

Introduction à R : Exercices d'application

Leslie Regad & Jennifer Becq

September 25, 2007

Exercices de base : Structures de données

Les vecteurs

1. Créer le vecteur d'entiers x allant de 1 à 20
2. Calculer le carré des 5 premiers éléments de x
3. Créer le vecteur Sx contenant le sinus de chaque élément de x
4. Calculer la somme et la somme cumulée de x . (fonctions `sum()`, `cumsum()`)
5. Créer le vecteur $xneg$ des indices des valeurs négatives de Sx (fonction `which()`). Donner le nombre de valeurs négatives. Remplacer les valeurs négatives de x par 0.
6. Créer les vecteurs $x.even$ contenant les éléments d'indice pair de x , et $x.odd$ les éléments de x d'indice impair. Créer avec ces deux vecteurs une matrice à deux colonnes.(fonction `cbind()`)
7. Créer 2 vecteurs aléatoires $x1$ et $x2$ de taille N dont les éléments suivent une loi normale centrée réduite et une loi uniforme définie sur l'intervalle $[0,10]$ pour $N = 10, 100, 1000$. (fonctions `rnorm` et `runif`)
8. Quelles sont les valeurs minimale et maximale de $x1$ pour $N = 100$ (fonctions `max()`, `min()`). Calculer le nombre d'éléments positifs de $x1$, la moyenne arithmétique et la variance, et l'écart-type des éléments de $x1$ pour $N = 100$ (fonctions `mean()`, `var()`, `sd()`).
9. Créer une séquence d'ADN aléatoire s de 1000 nucléotides (fonction `sample()`). Créer un vecteur *composition* qui contient le nombre de chacun des nucléotides dans la séquence (fonction `table()`). Créer les vecteurs indicateurs s_A, s_C, s_G et s_T tels que $(s_A)_i = 1$ si $s_i = "A"$ et $(s_A)_i = 0$ sinon (idem pour les autres).

Les matrices

1. Créer la matrice identité de taille 10×10 (des 1 sur la diagonale et des 0 ailleurs).
2. Créer une matrice aléatoire de taille 10×10 dont les éléments suivent une loi normale de moyenne nulle et de *variance* 5.
3. Créer une matrice aléatoire de taille 10×10 dont les éléments de la colonne j suivent une loi normale de moyenne nulle et de variance j^2 (faire une boucle). Calculer le nombre d'éléments positifs et négatifs dans la matrice. Remplacer les nombres négatifs par 0. Calculer la matrice de taille 11×11 dont la dernière colonne contient la moyenne de chacune des lignes et la dernière ligne la variance de chacune des colonnes (fonction `apply`).
4. Créer la matrice de taille 10×10 par exemple telle que

$$A_{i,i} = 2, A_{i,i+1} = -1 \text{ et } A_{i+1,i} = -1$$

(penser aux sous-matrices)) et le vecteur b tel que $b_i = 1$. Calculer le déterminant et les valeurs propres de A et résoudre le système linéaire $Ax = b$.

Graphisme

1. Afficher la fonction *sinus* entre $-\pi$ et π avec un titre, et des labels sur les axes. Exporter le graphe avec la fonction *postscript*.
2. Afficher la fonction à deux variables $f(x, y) = \sin(x)\cos(y)$ sur le domaine $[-\pi, \pi] \times [-\pi, \pi]$ avec la fonction *persp* et avec la fonction *image*. Essayer aussi la fonction *filled.contour*.
3. Créer un vecteur N dont les éléments suivent une loi normale centrée réduite. Représenter la distribution des éléments des vecteurs (fonction `hist()`). Superposer sur ce graphique la courbe de la loi normale centrée réduite (fonction `dnorm()`).
4. Reproduire le graphe représenté sur la figure (??).

