Ecole de Recherche CIMPA
Analyse et Probabilités
Université FHB, Abidjan
17-28 mars 2014

TP cours 7

Le but de ces séances de TP est de se familiariser avec le logiciel R et de voir la méthode de la matrice circulante pour simuler un processus gaussien stationnaire, avant de l'appliquer au mouvement brownien fractionnaire. Le logiciel R, principalement dévolu aux applications statistiques est libre et gratuit, il est disponible pour Linux, Windows, et MacOS sur le site de référence http ://cran.r-project.org. Une grande partie des commandes introductives est tirée de [2, 3].

1 Premiers pas avec R

1.1 L'interface RStudio

L'interface RStudio, disponible gratuitement sur https ://www.rstudio.com/ est composée de quatre fenêtres :

- Fenêtre d'édition (en haut à gauche) : dans cette fenêtre apparaissent les fichiers contenant les scripts R que l'utilisateur est en train de développer. Enregistrer le fichier avec une extension .R permet une coloration syntaxique adaptée au langage R. En entête de cette fenêtre, des icônes permettent de sauvegarder le fichier, d'exécuter un morceau de code sélectionné (icône run) ou l'intégralité du code contenu dans le fichier (icône source).
- Fenêtre de commande (en bas à gauche) : cette fenêtre contient une console dans laquelle les codes R sont saisis pour être exécutés.
- Fenêtre espace de travail / historique (en haut à droite) : contient les objets en mémoire, que l'on peut consulter en cliquant sur leur noms, ainsi que l'historique des commandes exécutées,
- Fenêtre explorateur / graphique / package / aide (en bas à droite) : l'explorateur permet de se déplacer dans l'arborescence des répertoires, la fenêtre graphique contient les graphiques tracés via R (il est possible de les exporter), la fenêtre package montre les packages installés et actuellement chargés et la fenêtre d'aide contient la documentation sur les fonctions et packages.

1.2 Le répertoire de travail

Le répertoire de travail est celui à partir duquel vous avez lancer l'interface RStudio. Il sera pratique de se placer dans un répertoire de travail bien défini, celui par exemple contenant le fichier .R dans lequel vous tapez vos scripts R. Pour ce faire, vous pouvez soit utiliser la commande `setwd` pour vous déplacer dans l'arborescence des répertoires, soit utiliser le menu de l'interface :
-Session
-Set Working Directory
-To Source File Location
Par la suite, lorsque vous serez amené à charger des jeux de données, si ceux-ci sont placés dans le répertoire courant dans lequel vous vous êtes placé, vous n'aurez pas à saisir le chemin complet de ce répertoire.

1.3 Les packages

Un grand nombre de fonctions, contenus dans différents packages, sont installés dans la version de base du logiciel R. Il est possible d'installer des packages supplémentaires. Pour cela, lorsque vous disposez d'une connexion internet, il suffit d'utiliser la commande suivante en indiquant le nom du package que l'on veut installer :

```
R>  install.packages('Matrix')
```

RStudio vous proposera alors de choisir le serveur à utiliser pour télécharger le package et procédera ensuite à l'installation. Il faudra ensuite charger le package à l'aide de la commande `library()` :

```
R>   library('Matrix')
```

L'installation n'est à réaliser qu'une seule fois, alors que le chargement du package doit être fait au lancement de chaque nouvelle session.

2 Premières commandes

Le ">" à la dernière ligne de la fenêtre de commande (en bas à gauche) est le *prompt* de R, il vous indique que vous êtes actuellement en train d'utiliser le logiciel R.

Lorsque vous voulez quitter R il vous suffit d'appuyer sur les touches CTRL+. R vous demandera si vous souhaitez sauvegarder l'environnement ; répondez non :

```
>
Save workspace image? [y/n/c]: n
```

La fonction `q()` permet également de quitter R.

2.1 Découvrir le mode interactif de R

Depuis le prompt de R vous pouvez exécuter toutes les commandes déjàă disponibles et même définir vos propres fonctions, mais avant d'en arriver làă nous allons d'abord nous familiariser avec les commandes de bases de R.

