COLLET Kilian et CONTAMIN Clément

Projet
Opéré au sein de :
l'Université Claude Bernard Lyon 1

Parcours
Master 1 : Mathématiques Appliquées, Statistique

Spécialité : Processus Stochastique
Discipline : Modélisation

Soutenance prévue le 02/05/2022, par :
Frédérique BIENVENÜE

Les marches aléatoires renforcées
Propriétés, modélisations et comportements

Devant le jury composé de :

Frédérique BIENVENÜE Université Claude-Bernard Lyon 1	**Responsable**
Anne PERRUT Université Claude-Bernard Lyon 1	**Responsable**
Promotion M1MAS 2022 Université Claude-Bernard Lyon 1	**Invités**

Table des matières

1 Introduction

1.1 Présentation

Une marche aléatoire désigne un modèle constitué d'une succession de pas effectués de manière aléatoire. Elle peut modéliser, de manière simple, n'importe quelle succession d'évènements aléatoires mis bouts-à-bouts : que ce soit la trajectoire de l'ivrogne qui oublie à chaque instant d'où il vient et où il veut aller, un mouvement brownien comme celui d'actifs financiers par exemple, ou encore comme choisi dans cette étude, les trajectoires empruntées par une ou plusieurs fourmis.
C'est donc un modèle mathématique représentant une dynamique discrète composée d'une succession de pas aléatoires effectués « au hasard ». On emploie également fréquemment les expressions marche au hasard, promenade aléatoire ou random walk en anglais. Ces pas aléatoires sont de plus totalement décorrélés les uns des autres : cette dernière propriété, fondamentale, est appelée caractère markovien du processus, du nom du mathématicien Markov. Elle signifie intuitivement qu'à chaque instant, le futur du système dépend de son état présent, mais pas de son passé, même le plus proche. Autrement dit, le système « perd la mémoire » à mesure qu'il évolue dans le temps. C'est pour cette raison que la marche aléatoire est parfois aussi appelée « marche de l'ivrogne ».[Wik] Ainsi, les marches aléatoires font normalement partie des ensembles des Chaines de Markov.

Cependant, ici, nous nous focaliserons sur les marches aléatoires dîtes renforcées par arêtes. Le processus est similaire à celui présenté précédemment, à la différence qu'à chaque fois que nous prenons un chemin, nous incrémentons légèrement la probabilité pour qu'à l'avenir nous réempruntions ces arêtes.

La notion de renforcement est prise de certains phénomènes naturels comme par exemple celui d'une colonie de fourmis qui, cherchant une source de nourriture près du nid de manière aléatoire, laissent derrière elles une piste de phéromones attractives. Celle-ci incite les aux autres fourmis à suivre le même chemin sans pour autant les empêcher d'aller explorer ailleurs. Ces autres fourmis laisseront à leur tour des pistes de phéromones.

Grâce à l'aléatoire, la colonie explore divers chemins, s'aventure à diverses distances que nous étudierons, et au fur et à mesure qu'un chemin est parcouru, la quantité de phéromones qu'il contient augmente : on dit que le chemin est renforcé.

Les modèles de marches aléatoires renforcées les plus populaires sont par exemple : les urnes de Pólya, de Dirichlet, la marche renforcée par arête, la marche renforcée par sommets ou l'attachement préférentiel. Dans cette étude nous étudierons des phénomènes de renforcement dûs à un individu puis à plusieurs. Nous nous intéresserons aux marches aléatoires renforcées par arêtes, qui sont des marches aléatoires qui ont tendance à choisir préférentiellement les directions déjà visitées. L'objet principal de cette étude sera la marche renforcée linéairement.

Il saute aux yeux que nous perdons la propriété de Markov et que nous devrons désormais tenir compte des mouvements passés, puisque ceux-ci influent sur les probabilités des mouvements à venir.

En partant de modèles simples tels qu'une marche aléatoire basique en une dimension, nous présenterons successivement des résultats plus complexes en faisant varier paramètres et règles du jeu. Nous proposerons des prévisions théoriques et analyserons aussi chacuns de nos sets de simulations.

Ainsi, dans une première partie, nous définirons le cadre de notre étude. Nous proposerons des définitions et isoleront des propriétés qui nous semblent intéressantes à modéliser par la suite. Nous fixerons notre intérêt sur les marches aléatoires renforcées par arête après avoir brièvement présenté d'autres types de marches aléatoires.
Dans une seconde partie, nous développerons diverses fonctions permettant de modéliser ces marches, implémentant successivement des variations sur les règles telles que le rebond sur les bords, l'arrêt en cas de contact au bord ou bien le retour interdit. Nous étudierons de manière empirique les conséquences de ces règles sur un individu via l'analyse des graphes fournis par nos algorithmes. Nous regarderons aussi les effets du renforcement en comparant nos fonctions avec des marches aléatoires non renforcées.
Dans une troisième partie, nous proposerons des études plus approfondies sur les variations des divers paramètres, des statistiques sur des éléments tels que la distance, la proportion de l'espace visité ou encore les arêtes les plus empruntées. Nous ferons aussi varier la taille de l'espace ainsi que les paramètres du renforcement afin d'étudier les conséquences de leurs variations sur nos statistiques.
Enfin, dans une quatrième et dernière partie, nous étendrons nos fonctions à plusieurs individus d'une même population ainsi que les conséquences de cet élargissement. Finalement nous tenterons de modéliser plusieurs populations ainsi leurs interactions. De plus nous expliquerons les interactions entre ces populations.

Nous avons nous-même développé l'ensemble des algorithmes présents dans ce rapport. Les codes sources (logiciel RStudio) pour chacune de nos représentations graphiques seront disponibles en Annexes à chaque étape de l'étude, et notre fichier $.R$ permettant de retrouver l'ensemble de nos travaux sera aussi mis à disposition dans la Bibliographie. [KC22]

1.2 Définitions

1.2.1 Introduction mathématique

Définition 1.2.1.1 : Une marche aléatoire renforcée est une séquence réalisée à partir d'une variable aléatoire entière $X = \{X_j : j \in \mathbb{Z}^d\}$ où d est le nombre de dimensions souhaitées.
Nous nous restreindrons cependant au cours de notre étude à une ou deux dimensions.

De plus, nous utiliserons la formalisation présentée par Michael PLANK dans le chapitre 3 de sa thèse, *Reinforced Random Walks* aux pages 2 et 3 [Pla03]. À savoir :

$$\Omega \text{ est tel que : } \Omega = \{w_{n,j} : n \in \mathbb{Z}^d \ , \ j \in \mathbb{Z}^{\geq 0}\}$$

De sorte que :

$$\text{(i) } \omega_{n,j+1} - \omega_{n,j} \geq 0 \quad ,$$
$$\text{(ii) } \mathbb{P}(X_{j+1} = n+1 \ \mid \ X_j = n) = \frac{w_{n,j}}{w_{n,j}+w_{n-1,j}}$$

Une autre formalisation similaire peut être trouvée dans la thèse de Élodie Bouchet [Bou14], *Marches aléatoires en environnement aléatoire faiblement elliptique* aux pages 27-28.

Définition 1.2.1.2 : Notons d'abord $(e1, ..., ed)$ la base canonique de $\mathbb{Z}^d$.
Posons $e_j := e_{j-d}$ pour $j \in [\![d+1, 2d]\!]$.
L'ensemble $U := \{e_1, ..., e_{2d}\}$ est alors l'ensemble des vecteurs unité de $\mathbb{Z}^d$.
On note $\|z\|_1 = \sum_{i=1}^{d} |z_i|$ la norme L_1 de $z \in \mathbb{Z}^d$.
On écrit $x \sim y$ si $\|y - x\|_1 = 1$ et on dit alors que x et y sont voisins.

Définition 1.2.1.3 : Une arête est un chemin reliant deux points voisins de l'espace E. Dans notre cas, elles seront orientées c'est-à-dire que les chemins aller et retours d'un point à un autre sont modélisés par deux vecteurs opposés.

Formalisons la notion d'arête orientée.
Nous nous aiderons pour ceci de la formalisation effectuée par Élodie BOUCHET dans sa thèse [Bou14] (page 15) :
L'ensemble des arêtes orientées associées au graphe $\mathbb{Z}^d$ est :

$$F := \{(x, y) \in (\mathbb{Z}^d)^2 \text{ tels que } x \sim y\}$$

Définition 1.2.1.4 : Nous modélisons la probabilité d'emprunter une arête telle que :

$$p_{i,j} = \mathbb{P}(X_{t+1} = j \ \mid \ X_t = i)$$

C'est la probabilité de passer de l'état i à l'état j entre les instants t et $t+1$.

En dimension 1, la famille de nombres $P = (p_{i,j})_{(i,j)\in\mathbb{N}^2}$ est appelée matrice de transition du processus. C'est une matrice stochastique : la somme des termes de n'importe quelle ligne de P donne toujours 1 :

$$\forall i \in E, \quad \sum_{j\in E} p_{i,j} = 1$$

En dimension 2, nous utilisons deux matrices différentes modélisant chacune les mouvements selon une des deux dimensions. Ainsi nous avons deux familles de nombres $P = (p_{i,j})_{(i,j)\in\mathbb{N}^2}$ et $Q = (q_{i,j})_{(i,j)\in\mathbb{N}^2}$ qui permettent de voir les probabilités associées à chacune des quatre directions possibles. Ce qui donne l'équation suivante :

$$\forall i \in E, \quad \sum_{j\in E} p_{i,j} + \sum_{k\in E} q_{i,k} = 1$$

On peut continuer dans cette logique pour les dimensions de taille 3,4 ... Nous utiliserons donc d matrices telles que chacune d'entre elles représente les probabilités de transitions propres à sa dimension associée.

Définition 1.2.1.5 : Une trajectoire est l'ensemble des points visités par l'individu, ordonnés par le temps.

Définition 1.2.1.6 : Le renforcement est le paramètre de l'incrément suite à l'emprunt d'une arête. Il définit la force du renforcement ainsi que sa nature.

Définition 1.2.1.7 : Dans le cas d'une marche aléatoire renforcée par arête, le poids d'une arête $\{w_{i,j}\}$ est initié à 1 et s'associe à l'arête qui va de l'état i à l'état j. Il est renforcé à chaque passage par cette arête. Il permet donc de voir les incréments successifs qui ont été effectués en une arête plus facilement que les probabilités. On peut jongler entre poids et probabilités de la manière suivante :

$$\forall i, j \in E, \qquad p_{i,j} = \frac{w_{i,j}}{\sum_{k\in E} w_{i,k}}$$

Définition 1.2.1.8 : Dans le cas d'une marche aléatoire renforcée par sommet, le poids d'un point $\{w_k\}$ est initié à 1 et s'associe au point k de coordonnées (a,b). Il est renforcé à chaque passage par ce point. Il permet donc de voir les incréments successifs qui ont été effectués en ce point plus facilement que les probabilités. On peut jongler entre poids et probabilités de la manière suivante :

$$\forall i, j \in E, \qquad p_{i,j} = \begin{cases} \frac{w_j}{\sum_{k\in M} w_k} \text{si} j \in M \\ 0 \text{ si } j \notin M \end{cases}$$

Avec M l'ensemble des points mitoyens à k. Par exemple dans le cas d'une grille à deux dimensions et d'un point k qui n'est pas sur un bord,
$M = \{(a-1,b),(a+1,b),(a,b-1),(a,b+1)\}$

Cette relation est aussi explicitée par Franz Merkl & Silke W. W. Rolles dans leur thèse *Linearly edge-reinforced random walks* [MR06] (page 2 : relation 2.3).

1.2.2 Quelques propriétés

Proposition 1.2.2 : Un renforcement linéaire est tel que le renforcement du poids d'une arête (ou d'un sommet) est fonction linéaire du temps local, c'est-à-dire que les renforcements du poids $\{w_k\}$ s'écrivent :

$$w_{k+1} = \alpha + \beta \times w_k \text{ , avec } \alpha, \beta \in \mathbb{R}^+ \text{ constantes fixées}$$

α est le coefficient de renforcement additif et β le coefficient de renforcement multiplicatif.

On note que c'est une écriture plus générale du renforcement étudié par Franz Merkl & Silke W. W. Rolles dans leur thèse *Linearly edge-reinforced random walks* [MR06] (page 2 : relation 2.2). En effet nous avons passé le renforcement 1 en paramètre *alpha* et rajoutons au passage un coefficient multiplicatif *beta*.

Pourquoi un renforcement linéaire ?
Prenons par exemple le cas d'une marche renforcée par arête sur $\mathbb{Z}^d$, soit $\{w_k\}$ le poids d'une arête à l'instant k et

$$w_{k+1} = 1 + w_k^p, \quad p > 0$$

Le Théorème suivant est présent dans la thèse *Marches aléatoires renforcées et opérateurs de Schrödinger* par Xiaolin ZENG [Zen15] (page 12-13)

Theorème 1.2.2 : *La marche renforcée par arête sur $\mathbb{Z}^d$ telle que les poids s'écrivent $w_k = 1 + w_k^p, \quad p > 0$ admet les comportements suivants :*
(1) Si $p < 1$, alors la marche visite p.s. une infinité de sites.
(2) Si $p > 1$, la marche visite p.s. un nombre fini de sites et fini par être coincée en une arête (aléatoire).

Ainsi le cas linéaire $p = 1$ peut être vu comme critique et c'est le point de vue que nous adopterons au cours de cette étude.

Propriété 1.2.2 : Échangeabilité partielle
Dans le cas du renforcement linéaire, on observe la présence d'une symétrie de distribution : l'échangeabilité partielle. En effet, les renforcements ne dépendant que du nombre de passages précédemment effectués, leur ordre n'est pas significatif. L'important étant la trajectoire et le nombre de répétitions de passage.

On peut donc qualifier d'équivalents deux chemins σ et τ si pour toute arête orientée e, les nombres de fois qu'elle a été empruntée sont égaux pour chacun de ces deux chemins.
Par exemple, l'image suivante montre trois chemins équivalents c'est-à-dire que

leur point de départ, d'arrivée et les probabilités obtenues par le renforcement sont les mêmes. Elle a été tirée de la thèse *Marches aléatoires renforcées et opérateurs de Schrödinger* par Xiaolin ZENG [Zen15] (page 13)

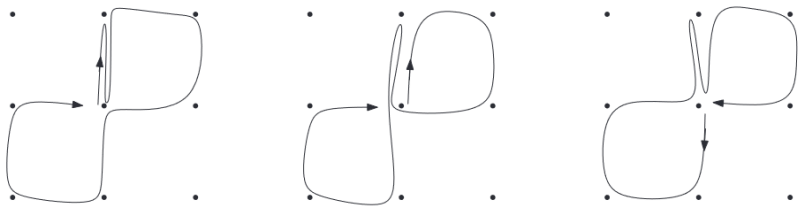


FIGURE 1.2 : Trois chemins équivalents.

1.3 Exemples de marches aléatoires renforcées

1.3.1 Les urnes

L'urne de Pólya : On considère une urne contenant a boules rouges et b boules blanches à l'instant $t = 0$. On tire une boule de l'urne et on la remet avec une autre boule de la même couleur ; on a donc maintenant $a+b+1$ boules. On recommence un certain nombre de fois l'opération qui consiste à tirer une boule de l'urne et à la remettre avec une autre boule de la même couleur. On cherche à déterminer l'évolution de la proportion de boules blanches dans l'urne.

On modélise donc ce processus discret $(X_n)_{n\geq 1}$ à valeurs dans 0,1 où 0 représente le tirage d'une boule rouge et 1 celui d'une boule blanche) avec condition initiale (a,b) ainsi :

$$\begin{cases} \mathbb{P}(X_1 = 0) = \frac{a}{a+b} \\ \mathbb{P}(X_{n+1} = 0) = \frac{R_n}{R_n+B_n} \quad n \geq 1 \end{cases}$$

Avec $R_n = a + \sum_{k=1}^{n} \mathbb{1}_{X_k=0}$ qui compte le nombre de boules rouges dans l'urne à l'instant n et $B_n = a+b+n-R_n$ compte le nombre de boules blanches.

La modélisation de l'évolution du nombre de boules d'une couleur dans cette urne est bien une variable aléatoire discrète qui respecte les propriétés d'une marche aléatoire renforcée. En effet, plus on tire de boules d'une couleur plus la probabilité de tirer à l'avenir une nouvelle boule de cette couleur augmente.

Pour plus d'informations nous vous renvoyons au document *Reinforced random walk* de Gady Kozma [Koz12] (page 7). Ce document possède un graphe similaire à la Figure 1.2 présentée précédemment, ainsi que les propiétés d'échangeabilité partielle et d'autres propriétés sur les urnes de Pólya.

L'urne de Dirichlet : C'est une généralisation de l'expérience précédente mais avec m couleurs au départ.

Voir *Marches aléatoires renforcées et opérateurs de Schrödinger* par Xiaolin ZENG [Zen15] (pages 18-20) pour les Propriétés et Théorèmes associés à cette expérience.

1.3.2 Marches aléatoires renforcées sur graphes finis

Proposition 1.3.2.1 : Considérons un graphe fini localement $G = (V, E)$ avec V l'ensemble des vecteurs reliant les points d'un espace E défini compact et complet. Des poids sont assignés à chaque vecteur V et changent avec le temps. Ces poids peuvent être assignés aux points au lieu des vecteurs en gardant les mêmes propriétés.
Le temps est défini comme une succession discrète de pas équivalents.
À chaque unité de temps, l'individu peut uniquement se déplacer vers un des quatre points voisins de la grille. Il ne peut donc pas rester sur place ni sauter de cases.
Dès lors qu'une arête est empruntée, on lui rajoute un renforcement qui peut être additif, multiplicatif ou les deux.

Proposition 1.3.2.2 : Nous définissons le modèle comme suit :
On identifie une arête e avec le couple orienté (i, j). Soit X_t la localisation du marcheur à l'instant t. Soit $a_e > 0$ et $e \in E$. Pour $t \in \mathbb{N}_0$ on définit le poids d'une arête de manière récursive en fonction du temps :

$$w_0(e) = 1 \quad \forall e \in E$$

$$w_{t+1}(e) = \begin{cases} a + b \times w_t(e) & \text{si } e = \{X_t, X_{t+1}\} \in E \\ w_t(e) & \text{si } e \in E \setminus \{X_t, X_{t+1}\} \end{cases}$$

En écriture avec des probabilités, cela donne :

$$\forall i, j \in E, \qquad p_{i,j} = \frac{w_{i,j}}{\sum_{k \in E} w_{i,k}}$$

Avec $w_{i,j} = 0$ si i et j ne sont pas immédiatement mitoyens (ou non distincts). On retrouve ainsi une probabilité nulle de sauter une case, rester sur place ou aller en diagonale.

1.4 Problématique

Proposition 1.4.1 : Après avoir défini le cadre mathématique, nous pouvons désormais sépcifier notre objet d'étude : les marches aléatoires renforcées sur graphe fini. À cet effet, nous développerons donc un panel de fonctions permettant de les modéliser. Nous avons choisi de ne pas dépasser deux dimensions pour des raisons pratiques et afin de pouvoir facilement effectuer des analyses visuelles.

Ainsi, dans le cas d'une étude de colonie de fourmis, on pourra considérer une vue du dessus où les mouvements sur la grille représentent les quatre points cardinaux d'une carte.

Nous devrons ensuite trouver des méthodes de modélisation graphique pour mettre en application ces fonctions et nous tenterons ensuite de modéliser les trajectoires d'une ou plusieurs fourmis et d'en comprendre les comportements.

Notre problématique : Les marches aléatoires renforcées : propriétés, modélisations et comportements.

2 Modélisations

Dans cette section et à l'avenir, nous abrégerons marche aléatoire renforcée en MAR et nous ne nous intéresserons qu'aux marches renforcées par arête. De plus, nous présenterons nos sorties logiciels RStudio dont les codes seront situés quant à eux en Annexes.

Cette partie a pour objectif de présenter les algorithmes que nous avons créés, leurs paramètres, quelques simulations pour chacun d'entre eux et une analyse préliminaire des phénomènes observés.

Pour une analyse plus appronfondie ou des comparaisons plus poussées, nous préfèrerons la section suivante.

2.1 Marche Aléatoire

Intéressons-nous tout d'abord à une marche aléatoire simple en une dimension.[Wik] C'est celle qui est communément appelée la "marche de l'ivrogne" et c'est donc une chaine de Markov.
Nous prenons ici une grille K de taille 200 et nous partons de son centre. Les bords extrêmes de la grille sont qualifiés de réfléchissants : il est impossible de sortir de la grille. Une fois au bord, les éventuelles directions qui feraient sortir l'individu de la grille sont inexistantes : leur poids est nul. L'individu ne pourra donc pas sortir de la grille, mais il pourra cependant longer les bords. La matrice de transition P1 est définie avec K lignes et 2 colonnes. Elle représente l'ordonnée de chaque point sur une ligne avec en première colonne le poids de la direction bas et en deuxième colonne le poids de la direction haut.

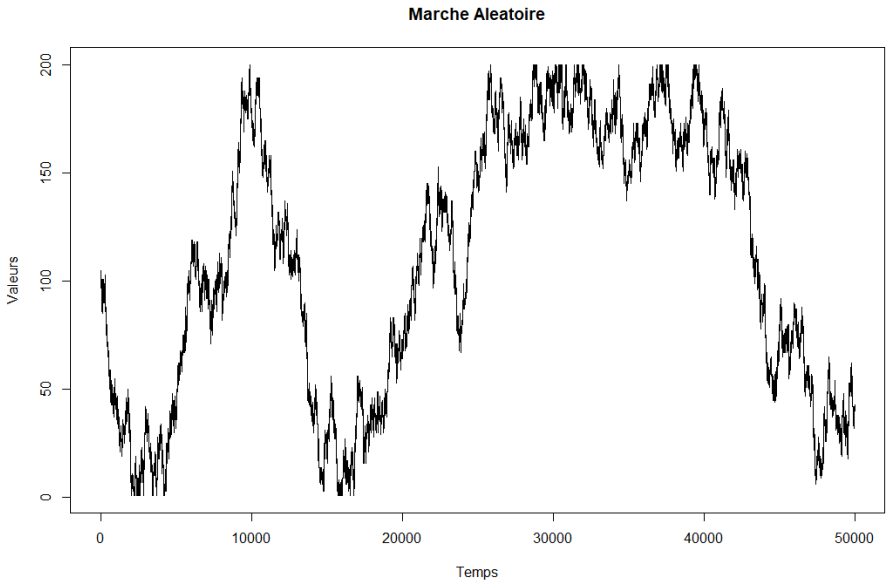
Nous utilisons la fonction MAR (code source en Annexe 6.1) avec les paramètres suivants :

$$\alpha = 0 \quad , \quad \beta = 1$$

$$K = 200 \quad , \quad N = 50000$$

Nous mettons les coefficients de renforcement additif α et multiplicatif β respectivement à 0 et à 1 afin de n'avoir aucun renforcement. Nous posons une grille d'une dimension de taille 200 et nous regarderons 50 000 itérations en partant du centre. (code source de la paramétrisation en Annexe 6.1)

Figure 2.1 : (code source de la figure en Annexe 6.1)



Analyse : Nous observons ici une marche aléatoire simple qui passe par tous les états possibles. Elle est volatile, tape les bords de notre espace plusieurs fois et est donc contrainte à osciller aléatoirement jusqu'à la fin des itérations.

2.2 Marche Aléatoire Renforcée en 1D

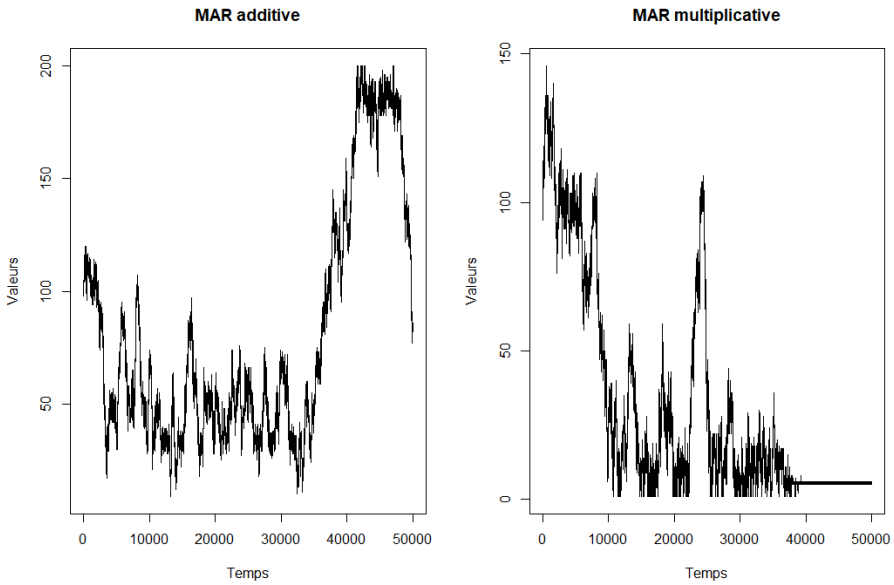
Afin d'obtenir une première MAR, nous utilisons une paramétrisation similaire à la précédente (code en Annexe 6.1).

$$\alpha \in [0, 1] \quad , \quad \beta \in [1, 2]$$

$$K = 200 \quad , \quad N = 50000$$

La grille, le point de départ et le nombre d'itérations restent inchangés. En revanche, nous ferons varier α et β depuis leur valeur la plus faible.

Figure 2.2 : (code source de la figure en Annexe 6.1) Ci-dessous deux graphes présentant à gauche une MAR additive ($\alpha = 0.1$ et $\beta = 1$) et à droite une MAR multiplicative ($\alpha = 0$ et $\beta = 1.005$)



Analyse : Nous observons d'ores et déjà les effets des renforcements sur les trajectoires de l'individu. En effet, sur la figure de gauche on constate que l'individu a passé beaucoup de temps dans la région du niveau 50 avant de finalement s'en extraire et visiter le reste de la grille. Ceci est explicable par un renforcement modéré qui l'a incité à rester dans cette zone sans pour autant l'y enfermer.
En revanche, dans le cas du renforcement multiplicatif, on constate qu'aux alentours de $t = 40.000$ le mouvement s'est bloqué. Ceci est explicable par le fait que le renforcement a atteint des proportions telles que l'individu n'a plus que d'infinitésimales chances de s'échapper de cet aller-retour entre deux états. De plus, le logiciel R possède des limites en termes de grandeur qui assimilent donc un renforcement trop important à un poids infini et donc une probabilité 1 d'emprunter une arête donnée.

Nous notons donc une influence bien plus intense du coefficient multiplicatif sur la convergence de la trajectoire. En effet, fixé seulement à $\beta = 1,005$, nous observons déjà un phénomène d'enfermement de la trajectoire en un couple d'arête opposées (phénomène d'aller-retour). En d'autres termes, la probabilité de réemprunter cette arête rend négligeable la somme des probabilités d'aller sur une arête voisine.
En une dimension, pour obtenir de tels résultats d'enfermement avec un renforcement additif, il faut fixer $\alpha \geq 1$ soit un renforcement de l'ordre du poids initial d'une arête.

2.3 Marche Aléatoire Renforcée en 2D

Afin de modéliser une MAR en deux dimensions, on se placera désormais dans une grille orthonormée de taille $K \times K$.
Nous utiliserons la fonction *MAR2D* font le code source est présenté en Annexe 6.1.
Nous créerons cette fois-ci deux matrices P et Q qui représenteront pour chaque point les poids des mouvements sur une des deux dimensions.

La matrice P sera cella des mouvements horizontaux et sera composée de K lignes et $2 \times K$ colonnes. Elle est organisée comme suit : les K premières colonnes représentent le poids des trajectoire *"aller à gauche"* pour chaque point de la grille, tandis que la deuxième moitié des colonnes représente le poids des trajectoires *"aller à droite"*.

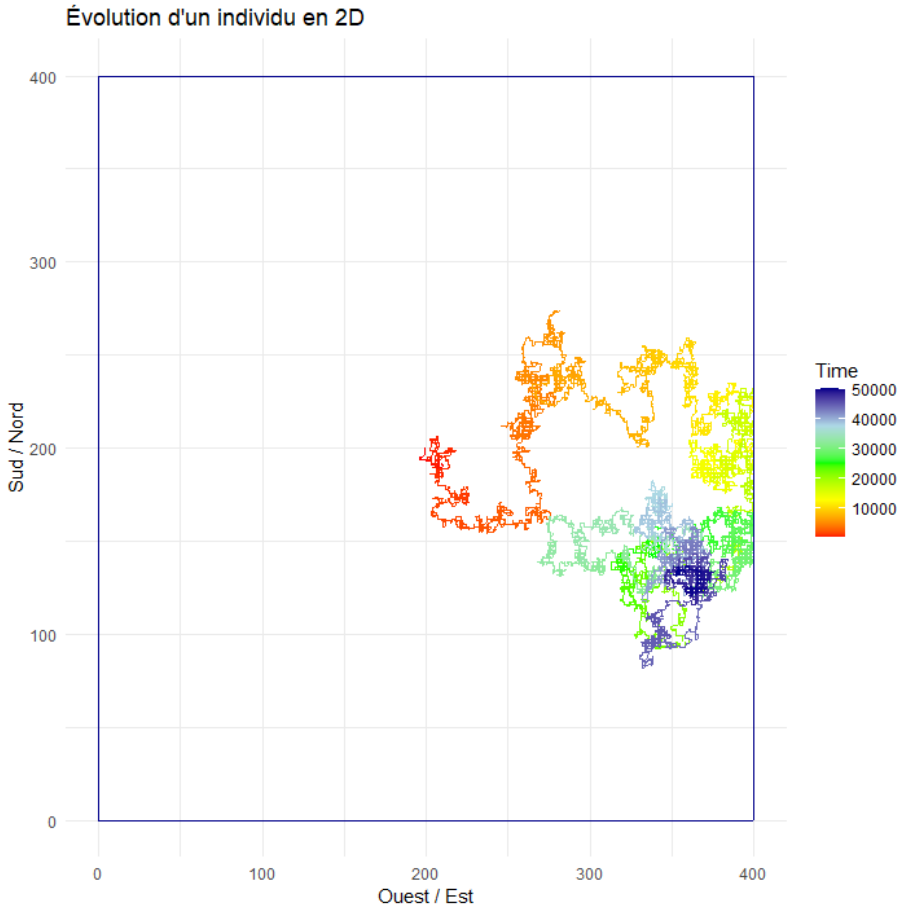
Ainsi, pour un point X de coordonnées (a, b), les poids des trajectoires horizontales se trouvent dans P aux coordonnées (a, b) pour "aller à gauche" et $(a, b + K)$ pour "aller à droite".
Cependant, le point (a, b) dans une matrice est le point *"ligne a colonne b"* tandis que dans un graphique ce sera un point *"abscisse a ordonnée b"* soit *"colonne a ligne b"*.
Nous devrons donc initialiser P de telle que sorte que les K premières colonnes de la première ligne et les K dernières colonnes de la dernière ligne soient à 0. Ceci afin de garantir que l'on ne puisse ni sortir à gauche ni à droite.

De manière analogue, la matrice Q sera celle des mouvements verticaux et aura les mêmes dimensions que P ainsi que le même fonctionnement. Pour un X fixé, elle représentera d'abord le poids de la trajectoire "bas" dans la première moitié des colonnes et "haut" dans la seconde moitié.
Ces deux matrices sont initialisées avec un poids 1 partout sauf sur les arêtes qui permettraient de sortir de la grille, ces dernières étant initialisées à 0.
Nous initialisons donc sa première et sa dernière colonne à 0.

Comme précédemment nous initialiserons une parametrisation telle que (code source de la paramétrisation en Annexe 6.1) :

$$\alpha \in [0, 1] \quad , \quad \beta \in [1, 2]$$
$$K = 400 \quad , \quad N = 50000$$

Figure 2.3.1 : (code source de la figure en Annexe 6.1)



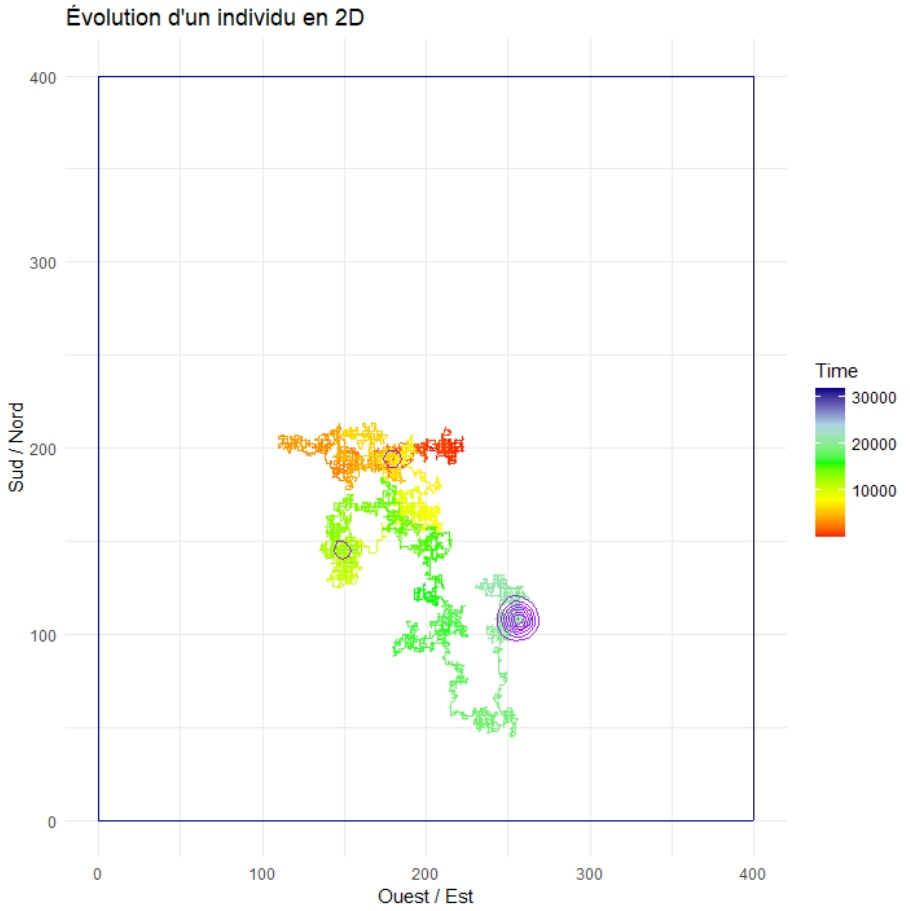
MAR additive en 2D ($\alpha = 1$ et $\beta = 1$)

Quelques explications sur le processus de génération du graphe :
Afin de représenter la dimension temporelle, nous avons utilisé une option de la fonction *ggplot* qui permet d'avoir un code couleur évolutif par rapport au temps. Comme indiqué sur la légende, la trajectoire de l'individu évolue en dégradé en partant du centre en rouge vers le bleu au fur et à mesure que le temps avance.

Les bords, définis comme réfléchissants, ont aussi été rendus visibles.

Analyse :
Nous avons ici volontairement choisis un renforcement relativement intense afin de visualiser son effet.
Nous avons choisi cette valeur après plusieurs essais en faisant varier α. Précedemment, c'était à partir de cet ordre de grandeur que l'on observait le phénomène d'enfermement.
Nous remarquons cette-fois plutôt un phénomène de zone dans la deuxième moitié de la simulation soit lorsque le temps arrive vers le vert foncé.

Figure 2.3.2 : (code source de la figure en Annexe 6.1)



MAR multiplicative en 2D ($\alpha = 0$ et $\beta = 1,15$)

Analyse :
Afin de comparer des MAR additive et multiplicative, nous avons ci-dessus représenté une MAR multiplicative de paramètre $\beta = 1,15$.
Nous remarquons tout d'abord que la simulation n'est pas allée au bout des 50.000 itérations initialement prévues. Ceci est dû au fait que, passé un renforcement considéré comme très grand sur une arête donnée (de l'ordre de 10^{300}), nous avions choisi d'inclure un stop dans la fonction, forçant celle-ci à s'arrêter. Ainsi on constate de nouveau l'aspect exponentiel du renforcement multiplicatif.

Nous avons rajouté sur ce graphique une option de *ggplot* qui nous permet de visualiser les zones où un fort renforcement a été effectué. Ceci nous permet de bien visualiser les éventuels bassins d'agrégation.

L'idée de regarder l'agrégation nous est venue de la thèse de Michael PLANK [Pla03] (page 16) où sont présentées les conditions théoriques de l'apparition d'une zone d'agrégation. Nous ne reviendrons pas sur le Théorème et ses démonstrations car il utilise différents renforcement (cf 3.2 page 3), cependant

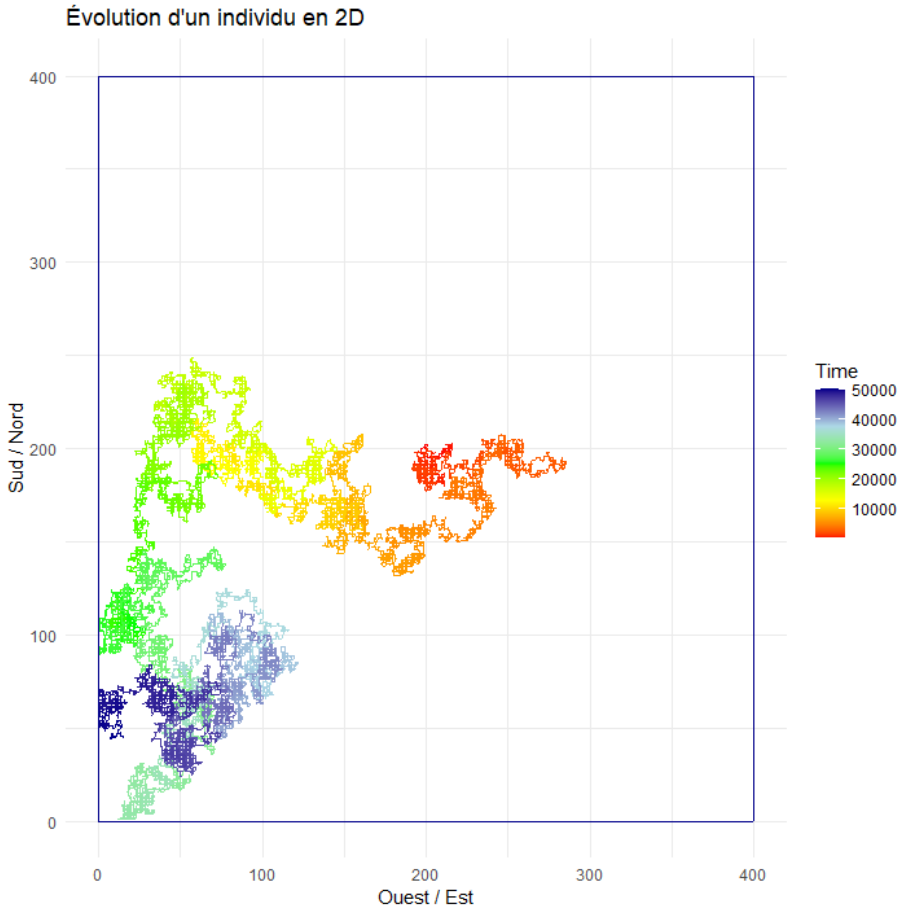
nous gardons l'idée d'existence possible d'une telle zone et nous nous contenterons de l'observer empiriquement ici.

