



Modélisation et simulation discrète
Laboratoire n° 2

Moyenne des lots, variables de contrôle et temps moyen d'attente

Daniel Lifschitz et Nicolas Seriot,
IL2005b

28 février 2005

Résumé

Ce laboratoire consiste en l'étude et la simulation du temps moyen d'attente dans une file $G|G|K$ stable.

Nous commençons par estimer ce temps d'attente par simulation et en profitons pour tester l'efficacité de la méthode des variables de contrôle dans ce contexte particulier.

Ensuite, nous testons la qualité de la formule approximative de Krämer/Langenbach-Belz en fonction de différents paramètres et faisons une petite analyse statistique des données obtenues.

Table des matières

| | | |
|----------|---|-----------|
| 1 | Introduction | 3 |
| 2 | Intégration de la variable de contrôle | 3 |
| 2.1 | Adéquation de la variable de contrôle | 3 |
| 3 | Analyse du nombre de requêtes nécessaires | 3 |
| 3.1 | Sans variables de contrôle | 3 |
| 3.2 | Avec variables de contrôle | 5 |
| 3.3 | Comparaison du nombre de requêtes nécessaires | 8 |
| 4 | Qualité de la formule | 8 |
| 4.1 | Analyse exploratoire des données | 8 |
| 4.2 | Les valeurs extrêmes | 9 |
| 4.3 | Analyse exploratoire sans valeurs extrêmes | 11 |
| 4.4 | Analyse exploratoire sans valeurs NA | 14 |
| 4.5 | Nos conseils | 17 |
| 4.6 | Annexe : code R utilisé | 18 |
| 5 | Conclusion | 18 |
| | Listings | 20 |

1 Introduction

Ce deuxième et dernier laboratoire propose de mettre en œuvre la théorie relative aux variables de contrôle et l'utilisation de moyenne de lots vue au cours.

Il s'agit d'analyser les gains que l'on peut obtenir en simulant le temps d'attente dans le buffer d'une file $G|G|K$ mais aussi de d'analyser le comportement d'une formule approximative.

Le but de cette recherche est de pouvoir conseiller quelqu'un qui voudrait utiliser cette formule, en fonction de quatre paramètres.

2 Intégration de la variable de contrôle

La variable de contrôle utilisée dans le cadre de ce laboratoire a été définie comme étant le nombre d'arrivées durant les $d \cdot E(A)$ unités de temps précédant l'arrivée de la requête dans le système. Pour connaître ce nombre d'arrivées, nous insérons les requêtes au fur et à mesure de leur arrivée dans une liste FIFO tout en prenant soin de les supprimer une fois les $d \cdot E(A)$ unités de temps écoulées. Ainsi, lors de l'arrivée d'une nouvelle requête, la taille de la liste FIFO nous donne la valeur de notre variable de contrôle que nous sauvegardons avec l'événement associé dans l'échéancier si la requête ne peut pas être traitée immédiatement.

Le traitement d'un événement avec l'utilisation de la variable de contrôle est donnée par l'algorithme 1.

2.1 Adéquation de la variable de contrôle

Le choix de la variable de contrôle semble à priori judicieux. En effet, on comprend intuitivement que la probabilité que la file d'attente ne soit pas vide est nettement plus grande si plusieurs requêtes viennent d'arriver. Il en est exactement de même dans la vie de tous les jours. Imaginons qu'un individu rentre dans une poste juste après 10 autres clients, il va certainement devoir prendre son mal en patience avant d'accéder au guichet.

3 Analyse du nombre de requêtes nécessaires

3.1 Sans variables de contrôle

Le nombre de requêtes nécessaires à l'obtention d'un intervalle de confiance adéquat, sans utiliser la méthode des variables de contrôle, est présenté à la figure 1.

Algorithme 1 Traitement d'un événement avec variable de contrôle

prélever le prochain événement
avancer l'horloge à la date de l'événement
supprimer de la liste FIFO les événements plus vieux que $horloge - d \cdot E(A)$

si l'événement est une arrivée **alors**

 événement.var_contrôle \leftarrow taille de la liste FIFO
 ajouter le temps marqué par l'horloge dans la liste FIFO
 planifier la prochaine arrivée

si tous les serveurs sont occupés **alors**

 ajouter l'événement dans la file d'attente

sinon

 planifier la fin de son service

 calculer et retourner son temps d'attente avec variable de contrôle

fini

sinon si l'événement est une fin de service **alors**

 un serveur devient libre

si la file d'attente n'est pas vide **alors**

 retirer la prochaine requête

 planifier sa fin de service

 calculer et retourner son temps d'attente avec variable de contrôle

fini

fini

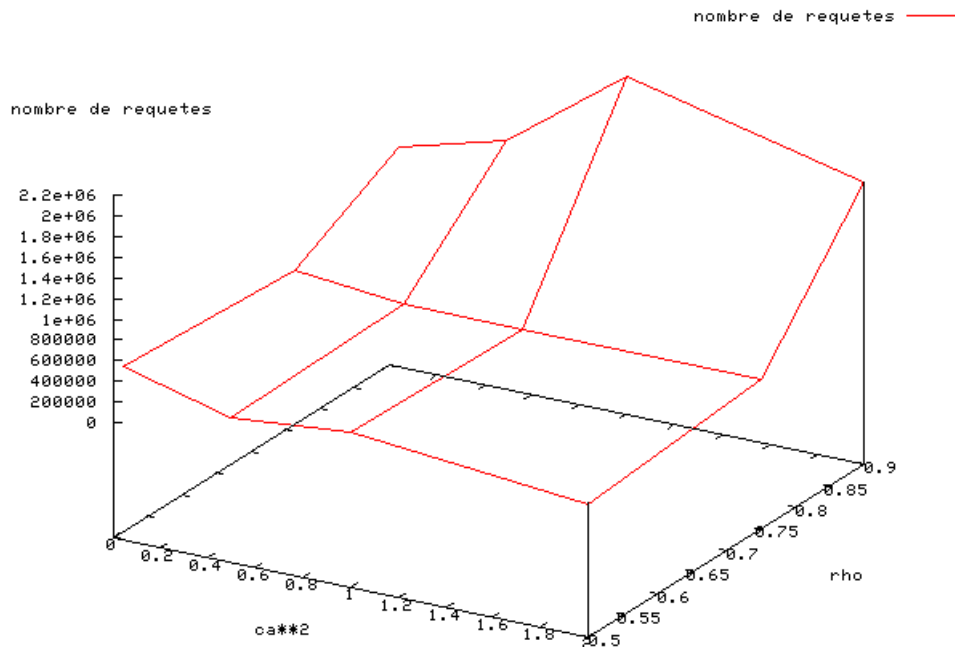


FIG. 1 – Nombre de requêtes nécessaires à l'obtention d'un intervalle de confiance adéquat, sans utiliser la méthode des variables de contrôle.

3.2 Avec variables de contrôle

Les figures 2 à 13 représentent le nombre de requêtes nécessaires à l'obtention d'un intervalle de confiance adéquat en utilisant la méthode des variables de contrôle, ceci avec les paramètres suivants : $C_A^2 \in \{0.05, 0.5, 1.0, 2.0\}$, $\rho \in \{0.5, 0.75, 0.9\}$, $d \in \{1, 2, \dots, 100\}$, $C_S^2 = 1$ et $K = 2$.

