

Maths 2 : Devoir Maison de statistiques appliquées

Marco Freire

Clément Legrand-Duchesne

25 mars 2018

Résumé

Mots-clefs :
Classification ACM :

Plan

1	Description du jeu de données	1
2	Pourcentage de lignes de code dupliquées	2
3	Comparaison du nombre de lignes dupliquées en C et en C++	2
4	Intervalle de confiance du taux de lignes dupliquées	3
5	Lien entre la longueur du code et le nombre de warning à la compilation	3
6	Questions personnelles	4
6.1	Lien entre le nombre et le type de warning à la compilation et la qualité de la gestion mémoire.	4
6.2	Autre question	4

1 Description du jeu de données

Le jeu de données fournis contient les mesures de la qualité des programmes couramment utilisés par les biologistes. Seize programmes sont ainsi examinés, selon des critères tels que le nombre de lignes ou de blocs dupliqués, le nombre et le type de warning lors de la compilation ou encore le statut donné par valgrind sur la gestion mémoire.

Un premier graphique nous permet de compter le nombre de programme écrit dans chaque langage du jeu de donnée (figure ??).

Nous avons ensuite décider de représenter le nombre de lignes de codes dupliquées par programme (figure 2). Afin que la représentation soit plus lisible, nous avons utilisé la commande `order` pour ordonner les colonnes par ordres décroissants de valeurs. De plus, nous avons affiché les noms des programmes en vertical, afin d'améliorer la lisibilité (grâce à l'argument `las = 2` de `barplot`). Il est de plus nécessaire de faire attention à ne pas prendre en compte les programmes pour lesquels cette valeur n'est pas renseignée (c'est le cas de *FDPPDIV* par exemple).

De même, nous avons créer un histogrammes contenant le nombre de blocs dupliqués par programme (figure 3).

Enfin, nous avons généré un dernier graphique affichant le nombre de programmes par domaine (figure 4).

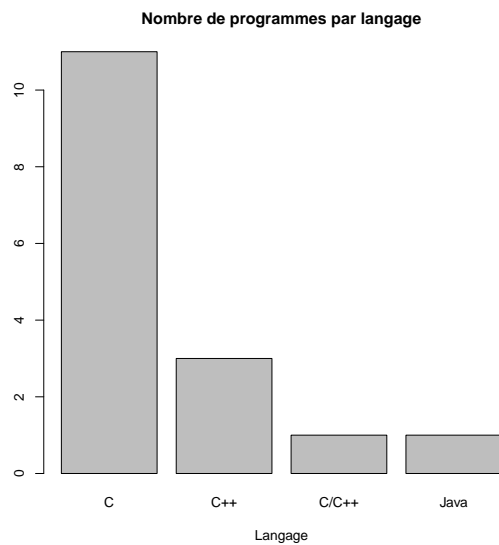


FIGURE 1

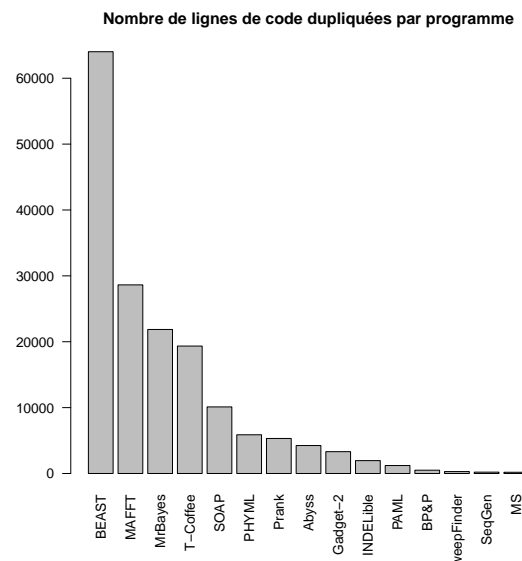


FIGURE 2

2 Pourcentage de lignes de code dupliquées

Afin de calculer le pourcentage de ligne dupliquées, nous avons créé une nouvelle table, contenant les noms des programmes, leurs domaines, les langages dans lesquels ils ont été écrits et leurs nombres de lignes dupliquées ainsi que leurs nombres de lignes totales. Nous avons enlevé de cette nouvelle table de données les programmes pour ayant un champs non renseigné (à l'aide de la commande `na.omit`). Nous avons ensuite rajouté une colonne à celle ci contenant les pourcentages de lignes de codes dupliquées, avant de créer un histogrammes de ces pourcentages pour chaque programme, ordonné par ordre décroissant (figure ??).