Exportation des données

1. Générer aléatoirement une séquence de 100 acides aminés. Pour obtenir la liste des acides aminés utiliser la fonction `letters()` et l'indexation négative.
2. Sauvegarder cette séquence dans un fichier "test.out" avec les fonctions `cat()`, `write()`, et `write.table()`. Expliquer les différences

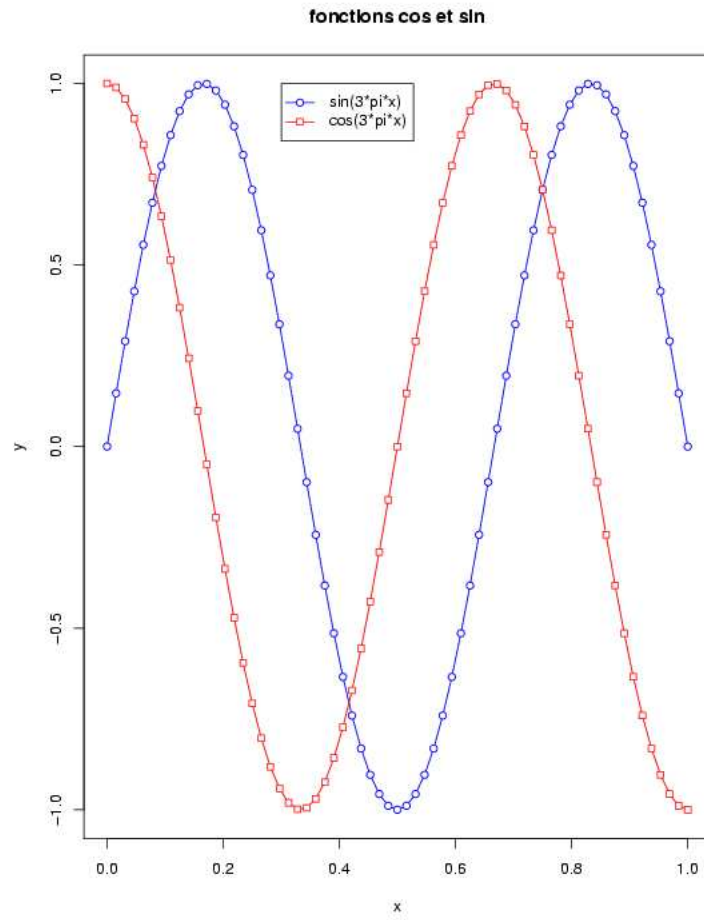


Figure 1: Exemple de graphes programmé avec les fonctions `plot`, `lines`, `points`, `legend`

Fonctions

1. Ouvrir le fichier "seq.aa" en utilisant la fonction `readLines()`. Ce fichier contient 2 séquences protéiques.
2. Faire une fonction `composition` qui permet de déterminer la composition en acides aminés d'une séquence protéique `s` et qui retourne l'acide aminé le plus fréquent (fonction `unlist(strsplit())`)
3. Utiliser cette fonction pour déterminer l'acide aminé le plus fréquent des 2 séquences "seq.aa". (fonction `lapply`)

Application 1: Calcul du temps d'arrêt

Lors d'un freinage d'urgence, le temps que met une voiture à s'arrêter se décompose en 2 parties:

- Le temps de réaction du conducteur: temps nécessaire au conducteur pour prendre conscience de la situation et d'appuyer sur le frein. Ce temps est généralement estimé à une seconde.
- Le temps que met la voiture à s'arrêter complètement

Pour calculer la distance d'arrêt d'un véhicule, il faut donc additionner :

- La distance parcourue pendant le temps de réaction (pendant 1 sec: DR)
- La distance de freinage DF

calcul de la distance de freinage: DF

La distance de freinage dépend de la vitesse (exprimée en m/s) et de la décélération. La décélération s'exprime en m/s^{-2} et varie en fonction de l'adhérence des pneus à la route. Les véhicules actuels permettent des décélération sur route sèche de 10 m/s^{-2} environ. La distance de freinage se calcule de la manière suivante:

$$DF = \frac{v^2}{2a} \quad (1)$$

avec v la vitesse en m/s

a la constante de décélération en m/s^{-2} (ici 10).

calcul de la distance d'arrêt:DA

La distance d'arrêt (en m) s'obtient donc par la formule:

$$DA = DF + DR \quad (2)$$

avec DF et DR en mètres.