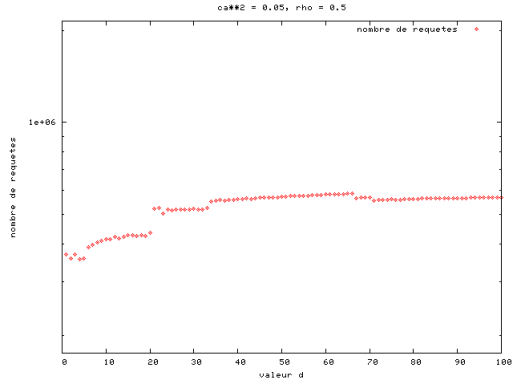


FIG. 2 - $C_A^2 = 0.05, \rho = 0.5$

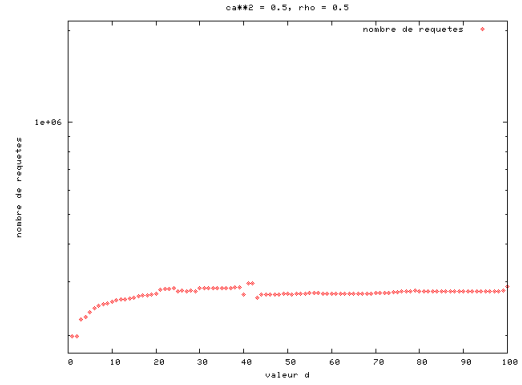


FIG. 3 - $C_A^2 = 0.5, \rho = 0.5$

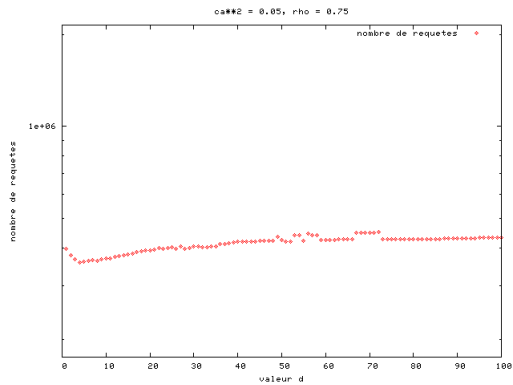


FIG. 4 - $C_A^2 = 0.05, \rho = 0.75$

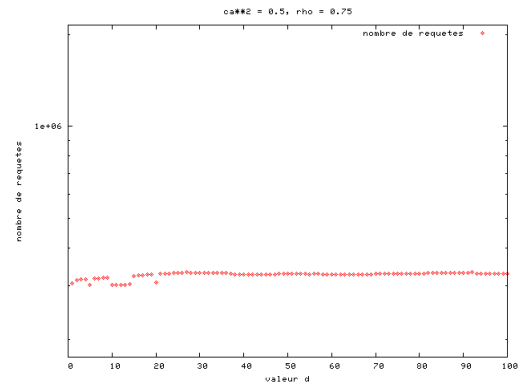


FIG. 5 - $C_A^2 = 0.5, \rho = 0.75$

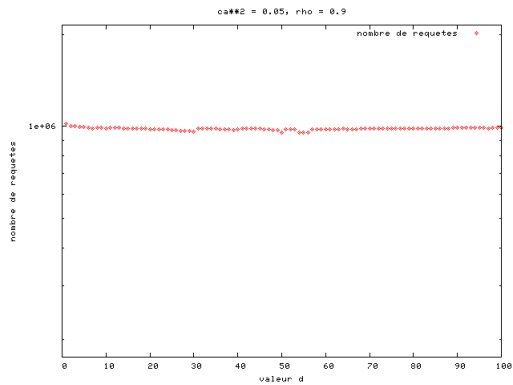


FIG. 6 - $C_A^2 = 0.05, \rho = 0.9$

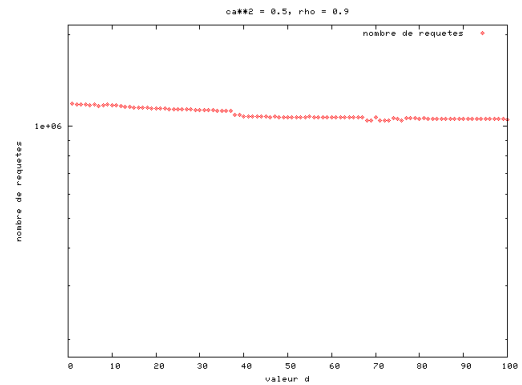


FIG. 7 - $C_A^2 = 0.5, \rho = 0.9$

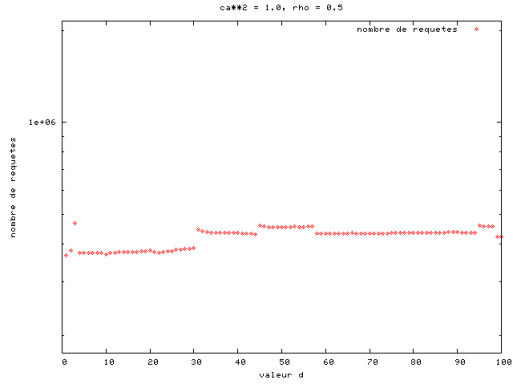


FIG. 8 – $C_A^2 = 1.0, \rho = 0.5$

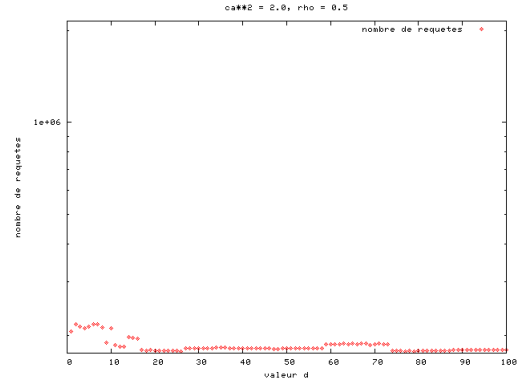


FIG. 9 – $C_A^2 = 2.0, \rho = 0.5$

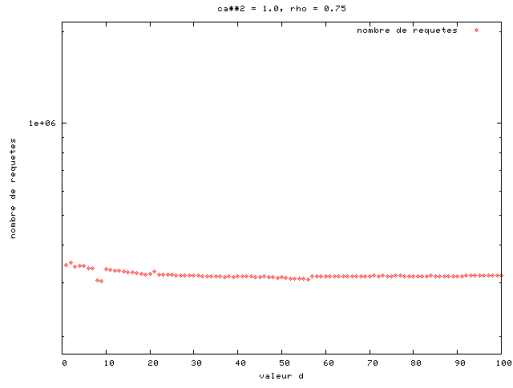


FIG. 10 – $C_A^2 = 1.0, \rho = 0.75$

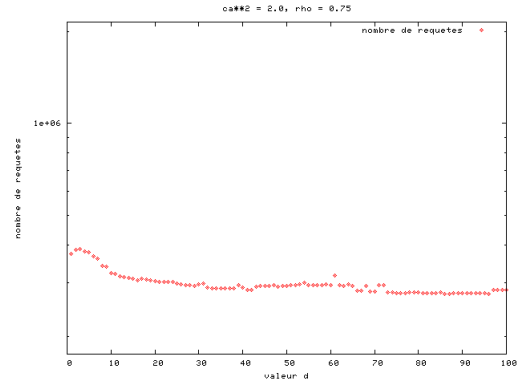


FIG. 11 – $C_A^2 = 2.0, \rho = 0.75$

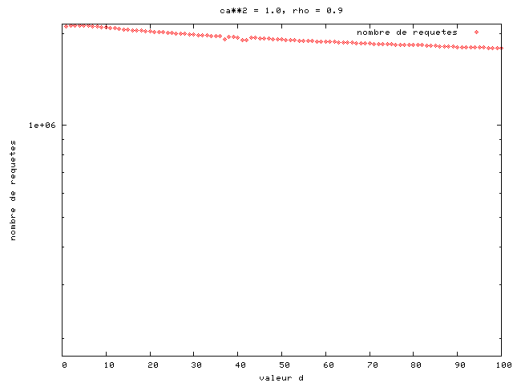


FIG. 12 – $C_A^2 = 1.0, \rho = 0.9$

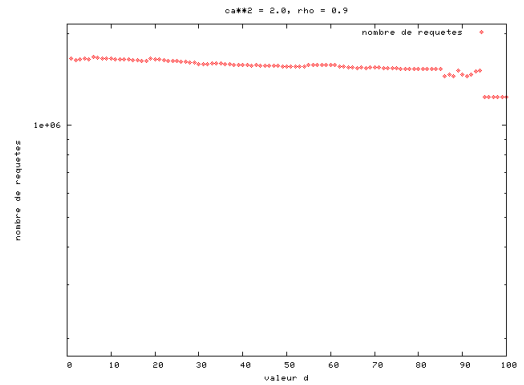


FIG. 13 – $C_A^2 = 2.0, \rho = 0.9$

En observant ces graphiques, on constate que, des trois paramètres (C_A^2, ρ, d) c'est ρ qui a une influence déterminante sur le nombre de requêtes nécessaires. En effet, un taux d'occupation $\rho = 0.9$, le nombre de requêtes est toujours supérieur à un million, tandis que pour $\rho \in 0.5, 0.75$ il est toujours en dessous.

La figure 14 reprend la représentation adoptée à la figure 1 pour représenter le nombre de requêtes nécessaires sans utiliser la méthodes des variables de contrôle (couleur rouge)

et indique sur le même graphique le nombre minimal de requêtes nécessaires en utilisant, cette fois, la méthode des variables de contrôle. Les données utilisées pour construire ce graphique sont résumées à la figure 15.