L'individu s'est donc retrouvé enfermé sur une très petite zone de l'espace avec une probabilité d'y rester convergeant vers 1.

De plus, l'individu n'ayant touché aucun bord lors de cette simulation, c'est une marche naturelle dont les mouvements n'ont pas été perturbés par les limites de l'espace. Nous verrons plus tard que la distance par rapport au point de départ est ici très faible comparée à une marche aléatoire standard.

Dans les sections suivantes nous préférerons donc étudier des MAR additives car celles-ci ont un comportement plus raisonnable, et nous pourrons davantage jouer avec les variations de renforcement.

Figure 2.3.3 : (code source similaire au précédent en Annexe 6.1, penser à changer α et β)



MA en 2D ($\alpha = 0$ et $\beta = 1$)

Analyse :
Cette figure a pour seul but de montrer une marche aléatoire simple en 2D,

sans renforcement, afin de pouvoir observer un mouvement libre. On observe des agrégations plus faibles que sur les deux graphes précédents ainsi qu'une exploration plus vaste de l'espace.

2.4 Marche Aléatoire Renforcée en 2D : avec retour interdit

Cette fonction sera appelée *MARri*.
Son code source est situé en Annexe 53.

L'idée de modéliser ce type de comportement nous est venue de la thèse de Line LE GOFF, *Formation spontanée de chemins : des fourmis aux marches aléatoires renforcées* [Le 14], dans laquelle tout un chapitre y est consacré (partie I, Chapitre 2 : page 39 et suite).

Nous initialiserons la paramétrisation suivante dont le code source est en Annexe 6.1 :

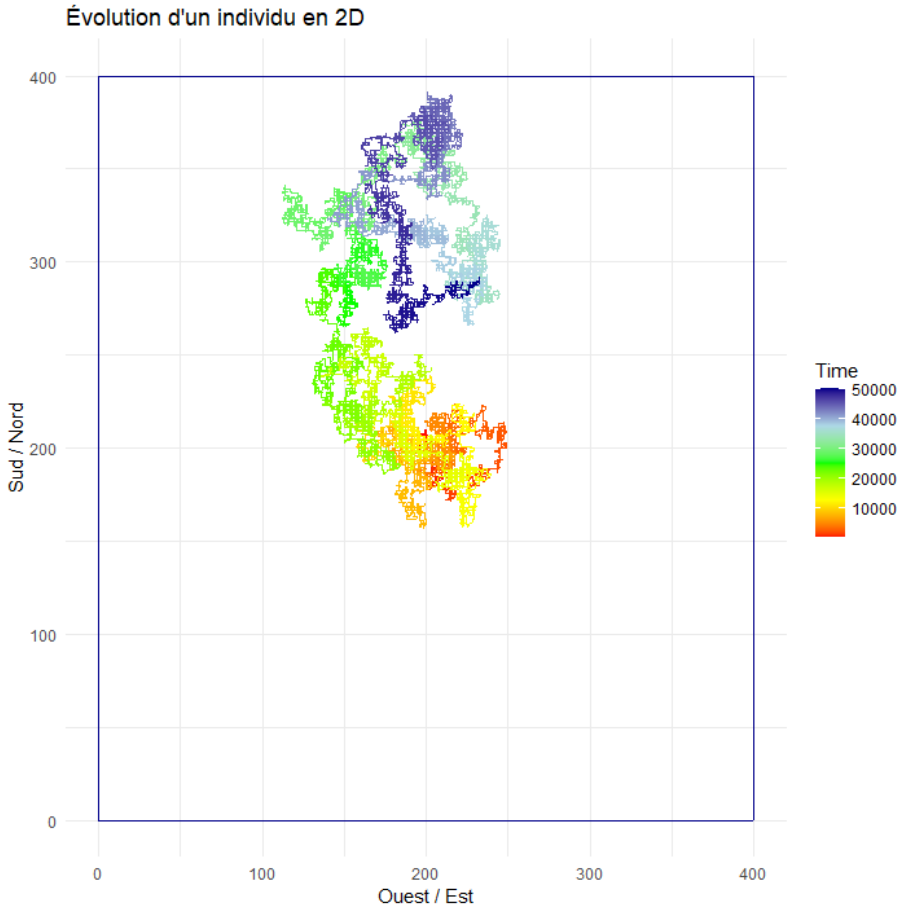
$$\alpha = 1 \quad , \quad \beta = 1$$

$$K = 500 \quad , \quad N = 50000$$

Cette fonction est très similaire à la fonction *MAR2D* à ceci près qu'on interdit ici le phénomène d'aller-retour direct : si le mouvement précédent était par exemple *"aller à droite"*, le mouvement immédiatement après ne peut pas être *"aller à gauche"*.

Cette implémentation devrait permettre d'éviter le phénomène d'aller-retour vu en Figure 2.3.2.

Figure 2.4.1 : (code source en Annexe 6.1)



MARri ($\alpha = 0$ et $\beta = 1,15$)

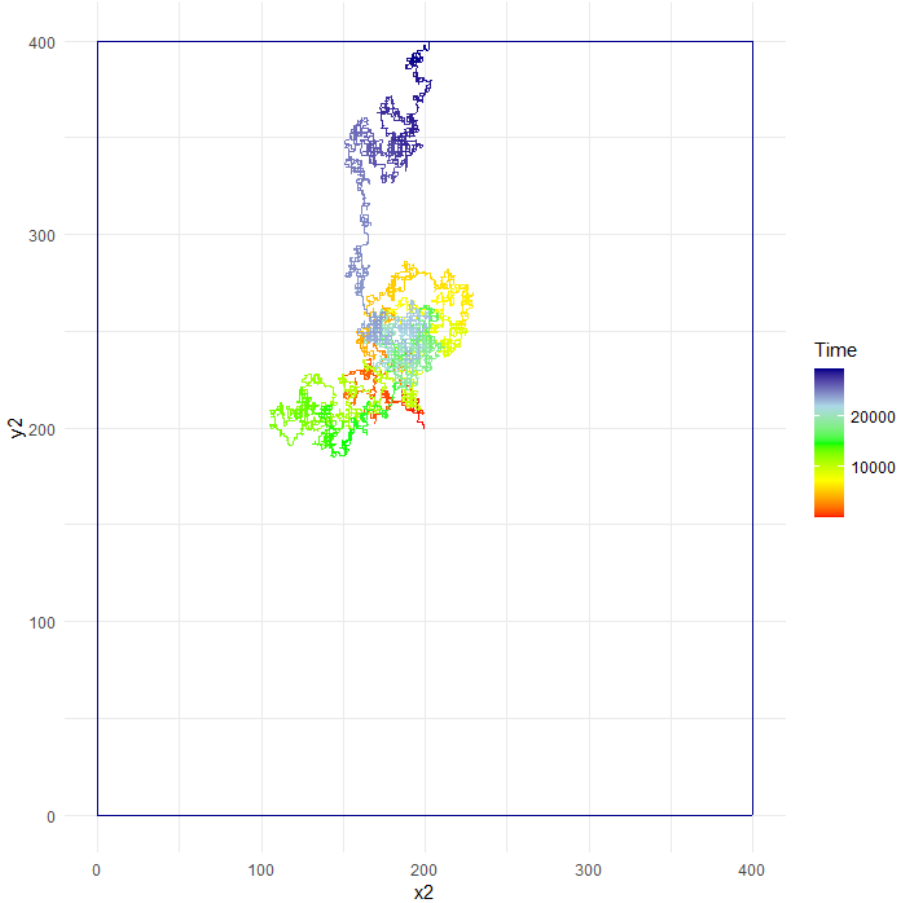
Cependant elle ne suffit pas à garantir que la amrche arrive à son terme. Il a en effet fallu plusieurs essais avant d'arriver à un parcours complet (avec les mêmes paramètres).

2.5 Marche Aléatoire Renforcée en 2D : arrêt quand on touche le bord

Cette fonction sera appelée *MARstop*.
Son code est trouvable en Annexe 6.1.
La paramétrisation reste identique, c'est-à-dire une grille $K \times K$ et $N = 50.000$ itérations.

Cette fonction est très similaire à la fonction *MAR2D* à ceci près que cette-fois, si l'individu touche un bord, le mouvement s'arrête définitivement.
Ainsi, grâce à cette fonction, nous pourrons examiner des temps d'arrêts à partir desquels la trajectoire de l'individu n'est plus naturelle.
Elle sera particulièrement utile dans la section suivante lors de nos analyses statistiques.

Figure 2.5.1 : (code source en Annexe 6.1)



MARstop additive ($\alpha = 1$ et $\beta = 1$)

Analyse :
Ici le temps d'arrêt peut-être obtenu avec la commande *ncol(Y2)* qui renvoie dans ce cas-ci $N = 29.602$ itérations. C'est donc notre temps d'arrêt relevé où le processus s'est arrêté.

Nous verrons dans la prochaine partie les différences que le renforcement produit sur le temps que l'on met à toucher le bord et donc à avoir un temps d'arrêt.

Il est évident que la taille de la grille influe aussi et que plus celle-ci est grande plus le temps avant arrêt le sera.

2.6 Comparaisons avec/sans renforcement

Dans cette section nous comparerons, pour chacune de nos fonctions précédemment développées, la différence entre une version "libre" (sans renforcement) et une version renforcée. Nous nous placerons dans le cas de MAR additives pour les raisons évoquées précédemment et choisirons volontairement un α significatif pour mettre en exergue son effet.

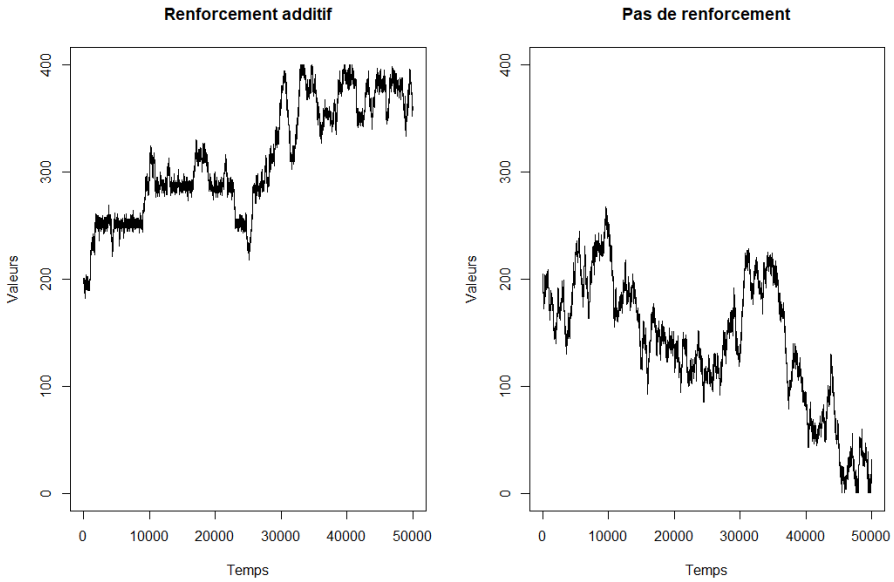
Nous utiliserons la paramétrisation habituelle et nous la garderons pour toute cette section de comparaisons :6.1) :

$$\alpha \geq 1 \quad , \quad \beta = 1$$
$$K = 400 \quad , \quad N = 50000$$

Les matrices P et Q restent définies comme aux sections précédentes.
La matrice P1 ne sera utilisée que dans le cas de la MAR en une dimension.

2.6.1 MAR 1D

Figure 2.6.1 : (code source en Annexe 6.1)



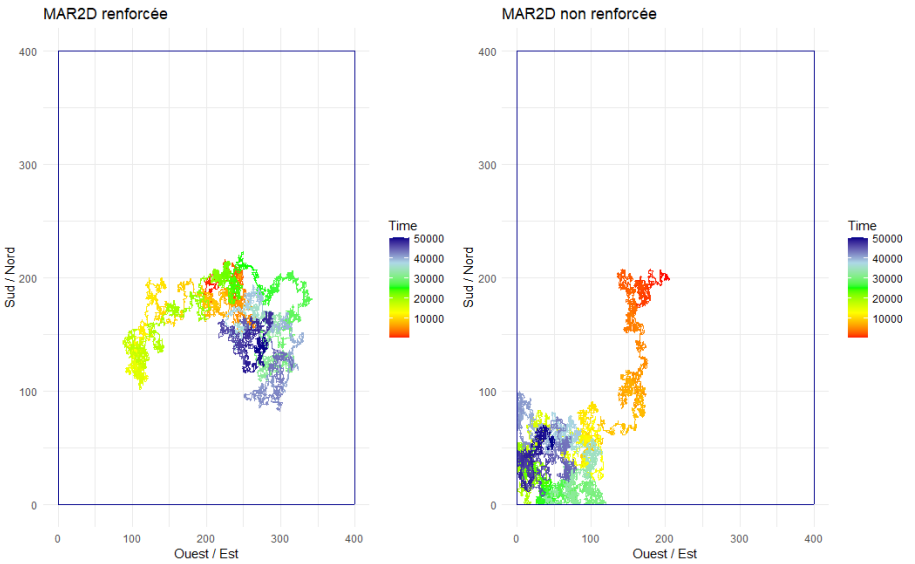
À gauche une MAR additive $\alpha = 0,1$ et à droite une marche aléatoire simple

Analyse : Nous observons une différence très significative entre la MAR renforcée et la marche aléatoire simple. Le renforcement a un effet très impactant en une dimension puisque les oscillations ramènent forcément à une arête déjà renforcée précédemment. Il est ainsi bien plus difficile de s'extraire de la zone d'agrégation.
Si l'on augmente le nombre d'itérations ou le α on obtient rapidement un phénomène d'enfermement sur un aller-retour déjà observé.

2.6.2 MAR2D

Nous présenterons d'abord ci-dessous une comparaison graphique pour chacune des trois fonctions *MARri*, *MAR2D* et *MARstop* puis leur analyse globale.

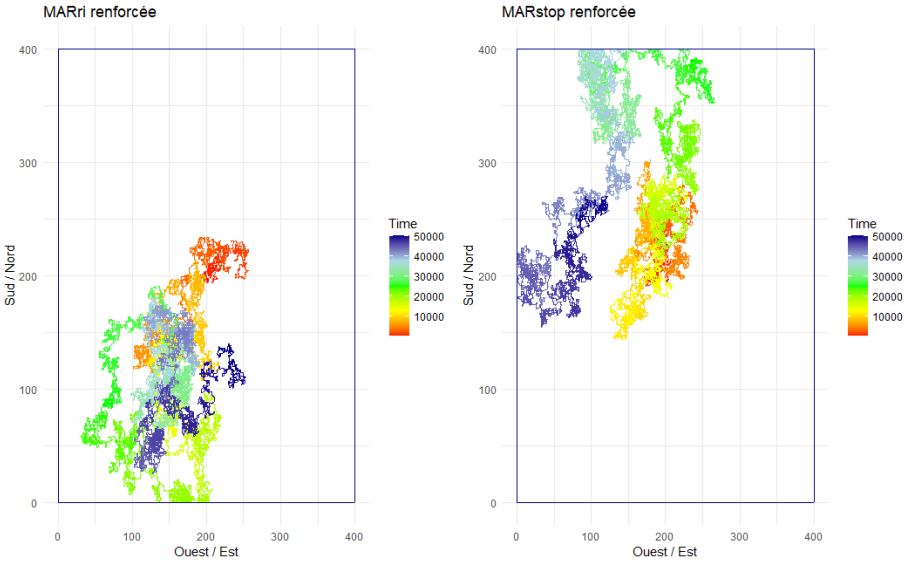
Figure 2.6.2 : (code source en Annexe 6.1)



À gauche une MAR2D additive $\alpha = 0,3$ et à droite une MA2D simple

2.6.3 MARri

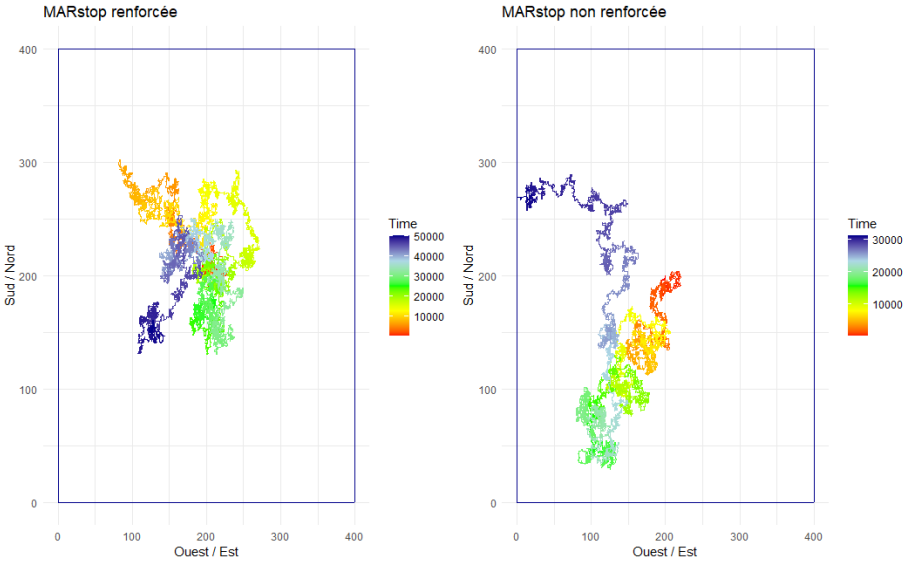
Figure 2.6.3 : (code source en Annexe 6.1)



À gauche une MARri additive $\alpha = 0,3$ et à droite une MA2D simple

2.6.4 MARstop

Figure 2.6.4 : (code source en Annexe 6.1)



À gauche une MARstop additive $\alpha = 0,3$ et à droite une MA2D simple

2.6.5 Analyse

On constate que dans ces trois cas, le fait d'avoir un renforcement réduit la quantité d'espace explorée et tire la trajectoire vers un repli sur soi. Les trajec-

toires sont plus compactes. Dans le premier et le troisième cas, l'individu ne va même plus jusqu'au bord de la grille et ce malgré sa taille relativement petite par rapport au nombre d'itérations.

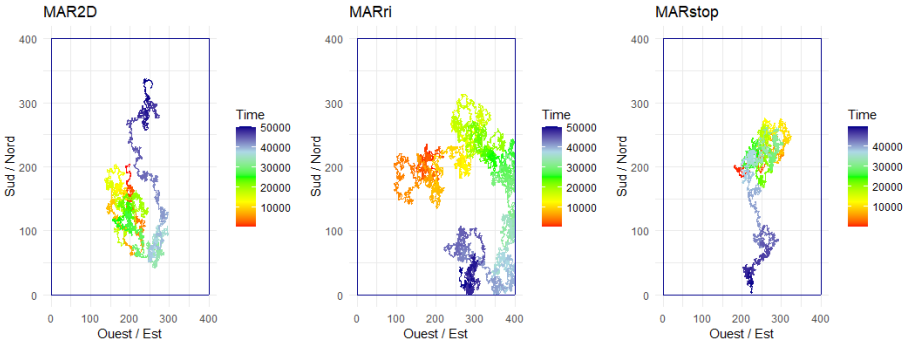
Avant de modéliser ces trajectoires, nous pensions observer un phénomène de "chemin" pouvant éventuellement mener à un circuit en boucle, mais nous observons plutôt ici des zones d'agrégation dûes au renforcement, le poussant à moins explorer et à rester dans une zone prédéfinie par les premières variations.

Le renforcement semble donc influencer négativement la durée pour toucher un bord ainsi que l'espace visité.

2.7 Sensibilités au renforcement

Nous cherchons dans cette partie à observer si une de nos trois fonctions est plus sensible que les autres à des variations du renforcement. Nous observerons visuellement pour cela la proportion d'espace exploré ainsi que le repli sur soi.

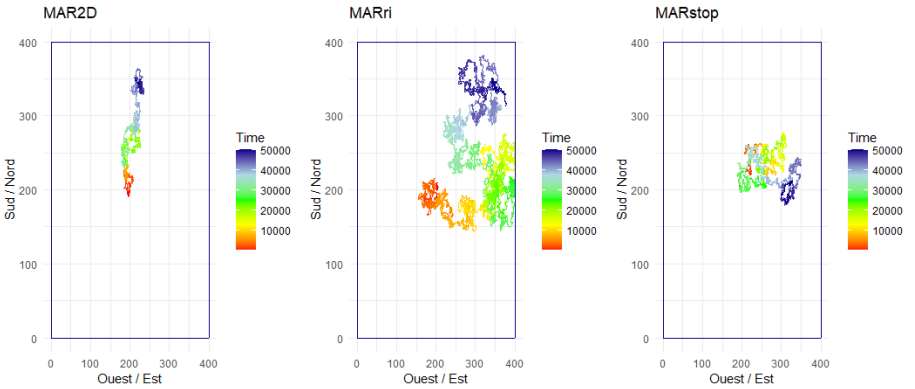
Figure 2.7.1 : (code source en Annexe 6.1)



Trois MAR additives avec $\alpha = 1$ et $\beta = 1$

Analyse : Les fonctions MAR2D et MARstop présentent des trajectoires et réactions similaires, tandis que la fonction MARri semble être la moins impactée par le α qui a été choisi. Elle est en effet allée toucher plsuieurs fois les bords et il semblerait qu'elle ait visité quasiment 1/4 de l'espace alloué. On peut donc émettre une première supposition : la MARri subit moins les renforcements que ses deux voisines.
Ceci peut-être expliqué par construction, puisqu'elle ne peut pas revenir immédiatement sur ses pas, elle est en quelques sortes poussée à explorer et il est plus dur d'observer les phénomènes de renforcements successifs.

Figure 2.7.2 : (code source identique à la figure précédente, changer $\alpha = 3$)



Trois MAR additives avec $\alpha = 3$ et $\beta = 1$

Ci-dessus la même expérience que précédemment mais avec un α fixé encore plus grand. Nous retrouvons les mêmes conclusions.

2.8 Conclusions

Dans cette partie, nous avons développé un panel de fonctions capables de générer des MAR pour un individu. Tout d'abord la fonction *MAR* pour modéliser un individu sur une seule dimension, puis *MAR2D* qui reprend le même individu mais lui permet cette fois de se mouvoir dans un espace à deux dimensions que l'on se sera proposé de définir comme une carte vue du dessus avec ses directions cardinales. Nous avons donc commencé à voir des phénomènes tels que la trajectoires, les bassins d'agrégation, l'impact réfléchissant des bords sur la trajectoire. Nous avons ensuite implémenté de nouvelles règles à notre fonction *MAR2D* afin de créer *MARri* et *MARstop* qui fonctionnent de manière similaire. La première interdit le retour direct sur ses pas, la deuxième s'arrête lorsqu'un bord est touché par l'individu.

En faisant varier les paramètres de renforcement α et β nous avons remarqué que les MAR multiplicatives ont des comportements de renforcement abusif dès lors que $\beta \geq 1,15$ et nous avons donc choisi d'arrêter de les modéliser pour se concentrer plutôt sur les MAR additives.

Munis de nos fonctions, nous les avons d'abord chacune comparé à une marche aléatoire simple, puis nous les avons comparé entre elles. De ces comparaisons semblent se dégager des premières hypothèses : la fonction *MARri* semble explorer plus d'espace et être moins sensible à des renforcements importants. Nous aurons l'opportunité de tester ces hypothèses dans la partie suivante.

3 Statistiques

Dans cette section, nous regarderons de plus près les différentes données observables. Nous ferons une seule paramétrisation pour l'ensemble de la section, dont le code est situé en Annexe 6.2 :

$$K = 400 \quad , \quad K1 = 100$$

$$n = 50 \quad , \quad N = 10.000$$

$$d = 10$$

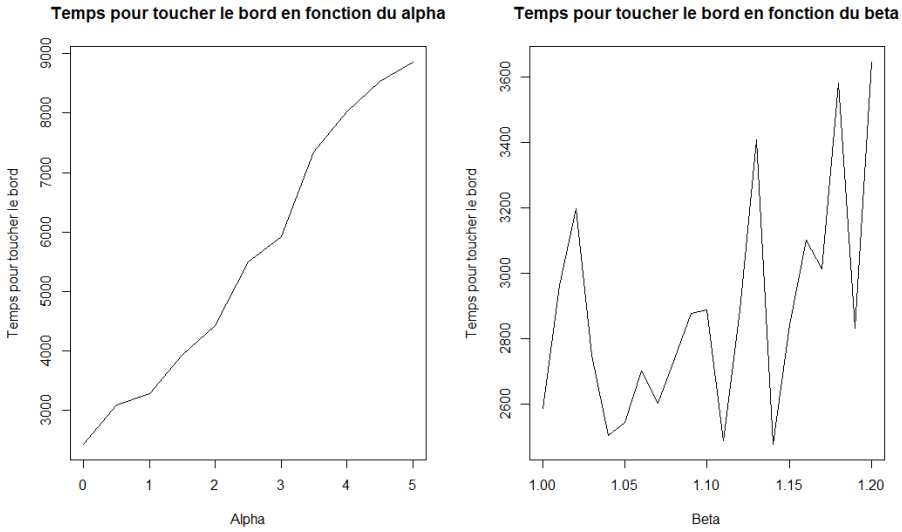
$$Alpha \in [0, 5] \quad , \quad Beta \in [1, 1.2]$$

- K et *K1* sont deux grilles de taille variable.
- Les couples de matrices *(P,Q)*, *(P2,Q2)* sont les matrices de transition associées à chacune de ces grilles.
- De la même manière nous définissons *debut* comme le centre de la première grille K, *debut1* comme le centre de la grille *K1*.
- $n = 50$: nombre de *replicate* des fonctions
- $N = 10.000$: nombre de pas de chaque simulation
- On définit deux couples $\begin{cases} \alpha = 1 \\ \beta = 1 \end{cases}$ et $\begin{cases} \alpha_1 = 0 \\ \beta_1 = 1,01 \end{cases}$
- On définit deux séquences *Alpha* et *Beta* qui nous permettront de tester l'ensemble des renforcements possibles sur l'intervalle donné.
- Nous introduirons aussi la séquence t pour tester différents nombres d'itérations (nous l'utiliserons donc au lieu de N) .
- d sera une limite qui fixera un temps d'arrêt dès qu'elle sera dépassée
- Nous créons deux vecteurs *test* pour plus tard

3.1 Temps pour toucher le bord

Dans cette section nous étudierons le temps que met la fonction *MARstop* pour toucher le bord en fonction du α et du β choisi. Nous utiliserons pour ceci les vecteurs *Alpha* et *Beta* précédemment définis pour obtenir le graphique suivant :

Figure 3.1.1 : (code source en Annexe 6.2)



Comparaison des temps d'atteinte du bord en fonction de α et β

Nous avons ici réduit le nombre d'itérations à 10.000 ainsi que la taille de la grille afin d'avoir des résultats simulés plus rapides.

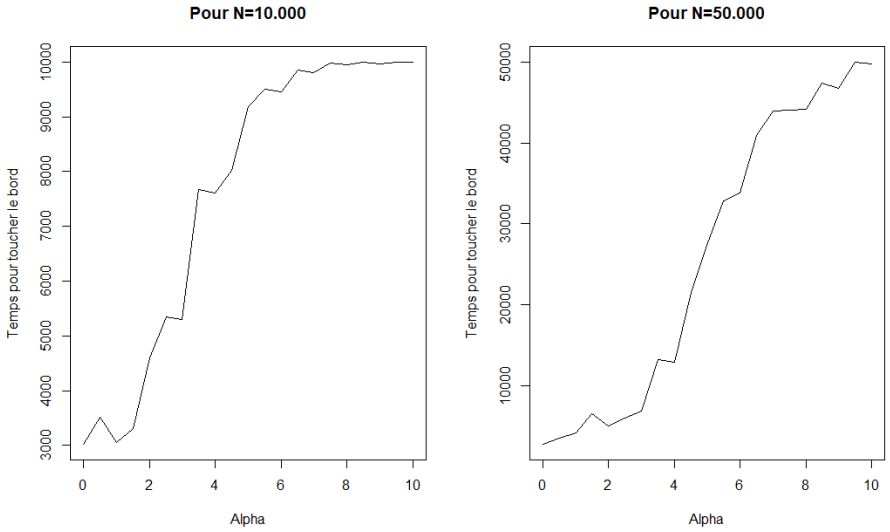
Nous regardons ici le temps moyen d'arrêt d'une répétition de n *MARstop*.

On remarque tout d'abord, à gauche, que le temps mis par l'individu augmente de façon linéaire lorsque α augmente. Celui-ci a donc un effet positif sur le temps d'arrêt.

Dans un second temps, à droite, la figure est plus chaotique mais semble elle aussi avoir une tendance croissante et donc la même influence positive de l'augmentation du β sur temps pour toucher le bord. La forte variance est dûe aux multiples erreurs lorsque le processus s'arrête à cause d'un renforcement trop élevé : nous avions inclu un 'stop' dans la fonction lorsqu'une arête a été renforcée à l'ordre de 10^{300}. Ces temps d'arrêt sont donc notés et viennent perturber nos analyses. Comme précédemment, nous nous contenterons donc des MAR additives dans la suite de nos analyses.

Le plafond pour le temps est à 10.000 car c'est le nombre d'itérations maximales peu importe si le bord est touché ou non.
Il peut être intéressant de chercher la valeur critique de α à partir de laquelle le bord n'est quasiment jamais touché.

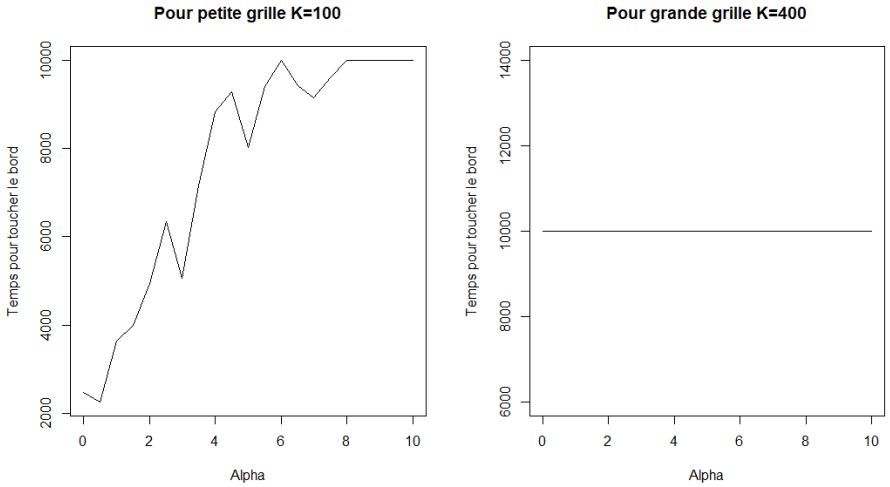
Figure 3.1.2 : (code source en Annexe 6.2)



Effet du nombre d'itérations

Analyse : Avant tout, précisons que ces résultats sont donnés sur une grille de taille 100×100.
Lors de la création de ces graphiques, nous avons étendus les valeurs prises par le vecteur Alpha de telle sorte que : $Alpha \in [0, 10]$
Nous remarquons que ces graphiques présentent des résultats similaires : peu importe le nombre d'itérations de chaque simulation, l'effet de α est positif et semble être à son paroxysme lorsque $\alpha \geq 8$ à gauche et $\alpha \geq 10$ à droite. On peut donc en déduire que, pour une grille de taille donnée K et un nombre d'itérations donné N, il existe presque sûrement une valeur critique pour α telle que la probabilité de toucher le bord en N itérations devient nulle.
De plus il semblerait que cette valeur critique augmente avec le nombre d'itérations.

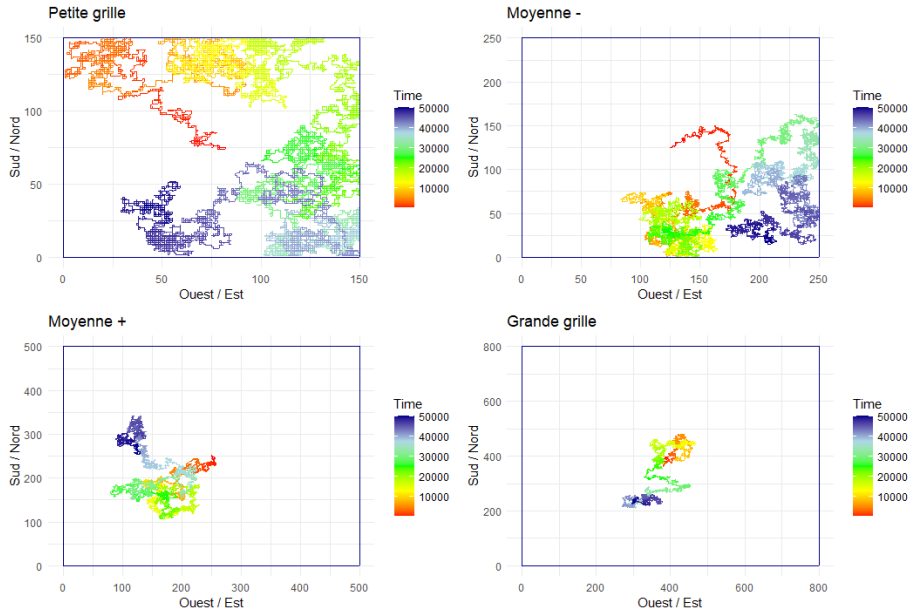
Figure 3.1.3 : (code source en Annexe 6.2)



Effet de la taille de la grille

La figure ci-dessus permet d'observer l'effet de la variation de la taille de la grille sur notre $\alpha_{critique}$.
On voit que la valeur de celui-ci diminue avec la taille de la grille et que lorsque la taille de celle-ci dépasse une certaine valeur, $\alpha_{critique}$ est même nul. Ceci s'explique par le fait que si la grille est suffisamment grande (graphique de droite), même une marche aléatoire simple ne toucherait pas le bord en N itérations et donc peu importe la valeur de notre renforcement, le temps d'arrêt pour toucher le bord sera le temps d'arrêt des simulations.

Figure 3.1.4 : Effet de la taille de la grille sur une MAR2D additive (code source en Annexe 6.2)

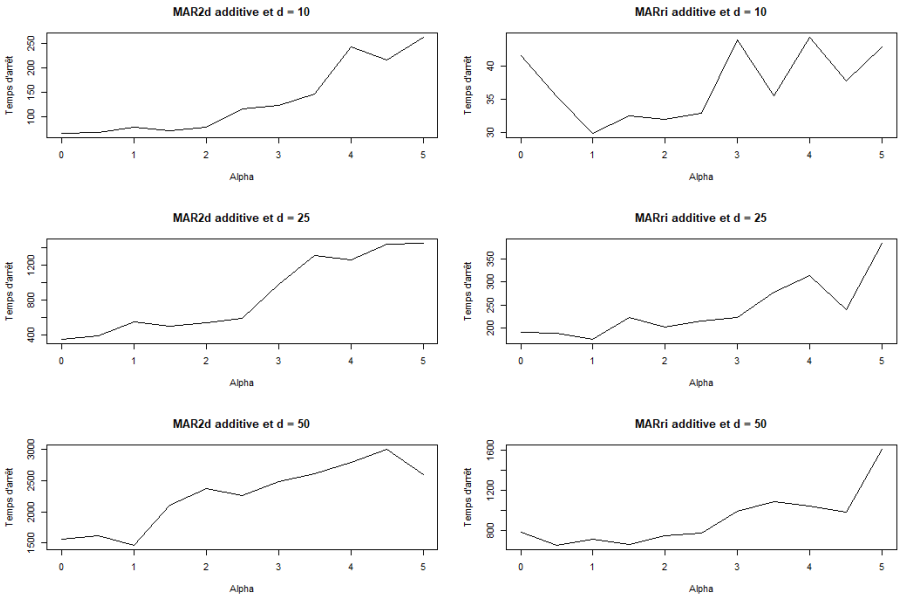


La figure ci-dessus permet de mettre en avant la phénomène illustré précédemment sur quatre grilles de plus en plus grandes. On voit bien qu'au fur et à mesure que la taille de la grille augmente la probabilité de toucher le bord diminue et avec elle, le temps pour toucher le bord en N itérations.

3.2 Temps d'arrêts

Dans cette section, nous nous concentrerons sur le temps que met une *MAR2D* additive test à dépasser une certaine distance au point de départ. Nous comparerons ce temps avec une MARri de mêmes paramètres. Les graphiques ont été obtenus grâce à la commande *replicate* afin d'avoir des résultats plus pertinents.

Figure 3.2.1 : Temps d'arrêts en fonction de alpha pour différents d (code source en Annexe 6.2)



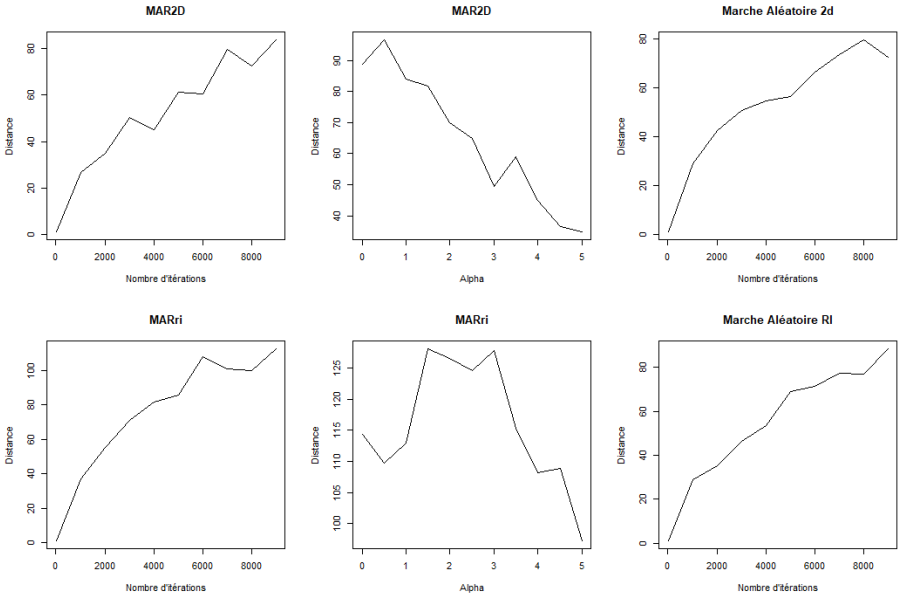
Analyse : Nous confirmons grâce à ces graphique l'effet positif du coefficient α sur le temps d'arrêt : lorsque α augmente, le temps d'arrêt pour dépasser un point d donné augmente aussi.
De même que pour les temps pour toucher les bords, il est raisonnable de supposer qu'il existe une valeur alpha critique pour laquelle la probabilité d'arriver à dépasser le point d est nulle.

Les mêmes assertions peuvent être faites pour la distance au départ du point d : à savoir, effet positif de la distance sur les temps d'arrêts et existence d'une distance d critique pour laquelle la probabilité de dépasser ce point en N itérations est nulle.

De plus, on remarque que la fonction MARri arrive en moyenne plus vite au point d que la fonction MAR2d. Ceci vient confirmer une fois de plus que sa capacité d'exploration est plus importante. Elle est aussi moins sensible à une augmentation de α.