Un autre moyen, approximatif, de calculer la distance d'arrêt est de diviser le carré de la vitesse exprimée en km/h par 100.

PARTIE I:

La question que l'on se pose est :

est-ce que l'utilisation de l'approximation entraîne une erreur sur le calcul de la distance d'arrêt ?

Pour répondre à cette question, on vous propose de calculer la distance d'arrêt en utilisant les 2 approches pour les vitesses suivantes : 30, 50, 90, 110, 130 et 200 km/h.

1. Créer le vecteur `v.lim` contenant les différentes vitesses. *fonction `c()`*.
2. Calculer la distance d'arrêt en utilisant la méthode d'approximation
3. Calculer la distance d'arrêt en utilisant l'approche exacte:
 - (a) Calculer le vecteur `v.lim2` qui contient les valeurs des vitesses converties en m/s
 - (b) Calculer à partir de ce vecteur la distance d'arrêt
4. Comparaison des 2 approches: plotter sur le même graphique l'évolution de la distance d'arrêt en fonction de la vitesse (en km/h). Plotter en rouge les valeurs de la distance d'arrêt calculés avec l'approche exacte et en bleu celle calculées avec l'approximation. *`plot()`, `axis()`...*
 - titre: $DA = f(\text{vitesse})$
 - axe des Y: distance d'arrêt (en m)
 - axe des X: vitesse (km/h)

PARTIE II

La distance d'arrêt peut varier en fonction de l'état de la voiture, suivant le type de voiture.... Pour prendre en compte ce paramètre, nous allons ajouter du bruit lors du calcul de la distance d'arrêt.

Dans ce cas cette distance d'arrêt DA_B se calcule de la manière suivante:

$$DA_B = DA + B \quad (3)$$

Avec DA calculé en utilisant l'approche exacte

Avec B qui suit une loi normale de moyenne 0 et de variance $\frac{DA}{10}$

remarque: En R le tirage aléatoire d'un nombre dans une loi normale de moyenne m et de variance v se fait en utilisant la fonction *`rnorm(nb, moyenne, sd)`*.

1. pour les vitesses suivantes: 30, 50, 90, 110, 130 et 200 km/h calculer la distance d'arrêt pour 30 individus. Donner les résultats sous la forme du tableau ??:
faire une boucle sur les vitesses (`for`), concaténation de vecteur (`cbind()` ou `rbind()`).

vitesse	30	50	90	110	130	200
ind1						
ind2						

Table 1:

- calculer pour chaque vitesse la distance moyenne (*mean*) et l'écart type (*sd*) de la distance. Ne pas faire une boucle sur les vitesses, utiliser la fonction *apply()*.
- plotter l'évolution de la distance en fonction de la vitesse. Pour cela, transformer la matrice de distance d'arrêt en vecteur (vecteur *Y* dans le plot). Créer le vecteur Vect.V (vecteur *X* dans le plot) qui contient les vitesses. On sait qu'on a 30 fois chaque vitesse (*rep()*).
Sur ce plot ajouter pour chaque vitesse la moyenne (en rouge) (*lines()*).
- Sur une même feuille plotter pour chaque vitesse la distribution des distances d'arrêt.
- Un obstacle se trouve à 70m de la voiture, quels sont les individus de la matrice qui s'arrêtent à temps ? (*utiliser la fonction which()*)

Application 2: Etude des RMSD

On considère que 2 fragments protéiques de même séquence en acides aminés peuvent avoir des formes (géométrie) différentes.

A partir d'une banque de structures protéiques, nous avons extrait toutes les structures 3D correspondantes à 36 séquences de 7 acides aminés.