Dans R tout est vecteur, par exemple si vous tapez 5 ou pi depuis le prompt, puis la touche entrée, le résultat affiché précise qu'il s'agit de la première coordonnée ([1]) d'un vecteur unidimensionnel 5 ou pi :

```
> 5
[1] 5
> pi
[1] 3.141593
> z=2+1i*3
> Re(z)
[1] 2
> Im(z)
[1] 3
```

Un vecteur peut être à valeurs numériques ou entières, comme dans les exemples précédents. Il peut aussi être àă valeurs alphanumériques :

```
> "a"
[1] "a"
> "ab"
[1] "ab"
```

Ou encore à valeurs booléennes :

```
> 0<1
[1] TRUE
> 0==1
[1] FALSE
```

Pour construire un vecteur, on peut procéder par concaténation en utilisant la fonction `c` :

```
> c(2,3,5)
[1] 2 3 5
> c(2,3,5)[1]
[1] 2
> c(2,3,5)[2]
[1] 3
> c(2,3,5)[c(1,2)]
[1] 2 3
```

Astuce : pour rappeler une commande précédente, on peut utiliser les flèches verticales qui dirigent le curseur.

On peut aussi construire un vecteur numérique en utilisant la commande `m:n` :

```
> 1:30
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
[21] 21 22 23 24 25 26 27 28 29 30
> 10:-10
 [1]  10   9   8   7   6   5   4   3   2   1   0  -1  -2  -3  -4
[16] -5 -6  -7  -8 -9 -10
> c(1:2,4)
[1] 1 2 4
```

On peut additionner deux vecteurs numériques de même longueur, additionner un vecteur et un scalaire (vecteur de longueur 1), multiplier deux vecteurs coordonnée par coordonnée, multiplier un vecteur par un scalaire, etc.

```
> 1:3+1:3
[1] 2 4 6
> 1:3+3
[1] 4 5 6
> 1:3*1:3
[1] 1 4 9
> 1:3*3
[1] 3 6 9
> 1:3==3:1
[1] FALSE  TRUE FALSE
> 1:3==3
[1] FALSE FALSE  TRUE
```

On peut affecter un vecteur àă une variable grâce à l'instruction `<-` ou `=` (dans certains cas, cette dernière instruction peut se révéler défectueuse) :

```
> a=3
> a
```

```
[1] 3
> a=a+2
> a
[1] 5
> alphabet<-c("a","b","c")
> alphabet
[1] "a" "b" "c"
> alphabet[2]
[1] "b"
> alphabet[2]<-"z"
> alphabet
[1] "a" "z" "c"
```

Dans ce dernier exemple, le vecteur doit avoir été défini auparavant :

```
> chiffre[1]<-5
Error: Object "chiffre" not found
> chiffre<-0
> chiffre[1]<-5
> chiffre
[1] 5
```

On peut également créer une matrice à partir de ses vecteurs colonnes

```
> x1=c(1,2,3)
> x2=c(3,4,5)
> matrix(c(x1,x2),3,2)
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[3,]    3    5
```

Résumé manipulation de données

- `x[n]` : n-ème élément du vecteur x.
- `x[n:m]` : n-ème au m-ème éléments du vecteur x.
- `x[c(k,l,m)]` : k-ème, l-ème et m-ème éléments du vecteur x.
- `x[x>m & x<n]` : éléments de x compris entre m et n.
- `l$x` ou `l[["x"]]` : élément x de la liste l.
- `M[i,j]` : élément ligne i et colonne j de la matrice M.
- `M[i,]` : i-ème ligne de la matrice M.
- `t(M)` : transposée de la matrice M.
- `solve(M)` : inverse de la matrice M.
- `M%*%N` : produit des matrices M et N.
- `sort(x)` : tri du vecteur x.