3.3 Distance au point de départ apres N itérations

Figure 3.3.1 : (code source en Annexe 6.2)



Comparaison des distances au point de départ après N itérations
pour une MAR2D et une MARri additives

Analyse : Sur la première ligne du graphique ci-dessus, nous avons mis nos analyses pour une *MAR2D* et sur la deuxième ligne pour une *MARri*.
La première colonne sert à observer la distance au point de départ en fonction du nombre d'itérations N pour un α donné (ici $\alpha = 1$).
La deuxième colonne fixe quand à elle un nombre d'itérations N et fait varier le coefficient de renforcement α.
La troisième colonne est notre colonne de référence puisqu'elle nous montre la distance en fonction du nombre d'itérations sur des marches non renforcées.

Dans la première colonne, nous confirmons encore une fois que la MARri explore davantage que la MAR2D : en effet, en 8.000 itérations la MAR2D est en moyenne à une distance de 80 tandis que la MARri est à une distance de 100.

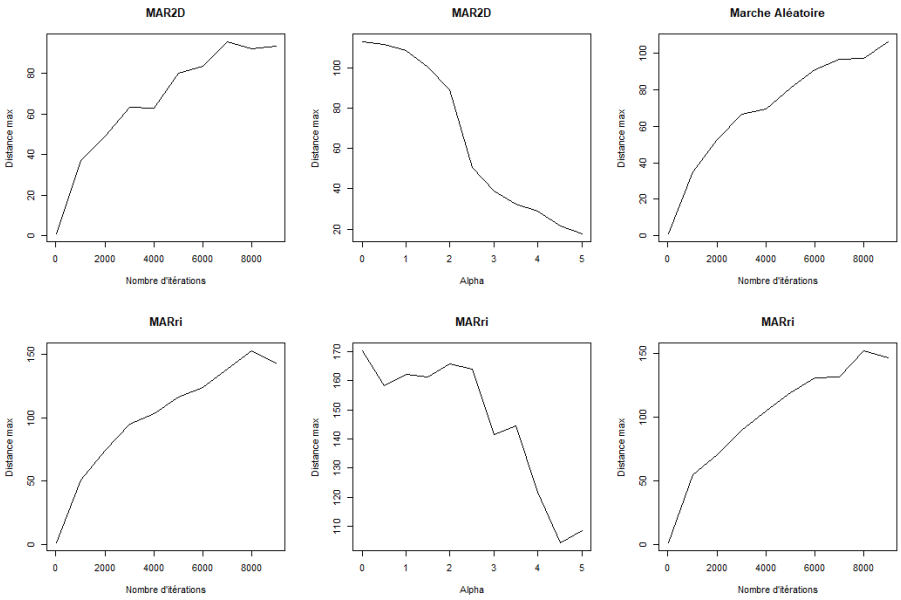
Enfin, grâce à la deuxième colonne, nous pouvons confirmer une influence très forte et négative du coefficient α sur la distance.

3.4 Distance max

Dans cette section, nous nous intéresserons à la distance maximale d'une MAR sur une grille donnée et un nombre d'itérations données.
Comme précédemment, la dernière colonne est une colonne test ayant pour but de comparer nos résultats à une marche aléatoire simple.

Cette idée nous est venue du graphe présenté en page 10 dans la thèse de Franz Merkl & Silke W. W. Rolles *Linearly edge-reinforced random walks* [MR06].

Figure 3.4.1 : (code source en Annexe 6.2)



Distances maximales en fonction du nombre d'itération ou d'Alpha pour MAR2D et MARri

Analyse : On remarque tout d'abord que le nombre d'itérations a un effet positif sur la distance maximale atteinte lors d'une marche aléatoire, qu'elle soit renforcée ou non.
L'augmentation du coefficient α entraine quand à elle une baisse violente de la distance maximale enregistrée.
Nous observons une nouvelle fois la propension d'exploration supérieure de *MARri* comparée à *MAR2D*, avec une distance maximale quasiment deux fois supérieur à celle de *MAR2D* (graphiques de gauche, N=8000 itérations et α arbitrairement fixé à 1).

3.5 Arêtes les plus empruntées

Cette section aura pour but de déterminer les arêtes les plus empruntées à l'issue d'une MAR2D. Nous regarderons leur emplacement global ainsi que l'éventuelle proximité entre elles.

Figure 3.5.1 : Résultats de l'algorithme de recherche des arêtes les plus empruntées sur une MAR2D dans une grille de taille 100x100 (code source en Annexe 6.2)

```
> id1P
     [,1] [,2] [,3] [,4]
[1,]   50   50    2   40
[2,]   50   50    1   35
[3,]   50   52    2   34
[4,]   51   50    1   31
[5,]   50   51    1   27
> id1Q
     [,1] [,2] [,3] [,4]
[1,]   50   50    2   42
[2,]   50   50    1   41
[3,]   49   50    2   34
[4,]   50   49    2   32
[5,]   50   51    2   32
```

Lecture du résultat : Nous détaillerons ici seulement le résultat final de l'algorithme et comment le lire. Pour plus d'informations sur l'entièreté du processus, merci de vous référer au code source commenté en Annexe via le lien ci-dessus.
Dans la première variable *id1P* nous avons stocké les 5 arêtes les plus renforcées (ie les plus visitées) sur le plan horizontal. Les deux premières colonnes nous donnent respectivement l'abscisse et l'ordonnée du point de départ de cette arête orientée. La troisième colonne indique la direction : 1 pour la direction Ouest et 2 pour la direction Est.
De même la deuxième variable *id1Q* a stocké les 5 arêtes les plus visitées sur le plan vertical. La première colonne nous donne l'abscisse et la deuxième l'ordonnée du point de départ de cette arête orientée. La troisième colonne indique la direction : 1 pour la direction Sud et 2 pour la direction Nord.

Nous notons donc que les arêtes les plus parcourues sont toujours situées très proches du point de départ. Cette simulation a été effectuée sur une grille de taille 100×100 et on constate que le point de départ (50,50) a toutes ses arêtes orientées qui figurent au top10 des arêtes les plus visitées. De plus toutes les autres arêtes les plus visitées sont voisines à ce point.

La quatrième colonne sert quand à elle à connaitre le nombre de passages totaux effectués sur l'arête orientée observée. Ce résultat est un compte pour n réalisations donc si nous voulons le nombre de passages moyen par simulation il faut penser à le diviser par n (ici nous avions pris $n = 50$).

Le résultat, assez intuitif, est qu'en moyenne les arêtes les plus empruntée seront celles immédiatement proches du point de départ. Ce sont celles qui ont la plus grande probabilité d'être empruntées à chaque simulation. Les autres,

plus éloignées, sont trop souvent ignorées pour être significatives lorsque l'on regarde une suite de plusieurs simulations.

En augmentant le nombre d'arêtes les plus visitées que nous souhaitons observer, nous obtenons le graphique suivant (code source identique au précédent, changer x à 10 ou à une autre valeur de votre choix) :

Figure 3.5.2 :

```
> id1P
      [,1] [,2] [,3] [,4]
 [1,]   50   50    1   28
 [2,]   50   50    2   26
 [3,]   52   51    2   21
 [4,]   53   51    1   20
 [5,]   53   51    2   18
 [6,]   54   51    1   17
 [7,]   44   49    2   17
 [8,]   49   50    1   16
 [9,]   51   52    1   16
[10,]   50   49    2   16
[11,]   51   50    2   16
[12,]   50   49    1   15
[13,]   51   50    1   15
[14,]   52   52    1   15
[15,]   45   50    1   14
[16,]   48   47    2   14
[17,]   49   50    2   14
[18,]   50   51    2   14
[19,]   52   53    2   14
[20,]   43   50    1   13
[21,]   46   50    1   13
[22,]   47   50    1   13
[23,]   51   52    2   13
[24,]   52   52    2   13
[25,]   49   47    1   12
```

```
> id1Q
      [,1] [,2] [,3] [,4]
 [1,]   50   50    1   33
 [2,]   51   50    2   31
 [3,]   51   51    1   22
 [4,]   52   52    2   22
 [5,]   50   50    2   21
 [6,]   52   53    1   20
 [7,]   50   52    1   18
 [8,]   50   51    2   18
 [9,]   52   51    2   18
[10,]   50   52    2   18
[11,]   53   52    1   17
[12,]   53   51    2   17
[13,]   50   51    1   16
[14,]   51   49    2   15
[15,]   49   47    1   14
[16,]   46   48    1   13
[17,]   50   53    1   13
[18,]   52   54    1   13
[19,]   49   49    2   13
[20,]   50   49    2   13
[21,]   49   51    2   13
[22,]   49   49    1   12
[23,]   49   50    1   12
[24,]   45   51    1   12
[25,]   53   51    1   12
```

Nous observons qu'en ayant choisi de regarder plus loin dans le classement des arêtes les plus empruntées, nous obtenons des arêtes de plus en plus eloignées du centre.

3.6 Sommets les plus visités

Cette section est similaire à la section précédente mais nous regarderons cette fois-ci les points les plus visités à l'issue d'une MAR2D.

Figure 3.6.1 : Sommets les plus visités à l'issue d'une MAR2D sur une grille 400x400 (code source en Annexe 6.2)

```
> Sommet
     [,1] [,2]
[1,]  201  202
[2,]  202  202
[3,]  199  199
[4,]  201  201
[5,]  206  213
```

Nous lisons facilement les coordonnées des points les plus visités. Encore une fois, ceux-ci sont extrêment proches du point de départ de la marche.

3.7 Proportion de points visités

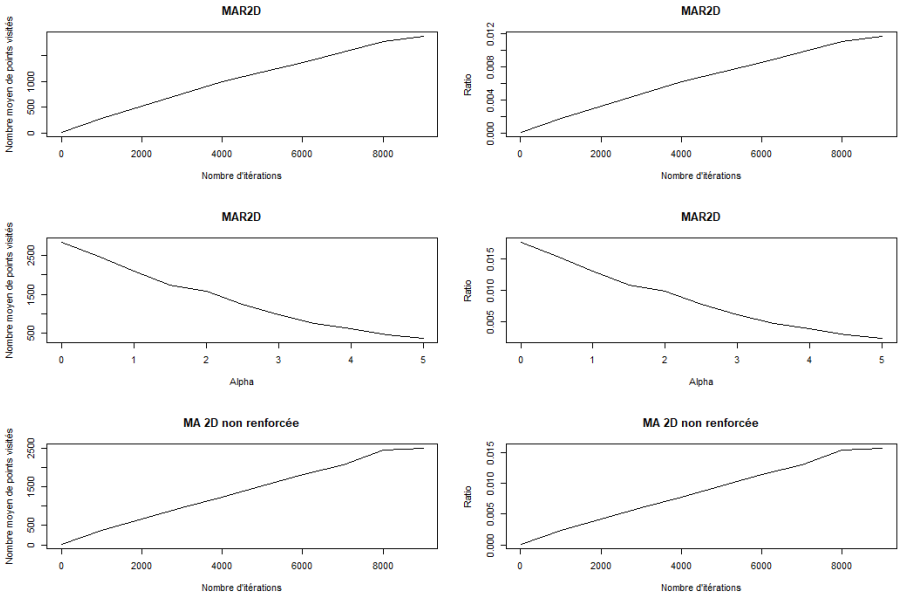
Dans cette dernière section, nous nous focaliserons sur la proportion de points visités par une MAR sur l'ensemble des points possibles de la grille.

D'après le **Théorème 5** du document *Nombre d'états visités par une marche aléatoire* [Mél22], le nombre de points visités par une marche aléatoire avant l'instant n est noté C_n et est tel que :

$$C_n \underset{n\to+\infty}{\longrightarrow} \pi \times \frac{n}{\ln n}$$

Ce résultat coïncide avec le graphique en bas à gauche ci-dessous (la formule prédit 2.800 points visités en 8.000 itérations et nous observons sur le graphique ci-dessous 2.500).

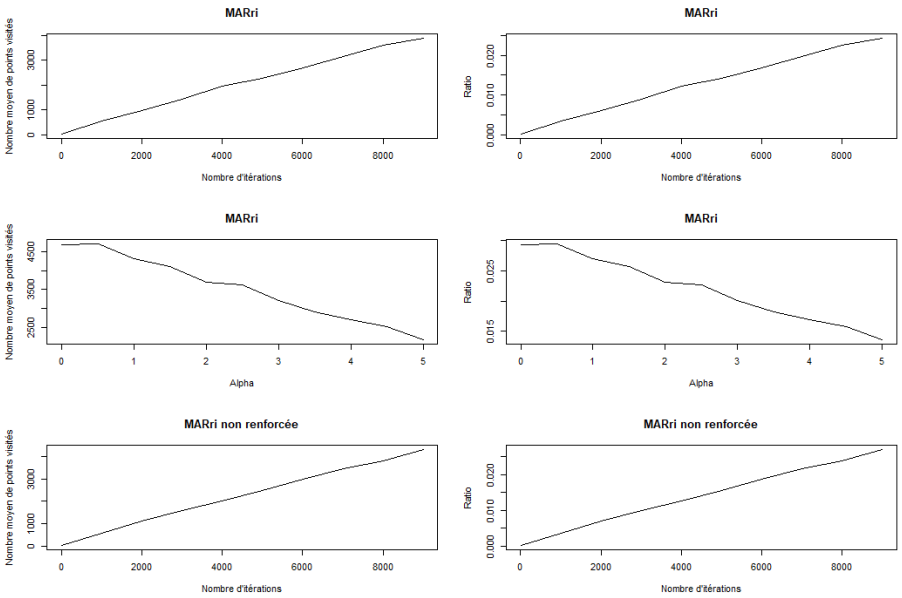
Figure 3.7.1 : (code source en Annexe 6.2)



Représentation graphique du nombre de points de la grille parcourus pour la fonction MAR2D et une grille de 400x400

Analyse : Comme précédemment, nous utiliserons la dernière ligne (celle d'une marche aléatoire simple) comme référence pour nos observations.
Nous présentons à chaque fois sur la colonne de gauche un compte précis du nombre de points visités au moins une fois et sur la droite nous ramenons ce nombre à un ratio (ou pourcentage). Nous voyons donc que le nombre d'itérations a un effet positif et qu'au bout de 10.000 itérations, pour une MAR2D additive avec $\alpha = 1$ nous avons visité environ 1,2% des points de l'espace.

Figure 3.7.2 : Représentation graphique du nombre de points de la grille parcourus pour la fonction MARri et une grille de 400x400 (code source en Annexe 6.2)



Analyse : Cette figure s'analyse comme la précédente et présente les mêmes résultats mais pour la fonction MARri.
Nous observons une fois de plus que la MARri explore plus que la MAR2D avec environ 2,5% des points de la grille explorés en 10.000 itérations (contre 1,2% précédemment).

3.8 Conclusions

Dans cette partie, nous avons examiné les comportements de nos fonctions sous une multitude d'aspects. Nous avons représenté des estimations telles que les différentes distances atteintes (fixées arbitrairement, bords), les temps que nos fonctions mettent à atteindre ses distances ou bien le nombre de passage en un point ou une arête donnée. Ceci nous permet de conjecturer sur les effets de variables telles que le nombre d'itérations ou l'intensité du coefficient de renforcement α.
Nous en sommes donc arrivés aux conclusions suivantes :
- Plus le coefficient α est élevé, plus l'exploration est réduite. Les temps pour toucher les bords (ou d'autres points de la grille) augmentent, les distances maximales atteintes diminuent et les sommets/arêtes les plus empruntés se concentrent près du centre.
- Pour le nombre d'itérations N c'est l'inverse : il favorise l'exploration et les distances maximales.
- La taille de la grille influence de manière intuitive : plus elle est grande, plus il est long et improbable d'arriver à toucher le bord, plus la proportion de points visités sera petite en N itérations.

4 Individus et Populations

Dans cette dernière section, nous modéliserons plusieurs individus successifs qui récupèreront tour-à-tour les matrices de transitions laissées par l'individu précédent. Évidemment un individu continue de renforcer sa propre marche comme modélisé jusqu'ici. Nous considérerons dans un premier temps une seule population (ou colonie), c'est-à-dire que les individus laissent un renforcement positif derrière eux.
Dans un second temps, nous introduirons diverses populations que nous pourrons faire partir du même endroit, ou avec chacune leurs points de départs respectifs. Ces populations auront un renforcement négatif les unes sur les autres : lorsqu'un individu passe par une arête, il la renforce pour lui-même et ses congénères mais il la déprécie pour les autres colonies. Nous nommons *delta* le coefficient d'affaiblissement multiplicatif et *gamma* l'additif. Nous avons choisi pour la suite $delta = 0,8$ et $gamma = 0$.

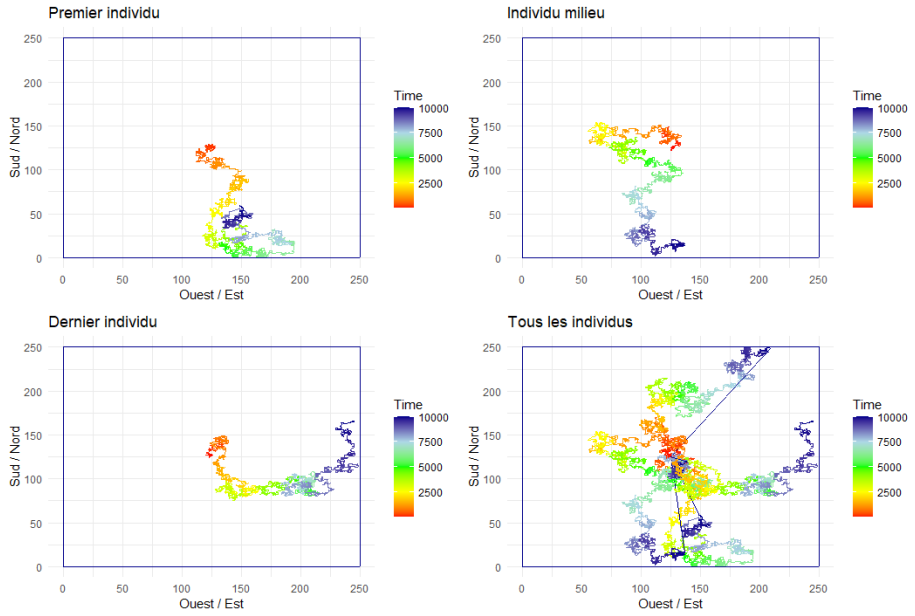
Nous introduirons de nouvelles fonctions (codes sources respectifs en Annexes) :
- **MAR2pop :** deux populations, un individu qui marche par population, chaque individu marche en simultané (l'ordre de passage est tiré aléatoirement à chaque unité de temps). Il n'y aura pas d'autres individus ensuite.
- **MAR3pop :** idem mais avec trois populations
- **MARpop :** fonctionne comme la fonction MAR2D mais renvoie aussi les matrices R et S affaiblies pour les éventuelles autres populations. Ces matrices auront surtout un but pratique pour nos modélisations de plusieurs populations (sections 4.2 et 4.3).

Nous utiliserons la paramétrisation présentée en Annexe 5.3.

4.1 Une seule population

Pour commencer, nous effectuerons seulement la simulation d'une seule population. Nous avons arbitrairement fixé le nombre d'individus à 5 mais il est paramétrable d'en rajouter autant que voulu à travers la variable *individu*.
Nous modélisons donc simplement une succession de MAR2D qui, pour chaque individu successif reprend la matrice laissée par le précédent comme matrice de transition initiale. Il la modifie à son tour au fur et à mesure qu'il avance et la laissera au suivant.

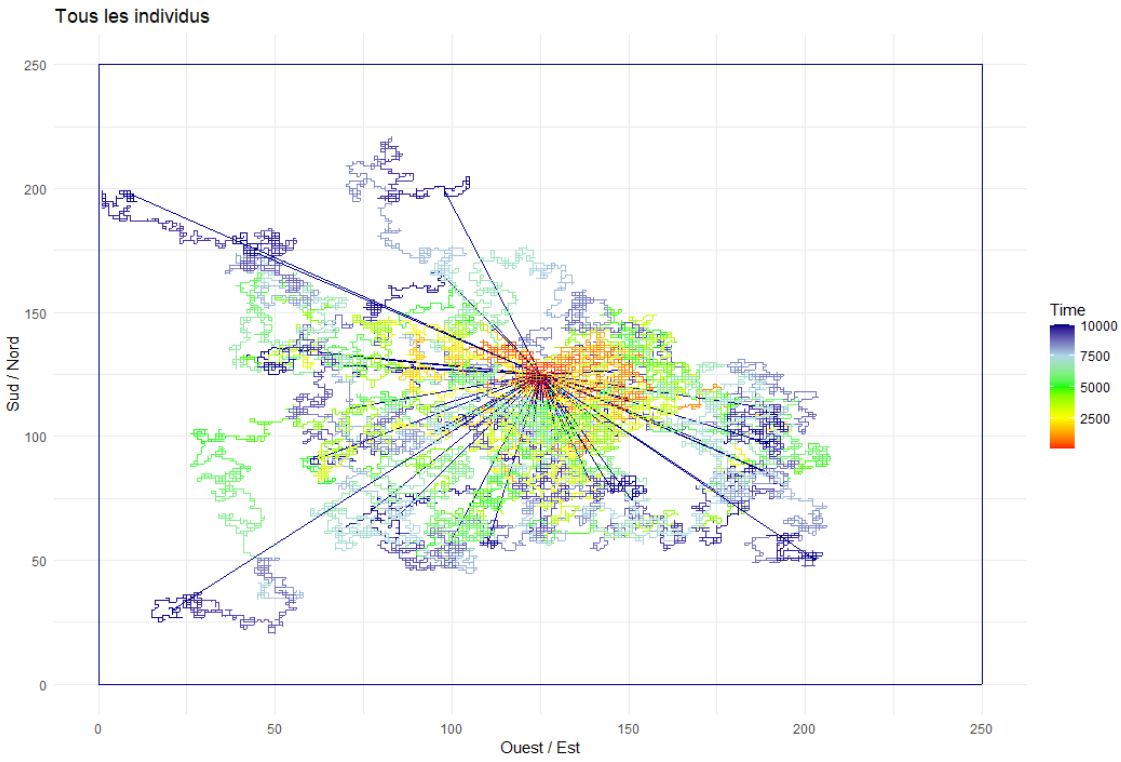
Figure 4.1.1 : Trois graphiques intermédiaires et le graphique global (code source en Annexe 6.3)



Nous avons choisi de montrer ici dans un premier temps le premier individu de notre population, puis un individu intermédiaire et enfin le dernier individu de la simulation. Ce choix s'est fait afin de voir une éventuelle tendance. Ce n'est pas le cas ici, mais en augmentant le coefficient de renforcement ou le nombre d'individus, cela peut s'observer (cf graphique ci-après).
Nous avons aussi inclus en quatrième graphe une vision globale pour les 5 individus sur un même graphe.

Les segments bleus sur le quatrième graphique indiquent les points de fin pour chaque individu. Par construction, vu que nous forçons la marche à reprendre depuis le point central, l'algorithme considère que nous nous sommes "téléportés" et indique donc ceci d'un trait reliant le point final avec le point de départ. Nous avons choisi de laisser cette propriété (intialement non voulue) afin d'illuster de manière visuelle l'ensemble des points de fin pour chaque individu, comme sur le graphique ci-après :

Figure 4.1.2 : Même processus mais avec 50 individus et un renforcement $\alpha = 3$ (code source en Annexe 6.3)



On peut donc, grâce à ces segments, observer un faisceau de trajectoires représentant les zones où l'ensemble d'individus à préférer finir sa course.
Par exemple, on remarque ici que le coin Nord-Est a été délaissé au profit des autres directions qui sont quand à elles assez homogènes.

4.2 Deux populations

Dans cette sous-section et celles à venir, nous allons implémenter d'autres populations. Nous limiterons cependant le nombre d'individus par population afin de garder des temps d'exécution corrects.
Nous proposerons aussi plusieurs implémentations possible en jouant sur l'ordre des mouvements : dans un premier temps, nous regarderons une simulation que nous appellerons *"pas-à-pas"*. Dans ce processus, chaque individu de chaque population doit faire un pas avant qu'un autre puisse en faire un deuxième. Nous avons implémenté le processus de telle sorte que l'ordre de passage à chaque itération est tiré de manière aléatoire.
Par exemple l'individu 1 de la population B est tiré et fait un pas, puis c'est au tour de l'individu 1 de la population A de faire un pas. Vient ensuite le deuxième

pas : cette fois-ci, ce serait d'abord l'individu 1A qui marche en premier, puis le deuxième pas pour l'individu 1B, etc.

Nous modéliserons ensuite une version *"Individu par individu"*, c'est-à-dire que nous ferons d'abord avancer l'individu 1A jusqu'au bout, puis l'individu 1B, avant de revenir à l'individu 2A puis 2B, etc

Enfin, dans un troisième temps, nous modéliserons une version *"Population par population"*, avec d'abord les mouvements de l'ensemble des individus de la population A (individu par individu comme vu précédemment), et ensuite seulement l'ensemble des individus de la population B.

4.2.1 Pas-à-pas

Figure 4.2.1.1 : À gauche la *Population A*, à droite la *Population B* (code source en Annexe 6.3)

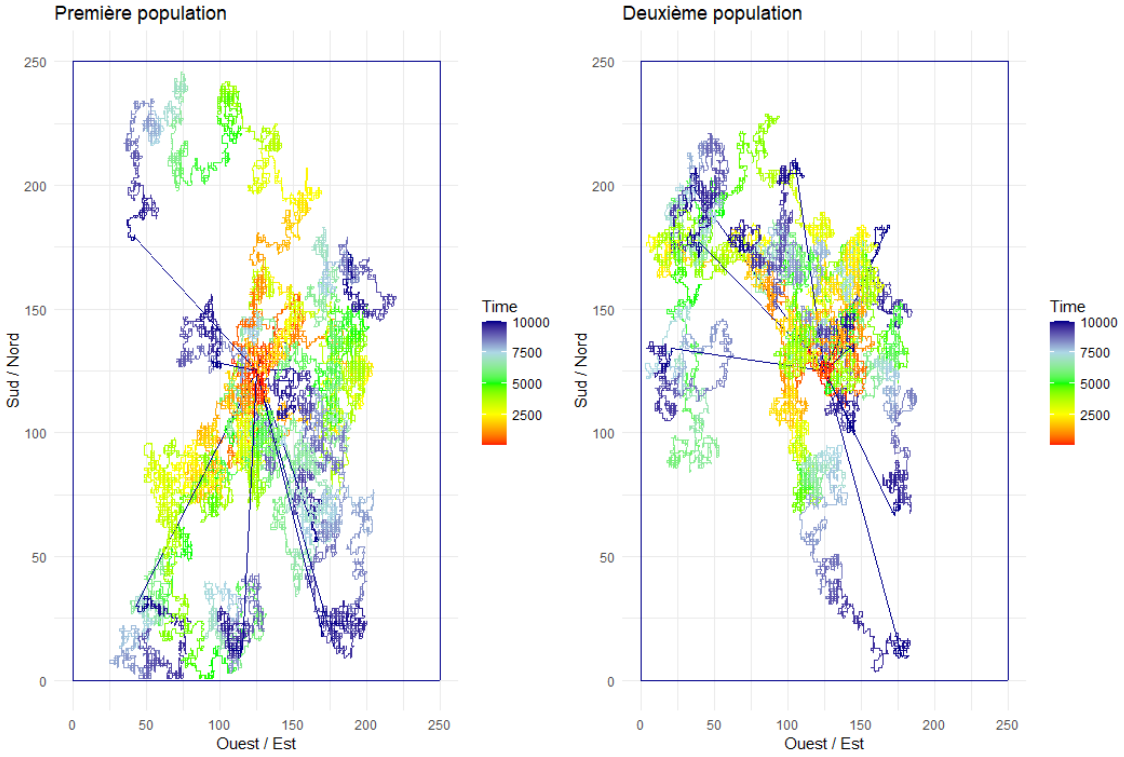
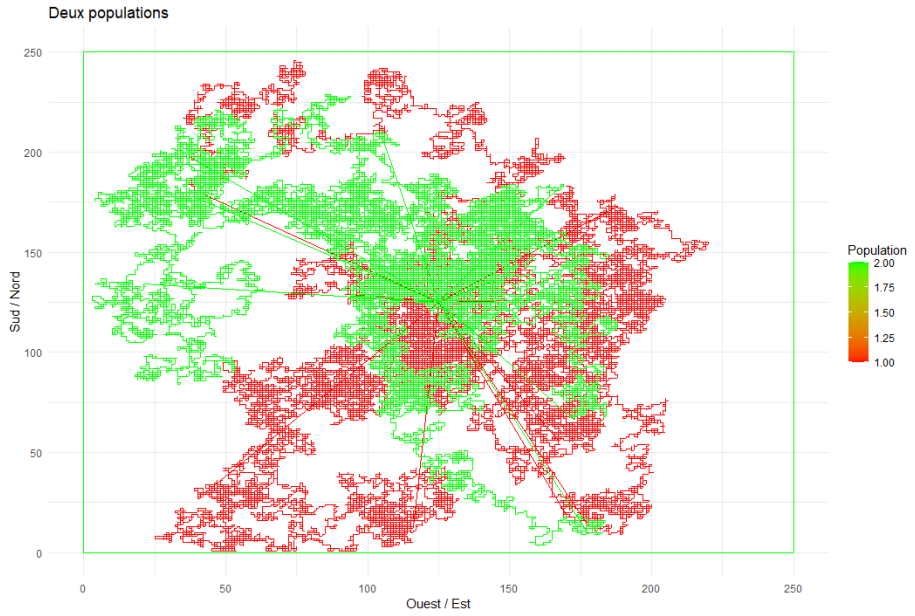


Figure 4.2.1.2 : Les deux populations sur le même graphique (code source compris dans le code de la figure précédente)



Nous remarquons que les individus de la *Population A* (à gauche sur le premier graphique, puis en rouge sur le deuxième) sont principalement allés renforcer la région Sud/Est, tandis que les individus de la *Population B* (à droite sur le premier graphique puis en vert sur le deuxième) ont préféré la zone Nord/Ouest. On observe donc bien le phénomène d'évitement que nous pouvions intuitivement prédire.

4.2.2 Individu par individu

Figure 4.2.2.1 : À gauche la *Population A*, à droite la *Population B* (code source en Annexe 6.3)

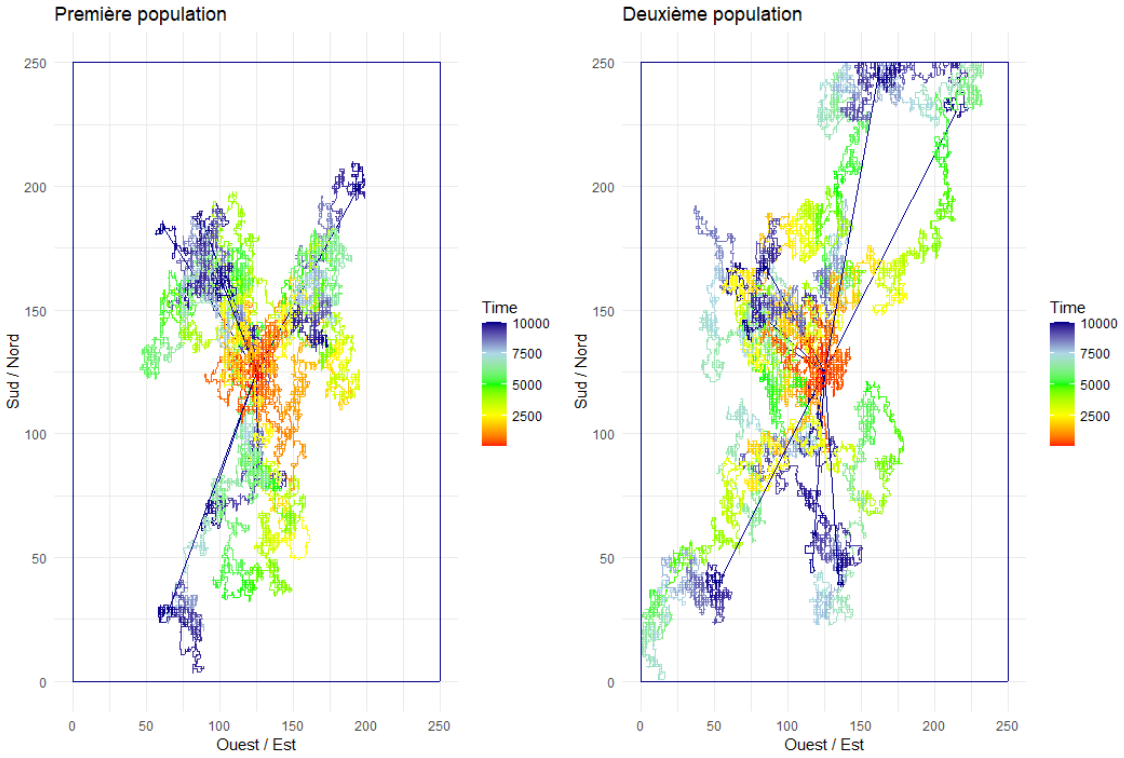
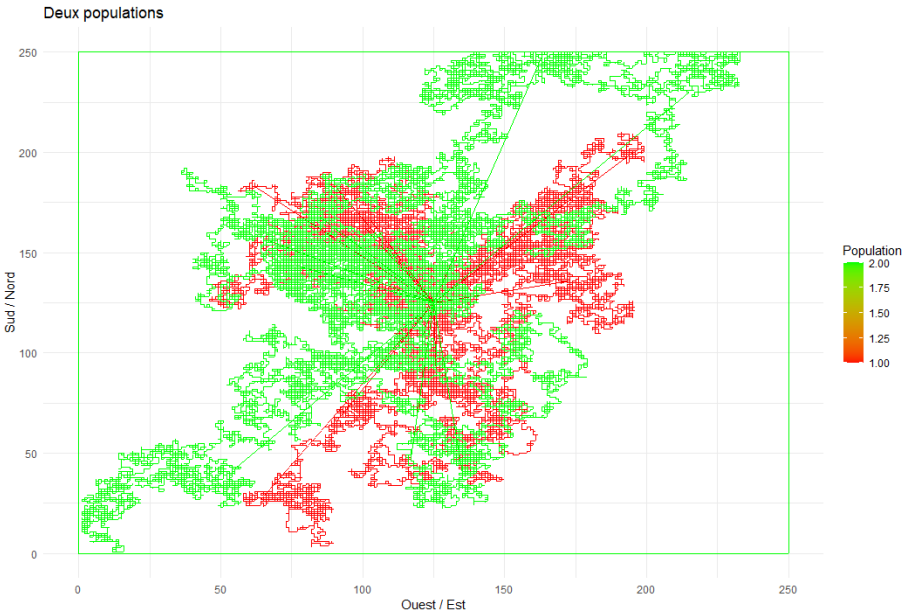


Figure 4.2.2.2 : Les deux populations sur le même graphique (code source compris dans le code de la figure précédente)



Nous remarquons que les individus de la *Population A* (à gauche sur le premier graphique, puis en rouge sur le deuxième) sont restés relativement proches du centre, ce qui a donc poussé les individus de la *Population B* (à droite sur le premier graphique puis en vert sur le deuxième) à aller chercher des points plus éloignés, notamment au Sud et au Nord.

4.2.3 Population par population

Figure 4.2.3.1 : À gauche la *Population A*, à droite la *Population B* (code source en Annexe 6.3)

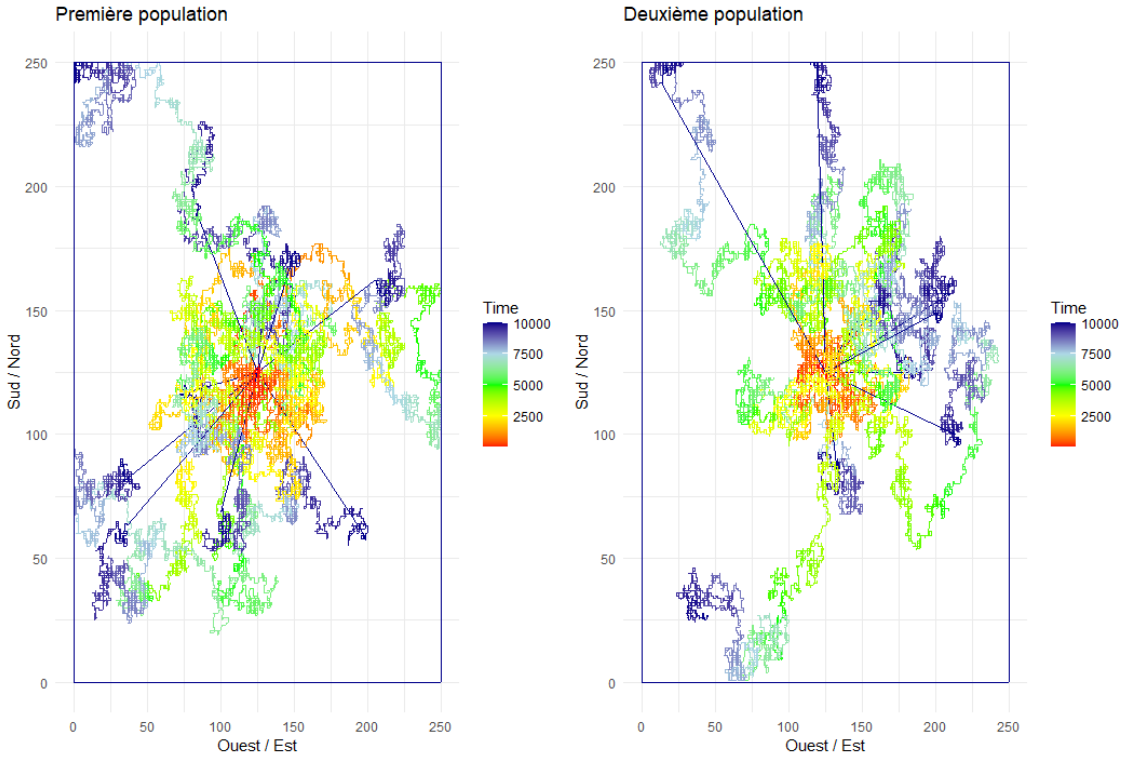
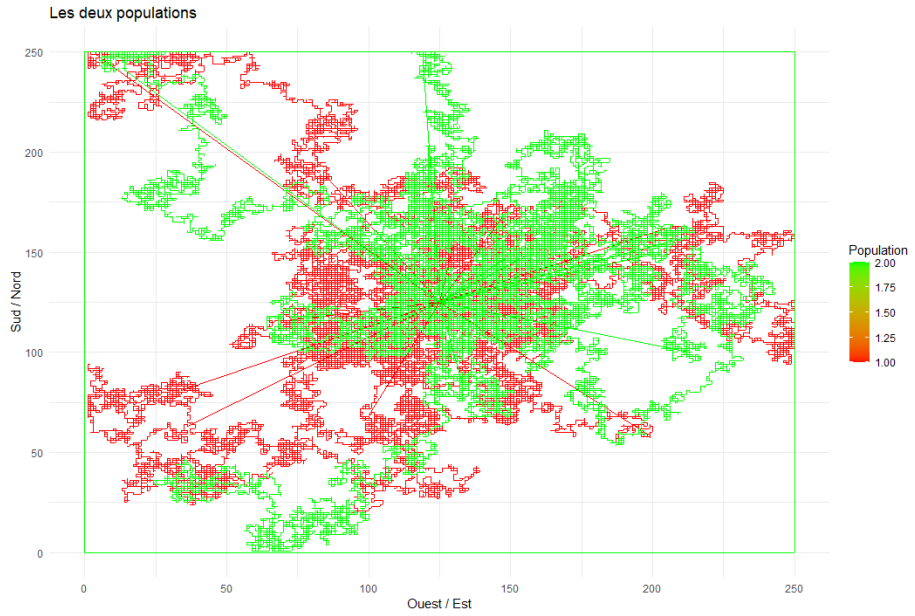


Figure 4.2.3.2 : Les deux populations sur le même graphique (code source compris dans le code de la figure précédente)



Nous supposions que le phénomène d'évitement serait le plus fort dans cette méthode de simulation car la *Population B* ne commence à marcher qu'une fois que tous les individus de la *Population A* ont fini de marcher.
Nous remarquons que la *Population A* a évolué librement en prenant beaucoup d'espace. Il est donc difficile de voir un véritable évitement de la part de la *Population B*. Ceci pourrait être mis en évidence de manière plus explicite si l'on utilisait des points de départs différents pour les deux populations. Cette idée sera donc développée dans la partie 4.4 .

