Comment créer des fonctions dans le langage R

Guillaume Blanchet (Mai 2005, Février 2006)
Marie-Hélène Ouellette (Novembre 2005)
Modifié par Sébastien Durand
11/08/2006

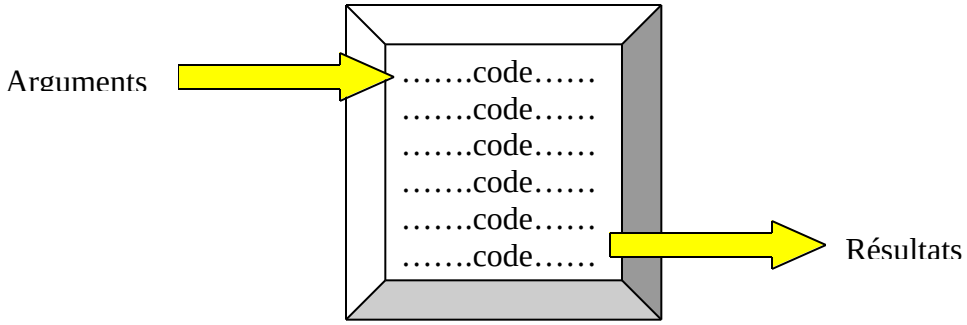
Bio2041
Département de sciences biologiques, Université de Montréal

Qu'est-ce qu'une fonction dans le langage R ?

Avant tout, une fonction est un objet R, au même titre qu'un vecteur, une matrice, un tableau de données ou une liste. Une fonction est une série de commandes qui effectuent une tâche prédéfinie. L'avantage de travailler avec des fonctions, est qu'une fois qu'elles sont écrites, elles permettent d'effectuer rapidement des tâches répétitives, et ce, sur différents jeux de données.

La représentation graphique du fonctionnement d'une fonction

Une fonction peut être vue comme une ''machine'' faite de code. Cette « machine » prend des informations sous forme d'**arguments** et en extrait des **résultats**.



Les notions de base pour créer une fonction

Une fonction se doit d'avoir 4 éléments :

1. Un **nom évocateur** pour vous (le nom est à votre discrétion !!).
2. La fonction **« function »** dans son en-tête, afin d'indiquer à R que l'objet que vous créez est une fonction.
3. Un ou plusieurs **arguments**
    a. Les arguments contiennent les informations qui seront traitées par la fonction. Un argument peut être un vecteur, une matrice, un « `data.frame` » ou bien une liste.
4. Un **script** qui effectue les tâches que la fonction va accomplir.
    a. Ces informations se transmettent à l'ordinateur sous forme de **commandes et de code de programmation**.

La description de la structure d'une fonction

Voici un exemple simple de fonction en R :

```
Ligne 1 --->  Salut <- function( x )
Ligne 2 --->  {
Ligne 3 --->        # La fonction "cat" imprime dans la console
Ligne 4 --->        # "\n" provoque un retour de chariot
Ligne 5 --->        cat("Bonjour", x, "\n")
Ligne 6 --->  }
```