2.2 Fonctions

Un grand nombre de fonctions sont disponibles sous R, présentes dans le module de base ou dans des bibliothèques séparées. Elles peuvent avoir zéro, un ou plusieurs arguments, et renvoient un vecteur ou un objet de structure plus élaborée. La syntaxe générale est `nom_de_fonction(arg1,arg2,...)`. L'une d'entre elles a déjà été vue : `c`. Voici quelques autres fonctions qui pourront être utiles :

- `seq` : cette fonction permet de construire des suites de nombres équidistants

```
> seq(0,1,length=11)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> seq(0,1,by=.2)
[1] 0.0 0.2 0.4 0.6 0.8 1.0
> seq(0,1,by=.3)
[1] 0.0 0.3 0.6 0.9
```

– `rep` : cette fonction permet de construire des suites de nombres par répétition d'un même vecteur

```
> rep(1,5)
[1] 1 1 1 1 1
> rep(1:4,2)
[1] 1 2 3 4 1 2 3 4
> rep(1:4,each=2)
[1] 1 1 2 2 3 3 4 4
> rep(1:4,length=10)
 [1] 1 2 3 4 1 2 3 4 1 2
> rep(1:4, c(2,1,2,1))
[1] 1 1 2 3 3 4
```

– `sort(v)` : cette fonction renvoie un vecteur dont les coordonnées sont celles de `v` après un tri croissant ou décroissant

```
> sort(4:1)
[1] 1 2 3 4
> sort(1:4, dec=TRUE)
[1] 4 3 2 1
> sort(c("c","a","b"))
[1] "a" "b" "c"
```

– `sum(v)` : cette fonction effectue la somme des coordonnées du vecteur `v` ; si ce vecteur est booléen, alors `sum(v)` est égal au nombre de ses coordonnées "vraies"

```
> sum(1:5)
[1] 15
> sum(1:2<2:1)
[1] 1
```

– `cumsum(v)` : cette fonction effectue la somme cumulée des coordonnées du vecteur `v`

```
> cumsum(1:5)
[1]  1  3  6 10 15
```

– `sample(v)` : cette fonction permet d'effectuer des tirages aléatoires uniformes dans l'ensemble des coordonnées du vecteur `v`, avec remise ou non

```
> sample(1:4)
[1] 3 1 2 4
> sample(1:4,1)
[1] 3
> sample(1:4,2)
[1] 2 4
> sample(1:4,5)
Erreur dans sample(length(x), size, replace, prob) :
        impossible de prendre un échantillon plus grand
que la population lorsque 'replace = FALSE'
> sample(1:4,5,rep=TRUE)
[1] 1 3 2 4 4
```

– `print(v)` : cette fonction permet d'afficher le vecteur `v` ; elle est utile dans une boucle ou lors de l'exécution d'un programme écrit dans un fichier
```
> print(pi)
[1] 3.141593
```
– `runif(1)` : cette fonction servira de générateur de nombres aléatoires ; elle renvoie un réel tiré aléatoirement et uniformément sur le segment $[0, 1[$; chaque appel de la fonction correspond àă un tirage indépendant
```
> runif(1)
[1] 0.5569725
> runif(1)
[1] 0.02946733
```
– `as.numeric(v)` : cette fonction transforme un vecteur booléen `v` en vecteur numérique
```
> as.numeric(TRUE)
[1] 1
> as.numeric(FALSE)
[1] 0
> as.numeric(1:2<2:1)
[1] 1 0
```

Astuce : lorsque vous commencez à taper le nom de la fonction, vous pouvez en appuyant sur la touche tabulation voir les différentes fonctions commençant par les lettres déjà saisies. Lorsque le nom de la fonction est totalement saisi, la tabulation permet de voir les arguments attendus par la fonction.

L'aide sur une fonction est accessible des deux façons suivantes :

```
>  help(rnorm)
> ?rnorm
```

Astuce : un bon moyen pour trouver de l'aide et des exemples sur une fonction consiste simplement à taper le nom de la fonction sous Google.