4.3 Trois populations

Dans cette section, nous reprendrons la même démarche que précédemment mais avec trois populations.

4.3.1 Pas-à-pas

Figure 4.3.1.1 : À gauche la *Population A*, au centre la *Population B*, à droite la *Population C* (code source en Annexe 6.3)

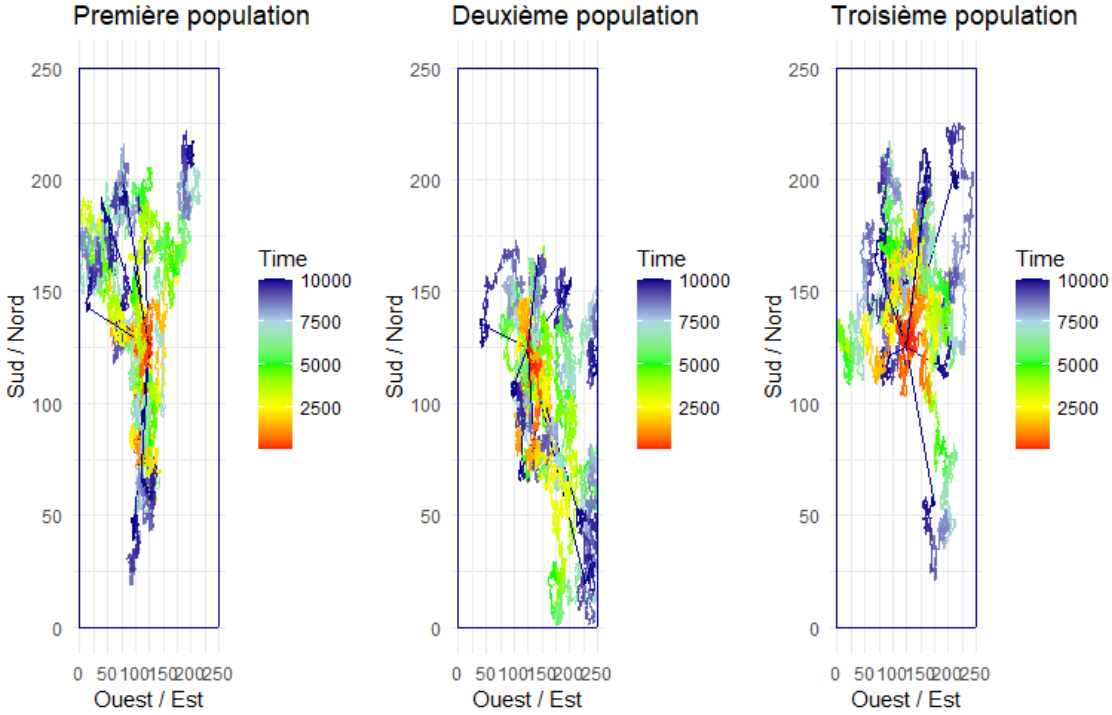


Figure 4.3.1.2 : Les trois populations sur le même graphique (code source compris dans le code de la figure précédente)

4.3.2 Individu par individu

Figure 4.3.2.1 : À gauche la *Population A*, au centre la *Population B*, à droite la *Population C* (code source en Annexe 6.3)

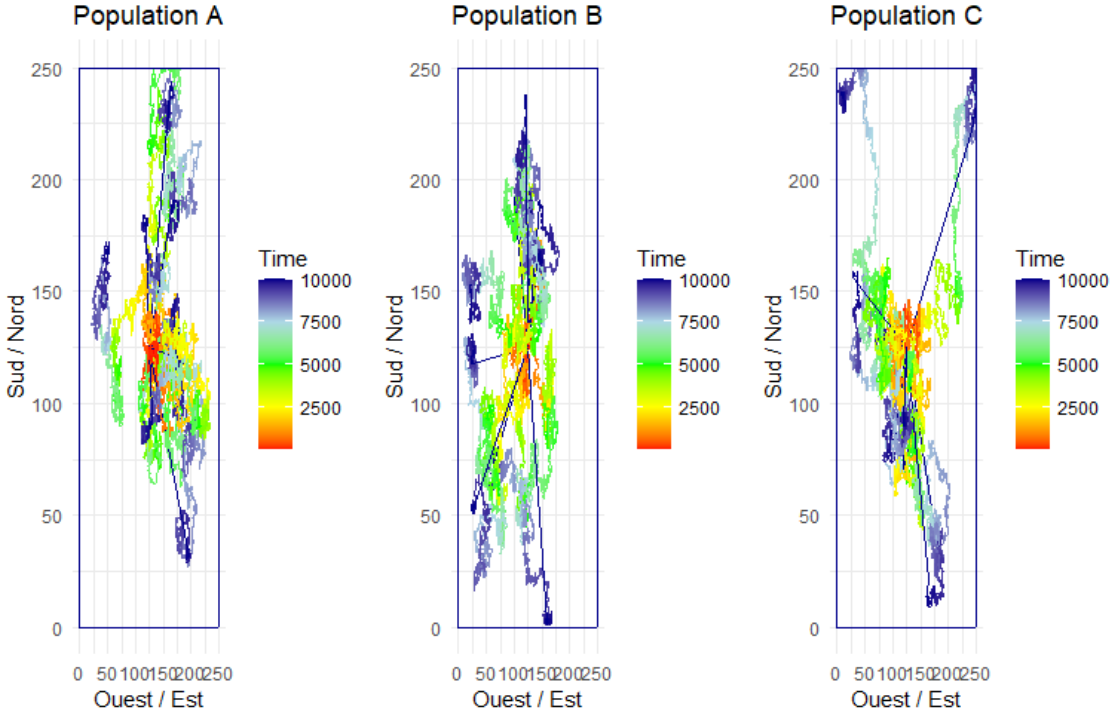
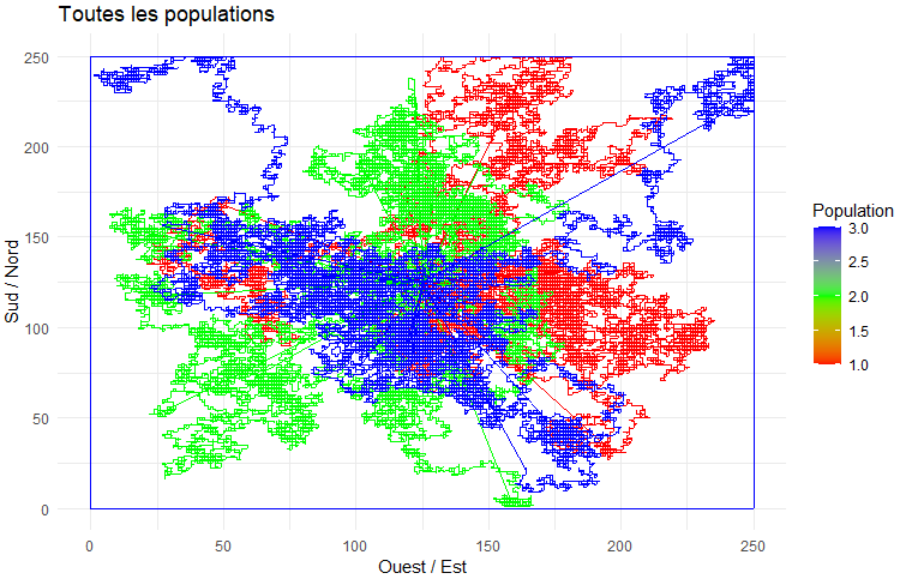


Figure 4.3.2.2 : Les trois populations sur le même graphique (code source compris dans le code de la figure précédente)



4.3.3 Population par population

Figure 4.3.3.1 : À gauche la *Population A*, au centre la *Population B*, à droite la *Population C* (code source en Annexe 6.3)

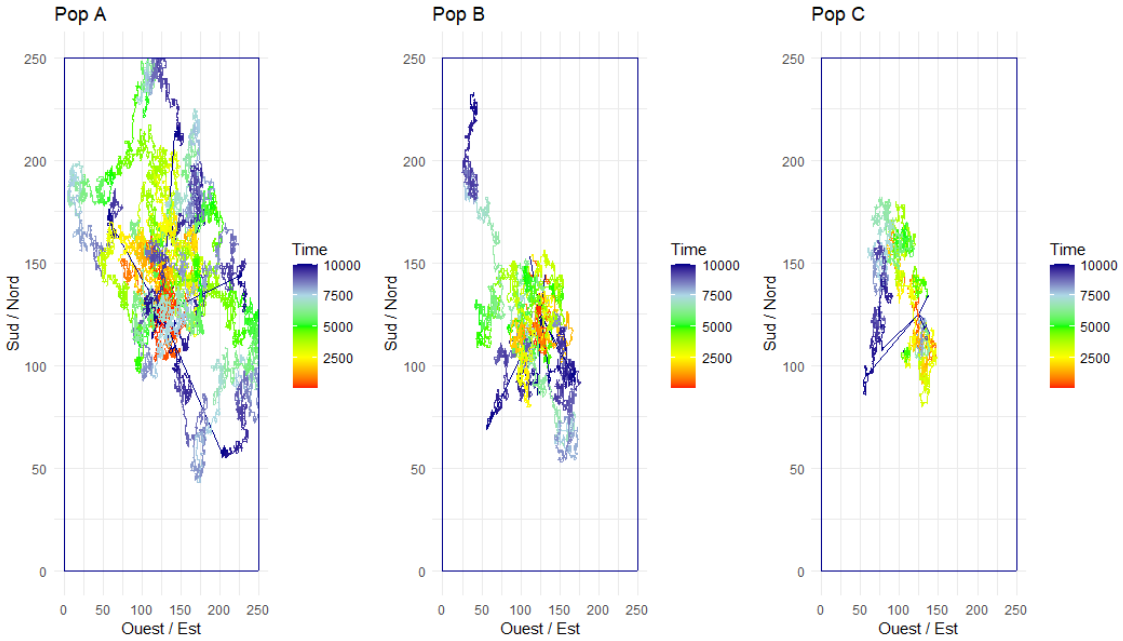
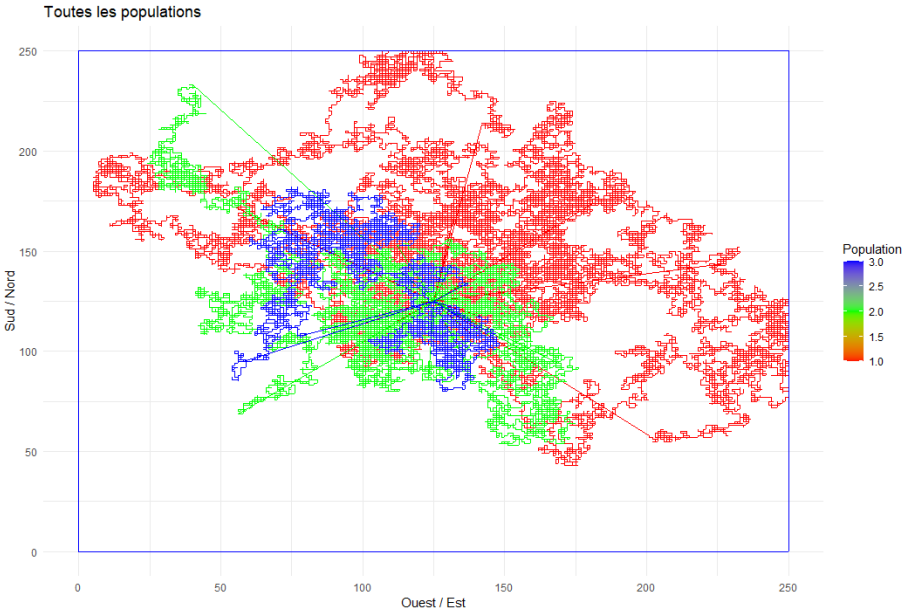


Figure 4.3.3.2 : Les trois populations sur le même graphique (code source compris dans le code de la figure précédente)

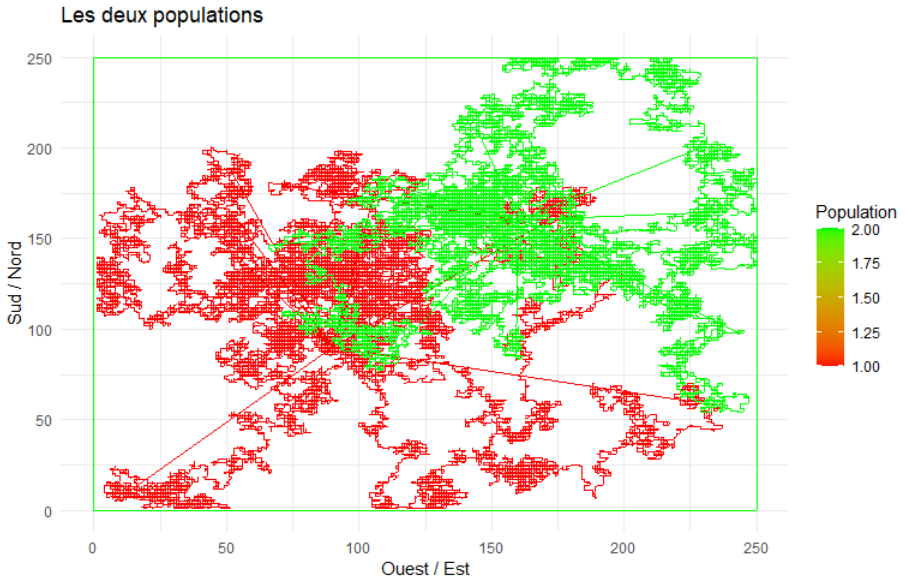


4.4 Points de départs différents

Comme supposé précédemment, nous allons maintenant refaire quelques simulations mais en mettant des zones de départ différentes pour chaque population.
Pour éviter les répétitions et maximimer l'effet de l'affaiblissement, nous utiliserons la méthode *"Population par population"* uniquement, pour 2 populations puis pour 3 populations.
Nous utiliserons un espacement entre les deux populations arbitrairement moyen : un espacement trop lointain empêcherait les interactions et un espacement trop faible est problématique car la *Population A* occupe trop l'espace de la *Population B* (comme vu dans la Figure 4.2.3.2). En effet, puisque la *Population A* est d'abord et entièrement modélisée avant de lancer la *Population B*, elle peut s'étendre à volonté tandis que la *Population B* n'a pas commencé.

4.4.1 Deux populations

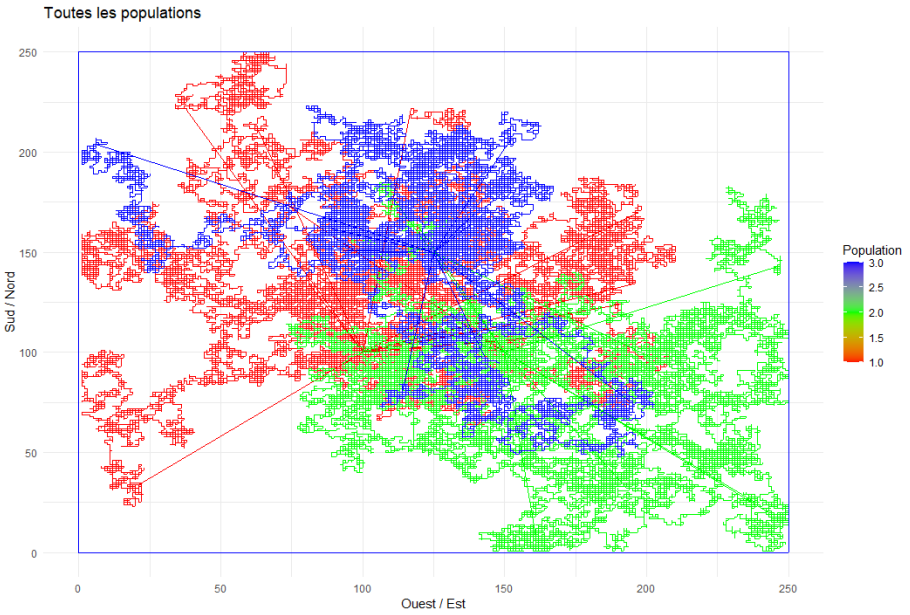
Figure 4.4.1.1 : Les deux populations sur le même graphique (code source en Annexe 6.3)



Nous avons ici mis comme points de départs (90,90) et (160,160).
On observe comme prévu un phénomène d'évitement, la *Population A* ayant principalement visité la partie Ouest, la *Population B* n'a pas pu aller explorer au Sud/Ouest et s'est donc contentée du cadre Nord/est.
De plus, lorsque l'on regarde la zone centrale entre les deux points de départ, on peut observer un "front" rouge que la *Population B* a eu beaucoup de mal à franchir.

4.4.2 Trois populations

Figure 4.4.2.1 : Les trois populations sur le même graphique (code source en Annexe 6.3)



Nous avons ici mis comme points de départs (100,100) et (150,100) et (100,150) afin d'avoir une disposition triangulaire homogène.
On observe comme prévu un phénomène d'évitement, la *Population B* en étant l'exemple le plus visible. Elle a principalement visité la partie Sud/est car la *Population A* l'avait bloquée..
De plus, la *Population C* a vu sa zone de départ fortement explorée par la *Population A* et s'est donc développée tant bien que mal. On notera que si une population est contrainte d'explorer une zone déjà fortement affaiblie, cela s'annule et la population explore donc à peu près comme s'il n'y avait pas eu de renforcement, et ce jusqu'à ce qu'elle trouve un endroit où les poids ne sont plus homogènes. Ainsi dès qu'elle trouvera une zone vierge, elle lui apparaitra renforcée par rapport aux autres arêtes disponibles. Nous appellerons ce phénomène "la fuite". Nous pouvons par exemple observer ce phénomène avec l'îlot bleu aux coordonnées (25,175) qui s'est frayé un chemin à travers le territoire rouge pour finir sa course en territoire vierge.

4.5 Conclusions

Durant cette section, nous avons développé de nouvelles fonctions permettant de modéliser plusieurs individus et plusieurs populations. Nous avons choisi de renforcer les matrices de transition pour les individus au sein d'une même population mais d'amoindrir ces mêmes matrices lorsqu'un individu d'une autre population emprunte une arête.

Nous avons donc pu implémenter un éco-système fortement interdépendant des autres individus ainsi que de l'ordre de modélisation. En effet, la manière dont sont générées les trajectoires influence fortement l'issue de la simulation : à savoir une simulation par pas, par individus ou par population.
Chacune de ces méthodes de modélisation va prioriser certains comportements, par exemple la simulation par pas est plus égalitaire car les premiers individus de chaque population seront assez libres alors que la méthode par population laisse champs libre à la première population et nous permet d'étudier le comportement d'une population qui évolue sous contrainte.

Nous avons observé plusieurs réalisations de cet éco-système avec une, deux ou trois populations. Nous avons aussi pu observer des phénomènes de territoire ainsi que d'axe de projection, tous deux conditionnés par la présence préalable d'un renforcement ou d'un affaiblissement.

Ces modélisations viennent mettre un terme à notre projet dont un maximum de variables ont été passées en paramètres afin de permettre une liberté d'utilisation du code, et de faciliter une éventuelle augmentation de l'échelle de la simulation pour les ordinateurs les plus robustes ou les personnes les plus patientes.

5 Conclusion du projet

5.1 Remerciements

Nous remercions Madame Frédérique BIENVENÜE pour son attention toute particulière ainsi que sa disponibilité.
Nous remercions aussi par avance toutes les personnes qui seront présentes lors de notre soutenance pour leur attention et leur intérêt.

5.2 Problématique et démarche

Ce projet s'est révélé intéressant et enrichissant pour diverses raisons. Il nous a poussé dans un premier temps à définir un cadre mathématique à notre sujet d'étude et donc à devoir découvrir puis fixer les propriétés de l'espace dans lequel nous évoluerons par la suite à savoir les grilles. Nous avons dû formaliser toutes les notions que nous devrions utiliser par la suite et cela nous a permis de mieux visualiser comment programmer nos algorithmes de marches aléatoires renforcées par la suite. Ceci nous a amené à formuler notre problématique de manière plus explicite : comment modéliser des marches aléatoires renforcées et que ces modélisations peuvent-elles nous indiquer ? Nous avons ensuite dû fixer l'ensemble des explorations que nous souhaitions faire pour répondre à cette problématique, ajoutant à nos idées de base les idées et concepts que nous trouvions dans les thèses fournies en bibliographie.
Nous avons choisi une approche expérimentale alternant algorithmie, modélisation, rendus graphiques et analyses afin d'obtenir des résultats visuels clairs couplés à une malléabilité maximale pour quiconque souhaite utiliser nos fonctions dans un autre cadre. Nous sommes ensuite passés à l'étape la plus prenante : la création des algorithmes. Ne trouvant que peu de documentation sur les algorithmes de simulation de marches aléatoires renforcées, nous avons pris parti de créer nos fonctions de toutes pièces. L'avantage de cette approche étant que nous pourrions par la suite modeler nos fonctions pour qu'elles s'adaptent à nos besoins. Une fois notre panel de fonctions créées et les paramétrisations de l'espace faites, nous avons dû utiliser notre sens de l'analyse pour extraire de nos modélisations des conjectures que nous pourrions par la suite mettre en évidence à travers des études statistiques sur les variations de chaque paramètre et leur effet.
Une fois ces statistiques effectuées, il nous tenait à coeur de faire tourner nos algorithmes sur un ensemble d'individus puis plusieurs ensembles d'individus. L'objectif étant de se rapprocher de simulations de trajectoires empruntées par les colonies de fourmis et d'en étudier si possible les comportements.

5.3 Problèmes rencontrés

Le principal problème rencontré a été sur le plan algorithmique. Nous nous sommes rendus compte plusieurs fois d'imprécisions de codage qui rendaient notre marche dénuée de sens ou d'efficacité. Une modification du code entrainant par cascades d'autres modifications, ces erreurs nous ont forcé à reprendre à plusieurs reprises nos simulations, graphiques et commentaires depuis le début. Un autre problème aura été le peu de documentation adaptée à notre

sujet d'étude. En effet les marches aléatoires sont très étudiées mais les marches aléatoires renforcées le sont moins et souvent sur des domaines d'études très spécifiques. Par conséquent, ces documents n'étaient pas toujours adaptables à nos attentes. Il est cependant probable que nous n'ayions pas non plus le niveau de compréhension suffisant pour en extraire les parties les plus pertinentes pour notre approche. À titre d'exemple la thèse de Madame Élodie Bouchet [Bou14] sur les marches aléatoires en milieu faiblement elliptique dont l'espace d'évolution ne correspond pas au notre et dont nous n'avons pas su adapter les recherches à notre cadre.

5.4 Conclusion des analyses

Dans la première partie, nous avons constaté que les marches aléatoires renforcées de manière multiplicative ont des comportements extrêmes dès que $\beta \geq 1,15$

Nous avons aussi pu conjecturer un premier effet de l'augmentation du renforcement : la compression de la trajectoire. En effet, d'une simple analyse visuelle nous avons remarqué que la trajectoire semble de plus en plus recroquevillée sur elle-même au fur et à mesure que le renforcement augmente. De plus, nous observons aussi un phénomène de zones d'agrégation au lieu des routes que nous pensions initialement observer.

C'est donc munis de cette conjecture que nous avons commencé la partie statistiques, regardant divers aspects de nos marches. Les conclusions de cette partie viennent corroborer notre conjecture sur plusieurs aspects : lorsque le renforcement augmente et le temps pour toucher le bord (ie dépasser une certaine distance) augmente, les distances maximales (ou au bout de N itérations) diminuent. On observe donc une compression de la trajectoire sur elle-même.

De plus, nous en avons profité pour comparer nos fonctions entre elles et nous sommes arrivés à la conclusion que la marche aléatoire renforcée avec demi-tour interdit explore davantage et donc plus vite que la marche aléatoire renforcée standard. Nous avons donc regardé cet aspect au travers des proportions de points visités.

Enfin, nous avons pu observer les effets de nos variables les unes sur les autres. Nous avons observé l'existence de valeurs critiques pour le renforcement, la taille de la grille ou bien le nombre d'itérations à partir desquelles la probabilité que l'individu touche ou non le bord sera de 0 ou 1 presque sûrement.

Dans la dernière partie, nous nous sommes concentrés sur les effets d'individus et populations les uns sur les autres. À ce stade nous pensions observer un phénomène de routes préférées (comme dans la nature) mais nous observons plutôt un phénomène de zones d'agrégations, qui se transforment en territoires lorsque l'on introduit plusieurs populations sur un même graphe.

Finalement, nous aboutissons sur une modélisation de mouvements de population. Là où nous pensions modéliser plutôt des chemins préférentiels pris par les fourmis d'une même colonie, nous avons ce que l'on pourrait imaginer comme une création de territoires pour des colonies de fourmis distinctes.

Dans une autre étude, il pourrait donc être envisageable de modéliser et d'étudier les évolutions de ces territoires peuplés au cours du temps.

6 Annexes

6.1 Annexes section Modélisations

Code source de la fonction MAR : (Retour à la page : 11)

```
MAR = function(P, alpha, beta, N, debut){
  Mar = debut
  for(i in 1:(N-1)){
    if((P[Mar[i],1] > 10**300) | (P[Mar[i],2] > 10**300)){
      warning('renforcement␣trop␣important')
      return(list(Mar = Mar, P = P))
      }

    #On tire un nombre aleatoire entre 0 et 1
    U = runif(1)

    #Si on est au bord de gauche et que notre U vaut 1
    #La boucle d'apres voudra aller a gauche (impossible)
    # Il faut donc s'assurer ici que nous ne sommes pas dans
        ce cas
    if((U == 1) && (Mar[i] == 1)){
      P[1,2] = beta*P[1,2] + alpha
      Mar = c(Mar, Mar[i] + 1)
    }
    else if(U < (P[Mar[i],2] / sum(P[Mar[i],]))){
      P[Mar[i],2] = beta*P[Mar[i],2] + alpha
                                  #On va a droite et on ajoute
          delta
      Mar = c(Mar, Mar[i] + 1)
    }
    else{

      #Sinon pareil mais on va a gauche
      P[Mar[i],1] = beta*P[Mar[i],1] + alpha
      Mar = c(Mar, Mar[i] - 1)
    }
  }
  return(list(Mar = Mar, P = P))
}
```

Code source de la fonction MAR2D : (Retour à la page : 13)

```
MAR2d = function(P, Q, alpha, beta, N, debut){
  Mar = matrix(0, nrow = 2, ncol = N)
  Mar[1,1] = debut
  Mar[2,1] = debut
  K = nrow(P)
  Sommet = matrix(0, nrow = K, ncol = K)
                               # On compte chaque passage
     chaque sommet
  ArreteP = matrix(0, nrow = K, ncol = 2*K)
                            # On compte chaque passage
     chaque ar te horizontale
  ArreteQ = matrix(0, nrow = K, ncol = 2*K)
                            # On compte chaque passage
```

```
    chaque ar te verticale
for(i in 1:(N-1)){
  U = runif(1)                                                    #On
      tire un nombre aleatoire entre 0 et 1
  a = P[Mar[1,i],  Mar[2,i]]
                                                  #On cr er des
      variables pour simplifier l' criture
  b = P[Mar[1,i],  Mar[2,i] + K]
  c = Q[Mar[1,i],  Mar[2,i] + K]
  d = Q[Mar[1,i],  Mar[2,i]]
  Sommet[Mar[1,i],  Mar[2,i]] = Sommet[Mar[1,i],  Mar[2,i
      ]] + 1
  if((a > 10**300) | (b > 10**300) | (c > 10**300) | (d >
      10**300)){
    warning('renforcement␣trop␣important')
    return(list(Mar = matrix(Mar[Mar > 0], nrow = 2, byrow
        = FALSE), P = P, Q = Q, i = i, ArreteP = ArreteP,
       ArreteQ = ArreteQ, Sommet = Sommet))
  }
  if(U < (a / (a + b + c + d))){        #Si U est plus
      petit que la proba d'aller a gauche (Poid d'aller a
      gauche diviser par la somme d'aller a droite, a
      gauche, devant et derriere)
    P[Mar[1,i], Mar[2,i]] = beta * P[Mar[1,i], Mar[2,i]] +
        alpha                              #On va a gauche et
        on ajoute delta
    Mar[1, i + 1] = Mar[1, i] - 1
    Mar[2, i + 1] = Mar[2, i]
    ArreteP[Mar[1,i], Mar[2,i]] = ArreteP[Mar[1,i], Mar[2,
        i]] + 1
  }
  else if(U < ((a + b) / (a + b + c + d))){    #Si U est
      entre les probas d'aller a gauche et d'aller a droite
       ou a gauche (Poid d'aller a gauche + a droite
      diviser par la somme d'aller a droite, a gauche,
      devant et derriere, car si arrive ici c'est que U est
       plus grand que la proba d'aller a gauche)
    P[Mar[1,i], Mar[2,i] + K] = beta * P[Mar[1,i], Mar[2,i
        ] + K] + alpha
                                                              #On
        va a droite et on enleve delta
    Mar[1, i + 1] = Mar[1, i] + 1
    Mar[2, i + 1] = Mar[2, i]
    ArreteP[Mar[1,i], Mar[2,i] + K] = ArreteP[Mar[1,i],
        Mar[2,i] + K] + 1
  }
  else if(U < ((a + b + c) / (a + b + c + d))){       #Si
      U est entre les probas d'aller a gauche ou a droite
      et d'aller a droite ou a gauche ou devant (Poid d'
      aller a gauche + a droite + devant diviser par la
      somme d'aller a droite, a gauche, devant et derriere,
       car si on arrive ici c'est que U est plus grand que
      la proba d'aller a gauche ou a droite)
```

```
      Q[Mar[1,i], Mar[2,i] + K] = beta * Q[Mar[1,i], Mar[2,i
          ] + K] + alpha

          #On va devant et on ajoute delta
      Mar[1, i + 1] = Mar[1, i]
      Mar[2, i + 1] = Mar[2, i] + 1
      ArreteQ[Mar[1,i], Mar[2,i] + K] = ArreteQ[Mar[1,i],
          Mar[2,i] + K] + 1
    }
    else if((U != 1) | (Mar[2,i] != 1)){

        #Si U est different de 1 ou que nous ne sommes pas
        tout en bas
      Q[Mar[1,i], Mar[2,i]] = beta * Q[Mar[1,i], Mar[2,i]] +
           alpha

          #On va derriere et on enleve delta
      Mar[1, i + 1] = Mar[1, i]
      Mar[2, i + 1] = Mar[2, i] - 1
      ArreteQ[Mar[1,i], Mar[2,i]] = ArreteQ[Mar[1,i], Mar[2,
          i]] + 1
    }
    else{

        #Si U est egal a 1 et que nous sommes tout en bas,
      Mar[1, i + 1] = Mar[1, i]
                                          #On choisit de
          rester sur place sans changer les poids pour ne pas
           fausser l'aleatoire (Juste pour cette iteration).
      Mar[2, i + 1] = Mar[2, i]
                                          #Ce cas n'a
          quasiment aucune chance d'arriver mais il faut tout
           de meme l'anticiper
    }
  }
  Sommet[Mar[1,N],  Mar[2,N]] = Sommet[Mar[1,N],  Mar[2,N]]
      + 1
  return(list(Mar = Mar, P = P, Q = Q, ArreteP = ArreteP,
      ArreteQ = ArreteQ, Sommet = Sommet))
}
```

Code source de la fonction MARri : (Retour à la page : 17)

```
MARri = function(P, Q, alpha, beta, N, debut){
  Mar = matrix(0, nrow = 2, ncol = N)
  Mar[1,1] = debut
  Mar[2,1] = debut
  K = nrow(P)
  Sommet = matrix(0, nrow = K, ncol = K)
                              # On compte chaque passage
      chaque sommet
  ArreteP = matrix(0, nrow = K, ncol = 2*K)
                            # On compte chaque passage
      chaque ar te horizontale
  ArreteQ = matrix(0, nrow = K, ncol = 2*K)
```

```
                        # On compte chaque passage
    chaque ar te verticale
a = P[debut,debut]
                                                        #On
    initialise 4 variable a, b, c et d qui contiennent
b = P[debut,debut + K]
                                                        # le
    poids d'aller dans chacune des directions par rapport
c = Q[debut,debut + K]
                                                        # a l'
    endroit o   nous sommes a l'instant t
d = Q[debut,debut]
e = 0

    #On initialise un e qui contiendra le mouvement realise
     precedemment (1 si on est alle a gauche, 2 a droite, 3
     devant et 4 derriere)
for(i in 1:(N-1)){
  if((P[Mar[1,i], Mar[2,i]] > 10**300) | (P[Mar[1,i], Mar[
      2,i] + K] > 10**300) | (Q[Mar[1,i], Mar[2,i]] > 10**3
      00) | (Q[Mar[1,i], Mar[2,i] + K] > 10**300)){
    warning('renforcement trop important')
    return(list(Mar = matrix(Mar[Mar > 0], nrow = 2, byrow
        = FALSE), P = P, Q = Q, i = i, ArreteP = ArreteP,
       ArreteQ = ArreteQ, Sommet = Sommet))
  }
  Sommet[Mar[1,i],  Mar[2,i]] = Sommet[Mar[1,i],  Mar[2,i
      ]] + 1
  U = runif(1)
                                                          #On
      tire un nombre aleatoire entre 0 et 1
  if(U < (a / (a + b + c + d))){
                                      #Si U est plus petit
      que la proba d'aller a gauche (Poid d'aller a gauche
      diviser par la somme d'aller a droite, a gauche,
      devant et derriere)
    P[Mar[1,i], Mar[2,i]] = beta * P[Mar[1,i], Mar[2,i]] +
        alpha                         #On va a gauche et on
       ajoute delta
    Mar[1, i + 1] = Mar[1, i] - 1
    Mar[2, i + 1] = Mar[2, i]
    ArreteP[Mar[1,i], Mar[2,i]] = ArreteP[Mar[1,i], Mar[2,
        i]] + 1
    e = 1

        #On enregistre notre mouvement (on est alle a
        gauche)
  }
  else if(U < ((a + b) / (a + b + c + d))){
                          #Si U est entre les probas d'
      aller a gauche et d'aller a droite ou a gauche (Poid
      d'aller a gauche + a droite diviser par la somme d'
      aller a droite, a gauche, devant et derriere, car si
      arrive ici c'est que U est plus grand que la proba d'
```

```
    aller a gauche)
  P[Mar[1,i], Mar[2,i] + K] = beta * P[Mar[1,i], Mar[2,i
      ] + K] + alpha                          #On va a
      droite et on enleve delta
  Mar[1, i + 1] = Mar[1, i] + 1
  Mar[2, i + 1] = Mar[2, i]
  ArreteP[Mar[1,i], Mar[2,i] + K] = ArreteP[Mar[1,i],
      Mar[2,i] + K] + 1
  e = 2

      #On enregistre notre mouvement (on est alle a
      droite)
}
else if(U < ((a + b + c) / (a + b + c + d))){
                    #Si U est entre les probas d'aller a
    gauche ou a droite et d'aller a droite ou a gauche ou
     devant (Poid d'aller a gauche + a droite + devant
    diviser par la somme d'aller a droite, a gauche,
    devant et derriere, car si on arrive ici c'est que U
    est plus grand que la proba d'aller a gauche ou a
    droite)
  Q[Mar[1,i], Mar[2,i] + K] = beta * Q[Mar[1,i], Mar[2,i
      ] + K] + alpha                          #On va devant
      et on ajoute delta
  Mar[1, i + 1] = Mar[1, i]
  Mar[2, i + 1] = Mar[2, i] + 1
  ArreteQ[Mar[1,i], Mar[2,i] + K] = ArreteQ[Mar[1,i],
      Mar[2,i] + K] + 1
  e = 3

      #On enregistre notre mouvement (on est alle devant)
}
else if((U != 1) | (Mar[2,i] != 1)){
                              #Si U est different de 1 ou
    que nous ne sommes pas tout en bas
  Q[Mar[1,i], Mar[2,i]] = beta * Q[Mar[1,i], Mar[2,i]] +
       alpha                          #On va derriere et on
      enleve delta
  Mar[1, i + 1] = Mar[1, i]
  Mar[2, i + 1] = Mar[2, i] - 1
  ArreteQ[Mar[1,i], Mar[2,i]] = ArreteQ[Mar[1,i], Mar[2,
      i]] + 1
  e = 4

      #On enregistre notre mouvement (on est alle
      derriere)
}
else{

    #Si U est egal a 1 et que nous sommes tout en bas,
  Mar[1, i + 1] = Mar[1, i]
                                        #On choisit de
      rester sur place sans changer les poids pour ne pas
       fausser l'aleatoire (Juste pour cette iteration).
```

```
      Mar[2, i + 1] = Mar[2, i]
                                                #Ce cas n'a
          quasiment aucune chance d'arrive mais il faut tout
          de meme l'anticiper
    }
    a = P[Mar[1, i + 1],  Mar[2,i + 1]]
                                                  #On actualise
        nos variables avec les nouvelles valeurs pour l'
        instant suivant
    b = P[Mar[1, i + 1],  Mar[2,i + 1] + K]
    c = Q[Mar[1, i + 1],  Mar[2,i + 1] + K]
    d = Q[Mar[1, i + 1],  Mar[2,i + 1]]
    if(e == 1){
                                                                #On
         met en place l'interdiction de retour en arriere
      b = 0

          #Si on est alle a gauche, on a l'interdiction d'
          aller a droite a l'instant suivant
    }
    else if(e == 2){
      a = 0

          #Si on est alle a droite, on a l'interdiction d'
          aller a gauche a l'instant suivant
    }
    else if(e == 3){
      d = 0

          #Si on est alle devant, on a l'interdiction d'aller
           derriere a l'instant suivant
    }
    else if(e ==4){
      c = 0

          #Si on est alle derriere, on a l'interdiction d'
          aller devant a l'instant suivant
    }
  }
  Sommet[Mar[1,N],  Mar[2,N]] = Sommet[Mar[1,N],  Mar[2,N]]
      + 1
  return(list(Mar = Mar, P = P, Q = Q, ArreteP = ArreteP,
      ArreteQ = ArreteQ, Sommet = Sommet))
}
```

