

Programmation avec R

Datastorm - B. Thieurmél

Boucles et fonctions

1. Proposer une implémentation du calcul $5!$ (factorielle 5) en utilisant une boucle **for**
2. Proposer une implémentation du calcul $5!$ en utilisant une boucle **while**
3. Proposer une implémentation du calcul $5!$ en utilisant une boucle **repeat**
4. Construire 3 fonctions, **factFor**, **factWhile**, **factRepeat**, prenant en argument un entier n , et renvoyant $n!$
5. Vérifier que vos trois fonctions fonctionnent et renvoient le même résultat...!
6. Construire maintenant une fonction **compareFact**, permettant le calcul factorielle avec, au choix, les 3 types de boucles et fonctions précédentes :
 - deux arguments : un entier n , et le type de boucle **type**, avec la valeur par défaut **for**
 - utiliser un **switch** pour traiter le calcul et le choix de la boucle
 - contrôler l'argument n (type, valeur positive, ...) et renvoyer des erreurs / messages explicites le cas échéant
 - contrôler l'argument **type** en utilisant **try** et **match.arg**
 - gérer le cas $0! = 1$
 - commenter votre fonction suivant la convention **doxygen**
7. Valider en testant les appels suivants :

```
compareFact(5)
compareFact(5, type = "while")
compareFact(5, type = "repeat")
compareFact(5, type = "frf")
compareFact(0)
compareFact(-1)
compareFact("a")
```

8. A l'aide du package **microbenchmark**, comparer les temps de calculs de $100000!$ par type de boucle. Que pouvons-nous conclure dans ce cas ?

```
require(microbenchmark)
?microbenchmark
```

Apply family

Rappelons-nous qu'en **R**, un **data.frame** est en fait une **list**...

```
# données disponibles ici :
# https://raw.githubusercontent.com/arunsrinivasan/flights/NYCflights14/flights14.csv
data <- read.table("flights14.csv", sep = ",", header = TRUE)
```

1. En utilisant un **sapply**, et la fonction **class**, récupérer le type de chaque colonne dans une variable
2. Avec un **apply**, calculer la moyenne des colonnes numériques

```
data$arr_time[1:10] <- NA
```

3. Re-calculer les moyennes en gérant les données manquantes

4. Avec un **apply** et **un seul**, calculer et retourner le minimum et le maximum des colonnes numériques (*indice : utiliser une fonction*)
5. Faire la même chose, mais avec un **sapply**
6. Créer une nouvelle colonne *itinerary*, concaténation des colonnes *origin* et *dest* (**paste**)
7. Calculer le coefficient de variation (ecart-type / moyenne) du temps de vol *air_time* par itinéraire. (Utiliser un **tapply** et un **aggregate** et comparer le temps d'exécution). Quel itinéraire a le plus grand coefficient de variation ?
8. Calculer la vitesse moyenne de l'ensemble des vols ($\text{vitesse} = \text{distance} / \text{air_time}$) des trois façons suivantes, et comparer les temps de calculs (**system.time**)
 - en utilisant une boucle **for**
 - en utilisant les propriétés de la vectorisation
 - en utilisant la fonction **mapply**

Calcul parallèle

```
# données disponibles ici :
# https://raw.githubusercontent.com/wiki/arunsrinivasan/flights/NYCflights14/flights14.csv
data <- read.table("flights14.csv", sep = ",", header = TRUE)
```

1. En utilisant la fonction **split**, créer une liste contenant un sous-jeu de données par mois
2. Avec le package **randomForest** et le modèle ci-dessous, essayer d'expliquer le retard au départ par rapport au jour et aux informations relatives au départ, en faisant un modèle par mois :
 - en utilisant la fonction **lapply**
 - en utilisant le package **parallel** et 2 coeurs
 - en utilisant le package **foreach**, **doParallel** et 2 coeurs

```
require(randomForest)
rf <- randomForest(dep_delay ~ day + dep_time + hour + min, don, ntree = 30)
```

3. Quel est le gain observé en utilisant un calcul parallèle ? (**system.time**)
4. Faire varier le nombre de coeurs en regardant l'évolution du temps de calcul