Information sur les variables

– `length` : longueur d'un vecteur.
– `ncol, nrow` : nombre de colonnes et de lignes d'une matrice.
– `str` : affiche le type d'un objet.
– `as.numeric, as.character` : change un objet en un nombre ou une chaine de caractères.
– `is.na` : teste si la variable est de type 'NA' (valeur manquante).

Statistiques

– `sum` : somme d'un vecteur.
– `mean` : moyenne d'un vecteur.
– `sd, var` : écart-type et variance d'un vecteur (dénominateur $n - 1$)
– `rowSums, rowMeans, colSums ou colSums` : somme et moyenne en ligne ou en colonne d'une matrice.
– `max, min` : maximum et minimum d'un vecteur.
– `quantile(x,0.1)` : quantile d'ordre 10% du vecteur x.

2.3 Fonctions de simulation de variables aléatoires non uniformes

Un large éventail de fonctions donne directement accès aux caractéristiques de plusieurs lois de probabilité dans R. Pour chaque racine loi, il existe quatre fonctions différentes :

1. `dloi` calcule la fonction de densité de probabilité (loi continue) ou la fonction de masse de probabilité (loi discrète) ;

2. `ploi` calcule la fonction de répartition ;
3. `qloi` calcule la fonction de quantile ;
4. `rloi` simule des observations de cette loi.

Différentes lois de probabilité disponibles dans le système R de base, leur racine et le nom de leurs paramètres sont rassemblés au tableau suivant.

Loi de probabilité	Racine dans R	Noms des paramètres
Binomiale	`binom`	size, prob
Géométrique	`geom`	prob
Poisson	`pois`	lambda
Uniforme	`unif`	min, max
Normale	`norm`	mean, sd
Khi carré	`chisq`	df
F (Fisher)	`f`	df1, df2
t (Student)	`t`	df
Exponentielle	`exp`	rate
Gamma	`gamma`	shape, rate ou scale

Toutes les fonctions du tableau sont vectorielles, c'est-à-dire qu'elles acceptent en argument un vecteur de points où la fonction (de densité, de ré- partition ou de quantile) doit être évaluée et même un vecteur de paramètres. Par exemple,

```
> dpois(c(3, 0, 8), lambda = c(1, 4, 10))
[1] 0.06131324 0.01831564 0.11259903
```

retourne la probabilité que des lois de Poisson de paramètre 1, 4 et 10 prennent les valeurs 3, 0 et 8, dans l'ordre. Le premier argument de toutes les fonctions de simulation est la quantité de nombres aléatoires désirée. Ainsi,

```
> rpois(3, lambda = c(1, 4, 10))
[1]  0  3 10
```

retourne trois nombres aléatoires issus de distributions de Poisson de paramètre 1, 4 et 10, respectivement.

3 Aller un peu plus loin avec R

Nous allons maintenant voir comment définir des fonctions et l'utilisation de la fonction `source` pour charger des fonctions depuis un fichier.

3.1 Les fonctions dans R

Il est possible de définir de la manière suivante une fonction :

```
> nom_de_la_fonction <- function( arg_1, arg_2, ...) {
+ # Corps de la fonction
+ }
```

Vous pouvez ensuite appeler la fonction que vous avez définie de la même façon que les fonctions déjàă définies dans R.

Un exemple :

```
> addition <- function(n) {
+ n + 1
}
> addition(4)
[1] 5
```

La valeur retournée par la fonction sera toujours la valeur de la dernière instructions de la fonction !

Remarque : le symbole `#` permet d'écrire des commentaires : tout ce qui est écrit après un `#` ne sera pas interprété par R.

3.2 Utiliser des fichiers avec la fonction `source`

Il est plus pratique d'écrire dans un fichier une suite d'instruction R, plutôt que d'utiliser le mode interactif. Allez dans votre répertoire de travail, puis créez un fichier vierge "test1.R". Les *scripts* R ont pour extension la lettre R, comme vous l'aurez deviné. Il ne vous reste plus qu'a ouvrir ce fichier avec votre éditeur de texte préféré.