Code source de la fonction MARstop : (Retour à la page : 19)

```
MARstop = function(P, Q, alpha, beta, N, debut){
  Mar = matrix(0, nrow = 2, ncol = N)
  Mar[1,1] = debut
  Mar[2,1] = debut
  K = nrow(P)
  Sommet = matrix(0, nrow = K, ncol = K)
                              # On compte chaque passage
      chaque sommet
```

```
ArreteP = matrix(0, nrow = K, ncol = 2*K)
                          # On compte chaque passage
    chaque ar te horizontale
ArreteQ = matrix(0, nrow = K, ncol = 2*K)
                          # On compte chaque passage
    chaque ar te verticale
for(i in 1:(N-1)){
  if((Mar[1,i] == K) | (Mar[1,i] == 1) | (Mar[2,i] == K) |
      (Mar[2,i] == 1)){                    #Si on est sur
      un bord
    return(list(Mar = matrix(Mar[Mar > 0], nrow = 2, byrow
        = FALSE), P = P, Q = Q, i = i, ArreteP = ArreteP,
       ArreteQ = ArreteQ, Sommet = Sommet))     #On
       renvoie la trajectoire
  }
  Sommet[Mar[1,i],  Mar[2,i]] = Sommet[Mar[1,i],  Mar[2,i
      ]] + 1
  a = P[Mar[1,i],  Mar[2,i]]
                                                #On cr er des
      variables pour simplifier l' criture
  b = P[Mar[1,i],  Mar[2,i] + K]
  c = Q[Mar[1,i],  Mar[2,i] + K]
  d = Q[Mar[1,i],  Mar[2,i]]
  if((a > 10**300) | (b > 10**300) | (c > 10**300) | (d >
      10**300)){
    warning('renforcement␣trop␣important')
    return(list(Mar = matrix(Mar[Mar > 0], nrow = 2, byrow
        = FALSE), P = P, Q = Q, i = i, ArreteP = ArreteP,
       ArreteQ = ArreteQ, Sommet = Sommet))
  }
  U = runif(1)

     #On tire un nombre aleatoire entre 0 et 1
  if(U < (a / (a + b + c + d))){        #Si U est plus
     petit que la proba d'aller a gauche (Poid d'aller a
     gauche diviser par la somme d'aller a droite, a
     gauche, devant et derriere)
    P[Mar[1,i], Mar[2,i]] = beta * P[Mar[1,i], Mar[2,i]] +
        alpha                           #On va a gauche et
        on ajoute delta
    Mar[1, i + 1] = Mar[1, i] - 1
    Mar[2, i + 1] = Mar[2, i]
    ArreteP[Mar[1,i], Mar[2,i]] = ArreteP[Mar[1,i], Mar[2,
       i]] + 1
  }
  else if(U < ((a + b) / (a + b + c + d))){    #Si U est
     entre les probas d'aller a gauche et d'aller a droite
      ou a gauche (Poid d'aller a gauche + a droite
     diviser par la somme d'aller a droite, a gauche,
     devant et derriere, car si arrive ici c'est que U est
      plus grand que la proba d'aller a gauche)
    P[Mar[1,i], Mar[2,i] + K] = beta * P[Mar[1,i], Mar[2,i
       ] + K] + alpha
                                                      #On
```

```
            va a droite et on enleve delta
          Mar[1, i + 1] = Mar[1, i] + 1
          Mar[2, i + 1] = Mar[2, i]
          ArreteP[Mar[1,i], Mar[2,i] + K] = ArreteP[Mar[1,i],
              Mar[2,i] + K] + 1
        }
        else if(U < ((a + b + c) / (a + b + c + d))){       #Si
            U est entre les probas d'aller a gauche ou a droite
            et d'aller a droite ou a gauche ou devant (Poid d'
            aller a gauche + a droite + devant diviser par la
            somme d'aller a droite, a gauche, devant et derriere,
             car si on arrive ici c'est que U est plus grand que
            la proba d'aller a gauche ou a droite)
          Q[Mar[1,i], Mar[2,i] + K] = beta * Q[Mar[1,i], Mar[2,i
              ] + K] + alpha

            #On va devant et on ajoute delta
          Mar[1, i + 1] = Mar[1, i]
          Mar[2, i + 1] = Mar[2, i] + 1
          ArreteQ[Mar[1,i], Mar[2,i] + K] = ArreteQ[Mar[1,i],
              Mar[2,i] + K] + 1
        }
        else if((U != 1) | (Mar[2,i] != 1)){

            #Si U est different de 1 ou que nous ne sommes pas
            tout en bas
          Q[Mar[1,i], Mar[2,i]] = beta * Q[Mar[1,i], Mar[2,i]] +
               alpha

            #On va derriere et on enleve delta
          Mar[1, i + 1] = Mar[1, i]
          Mar[2, i + 1] = Mar[2, i] - 1
          ArreteQ[Mar[1,i], Mar[2,i]] = ArreteQ[Mar[1,i], Mar[2,
              i]] + 1
        }
        else{

            #Si U est egal a 1 et que nous sommes tout en bas,
          Mar[1, i + 1] = Mar[1, i]
                                                #On choisit de
            rester sur place sans changer les poids pour ne pas
             fausser l'aleatoire (Juste pour cette iteration).
          Mar[2, i + 1] = Mar[2, i]
                                                #Ce cas n'a
            quasiment aucune chance d'arrive mais il faut tout
            de meme l'anticiper
        }
      }
      Sommet[Mar[1,N],  Mar[2,N]] = Sommet[Mar[1,N],  Mar[2,N]]
          + 1
      return(list(Mar = Mar, P = P,Q = Q, ArreteP = ArreteP,
          ArreteQ = ArreteQ, Sommet = Sommet))
    }
    }
```

Code source de la paramétrisation MA : (Retour à la page : 11)

```
K = 200          #On a une grille de taille K

P1 = matrix(1,nrow=K,ncol=2)    #Matrice des mouvements
P1[1,1]= 0
P1[K,2]= 0

debut = floor(K/2)
alpha = 0 # parametre additif ici    0
beta = 1 # parametre multiplicatif ici    1
# On a donc bien une marche al atoire sans renforcement


N = 50000 # nombre d'it rations du processus
```

Code source de la paramétrisation MAR : (Retour à la page : 11)

```
K = 200          #On a une grille de taille K

P1 = matrix(1,nrow=K,ncol=2)    #Matrice des mouvements
P1[1,1]= 0
P1[K,2]= 0

debut = floor(K/2)
alpha = 0 #parametre additif
beta = 1 #parametre multiplicatif

N = 50000
```

Code source de la paramétrisation MAR2D : (Retour à la page : 13)

```
K = 400                          #On a une grille de taille
    K*K

P = matrix(1,nrow=K,ncol=2*K)    #Matrice des mouvements
    horizontaux
for(i in 1:K){
  P[1, i] = 0
  P[K, K+i] = 0
}

Q = matrix(1,nrow=K,ncol=2*K)    #Matrice des mouvements
    verticaux
for(i in 1:K){
  Q[i,1] = 0
  Q[i, 2*K] = 0
}

debut = floor(K/2)
alpha = 0.5 # parametre additif
beta = 1 # parametre multiplicatif

N = 50000 # nombre d'it rations du processus
```

Code source de la paramétrisation MARri : (Retour à la page : 17)

```
K = 400                              #On a une grille de taille
    K*K

P = matrix(1,nrow=K,ncol=2*K)        #Matrice des mouvements
    horizontaux
for(i in 1:K){
  P[1, i] = 0
  P[K, K+i] = 0
}

Q = matrix(1,nrow=K,ncol=2*K)        #Matrice des mouvements
    verticaux
for(i in 1:K){
  Q[i,1] = 0
  Q[i, 2*K] = 0
}

debut = floor(K/2)
alpha = 1 # parametre additif
beta = 1 # parametre multiplicatif

N = 50000 # nombre d'it rations du processus
```

Code source de la paramétrisation Comparaisons : (Retour à la page : 20)

```
K = 400                              #On a une grille de taille
    K*K

P1 = matrix(1,nrow=K,ncol=2)          #Matrice pour les MAR
    une dimension
P1[1,1]= 0
P1[K,2]= 0

P = matrix(1,nrow=K,ncol=2*K)        #Matrice des mouvements
    horizontaux
for(i in 1:K){
  P[1, i] = 0
  P[K, K+i] = 0
}

Q = matrix(1,nrow=K,ncol=2*K)        #Matrice des mouvements
    verticaux
for(i in 1:K){
  Q[i,1] = 0
  Q[i, 2*K] = 0
}

debut = floor(K/2)
N = 50000
```

Code source de la Figure 2.1 : (Retour à la page : 11)

```
X = MAR(P1, alpha, beta, N, debut)
plot(X ,type='s',main="Marche␣Aleatoire",
     xlab="Temps",
```

```
    ylab="Valeurs")
```

Code source de la Figure 2.2 : (Retour à la page : 12)

```
alpha = 0.1
beta = 1
par(mfrow=c(1,2))
X = MAR(P1, alpha, beta, N, debut)$Mar
plot(X ,type='s',main="MAR additive",
     xlab="Temps",
     ylab="Valeurs")

alpha = 0
beta = 1.005
Y = MAR(P1, alpha, beta, N, debut)$Mar
plot(Y ,type='s',main="MAR multiplicative",
     xlab="Temps",
     ylab="Valeurs")
```

Code source de la Figure 2.3.1 : (Retour à la page)

```
alpha = 1
beta = 1

Y = MAR2d(P, Q, alpha, beta, N, debut)$Mar
x = Y[1,]
y = Y[2,]
walk = 1:ncol(Y)
df <- data.frame(walk, x, y)

gradient = c("red","yellow","green", "lightblue","darkblue")


plot = ggplot(df, aes(x, y, color = 1:ncol(Y))) + geom_path
    () + labs(color = "Time") + scale_colour_gradientn(colors
     = gradient) + theme_minimal()
#print(plot)                                        #On remarque
    qu'il est difficile de voir les zones d'aller retour

#plot = plot + geom_density2d(color = "purple")
#print(plot)                                        #Nous avons
    donc rajoute ici les bassins d'attractions afin de mieux
    visualiser les zones d'aggregations

#Pour rajouter les bords
plot = plot + geom_segment(aes(x = 0, y = 0, xend = K, yend
    = 0)) + geom_segment(aes(x = 0, y = K, xend = K, yend = K
    )) + geom_segment(aes(x = 0, y = 0, xend = 0, yend = K))
    + geom_segment(aes(x = K, y = 0, xend = K, yend = K))
plot = plot + labs(title="  volution  d'un individu en 2D",x
    ="Ouest / Est", y = "Sud / Nord")

print(plot)
```

Code source de la Figure 2.3.2 : (Retour à la page)

```
alpha = 0
beta = 1.15

Y = MAR2d(P, Q, alpha, beta, N, debut)$Mar
x = Y[1,]
y = Y[2,]
walk = 1:ncol(Y)
df <- data.frame(walk, x, y)

gradient = c("red","yellow","green", "lightblue","darkblue")


plot = ggplot(df, aes(x, y, color = 1:ncol(Y))) + geom_path
    () + labs(color = "Time") + scale_colour_gradientn(colors
     = gradient) + theme_minimal()
#print(plot)                                        #On remarque
    qu'il est difficile de voir les zones d'aller retour

plot = plot + geom_density2d(color = "purple")
#print(plot)                                        #Nous avons
    donc rajoute ici les bassins d'attractions afin de mieux
    visualiser les zones d'aggregations

#Pour rajouter les bords
plot = plot + geom_segment(aes(x = 0, y = 0, xend = K, yend
    = 0)) + geom_segment(aes(x = 0, y = K, xend = K, yend = K
    )) + geom_segment(aes(x = 0, y = 0, xend = 0, yend = K))
    + geom_segment(aes(x = K, y = 0, xend = K, yend = K))
plot = plot + labs(title=" volution ␣d'un␣individu␣en␣2D",x
    ="Ouest␣/␣Est", y = "Sud␣/␣Nord")

print(plot)
```

Code source de la Figure 2.4.1 : (Retour à la page)

```
alpha = 0
beta = 1.15

Y = MAR2d(P, Q, alpha, beta, N, debut)$Mar
x = Y[1,]
y = Y[2,]
walk = 1:ncol(Y)
df <- data.frame(walk, x, y)

gradient = c("red","yellow","green", "lightblue","darkblue")


plot = ggplot(df, aes(x, y, color = 1:ncol(Y))) + geom_path
    () + labs(color = "Time") + scale_colour_gradientn(colors
     = gradient) + theme_minimal()
#print(plot)                                        #On remarque
    qu'il est difficile de voir les zones d'aller retour

#plot = plot + geom_density2d(color = "purple")
#print(plot)                                        #Nous avons
```

```
    donc rajoute ici les bassins d'attractions afin de mieux
    visualiser les zones d'aggregations

#Pour rajouter les bords
plot = plot + geom_segment(aes(x = 0, y = 0, xend = K, yend
    = 0)) + geom_segment(aes(x = 0, y = K, xend = K, yend = K
    )) + geom_segment(aes(x = 0, y = 0, xend = 0, yend = K))
    + geom_segment(aes(x = K, y = 0, xend = K, yend = K))
plot = plot + labs(title="  volution  d'un individu en 2D",x
    ="Ouest / Est", y = "Sud / Nord")

print(plot)
```

Code source de la Figure 2.5.1 : (Retour à la page)

```
alpha = 1
beta = 1

Y2 = MARstop(P, Q, alpha, beta, N, debut)$Mar

x2 = Y2[1,]
y2 = Y2[2,]
walk = 1:ncol(Y2)
df2 <- data.frame(walk, x2, y2)

gradient = c("red","yellow","green", "lightblue","darkblue")


plot2 = ggplot(df2, aes(x2, y2, color = 1:ncol(Y2))) + geom_
    path() + labs(color = "Time") + scale_colour_gradientn(
    colors = gradient) + theme_minimal()
#print(plot1)                                    #On remarque
    qu'il est difficile de voir les zones d'aller retour

#plot2 = plot1 + geom_density2d(color = "purple")
#print(plot1)                                    #Nous avons
    donc rajoute ici les bassins d'attractions afin de mieux
    visualiser les zones d'aggregations

# Pour rajouter les bords
plot2 = plot2 + geom_segment(aes(x = 0, y = 0, xend = K,
    yend = 0)) + geom_segment(aes(x = 0, y = K, xend = K,
    yend = K)) + geom_segment(aes(x = 0, y = 0, xend = 0,
    yend = K)) + geom_segment(aes(x = K, y = 0, xend = K,
    yend = K))
print(plot2)
```

Code source de la Figure 2.6.1 : (Retour à la page)

```
### 1 dimension (renforcement additif de 0.1)

alpha = 0.1
beta = 1
par(mfrow = c(1,2))
plot(MAR(P1, alpha, beta, N, debut)$Mar, type = 's', ylim =
    c(0,K),xlab="Temps",ylab="Valeurs", main="Renforcement
```

```
    additif")
plot(MAR(P1, 0, 1, N, debut)$Mar, type = 's', ylim = c(0,K),
    xlab="Temps",ylab="Valeurs", main="Pas de renforcement")
```

Code source de la Figure 2.6.2 : (Retour à la page)

```
### 2 dimensions (renforcement additif de 0.3)
alpha = 0.3
beta = 1

Xrenf = MAR2d(P, Q, alpha, beta, N, debut)$Mar
Xnonrenf = MAR2d(P, Q, 0, 1, N, debut)$Mar

gradient = c("red","yellow","green", "lightblue","darkblue")
walk = 1:N

xrenf = Xrenf[1,]
yrenf = Xrenf[2,]
dfrenf <- data.frame(walk, xrenf, yrenf)

xnonrenf = Xnonrenf[1,]
ynonrenf = Xnonrenf[2,]
dfnonrenf <- data.frame(walk, xnonrenf, ynonrenf)


plot = ggplot(dfrenf, aes(xrenf, yrenf, color = 1:N)) + geom
    _path() + labs(color = "Time") + scale_colour_gradientn(
    colors = gradient) + theme_minimal() + geom_segment(aes(x
     = 0, y = 0, xend = K, yend = 0)) + geom_segment(aes(x =
    0, y = K, xend = K, yend = K)) + geom_segment(aes(x = 0,
    y = 0, xend = 0, yend = K)) + geom_segment(aes(x = K, y =
     0, xend = K, yend = K))+ labs (title = "MAR2D renforc e
    " ,x="Ouest / Est" , y = "Sud / Nord")
plot1 = ggplot(dfnonrenf, aes(xnonrenf, ynonrenf, color = 1:
    N)) + geom_path() + labs(color = "Time") + scale_colour_
    gradientn(colors = gradient) + theme_minimal() + geom_
    segment(aes(x = 0, y = 0, xend = K, yend = 0)) + geom_
    segment(aes(x = 0, y = K, xend = K, yend = K)) + geom_
    segment(aes(x = 0, y = 0, xend = 0, yend = K)) + geom_
    segment(aes(x = K, y = 0, xend = K, yend = K))+ labs (
    title = "MAR2D non renforc e" ,x="Ouest / Est" , y = "
    Sud / Nord")
grid.arrange(plot, plot1, ncol=2)
```

Code source de la Figure 2.6.3 : (Retour à la page)

```
### 2 dimensions retour interdit (renforcement additif de 0.
    3)
alpha = 0.3
beta = 1

Yrenf = MARri(P, Q, alpha, beta, N, debut)$Mar
Ynonrenf = MARri(P, Q, 0, 1, N, debut)$Mar

gradient = c("red","yellow","green", "lightblue","darkblue")
walk = 1:N
```

```
x1renf = Yrenf[1,]
y1renf = Yrenf[2,]
df1renf <- data.frame(walk, x1renf, y1renf)

x1nonrenf = Ynonrenf[1,]
y1nonrenf = Ynonrenf[2,]
df1nonrenf <- data.frame(walk, x1nonrenf, y1nonrenf)


plot = ggplot(df1renf, aes(x1renf, y1renf, color = 1:N)) +
    geom_path() + labs(color = "Time") + scale_colour_
    gradientn(colors = gradient) + theme_minimal() + geom_
    segment(aes(x = 0, y = 0, xend = K, yend = 0)) + geom_
    segment(aes(x = 0, y = K, xend = K, yend = K)) + geom_
    segment(aes(x = 0, y = 0, xend = 0, yend = K)) + geom_
    segment(aes(x = K, y = 0, xend = K, yend = K))+ labs (
    title = "MARri␣renforc e" ,x="Ouest␣/␣Est" , y = "Sud␣/␣
    Nord")
plot1 = ggplot(df1nonrenf, aes(x1nonrenf, y1nonrenf, color =
     1:N)) + geom_path() + labs(color = "Time") + scale_
    colour_gradientn(colors = gradient) + theme_minimal() +
    geom_segment(aes(x = 0, y = 0, xend = K, yend = 0)) +
    geom_segment(aes(x = 0, y = K, xend = K, yend = K)) +
    geom_segment(aes(x = 0, y = 0, xend = 0, yend = K)) +
    geom_segment(aes(x = K, y = 0, xend = K, yend = K))+ labs
     (title = "MARstop␣renforc e" ,x="Ouest␣/␣Est" , y = "
    Sud␣/␣Nord")
grid.arrange(plot, plot1, ncol=2)
```

Code source de la Figure 2.6.4 : (Retour à la page)

```
### 2 dimensions, arret aux bords (renforcement additif de 0
    .3)
alpha = 0.3
beta = 1

Zrenf = MARstop(P, Q, alpha, beta, N, debut)$Mar
Znonrenf = MARstop(P, Q, 0, 1, N, debut)$Mar

gradient = c("red","yellow","green", "lightblue","darkblue")
N1 = ncol(Zrenf)
N2 = ncol(Znonrenf)
walk1 = 1:N1
walk2 = 1:N2

x2renf = Zrenf[1,]
y2renf = Zrenf[2,]
df2renf <- data.frame(walk1, x2renf, y2renf)

x2nonrenf = Znonrenf[1,]
y2nonrenf = Znonrenf[2,]
df2nonrenf <- data.frame(walk2, x2nonrenf, y2nonrenf)
```

```
plot = ggplot(df2renf, aes(x2renf, y2renf, color = 1:N1)) +
    geom_path() + labs(color = "Time") + scale_colour_
    gradientn(colors = gradient) + theme_minimal() + geom_
    segment(aes(x = 0, y = 0, xend = K, yend = 0)) + geom_
    segment(aes(x = 0, y = K, xend = K, yend = K)) + geom_
    segment(aes(x = 0, y = 0, xend = 0, yend = K)) + geom_
    segment(aes(x = K, y = 0, xend = K, yend = K))+ labs (
    title = "MARstop renforc e" ,x="Ouest / Est" , y = "Sud
    / Nord")
plot1 = ggplot(df2nonrenf, aes(x2nonrenf, y2nonrenf, color =
     1:N2)) + geom_path() + labs(color = "Time") + scale_
    colour_gradientn(colors = gradient) + theme_minimal() +
    geom_segment(aes(x = 0, y = 0, xend = K, yend = 0)) +
    geom_segment(aes(x = 0, y = K, xend = K, yend = K)) +
    geom_segment(aes(x = 0, y = 0, xend = 0, yend = K)) +
    geom_segment(aes(x = K, y = 0, xend = K, yend = K))+ labs
     (title = "MARstop non renforc e" ,x="Ouest / Est" , y =
     "Sud / Nord")
grid.arrange(plot, plot1, ncol=2)
```

Code source de la Figure 2.7.1 : (Retour à la page)

```
### Comparaisons des sensibilites aux renforcements
alpha = 1
beta = 1

Z1 = MAR2d(P, Q, alpha, beta, N, debut)$Mar
Z2 = MARri(P, Q, alpha, beta, N, debut)$Mar
Z3 = MARstop(P, Q, alpha, beta, N, debut)$Mar

gradient = c("red","yellow","green", "lightblue","darkblue")
N1 = ncol(Z1)
N2 = ncol(Z2)
N3 = ncol(Z3)
walk1 = 1:N1
walk2 = 1:N2
walk3 = 1:N3

x1 = Z1[1,]
y1 = Z1[2,]
df1 <- data.frame(walk1, x1, y1)

x2 = Z2[1,]
y2 = Z2[2,]
df2 <- data.frame(walk2, x2, y2)

x3 = Z3[1,]
y3 = Z3[2,]
df3 <- data.frame(walk3, x3, y3)


plot1 = ggplot(df1, aes(x1, y1, color = 1:N1)) + geom_path()
     + labs(color = "Time") + scale_colour_gradientn(colors =
     gradient) + theme_minimal() + geom_segment(aes(x = 0, y
    = 0, xend = K, yend = 0)) + geom_segment(aes(x = 0, y = K
```

```
    , xend = K, yend = K)) + geom_segment(aes(x = 0, y = 0,
    xend = 0, yend = K)) + geom_segment(aes(x = K, y = 0,
    xend = K, yend = K))+ labs (title = "MAR2D" ,x="Ouest␣/␣
    Est" , y = "Sud␣/␣Nord")
plot2 = ggplot(df2, aes(x2, y2, color = 1:N2)) + geom_path()
     + labs(color = "Time") + scale_colour_gradientn(colors =
     gradient) + theme_minimal() + geom_segment(aes(x = 0, y
    = 0, xend = K, yend = 0)) + geom_segment(aes(x = 0, y = K
    , xend = K, yend = K)) + geom_segment(aes(x = 0, y = 0,
    xend = 0, yend = K)) + geom_segment(aes(x = K, y = 0,
    xend = K, yend = K))+ labs (title = "MARri" ,x="Ouest␣/␣
    Est" , y = "Sud␣/␣Nord")
plot3 = ggplot(df3, aes(x3, y3, color = 1:N3)) + geom_path()
     + labs(color = "Time") + scale_colour_gradientn(colors =
     gradient) + theme_minimal() + geom_segment(aes(x = 0, y
    = 0, xend = K, yend = 0)) + geom_segment(aes(x = 0, y = K
    , xend = K, yend = K)) + geom_segment(aes(x = 0, y = 0,
    xend = 0, yend = K)) + geom_segment(aes(x = K, y = 0,
    xend = K, yend = K))+ labs (title = "MARstop" ,x="Ouest␣/
    ␣Est" , y = "Sud␣/␣Nord")
grid.arrange(plot1, plot2, plot3, ncol=3)
```

6.2 Annexes Section Statistique

Code source de la paramétrisation partie Statistiques : (Retour à la page : 25)

```
#### Initialisation pour differents tests ####
K = 400                          #On a une grille de taille
    K*K
K1 = 100
debut = floor(K/2)
debut1 = floor(K1/2)

N = 50000
t = seq(2,10000, by = 1000)
t1 = 10000

n = 50
d = 10

alpha = 1
beta = 1
alpha1 = 0
beta1 = 1.01

Alpha = seq(0, 5, by = 0.5)
Beta = seq(1, 1.2, by = 0.01)

#### Matrices ####
P = matrix(1,nrow=K,ncol=2*K)    #Matrice des mouvements
    horizontaux
for(i in 1:K){
  P[1, i] = 0
  P[K, K+i] = 0
}
Q = matrix(1,nrow=K,ncol=2*K)    #Matrice des mouvements
    verticaux
for(i in 1:K){
  Q[i,1] = 0
  Q[i, 2*K] = 0
}
P2 = matrix(1,nrow=K1,ncol=2*K1)    #Matrice des mouvements
    horizontaux
for(i in 1:K1){
  P2[1, i] = 0
  P2[K1, K1+i] = 0
}
Q2 = matrix(1,nrow=K1,ncol=2*K1)    #Matrice des mouvements
    verticaux
for(i in 1:K1){
  Q2[i,1] = 0
  Q2[i, 2*K1] = 0
}

testP = array(0, dim = c(K1,2*K1,n))
testQ = array(0, dim = c(K1,2*K1,n))
```

Code source de la Figure 3.1.1 : (Retour à la page)

```
N = 10000

Tb1 = rep(NA, length(Alpha))
for(i in 1:length(Alpha)){
  Tb1[i] = mean(replicate(n,dim(MARstop(P2, Q2, Alpha[i],
      beta, N, debut1)$Mar))[2,])
}

Tb2 = rep(NA, length(Beta))
for(i in 1:length(Beta)){
  Tb2[i] = mean(replicate(n,dim(MARstop(P2, Q2, alpha1, Beta
      [i], N, debut1)$Mar))[2,])
}

par(mfrow=c(1,2))
plot(Alpha, Tb1, type = 'l',ylab="Temps␣pour␣toucher␣le␣bord
    ",main="Temps␣pour␣toucher␣le␣bord␣en␣fonction␣du␣alpha")
plot(Beta, Tb2, type = 'l',ylab="Temps␣pour␣toucher␣le␣bord"
    ,main="Temps␣pour␣toucher␣le␣bord␣en␣fonction␣du␣beta")
```

Code source de la Figure 3.1.2 : (Retour à la page)

```
#Valeurs de alpha critiques
N = 50000
Alpha = seq (0,10, by =0.5 )

Tb1 = rep(NA, length(Alpha))
for(i in 1:length(Alpha)){
  Tb1[i] = mean(replicate(n,dim(MARstop(P2, Q2, Alpha[i],
      beta, 10000, debut1)$Mar))[2,])
}

Tb2 = rep(NA, length(Alpha))
for(i in 1:length(Alpha)){
  Tb2[i] = mean(replicate(n,dim(MARstop(P2, Q2, Alpha[i],
      Beta, 50000, debut1)$Mar))[2,])
}

par(mfrow=c(1,2))
plot(Alpha, Tb1, type = 'l',ylab="Temps␣pour␣toucher␣le␣bord
    ",main="Pour␣N=10.000")
plot(Alpha, Tb2, type = 'l',ylab="Temps␣pour␣toucher␣le␣bord
    ",main="Pour␣N=50.000")
```

Code source de la Figure 3.1.3 : (Retour à la page)

```
#Valeurs de alpha critiques -- Tailles grilles
N = 50000
n=10
Alpha = seq (0,10, by = 0.5 )

Tb1 = rep(NA, length(Alpha))
for(i in 1:length(Alpha)){
  Tb1[i] = mean(replicate(n,dim(MARstop(P2, Q2, Alpha[i],
      beta, 10000, debut1)$Mar))[2,])
```

```
}

Tb2 = rep(NA, length(Alpha))
for(i in 1:length(Alpha)){
  Tb2[i] = mean(replicate(n,dim(MARstop(P, Q, Alpha[i], beta
      , 10000, debut)$Mar))[2,])
}

par(mfrow=c(1,2))
plot(Alpha, Tb1, type = 'l',ylab="Temps␣pour␣toucher␣le␣bord
    ",main="Pour␣petite␣grille␣K=100")
plot(Alpha, Tb2, type = 'l',ylab="Temps␣pour␣toucher␣le␣bord
    ",main="Pour␣grande␣grille␣K=400")
```

Code source de la Figure 3.1.4 : (Retour à la page)

```
#### Variation de la taille de la grille ####
#Petite grille
K3 = 150                          #On a une grille de taille K
    *K

P3 = matrix(1,nrow=K3,ncol=2*K3)    #Matrice des mouvements
    horizontaux
for(i in 1:K3){
  P3[1, i] = 0
  P3[K3, K3+i] = 0
}


Q3 = matrix(1,nrow=K3,ncol=2*K3)    #Matrice des mouvements
    verticaux
for(i in 1:K3){
  Q3[i,1] = 0
  Q3[i, 2*K3] = 0
}

#Moyenne -
K4 = 250                          #On a une grille de taille K
    *K

P4 = matrix(1,nrow=K4,ncol=2*K4)    #Matrice des mouvements
    horizontaux
for(i in 1:K4){
  P4[1, i] = 0
  P4[K4, K4+i] = 0
}


Q4 = matrix(1,nrow=K4,ncol=2*K4)    #Matrice des mouvements
    verticaux
for(i in 1:K4){
  Q4[i,1] = 0
  Q4[i, 2*K4] = 0
}
```

```
#Moyenne +
K5 = 500                          #On a une grille de taille K
    *K

P5 = matrix(1,nrow=K5,ncol=2*K5)    #Matrice des mouvements
    horizontaux
for(i in 1:K5){
  P5[1, i] = 0
  P5[K5, K5+i] = 0
}


Q5 = matrix(1,nrow=K5,ncol=2*K5)    #Matrice des mouvements
    verticaux
for(i in 1:K5){
  Q5[i,1] = 0
  Q5[i, 2*K5] = 0
}

#Grande grille
K6 = 800                          #On a une grille de taille K
    *K

P6 = matrix(1,nrow=K6,ncol=2*K6)    #Matrice des mouvements
    horizontaux
for(i in 1:K6){
  P6[1, i] = 0
  P6[K6, K6+i] = 0
}


Q6 = matrix(1,nrow=K6,ncol=2*K6)    #Matrice des mouvements
    verticaux
for(i in 1:K6){
  Q6[i,1] = 0
  Q6[i, 2*K6] = 0
}

debut3 = floor(K3/2)
debut4 = floor(K4/2)
debut5 = floor(K5/2)
debut6 = floor(K6/2)


T7 = MAR2d(P3, Q3, alpha, beta, N, debut3)$Mar
T8 = MAR2d(P4, Q4, alpha, beta, N, debut4)$Mar
T9 = MAR2d(P5, Q5, alpha, beta, N, debut5)$Mar
T10 = MAR2d(P6, Q6, alpha, beta, N, debut6)$Mar

gradient = c("red","yellow","green", "lightblue","darkblue")
walk7 = 1:(ncol(T7))
walk8 = 1:(ncol(T8))
walk9 = 1:(ncol(T9))
walk10 = 1:(ncol(T10))
```

```
u7 = T7[1,]
v7 = T7[2,]
df7 <- data.frame(walk7, u7, v7)

u8 = T8[1,]
v8 = T8[2,]
df8 <- data.frame(walk8, u8, v8)

u9 = T9[1,]
v9 = T9[2,]
df9 <- data.frame(walk9, u9, v9)

u10 = T10[1,]
v10 = T10[2,]
df10 <- data.frame(walk10, u10, v10)


plot7 = ggplot(df7, aes(u7, v7, color = 1:ncol(T7))) + geom_
    path() + labs(color = "Time") + scale_colour_gradientn(
    colors = gradient) + theme_minimal() + geom_segment(aes(x
     = 0, y = 0, xend = K3, yend = 0)) + geom_segment(aes(x =
     0, y = K3, xend = K3, yend = K3)) + geom_segment(aes(x =
     0, y = 0, xend = 0, yend = K3)) + geom_segment(aes(x = K
    3, y = 0, xend = K3, yend = K3))+ labs (title = "Petite␣
    grille" ,x="Ouest␣/␣Est" , y = "Sud␣/␣Nord")
plot8 = ggplot(df8, aes(u8, v8, color = 1:ncol(T8))) + geom_
    path() + labs(color = "Time") + scale_colour_gradientn(
    colors = gradient) + theme_minimal() + geom_segment(aes(x
     = 0, y = 0, xend = K4, yend = 0)) + geom_segment(aes(x =
     0, y = K4, xend = K4, yend = K4)) + geom_segment(aes(x =
     0, y = 0, xend = 0, yend = K4)) + geom_segment(aes(x = K
    4, y = 0, xend = K4, yend = K4))+ labs (title = "Moyenne␣
    -" ,x="Ouest␣/␣Est" , y = "Sud␣/␣Nord")
plot9 = ggplot(df9, aes(u9, v9, color = 1:ncol(T9))) + geom_
    path() + labs(color = "Time") + scale_colour_gradientn(
    colors = gradient) + theme_minimal() + geom_segment(aes(x
     = 0, y = 0, xend = K5, yend = 0)) + geom_segment(aes(x =
     0, y = K5, xend = K5, yend = K5)) + geom_segment(aes(x =
     0, y = 0, xend = 0, yend = K5)) + geom_segment(aes(x = K
    5, y = 0, xend = K5, yend = K5))+ labs (title = "Moyenne␣
    +" ,x="Ouest␣/␣Est" , y = "Sud␣/␣Nord")
plot10 = ggplot(df10, aes(u10, v10, color = 1:ncol(T10))) +
    geom_path() + labs(color = "Time") + scale_colour_
    gradientn(colors = gradient) + theme_minimal() + geom_
    segment(aes(x = 0, y = 0, xend = K6, yend = 0)) + geom_
    segment(aes(x = 0, y = K6, xend = K6, yend = K6)) + geom_
    segment(aes(x = 0, y = 0, xend = 0, yend = K6)) + geom_
    segment(aes(x = K6, y = 0, xend = K6, yend = K6))+ labs (
    title = "Grande␣grille" ,x="Ouest␣/␣Est" , y = "Sud␣/␣
    Nord")
grid.arrange(plot7, plot8, plot9, plot10, ncol=2)
```

Code source de la Figure 3.2.1 : (Retour à la page)

Faire varier d et relancer

```
par(mfrow = c(3,2))
n=50
d=10

#On cherche le nombre d'iteration necessaire pour depasser
    une distance de d
#MAR2d additive

Ta1 = rep(0, length(Alpha))
for(k in 1:length(Alpha)){
  Tatemp = rep(0,n)
  X = replicate(n, MAR2d(P, Q, Alpha[k], beta, t1, debut)$
      Mar)
  for(j in 1:n){
    i = 1
    while((abs(X[1,i,j] - debut) + abs(X[2,i,j] - debut) < d
        ) & (i < t1)){
      i = i + 1
    }
    if(i >= t1){
      Tatemp[j] = 0
    }
    else{
      Tatemp[j] = i
    }
  }
  Ta1[k] = mean(Tatemp)
}
plot(Alpha, Ta1, type = 'l',ylab="Temps d'arr t",main="MAR2
    d additive et d = 10")


#MARri additive

Ta2 = rep(0, length(Alpha))
for(k in 1:length(Alpha)){
  Tatemp = rep(0,n)
  X = replicate(n, MARri(P, Q, Alpha[k], beta, t1, debut)$
      Mar)
  for(j in 1:n){
    i = 1
    while((abs(X[1,i,j] - debut) + abs(X[2,i,j] - debut) < d
        ) & (i < t1)){
      i = i + 1
    }
    if(i >= t1){
      Tatemp[j] = 0
    }
    else{
      Tatemp[j] = i
    }
  }
  Ta2[k] = mean(Tatemp)
```

```
}
plot(Alpha, Ta2, type = 'l',ylab="Temps d'arr t",main="
    MARri additive et d = 10")
```

Code source de la Figure 3.3.1 : (Retour à la page)

```
#### Distance du point de depart apres N iterations ####

par(mfrow = c(2,3))

#MAR2d additive, variation des iterations

dist1 = rep(NA, length(t))
for(i in 1:length(t)){
  X = replicate(n, MAR2d(P, Q, alpha, beta, t[i], debut)$Mar
      )
  distance = sqrt((abs((X[1, t[i],] - debut)))**2 + (abs((X[
      2, t[i],] - debut)))**2)
  dist1[i] = mean(distance)
}
plot(t, dist1, type = 'l',xlab="Nombre d'it rations",ylab="
    Distance",main="MAR2D")


#MAR2d additive, variation du renforcement

dist2 = rep(NA, length(Alpha))
for(i in 1:length(Alpha)){
  X = replicate(n, MAR2d(P, Q, Alpha[i], beta, t1, debut)$
      Mar)
  distance = sqrt((abs((X[1, t1,] - debut)))**2 + (abs((X[2,
       t1,] - debut)))**2)
  dist2[i] = mean(distance)
}
plot(Alpha, dist2, type = 'l',xlab="Alpha",ylab="Distance",
    main="MAR2D")

#MAR2d non renforc , variation des iterations

dist3 = rep(NA, length(t))
for(i in 1:length(t)){
  X = replicate(n, MAR2d(P, Q, alpha1, beta, t[i], debut)$
      Mar)
  distance = sqrt((abs((X[1, t[i],] - debut)))**2 + (abs((X[
      2, t[i],] - debut)))**2)
  dist3[i] = mean(distance)
}
plot(t, dist3, type = 'l',xlab="Nombre d'it rations",ylab="
    Distance",main="Marche Al atoire 2d")

#MARri additive, variation des iterations

dist4 = rep(NA, length(t))
for(i in 1:length(t)){
  X = replicate(n, MARri(P, Q, alpha, beta, t[i], debut)$Mar
```

```
    )
  distance = sqrt((abs((X[1, t[i],] - debut)))**2 + (abs((X[
      2, t[i],] - debut)))**2)
  dist4[i] = mean(distance)
}
plot(t, dist4, type = 'l',xlab="Nombre␣d'it rations",ylab="
    Distance",main="MARri")

#MARri additive, variation du renforcement

dist5 = rep(NA, length(Alpha))
for(i in 1:length(Alpha)){
  X = replicate(n, MARri(P, Q, Alpha[i], beta, t1, debut)$
      Mar)
  distance = sqrt((abs((X[1, t1,] - debut)))**2 + (abs((X[2,
      t1,] - debut)))**2)
  dist5[i] = mean(distance)
}
plot(Alpha, dist5, type = 'l',xlab="Alpha",ylab="Distance",
    main="MARri")

#MARri non renforc , variation des iterations

dist6 = rep(NA, length(t))
for(i in 1:length(t)){
  X = replicate(n, MAR2d(P, Q, alpha1, beta, t[i], debut)$
      Mar)
  distance = sqrt((abs((X[1, t[i],] - debut)))**2 + (abs((X[
      2, t[i],] - debut)))**2)
  dist6[i] = mean(distance)
}
plot(t, dist6, type = 'l',xlab="Nombre␣d'it rations",ylab="
    Distance",main="Marche␣Al atoire␣RI")
```