- À la *Ligne 1*, nous avons l'en-tête de la fonction . L'utilisation du mot « `function` » indique à R que **nous créons un nouvel objet,** qui dans le cas présent est une **fonction nommée** « `Salut` ». Les arguments qui seront utilisés par la fonction doivent être nommés et mis entre les parenthèses de l'en-tête. L'utilité de ces **arguments** est simplement de passer des données à la fonction. Donc dans le cas présent, « x » est l'argument qu'utilise la fonction « `Salut` ». Je suppose que vous avez remarqué le symbole d'attribution « `<-` » dans l'en-tête. L'en-tête peut se lire comme suit : « Salut » devient une fonction ayant comme seul argument « x ».
- À la *Ligne 2*, une **accolade est ouverte**. Cette accolade définit le début du **script**. Une fonction normalement constituée ne travaille directement que sur ses arguments et non sur les objets stockés dans la console. L'utilité des **arguments** est de passer au script les objets que nous avons stockés dans notre console et que l'on souhaite voir traités par la fonction. Ces arguments peuvent être de n'importe quel type à condition qu'ils soient définis dans l'en-tête de la fonction. En somme, c'est simple ! **Tous les objets qui seront utilisés dans le script doivent avoir été créés dans le script**. Les seules exceptions à cette règle sont les arguments qui eux sont définis dans l'en-tête.
- À la *Ligne 3* et *Ligne 4*, le symbole « # » est utilisé en début de ligne, il transforme les lignes en **commentaires**. Lorsque la fonction sera exécutée, R ignorera simplement ces lignes.
- À la *Ligne 5*, la fonction « `cat`» est utilisée afin d'imprimer à l'écran l'élément ou les éléments contenus dans l'argument « x ».
- À la *Ligne 6*, nous fermons le script en **fermant l'accolade** et terminons ainsi la fonction.

Afin de démystifier le tout, suivez les instructions décrites dans la prochaine section.

Utiliser une fonction copiée dans la console R[1]

Voici une façon simple de tester et d'utiliser des fonctions que vous créez. **Copiez et collez la fonction « `Salut` » dans la console et tapez la commande d'appel qui suit** :

```
Salut( "Marie-Hélène et Corinne" )
```

Avec une fonction un peu plus complexe, voyons ce que nous pouvons faire.

```
Additionne <- function( a, b )
{
      # On additionne les valeurs a et b
      c <- a + b
      # Imprime le résultat de l'addition
      cat( c , "\n")
}
```

Les arguments de la fonction Salut sont « a » et « b ». Notez que nous décrivons à l'aide de **commentaires**, **la nature et l'utilité des commandes** qu'utilise la fonction « `Additionne` ».

[1] Il se peut que certaines fonctions R soient trop longues pour qu'on les copie-colle directement dans la console. Vous obtenez alors un grand message d'erreur en rouge. Pour remédier à la situation, référez-vous à la section *Deuxième étape : Mettre en mémoire « sourcer » votre fonction* de la page 7!

Copiez maintenant la nouvelle fonction « Additionne » dans la console et **tapez les commandes suivantes :**

```
Obj1 <- 5
Obj2 <- 2
Additionne( a = Obj1, b = Obj2 )
```

Comment retourner les résultats d'une fonction

Toutes les commandes que vous utilisez et inscrivez dans la console R peuvent être intégrées dans le script d'une fonction. Voyons ici une fonction un peu plus complexe qui effectue des tâches mathématiques et renvoie une valeur :

```
NbPneu <- function(voiture=0, bicyclette=0, monocycle=0, tricycle=0)
{
      # Dans cette fonction, nous calculons le nombre total de pneus
      A <- voiture * 4
      # Le Nb de pneus de voiture
      B <- tricycle * 3
      # Le Nb de pneus de tricycle
      C <- bicyclette * 2
      # Le Nb de pneus de bicyclette
      D <- monocycle * 1
      # Le Nb de pneus de monocycle
      NbPneuTotal <- A + B + C + D
      # Retourne le nombre total de pneus
      return( NbPneuTotal )
}
```

Notez ici qu'afin de retourner la valeur stockée dans « `NbPneuTotal` », la fonction « `return( )` » est utilisée. **Cette fonction est nécessaire**. Sans elle, il n'y aurait aucune valeur retournée à la console R. Le fait de retourner une valeur nous permet de stocker en mémoire, ou simplement d'afficher le résultat de la fonction « NbPneu ». De plus, remarquez que dans l'en-tête chacun des arguments a une valeur par défaut de « 0 ». Advenant le cas où la fonction serait utilisée sans l'un de ses arguments, l'argument en question prendrait par défaut la valeur de zéro.