Exercice 1. *Ecrivez dans le fichier que vous venez d'ouvrir la définition d'une fonction "produit" permettant de multiplier deux nombres, et enregistrez le fichier. Puis depuis l'interpréteur R exécutez la commande suivante :* `source("test1.R")` *Que se passe-t-il ?*

La fonction `source` permet de lire un fichier et d'exécuter les instructions qui s'y trouvent. Si vous définissez des fonctions dans votre fichier elles seront visibles dans R une fois le fichier chargé via la commande `source`. Il est cependant nécessaire de recharger le fichier à chaque modification de son contenu, toujours avec la commande `source`.

3.3 Programmation

Les structures de contrôle sous Pascal (alternatives et répétitives) existent sous R, avec une syntaxe très proche

- `if(`*`booléen`*`) {...} else {...}`
- `for(`*`variable`* `in` *`vecteur`*`) {...}`
- `while(`*`booléen`*`) {...}`
- `repeat {...}`

Remarques :

1. L'instruction `break` permet de terminer une boucle ; c'est la seule façon de quitter une boucle `repeat`.
2. L'instruction `next` permet de passer directement au cycle suivant.
3. Deux commandes successives doivent être séparées par un point-virgule, ou être sur deux lignes distinctes.
4. Les connecteurs logiques sont `!` pour “non”, `&` ou `&&` pour “et”, `|` ou `||` pour “ou”, et `xor` pour “ou exclusif”.
5. Les prédicats élémentaires sont `==`, `!=`, `<=`, `<`, `>=`, et `>`.

3.4 Graphiques

Le logiciel R permet aussi les sorties graphiques. La fonction `plot(vec1,vec2)` ouvre une fenêtre graphique et affiche les points d'abcisses `vec1` et d'ordonnées `vec2`

```
> x <- seq(-pi,pi,.01); y <- cos(x); plot(x,y)
```

La fonction `points`, avec les mêmes arguments et les mêmes résultats que la fonction `plot`, permet d'afficher des points dans une fenêtre graphique déjà ouverte.

```
> plot(x,y)
> points(x,sin(x))
```

On peut préciser, parmi les arguments des fonctions graphiques `plot` et `points`, la forme du point (`pch`), sa couleur (`col`), son type (`type`), sa taille (`cex`)...

- La forme du point peut être donnée par un argument explicite : `'+'`, `'-'`, `'&'`... ou par un nombre compris entre 0 et 25.
- Comme autres couleurs, on a notamment `'yellow'`, `'pink'`, `'gray'`, `'orange'`, `'green'`, `'chocolate'`, `'tomato'`,... La commande `colors()` renvoie les noms des différentes couleurs disponibles.
- Il y a huit types de tracé possibles :
  1. `'p'` pour afficher des points (type par défaut)
  2. `'l'` pour afficher des lignes
  3. `'b'` pour afficher des points reliés par des lignes sans superposition
  4. `'o'` pour afficher des points reliés par des lignes avec superposition
  5. `'h'` pour afficher des lignes verticales depuis l'axe des abscisses
  6. `'s'` et `'S'` pour afficher des fonctions en escalier
  7. `'n'` pour afficher les points de façon invisible
- Enfin `cex` définit simplement un coefficient multiplicateur pour la taille des points.

```
> plot(1:3,c(1,3,2))
> plot(1:3,c(1,3,2),col='red')
> plot(1:3,c(1,3,2),col='red',type='b')
> plot(1:3,c(1,3,2),col='red',type='b',pch='*')
> plot(1:3,c(1,3,2),col='red',type='b',pch='*',cex=3)
> points(1:3,3:1,col='blue',type='h')
```