Le but de ce travail est de retrouver les fragments qui sont proches en termes de structures/géométrie. Pour cela, nous avons déterminé la matrice de RMSD (= Root mean squared deviation: distance euclidienne entre les coordonnées des carbones alpha de 2 fragments). Cette matrice contient 2 types de RMSD:

- RMSD intra: RMSD entre 2 fragments de même séquence. Ce critère permet de quantifier la variabilité de structure pour un fragment. On considère qu'un fragment est structuré (variabilité faible) si le RMSD intra est inférieur à 0.5Å.
- RMSD inter: RMSD entre 2 fragments de séquences différentes. Il permet de déterminer si 2 fragments sont proches en termes de structure/géométrie (= RMSD inter < 1Å)

- Ouverture de la matrice de RMSD *Mat_rmsd.res.* (*read.table()*)

2. Est ce que certains motifs sont fortement conservés en termes de structure (RMSD intra < 0.5Å) ou est ce que certains motifs sont flous (RMSD intra >1Å). (*which()*)
3. Pour chaque fragment calculer le RMSD moyen et la variance (*mean()*, *var()*)
4. Représenter sur le même graphique le RMSD intra et le RMSD moyen de chaque fragment
5. Représenter l'image de la matrice (*image()*).
6. Extraire pour chaque motif, le fragment qui le plus proche en termes de structures (RMSD inter le plus petit).
7. Créer un tableau comportant : le nom du fragment, le RMSD intra, le RMSD moyen, la variance du RMSD, le fragment le plus proche. Exporter ce fichier sous le nom de *Res_rmsd_res.dat*
8. Combien de fragments ont au moins un RMSD inter inférieur au RMSD intra
9. On voudrait savoir si certains motifs peuvent être regroupés en termes de structures. Pour cela, réaliser une classification hiérarchique des 36 fragments à l'aide du RMSD inter.
 - Remplacer les RMSD intra par 0
 - Réaliser la classification hiérarchique *hclust()*, *plot()*
 - Si on considère que 2 motifs sont proches, si le RMSD entre les 2 fragments est inférieur à 1Å, est ce que certains motifs peuvent être regroupés. (*cutree()*)

Application 3: Etude de la composition des séquences protéiques

On dispose du fichier "Seq_aa.dat" qui contient les séquences de chaînes protéiques appartenant à la famille des Immunoglobulin/albumin-binding domain-like d'après la classification SCOP .

On va étudier la composition en acides aminés de ces séquences en analysant la sur- ou sous-représentation des mots composés de 7 acides aminés. Pour déterminer si un mot est sur- ou sous-représenté, on compare son occurrence observée Occ_{Obs} et théorique Occ_{Theo} à l'aide d'un Zscore.

Le Zscore du mot w se définit par :

$$\frac{Occ_{Obs} - Occ_{Theo}}{\sqrt{Occ_{Theo}}}$$

Un mot est sur-représenté si son Zscore est supérieur à 1.96 x

1. Déterminer les occurrences des différents mots (correspond à une série de 7 acides aminés) fonction `substring()`; `table()`. Combien de mot sont vus plus d'une fois?
2. Calculer le Zscore des mots vu au moins 2 fois. Créer une fonction qui calcul le Zscore d'un mot w

- Effectif théorique: $Occ_{Theo}[w] = \text{nbr de positions possibles dans la séquence} \times \text{la probabilité d'apparition du mot}$
- Probabilité d'apparition $P(w)$ du mot w de taille n est définie par:

$$P(w) = \prod_{aa \in w} [p(aa)]$$

avec $p(aa)$ la fréquence de l'acide aminé aa .

- Fréquence de chaque acide aminé: Pour l'ensemble des séquences calculer l'occurrence de chaque acide aminé et en déduire sa fréquence.
 - Calcul du Zscore
3. En prenant un seuil de 1.96, combien de mots sont sur-représentés dans ces séquences. Pourquoi?