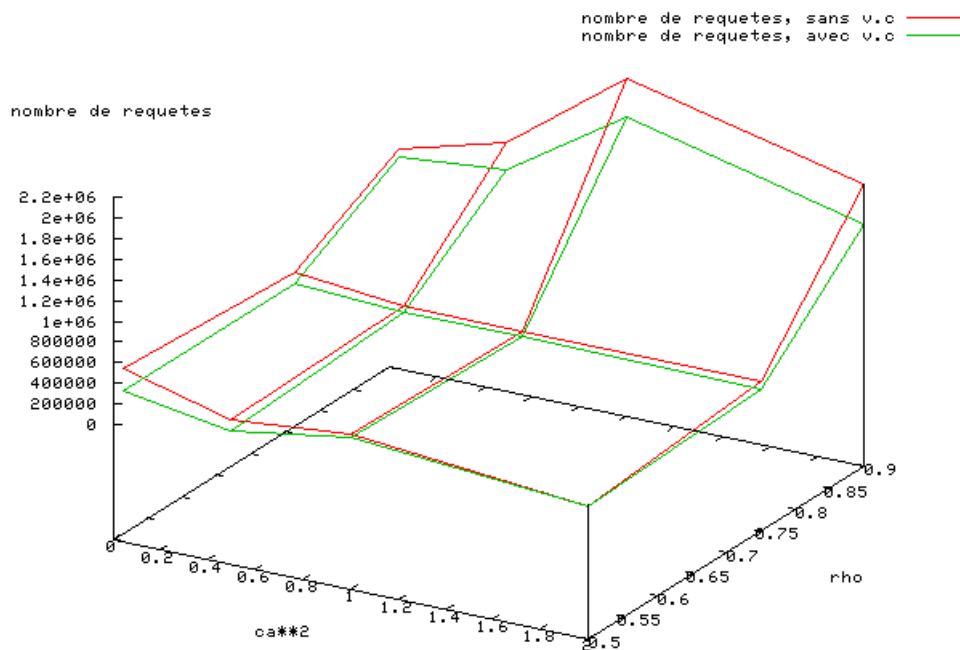


FIG. 14 – Comparaison du nombre de requêtes nécessaire à l’obtention d’un intervalle de confiance adéquat, avec et sans utiliser la méthode des variables de contrôle.

| C_A^2 | C_S^2 | K | ρ | nb. req. ss. v.c. | nb. req. avec v.c. | d | gain [%] |
|---------|---------|-----|--------|-------------------|--------------------|-----|----------|
| 0.05 | 1.0 | 2 | 0.5 | 576449 | 354541 | 4 | 38.50 |
| 0.05 | 1.0 | 2 | 0.75 | 470082 | 358122 | 4 | 23.82 |
| 0.05 | 1.0 | 2 | 0.9 | 1026823 | 955372 | 50 | 06.96 |
| 0.5 | 1.0 | 2 | 0.5 | 304790 | 198309 | 1 | 34.94 |
| 0.5 | 1.0 | 2 | 0.75 | 363938 | 301100 | 10 | 17.27 |
| 0.5 | 1.0 | 2 | 0.9 | 1300299 | 1043062 | 73 | 19.78 |
| 1.0 | 1.0 | 2 | 0.5 | 400792 | 366826 | 1 | 08.47 |
| 1.0 | 1.0 | 2 | 0.75 | 352082 | 303718 | 9 | 13.74 |
| 1.0 | 1.0 | 2 | 0.9 | 2164736 | 1793609 | 100 | 17.14 |
| 2.0 | 1.0 | 2 | 0.5 | 184230 | 177585 | 26 | 03.61 |
| 2.0 | 1.0 | 2 | 0.75 | 349543 | 276167 | 87 | 20.99 |
| 2.0 | 1.0 | 2 | 0.9 | 1619345 | 1236587 | 99 | 23.64 |

FIG. 15 – Résumé des données.

3.3 Comparaison du nombre de requêtes nécessaires

Le tableau 15 indique le nombre de requêtes nécessaires avec et sans utiliser la méthode des variables de contrôle. Il appelle les commentaires suivants.

Gain Quand le carré du coefficient de variation du temps d'attente C_S^2 est inférieur à 1, le gain est inversement proportionnel au taux d'occupation ρ , tandis que pour des valeurs de $C_S^2 > 1$ c'est l'inverse : le gain est proportionnel au taux d'occupation.

Le paramètre d La valeur de d représentée dans le tableau est celle pour laquelle le gain est le plus important, pour des valeurs allant de 1 à 100. La tendance que l'on observe est que, quelque soit la valeur de C_A^2 , d croit avec ρ ; plus le carré du coefficient de variation du temps d'attente augmente et plus le paramètre d — donc l'intervalle de temps utilisé pour la variable de contrôle — croit.

4 Qualité de la formule

4.1 Analyse exploratoire des données

Pour déterminer la qualité de la formule de Krämer/Langenbach-Belz (KLB), nous avons effectué les simulations pour les valeurs des paramètres proposées dans la donnée du laboratoire. Nous nous sommes donc retrouvés avec un fichier contenant 256 résultats, chacun étant déterminé par 4 paramètres. Comment utiliser ces résultats pour étudier la pertinence de la formule ?

Il faut commencer par définir ce que nous allons étudier. C'est ici l'erreur entre le temps d'attente calculé T_a et la moyenne des temps mesurés m . Il y a plusieurs manières de qualifier cette erreur : nous avons utilisé plusieurs définitions comme la différence en valeur absolue ou le rapport (la proportion) de T_a et m pour finalement choisir la différence en pour-cent, soit $100 \cdot (m - T_a)/m$. Cette valeur sera par la suite appelée `diff`.

Chaque définition de l'erreur a des inconvénients : avec une mesure en valeur absolue, de grosses erreurs sur de petits temps d'attente passeront inaperçues. Avec une mesure en pour-cents, les erreurs sur les petits temps d'attente risquent d'apparaître énormes, alors qu'il ne s'agit que de fractions d'unités de temps qui, selon ce que l'on cherche à mesurer, peuvent être négligeables. Cela dépend en fait du contexte¹. Nous avons choisi les pour-cents, car l'unité de temps utilisée dans le cadre de ce laboratoire n'est pas définie et qu'il peut aussi bien s'agir de secondes que d'années lumières.

Nous avons donc commencé par trier le fichier d'après cette valeur `diff`. En observant les paramètres des lignes se trouvant en tête de la liste et en queue de la liste, on peut déjà déceler une tendance assez marquée ; les petits valeurs de ρ (taux d'occupation des

¹Imaginons que l'on cherche à connaître, avec cette formule, le temps d'attente de clients dans la file d'une banque. Peut-être que, dans ce cadre, là on aurait intérêt à définir l'erreur en valeur absolue, puisque ce qui importe alors n'est pas vraiment que la formule se trompe de 300% sur le client qui attend 10 secondes mais que les clients n'attendent pas plus d'un certain temps.

serveurs) sont plutôt associées à des erreurs importantes, tandis que les grandes valeurs de ρ semblent déterminer des erreurs très faibles.

Pour pouvoir analyser cela, nous avons chargé ces données dans le logiciel R [3] en espérant pouvoir étudier la corrélation entre les paramètres des simulations et la différence de résultat avec la formule de KLB. Pour avoir une idée de la distribution des valeurs dans notre fichier, nous avons affiché quelques statistiques de base sur la colonne `diff`. Pour les valeurs de m égales à zéro, nous traitons la valeur NA ; 30 valeurs sur 256 sont concernées. Nous avons également ajouté une colonne `absdiff` comprenant la valeur absolue de l'erreur (toujours en pour-cents). Nous avons ensuite représenté une boîte à moustache (figure 16) et un histogramme (figure 17) de `diff`, ceci dans le but d'avoir un aperçu de la distribution des erreurs.

```
> summary(diff)
      Min.   1st Qu.   Median     Mean   3rd Qu.     Max.    NA's
-1198.000  -41.700   -10.230   -34.350   -1.707    99.220    30.000
```

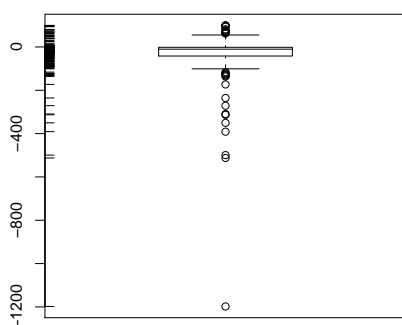


FIG. 16 – Différence entre le temps d’attente simulé et la formule de Krämer/Langenbach-Belz sur les valeurs brutes : boîte à moustaches

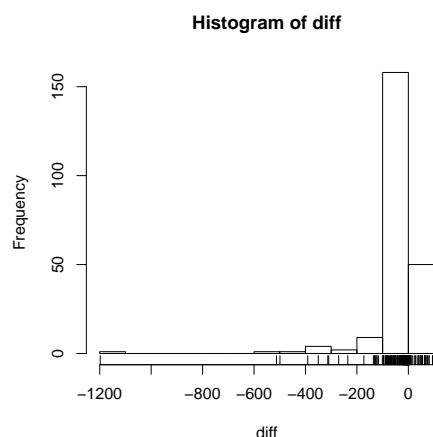


FIG. 17 – Différence entre le temps d’attente simulé et la formule de Krämer/Langenbach-Belz sur les valeurs brutes : histogramme

4.2 Les valeurs extrêmes

La première chose qui ressort, c’est la présence de cinq valeurs extrêmes. Ces valeurs sont présentées à la figure 18.