- `plot(x)` : représente une série de points (ordonnée x et numéro d'indice en abscisse).
- `plot(x,y)` : représente un nuage de points d'abscisse x et d'ordonnée y.
- `image(x,y,z)` : représente en niveau de couleur une image où z représente l'intensité au point x,y (z est une matrice dont le nombre de ligne est la longueur de x et le nombre de colonne celle de y).
- `lines, points` : ajoute une ligne ou des points sur un graphique existant.
- `hist` : histogramme.
- `barplot` : graphique en barre.
- `abline` : représente une ligne en précisant la pente b et l'ordonnée à l'origine a. Une ligne verticale d'abscisse x ($v = x$) ou horizontale d'ordonnée y ($v = y$)
- `legend` : ajoute une légende en précisant les symboles (lty ou pch et col), le texte (text) et l'emplacement (x="topright").
- `axis` : ajoute un axe. Argument : side (1 : bas, 2 : gauche, 3 : haut, 4 : droite).
- `grid` : ajoute un quadrillage.
- `par(mfrow=c(n,p))` : partage la fenêtre graphique en $n \times p$ sous graphiques.

4 Simulation d'un vecteur gaussien

4.1 Méthode de Choleski

Pour simuler un vecteur gaussien ε de loi $\mathcal{N}(0, I_n)$ il suffit d'utliser `rnorm(n)`. Supposons à présent que l'on souhaite simuler un vecteur $X = (X_1, \ldots, X_n)$ gaussien centré de matrice de covariance R. Puisque R est une matrice symétrique positive, par la décomposition de Choleski, il existe $A \in \mathcal{M}_n(\mathbb{R})$ triangulaire supérieure telle que $A^t A = R$. On utilise alors $X \overset{d}{=} A^t\varepsilon$. L'algorithme de la décomposition de Choleski est implémenté dans le package (Matrix), la commande est `chol( )`.

Exercice 2. *Simuler $(B_1, \ldots, B_n)$ par la méthode de Choleski pour $(B_t)_{t\in\mathbb{R}}$ un mouvement brownien standard et tracer les valeurs obtenues pour $n = 10$, $n = 100$, $n = 1000$. On pourra indiquer le temps d'execution à l'aide de la commande*

```
ptm<-proc.time()
()
proc.time() -ptm
```

dans le programme ou avec `system.time`. *Comparer la vitesse de simulation avec une simulation utilisant la fonction* `cumsum`.

L'algorithme de Choleski est très coûteux pour des grandes valeurs de n. Il est de l'ordre de $O(n^3)$ (réduit à $O(n^2)$ si Toeplitz).

4.2 Alternative à la méthode de Choleski pour une matrice de covariance Toeplitz

Soit $Y = (Y_0, \ldots, Y_n) \sim \mathcal{N}(0, R)$ avec R matrice Toeplitz caractérisée par sa première ligne

$$R = \begin{pmatrix} r_0 & r_1 & \ldots & r_n \\ & \ddots & & \vdots \\ & & \ddots & \vdots \\ & & & r_0 \end{pmatrix},$$

ie $R_{i,j} = r_{|i-j|} = \mathrm{Cov}(Y_{i-1}, Y_{j-1})$. La fonction `toeplitz` permet de créer une matrice Toeplitz à partir du vecteur de sa première ligne.

```
>r=exp(-(0:n))
>R=toeplitz(r)
```

Dans cet exemple R est la matrice de covariance de $(Y_0, \ldots, Y_n)$ avec $(Y_t)_{t\in\mathbb{R}}$ un processus gaussien centré Ornstein Uhlenbeck de covariance $\mathrm{Cov}(Y_t, Y_s) = e^{-|t-s|}$.
Une alternative à la méthode de Choleski est de plonger la matrice R dans une matrice circulante $S = circ(s)$ dont on connaît les vecteurs propres et les valeurs propres. Rappelons qu'une matrice circulante est également définie à partir du vecteur de sa première ligne que l'on décale pour obtenir les lignes suivantes. On s'intéressera ici au plongement minimal défini par

$$s = (r_0\, r_1\, \ldots\, r_n\, r_{n-1}\, \ldots\, r_1) = (s_0\, s_1\, \ldots\, s_n\, s_{n+1}\, \ldots\, s_{2n-1})$$

ie

$$S = \begin{pmatrix} s_0 & s_{2n-1} & \ldots & s_2 & s_1 \\ s_1 & s_0 & s_{2n-1} & & s_2 \\ \vdots & s_1 & s_0 & \ddots & \vdots \\ s_{2n-2} & & \ddots & \ddots & s_{2n-1} \\ s_{2n-1} & s_{2n-2} & \ldots & s_1 & s_0 \end{pmatrix} = \begin{pmatrix} R & S_1 \\ S_1^t & S_2 \end{pmatrix}$$