Code source de la Figure 3.4.1 : (Retour à la page)

```
#### Distance max ####

par(mfrow = c(2,3))

#MAR2d additive, variation des iterations

MeanDistMax = rep(0, length(t))
for(l in 1:length(t)){
  X6 = replicate(n, MAR2d(P, Q, alpha, beta, t[l], debut)$
      Mar)
  DistMax = rep(0,n)
  for(i in 1:n){
    for(j in 1:t[l]){
      Dist = sqrt((abs((X6[1, j, i] - debut)))**2 + (abs((X6
          [2, j, i] - debut)))**2)
      if(Dist > DistMax[i]){
        DistMax[i] = Dist
        }
      }
```

```
    }
  MeanDistMax[l] = mean(DistMax)
  }

plot(t, MeanDistMax, type = 'l',xlab="Nombre d'it rations",
    ylab="Distance max",main="MAR2D")

#MAR2d additive, variation du renforcement

MeanDistMax = rep(0, length(Alpha))
for(l in 1:length(Alpha)){
  X6 = replicate(n, MAR2d(P, Q, Alpha[l], beta1, t1, debut)$
      Mar)
  DistMax = rep(0,n)
  for(i in 1:n){
    for(j in 1:t1){
      Dist = sqrt((abs((X6[1, j, i] - debut)))**2 + (abs((X6
          [2, j, i] - debut)))**2)
      if(Dist > DistMax[i]){
        DistMax[i] = Dist
      }
    }
  }
  MeanDistMax[l] = mean(DistMax)
}

plot(Alpha, MeanDistMax, type = 'l',ylab="Distance max",main
    ="MAR2D")

#MAR2d non renforc , variation des iterations

MeanDistMax = rep(0, length(t))
for(l in 1:length(t)){
  X6 = replicate(n, MAR2d(P, Q, alpha1, beta, t[l], debut)$
      Mar)
  DistMax = rep(0,n)
  for(i in 1:n){
    for(j in 1:t[l]){
      Dist = sqrt((abs((X6[1, j, i] - debut)))**2 + (abs((X6
          [2, j, i] - debut)))**2)
      if(Dist > DistMax[i]){
        DistMax[i] = Dist
      }
    }
  }
  MeanDistMax[l] = mean(DistMax)
}

plot(t, MeanDistMax, type = 'l',xlab="Nombre d'it rations",
    ylab="Distance max",main="Marche Al atoire")

#MARri additive, variation des iterations

MeanDistMax = rep(0, length(t))
```

```
for(l in 1:length(t)){
  X6 = replicate(n, MARri(P, Q, alpha, beta, t[l], debut)$
      Mar)
  DistMax = rep(0,n)
  for(i in 1:n){
    for(j in 1:t[l]){
      Dist = sqrt((abs((X6[1, j, i] - debut)))**2 + (abs((X6
          [2, j, i] - debut)))**2)
      if(Dist > DistMax[i]){
        DistMax[i] = Dist
      }
    }
  }
  MeanDistMax[l] = mean(DistMax)
}

plot(t, MeanDistMax, type = 'l',xlab="Nombre d'it rations",
    ylab="Distance max",main="MARri")

#MARri additive, variation du renforcement

MeanDistMax = rep(0, length(Alpha))
for(l in 1:length(Alpha)){
  X6 = replicate(n, MARri(P, Q, Alpha[l], beta1, t1, debut)$
      Mar)
  DistMax = rep(0,n)
  for(i in 1:n){
    for(j in 1:t1){
      Dist = sqrt((abs((X6[1, j, i] - debut)))**2 + (abs((X6
          [2, j, i] - debut)))**2)
      if(Dist > DistMax[i]){
        DistMax[i] = Dist
      }
    }
  }
  MeanDistMax[l] = mean(DistMax)
}

plot(Alpha, MeanDistMax, type = 'l',ylab="Distance max",main
    ="MARri")

#MARri non renforc , variation des iterations

MeanDistMax = rep(0, length(t))
for(l in 1:length(t)){
  X6 = replicate(n, MARri(P, Q, alpha1, beta, t[l], debut)$
      Mar)
  DistMax = rep(0,n)
  for(i in 1:n){
    for(j in 1:t[l]){
      Dist = sqrt((abs((X6[1, j, i] - debut)))**2 + (abs((X6
          [2, j, i] - debut)))**2)
      if(Dist > DistMax[i]){
        DistMax[i] = Dist
```

```
      }
    }
  }
  MeanDistMax[l] = mean(DistMax)
}

plot(t, MeanDistMax, type = 'l',xlab="Nombre d'it rations",
    ylab="Distance max",main="MARri")
```

Code source de la Figure 3.5.1 : (Retour à la page)

```
x = 5   #Les x sommets/arretes les plus visit s

#On choisit quel type de MAR on fait ici (pour les arretes
    les plus emprunt es) :


for(i in 1:n){
  A = MAR2d(P2, Q2, alpha, beta, 100, debut1)
  ArreteP[1:K1, 1:(2*K1),i] = A$ArreteP
  ArreteQ[1:K1, 1:(2*K1),i] = A$ArreteQ
}

#On commence par regarder les indices des aretes les plus
    empruntees pour chaque simulation

idP = matrix(0, nrow = n, ncol = 2)
idQ = matrix(0, nrow = n, ncol = 2)


for(i in 1:n){
  idp = which.max(ArreteP[1:K1, 1:(2*K1), i])
  if((idp / K1) - (floor(idp / K1)) == 0){
    idP[i, 1] = K1
    idP[i, 2] = idp / K1
  } else{
    idP[i, 1] = ((idp / K1) - floor(idp / K1))*K1
    idP[i, 2] = floor(idp / K1) + 1
  }
  idq = which.max(ArreteQ[1:K1, 1:(2*K1), i])
  if((idq / K1) - (floor(idq / K1)) == 0){
    idQ[i, 1] = K1
    idQ[i, 2] = idq / K1
  } else{
    idQ[i, 1] = ((idq / K1) - floor(idq / K1))*K1
    idQ[i, 2] = floor(idq / K1) + 1
  }
}

idP     #On obtient les sommets par lesquels partent chaque
    arr te. Si la deuxi me colone est > K1, on va a droite,
     sinon on va a gauche
idQ     #On obtient les sommets par lesquels partent chaque
    arr te. Si la deuxi me colone est > K1, on monte, sinon
     on descend
```

```
#On regarde maintenant les aretes les plus empruntees toutes
     simulation confondu. Pour cela on commence par sommer
    les matrices de chaque simulations

arreteP = matrix(0, nrow = K1, ncol = 2*K1)
arreteQ = matrix(0, nrow = K1, ncol = 2*K1)

for(i in 1:K1){
  for(j in 1:K1){
    arreteP[i, j] = sum(ArreteP[i, j, 1:n])
    arreteP[i, j + K1] = sum(ArreteP[i, j + K1, 1:n])
    arreteQ[i, j] = sum(ArreteQ[i, j, 1:n])
    arreteQ[i, j + K1] = sum(ArreteQ[i, j + K1, 1:n])
  }
}

#On cherche maintenant les x aretes les plus empruntees (
    horizontalement et verticalement)
#On va d'abord copie les matrices crees precedemment et a
    chaque fois qu'on obtient l'indice du max, on remplace la
     valeur du max par 0 dans la copie
#Pour obtenir au final les x aretes les plus empruntees

CarreteP = arreteP
CarreteQ = arreteQ

id1P = matrix(0, nrow = x, ncol = 4)
id1Q = matrix(0, nrow = x, ncol = 4)

for(i in 1:x){
  id1p = which.max(CarreteP)
  id1P[i, 1] = (id1p - 1)%%nrow(CarreteP) + 1
  id1P[i, 2] = ceiling(id1p/nrow(CarreteP))
  CarreteP[id1P[i, 1], id1P[i, 2]] = 0

  id1q = which.max(CarreteQ)
  id1Q[i, 1] = (id1q - 1)%%nrow(CarreteQ) + 1
  id1Q[i, 2] = ceiling(id1q/nrow(CarreteQ))
  CarreteQ[id1Q[i, 1], id1Q[i, 2]] = 0

  if(id1P[i, 2] > K1){
    id1P[i, 2] = id1P[i, 2] - K1
    id1P[i, 3] = 2
    id1P[i, 4] = arreteP[id1P[i,1], id1P[i,2] + K1]
  }
  else{
    id1P[i, 3] = 1
    id1P[i, 4] = arreteP[id1P[i,1], id1P[i,2]]
  }

  if(id1Q[i, 2] > K1){
    id1Q[i, 2] = id1Q[i, 2] - K1
    id1Q[i, 3] = 2
```

```
    id1Q[i, 4] = arreteQ[id1Q[i,1], id1Q[i,2] + K1]
  }
  else{
    id1Q[i, 3] = 1
    id1Q[i, 4] = arreteQ[id1Q[i,1], id1Q[i,2]]
  }
}

#On remarque que les aretes les plus empruntees sont proches
     les unes des autres
id1P
id1Q
```

Code source de la Figure 3.6.1 : (Retour à la page)

```
#### Sommets les plus visit s ####


A = replicate(3 ,MAR2d(P, Q, alpha, beta, 200, debut)$Sommet
    )

Sommet = matrix(0, nrow = x, ncol = 2)
Temp = matrix(0, nrow = K, ncol = K)

for(j in 1:K){
  for(k in 1:K){
    Temp[j,k] = sum(A[j, k, 1:3])
  }
}

for(l in 1:x){
  idMax = which.max(Temp)
  Sommet[l, 1] = (idMax - 1)%%nrow(Temp) + 1
  Sommet[l, 2] = ceiling(idMax/nrow(Temp))
  Temp[Sommet[l, 1], Sommet[l, 2]] = 0
}

Sommet
```

Code source de la Figure 3.7.1 : (Retour à la page)

```
#### Nombres de sommet visit  ####
par(mfrow=c(3,2))

#Mar2d additive, variation des iterations (Attention,  a
    dure environ 11 secondes pour toute la boucle for(i))

MeanNbS = rep(0, length(t))

for(i in 1:length(t)){
  A = replicate(n,MAR2d(P, Q, alpha, beta, t[i], debut)$
      Sommet)
  MeanNbS[i] = length(A[A > 0]) / n
}

PS = MeanNbS / (K*K)
```

```
plot(t, MeanNbS, type ='l',xlab="Nombre d'it rations",ylab=
    "Nombre moyen de points visit s",main="MAR2D")
plot(t, PS, type ='l',xlab="Nombre d'it rations",ylab="
    Ratio",main="MAR2D")


#Mar2d additive, variation du Alpha

MeanNbS = rep(0, length(Alpha))

for(i in 1:length(Alpha)){
  A = replicate(n,MAR2d(P, Q, Alpha[i], beta, t1, debut)$
      Sommet)
  MeanNbS[i] = length(A[A > 0]) / n
}
PS = MeanNbS / (K*K)
plot(Alpha, MeanNbS, type ='l',xlab="Alpha",ylab="Nombre
    moyen de points visit s",main="MAR2D")
plot(Alpha, PS, type ='l',xlab="Alpha",ylab="Ratio",main="
    MAR2D")


#Mar2d non renforc e, variation des it rations (Attention,
      a  dure, je pense, 10 min environ)


MeanNbS = rep(0, length(t))

for(i in 1:length(t)){
  A = replicate(n,MAR2d(P, Q, alpha1, beta, t[i], debut)$
      Sommet)
  MeanNbS[i] = length(A[A > 0]) / n
}

PS = MeanNbS / (K*K)

plot(t, MeanNbS, type ='l',xlab="Nombre d'it rations",ylab=
    "Nombre moyen de points visit s",main="MA 2D non
    renforc e")
plot(t, PS, type ='l',xlab="Nombre d'it rations",ylab="
    Ratio",main="MA 2D non renforc e")
```

Code source de la Figure 3.7.2 : (Retour à la page)

```
#Marri additive, variation des iterations (Attention,  a
    dure environ 10 min)
par(mfrow=c(3,2))

MeanNbS = rep(0, length(t))

for(i in 1:length(t)){
  A = replicate(n,MARri(P, Q, alpha, beta, t[i], debut)$
      Sommet)
  MeanNbS[i] = length(A[A > 0]) / n
```

```
}

PS = MeanNbS / (K*K)

plot(t, MeanNbS, type ='l',xlab="Nombre d'it rations",ylab=
    "Nombre moyen de points visit s",main="MARri")
plot(t, PS, type ='l',xlab="Nombre d'it rations",ylab="
    Ratio",main="MARri")


#Marri additive, variation du Alpha (Attention,  a  dure
    bien plus longtemps)


MeanNbS = rep(0, length(Alpha))

for(i in 1:length(Alpha)){
  A = replicate(n,MARri(P, Q, Alpha[i], beta, t1, debut)$
      Sommet)
  MeanNbS[i] = length(A[A > 0]) / n
}

PS = MeanNbS / (K*K)

plot(Alpha, MeanNbS, type ='l',xlab="Alpha",ylab="Nombre
    moyen de points visit s",main="MARri")
plot(Alpha, PS, type ='l',xlab="Alpha",ylab="Ratio",main="
    MARri")


#Mar2d non renforc e, variation des it rations (Attention,
      a  dure)


MeanNbS = rep(0, length(t))

for(i in 1:length(t)){
  A = replicate(n,MARri(P, Q, alpha1, beta, t[i], debut)$
      Sommet)
  MeanNbS[i] = length(A[A > 0]) / n
}

PS = MeanNbS / (K*K)

plot(t, MeanNbS, type ='l',xlab="Nombre d'it rations",ylab=
    "Nombre moyen de points visit s",main="MARri non
    renforc e")
plot(t, PS, type ='l',xlab="Nombre d'it rations",ylab="
    Ratio",main="MARri non renforc e")
```

6.3 Annexes section Populations

Code source de la fonction MAR2pop : (Retour à la page)

```
MAR2pop = function(P, Q, R, S, alpha, beta, gamma, delta, N,
    xdebut, ydebut, xdebut1, ydebut1){
  Mar1 = matrix(0, nrow = 2, ncol = N)
  Mar2 = matrix(0, nrow = 2, ncol = N)
  Mar1[1,1] = xdebut
  Mar2[1,1] = xdebut1
  Mar1[2,1] = ydebut
  Mar2[2,1] = ydebut1
  K = nrow(P)
  for(i in 1:(N-1)){
    W = runif(1)

    U = runif(1)
                                                        #On
        tire un nombre aleatoire entre 0 et 1
    a = P[Mar1[1,i], Mar1[2,i]]
                                                #On cr er des
        variables pour simplifier l' criture
    b = P[Mar1[1,i], Mar1[2,i] + K]
    c = Q[Mar1[1,i], Mar1[2,i] + K]
    d = Q[Mar1[1,i], Mar1[2,i]]

    V = runif(1)
    e = R[Mar2[1,i], Mar2[2,i]]
                                                #On cr er des
        variables pour simplifier l' criture
    f = R[Mar2[1,i], Mar2[2,i] + K]
    g = S[Mar2[1,i], Mar2[2,i] + K]
    h = S[Mar2[1,i], Mar2[2,i]]

    if((a > 10**300) | (b > 10**300) | (c > 10**300) | (d >
        10**300)){
      warning('renforcement␣trop␣important')
      return(list(Mar1 = matrix(Mar1[Mar1 > 0], nrow = 2,
          byrow = FALSE), Mar2 = matrix(Mar2[Mar2 > 0], nrow
          = 2, byrow = FALSE), P = P, Q = Q, R = R, S = S, i
          = i))
    }

    if((e > 10**300) | (f > 10**300) | (g > 10**300) | (h >
        10**300)){
      warning('renforcement␣trop␣important')
      return(list(Mar1 = matrix(Mar1[Mar1 > 0], nrow = 2,
          byrow = FALSE), Mar2 = matrix(Mar2[Mar2 > 0], nrow
          = 2, byrow = FALSE), P = P, Q = Q, R = R, S = S, i
          = i))
    }

    if(W < 0.5){
      if(U < (a / (a + b + c + d))){        #Si U est plus
          petit que la proba d'aller a gauche (Poid d'aller a
```

```
    gauche diviser par la somme d'aller a droite, a
    gauche, devant et derriere)
  P[Mar1[1,i], Mar1[2,i]] = beta * P[Mar1[1,i], Mar1[2
      ,i]] + alpha                              #On va a
      gauche et on ajoute delta
  R[Mar1[1,i], Mar1[2,i]] = max(0, delta * R[Mar1[1,i
      ], Mar1[2,i]] - gamma)
  Mar1[1, i + 1] = Mar1[1, i] - 1
  Mar1[2, i + 1] = Mar1[2, i]
}
else if(U < ((a + b) / (a + b + c + d))){    #Si U est
     entre les probas d'aller a gauche et d'aller a
    droite ou a gauche (Poid d'aller a gauche + a
    droite diviser par la somme d'aller a droite, a
    gauche, devant et derriere, car si arrive ici c'est
     que U est plus grand que la proba d'aller a gauche
    )
  P[Mar1[1,i], Mar1[2,i] + K] = beta * P[Mar1[1,i],
      Mar1[2,i] + K] + alpha
                                                    #On
       va a droite et on enleve delta
  R[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * R[Mar1[
      1,i], Mar1[2,i] + K] - gamma)
  Mar1[1, i + 1] = Mar1[1, i] + 1
  Mar1[2, i + 1] = Mar1[2, i]
}
else if(U < ((a + b + c) / (a + b + c + d))){       #
    Si U est entre les probas d'aller a gauche ou a
    droite et d'aller a droite ou a gauche ou devant (
    Poid d'aller a gauche + a droite + devant diviser
    par la somme d'aller a droite, a gauche, devant et
    derriere, car si on arrive ici c'est que U est plus
     grand que la proba d'aller a gauche ou a droite)
  Q[Mar1[1,i], Mar1[2,i] + K] = beta * Q[Mar1[1,i],
      Mar1[2,i] + K] + alpha

      #On va devant et on ajoute delta
  S[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * S[Mar1[
      1,i], Mar1[2,i] + K] - gamma)
  Mar1[1, i + 1] = Mar1[1, i]
  Mar1[2, i + 1] = Mar1[2, i] + 1
}
else if((U != 1) | (Mar1[2,i] != 1)){

    #Si U est different de 1 ou que nous ne sommes pas
    tout en bas
  Q[Mar1[1,i], Mar1[2,i]] = beta * Q[Mar1[1,i], Mar1[2
      ,i]] + alpha

      #On va derriere et on enleve delta
  S[Mar1[1,i], Mar1[2,i]] = max(0, delta * S[Mar1[1,i
      ], Mar1[2,i]] - gamma)
  Mar1[1, i + 1] = Mar1[1, i]
  Mar1[2, i + 1] = Mar1[2, i] - 1
```

```
}
else{

    #Si U est egal a 1 et que nous sommes tout en bas,
  Mar1[1, i + 1] = Mar1[1, i]
                                      #On choisit de
      rester sur place sans changer les poids pour ne
      pas fausser l'aleatoire (Juste pour cette
      iteration).
  Mar1[2, i + 1] = Mar1[2, i]
                                      #Ce cas n'a
      quasiment aucune chance d'arriver mais il faut
      tout de meme l'anticiper
}

if(V < (e / (e + f + g + h))){        #Si U est plus
    petit que la proba d'aller a gauche (Poid d'aller a
     gauche diviser par la somme d'aller a droite, a
    gauche, devant et derriere)
  R[Mar2[1,i], Mar2[2,i]] = beta * R[Mar2[1,i], Mar2[2
      ,i]] + alpha                              #On va a
      gauche et on ajoute delta
  P[Mar2[1,i], Mar2[2,i]] = max(0, delta * P[Mar2[1,i
      ], Mar2[2,i]] - gamma)
  Mar2[1, i + 1] = Mar2[1, i] - 1
  Mar2[2, i + 1] = Mar2[2, i]
}
else if(V < ((e + f) / (e + f + g + h))){    #Si U est
     entre les probas d'aller a gauche et d'aller a
    droite ou a gauche (Poid d'aller a gauche + a
    droite diviser par la somme d'aller a droite, a
    gauche, devant et derriere, car si arrive ici c'est
     que U est plus grand que la proba d'aller a gauche
    )
  R[Mar2[1,i], Mar2[2,i] + K] = beta * R[Mar2[1,i],
      Mar2[2,i] + K] + alpha
                                                    #On
       va a droite et on enleve delta
  P[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * P[Mar2[
      1,i],  Mar2[2,i] + K] - gamma)
  Mar2[1, i + 1] = Mar2[1, i] + 1
  Mar2[2, i + 1] = Mar2[2, i]
}
else if(V < ((e + f + g) / (e + f + g + h))){       #
    Si U est entre les probas d'aller a gauche ou a
    droite et d'aller a droite ou a gauche ou devant (
    Poid d'aller a gauche + a droite + devant diviser
    par la somme d'aller a droite, a gauche, devant et
    derriere, car si on arrive ici c'est que U est plus
     grand que la proba d'aller a gauche ou a droite)
  S[Mar2[1,i], Mar2[2,i] + K] = beta * S[Mar2[1,i],
      Mar2[2,i] + K] + alpha

      #On va devant et on ajoute delta
```

```
    Q[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * Q[Mar2[
        1,i], Mar2[2,i] + K] - gamma)
    Mar2[1, i + 1] = Mar2[1, i]
    Mar2[2, i + 1] = Mar2[2, i] + 1
  }
  else if((V != 1) | (Mar2[2,i] != 1)){

    #Si U est different de 1 ou que nous ne sommes pas
    tout en bas
    S[Mar2[1,i], Mar2[2,i]] = beta * S[Mar2[1,i], Mar2[2
        ,i]] + alpha

    #On va derriere et on enleve delta
    Q[Mar2[1,i], Mar2[2,i]] = max(0, delta * Q[Mar2[1,i
        ], Mar2[2,i]] - gamma)
    Mar2[1, i + 1] = Mar2[1, i]
    Mar2[2, i + 1] = Mar2[2, i] - 1
  }
  else{

    #Si U est egal a 1 et que nous sommes tout en bas,
    Mar2[1, i + 1] = Mar2[1, i]
                                      #On choisit de
        rester sur place sans changer les poids pour ne
        pas fausser l'aleatoire (Juste pour cette
        iteration).
    Mar2[2, i + 1] = Mar2[2, i]
                                      #Ce cas n'a
        quasiment aucune chance d'arriver mais il faut
        tout de meme l'anticiper
  }
} else{
    if(V < (e / (e + f + g + h))){        #Si U est plus
         petit que la proba d'aller a gauche (Poid d'
        aller a gauche diviser par la somme d'aller a
        droite, a gauche, devant et derriere)
      R[Mar2[1,i], Mar2[2,i]] = beta * R[Mar2[1,i], Mar2
          [2,i]] + alpha                          #On va
           a gauche et on ajoute delta
      P[Mar2[1,i], Mar2[2,i]] = max(0, delta * P[Mar2[1,
          i], Mar2[2,i]] - gamma)
      Mar2[1, i + 1] = Mar2[1, i] - 1
      Mar2[2, i + 1] = Mar2[2, i]
    }
    else if(V < ((e + f) / (e + f + g + h))){    #Si U
        est entre les probas d'aller a gauche et d'aller
        a droite ou a gauche (Poid d'aller a gauche + a
        droite diviser par la somme d'aller a droite, a
        gauche, devant et derriere, car si arrive ici c'
        est que U est plus grand que la proba d'aller a
        gauche)
      R[Mar2[1,i], Mar2[2,i] + K] = beta * R[Mar2[1,i],
          Mar2[2,i] + K] + alpha
                                                      #
```

```
      On va a droite et on enleve delta
  P[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * P[Mar
      2[1,i],  Mar2[2,i] + K] - gamma)
  Mar2[1, i + 1] = Mar2[1, i] + 1
  Mar2[2, i + 1] = Mar2[2, i]
}
else if(V < ((e + f + g) / (e + f + g + h))){
    #Si U est entre les probas d'aller a gauche ou a
    droite et d'aller a droite ou a gauche ou devant
    (Poid d'aller a gauche + a droite + devant
    diviser par la somme d'aller a droite, a gauche,
    devant et derriere, car si on arrive ici c'est
    que U est plus grand que la proba d'aller a
    gauche ou a droite)
  S[Mar2[1,i], Mar2[2,i] + K] = beta * S[Mar2[1,i],
      Mar2[2,i] + K] + alpha

      #On va devant et on ajoute delta
  Q[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * Q[Mar
      2[1,i], Mar2[2,i] + K] - gamma)
  Mar2[1, i + 1] = Mar2[1, i]
  Mar2[2, i + 1] = Mar2[2, i] + 1
}
else if((V != 1) | (Mar2[2,i] != 1)){

    #Si U est different de 1 ou que nous ne sommes
    pas tout en bas
  S[Mar2[1,i], Mar2[2,i]] = beta * S[Mar2[1,i], Mar2
      [2,i]] + alpha

      #On va derriere et on enleve delta
  Q[Mar2[1,i], Mar2[2,i]] = max(0, delta * Q[Mar2[1,
      i], Mar2[2,i]] - gamma)
  Mar2[1, i + 1] = Mar2[1, i]
  Mar2[2, i + 1] = Mar2[2, i] - 1
}
else{

    #Si U est egal a 1 et que nous sommes tout en bas
    ,
  Mar2[1, i + 1] = Mar2[1, i]
                                      #On choisit de
      rester sur place sans changer les poids pour ne
       pas fausser l'aleatoire (Juste pour cette
      iteration).
  Mar2[2, i + 1] = Mar2[2, i]
                                      #Ce cas n'a
      quasiment aucune chance d'arriver mais il faut
      tout de meme l'anticiper
}

if(U < (a / (a + b + c + d))){        #Si U est plus
     petit que la proba d'aller a gauche (Poid d'
    aller a gauche diviser par la somme d'aller a
```

```
    droite, a gauche, devant et derriere)
  P[Mar1[1,i], Mar1[2,i]] = beta * P[Mar1[1,i], Mar1
      [2,i]] + alpha                              #On va
       a gauche et on ajoute delta
  R[Mar1[1,i], Mar1[2,i]] = max(0, delta * R[Mar1[1,
      i], Mar1[2,i]] - gamma)
  Mar1[1, i + 1] = Mar1[1, i] - 1
  Mar1[2, i + 1] = Mar1[2, i]
}
else if(U < ((a + b) / (a + b + c + d))){     #Si U
    est entre les probas d'aller a gauche et d'aller
    a droite ou a gauche (Poid d'aller a gauche + a
    droite diviser par la somme d'aller a droite, a
    gauche, devant et derriere, car si arrive ici c'
    est que U est plus grand que la proba d'aller a
    gauche)
  P[Mar1[1,i], Mar1[2,i] + K] = beta * P[Mar1[1,i],
      Mar1[2,i] + K] + alpha
                                                    #
      On va a droite et on enleve delta
  R[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * R[Mar
      1[1,i], Mar1[2,i] + K] - gamma)
  Mar1[1, i + 1] = Mar1[1, i] + 1
  Mar1[2, i + 1] = Mar1[2, i]
}
else if(U < ((a + b + c) / (a + b + c + d))){
    #Si U est entre les probas d'aller a gauche ou a
    droite et d'aller a droite ou a gauche ou devant
    (Poid d'aller a gauche + a droite + devant
    diviser par la somme d'aller a droite, a gauche,
    devant et derriere, car si on arrive ici c'est
    que U est plus grand que la proba d'aller a
    gauche ou a droite)
  Q[Mar1[1,i], Mar1[2,i] + K] = beta * Q[Mar1[1,i],
      Mar1[2,i] + K] + alpha

      #On va devant et on ajoute delta
  S[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * S[Mar
      1[1,i], Mar1[2,i] + K] - gamma)
  Mar1[1, i + 1] = Mar1[1, i]
  Mar1[2, i + 1] = Mar1[2, i] + 1
}
else if((U != 1) | (Mar1[2,i] != 1)){

    #Si U est different de 1 ou que nous ne sommes
    pas tout en bas
  Q[Mar1[1,i], Mar1[2,i]] = beta * Q[Mar1[1,i], Mar1
      [2,i]] + alpha

      #On va derriere et on enleve delta
  S[Mar1[1,i], Mar1[2,i]] = max(0, delta * S[Mar1[1,
      i], Mar1[2,i]] - gamma)
  Mar1[1, i + 1] = Mar1[1, i]
  Mar1[2, i + 1] = Mar1[2, i] - 1
```

```
        }
        else{

          #Si U est egal a 1 et que nous sommes tout en bas
          ,
          Mar1[1, i + 1] = Mar1[1, i]
                                              #On choisit de
              rester sur place sans changer les poids pour ne
               pas fausser l'aleatoire (Juste pour cette
              iteration).
          Mar1[2, i + 1] = Mar1[2, i]
                                              #Ce cas n'a
              quasiment aucune chance d'arriver mais il faut
              tout de meme l'anticiper
        }
      }
    }
  return(list(Mar1 = Mar1, Mar2 = Mar2, P = P,Q = Q,  R = R,
      S = S))
}
```