3 Comparaison du nombre de lignes duppliées en C et en C++

Pour effectuer un comparaison sur le nombre de lignes de codes des programmes C et C++, il nous faut faire un test de Mann-Whitney (grâce à la fonction `wilcox.test` en R. Pour que ce test

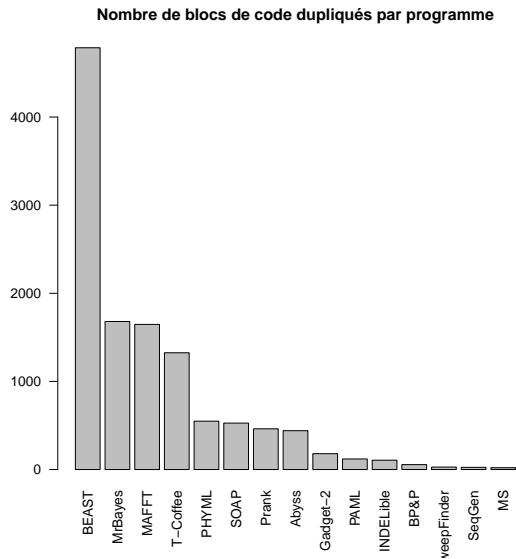


FIGURE 3

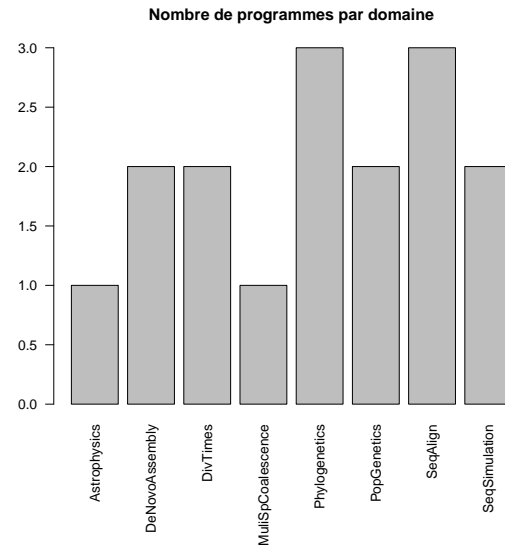


FIGURE 4

soit valide, nous sommes obligés de supposer que les deux échantillons considérés suivent une loi normale, de variance identique. De plus, les mesures sont bien indépendantes les unes des autres.

Nous faisons pour hypothèse nulle "Il n'y a pas de différences significatives du nombre de lignes de code dupliques en fonction du langage de programmation utilisé.". La p-value retournée par le test est de 0.94, au risque d'erreur 5%, nous ne pouvons donc pas rejeter l'hypothèse nulle. Nous en concluons que le nombre de lignes de codes dupliques et le langage de programmation sont décorrélés.

Nous noterons que la longueur des codes biaise l'interprétation de la mesure : en effet, plus un code est long, plus le nombre de lignes dupliques a de chance d'être élevé, aussi, un grand nombre de lignes de codes dupliques ne reflète pas une mauvaise technique de programmation, tout dépend de la taille du code ! Nous avons donc aussi comparé le pourcentage des lignes de codes écrits en C et en C++.

L'hypothèse nulle devient "Il n'y a pas de différences significatives du pourcentage de lignes de code dupliques en fonction du langage de programmation utilisé.". La p-value retournée par le test est de 0.41, au risque d'erreur 5%, nous ne pouvons donc pas rejeter l'hypothèse nulle. Nous en concluons que le langage de programmation et le pourcentage de lignes de code dupliques sont décorrélés.

4 Intervalle de confiance du taux de lignes dupliques

5 Lien entre la longueur du code et le nombre de warning à la compilation

Nous voulons explorer l'hypothèse selon laquelle les codes longs comporteraient plus de warnings à la compilation que les courts. Nous avons commencé à séparer notre table en deux,

l'une pour les programmes au nombre de lignes de codes inférieur à la médiane et l'autre pour les autres programmes.

Nous avons ensuite effectué trois test de Mann Whitney, un pour chaque type de warning (Clang, MinorWarning de gcc, MajorWarning de gcc), avec pour hypothèse nulle "Il n'y a pas de différence significatives du nombre de warning à la compilation entre les programmes de longueurs supérieure et inférieure à la médiane.". Nous avons pour cela encore une fois supposé que les échantillons considérés suivaient une loi normale de variance identique. Lors du test, nous avons passé *alternative = "less"* en argument à la fonction *wilcox.test*, cela correspond au fait que nous nous attendons à ce que le nombre de warning soit supérieur pour pour les programmes long.

Pour les Warning Clang, la p-value retournée par le test est de 0.0006216, au risque d'erreur 5%, nous pouvons donc rejeter l'hypothèse nulle. Nous en concluons que le nombre de warning Clang est supérieur pour les programmes longs.

Pour les Warning gcc, la p-value retournée par le test est de 0.6039 et 0.05896 pour les warning mineurs et majeurs respectivement. Au risque d'erreur 5%, nous ne pouvons donc pas rejeter l'hypothèse nulle. Nous en concluons que le nombre de warning gcc n'est pas sensiblement supérieur pour les programmes longs.