Alors $S = \frac{1}{2n} F_{2n}^* \text{diag}(F_{2n} s) F_{2n}$ avec F_{2n} la matrice de la transformée de Fourier discrète :

$$F_{2n} = \left(\omega_{2n}^{(k-1)(l-1)}\right)_{1 \leq k,l \leq 2n} \text{ et } \omega_{2n} = e^{-\frac{2i\pi}{2n}}.$$

De plus S est une matrice de covariance si et seulement si $F_{2n} s \geq 0$. En choisissant $A = \frac{1}{\sqrt{2n}} F_{2n}^* \text{diag}(F_{2n} s)^{1/2} \in \mathcal{M}_{2n}(\mathbb{C})$, $\varepsilon^{(1)}, \varepsilon^{(2)}$ iid $\mathcal{N}(0, I_{2n})$,

$$A[\varepsilon^{(1)} + i\varepsilon^{(2)}] = Z^{(1)} + iZ^{(2)},$$

avec Z_1, Z_2 iid $\mathcal{N}(0, S)$. On obtient ainsi 2 réalisations indépendantes de Y

$$Y \stackrel{d}{=} \left(Z_k^{(1)}\right)_{0 \leq k \leq n} \stackrel{d}{=} \left(Z_k^{(2)}\right)_{0 \leq k \leq n} \sim \mathcal{N}(0, R).$$

On peut utiliser l'algorithme de la transformation de Fourier rapide `fft` pour réduire le coût à $O(n \log(n))$ pour $n = 2^p$.

Exercice 3. *Ecrire une fonction* `CircEmbCov(r)` *qui test si la matrice S construite à partir du vecteur r est bien positive dans les cas suivants*

1. *Covariance exponentielle, $r(t) = e^{-|t|}$,* `r=exp(-(0:n)/n)` *$n = 2^8, 2^9, 2^{10}$.*
2. *Covariance gaussienne, $r(t) = e^{-t^2}$,* `r=exp(-((0:n)/n))^2` *$n = 2^8, 2^9, 2^{10}$.*
3. *Covariance puissance exponentielle, $r(t) = e^{-|t|^{1/2}}$,* `r=exp(-((0:n)/n))^(1/2)` *$n = 2^8, 2^9, 2^{10}$.*

D'un point de vue théorique on dispose du Théorème 2 [1] :

Théorème 1 *Si les entrées du vecteur $(r_0, \ldots, r_n)$ forment une suite convexe, décroissante, positive alors S est une matrice de covariance.*

Exercice 4. *Ecrire une fonction* `CircEmbSim(r)` *pour simuler et tracer un vecteur gaussien gaussien sur $[0, 1]$ avec le pas $1/n$ pour $n = 2^{10}$ et $n = 2^{11}$, de covariance*

1. *Exponentielle, $r(t) = e^{-|t|}$;*
2. *$r(t) = e^{-|t|^{1/2}}$.*

Comparer les temps de calculs avec la méthode de Choleski en utilisant `system.time`.

5 Application au mouvement brownien fractionnaire

Pour $H \in (0, 1)$, le mouvement brownien fractionnaire [4, 5] $B_H = \{B_H(t) ; t \in \mathbb{R}\}$ est un processus guassien centré tel que $B_H(0) = 0$ p.s. et

$$\text{Cov}(B_H(t), B_H(s)) = \frac{1}{2}\left(v_H(t) + v_H(s) - v_H(t-s)\right), \forall t, s \in \mathbb{R},$$

avec $\forall t \in \mathbb{R}$, $v_H(t) = \text{Var}(B_H(t)) = |t|^{2H}$, appelé variogramme.