Elles ont en commun un taux d’occupation $\rho = 0.25$, un coefficient $C_A^2 = 1$ et des temps d’attente calculés $T_a < 0.72$. Pour la lisibilité de l’analyse, mettons ces valeurs de côté pour l’instant, c’est-à-dire que nous leur attribuons la valeur NA : notre nouveau vecteur de travail s’appelle `diffna`. Nous avons maintenant $30 + 5 = 35$ valeurs NA sur 256. En effectuant une nouvelle fois les démarches exploratoires précédentes, les données deviennent beaucoup plus claires.

| C_A^2 | C_S^2 | K | ρ | T_a | m | diff [%] |
|---------|---------|-----|--------|----------|----------|--------------|
| 1 | 0.05 | 4 | 0.25 | 0.109843 | 0.008462 | -1198.073741 |
| 1 | 0.05 | 2 | 0.25 | 0.439371 | 0.071746 | -512.3979037 |
| 1 | 0.5 | 4 | 0.25 | 0.156918 | 0.026185 | -499.2667558 |
| 1 | 1 | 4 | 0.25 | 0.209224 | 0.042598 | -391.1592094 |
| 1 | 2 | 4 | 0.25 | 0.313837 | 0.069691 | -350.326441 |

FIG. 18 – Valeurs extrêmes

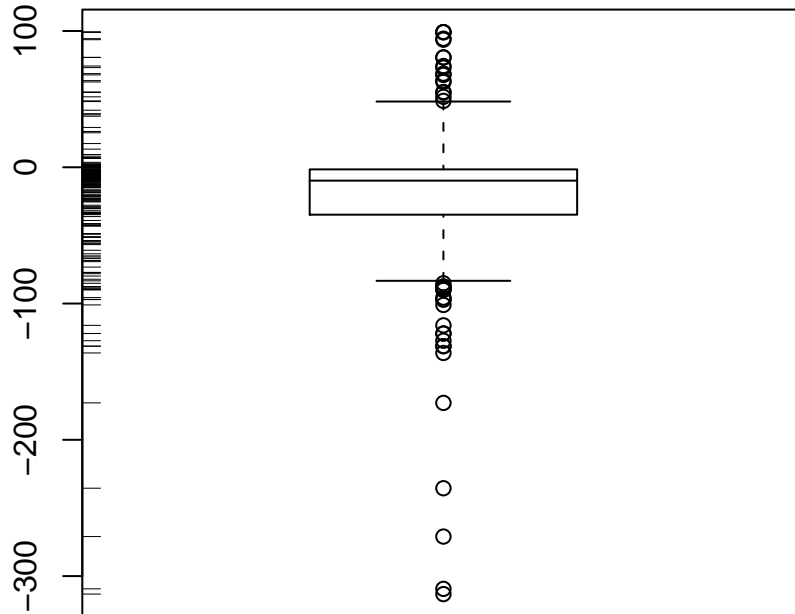


FIG. 19 – Différence entre le temps d'attente simulé et la formule de Krämer/Langenbach-Belz sans les cinq valeurs les plus extrêmes : boîte à moustaches

Histogram of diffna

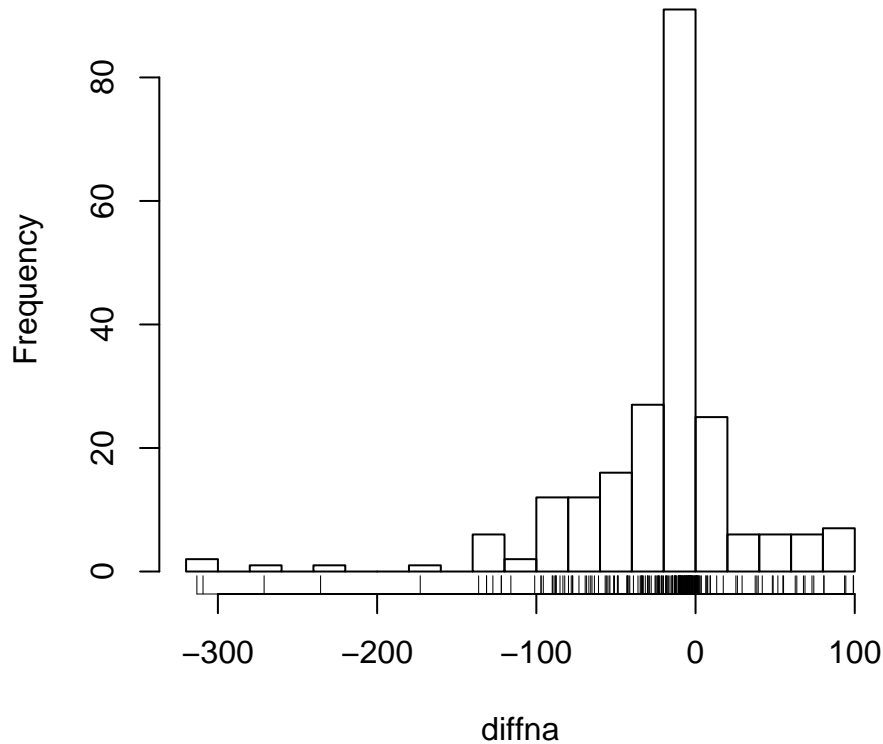


FIG. 20 – Différence entre le temps d’attente simulé et la formule de Krämer/Langenbach-Belz sans les cinq valeurs les plus extrêmes : histogramme

```
> summary(diffna)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.   NA's
-313.200 -34.780  -9.807  -21.780  -1.548   99.220  35.000
```

4.3 Analyse exploratoire sans valeurs extrêmes

Les statistiques et les graphiques nous permettent maintenant de faire plusieurs observations² sur la formule de Krämer/Langenbach-Belz (KLB).

1. la suppression de 5 valeurs extrêmes a ramené la moyenne des erreurs de -34.35 à -21.78 pour-cents ;
2. une grande majorité des valeurs est négative, ce qui veut dire que dans la plupart des cas la formule KLB minore le temps d’attente mesuré ;
3. le tiers des données le plus pertinent offre une approximation entre 0 et -20% en dessous de la valeur réelle ;

²Ces observations se font bien sûr sur la base des valeurs proposées dans la donnée du laboratoire, elles pourraient être différentes pour d’autres valeurs.

4. il reste cinq simulations pour lesquelles l'erreur dépasse 150%.

Ce sont des informations précieuses, mais qui ne permettent pas encore de savoir dans quelle mesure chaque paramètre influence la qualité de la formule. Pour cela, nous avons utilisé le module `lattice` de R et sa fonction `bwplot()`, qui permet de représenter une distribution selon plusieurs autres variables. Nous avons représenté la valeur absolue de l'erreur `absdiffna` dans des boîtes à moustache selon les paramètres C_A^2 , C_S^2 , K et ρ . Les résultats sont présentés aux figures 21, 22, 23 et 24. Notons d'emblée une question graphique : la graduation de l'axe des ordonnées ne correspond pas à la valeur numérique représentée mais au numéro de la classe : 1 pour la 1ère valeur dans l'ordre croissant, 2 pour la 2ème, etc. Nous n'avons pas encore trouvé comment faire autrement.