6 Questions personnelles

6.1 Lien entre le nombre et le type de warning à la compilation et la qualité de la gestion mémoire.

6.2 Autre question

Annexe

```
setwd("~/marco/Documents/Cours/sem2/Stat/TPs/Eval/urban-funicular")
sink("rapport/tests/tests.txt", append = TRUE)
tab = read.table("programs.txt", header = TRUE, sep = "\t" )
head(tab)
summary(tab)

#####

# Question 1

## Nombre de programmes par langage
pdf('rapport/figures/prog_lang.pdf')
barplot(table(tab$Language),
        main = "Nombre_de_programmes_par_langage",
        xlab = "Langage")
dev.off()

## Nombre de programmes par domaine
pdf('rapport/figures/prog_dom.pdf')
```

```

op = par(mar = c(9, 4, 4, 2) + 0.1)
barplot(table(tab$Domain),
        main = "Nombre_de_programmes_par_domaine",
        las = 2)
dev.off()

## Nombre de lignes de code par programme, ordonné par ordre décroissant
pdf('rapport/figures/lin_prog.pdf')
op = par(mar = c(7, 4, 4, 2) + 0.1)
tab_lines = tab[order(tab$LinesOfCode, decreasing = TRUE),]
barplot(tab_lines$LinesOfCode,
        names.arg = tab_lines$Code,
        las = 2,
        main = "Nombre_de_lignes_de_code_par_programme")
dev.off()

## Nombre de lignes dupliquées par programme, ordonné par ordre décroissant

pdf('rapport/figures/dlin_prog.pdf')
tab_dlines = tab[order(tab$DuplicatedLines, decreasing = TRUE),]
tab_dlines_noNA = tab_dlines[!is.na(tab_dlines$DuplicatedLines),]
barplot(tab_dlines_noNA$DuplicatedLines,
        names.arg = tab_dlines_noNA$Code,
        las = 2,
        main = 'Nombre_de_lignes_de_code_dupliquées_par_programme')
dev.off()

## Nombre de blocs dupliquées par programme, ordonné par ordre décroissant

pdf('rapport/figures/dbl_prog.pdf')
tab_dblocks = tab[order(tab$DuplicatedBlocks, decreasing = TRUE),]
tab_dblocks_noNA = tab_dblocks[!is.na(tab_dblocks$DuplicatedBlocks),]
barplot(tab_dblocks_noNA$DuplicatedBlocks,
        names.arg = tab_dblocks_noNA$Code,
        las = 2,
        main = 'Nombre_de_blocs_de_code_dupliqués_par_programme')
dev.off()

#####

# Question 2

tabP = na.omit(tab[, c("Code", "Domain", "Language", "DuplicatedLines", "LinesOfCode")])
tabP$DupLin_per = tabP$DuplicatedLines / tabP$LinesOfCode
tabP = tabP[order(tabP$DupLin_per, decreasing = TRUE),]

## Pourcentage de lignes dupliquées par programme
pdf('rapport/figures/pdlin_prog.pdf')
barplot(tabP$DupLin_per,

```

```

names.arg = tabP$Code,
las = 2,
main = "Pourcentage_de_lignes_dupliquées_par_programme")
dev.off()

#####

# Question 3

tabP_C = tabP[tab$Language == "C",]
tabP_CPP = tabP[tab$Language == "C++",]

wilcox.test(tabP_C$DupLin_per, tabP_CPP$DupLin_per)

#####

# Question 4

tabP_CO = tabP[tab$Language %in% c("C++", "C/C++"),]

t.test(tabP_C$DupLin_per, conf.level = 0.9)
t.test(tabP_CO$DupLin_per, conf.level = 0.9)

#####

# Question 5

tab_short = tab[tab$LinesOfCode <= median(tab$LinesOfCode),]
tab_long = tab[tab$LinesOfCode > median(tab$LinesOfCode),]

wilcox.test(tab_short$ClangWarning, tab_long$ClangWarning, alternative = "less")

# Reset graphic parameters
op = par(mar = c(5, 4, 4, 2) + 0.1)

#####

# Question 6

## Influence des evaluations de valgrind sur le nombre de warnings

tabW = na.omit(tab[, c("MajorWarning", "MinorWarning", "ClangWarning", "Valgrind")])

tabW_CL = tabW[tabW$Valgrind == "clean" | tabW$Valgrind == "leaks",]
Clean = tabW_CL[tabW$Valgrind == "clean",]
Leaks = tabW_CL[tabW$Valgrind == "leaks",]

# Comparaison des moyennes des programmes ayant une évaluation valgrind "CLEAN" et "LEAKS"

```

```
# > On peut affirmer qu'il existe une différence significative entre les
# > moyennes au risque de 5% pour les major warnings
wilcox.test(Clean$MajorWarning, Leaks$MajorWarning)
wilcox.test(Clean$MinorWarning, Leaks$MinorWarning)
wilcox.test(Clean$ClangWarning, Leaks$ClangWarning)

# > Pas de différence des moyennes significative au risque de 5% en considérant
# > toutes les évaluations valgrind possibles
# Données ni normales ni homoscédastiques
kruskal.test(tabW$MajorWarning ~ tabW$Valgrind)
kruskal.test(tabW$MinorWarning ~ tabW$Valgrind)
kruskal.test(tabW$ClangWarning ~ tabW$Valgrind)

## Influence du domaine scientifique sur le pourcentage de lignes dupliquées

# Pas de différence des moyennes significative au risque de 5%
kruskal.test(tabP$DupLin_per ~ tabP$Domain)
```