Exercice : Si, dans votre garage vous avez 4 voitures, 12 tricycles, 5 bicyclettes, et 0 monocycle, **combien de pneus avez-vous.** Copiez la fonction dans la console et exécutez la fonction « `NbPneu` » en respectant l'ordre des arguments de la fonction. Notez que si vous n'utilisez pas les noms des arguments avant d'inscrire les valeurs ou les objets passés à la fonction, l'ordre des arguments est crucial. Voyez par vous-même et exécutez les lignes suivantes :

```
NbPneu( voiture=2, bicyclette=0, monocycle=0, tricycle=1 )
# Vous obtenez 11
NbPneu( 2, 0, 0, 1 )          # Vous obtenez 11
NbPneu( voiture=2, tricycle=1 )     # Vous obtenez 11
NbPneu( 2, 1 )                # Oups… le résultat n'est plus le même!
```

Voilà pourquoi nous vous suggérons d'inscrire le nom des arguments que vous utilisez lorsque vous appelez des fonctions.

Créer une fonction permettant de calculer le coefficient de variation

Formule du coefficient de variation

$$\text{C.V.} = \frac{100 \times s_x}{\bar{x}}$$

Code d'une fonction calculant le coefficient de variation (avec commentaires bien sûr) :

```
CoefVar <- function( datum )
{
      # Fonction permettant de calculer le coefficient de variation
      # Applicable seulement pour des données ayant un véritable zéro
      # Calcul de l'écart type
      EcartType <- sd(datum)
      # Calcul de la moyenne
      Moyenne <- mean(datum)
      # Calcul du coefficient de variation
      CV <- 100 * EcartType / Moyenne
      # Retourne le résultat
      return(CV)
}
```

Lorsque vous pensez que votre fonction est sans erreur, enregistrez votre fichier de travail, copiez la fonction dans la console et testez-la.

```
TestData <- c( 1, 2, 3, 4, 5 )
CoefVar( datum = TestData )
```

Utiliser une fonction sauvée dans un fichier .R indépendant

Dans le cadre du TP3, les fonctions que nous allons créer seront écrites dans un fichier texte (TextWrangler) et enregistrées dans le dossier de travail (ou un dossier contenant d'autres fonctions) avec l'extension **.R**. Les usagers courants de R constituent leurs propres bibliothèques de fonctions pour faciliter l'exécution de tâches qu'ils doivent réaliser souvent et pour lesquelles il n'y a pas de commande complète disponible.

Méthodes de travail

Comment ça marche

Lors de la construction d'une fonction, plusieurs fichiers TextWrangler doivent être ouverts. Il est important que chacun soit utilisé de la bonne façon. Voici une méthode de travail pour vous aider à mieux structurer le tout.

Fichier de la fonction	**Fichier de données**	**Guide**

NomFonct<-function(arguments)
{
# Commentaires 	
…
commandes
…
return(LeResultat)
}
	V1 V2…
	2 3
	3 2
	3 4
		# Importe les données
		objet <- read.table(…)
		
		# Lie le fichier de la fonction
		source(Fichier_de_la_fonction…)
		
		# Exécute la fonction
		NomFonct(…)

.R	**.txt**	**.txt**

Fichier de la fonction : Fichier contenant seulement le **code** de la fonction

Fichier de données : Fichier contenant seulement la série de **données** utilisée avec la fonction

Guide : Fichier contenant seulement les commandes et les commentaires. Ce fichier contient les **lignes d'appel de R**. Vous recopierez ce fichier dans R pour obtenir la console R finale.