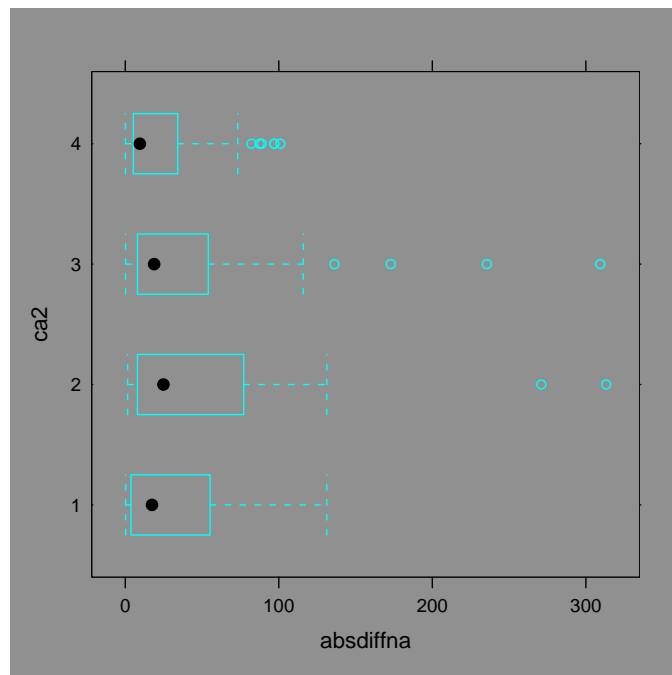


FIG. 21 – `absdiffna` selon C_A^2

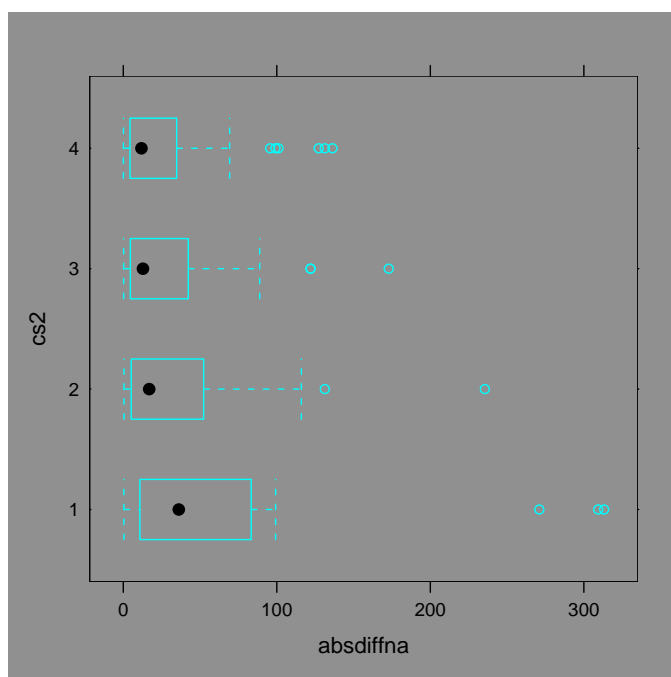


FIG. 22 – absdiffna selon C_S^2

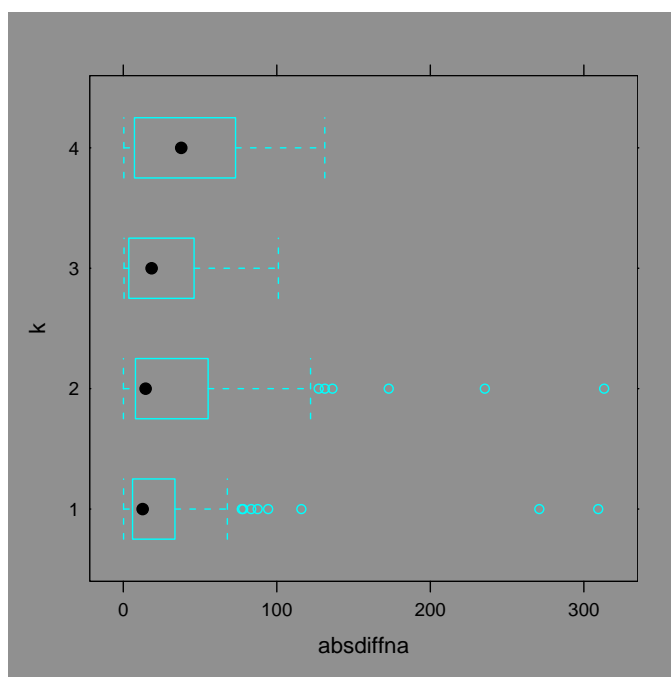


FIG. 23 – absdiffna selon K

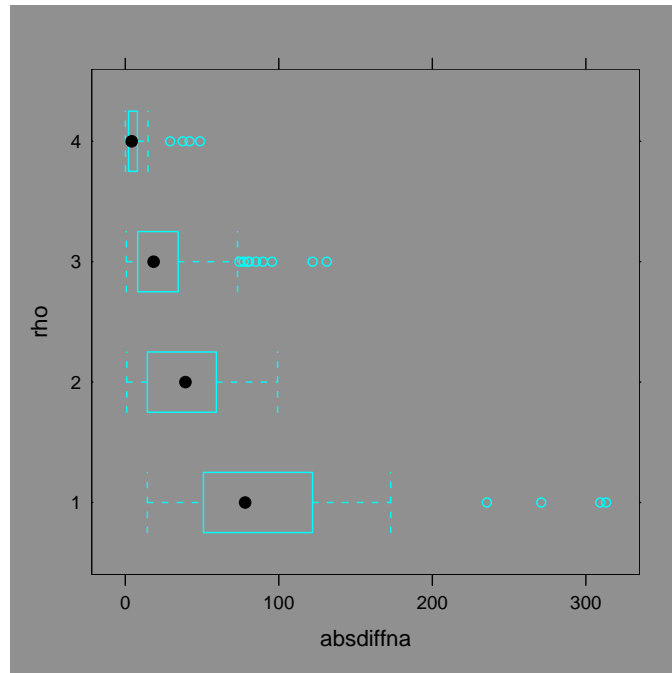


FIG. 24 – absdiffna selon ρ

On cherche donc à déceler l'influence d'un paramètre sur l'erreur de la formule. En observant les graphiques, il apparaît assez nettement que ρ est lié à `diffna` (figure 24). Le calcul des corrélations nous permet de confirmer ce que nous disent les graphiques.

4.4 Analyse exploratoire sans valeurs NA

La fonction de corrélation de R n'accepte que des vecteurs sans valeurs NA, raison pour laquelle nous recopions notre jeu de données `klb` dans un nouvel objet `klbnona` sans les valeurs nulles :

```
> klbnona <- na.omit(klb)
```

Voici donc les valeurs obtenues :

```
> cor(klbnona$absdiff, klbnona$ca2)
[1] -0.1125325
> cor(klbnona$absdiff, klbnona$cs2)
[1] -0.1936404
> cor(klbnona$absdiff, klbnona$k)
[1] 0.04727066
> cor(klbnona$absdiff, klbnona$rho)
[1] -0.581303
```

Ces valeurs confirment l'influence prépondérante de ρ , par rapport aux autres paramètres, sur la qualité de la formule, avec un coefficient de corrélation de 0.58.

Il nous faut aussi envisager la possibilité que des influences déterminantes sur la qualité de la formule soient le fait de deux facteurs combinés. Nous avons donc construit plusieurs graphiques en observant l'influence conjointe des valeurs de deux variables sur absdiffna (figures 25, 26 et 27).

