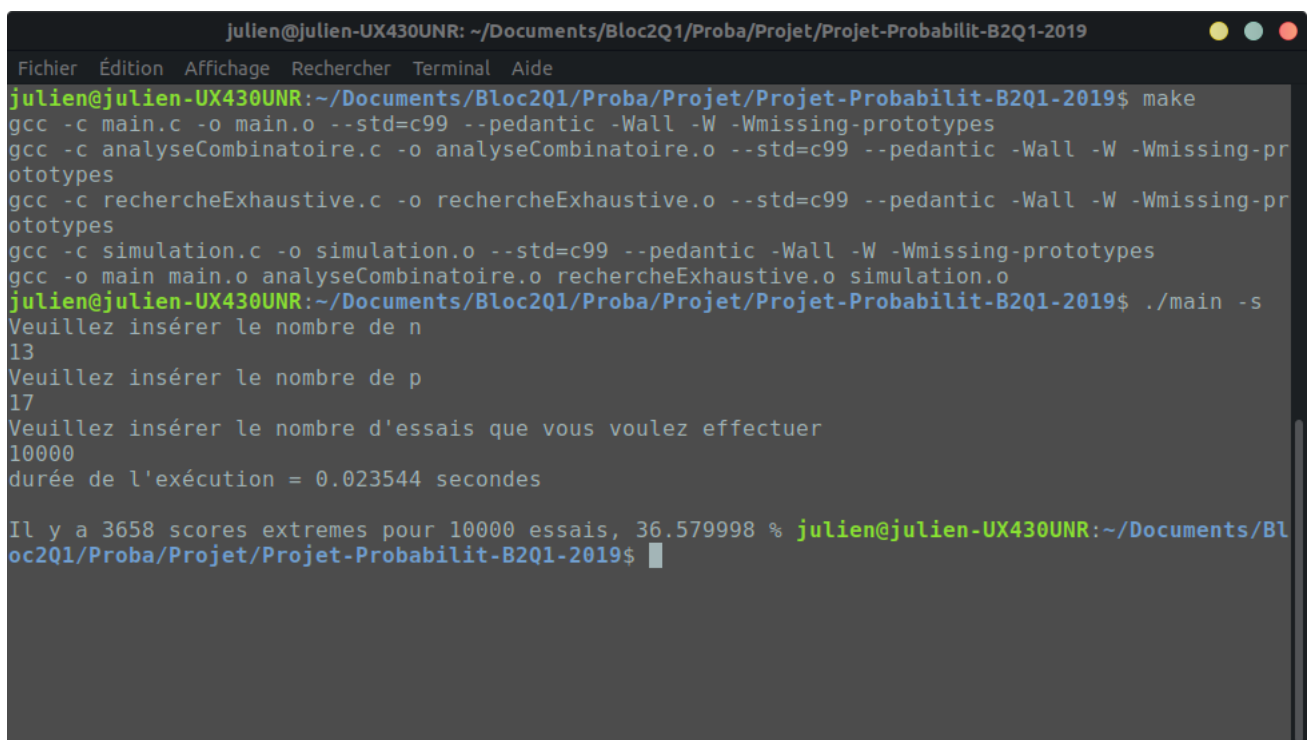
2 Utilisation du programme

Pour utiliser le programme il suffit de suivre ces consignes :

1. Ouvrir la console dans le répertoire du fichier et écrivez `make` ↔;
2. ☐ Compiler avec `./MAIN -S` pour faire une simulation ↔
☐ Compiler avec `./MAIN -C` pour l'analyse combinatoire ↔
☐ Compiler avec `./MAIN -E` pour la recherche exhaustive ↔
3. ☐ Insérer le nombre de n (première variable) ↔
☐ Insérer le nombre de p (seconde variable) ↔
☐ (Seulement pour la simulation, insérer le nombres d'*essais* à effectuer ↔)

⚠ voir limites du programmes (.3)

4. Il ne reste plus qu'à appuyer sur escape et le programme ce lance. "Scores extrêmes" est le nombre de fois où on obtient un score extrême, l'écart de score entre "l'équipe gagnante ou perdante" (n ou p) est ≥ 45
5. Exemple d'utilisation :



```

julien@julien-UX430UNR: ~/Documents/Bloc2Q1/Proba/Projet/Projet-Probabilit-B2Q1-2019
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
julien@julien-UX430UNR:~/Documents/Bloc2Q1/Proba/Projet/Projet-Probabilit-B2Q1-2019$ make
gcc -c main.c -o main.o --std=c99 --pedantic -Wall -W -Wmissing-prototypes
gcc -c analyseCombinatoire.c -o analyseCombinatoire.o --std=c99 --pedantic -Wall -W -Wmissing-pr
ototypes
gcc -c rechercheExhaustive.c -o rechercheExhaustive.o --std=c99 --pedantic -Wall -W -Wmissing-pr
ototypes
gcc -c simulation.c -o simulation.o --std=c99 --pedantic -Wall -W -Wmissing-prototypes
gcc -o main main.o analyseCombinatoire.o rechercheExhaustive.o simulation.o
julien@julien-UX430UNR:~/Documents/Bloc2Q1/Proba/Projet/Projet-Probabilit-B2Q1-2019$ ./main -s
Veuillez insérer le nombre de n
13
Veuillez insérer le nombre de p
17
Veuillez insérer le nombre d'essais que vous voulez effectuer
10000
durée de l'exécution = 0.023544 secondes

Il y a 3658 scores extremes pour 10000 essais, 36.579998 %
julien@julien-UX430UNR:~/Documents/BL
oc2Q1/Proba/Projet/Projet-Probabilit-B2Q1-2019$
```