Première étape : Créer un fichier pour votre fonction

Créez un nouveau fichier à l'intérieur duquel vous écrirez votre fonction. Prenez le soin de sauvegarder immédiatement votre document. Utilisez le nom de votre fonction comme nom de fichier. Dans notre cas, le nom de la fonction que créons est « CentrageReduction », donc le nom du fichier de la fonction est « CentrageReduction.R ». L'extension **.R** permet au langage R de savoir que le fichier contient des commandes R. **Inscrivez ce qui suit dans un fichier TextWrangler et sauvez-le dans votre dossier de travail sous le nom « CentrageReduction.R » .**

```
CentrageReduction <- function( )
{

}
```

Deuxième étape : Mettre en mémoire « sourcer » votre fonction

Dans la console R, nous allons utiliser la commande « source( ) » . **Sourcer** une fonction dans le langage R permet d'enregistrer le code de la fonction pour pouvoir l'utiliser. Pour ce faire, allez dans **Fichier -> Sourcer du code R...** et choisissez votre fonction. Vous pouvez aussi simplement taper dans la console :

```
source("CentrageReduction.R") # Source la fonction CentrageReduction
CentrageReduction( )          # Exécute la fonction CentrageReduction
```

Vous obtiendrez : « NULL », ne vous inquiétez pas, tout fonctionne.

Troisième étape : Tester, modifier, sauver et re-sourcer la fonction de façon efficace

Nous voulons que cette fonction traite un vecteur, le nom de l'argument qu'utilisera la fonction est à votre discrétion, mais ici nous utilisons « x ». Nous ajoutons aussi dans le script une commande qui retournera le résultat de notre fonction. Dans le cas présent nous retournons directement l'argument « x ». N'oubliez pas de sauver votre fichier de fonction après chaque édition, sinon les changements ne seront pas considérés lorsque vous sourcerez votre fonction.

```
CentrageReduction <- function( x )
{
      return( x )
}
```

Maintenant que nous avons modifié notre fonction, sauvez votre fichier « CentrageReduction.R » et tapez ceci dans la console R. Notez que la dernière ligne de commande charge et exécute la fonction.

```
vec <- c(1, 2, 3, 4, 5)            # Crée un vecteur de données
source("CentrageReduction.R"); CentrageReduction( x = vec )
```

Si tout fonctionne, vous obtiendrez :

[1] 1 2 3 4 5

Ajoutons maintenant des commandes à l'intérieur de la fonction.

```
CentrageReduction <- function( x )
{
      # Soustrait la moyenne au vecteur x
      xCentre <- x - mean( x )
      return( xCentre )
}
```

Sauvez le fichier. Ensuite, afin de charger et de tester efficacement la fonction que l'on vient d'éditer, sélectionnez la console R et **appuyez sur la flèche du haut afin de rappeler la dernière commande exécutée dans la console R**. Il ne vous reste plus qu'à appuyer sur « Enter », et voilà, le tour est joué, la fonction évolue au rythme où vous la testez.

```
source("CentrageReduction.R"); CentrageReduction( x = vec )
```

Quatrième étape : Comment trouver les erreurs et les problèmes d'exécution.

Le premier type d'erreur, si la fonction « source("CentrageReduction.R") » vous indique une erreur, il se peut que vous ayez introduit dans la fonction des caractères incompatibles avec R.

```
CentrageReduction <- function( x )
{
      # Soustrait la moyenne au vecteur x
      xCentre<- x - mean( x )
      variance <- var( x )
      EcartType<- variance ¨ ( 1/2 )
      xCentreReduit <- xCentre / Ecart.Type
      return( xCentreReduit )
}
```

```
source("CentrageReduction.R"); CentrageReduction( x = vec )
```

Vous obtenez l'erreur suivante :

```
Erreur dans source("CentrageReduction.R") : caractères multioctets
incorrects dans mbcs_get_next
```

Dans ce cas, masquez avec le « # » toutes les commandes que vous venez d'éditer ou rajouter, **sauf la première**. Sauvez, et testez le fichier afin de voir si la première ligne éditée est correcte. Si « source » fonctionne, enlevez le prochain dièse. N'oubliez pas de sauver et de « `sourcer` » à chaque test.

```
CentrageReduction <- function( x )
{
      # Soustrait la moyenne au vecteur x
      xCentre<- x - mean( x )
      variance <- var( x )
      # EcartType<- variance ¨ ( 1/2 )
      # xCentreReduit <- xCentre / Ecart.Type
      return( xCentreReduit )
}
```

```
source("CentrageReduction.R")      # Tente de sourcer la fonction
```