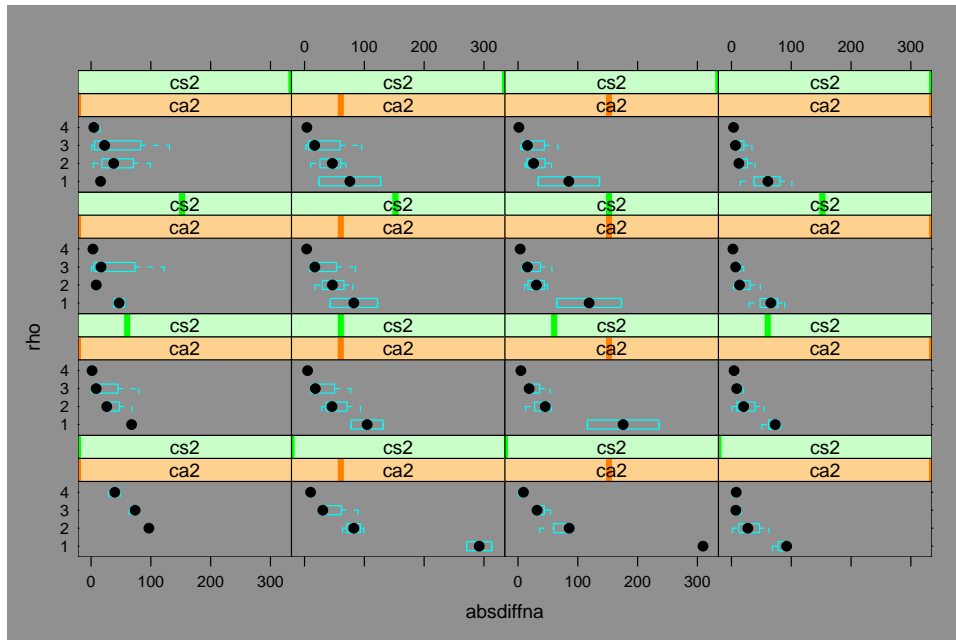


FIG. 25 – diff selon ρ

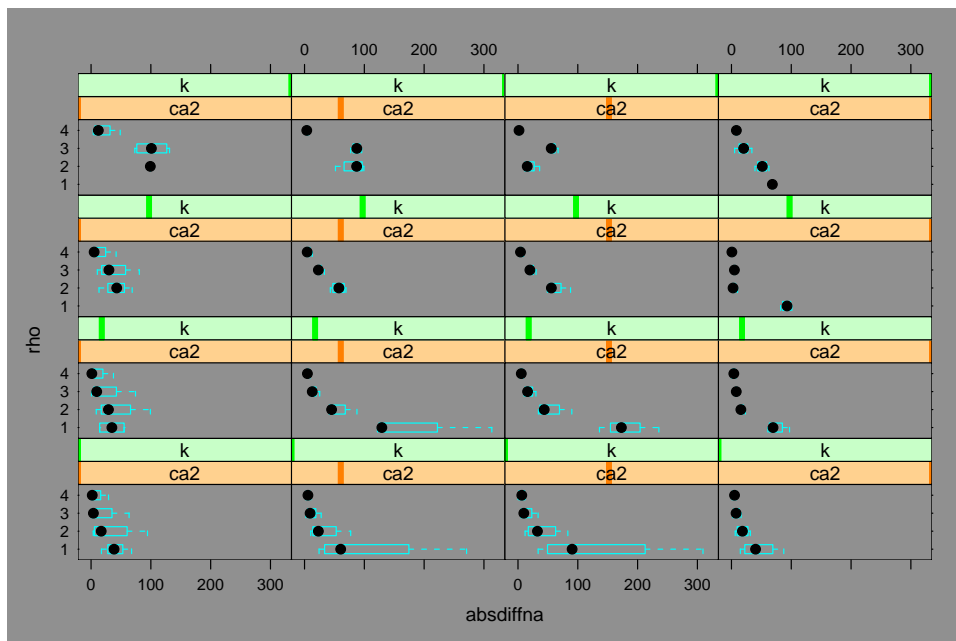


FIG. 26 – diff selon ρ

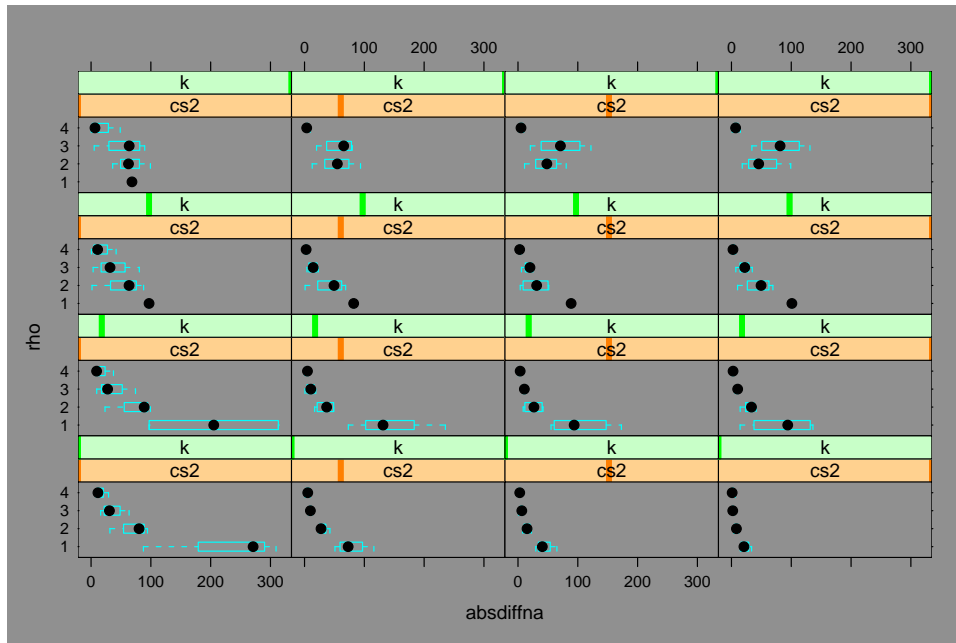


FIG. 27 – diff selon ρ

Comment lire ces graphiques ? Par exemple dans le cas de la figure 25 les classes de ρ sont représentées en ordonnée : "1" correspond à 0.25, "2" à 0.5, "3" à 1.0 et "4" à 2.0. Il y a plusieurs abscisses : la principale est la valeur `absdiffna` absolue de l'erreur ; les points les plus à droite de chaque case indiquent les valeurs pour lesquelles l'erreur est la plus importante. Enfin, la position des 16 cases dans le tableau se lit de la manière suivante : les lignes indiquent les valeurs de C_A^2 et les colonnes celles de C_S^2 . Des marques dans le titre de la case rappellent ces valeurs.

Qu'est-ce que ces graphiques nous apprennent ? Ils nous permettent d'identifier les combinaisons de paramètres pour lesquels l'erreur est la plus grande. Il faut bien sûr tenir compte des valeurs supprimées pour améliorer la visibilité — pour mémoire ce sont des valeurs telles que $C_A^2 = 0.05, C_S^2 \in \{0.05, 0.5\}$ (figure 18).

Dans les trois graphiques, on voit que ρ est bien le facteur déterminant. On voit aussi sur la figure 25 que les erreurs sont particulièrement importantes pour des valeurs comme $C_A^2 \in \{0.5, 1.0\}$ et $\rho = 0.25$, de l'ordre de 50 à 100%.

Sur la figure 26, on voit de nouveau l'effet désastreux de $C_A^2 \in \{0.5, 1.0\}$ et $\rho = 0.25$ mais aussi l'influence de K , qui semble limiter l'explosion des erreurs quand il est petit (1 ou 2 serveurs, situation qui va un peu prolonger le temps d'attente).

Enfin sur la figure 27, on voit aussi l'influence de C_S^2 qui, pour $\rho = 0.5$, semble être proportionnel à l'erreur pour les grands K et inversement proportionnel pour les petits K .

Soulignons que tout ceci n'est rien d'autre que des « impressions » et qu'en l'absence de calculs rigoureux il est impossible d'être catégoriques comme nous l'avons été à propos

de l'influence de ρ . Ce sont cependant des éléments qui peuvent aider les utilisateurs de la formule.

Il est donc extrêmement délicat de faire des généralités, qui plus est à partir de quatre valeurs par variable. Cependant, les éliminations initiales de données, le constat d'erreurs importants avec ρ petit ainsi que l'observation des graphique nous mettent « la puce à l'oreille » sur le fait que le temps d'attente doit être assez fortement corrélé avec le pourcentage d'erreur. Voyons ce que R nous dit :

```
> cor(klbnona$absdiff, klbnona$ta)
[1] -0.4158042
```

Le coefficient de corrélation est en effet important (celui de ρ était de -0.58). C'est un élément que nous devons communiquer dans nos conclusions.

4.5 Nos conseils

Voici les conseils que nous donnerions à quelqu'un souhaitant utiliser la formule de Krämer/Langenbach-Belz (KLB). Nous commencerions par trois remarques sur la distribution des erreurs (voir page 11) :

1. le tiers des données le plus pertinent offre une approximation entre 0 et 20% en dessous de la valeur réelle ;
2. dans la grande majorité des cas l'erreur est de l'ordre de 50 à 100 pour-cents³ ;
3. dans 10 cas sur les 256 que nous avons simulé, l'erreur dépasse 150 % ;
4. les erreurs de la formule en pour-cents sont assez fortement corrélées avec le temps d'attente donné par la formule (coefficient : -0.41), ce qui veut dire que pour des petits temps d'attente de l'ordre de moins d'une unité, la formule est à prendre avec des pincettes.

Maintenant, à propos des plages de valeurs pour lesquelles la formule serait moins fiable, nous pouvons dire ceci : la formule est généralement une assez bonne approximation de l'ordre de grandeur du temps simulé, sauf dans certains cas pathologiques dont le principal est un temps d'occupation très petit : $\rho = 0.25$. Dans ce cas, près de la moitié des valeurs dépassent le 100% d'erreur. Cela s'aggrave encore quand C_A^2 est petit ou quand le nombre de serveurs K est grand.

Les valeurs exposées ci-dessus concernent en fait des cas où le temps d'attente est très très court voir nul (pensons aux valeurs supprimées). On ne peut donc pas dire grand chose du temps d'attente et c'est là que la formule dérape ; la corrélation entre les erreurs et le temps d'attente calculé le prouve, sans qu'un temps d'attente très court implique forcément une erreur importante.

³Si la formule semble donner des erreurs tout de même importantes, la quantification de cette erreur dépend de ce que nous appelons erreur : voir la discussion de ce point page 8.

4.6 Annexe : code R utilisé

```
> klb <- read.table('5.txt', header=T)
> attach(klb)
```

Figure 19

```
> boxplot(diffna)
> rug(jitter(diffna), side=2)
```

Figure 20

```
> hist(diffna, breaks=16)
> rug(jitter(diffna))
```

Figures 21 à 24

```
> library(lattice)
> bwplot(ca2 ~ absdiffna)
> bwplot(cs2 ~ absdiffna)
> bwplot(k ~ absdiffna)
> bwplot(rho ~ absdiffna)
```

Figures 25 à 27

```
> bwplot(rho ~ absdiffna | ca2 * cs2)
> bwplot(rho ~ absdiffna | ca2 * k)
> bwplot(rho ~ absdiffna | cs2 * k)
```

5 Conclusion

Ce laboratoire a été l'occasion d'aborder sous un côté pratique une matière souvent très théorique, de comprendre et d'utiliser des techniques comme les variables de contrôle ou les lots.

Une des difficultés résidait dans l'analyse de (petits mais tout de même complexes) jeux de données. Nous nous sommes aidés de [4] pour apprendre à utiliser R, ce qui a pris un temps non négligeable. Le fait que nous n'ayions pas tout à fait les connaissances statistiques nécessaires nous a obligé à tâtonner en essayant de rechercher des indices visuels, alors que R propose des méthodes plus formelles mais que nous comprenons mal. Dans cette optique, ce laboratoire nous avons permis de défricher le terrain et nous réjouissons de suivre le cours de data-mining pour pouvoir faire parler des gros volumes de données.

Références

- [1] Éric Thiémard. *Transparents du cours « Modélisation et simulation discrètes »*. EIVD, 2004-2005.
- [2] Gnuplot,
<http://gnuplot.sf.net/>.
- [3] The R Project for Statistical Computing,
<http://www.r-project.org/>.
- [4] Data-Mining with R,
<http://www.liacc.up.pt/~ltorgo/DataMiningWithR/>.