FIGURE 1 – Exemple

3 Limites du programme

3.1 Limites avec une durée infinie

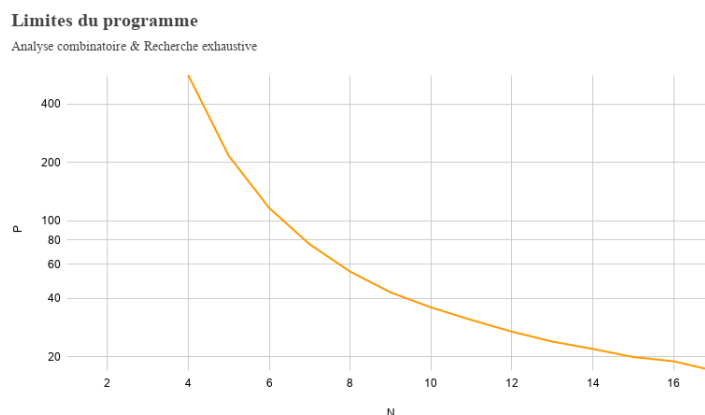


FIGURE 2 – Graphique montrant les limites du programme

Temps infinis	
N	P
1	4294967295
2	(...)
3	(...)
4	564
5	217
6	117
7	76
8	55
9	43
10	36
11	31
12	27
13	24
14	22
15	20
16	19
17	17

FIGURE 3 – Tableau sur le quel le graphique est basé (pour un p compris entre deux valeurs de ce tableau il est plus judicieux de choisir un n borné vers le bas, exemple : $p = 18$ et $n = 17$

Pour un temps infinis le programme qui est compilé en **analyse combinatoire** ou **recherche exhaustive** peut gérer aux maximums ces cas ci (voir tableau) en effet la plus part des valeurs, dont celle qui stocke le nombres de "n-p mots" sont des UNSIGNED INT ainsi le nombre de mots différents possible est : $2^{32} - 1$. Pour trouver ces résultats il a juste fallu calculer les n et p maximum pour les quels $C_{n+p}^n \leq 2^{32} - 1$. Bien évidemment n et p sont interchangeables.

Pour la **simulation**, c'est un peu différent, la limite réside dans le nombres d'essais et doit donc être inférieur ou égale à $2^{32} - 1$.

Cependant en pratique le programme n'est pas tellement efficace pour des nombres aussi élevé... Prenons un n égale à 13 et un p égale à 24 par exemple, l'exécution du programme prends ~ 143 secondes... Ce qui est énorme et donc pas tellement utilisable. Dans ce genre de situation la simulation bien que moins "correcte" sera plus intéressante, en effet pour un n et p équivalent et un nombres d'essais élevé nous tombons sur une réponses presque équivalente en moins de 6 secondes.

```

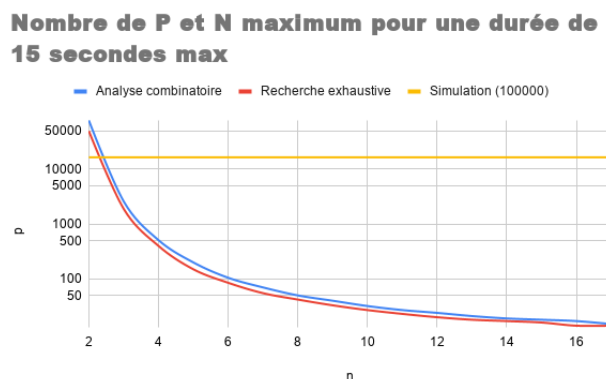
juliens@juliens-UX430UNR:~/Documents/Bloc201/Proba/Projet/Projet-Probabilit-B201-201
$5 ./main -c
Veuillez insérer le nombre de n
13
Veuillez insérer le nombre de p
24
Durée de l'exécution = 142.970169 secondes
Il y a exactement 3562467300 mots, parmi lesquels 1707329636 donnent lieu à des se
bres "extremes" 47.925484 %

```

FIGURE 4 – Exemple d'exécution

3.2 Limites avec une durée finie (15 secondes)

Le graphique ci-dessous montres les n et p maximum pour les quels la compilation prends moins de quinze secondes, ce qui rend ainsi le programme beaucoup plus utilisable.



Nous pouvons constater des phénomènes intéressant, déjà l'analyse combinatoire et la recherche exhaustive sont assez proche, bien que l'analyse combinatoire reste la plus efficace des deux, nous pouvons utiliser des nombres plus élevé pour un résultat plus rapide. Cependant la simulation (1000000 essais) qu'importe le nombre de n ou $p \leq 16500$. La compilation prendra moins de quinze secondes et restera à ce temps fixe, ce qui peut être fortement utile si nous recherchons des approximations.

Temps <= 15 sec (machine université)							
Recherche exhaustive			Analyse combinatoire			Simulation (100000)	
N	P		N	P		N	P
1	50000		1	78000		1	16500
2	1900		2	2600		2	16500
3	400		3	510		3	16500
4	150		4	200		4	16500
5	85		5	105		5	16500
6	55		6	70		6	16500
7	42		7	50		7	16500
8	33		8	40		8	16500
9	27		9	32		9	16500
10	23		10	27		10	16500
11	20		11	24		11	16500
12	18		12	21		12	16500
13	17		13	19		13	16500
14	16		14	18		14	16500
15	14		15	17		15	16500
16	14		16	15		16	16500
17	13		17	15		17	16500

FIGURE 5 – Tableau sur le quel le graphique est basé
Tous ces tests ont été fait sur les machines de l'université et montre ainsi les limites du programme.