Proposition 1 *Soit B_H un mouvement brownien fractionnaire de paramètre de Hurst H :*

- *accroissements stationnaires : $\forall s \in \mathbb{R}$, $B_H(s + \cdot) - B_H(s) \stackrel{fdd}{=} B_H(\cdot)$*
- *H autosimilarité : $\forall \lambda > 0$, $B_H(\lambda \cdot) \stackrel{fdd}{=} \lambda^H B_H(\cdot)$*

Soit $H \in (0, 1)$ et $n \in \mathbb{N}^*$, on cherche à simuler le mouvement brownien fractionnaire B_H sur $[0, 1]$ avec le pas $1/n$ ie le vecteur guassien centré

$$(B_H(0), B_H(1/n), \ldots, B_H(k/n), \ldots, B_H(1)).$$

– par autosimilarité,

$$\left(B_H\left(\frac{k}{n}\right)\right)_{0\leq k\leq n} \stackrel{d}{=} n^{-H}\left(B_H\left(k\right)\right)_{0\leq k\leq n}.$$

– puisque $B_H(0)=0$ p.s., $B_H(k)=\sum_{j=0}^{k-1}(B_H(j+1)-B_H(j))$ for $k\geq 1$

Le bruit gaussien fractionnaire est défini par

$$Y_j = B_H(j+1) - B_H(j), \text{ pour tout } j\in\mathbb{Z}.$$

Ainsi, $(Y_j)_{j\in\mathbb{Z}}$ est un processus gaussien centré stationnaire de covariance

$$r_k = \mathrm{Cov}(Y_{k+j}, Y_j) = \frac{1}{2}\left(|k+1|^{2H} - 2|k|^{2H} + |k-1|^{2H}\right), \forall k\in\mathbb{Z}.$$

Exercice 5.

1. *Ecrire une fonction* `CovMBF(H,n)` *qui donne en sortie le vecteur* $r=(r_k)_{0\leq k\leq n}$ *du bruit gaussien fractionnaire* $(Y_k)_{0\leq k\leq n}$.
2. *Tester la validité de la méthode de la matrice circulante pour le bruit gaussien fractionnaire pour* `H=seq(0.1,0.9,0.1)` *et* $n=2^(10)$, *puis* $n=2^1 1$.

D'un point de vue théorique, on dispose du Théorème [6] suivant

Théorème 2 *Pour tout* $H\in(0,1)$ *et* $n\in\mathbb{N}^*$, *la matrice circulante minimale du bruit gaussien fractionnaire est bien une matrice de covariance.*

Exercice 6. *Ecrire une fonction* `MBF(H,n)` *qui permette de simuler et tracer* $(B_H(k/n))_{0\leq k\leq n}$.

Références

[1] C. R. Dietrich and G. N. Newsam. Fast and exact simulation of stationary gaussian processes through circulant embedding of the covariance matrix. *SIAM J. Sci. Comput.*, 18(4) :1088–1107, 1997.

[2] V. Goulet. Introduction à la programmation en r. `http://cran.r-project.org/doc/contrib/Goulet_introduction_programmation_R.pdf`, 2014.

[3] J. Jacques. Introduction à r via rstudio. `http://labomath.univ-lille1.fr/~jacques/`.

[4] A. N. Kolmogorov. The local structure of turbulence in an incompressible viscous fluid for very large reynolds number. *Dokl. Akad. Nauk SSSR*, 30 :301–305, 1941.

[5] B. B. Mandelbrot and J. Van Ness. Fractional Brownian motion, fractionnal noises and applications. *Siam Review*, 10 :422–437, 1968.

[6] E. Perrin, R. Harba, R. Jennane, and I. Iribarren. Fast and Exact Synthesis for 1-D Fractional Brownian Motion and Fractional Gaussian Noises. *IEEE Signal Processing Letters*, 9(11) :382–384, 2002.