Listings

| | | |
|---|--------------------------|----|
| 1 | simulation.c | 20 |
| 2 | statistiques.c | 25 |

Listing 1 – simulation.c

```
/*
/*
/* Fichier : simulation.c
/*
5 /* Auteur : E. Thiéard, département E+I, EIVD
/* Version : Janvier 2005
/* Contexte : Modélisation et simulation discrètes, filière IL
/*
/* But : Module permettant de simuler le fonctionnement d'une
10 /* file G|G|K (lois A et S log-normales). Retourne le
/* temps d'attente des requêtes successives lorsqu'on
/* n'utilise pas de variable de contrôle, ou x+c(y-mY)
/* lorsqu'on utilise la variable de contrôle Y="nombre
/* d'arrivées depuis d*E(A) unités de temps.
15 /*
/*
/*
/* Auteurs : Daniel Lifschitz & Nicolas Seriot
/* Date : 7 février 2005
20 /* modifs : codage de la fonction
/* simulation_prochaine_observation_avec_var_controle(void)
/*
/*
/*
25 /* Consigne : la simulation sans variable de contrôle fonctionne.
/* Compléter la partie avec variable de contrôle (en
/* utilisant le module "liste") ; ajouter au besoin les
/* variables et fonctions auxiliaires nécessaires.
/*
30 /*
#include <stdlib.h>
#include <stdio.h>
35 #include <math.h>
#include "simulation.h"
#include "echeancier.h"
#include "MT19937.h"
#include "generateurs.h"
40 #include "buffer.h"
#include "liste.h"

static double CA2;
static double CS2;
45 static int K;
static double rho;
static double muA;
static double sigma2A;
static double muS;
50 static double sigma2S;
static double horloge = 0.0;
static int nb_serv_occupes = 0;

static double duree_var_controle;
55 static int moyenne_var_controle;
static double covariance;
static double moyenne_X;
static double moyenne_Y;
static double variance_Y;
60 static int nombre_observation;
```

```

/*****/
/* Planifie la prochaine arrivée d'une requête et envoie l'événement */
65 /* correspondant à l'échéancier. */
/*****/
static void planifie_arrivee(void)
{
    echeancier_type_eve evenement;
70
    evenement.type = arrivee;
    evenement.date_evenement = horloge+generateurs_log_normale(genrand_real3,muA,
        sigma2A);
    echeancier_nouvel_evenement(evenement);
}
75

/*****/
/* Planifie la fin du service d'une requête e tenvoie l'événement */
/* correspondant à l'échéancier. */
80 /*****/
static void planifie_fin_service(void)
{
    echeancier_type_eve evenement;

85 nb_serv_occupes++; /* Un serveur de plus devient occupé. */

    evenement.type = fin_service;
    evenement.date_evenement = horloge+generateurs_log_normale(genrand_real3,muS,
        sigma2S);
    echeancier_nouvel_evenement(evenement);
90 }

/*****/
/* Initialisation du générateur pseudo-aléatoire, de l'échéancier et */
95 /* des divers paramètres, planification de la première arrivée et */
/* affichage de la formule approximative de Krämer/Langenbach-Belz. */
/*****/
void simulation_initialisation(double carre_cvar_arrivee, double
    carre_cvar_service, int nb_serveurs, double taux_occup)
{
100 double Ta, W, KK;
    unsigned long init[4]={0x123,0x345,0x567,0x789}, length=4; /* Germe */

    init_by_array(init,length); /* Initialisation gén pseudo-aléatoire */
    echeancier_creer(); /* Création d'un échéancier vide */
105
    CA2 = carre_cvar_arrivee;
    CS2 = carre_cvar_service;
    K = nb_serveurs;
    rho = taux_occup;
110 KK = (double) K;

    sigma2A = log(CA2+1.0); /* Initialisation des paramètres des lois */
    sigma2S = log(CS2+1.0); /* A: L(muA,sigma2A) et S: L(muS,sigma2S) */
    muA = 3.0 - sigma2A / 2.0;
115 muS = 3.0 + log(KK*rho) - sigma2S / 2.0;

    printf("\nCaractéristiques du processus d'arrivée :\n");
    printf(" Paramètres loi log-normale : mu_A = %f, sigma^2_A = %f\n",muA,
        sigma2A);
    printf(" Espérance = %f\n",exp(muA+sigma2A/2.0)); /* = e^3 = 20.08 */
120 printf(" Variance = %f\n",exp(2.0*muA+sigma2A)*(exp(sigma2A)-1.0));
    printf(" Carré du coefficient de variation = %f\n",CA2); /* = Var/(Esp^2) */

    printf("\nCaractéristiques du processus de service :\n");
    printf(" Paramètres loi log-normale : mu_S = %f, sigma^2_S = %f\n",muS,
        sigma2S);

```

```

125 printf("    Espérance = %f\n",exp(muS+sigma2S/2.0));
printf("    Variance = %f\n",exp(2.0*muS+sigma2S)*(exp(sigma2S)-1.0));
printf("    Carré du coefficient de variation = %f\n",CS2); /* = Var/(Esp^2) */
printf("    Nombre de serveurs : %d\n",K);
printf("    Taux d'occupation : %f\n",rho);
130
    if (rho <= 0.7)
        W = pow(rho,(KK+1.0)/2.0);
    else
        W = (rho+pow(rho,KK))/2.0;
135 Ta = W*exp(muS+sigma2S/2.0)*(CA2+CS2)/((1.0-rho)*2.0*KK);
    if (CA2 <= 1.0)
        Ta *= exp(-2.0*(1.0-rho)*pow(1.0-CA2,2.0)/(3.0*W*(CA2+CS2)));
    else
        Ta *= exp((rho-1.0)*(CA2-1.0)/(CA2+4.0*CS2));
140 printf("\nFormule approximative pour le temps moyen d'attente : %f\n\n",Ta);

    planifie_arrivee();          /* Planification de la première arrivée */
}

145
/*****
/* Initialisation du module "variable de contrôle"
/*****
void simulation_initialisation_var_controle(int multiple)
150 {
    moyenne_var_controle = multiple;
    duree_var_controle = (double) multiple * exp(3.0);
    printf("Simulation avec variable de contrôle pour une durée %d * %f = %f\n\n",
        moyenne_var_controle,exp(3.0),duree_var_controle);

155    nombre_observation = 0;
    covariance = 0.0;
    moyenne_X = 0.0;
    moyenne_Y = 0.0;
    variance_Y = 0.0;
160 }

/*****
/* Cette fonction poursuit la simulation jusqu'à ce qu'elle soit en
165 /* mesure de retourner le temps d'attente de la requête suivante.
/*****
double simulation_prochaine_observation(void)
{
    echeancier_type_eve evenement;
170
    while(1)
    {
        evenement = echeancier_retire_prochain_evenement();
        if (evenement.date_evenement<horloge)
175 {
            fprintf(stderr,"Erreur : chronologie non respectée !\n");
            exit(EXIT_FAILURE);
        }
        horloge = evenement.date_evenement; /* Mise à jour de l'horloge */

180
        switch(evenement.type)
        {
        case arrivee :
            planifie_arrivee();          /* Planifie prochaine arrivée */
185            if (nb_serv_occupes != K)
                {
                    planifie_fin_service();
                    return 0.0;          /* La requête n'a pas attendu */
                }
            else
190                buffer_ajouter(evenement); /* Doit attendre dans le buffer */

```

```

        break;

    case fin_service :
195     nb_serv_occupes--;          /* Un serveur redevient libre */
        if (buffer_pas_vide())
        {
            evenement = buffer_retirer();    /* Une requête est retirée du buffer */
            planifie_fin_service();         /* et peut donc commencer son service. */
200         return horloge-evenement.date_evenement; /* On a son temps d'attente. */
        }
        break;

    default :
205     fprintf(stderr,"Erreur : événement de type inapproprié\n");
        exit(EXIT_FAILURE);
    }
}
210 }

/*****
/* Cette fonction retourne le temps d'attente de la requête modifiée */
/* en tenant compte de la variable de contrôle. */
/* Paramètre : duree - le temps d'attente de la requête dans le buffer*/
215 /* nombre_arrivee - le nombre d'arrivée durant les */
/* d * E(A) unités de temps précédant l'arrivée de la */
/* requête. */
/* Retour : x+c(y-my) */
/* mesure de retourner x+c(y-mY), le temps d'attente de la requête */
220 *****/
double temps_avac_variable_controle(double duree, int nombre_arrivee) {

    double ancienne_moyenne_X = moyenne_X;
    double ancienne_moyenne_Y = moyenne_Y;

225     nombre_observation++;

    /* calcul des nouvelles moyennes */
    moyenne_X = moyenne_X + ((duree - moyenne_X) / (nombre_observation));
230     moyenne_Y = moyenne_Y + ((nombre_arrivee - moyenne_Y) / (nombre_observation));

    if (nombre_observation>1) {
        /* variance de la variable de contrôle */
        variance_Y = ((nombre_observation-2)/(double)(nombre_observation-1)) *
235         variance_Y
            + (nombre_observation) * pow(moyenne_Y-ancienne_moyenne_Y,
                2.0);
        /* covariance entre la variable de contrôle et le temps d'attente de la
           requête */
        covariance = ((nombre_observation-2)/(double)(nombre_observation-1)) *
            covariance
                + (nombre_observation) * (moyenne_X - ancienne_moyenne_X)
                * (moyenne_Y - ancienne_moyenne_Y);
240     return duree + ( -(covariance/variance_Y))*(nombre_arrivee -
        moyenne_var_controle);
    } else {
        return duree;
    }
}
245 }

/*****
/* Cette fonction poursuit la simulation jusqu'à ce qu'elle soit en */
250 /* mesure de retourner x+c(y-mY), le temps d'attente de la requête */
/* suivante, modifié en tenant compte de la variable de contrôle. */
/* Cette variable de contrôle "nombre d'arrivées durant les d * E(A) */
/* unités de temps précédant l'arrivée de la requête" est mesurée au */
/* moment de l'arrivée d'une requête en faisant appel au module liste */