Code source de la fonction MAR3pop : (Retour à la page)

```
MAR3pop = function(P, Q, R, S, U, V, alpha, beta, gamma,
    delta, N, xdebut, ydebut, xdebut1, ydebut1, xdebut2,
    ydebut2){
  Mar1 = matrix(0, nrow = 2, ncol = N)
  Mar2 = matrix(0, nrow = 2, ncol = N)
  Mar3 = matrix(0, nrow = 2, ncol = N)
  Mar1[1,1] = xdebut
  Mar2[1,1] = xdebut1
  Mar3[1,1] = xdebut2
  Mar1[2,1] = ydebut
  Mar2[2,1] = ydebut1
  Mar3[2,1] = ydebut2
  K = nrow(P)
  for(i in 1:(N-1)){
    x = runif(1)

    u = runif(1)
                                                    #On
        tire un nombre aleatoire entre 0 et 1
    a = P[Mar1[1,i], Mar1[2,i]]
                                              #On cr er des
        variables pour simplifier l'  criture
    b = P[Mar1[1,i], Mar1[2,i] + K]
    c = Q[Mar1[1,i], Mar1[2,i] + K]
    d = Q[Mar1[1,i], Mar1[2,i]]

    v = runif(1)
    e = R[Mar2[1,i], Mar2[2,i]]
                                              #On cr er des
        variables pour simplifier l'  criture
    f = R[Mar2[1,i], Mar2[2,i] + K]
    g = S[Mar2[1,i], Mar2[2,i] + K]
```

```
h = S[Mar2[1,i], Mar2[2,i]]

w = runif(1)
j = U[Mar3[1,i], Mar3[2,i]]
                                          #On cr er des
    variables pour simplifier l' criture
k = U[Mar3[1,i], Mar3[2,i] + K]
l = V[Mar3[1,i], Mar3[2,i] + K]
m = V[Mar3[1,i], Mar3[2,i]]

if((a > 10**300) | (b > 10**300) | (c > 10**300) | (d >
    10**300)){
  warning('renforcement␣trop␣important')
  return(list(Mar1 = matrix(Mar1[Mar1 > 0], nrow = 2,
      byrow = FALSE), Mar2 = matrix(Mar2[Mar2 > 0], nrow
      = 2, byrow = FALSE), Mar3 = matrix(Mar3[Mar3 > 0],
      nrow = 2, byrow = FALSE), P = P, Q = Q, R = R, S =
      S, U = U, V = V, i = i))
}

if((e > 10**300) | (f > 10**300) | (g > 10**300) | (h >
    10**300)){
  warning('renforcement␣trop␣important')
  return(list(Mar1 = matrix(Mar1[Mar1 > 0], nrow = 2,
      byrow = FALSE), Mar2 = matrix(Mar2[Mar2 > 0], nrow
      = 2, byrow = FALSE), Mar3 = matrix(Mar3[Mar3 > 0],
      nrow = 2, byrow = FALSE), P = P, Q = Q, R = R, S =
      S, U = U, V = V, i = i))
}

if((j > 10**300) | (k > 10**300) | (l > 10**300) | (m >
    10**300)){
  warning('renforcement␣trop␣important')
  return(list(Mar1 = matrix(Mar1[Mar1 > 0], nrow = 2,
      byrow = FALSE), Mar2 = matrix(Mar2[Mar2 > 0], nrow
      = 2, byrow = FALSE), Mar3 = matrix(Mar3[Mar3 > 0],
      nrow = 2, byrow = FALSE), P = P, Q = Q, R = R, S =
      S, U = U, V = V, i = i))
}

if(x < (1/3)){

  if(u < (a / (a + b + c + d))){        #Si U est plus
      petit que la proba d'aller a gauche (Poid d'aller a
       gauche diviser par la somme d'aller a droite, a
      gauche, devant et derriere)
    P[Mar1[1,i], Mar1[2,i]] = beta * P[Mar1[1,i], Mar1[2
        ,i]] + alpha                          #On va a
        gauche et on ajoute delta
    R[Mar1[1,i], Mar1[2,i]] = max(0, delta * R[Mar1[1,i
        ], Mar1[2,i]] - gamma)
    U[Mar1[1,i], Mar1[2,i]] = max(0, delta * U[Mar1[1,i
        ], Mar1[2,i]] - gamma)
    Mar1[1, i + 1] = Mar1[1, i] - 1
```

```
  Mar1[2, i + 1] = Mar1[2, i]
}
else if(u < ((a + b) / (a + b + c + d))){    #Si U est
    entre les probas d'aller a gauche et d'aller a
   droite ou a gauche (Poid d'aller a gauche + a
   droite diviser par la somme d'aller a droite, a
   gauche, devant et derriere, car si arrive ici c'est
    que U est plus grand que la proba d'aller a gauche
   )
  P[Mar1[1,i], Mar1[2,i] + K] = beta * P[Mar1[1,i],
      Mar1[2,i] + K] + alpha
                                                  #On
       va a droite et on enleve delta
  R[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * R[Mar1[
      1,i], Mar1[2,i] + K] - gamma)
  U[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * U[Mar1[
      1,i], Mar1[2,i] + K] - gamma)
  Mar1[1, i + 1] = Mar1[1, i] + 1
  Mar1[2, i + 1] = Mar1[2, i]
}
else if(u < ((a + b + c) / (a + b + c + d))){       #
    Si U est entre les probas d'aller a gauche ou a
   droite et d'aller a droite ou a gauche ou devant (
   Poid d'aller a gauche + a droite + devant diviser
   par la somme d'aller a droite, a gauche, devant et
   derriere, car si on arrive ici c'est que U est plus
    grand que la proba d'aller a gauche ou a droite)
  Q[Mar1[1,i], Mar1[2,i] + K] = beta * Q[Mar1[1,i],
      Mar1[2,i] + K] + alpha

      #On va devant et on ajoute delta
  S[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * S[Mar1[
      1,i], Mar1[2,i] + K] - gamma)
  V[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * V[Mar1[
      1,i], Mar1[2,i] + K] - gamma)
  Mar1[1, i + 1] = Mar1[1, i]
  Mar1[2, i + 1] = Mar1[2, i] + 1
}
else if((u != 1) | (Mar1[2,i] != 1)){

   #Si U est different de 1 ou que nous ne sommes pas
   tout en bas
  Q[Mar1[1,i], Mar1[2,i]] = beta * Q[Mar1[1,i], Mar1[2
      ,i]] + alpha

      #On va derriere et on enleve delta
  S[Mar1[1,i], Mar1[2,i]] = max(0, delta * S[Mar1[1,i
      ], Mar1[2,i]] - gamma)
  V[Mar1[1,i], Mar1[2,i]] = max(0, delta * V[Mar1[1,i
      ], Mar1[2,i]] - gamma)
  Mar1[1, i + 1] = Mar1[1, i]
  Mar1[2, i + 1] = Mar1[2, i] - 1
}
else{
```

```
    #Si U est egal a 1 et que nous sommes tout en bas,
  Mar1[1, i + 1] = Mar1[1, i]
                                        #On choisit de
      rester sur place sans changer les poids pour ne
      pas fausser l'aleatoire (Juste pour cette
      iteration).
  Mar1[2, i + 1] = Mar1[2, i]
                                        #Ce cas n'a
      quasiment aucune chance d'arriver mais il faut
      tout de meme l'anticiper
}
if(x < 1/6){

  if(v < (e / (e + f + g + h))){        #Si U est plus
       petit que la proba d'aller a gauche (Poid d'
      aller a gauche diviser par la somme d'aller a
      droite, a gauche, devant et derriere)
    R[Mar2[1,i], Mar2[2,i]] = beta * R[Mar2[1,i], Mar2
        [2,i]] + alpha                              #On va
         a gauche et on ajoute delta
    P[Mar2[1,i], Mar2[2,i]] = max(0, delta * P[Mar2[1,
        i], Mar2[2,i]] - gamma)
    U[Mar2[1,i], Mar2[2,i]] = max(0, delta * U[Mar2[1,
        i], Mar2[2,i]] - gamma)
    Mar2[1, i + 1] = Mar2[1, i] - 1
    Mar2[2, i + 1] = Mar2[2, i]
  }
  else if(v < ((e + f) / (e + f + g + h))){    #Si U
      est entre les probas d'aller a gauche et d'aller
      a droite ou a gauche (Poid d'aller a gauche + a
      droite diviser par la somme d'aller a droite, a
      gauche, devant et derriere, car si arrive ici c'
      est que U est plus grand que la proba d'aller a
      gauche)
    R[Mar2[1,i], Mar2[2,i] + K] = beta * R[Mar2[1,i],
        Mar2[2,i] + K] + alpha
                                                      #
        On va a droite et on enleve delta
    P[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * P[Mar
        2[1,i],  Mar2[2,i] + K] - gamma)
    U[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * U[Mar
        2[1,i],  Mar2[2,i] + K] - gamma)
    Mar2[1, i + 1] = Mar2[1, i] + 1
    Mar2[2, i + 1] = Mar2[2, i]
  }
  else if(v < ((e + f + g) / (e + f + g + h))){
      #Si U est entre les probas d'aller a gauche ou a
      droite et d'aller a droite ou a gauche ou devant
      (Poid d'aller a gauche + a droite + devant
      diviser par la somme d'aller a droite, a gauche,
      devant et derriere, car si on arrive ici c'est
      que U est plus grand que la proba d'aller a
      gauche ou a droite)
```

```
  S[Mar2[1,i], Mar2[2,i] + K] = beta * S[Mar2[1,i],
      Mar2[2,i] + K] + alpha

      #On va devant et on ajoute delta
  Q[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * Q[Mar
      2[1,i], Mar2[2,i] + K] - gamma)
  V[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * V[Mar
      2[1,i], Mar2[2,i] + K] - gamma)
  Mar2[1, i + 1] = Mar2[1, i]
  Mar2[2, i + 1] = Mar2[2, i] + 1
}
else if((v != 1) | (Mar2[2,i] != 1)){

    #Si U est different de 1 ou que nous ne sommes
    pas tout en bas
  S[Mar2[1,i], Mar2[2,i]] = beta * S[Mar2[1,i], Mar2
      [2,i]] + alpha

      #On va derriere et on enleve delta
  Q[Mar2[1,i], Mar2[2,i]] = max(0, delta * Q[Mar2[1,
      i], Mar2[2,i]] - gamma)
  V[Mar2[1,i], Mar2[2,i]] = max(0, delta * V[Mar2[1,
      i], Mar2[2,i]] - gamma)
  Mar2[1, i + 1] = Mar2[1, i]
  Mar2[2, i + 1] = Mar2[2, i] - 1
}
else{

    #Si U est egal a 1 et que nous sommes tout en bas
    ,
  Mar2[1, i + 1] = Mar2[1, i]
                                    #On choisit de
      rester sur place sans changer les poids pour ne
       pas fausser l'aleatoire (Juste pour cette
      iteration).
  Mar2[2, i + 1] = Mar2[2, i]
                                    #Ce cas n'a
      quasiment aucune chance d'arriver mais il faut
      tout de meme l'anticiper
}

if(w < (j / (j + k + l + m))){        #Si U est plus
     petit que la proba d'aller a gauche (Poid d'
    aller a gauche diviser par la somme d'aller a
    droite, a gauche, devant et derriere)
  U[Mar3[1,i], Mar3[2,i]] = beta * U[Mar3[1,i], Mar3
      [2,i]] + alpha                            #On va
       a gauche et on ajoute delta
  P[Mar3[1,i], Mar3[2,i]] = max(0, delta * P[Mar3[1,
      i], Mar3[2,i]] - gamma)
  R[Mar3[1,i], Mar3[2,i]] = max(0, delta * R[Mar3[1,
      i], Mar3[2,i]] - gamma)
  Mar3[1, i + 1] = Mar3[1, i] - 1
  Mar3[2, i + 1] = Mar3[2, i]
```

```
}
else if(w < ((j + k) / (j + k + l + m))){    #Si U
    est entre les probas d'aller a gauche et d'aller
    a droite ou a gauche (Poid d'aller a gauche + a
    droite diviser par la somme d'aller a droite, a
    gauche, devant et derriere, car si arrive ici c'
    est que U est plus grand que la proba d'aller a
    gauche)
  U[Mar3[1,i], Mar3[2,i] + K] = beta * U[Mar3[1,i],
      Mar3[2,i] + K] + alpha
                                                    #
      On va a droite et on enleve delta
  P[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * P[Mar
      3[1,i],  Mar3[2,i] + K] - gamma)
  R[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * R[Mar
      3[1,i],  Mar3[2,i] + K] - gamma)
  Mar3[1, i + 1] = Mar3[1, i] + 1
  Mar3[2, i + 1] = Mar3[2, i]
}
else if(w < ((j + k + l) / (j + k + l + m))){
    #Si U est entre les probas d'aller a gauche ou a
    droite et d'aller a droite ou a gauche ou devant
    (Poid d'aller a gauche + a droite + devant
    diviser par la somme d'aller a droite, a gauche,
    devant et derriere, car si on arrive ici c'est
    que U est plus grand que la proba d'aller a
    gauche ou a droite)
  V[Mar3[1,i], Mar3[2,i] + K] = beta * V[Mar3[1,i],
      Mar3[2,i] + K] + alpha

      #On va devant et on ajoute delta
  Q[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * Q[Mar
      3[1,i], Mar3[2,i] + K] - gamma)
  S[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * S[Mar
      3[1,i], Mar3[2,i] + K] - gamma)
  Mar3[1, i + 1] = Mar3[1, i]
  Mar3[2, i + 1] = Mar3[2, i] + 1
}
else if((w != 1) | (Mar3[2,i] != 1)){

    #Si U est different de 1 ou que nous ne sommes
    pas tout en bas
  V[Mar3[1,i], Mar3[2,i]] = beta * V[Mar3[1,i], Mar3
      [2,i]] + alpha

      #On va derriere et on enleve delta
  Q[Mar3[1,i], Mar3[2,i]] = max(0, delta * Q[Mar3[1,
      i], Mar3[2,i]] - gamma)
  S[Mar3[1,i], Mar3[2,i]] = max(0, delta * S[Mar3[1,
      i], Mar3[2,i]] - gamma)
  Mar3[1, i + 1] = Mar3[1, i]
  Mar3[2, i + 1] = Mar3[2, i] - 1
}
else{
```

```
      #Si U est egal a 1 et que nous sommes tout en bas
      ,
    Mar3[1, i + 1] = Mar3[1, i]
                                        #On choisit de
       rester sur place sans changer les poids pour ne
        pas fausser l'aleatoire (Juste pour cette
       iteration).
    Mar3[2, i + 1] = Mar3[2, i]
                                        #Ce cas n'a
       quasiment aucune chance d'arriver mais il faut
       tout de meme l'anticiper
  }
}
else{

  if(w < (j / (j + k + l + m))){        #Si U est plus
       petit que la proba d'aller a gauche (Poid d'
      aller a gauche diviser par la somme d'aller a
      droite, a gauche, devant et derriere)
    U[Mar3[1,i], Mar3[2,i]] = beta * U[Mar3[1,i], Mar3
        [2,i]] + alpha                              #On va
         a gauche et on ajoute delta
    P[Mar3[1,i], Mar3[2,i]] = max(0, delta * P[Mar3[1,
        i], Mar3[2,i]] - gamma)
    R[Mar3[1,i], Mar3[2,i]] = max(0, delta * R[Mar3[1,
        i], Mar3[2,i]] - gamma)
    Mar3[1, i + 1] = Mar3[1, i] - 1
    Mar3[2, i + 1] = Mar3[2, i]
  }
  else if(w < ((j + k) / (j + k + l + m))){    #Si U
      est entre les probas d'aller a gauche et d'aller
      a droite ou a gauche (Poid d'aller a gauche + a
      droite diviser par la somme d'aller a droite, a
      gauche, devant et derriere, car si arrive ici c'
      est que U est plus grand que la proba d'aller a
      gauche)
    U[Mar3[1,i], Mar3[2,i] + K] = beta * U[Mar3[1,i],
        Mar3[2,i] + K] + alpha
                                                      #
        On va a droite et on enleve delta
    P[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * P[Mar
        3[1,i],  Mar3[2,i] + K] - gamma)
    R[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * R[Mar
        3[1,i],  Mar3[2,i] + K] - gamma)
    Mar3[1, i + 1] = Mar3[1, i] + 1
    Mar3[2, i + 1] = Mar3[2, i]
  }
  else if(w < ((j + k + l) / (j + k + l + m))){
      #Si U est entre les probas d'aller a gauche ou a
      droite et d'aller a droite ou a gauche ou devant
      (Poid d'aller a gauche + a droite + devant
      diviser par la somme d'aller a droite, a gauche,
      devant et derriere, car si on arrive ici c'est
```

```
    que U est plus grand que la proba d'aller a
    gauche ou a droite)
  V[Mar3[1,i], Mar3[2,i] + K] = beta * V[Mar3[1,i],
      Mar3[2,i] + K] + alpha

      #On va devant et on ajoute delta
  Q[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * Q[Mar
      3[1,i], Mar3[2,i] + K] - gamma)
  S[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * S[Mar
      3[1,i], Mar3[2,i] + K] - gamma)
  Mar3[1, i + 1] = Mar3[1, i]
  Mar3[2, i + 1] = Mar3[2, i] + 1
}
else if((w != 1) | (Mar3[2,i] != 1)){

    #Si U est different de 1 ou que nous ne sommes
    pas tout en bas
  V[Mar3[1,i], Mar3[2,i]] = beta * V[Mar3[1,i], Mar3
      [2,i]] + alpha

      #On va derriere et on enleve delta
  Q[Mar3[1,i], Mar3[2,i]] = max(0, delta * Q[Mar3[1,
      i], Mar3[2,i]] - gamma)
  S[Mar3[1,i], Mar3[2,i]] = max(0, delta * S[Mar3[1,
      i], Mar3[2,i]] - gamma)
  Mar3[1, i + 1] = Mar3[1, i]
  Mar3[2, i + 1] = Mar3[2, i] - 1
}
else{

    #Si U est egal a 1 et que nous sommes tout en bas
    ,
  Mar3[1, i + 1] = Mar3[1, i]
                                      #On choisit de
      rester sur place sans changer les poids pour ne
       pas fausser l'aleatoire (Juste pour cette
      iteration).
  Mar3[2, i + 1] = Mar3[2, i]
                                      #Ce cas n'a
      quasiment aucune chance d'arriver mais il faut
      tout de meme l'anticiper
}

if(v < (e / (e + f + g + h))){        #Si U est plus
     petit que la proba d'aller a gauche (Poid d'
    aller a gauche diviser par la somme d'aller a
    droite, a gauche, devant et derriere)
  R[Mar2[1,i], Mar2[2,i]] = beta * R[Mar2[1,i], Mar2
      [2,i]] + alpha                              #On va
       a gauche et on ajoute delta
  P[Mar2[1,i], Mar2[2,i]] = max(0, delta * P[Mar2[1,
      i], Mar2[2,i]] - gamma)
  U[Mar2[1,i], Mar2[2,i]] = max(0, delta * U[Mar2[1,
      i], Mar2[2,i]] - gamma)
```

```
  Mar2[1, i + 1] = Mar2[1, i] - 1
  Mar2[2, i + 1] = Mar2[2, i]
}
else if(v < ((e + f) / (e + f + g + h))){    #Si U
    est entre les probas d'aller a gauche et d'aller
    a droite ou a gauche (Poid d'aller a gauche + a
    droite diviser par la somme d'aller a droite, a
    gauche, devant et derriere, car si arrive ici c'
    est que U est plus grand que la proba d'aller a
    gauche)
  R[Mar2[1,i], Mar2[2,i] + K] = beta * R[Mar2[1,i],
      Mar2[2,i] + K] + alpha
                                                    #
      On va a droite et on enleve delta
  P[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * P[Mar
      2[1,i],  Mar2[2,i] + K] - gamma)
  U[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * U[Mar
      2[1,i],  Mar2[2,i] + K] - gamma)
  Mar2[1, i + 1] = Mar2[1, i] + 1
  Mar2[2, i + 1] = Mar2[2, i]
}
else if(v < ((e + f + g) / (e + f + g + h))){
    #Si U est entre les probas d'aller a gauche ou a
    droite et d'aller a droite ou a gauche ou devant
    (Poid d'aller a gauche + a droite + devant
    diviser par la somme d'aller a droite, a gauche,
    devant et derriere, car si on arrive ici c'est
    que U est plus grand que la proba d'aller a
    gauche ou a droite)
  S[Mar2[1,i], Mar2[2,i] + K] = beta * S[Mar2[1,i],
      Mar2[2,i] + K] + alpha

      #On va devant et on ajoute delta
  Q[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * Q[Mar
      2[1,i], Mar2[2,i] + K] - gamma)
  V[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * V[Mar
      2[1,i], Mar2[2,i] + K] - gamma)
  Mar2[1, i + 1] = Mar2[1, i]
  Mar2[2, i + 1] = Mar2[2, i] + 1
}
else if((v != 1) | (Mar2[2,i] != 1)){

    #Si U est different de 1 ou que nous ne sommes
    pas tout en bas
  S[Mar2[1,i], Mar2[2,i]] = beta * S[Mar2[1,i], Mar2
      [2,i]] + alpha

      #On va derriere et on enleve delta
  Q[Mar2[1,i], Mar2[2,i]] = max(0, delta * Q[Mar2[1,
      i], Mar2[2,i]] - gamma)
  V[Mar2[1,i], Mar2[2,i]] = max(0, delta * V[Mar2[1,
      i], Mar2[2,i]] - gamma)
  Mar2[1, i + 1] = Mar2[1, i]
  Mar2[2, i + 1] = Mar2[2, i] - 1
```

```
    }
    else{

      #Si U est egal a 1 et que nous sommes tout en bas
      ,
      Mar2[1, i + 1] = Mar2[1, i]
                                          #On choisit de
        rester sur place sans changer les poids pour ne
         pas fausser l'aleatoire (Juste pour cette
        iteration).
      Mar2[2, i + 1] = Mar2[2, i]
                                          #Ce cas n'a
        quasiment aucune chance d'arriver mais il faut
        tout de meme l'anticiper
    }

  }
}
else if(x < (2/3)){

  if(v < (e / (e + f + g + h))){        #Si U est plus
      petit que la proba d'aller a gauche (Poid d'aller a
       gauche diviser par la somme d'aller a droite, a
      gauche, devant et derriere)
    R[Mar2[1,i], Mar2[2,i]] = beta * R[Mar2[1,i], Mar2[2
        ,i]] + alpha                            #On va a
        gauche et on ajoute delta
    P[Mar2[1,i], Mar2[2,i]] = max(0, delta * P[Mar2[1,i
        ], Mar2[2,i]] - gamma)
    U[Mar2[1,i], Mar2[2,i]] = max(0, delta * U[Mar2[1,i
        ], Mar2[2,i]] - gamma)
    Mar2[1, i + 1] = Mar2[1, i] - 1
    Mar2[2, i + 1] = Mar2[2, i]
  }
  else if(v < ((e + f) / (e + f + g + h))){    #Si U est
       entre les probas d'aller a gauche et d'aller a
      droite ou a gauche (Poid d'aller a gauche + a
      droite diviser par la somme d'aller a droite, a
      gauche, devant et derriere, car si arrive ici c'est
       que U est plus grand que la proba d'aller a gauche
      )
    R[Mar2[1,i], Mar2[2,i] + K] = beta * R[Mar2[1,i],
        Mar2[2,i] + K] + alpha
                                                    #On
         va a droite et on enleve delta
    P[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * P[Mar2[
        1,i],  Mar2[2,i] + K] - gamma)
    U[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * U[Mar2[
        1,i],  Mar2[2,i] + K] - gamma)
    Mar2[1, i + 1] = Mar2[1, i] + 1
    Mar2[2, i + 1] = Mar2[2, i]
  }
  else if(v < ((e + f + g) / (e + f + g + h))){       #
      Si U est entre les probas d'aller a gauche ou a
```

```
    droite et d'aller a droite ou a gauche ou devant (
    Poid d'aller a gauche + a droite + devant diviser
    par la somme d'aller a droite, a gauche, devant et
    derriere, car si on arrive ici c'est que U est plus
     grand que la proba d'aller a gauche ou a droite)
  S[Mar2[1,i], Mar2[2,i] + K] = beta * S[Mar2[1,i],
      Mar2[2,i] + K] + alpha

      #On va devant et on ajoute delta
  Q[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * Q[Mar2[
      1,i], Mar2[2,i] + K] - gamma)
  V[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * V[Mar2[
      1,i], Mar2[2,i] + K] - gamma)
  Mar2[1, i + 1] = Mar2[1, i]
  Mar2[2, i + 1] = Mar2[2, i] + 1
}
else if((v != 1) | (Mar2[2,i] != 1)){

    #Si U est different de 1 ou que nous ne sommes pas
    tout en bas
  S[Mar2[1,i], Mar2[2,i]] = beta * S[Mar2[1,i], Mar2[2
      ,i]] + alpha

      #On va derriere et on enleve delta
  Q[Mar2[1,i], Mar2[2,i]] = max(0, delta * Q[Mar2[1,i
      ], Mar2[2,i]] - gamma)
  V[Mar2[1,i], Mar2[2,i]] = max(0, delta * V[Mar2[1,i
      ], Mar2[2,i]] - gamma)
  Mar2[1, i + 1] = Mar2[1, i]
  Mar2[2, i + 1] = Mar2[2, i] - 1
}
else{

    #Si U est egal a 1 et que nous sommes tout en bas,
  Mar2[1, i + 1] = Mar2[1, i]
                                      #On choisit de
      rester sur place sans changer les poids pour ne
      pas fausser l'aleatoire (Juste pour cette
      iteration).
  Mar2[2, i + 1] = Mar2[2, i]
                                      #Ce cas n'a
      quasiment aucune chance d'arriver mais il faut
      tout de meme l'anticiper
}

if(x < 1/2){

  if(u < (a / (a + b + c + d))){        #Si U est plus
       petit que la proba d'aller a gauche (Poid d'
      aller a gauche diviser par la somme d'aller a
      droite, a gauche, devant et derriere)
    P[Mar1[1,i], Mar1[2,i]] = beta * P[Mar1[1,i], Mar1
        [2,i]] + alpha                              #On va
         a gauche et on ajoute delta
```

```
  R[Mar1[1,i], Mar1[2,i]] = max(0, delta * R[Mar1[1,
      i], Mar1[2,i]] - gamma)
  U[Mar1[1,i], Mar1[2,i]] = max(0, delta * U[Mar1[1,
      i], Mar1[2,i]] - gamma)
  Mar1[1, i + 1] = Mar1[1, i] - 1
  Mar1[2, i + 1] = Mar1[2, i]
}
else if(u < ((a + b) / (a + b + c + d))){    #Si U
    est entre les probas d'aller a gauche et d'aller
    a droite ou a gauche (Poid d'aller a gauche + a
    droite diviser par la somme d'aller a droite, a
    gauche, devant et derriere, car si arrive ici c'
    est que U est plus grand que la proba d'aller a
    gauche)
  P[Mar1[1,i], Mar1[2,i] + K] = beta * P[Mar1[1,i],
      Mar1[2,i] + K] + alpha
                                                    #
      On va a droite et on enleve delta
  R[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * R[Mar
      1[1,i], Mar1[2,i] + K] - gamma)
  U[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * U[Mar
      1[1,i], Mar1[2,i] + K] - gamma)
  Mar1[1, i + 1] = Mar1[1, i] + 1
  Mar1[2, i + 1] = Mar1[2, i]
}
else if(u < ((a + b + c) / (a + b + c + d))){
    #Si U est entre les probas d'aller a gauche ou a
    droite et d'aller a droite ou a gauche ou devant
    (Poid d'aller a gauche + a droite + devant
    diviser par la somme d'aller a droite, a gauche,
    devant et derriere, car si on arrive ici c'est
    que U est plus grand que la proba d'aller a
    gauche ou a droite)
  Q[Mar1[1,i], Mar1[2,i] + K] = beta * Q[Mar1[1,i],
      Mar1[2,i] + K] + alpha

      #On va devant et on ajoute delta
  S[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * S[Mar
      1[1,i], Mar1[2,i] + K] - gamma)
  V[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * V[Mar
      1[1,i], Mar1[2,i] + K] - gamma)
  Mar1[1, i + 1] = Mar1[1, i]
  Mar1[2, i + 1] = Mar1[2, i] + 1
}
else if((u != 1) | (Mar1[2,i] != 1)){

    #Si U est different de 1 ou que nous ne sommes
    pas tout en bas
  Q[Mar1[1,i], Mar1[2,i]] = beta * Q[Mar1[1,i], Mar1
      [2,i]] + alpha

      #On va derriere et on enleve delta
  S[Mar1[1,i], Mar1[2,i]] = max(0, delta * S[Mar1[1,
      i], Mar1[2,i]] - gamma)
```

```
  V[Mar1[1,i], Mar1[2,i]] = max(0, delta * V[Mar1[1,
      i], Mar1[2,i]] - gamma)
  Mar1[1, i + 1] = Mar1[1, i]
  Mar1[2, i + 1] = Mar1[2, i] - 1
}
else{

    #Si U est egal a 1 et que nous sommes tout en bas
    ,
  Mar1[1, i + 1] = Mar1[1, i]
                                      #On choisit de
      rester sur place sans changer les poids pour ne
       pas fausser l'aleatoire (Juste pour cette
      iteration).
  Mar1[2, i + 1] = Mar1[2, i]
                                      #Ce cas n'a
      quasiment aucune chance d'arriver mais il faut
      tout de meme l'anticiper
}

if(w < (j / (j + k + l + m))){        #Si U est plus
     petit que la proba d'aller a gauche (Poid d'
    aller a gauche diviser par la somme d'aller a
    droite, a gauche, devant et derriere)
  U[Mar3[1,i], Mar3[2,i]] = beta * U[Mar3[1,i], Mar3
      [2,i]] + alpha                              #On va
       a gauche et on ajoute delta
  P[Mar3[1,i], Mar3[2,i]] = max(0, delta * P[Mar3[1,
      i], Mar3[2,i]] - gamma)
  R[Mar3[1,i], Mar3[2,i]] = max(0, delta * R[Mar3[1,
      i], Mar3[2,i]] - gamma)
  Mar3[1, i + 1] = Mar3[1, i] - 1
  Mar3[2, i + 1] = Mar3[2, i]
}
else if(w < ((j + k) / (j + k + l + m))){    #Si U
    est entre les probas d'aller a gauche et d'aller
    a droite ou a gauche (Poid d'aller a gauche + a
    droite diviser par la somme d'aller a droite, a
    gauche, devant et derriere, car si arrive ici c'
    est que U est plus grand que la proba d'aller a
    gauche)
  U[Mar3[1,i], Mar3[2,i] + K] = beta * U[Mar3[1,i],
      Mar3[2,i] + K] + alpha
                                                    #
      On va a droite et on enleve delta
  P[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * P[Mar
      3[1,i],  Mar3[2,i] + K] - gamma)
  R[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * R[Mar
      3[1,i],  Mar3[2,i] + K] - gamma)
  Mar3[1, i + 1] = Mar3[1, i] + 1
  Mar3[2, i + 1] = Mar3[2, i]
}
else if(w < ((j + k + l) / (j + k + l + m))){
    #Si U est entre les probas d'aller a gauche ou a
```

```
      droite et d'aller a droite ou a gauche ou devant
      (Poid d'aller a gauche + a droite + devant
      diviser par la somme d'aller a droite, a gauche,
      devant et derriere, car si on arrive ici c'est
      que U est plus grand que la proba d'aller a
      gauche ou a droite)
    V[Mar3[1,i], Mar3[2,i] + K] = beta * V[Mar3[1,i],
        Mar3[2,i] + K] + alpha

        #On va devant et on ajoute delta
    Q[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * Q[Mar
        3[1,i], Mar3[2,i] + K] - gamma)
    S[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * S[Mar
        3[1,i], Mar3[2,i] + K] - gamma)
    Mar3[1, i + 1] = Mar3[1, i]
    Mar3[2, i + 1] = Mar3[2, i] + 1
  }
  else if((w != 1) | (Mar3[2,i] != 1)){

      #Si U est different de 1 ou que nous ne sommes
      pas tout en bas
    V[Mar3[1,i], Mar3[2,i]] = beta * V[Mar3[1,i], Mar3
        [2,i]] + alpha

        #On va derriere et on enleve delta
    Q[Mar3[1,i], Mar3[2,i]] = max(0, delta * Q[Mar3[1,
        i], Mar3[2,i]] - gamma)
    S[Mar3[1,i], Mar3[2,i]] = max(0, delta * S[Mar3[1,
        i], Mar3[2,i]] - gamma)
    Mar3[1, i + 1] = Mar3[1, i]
    Mar3[2, i + 1] = Mar3[2, i] - 1
  }
  else{

      #Si U est egal a 1 et que nous sommes tout en bas
      ,
    Mar3[1, i + 1] = Mar3[1, i]
                                        #On choisit de
        rester sur place sans changer les poids pour ne
         pas fausser l'aleatoire (Juste pour cette
        iteration).
    Mar3[2, i + 1] = Mar3[2, i]
                                        #Ce cas n'a
        quasiment aucune chance d'arriver mais il faut
        tout de meme l'anticiper
  }
}

else{

  if(w < (j / (j + k + l + m))){        #Si U est plus
       petit que la proba d'aller a gauche (Poid d'
      aller a gauche diviser par la somme d'aller a
      droite, a gauche, devant et derriere)
```

```
  U[Mar3[1,i], Mar3[2,i]] = beta * U[Mar3[1,i], Mar3
      [2,i]] + alpha                              #On va
       a gauche et on ajoute delta
  P[Mar3[1,i], Mar3[2,i]] = max(0, delta * P[Mar3[1,
      i], Mar3[2,i]] - gamma)
  R[Mar3[1,i], Mar3[2,i]] = max(0, delta * R[Mar3[1,
      i], Mar3[2,i]] - gamma)
  Mar3[1, i + 1] = Mar3[1, i] - 1
  Mar3[2, i + 1] = Mar3[2, i]
}
else if(w < ((j + k) / (j + k + l + m))){    #Si U
    est entre les probas d'aller a gauche et d'aller
    a droite ou a gauche (Poid d'aller a gauche + a
    droite diviser par la somme d'aller a droite, a
    gauche, devant et derriere, car si arrive ici c'
    est que U est plus grand que la proba d'aller a
    gauche)
  U[Mar3[1,i], Mar3[2,i] + K] = beta * U[Mar3[1,i],
      Mar3[2,i] + K] + alpha
                                                    #
      On va a droite et on enleve delta
  P[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * P[Mar
      3[1,i],  Mar3[2,i] + K] - gamma)
  R[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * R[Mar
      3[1,i],  Mar3[2,i] + K] - gamma)
  Mar3[1, i + 1] = Mar3[1, i] + 1
  Mar3[2, i + 1] = Mar3[2, i]
}
else if(w < ((j + k + l) / (j + k + l + m))){
    #Si U est entre les probas d'aller a gauche ou a
    droite et d'aller a droite ou a gauche ou devant
    (Poid d'aller a gauche + a droite + devant
    diviser par la somme d'aller a droite, a gauche,
    devant et derriere, car si on arrive ici c'est
    que U est plus grand que la proba d'aller a
    gauche ou a droite)
  V[Mar3[1,i], Mar3[2,i] + K] = beta * V[Mar3[1,i],
      Mar3[2,i] + K] + alpha

      #On va devant et on ajoute delta
  Q[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * Q[Mar
      3[1,i], Mar3[2,i] + K] - gamma)
  S[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * S[Mar
      3[1,i], Mar3[2,i] + K] - gamma)
  Mar3[1, i + 1] = Mar3[1, i]
  Mar3[2, i + 1] = Mar3[2, i] + 1
}
else if((w != 1) | (Mar3[2,i] != 1)){

    #Si U est different de 1 ou que nous ne sommes
    pas tout en bas
  V[Mar3[1,i], Mar3[2,i]] = beta * V[Mar3[1,i], Mar3
      [2,i]] + alpha
```

```
      #On va derriere et on enleve delta
  Q[Mar3[1,i], Mar3[2,i]] = max(0, delta * Q[Mar3[1,
      i], Mar3[2,i]] - gamma)
  S[Mar3[1,i], Mar3[2,i]] = max(0, delta * S[Mar3[1,
      i], Mar3[2,i]] - gamma)
  Mar3[1, i + 1] = Mar3[1, i]
  Mar3[2, i + 1] = Mar3[2, i] - 1
}
else{

    #Si U est egal a 1 et que nous sommes tout en bas
    ,
  Mar3[1, i + 1] = Mar3[1, i]
                                      #On choisit de
      rester sur place sans changer les poids pour ne
       pas fausser l'aleatoire (Juste pour cette
      iteration).
  Mar3[2, i + 1] = Mar3[2, i]
                                      #Ce cas n'a
      quasiment aucune chance d'arriver mais il faut
      tout de meme l'anticiper
}

if(u < (a / (a + b + c + d))){        #Si U est plus
     petit que la proba d'aller a gauche (Poid d'
    aller a gauche diviser par la somme d'aller a
    droite, a gauche, devant et derriere)
  P[Mar1[1,i], Mar1[2,i]] = beta * P[Mar1[1,i], Mar1
      [2,i]] + alpha                          #On va
       a gauche et on ajoute delta
  R[Mar1[1,i], Mar1[2,i]] = max(0, delta * R[Mar1[1,
      i], Mar1[2,i]] - gamma)
  U[Mar1[1,i], Mar1[2,i]] = max(0, delta * U[Mar1[1,
      i], Mar1[2,i]] - gamma)
  Mar1[1, i + 1] = Mar1[1, i] - 1
  Mar1[2, i + 1] = Mar1[2, i]
}
else if(u < ((a + b) / (a + b + c + d))){    #Si U
    est entre les probas d'aller a gauche et d'aller
    a droite ou a gauche (Poid d'aller a gauche + a
    droite diviser par la somme d'aller a droite, a
    gauche, devant et derriere, car si arrive ici c'
    est que U est plus grand que la proba d'aller a
    gauche)
  P[Mar1[1,i], Mar1[2,i] + K] = beta * P[Mar1[1,i],
      Mar1[2,i] + K] + alpha
                                                  #
      On va a droite et on enleve delta
  R[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * R[Mar
      1[1,i], Mar1[2,i] + K] - gamma)
  U[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * U[Mar
      1[1,i], Mar1[2,i] + K] - gamma)
  Mar1[1, i + 1] = Mar1[1, i] + 1
  Mar1[2, i + 1] = Mar1[2, i]
```

```
    }
    else if(u < ((a + b + c) / (a + b + c + d))){
        #Si U est entre les probas d'aller a gauche ou a
        droite et d'aller a droite ou a gauche ou devant
        (Poid d'aller a gauche + a droite + devant
        diviser par la somme d'aller a droite, a gauche,
        devant et derriere, car si on arrive ici c'est
        que U est plus grand que la proba d'aller a
        gauche ou a droite)
      Q[Mar1[1,i], Mar1[2,i] + K] = beta * Q[Mar1[1,i],
          Mar1[2,i] + K] + alpha

          #On va devant et on ajoute delta
      S[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * S[Mar
          1[1,i], Mar1[2,i] + K] - gamma)
      V[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * V[Mar
          1[1,i], Mar1[2,i] + K] - gamma)
      Mar1[1, i + 1] = Mar1[1, i]
      Mar1[2, i + 1] = Mar1[2, i] + 1
    }
    else if((u != 1) | (Mar1[2,i] != 1)){

        #Si U est different de 1 ou que nous ne sommes
        pas tout en bas
      Q[Mar1[1,i], Mar1[2,i]] = beta * Q[Mar1[1,i], Mar1
          [2,i]] + alpha

          #On va derriere et on enleve delta
      S[Mar1[1,i], Mar1[2,i]] = max(0, delta * S[Mar1[1,
          i], Mar1[2,i]] - gamma)
      V[Mar1[1,i], Mar1[2,i]] = max(0, delta * V[Mar1[1,
          i], Mar1[2,i]] - gamma)
      Mar1[1, i + 1] = Mar1[1, i]
      Mar1[2, i + 1] = Mar1[2, i] - 1
    }
    else{

        #Si U est egal a 1 et que nous sommes tout en bas
        ,
      Mar1[1, i + 1] = Mar1[1, i]
                                          #On choisit de
          rester sur place sans changer les poids pour ne
           pas fausser l'aleatoire (Juste pour cette
          iteration).
      Mar1[2, i + 1] = Mar1[2, i]
                                          #Ce cas n'a
          quasiment aucune chance d'arriver mais il faut
          tout de meme l'anticiper
    }


  }

}
```

```
else{

  if(w < (j / (j + k + l + m))){        #Si U est plus
     petit que la proba d'aller a gauche (Poid d'aller a
      gauche diviser par la somme d'aller a droite, a
     gauche, devant et derriere)
    U[Mar3[1,i], Mar3[2,i]] = beta * U[Mar3[1,i], Mar3[2
        ,i]] + alpha                              #On va a
        gauche et on ajoute delta
    P[Mar3[1,i], Mar3[2,i]] = max(0, delta * P[Mar3[1,i
        ], Mar3[2,i]] - gamma)
    R[Mar3[1,i], Mar3[2,i]] = max(0, delta * R[Mar3[1,i
        ], Mar3[2,i]] - gamma)
    Mar3[1, i + 1] = Mar3[1, i] - 1
    Mar3[2, i + 1] = Mar3[2, i]
  }
  else if(w < ((j + k) / (j + k + l + m))){    #Si U est
       entre les probas d'aller a gauche et d'aller a
     droite ou a gauche (Poid d'aller a gauche + a
     droite diviser par la somme d'aller a droite, a
     gauche, devant et derriere, car si arrive ici c'est
      que U est plus grand que la proba d'aller a gauche
     )
    U[Mar3[1,i], Mar3[2,i] + K] = beta * U[Mar3[1,i],
        Mar3[2,i] + K] + alpha
                                                      #On
         va a droite et on enleve delta
    P[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * P[Mar3[
        1,i],  Mar3[2,i] + K] - gamma)
    R[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * R[Mar3[
        1,i],  Mar3[2,i] + K] - gamma)
    Mar3[1, i + 1] = Mar3[1, i] + 1
    Mar3[2, i + 1] = Mar3[2, i]
  }
  else if(w < ((j + k + l) / (j + k + l + m))){       #
     Si U est entre les probas d'aller a gauche ou a
     droite et d'aller a droite ou a gauche ou devant (
     Poid d'aller a gauche + a droite + devant diviser
     par la somme d'aller a droite, a gauche, devant et
     derriere, car si on arrive ici c'est que U est plus
      grand que la proba d'aller a gauche ou a droite)
    V[Mar3[1,i], Mar3[2,i] + K] = beta * V[Mar3[1,i],
        Mar3[2,i] + K] + alpha

        #On va devant et on ajoute delta
    Q[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * Q[Mar3[
        1,i], Mar3[2,i] + K] - gamma)
    S[Mar3[1,i], Mar3[2,i] + K] = max(0, delta * S[Mar3[
        1,i], Mar3[2,i] + K] - gamma)
    Mar3[1, i + 1] = Mar3[1, i]
    Mar3[2, i + 1] = Mar3[2, i] + 1
  }
  else if((w != 1) | (Mar3[2,i] != 1)){
```

```
    #Si U est different de 1 ou que nous ne sommes pas
    tout en bas
  V[Mar3[1,i], Mar3[2,i]] = beta * V[Mar3[1,i], Mar3[2
      ,i]] + alpha

      #On va derriere et on enleve delta
  Q[Mar3[1,i], Mar3[2,i]] = max(0, delta * Q[Mar3[1,i
      ], Mar3[2,i]] - gamma)
  S[Mar3[1,i], Mar3[2,i]] = max(0, delta * S[Mar3[1,i
      ], Mar3[2,i]] - gamma)
  Mar3[1, i + 1] = Mar3[1, i]
  Mar3[2, i + 1] = Mar3[2, i] - 1
}
else{

    #Si U est egal a 1 et que nous sommes tout en bas,
  Mar3[1, i + 1] = Mar3[1, i]
                                      #On choisit de
      rester sur place sans changer les poids pour ne
      pas fausser l'aleatoire (Juste pour cette
      iteration).
  Mar3[2, i + 1] = Mar3[2, i]
                                      #Ce cas n'a
      quasiment aucune chance d'arriver mais il faut
      tout de meme l'anticiper
}

if(x < 5/6){

  if(u < (a / (a + b + c + d))){        #Si U est plus
       petit que la proba d'aller a gauche (Poid d'
      aller a gauche diviser par la somme d'aller a
      droite, a gauche, devant et derriere)
    P[Mar1[1,i], Mar1[2,i]] = beta * P[Mar1[1,i], Mar1
        [2,i]] + alpha                          #On va
         a gauche et on ajoute delta
    R[Mar1[1,i], Mar1[2,i]] = max(0, delta * R[Mar1[1,
        i], Mar1[2,i]] - gamma)
    U[Mar1[1,i], Mar1[2,i]] = max(0, delta * U[Mar1[1,
        i], Mar1[2,i]] - gamma)
    Mar1[1, i + 1] = Mar1[1, i] - 1
    Mar1[2, i + 1] = Mar1[2, i]
  }
  else if(u < ((a + b) / (a + b + c + d))){    #Si U
      est entre les probas d'aller a gauche et d'aller
      a droite ou a gauche (Poid d'aller a gauche + a
      droite diviser par la somme d'aller a droite, a
      gauche, devant et derriere, car si arrive ici c'
      est que U est plus grand que la proba d'aller a
      gauche)
    P[Mar1[1,i], Mar1[2,i] + K] = beta * P[Mar1[1,i],
        Mar1[2,i] + K] + alpha
                                                    #
        On va a droite et on enleve delta
```

```
  R[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * R[Mar
      1[1,i], Mar1[2,i] + K] - gamma)
  U[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * U[Mar
      1[1,i], Mar1[2,i] + K] - gamma)
  Mar1[1, i + 1] = Mar1[1, i] + 1
  Mar1[2, i + 1] = Mar1[2, i]
}
else if(u < ((a + b + c) / (a + b + c + d))){
    #Si U est entre les probas d'aller a gauche ou a
    droite et d'aller a droite ou a gauche ou devant
    (Poid d'aller a gauche + a droite + devant
    diviser par la somme d'aller a droite, a gauche,
    devant et derriere, car si on arrive ici c'est
    que U est plus grand que la proba d'aller a
    gauche ou a droite)
  Q[Mar1[1,i], Mar1[2,i] + K] = beta * Q[Mar1[1,i],
      Mar1[2,i] + K] + alpha

      #On va devant et on ajoute delta
  S[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * S[Mar
      1[1,i], Mar1[2,i] + K] - gamma)
  V[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * V[Mar
      1[1,i], Mar1[2,i] + K] - gamma)
  Mar1[1, i + 1] = Mar1[1, i]
  Mar1[2, i + 1] = Mar1[2, i] + 1
}
else if((u != 1) | (Mar1[2,i] != 1)){

    #Si U est different de 1 ou que nous ne sommes
    pas tout en bas
  Q[Mar1[1,i], Mar1[2,i]] = beta * Q[Mar1[1,i], Mar1
      [2,i]] + alpha

      #On va derriere et on enleve delta
  S[Mar1[1,i], Mar1[2,i]] = max(0, delta * S[Mar1[1,
      i], Mar1[2,i]] - gamma)
  V[Mar1[1,i], Mar1[2,i]] = max(0, delta * V[Mar1[1,
      i], Mar1[2,i]] - gamma)
  Mar1[1, i + 1] = Mar1[1, i]
  Mar1[2, i + 1] = Mar1[2, i] - 1
}
else{

    #Si U est egal a 1 et que nous sommes tout en bas
    ,
  Mar1[1, i + 1] = Mar1[1, i]
                                      #On choisit de
      rester sur place sans changer les poids pour ne
       pas fausser l'aleatoire (Juste pour cette
      iteration).
  Mar1[2, i + 1] = Mar1[2, i]
                                      #Ce cas n'a
      quasiment aucune chance d'arriver mais il faut
      tout de meme l'anticiper
```

```
}

if(v < (e / (e + f + g + h))){        #Si U est plus
    petit que la proba d'aller a gauche (Poid d'
    aller a gauche diviser par la somme d'aller a
    droite, a gauche, devant et derriere)
  R[Mar2[1,i], Mar2[2,i]] = beta * R[Mar2[1,i], Mar2
      [2,i]] + alpha                              #On va
       a gauche et on ajoute delta
  P[Mar2[1,i], Mar2[2,i]] = max(0, delta * P[Mar2[1,
      i], Mar2[2,i]] - gamma)
  U[Mar2[1,i], Mar2[2,i]] = max(0, delta * U[Mar2[1,
      i], Mar2[2,i]] - gamma)
  Mar2[1, i + 1] = Mar2[1, i] - 1
  Mar2[2, i + 1] = Mar2[2, i]
}
else if(v < ((e + f) / (e + f + g + h))){    #Si U
    est entre les probas d'aller a gauche et d'aller
    a droite ou a gauche (Poid d'aller a gauche + a
    droite diviser par la somme d'aller a droite, a
    gauche, devant et derriere, car si arrive ici c'
    est que U est plus grand que la proba d'aller a
    gauche)
  R[Mar2[1,i], Mar2[2,i] + K] = beta * R[Mar2[1,i],
      Mar2[2,i] + K] + alpha
                                                    #
      On va a droite et on enleve delta
  P[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * P[Mar
      2[1,i],  Mar2[2,i] + K] - gamma)
  U[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * U[Mar
      2[1,i],  Mar2[2,i] + K] - gamma)
  Mar2[1, i + 1] = Mar2[1, i] + 1
  Mar2[2, i + 1] = Mar2[2, i]
}
else if(v < ((e + f + g) / (e + f + g + h))){
    #Si U est entre les probas d'aller a gauche ou a
    droite et d'aller a droite ou a gauche ou devant
    (Poid d'aller a gauche + a droite + devant
    diviser par la somme d'aller a droite, a gauche,
    devant et derriere, car si on arrive ici c'est
    que U est plus grand que la proba d'aller a
    gauche ou a droite)
  S[Mar2[1,i], Mar2[2,i] + K] = beta * S[Mar2[1,i],
      Mar2[2,i] + K] + alpha

      #On va devant et on ajoute delta
  Q[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * Q[Mar
      2[1,i], Mar2[2,i] + K] - gamma)
  V[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * V[Mar
      2[1,i], Mar2[2,i] + K] - gamma)
  Mar2[1, i + 1] = Mar2[1, i]
  Mar2[2, i + 1] = Mar2[2, i] + 1
}
else if((v != 1) | (Mar2[2,i] != 1)){
```

```
      #Si U est different de 1 ou que nous ne sommes
      pas tout en bas
    S[Mar2[1,i], Mar2[2,i]] = beta * S[Mar2[1,i], Mar2
        [2,i]] + alpha

      #On va derriere et on enleve delta
    Q[Mar2[1,i], Mar2[2,i]] = max(0, delta * Q[Mar2[1,
        i], Mar2[2,i]] - gamma)
    V[Mar2[1,i], Mar2[2,i]] = max(0, delta * V[Mar2[1,
        i], Mar2[2,i]] - gamma)
    Mar2[1, i + 1] = Mar2[1, i]
    Mar2[2, i + 1] = Mar2[2, i] - 1
  }
  else{

    #Si U est egal a 1 et que nous sommes tout en bas
    ,
    Mar2[1, i + 1] = Mar2[1, i]
                                        #On choisit de
      rester sur place sans changer les poids pour ne
       pas fausser l'aleatoire (Juste pour cette
      iteration).
    Mar2[2, i + 1] = Mar2[2, i]
                                        #Ce cas n'a
      quasiment aucune chance d'arriver mais il faut
      tout de meme l'anticiper
  }

}
else{

  if(v < (e / (e + f + g + h))){        #Si U est plus
       petit que la proba d'aller a gauche (Poid d'
      aller a gauche diviser par la somme d'aller a
      droite, a gauche, devant et derriere)
    R[Mar2[1,i], Mar2[2,i]] = beta * R[Mar2[1,i], Mar2
        [2,i]] + alpha                          #On va
         a gauche et on ajoute delta
    P[Mar2[1,i], Mar2[2,i]] = max(0, delta * P[Mar2[1,
        i], Mar2[2,i]] - gamma)
    U[Mar2[1,i], Mar2[2,i]] = max(0, delta * U[Mar2[1,
        i], Mar2[2,i]] - gamma)
    Mar2[1, i + 1] = Mar2[1, i] - 1
    Mar2[2, i + 1] = Mar2[2, i]
  }
  else if(v < ((e + f) / (e + f + g + h))){    #Si U
      est entre les probas d'aller a gauche et d'aller
      a droite ou a gauche (Poid d'aller a gauche + a
      droite diviser par la somme d'aller a droite, a
      gauche, devant et derriere, car si arrive ici c'
      est que U est plus grand que la proba d'aller a
      gauche)
    R[Mar2[1,i], Mar2[2,i] + K] = beta * R[Mar2[1,i],
```

```
      Mar2[2,i] + K] + alpha
                                                          #
      On va a droite et on enleve delta
  P[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * P[Mar
      2[1,i],  Mar2[2,i] + K] - gamma)
  U[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * U[Mar
      2[1,i],  Mar2[2,i] + K] - gamma)
  Mar2[1, i + 1] = Mar2[1, i] + 1
  Mar2[2, i + 1] = Mar2[2, i]
}
else if(v < ((e + f + g) / (e + f + g + h))){
    #Si U est entre les probas d'aller a gauche ou a
    droite et d'aller a droite ou a gauche ou devant
    (Poid d'aller a gauche + a droite + devant
    diviser par la somme d'aller a droite, a gauche,
    devant et derriere, car si on arrive ici c'est
    que U est plus grand que la proba d'aller a
    gauche ou a droite)
  S[Mar2[1,i], Mar2[2,i] + K] = beta * S[Mar2[1,i],
      Mar2[2,i] + K] + alpha

      #On va devant et on ajoute delta
  Q[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * Q[Mar
      2[1,i], Mar2[2,i] + K] - gamma)
  V[Mar2[1,i], Mar2[2,i] + K] = max(0, delta * V[Mar
      2[1,i], Mar2[2,i] + K] - gamma)
  Mar2[1, i + 1] = Mar2[1, i]
  Mar2[2, i + 1] = Mar2[2, i] + 1
}
else if((v != 1) | (Mar2[2,i] != 1)){

    #Si U est different de 1 ou que nous ne sommes
    pas tout en bas
  S[Mar2[1,i], Mar2[2,i]] = beta * S[Mar2[1,i], Mar2
      [2,i]] + alpha

      #On va derriere et on enleve delta
  Q[Mar2[1,i], Mar2[2,i]] = max(0, delta * Q[Mar2[1,
      i], Mar2[2,i]] - gamma)
  V[Mar2[1,i], Mar2[2,i]] = max(0, delta * V[Mar2[1,
      i], Mar2[2,i]] - gamma)
  Mar2[1, i + 1] = Mar2[1, i]
  Mar2[2, i + 1] = Mar2[2, i] - 1
}
else{

    #Si U est egal a 1 et que nous sommes tout en bas
    ,
  Mar2[1, i + 1] = Mar2[1, i]
                                      #On choisit de
      rester sur place sans changer les poids pour ne
       pas fausser l'aleatoire (Juste pour cette
      iteration).
  Mar2[2, i + 1] = Mar2[2, i]
```

```
                                        #Ce cas n'a
      quasiment aucune chance d'arriver mais il faut
      tout de meme l'anticiper
}

if(u < (a / (a + b + c + d))){        #Si U est plus
     petit que la proba d'aller a gauche (Poid d'
    aller a gauche diviser par la somme d'aller a
    droite, a gauche, devant et derriere)
  P[Mar1[1,i], Mar1[2,i]] = beta * P[Mar1[1,i], Mar1
      [2,i]] + alpha                              #On va
       a gauche et on ajoute delta
  R[Mar1[1,i], Mar1[2,i]] = max(0, delta * R[Mar1[1,
      i], Mar1[2,i]] - gamma)
  U[Mar1[1,i], Mar1[2,i]] = max(0, delta * U[Mar1[1,
      i], Mar1[2,i]] - gamma)
  Mar1[1, i + 1] = Mar1[1, i] - 1
  Mar1[2, i + 1] = Mar1[2, i]
}
else if(u < ((a + b) / (a + b + c + d))){     #Si U
    est entre les probas d'aller a gauche et d'aller
    a droite ou a gauche (Poid d'aller a gauche + a
    droite diviser par la somme d'aller a droite, a
    gauche, devant et derriere, car si arrive ici c'
    est que U est plus grand que la proba d'aller a
    gauche)
  P[Mar1[1,i], Mar1[2,i] + K] = beta * P[Mar1[1,i],
      Mar1[2,i] + K] + alpha
                                                    #
      On va a droite et on enleve delta
  R[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * R[Mar
      1[1,i], Mar1[2,i] + K] - gamma)
  U[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * U[Mar
      1[1,i], Mar1[2,i] + K] - gamma)
  Mar1[1, i + 1] = Mar1[1, i] + 1
  Mar1[2, i + 1] = Mar1[2, i]
}
else if(u < ((a + b + c) / (a + b + c + d))){
    #Si U est entre les probas d'aller a gauche ou a
    droite et d'aller a droite ou a gauche ou devant
    (Poid d'aller a gauche + a droite + devant
    diviser par la somme d'aller a droite, a gauche,
    devant et derriere, car si on arrive ici c'est
    que U est plus grand que la proba d'aller a
    gauche ou a droite)
  Q[Mar1[1,i], Mar1[2,i] + K] = beta * Q[Mar1[1,i],
      Mar1[2,i] + K] + alpha

      #On va devant et on ajoute delta
  S[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * S[Mar
      1[1,i], Mar1[2,i] + K] - gamma)
  V[Mar1[1,i], Mar1[2,i] + K] = max(0, delta * V[Mar
      1[1,i], Mar1[2,i] + K] - gamma)
  Mar1[1, i + 1] = Mar1[1, i]
```

```
        Mar1[2, i + 1] = Mar1[2, i] + 1
      }
      else if((u != 1) | (Mar1[2,i] != 1)){

        #Si U est different de 1 ou que nous ne sommes
        pas tout en bas
        Q[Mar1[1,i], Mar1[2,i]] = beta * Q[Mar1[1,i], Mar1
            [2,i]] + alpha

            #On va derriere et on enleve delta
        S[Mar1[1,i], Mar1[2,i]] = max(0, delta * S[Mar1[1,
            i], Mar1[2,i]] - gamma)
        V[Mar1[1,i], Mar1[2,i]] = max(0, delta * V[Mar1[1,
            i], Mar1[2,i]] - gamma)
        Mar1[1, i + 1] = Mar1[1, i]
        Mar1[2, i + 1] = Mar1[2, i] - 1
      }
      else{

        #Si U est egal a 1 et que nous sommes tout en bas
        ,
        Mar1[1, i + 1] = Mar1[1, i]
                                          #On choisit de
            rester sur place sans changer les poids pour ne
             pas fausser l'aleatoire (Juste pour cette
            iteration).
        Mar1[2, i + 1] = Mar1[2, i]
                                          #Ce cas n'a
            quasiment aucune chance d'arriver mais il faut
            tout de meme l'anticiper
      }

    }

  }
  return(list(Mar1 = Mar1, Mar2 = Mar2, Mar3 = Mar3, P = P,Q
      = Q,  R = R, S = S, U = U, V = V))
}
```

