Parallel Computing for Data Science
Lab x002 : Clean & Fast Code

Jairo Cugliari

S1 2020–2021

1 Session de débogage

Examinez le code vue en cours (extraction des indexes des rafales sur une suite binaire). Rappelez vous la fonction appelée sur le vecteur $x = (1,0,0,1,1,0,1,1,1)$ et $k = 2$ doit renvoyer les indices (4, 7, 8) (i.e. le début de rafales de longueur 2 ou plus).

```
joe=function(x,k){
n=length(x)
r=NULL
for(i in 1:(n-k)) if(all(x[i:i+k-1]==1)) r<-c(r,i)
r
}
```

2 Débogage de code

Cet exercice décrit une procédure qu'on peut suivre pour tester le code de quelqu'un d'autre.

Un collègue vous envoie cette fonction qui est sensée calculer la somme des premières n entiers :
`f1<-function(n)res<-NULL;for(i in 1:n)res<-res+1`

1. Commencez un nouvel script R, nommez le fichier, rajoutez une en-tête descriptive du contenu du fichier et collez le code de votre collègue.
2. Utilisez les règles de style R pour mieux visualiser la fonction.
3. Testez la fonction à partir de deux ou trois cas simples.
4. Utilisez l'outil de débogage interactif pour corriger le code. Vérifiez que le nouveau code est correct avec les cas simples que vous avez utilisé plus haut.

3 Chronométrage du temps d'exécution

Cet exercice est la continuation l'exercice précédant.

1. Proposez deux autres versions (`f2` et `f3`) de la fonction qui renvoie la somme des premières n entiers en utilisant
   - `sum` pour `f2`, et
   - le résultat connu $\sum_{i=1}^{n} i = n(n+1)/2$ pour `f3`
2. Examinez le temps d'exécution du code pour chacune des trois fonctions. Utilisez `system.time` avec des valeurs de n raisonnablement grandes (cad en fonction du matériel que vous utilisez, cela peux être de l'ordre de 10^5,10^6,10^7,10^8)
3. Pour chaque fonction, déterminez le ressource limitant du calcul (mémoire? disque? réseau? vous même ? =)

4 Simulation par la méthode de rejet

1. Écrire sur `R` la fonction `f_tri(x)` que calcule la valeur de la densité $f(x)$ d'une variable aléatoire triangulaire sur [0,2] pour l'argument x donné en entrée.
2. Écrire sur `R` la fonction `rejection(f, a, b, M)` qui simule une valeur de la densité f en utilisant la méthode de rejet. Nous supposons que f est une densité à support sur le segment $[a,b]$ et qu'elle est majorée uniformément sur son support par M.
3. Simuler 1000 réalisations d'une variable aléatoire triangulaire en utilisant la fonction écrite dans le point précédent. Sauvegarder les valeurs simulés dans un fichier de texte (on les utilisera par la suite).

5 Problème

Le code qui suit doit renvoyer la plus petite distance sur une matrice de distance ainsi que la ligne et la colonne où le minimum se trouve. Vérifiez que la fonction donne le bon résultat. Vous pouvez utiliser comme test la matrice `m <- rbind(c(0,12,5),c(12,0,8),c(5,8,0))`.

```
# returns the minimum value of d[i,j], i != j, and the row/col
# attaining that minimum, for square symmetric matrix d;
# no special policy on ties; motivated by distance matrices

mind <- function(d) {
  n <- nrow(d)
  # add a column to identify row number for apply()
  dd <- cbind(d, 1:n)
  wmins <- apply(dd[-n, ], 1, imin)
  # wmins will be 2xn, 1st row being indices and 2nd being values
  i <- which.min(wmins[1, ])
  j <- wmins[2, i]
  return(c(d[i, j], i, j))
}


# finds the location, value of the minimum in a row x
imin <- function(x) {
  n <- length(x)
  i <- x[n]
  j <- which.min(x[(i + 1):(n - 1)])
  return(c(j, x[j]))
}
```

Exercices additionnels

Chronométrage du temps d'exécution (bis) Calculez les vitesses d'exécution des codes de génération d'une densité triangulaire. Pourquoi la fonction `f_tri2` est plus rapide que la version vectorisée de la fonction `f_tri`?

Lecture

- Debugging, chap. 13 du livre de N. Matloff (cf. Moodle)