```

```

255 *****
double simulation_prochaine_observation_avec_var_controle(void)
{
    echeancier_type_eve evenement;

260   while(1) {
        evenement = echeancier_retire_prochain_evenement();
        if (evenement.date_evenement < horloge) {
            fprintf(stderr,"Erreur : chronologie non respectée !\n");
            exit(EXIT_FAILURE);
265     }
        horloge = evenement.date_evenement; /* Mise à jour de l'horloge */

        /* on met à jour la liste pour la variable de contrôle en enlevant
           les anciennes requêtes */
270     while (liste_taille() && liste_retourne_tete() < horloge - duree_var_controle
            ) {
                liste_retirer_tete();
            }

        switch(evenement.type) {
275         case arrivee :
            /* nombre de requêtes se trouvant dans la liste de contrôle */
            evenement.nombre_requetes = liste_taille();
            liste_ajouter(horloge); /* on ajoute cette requête dans la liste
            */
            planifie_arrivee(); /* Planifie prochaine arrivée */
280         if (nb_serv_occupes != K) { /* la requête n'attend pas */
                planifie_fin_service();
                /* on retourne son temps de service avec variable de contrôle */
                return temps_avac_variable_controle(0.0, evenement.nombre_requetes);
            } else
285         buffer_ajouter(evenement); /* Doit attendre dans le buffer */
            break;

        case fin_service :
            nb_serv_occupes--; /* Un serveur redevient libre */
290         if (buffer_pas_vide()) {
                evenement = buffer_retirer(); /* Une requête est retirée du buffer */
                planifie_fin_service(); /* et peut donc commencer son service.
            */

                /* on retourne son temps de service avec variable de contrôle */
295         return temps_avac_variable_controle(horloge -
            evenement.date_evenement, evenement.nombre_requetes);
        }
            break;

        default :
300         fprintf(stderr,"Erreur : événement de type inapproprié\n");
            exit(EXIT_FAILURE);
        }
    }
}

```

Listing 2 – statistiques.c

```

/*****
/*
/* Fichier : statistiques.c
/*
5 /* Auteur : E. Thiéard, département E+I, EIVD
/* Version : Janvier 2005
/* Contexte : Modélisation et simulation discrètes, filière IL
/*
/* But : Ce module assure l'analyse statistique des résultats
10 /* à partir des réalisations qui lui sont envoyées. Les
/* mesures étant autocorrélées, on utilise la méthode
/* de la moyenne des lots afin de déterminer un
/* intervalle de confiance à 99% pour la moyenne.
/*
15 *****/
/*
/* Consigne : compléter ce fichier à votre guise
/*
*****/
20 /*
/* Auteurs : Daniel Lifschitz & Nicolas Seriot
/* Date : 7 février 2005
/* modifs : codage de statistiques_nb_obs_encore_a_generer
/*
25 *****/

#include <stdlib.h>
#include <stdio.h>
30 #include <math.h>
#include "statistiques.h"

#define taille_initiale 5000

35 static int taille_allouee;
static double *donnees;
static int nbre_observations;
static double moyenne;
static double variance;
40 static double demi_largeur;
static int taille_lot;
static int nombre_lots;

45 *****/
/* Erreur au niveau allocation dynamique de mémoire
*****/
static void erreur_memoire(void)
{
50 fprintf(stderr,"Problème d'allocation mémoire\n");
exit(EXIT_FAILURE);
}

55 *****/
/* Initialisation du module statistique.
*****/
void statistiques_initialisation(void)
{
60 donnees = (double*) malloc(taille_initiale*sizeof(double));

if (donnees==NULL)
erreur_memoire();
taille_allouee = taille_initiale;
65 nbre_observations = 0;
moyenne = 0.0;
variance = 0.0;

```

```

    taille_lot = 1;
}
70

/*****/
/* Stocker une nouvelle observation. */
/*****/
75 void statistiques_nouvelle_observation(double observation)
{
    if (nbre_observations == taille_allouee) {
        taille_allouee *= 2;
        donnees = (double*) realloc(donnees, taille_allouee*sizeof(double));
80     if (donnees == NULL) {
        erreur_memoire();
    }

}

85     donnees[nbre_observations++] = observation;
}

/*****/
/* Calcul la variance du lot passé en paramètre */
/*****/
90 double variance_lots(double * lots, int nombre_lots) {
    int i;
    double somme = 0.0;
    for (i = 0; i < nombre_lots; i++) {
95     somme += pow(lots[i] - moyenne, 2.0);
    }
    return somme / (double) (nombre_lots - 1);
}

100 /*****/
/* Construit 'nombre_lots' lots de taille 'taille_lot' dans l'espace */
/* mémoire pointé par lots et retourne la moyenne */
/*****/
double moyenne_lots(double * lots, int nombre_lots, int taille_lot) {
105     int i, j;
    double somme = 0.0;
    for (i = 0; i < nombre_lots; i++) {
        lots[i] = 0.0;
        for (j = 0; j < taille_lot; j++) {
110     lots[i] += donnees[i*taille_lot + j];
        }
        lots[i] /= (double) taille_lot;
        somme += lots[i];
    }
115     return somme / (double) nombre_lots;
}

/*****/
/* Retourne l'autocorrélation empirique des lots */
120 /* Attention, la fonction retourne NAN si la variance vaut 0 */
/*****/
double autocorrelation(double * lots, int nombre_lots) {
    int i;
    double somme = 0.0;
125     for (i = 0; i < nombre_lots - 1; i++) {
        somme += (lots[i] - moyenne) * (lots[i+1] - moyenne);
    }
    return (somme / (double) (nombre_lots - 2)) / variance;
}

130

/*****/
/* Essaie de construire un intervalle de confiance à 99% à partir des */
/* observations déjà générées (moyenne des lots). */
135 /* Retourne 0 lorsque la condition d'arrêt est remplie, ou le nombre */

```

```

/* de réalisations qu'il faut encore générer avant d'effectuer une      */
/* nouvelle tentative, dans le cas contraire.                             */
/*****
140 int statistiques_nb_obs_encore_a_generer(void)
{
    double * lots;

    /* il faut au minimum 30 observations */
145     if (nbre_observations < 30) {
        return 30 - nbre_observations;
    }

    /* espace mémoire pour les lots */
150     lots = (double*) malloc(nbre_observations * sizeof(double));
    if (lots == NULL) {
        erreur_memoire();
    }

    /* on commence avec des lots de taille 1 */
155     taille_lot = 1;

    /* tant que les lots contiennent au moins 30 mesures */
    while((nombre_lots = nbre_observations / taille_lot) >= 30) {
160         moyenne = moyenne_lots(lots, nombre_lots, taille_lot);
        variance = variance_lots(lots, nombre_lots);

        /* Attention! la fonction autocorrelation peut retourner NAN
           si la variance vaut 0. Il est donc important de faire le
           test dans ce sens et non pas de la manière suivante :
165         if (fabs(autocorrelation(lots, nombre_lots)) < 0.02)
           faute de quoi le programme entrera dans une boucle
           infinie */
        if (fabs(autocorrelation(lots, nombre_lots)) >= 0.02) {
            taille_lot++;
170         } else {
            demi_largeur = 2.58 * sqrt(variance) / sqrt(nombre_lots);
            free(lots);
            if (demi_largeur <= moyenne * 0.05) {
                return 0;
175             } else {
                return (pow(2.0*demi_largeur, 2.0) / pow(0.1*moyenne, 2.0) -1.0) *
                nbre_observations;
            }
        }
180     }
    free(lots);
    return nbre_observations;
}

185

/*****
/* Affiche les résultats : nombre de réalisations, nombre et taille */
/* des lots, valeur moyenne, demi-largeur et intervalle de confiance */
190 /* avec seuil de confiance à 99%. */
/*****
void statistiques_affiche_resultats(void)
{
195     printf("nombre d'observations : %d\n", nbre_observations);
    printf("nombre de lots          : %d\n", nombre_lots);
    printf("taille d'un lot          : %d\n", taille_lot);
    printf("moyenne                  : %f\n", moyenne);
    printf("variance                  : %f\n", variance);
    printf("demi-largeur                : %f\n", demi_largeur);
200 }

```