Ça fonctionne, nous enlevons donc le prochain dièse

```
CentrageReduction <- function( x )
{
      # Soustrait la moyenne au vecteur x
      xCentre<- x - mean( x )
      variance <- var( x )
      EcartType<- variance ¨ (1/2)
      # xCentreReduit <- xCentre / Ecart.Type
      return( xCentreReduit )
}
```

```
source("CentrageReduction.R")      # Tente de sourcer la fonction
```

L'erreur réapparaît ! Nous avons donc trouvé l'emplacement de l'erreur. Nous remplaçons le symbole erroné « ¨ » par le bon symbole « ^ ». Une fois l'erreur corrigée, nous pouvons donc enlever les dièses subséquents.

```
CentrageReduction <- function( x )
{
      # Soustrait la moyenne au vecteur x
      xCentre<- x - mean( x )
      variance <- var( x )
      EcartType<- variance ^ (1/2)
      xCentreReduit <- xCentre / Ecart.Type
      return( xCentreReduit )
}
```

```
source("CentrageReduction.R")      # Tente de sourcer la fonction
```

Maintenant, que la fonction est sourcée, nous pouvons la tester.

```
CentrageReduction( x = vec )
```

Hey oui, une autre erreur suivante apparaît, j'ai fait exprès! Voici l'erreur! :

```
Erreur dans CentrageReduction(x = vec) : objet "Ecart.Type" non trouvé
```

Advenant le cas ou n'avez aucune idée où se trouve l'erreur, nous allons utiliser la fonction « `print` ».

```
CentrageReduction <- function( x )
{
      # Soustrait la moyenne au vecteur x
      print(1)
      xCentre<- x - mean( x )
      print(2)
      variance <- var( x )
      print(3)
      EcartType<- variance ^ (1/2)
      print(4)
      xCentreReduit <- xCentre / Ecart.Type
      print(5)
      return( xCentreReduit )
}
```

```
# Source et on teste la fonction
source("CentrageReduction.R"); CentrageReduction( x = vec )
```

Vous obtenez :

```
[1] 1
[1] 2
[1] 3
[1] 4
Erreur dans CentrageReduction(x = vec) : objet "Ecart.Type" non
trouvé
```

Ce qui vous indique que l'erreur est apparue après le « `print(4)` ». Effectivement, il y a un point de trop dans « Ecart.Type ». Corrigez-le et n'oubliez pas de sauver votre fichier.

Cinquième étape : La touche finale

Après avoir testé votre fonction, sachez qu'il est très important d'ajouter des commentaires à votre fonction, ces commentaires vous faciliteront énormément la vie lorsque vous ouvrirez de nouveau ces documents dans un futur plus ou moins lointain.

```
CentrageReduction <- function( x )
{
      # Cette fonction centre et réduit les données d'un vecteur
      # nommé x. Elle effectue la même tâche que la fonction «scale».

      # Soustrait la moyenne au vecteur x
      xCentre<- x - mean( x )
      # Calcule la variance
      variance <- var( x )
      # Trouve le l'écart type
      EcartType<- variance ^ (1/2)
      # Complète le centrage et la réduction des données
      xCentreReduit <- xCentre / EcartType
      # Retourne le vecteur de données centrées et réduites
      return( xCentreReduit )
}
```

```
 # Teste la dernière version de notre fonction
source("CentrageReduction.R"); CentrageReduction( x = vec )
```

Vous pouvez comparer vos résultats avec ceux de la fonction « scale ». S'ils sont identiques et bien, vous avez réussi. Bravo!

```
# Teste la dernière version de notre fonction
as.vector( scale( vec ) )
```