Code source de la fonction MARpop : (Retour à la page)

```
MARpop = function(P, Q, R, S, alpha, beta, gamma, delta, N,
    xdebut, ydebut){
  Mar = matrix(0, nrow = 2, ncol = N)
  Mar[1,1] = xdebut
  Mar[2,1] = ydebut
  K = nrow(P)
  for(i in 1:(N-1)){
    U = runif(1)
                                                    #On
        tire un nombre aleatoire entre 0 et 1
    a = P[Mar[1,i], Mar[2,i]]
```

```
                                                #On cr er des
    variables pour simplifier l' criture
b = P[Mar[1,i], Mar[2,i] + K]
c = Q[Mar[1,i], Mar[2,i] + K]
d = Q[Mar[1,i], Mar[2,i]]
if((a > 10**300) | (b > 10**300) | (c > 10**300) | (d >
    10**300)){
  warning('renforcement␣trop␣important')
  return(list(Mar = matrix(Mar[Mar > 0], nrow = 2, byrow
      = FALSE), P = P, Q = Q, R = R, S = S, i = i))
}
if(U < (a / (a + b + c + d))){        #Si U est plus
    petit que la proba d'aller a gauche (Poid d'aller a
    gauche diviser par la somme d'aller a droite, a
    gauche, devant et derriere)
  P[Mar[1,i], Mar[2,i]] = beta * P[Mar[1,i], Mar[2,i]] +
      alpha                             #On va a gauche et
      on ajoute delta
  R[Mar[1,i], Mar[2,i]] = max(0, delta * R[Mar[1,i], Mar
      [2,i]] - gamma)
  Mar[1, i + 1] = Mar[1, i] - 1
  Mar[2, i + 1] = Mar[2, i]
}
else if(U < ((a + b) / (a + b + c + d))){    #Si U est
    entre les probas d'aller a gauche et d'aller a droite
     ou a gauche (Poid d'aller a gauche + a droite
    diviser par la somme d'aller a droite, a gauche,
    devant et derriere, car si arrive ici c'est que U est
     plus grand que la proba d'aller a gauche)
  P[Mar[1,i], Mar[2,i] + K] = beta * P[Mar[1,i], Mar[2,i
      ] + K] + alpha
                                                      #On
      va a droite et on enleve delta
  R[Mar[1,i], Mar[2,i] + K] = max(0, delta * R[Mar[1,i],
       Mar[2,i] + K] - gamma)
  Mar[1, i + 1] = Mar[1, i] + 1
  Mar[2, i + 1] = Mar[2, i]
}
else if(U < ((a + b + c) / (a + b + c + d))){       #Si
    U est entre les probas d'aller a gauche ou a droite
    et d'aller a droite ou a gauche ou devant (Poid d'
    aller a gauche + a droite + devant diviser par la
    somme d'aller a droite, a gauche, devant et derriere,
     car si on arrive ici c'est que U est plus grand que
    la proba d'aller a gauche ou a droite)
  Q[Mar[1,i], Mar[2,i] + K] = beta * Q[Mar[1,i], Mar[2,i
      ] + K] + alpha

      #On va devant et on ajoute delta
  S[Mar[1,i], Mar[2,i] + K] = max(0, delta * S[Mar[1,i],
       Mar[2,i] + K] - gamma)
  Mar[1, i + 1] = Mar[1, i]
  Mar[2, i + 1] = Mar[2, i] + 1
}
```

```
    else if((U != 1) | (Mar[2,i] != 1)){

       #Si U est different de 1 ou que nous ne sommes pas
       tout en bas
      Q[Mar[1,i], Mar[2,i]] = beta * Q[Mar[1,i], Mar[2,i]] +
          alpha

         #On va derriere et on enleve delta
      S[Mar[1,i], Mar[2,i]] = max(0, delta * S[Mar[1,i], Mar
          [2,i]] - gamma)
      Mar[1, i + 1] = Mar[1, i]
      Mar[2, i + 1] = Mar[2, i] - 1
    }
    else{

       #Si U est egal a 1 et que nous sommes tout en bas,
      Mar[1, i + 1] = Mar[1, i]
                                          #On choisit de
          rester sur place sans changer les poids pour ne pas
           fausser l'aleatoire (Juste pour cette iteration).
      Mar[2, i + 1] = Mar[2, i]
                                          #Ce cas n'a
          quasiment aucune chance d'arriver mais il faut tout
           de meme l'anticiper
    }
  }
  return(list(Mar = Mar, P = P, Q = Q,  R = R, S = S))
}
```

Code source de la paramétrisation Populations : (Retour à la page)

```
#### Initialisation des populations ####


individu = 5   #nous testerons diverses valeurs ici pour le
    nombre d individus simul s (5,10,50)
N = 10000
alpha = 1
beta = 1
delta = 0.8
gamma = 0
K = 250                          #On a une grille de taille
    K*K
debut = floor(K/2)
xdebut = debut
ydebut = debut
xdebut1 = debut
ydebut1 = debut
xdebut2 = debut
ydebut2 = debut


P = matrix(1,nrow=K,ncol=2*K)    #Matrice des mouvements
    horizontaux
for(i in 1:K){
  P[1, i] = 0
  P[K, K+i] = 0
}


Q = matrix(1,nrow=K,ncol=2*K)    #Matrice des mouvements
    verticaux
for(i in 1:K){
  Q[i,1] = 0
  Q[i, 2*K] = 0
}


R = matrix(1,nrow=K,ncol=2*K)    #Matrice des mouvements
    horizontaux
for(i in 1:K){
  R[1, i] = 0
  R[K, K+i] = 0
}


S = matrix(1,nrow=K,ncol=2*K)    #Matrice des mouvements
    verticaux
for(i in 1:K){
  S[i,1] = 0
  S[i, 2*K] = 0
}
```

```
U = matrix(1,nrow=K,ncol=2*K)     #Matrice des mouvements
    verticaux
for(i in 1:K){
  U[1, i] = 0
  U[K, K + i] = 0
}


V = matrix(1,nrow=K,ncol=2*K)     #Matrice des mouvements
    verticaux
for(i in 1:K){
  V[i,1] = 0
  V[i, 2*K] = 0
}


gradient = c("red","yellow","green", "lightblue","darkblue")


X3 = array(0, dim = c(2, N, individu))
X4 = array(0, dim = c(2, N, individu))
X5 = array(0, dim = c(2, N, individu))
X15 = array(0, dim = c(2, N, individu))
X16 = array(0, dim = c(2, N, individu))
X17 = array(0, dim = c(2, N, individu))

X39 = array(0, dim = c(2, N, individu))
X40 = array(0, dim = c(2, N, individu))
X47 = array(0, dim = c(2, N, individu))
X48 = array(0, dim = c(2, N, individu))
X49 = array(0, dim = c(2, N, individu))

X59 = array(0, dim = c(2, N, individu))
X60 = array(0, dim = c(2, N, individu))
X67 = array(0, dim = c(2, N, individu))
X68 = array(0, dim = c(2, N, individu))
X69 = array(0, dim = c(2, N, individu))
```

Code source de la Figure 4.1.1 : (Retour à la page)

```
#### 1 Population ####


for(i in 1:(individu)){
  A = MAR2d(P, Q, alpha, beta, N, debut)
  P = A$P
  Q = A$Q
  X3[1:2, 1:N, i] = A$Mar
}

walk = 1:N
x3 = X3[1, 1:N, 1]
y3 = X3[2, 1:N, 1]
df3 = data.frame(walk = walk, x = x3, y = y3)

for(i in 2:individu){
  xtemp = X3[1, 1:N, i]
  ytemp = X3[2, 1:N, i]
  dftemp = data.frame(walk = walk, x = xtemp, y = ytemp)
  df3 = rbind(df3, dftemp)
}

plot3 = ggplot(df3, aes(x = x, y = y, color = walk)) + geom_
    path() + labs(color = "Time") + scale_colour_gradientn(
    colors = gradient) + theme_minimal() + geom_segment(aes(x
     = 0, y = 0, xend = K, yend = 0)) + geom_segment(aes(x =
    0, y = K, xend = K, yend = K)) + geom_segment(aes(x = 0,
    y = 0, xend = 0, yend = K)) + geom_segment(aes(x = K, y =
     0, xend = K, yend = K))+ labs (title = "Tous␣les␣
    individus" ,x="Ouest␣/␣Est" , y = "Sud␣/␣Nord")
#plot3 = plot3 + geom_density2d(color = "purple")
print(plot3)



x3 = X3[1, 1:N, 1]
y3 = X3[2, 1:N, 1]
df6 <- data.frame(walk = walk, x = x3, y = y3)

x3 = X3[1, 1:N, floor(individu/2)]
y3 = X3[2, 1:N, floor(individu/2)]
df7 = data.frame(walk = walk, x = x3, y = y3)

x3 = X3[1, 1:N, individu]
y3 = X3[2, 1:N, individu]
df8 = data.frame(walk = walk, x = x3, y = y3)


plot6 = ggplot(df6, aes(x = x, y = y, color = walk)) + geom_
    path() + labs(color = "Time") + scale_colour_gradientn(
    colors = gradient) + theme_minimal() + geom_segment(aes(x
     = 0, y = 0, xend = K, yend = 0)) + geom_segment(aes(x =
    0, y = K, xend = K, yend = K)) + geom_segment(aes(x = 0,
```

```
    y = 0, xend = 0, yend = K)) + geom_segment(aes(x = K, y =
     0, xend = K, yend = K))+ labs (title = "Premier␣individu
    " ,x="Ouest␣/␣Est" , y = "Sud␣/␣Nord")
plot7 = ggplot(df7, aes(x = x, y = y, color = walk)) + geom_
    path() + labs(color = "Time") + scale_colour_gradientn(
    colors = gradient) + theme_minimal() + geom_segment(aes(x
     = 0, y = 0, xend = K, yend = 0)) + geom_segment(aes(x =
    0, y = K, xend = K, yend = K)) + geom_segment(aes(x = 0,
    y = 0, xend = 0, yend = K)) + geom_segment(aes(x = K, y =
     0, xend = K, yend = K))+ labs (title = "Individu␣milieu"
     ,x="Ouest␣/␣Est" , y = "Sud␣/␣Nord")
plot8 = ggplot(df8, aes(x = x, y = y, color = walk)) + geom_
    path() + labs(color = "Time") + scale_colour_gradientn(
    colors = gradient) + theme_minimal() + geom_segment(aes(x
     = 0, y = 0, xend = K, yend = 0)) + geom_segment(aes(x =
    0, y = K, xend = K, yend = K)) + geom_segment(aes(x = 0,
    y = 0, xend = 0, yend = K)) + geom_segment(aes(x = K, y =
     0, xend = K, yend = K))+ labs (title = "Dernier␣individu
    " ,x="Ouest␣/␣Est" , y = "Sud␣/␣Nord")

grid.arrange(plot6, plot7, plot8, plot3, ncol=2, nrow = 2)
```

Code source de la Figure 4.1.2 : (Retour à la page)

```
individu = 50
alpha=3

for(i in 1:(individu)){
  A = MAR2d(P, Q, alpha, beta, N, debut)
  P = A$P
  Q = A$Q
  X3[1:2, 1:N, i] = A$Mar
}

walk = 1:N
x3 = X3[1, 1:N, 1]
y3 = X3[2, 1:N, 1]
df3 = data.frame(walk = walk, x = x3, y = y3)

for(i in 2:individu){
  xtemp = X3[1, 1:N, i]
  ytemp = X3[2, 1:N, i]
  dftemp = data.frame(walk = walk, x = xtemp, y = ytemp)
  df3 = rbind(df3, dftemp)
}

plot3 = ggplot(df3, aes(x = x, y = y, color = walk)) + geom_
    path() + labs(color = "Time") + scale_colour_gradientn(
    colors = gradient) + theme_minimal() + geom_segment(aes(x
     = 0, y = 0, xend = K, yend = 0)) + geom_segment(aes(x =
    0, y = K, xend = K, yend = K)) + geom_segment(aes(x = 0,
    y = 0, xend = 0, yend = K)) + geom_segment(aes(x = K, y =
     0, xend = K, yend = K))+ labs (title = "Tous␣les␣
    individus" ,x="Ouest␣/␣Est" , y = "Sud␣/␣Nord")
#plot3 = plot3 + geom_density2d(color = "purple")
```

```
print(plot3)
```

Code source de la Figure 4.2.1.1 : (Retour à la page)

```
#Pas par pas
for(i in 1:(individu)){
  B = MAR2pop(P, Q, R, S, alpha, beta, gamma, delta, N,
      xdebut, ydebut, xdebut1, ydebut1)
  P = B$P
  Q = B$Q
  R = B$R
  S = B$S
  X4[1:2, 1:N, i] = B$Mar1
  X5[1:2, 1:N, i] = B$Mar2
}

walk = 1:N
x4 = X4[1, 1:N, 1]
y4 = X4[2, 1:N, 1]
df4 = data.frame(walk = walk, x = x4, y = y4)

x5 = X5[1, 1:N, 1]
y5 = X5[2, 1:N, 1]
df5 = data.frame(walk = walk, x = x5, y = y5)

for(i in 2:individu){
  xtemp1 = X4[1, 1:N, i]
  ytemp1 = X4[2, 1:N, i]
  dftemp1 = data.frame(walk = walk, x = xtemp1, y = ytemp1)
  xtemp2 = X5[1, 1:N, i]
  ytemp2 = X5[2, 1:N, i]
  dftemp2 = data.frame(walk = walk, x = xtemp2, y = ytemp2)
  df4 = rbind(df4, dftemp1)
  df5 = rbind(df5, dftemp2)
}


df4$pop = 1
df5$pop = 2
df79 = rbind(df4,df5)


plot79 = ggplot(df79, aes(x = x, y = y, color = pop)) + geom
    _path() + labs(color = "Population") + scale_colour_
    gradientn(colors = c("red","green"))  + theme_minimal() +
     geom_segment(aes(x = 0, y = 0, xend = K, yend = 0)) +
    geom_segment(aes(x = 0, y = K, xend = K, yend = K)) +
    geom_segment(aes(x = 0, y = 0, xend = 0, yend = K)) +
    geom_segment(aes(x = K, y = 0, xend = K, yend = K))+ labs
     (title = "Deux␣populations" ,x="Ouest␣/␣Est" , y = "Sud␣
    /␣Nord")


plot4 = ggplot(df4, aes(x = x, y = y, color = walk)) + geom_
    path() + labs(color = "Time") + scale_colour_gradientn(
```

```
    colors = gradient) + theme_minimal() + geom_segment(aes(x
     = 0, y = 0, xend = K, yend = 0)) + geom_segment(aes(x =
    0, y = K, xend = K, yend = K)) + geom_segment(aes(x = 0,
    y = 0, xend = 0, yend = K)) + geom_segment(aes(x = K, y =
     0, xend = K, yend = K))+ labs (title = "Premi re␣
    population" ,x="Ouest␣/␣Est" , y = "Sud␣/␣Nord")
#plot4 = plot4 + geom_density2d(color = "purple")

plot5 = ggplot(df5, aes(x = x, y = y, color = walk)) + geom_
    path() + labs(color = "Time") + scale_colour_gradientn(
    colors = gradient) + theme_minimal() + geom_segment(aes(x
     = 0, y = 0, xend = K, yend = 0)) + geom_segment(aes(x =
    0, y = K, xend = K, yend = K)) + geom_segment(aes(x = 0,
    y = 0, xend = 0, yend = K)) + geom_segment(aes(x = K, y =
     0, xend = K, yend = K))+ labs (title = "Deuxi me␣
    population" ,x="Ouest␣/␣Est" , y = "Sud␣/␣Nord")
#plot5 = plot5 + geom_density2d(color = "pink")

grid.arrange(plot4, plot5, ncol=2)
plot79
```

Code source de la Figure 4.2.2.1 : (Retour à la page)

```
#Individu par individu
for(i in 1:(individu)){
  B = MARpop(P, Q, R, S, alpha, beta, gamma, delta, N,
      xdebut, ydebut)
  P = B$P
  Q = B$Q
  R = B$R
  S = B$S
  X59[1:2, 1:N, i] = B$Mar
  B = MARpop(R, S, P, Q, alpha, beta, gamma, delta, N,
      xdebut1, ydebut1)
  P = B$R
  Q = B$S
  R = B$P
  S = B$Q
  X60[1:2, 1:N, i] = B$Mar
}


walk = 1:N
x59 = X59[1, 1:N, 1]
y59 = X59[2, 1:N, 1]
df59 = data.frame(walk = walk, x = x59, y = y59)

x60 = X60[1, 1:N, 1]
y60 = X60[2, 1:N, 1]
df60 = data.frame(walk = walk, x = x60, y = y60)

for(i in 2:individu){
  xtemp1 = X59[1, 1:N, i]
  ytemp1 = X59[2, 1:N, i]
  dftemp1 = data.frame(walk = walk, x = xtemp1, y = ytemp1)
```

```
  xtemp2 = X60[1, 1:N, i]
  ytemp2 = X60[2, 1:N, i]
  dftemp2 = data.frame(walk = walk, x = xtemp2, y = ytemp2)
  df59 = rbind(df59, dftemp1)
  df60 = rbind(df60, dftemp2)
}


df59$pop = 1
df60$pop = 2
df81 = rbind(df59,df60)


plot81 = ggplot(df81, aes(x = x, y = y, color = pop)) + geom
    _path() + labs(color = "Population") + scale_colour_
    gradientn(colors = c("red","green"))  + theme_minimal() +
     geom_segment(aes(x = 0, y = 0, xend = K, yend = 0)) +
    geom_segment(aes(x = 0, y = K, xend = K, yend = K)) +
    geom_segment(aes(x = 0, y = 0, xend = 0, yend = K)) +
    geom_segment(aes(x = K, y = 0, xend = K, yend = K))+ labs
     (title = "Deux␣populations" ,x="Ouest␣/␣Est" , y = "Sud␣
    /␣Nord")



plot59 = ggplot(df59, aes(x = x, y = y, color = walk)) +
    geom_path() + labs(color = "Time") + scale_colour_
    gradientn(colors = gradient) + theme_minimal() + geom_
    segment(aes(x = 0, y = 0, xend = K, yend = 0)) + geom_
    segment(aes(x = 0, y = K, xend = K, yend = K)) + geom_
    segment(aes(x = 0, y = 0, xend = 0, yend = K)) + geom_
    segment(aes(x = K, y = 0, xend = K, yend = K))+ labs (
    title = "Premi re␣population" ,x="Ouest␣/␣Est" , y = "
    Sud␣/␣Nord")
#plot4 = plot4 + geom_density2d(color = "purple")

plot60 = ggplot(df60, aes(x = x, y = y, color = walk)) +
    geom_path() + labs(color = "Time") + scale_colour_
    gradientn(colors = gradient) + theme_minimal() + geom_
    segment(aes(x = 0, y = 0, xend = K, yend = 0)) + geom_
    segment(aes(x = 0, y = K, xend = K, yend = K)) + geom_
    segment(aes(x = 0, y = 0, xend = 0, yend = K)) + geom_
    segment(aes(x = K, y = 0, xend = K, yend = K))+ labs (
    title = "Deuxi me␣population" ,x="Ouest␣/␣Est" , y = "
    Sud␣/␣Nord")
#plot5 = plot5 + geom_density2d(color = "purple")

grid.arrange(plot59, plot60, ncol=2)
plot81
```

Code source de la Figure 4.2.3.1 : (Retour à la page)

```
#Population par population

for(i in 1:(individu)){
```

```
  B = MARpop(P, Q, R, S, alpha, beta, gamma, delta, N,
      xdebut, ydebut)
  P = B$P
  Q = B$Q
  R = B$R
  S = B$S
  X39[1:2, 1:N, i] = B$Mar
}

for(i in 1:(individu)){
  B = MARpop(R, S, P, Q, alpha, beta, gamma, delta, N,
      xdebut1, ydebut1)
  P = B$R
  Q = B$S
  R = B$P
  S = B$Q
  X40[1:2, 1:N, i] = B$Mar
}

walk = 1:N
x39 = X39[1, 1:N, 1]
y39 = X39[2, 1:N, 1]
df39 = data.frame(walk = walk, x = x39, y = y39)

x40 = X40[1, 1:N, 1]
y40 = X40[2, 1:N, 1]
df40 = data.frame(walk = walk, x = x40, y = y40)

for(i in 2:individu){
  xtemp1 = X39[1, 1:N, i]
  ytemp1 = X39[2, 1:N, i]
  dftemp1 = data.frame(walk = walk, x = xtemp1, y = ytemp1)
  xtemp2 = X40[1, 1:N, i]
  ytemp2 = X40[2, 1:N, i]
  dftemp2 = data.frame(walk = walk, x = xtemp2, y = ytemp2)
  df39 = rbind(df39, dftemp1)
  df40 = rbind(df40, dftemp2)
}


df39$pop = 1
df40$pop = 2
df80 = rbind(df39,df40)


plot80 = ggplot(df80, aes(x = x, y = y, color = pop)) + geom
    _path() + labs(color = "Population") + scale_colour_
    gradientn(colors = c("red","green"))  + theme_minimal() +
     geom_segment(aes(x = 0, y = 0, xend = K, yend = 0)) +
    geom_segment(aes(x = 0, y = K, xend = K, yend = K)) +
    geom_segment(aes(x = 0, y = 0, xend = 0, yend = K)) +
    geom_segment(aes(x = K, y = 0, xend = K, yend = K))+ labs
     (title = "Les␣deux␣populations" ,x="Ouest␣/␣Est" , y = "
    Sud␣/␣Nord")
```

```
plot39 = ggplot(df39, aes(x = x, y = y, color = walk)) +
    geom_path() + labs(color = "Time") + scale_colour_
    gradientn(colors = gradient) + theme_minimal() + geom_
    segment(aes(x = 0, y = 0, xend = K, yend = 0)) + geom_
    segment(aes(x = 0, y = K, xend = K, yend = K)) + geom_
    segment(aes(x = 0, y = 0, xend = 0, yend = K)) + geom_
    segment(aes(x = K, y = 0, xend = K, yend = K))+ labs (
    title = "Premi re␣population" ,x="Ouest␣/␣Est" , y = "
    Sud␣/␣Nord")
#plot4 = plot4 + geom_density2d(color = "purple")

plot40 = ggplot(df40, aes(x = x, y = y, color = walk)) +
    geom_path() + labs(color = "Time") + scale_colour_
    gradientn(colors = gradient) + theme_minimal() + geom_
    segment(aes(x = 0, y = 0, xend = K, yend = 0)) + geom_
    segment(aes(x = 0, y = K, xend = K, yend = K)) + geom_
    segment(aes(x = 0, y = 0, xend = 0, yend = K)) + geom_
    segment(aes(x = K, y = 0, xend = K, yend = K))+ labs (
    title = "Deuxi me␣population" ,x="Ouest␣/␣Est" , y = "
    Sud␣/␣Nord")
#plot5 = plot5 + geom_density2d(color = "purple")

grid.arrange(plot39, plot40, ncol=2)
plot80
```

Code source de la Figure 4.3.1.1 : (Retour à la page)

```
#Pas par pas
for(i in 1:(individu)){
  C = MAR3pop(P, Q, R, S, U, V, alpha, beta, gamma, delta, N
      , xdebut, ydebut, xdebut1, ydebut1, xdebut2, ydebut2)
  P = C$P
  Q = C$Q
  R = C$R
  S = C$S
  U = C$U
  V = C$V
  X15[1:2, 1:N, i] = C$Mar1
  X16[1:2, 1:N, i] = C$Mar2
  X17[1:2, 1:N, i] = C$Mar3
}

walk = 1:N
x15 = X15[1, 1:N, 1]
y15 = X15[2, 1:N, 1]
df15 = data.frame(walk = walk, x = x15, y = y15)

x16 = X16[1, 1:N, 1]
y16 = X16[2, 1:N, 1]
df16 = data.frame(walk = walk, x = x16, y = y16)

x17 = X17[1, 1:N, 1]
```

```
y17 = X17[2, 1:N, 1]
df17 = data.frame(walk = walk, x = x17, y = y17)

for(i in 2:individu){
  xtemp1 = X15[1, 1:N, i]
  ytemp1 = X15[2, 1:N, i]
  dftemp1 = data.frame(walk = walk, x = xtemp1, y = ytemp1)
  xtemp2 = X16[1, 1:N, i]
  ytemp2 = X16[2, 1:N, i]
  dftemp2 = data.frame(walk = walk, x = xtemp2, y = ytemp2)
  xtemp3 = X17[1, 1:N, i]
  ytemp3 = X17[2, 1:N, i]
  dftemp3 = data.frame(walk = walk, x = xtemp3, y = ytemp3)
  df15 = rbind(df15, dftemp1)
  df16 = rbind(df16, dftemp2)
  df17 = rbind(df17, dftemp3)
}


df15$pop = 1
df16$pop = 2
df17$pop = 3
df82 = rbind(df15,df16,df17)


plot82 = ggplot(df82, aes(x = x, y = y, color = pop)) + geom
    _path() + labs(color = "Population") + scale_colour_
    gradientn(colors = c("red","green","blue"))  + theme_
    minimal() + geom_segment(aes(x = 0, y = 0, xend = K, yend
     = 0)) + geom_segment(aes(x = 0, y = K, xend = K, yend =
    K)) + geom_segment(aes(x = 0, y = 0, xend = 0, yend = K))
     + geom_segment(aes(x = K, y = 0, xend = K, yend = K))+
    labs (title = "Toutes les populations" ,x="Ouest / Est" ,
     y = "Sud / Nord")


plot15 = ggplot(df15, aes(x = x, y = y, color = walk)) +
    geom_path() + labs(color = "Time") + scale_colour_
    gradientn(colors = gradient) + theme_minimal() + geom_
    segment(aes(x = 0, y = 0, xend = K, yend = 0)) + geom_
    segment(aes(x = 0, y = K, xend = K, yend = K)) + geom_
    segment(aes(x = 0, y = 0, xend = 0, yend = K)) + geom_
    segment(aes(x = K, y = 0, xend = K, yend = K))+ labs (
    title = "Premi re population" ,x="Ouest / Est" , y = "
    Sud / Nord")
#plot4 = plot4 + geom_density2d(color = "purple")

plot16 = ggplot(df16, aes(x = x, y = y, color = walk)) +
    geom_path() + labs(color = "Time") + scale_colour_
    gradientn(colors = gradient) + theme_minimal() + geom_
    segment(aes(x = 0, y = 0, xend = K, yend = 0)) + geom_
    segment(aes(x = 0, y = K, xend = K, yend = K)) + geom_
    segment(aes(x = 0, y = 0, xend = 0, yend = K)) + geom_
    segment(aes(x = K, y = 0, xend = K, yend = K))+ labs (
```

```
    title = "Deuxi me␣population" ,x="Ouest␣/␣Est" , y = "
    Sud␣/␣Nord")
#plot5 = plot5 + geom_density2d(color = "purple")

plot17 = ggplot(df17, aes(x = x, y = y, color = walk)) +
    geom_path() + labs(color = "Time") + scale_colour_
    gradientn(colors = gradient) + theme_minimal() + geom_
    segment(aes(x = 0, y = 0, xend = K, yend = 0)) + geom_
    segment(aes(x = 0, y = K, xend = K, yend = K)) + geom_
    segment(aes(x = 0, y = 0, xend = 0, yend = K)) + geom_
    segment(aes(x = K, y = 0, xend = K, yend = K))+ labs (
    title = "Troisi me␣population" ,x="Ouest␣/␣Est" , y = "
    Sud␣/␣Nord")
#plot4 = plot4 + geom_density2d(color = "purple")


grid.arrange(plot15, plot16, plot17, ncol=3)
plot82
```

Code source de la Figure 4.3.2.1 : (Retour à la page)

```
for(i in 1:(individu)){
  C = MARpopBis(P, Q, R, S, U, V, alpha, beta, gamma, delta,
      N, xdebut, ydebut)
  P = C$P
  Q = C$Q
  R = C$R
  S = C$S
  U = C$U
  V = C$V
  X67[1:2, 1:N, i] = C$Mar

  C = MARpopBis(R, S, U, V, P, Q, alpha, beta, gamma, delta,
      N, xdebut1, ydebut1)
  R = C$P
  S = C$Q
  U = C$R
  V = C$S
  P = C$U
  Q = C$V
  X68[1:2, 1:N, i] = C$Mar

  C = MARpopBis(U, V, P, Q, R, S, alpha, beta, gamma, delta,
      N, xdebut2, ydebut2)
  U = C$P
  V = C$Q
  P = C$R
  Q = C$S
  R = C$U
  S = C$V
  X69[1:2, 1:N, i] = C$Mar
}


walk = 1:N
```

```
x67 = X67[1, 1:N, 1]
y67 = X67[2, 1:N, 1]
df67 = data.frame(walk = walk, x = x67, y = y67)

x68 = X68[1, 1:N, 1]
y68 = X68[2, 1:N, 1]
df68 = data.frame(walk = walk, x = x68, y = y68)

x69 = X69[1, 1:N, 1]
y69 = X69[2, 1:N, 1]
df69 = data.frame(walk = walk, x = x69, y = y69)

for(i in 2:individu){
  xtemp1 = X67[1, 1:N, i]
  ytemp1 = X67[2, 1:N, i]
  dftemp1 = data.frame(walk = walk, x = xtemp1, y = ytemp1)
  xtemp2 = X68[1, 1:N, i]
  ytemp2 = X68[2, 1:N, i]
  dftemp2 = data.frame(walk = walk, x = xtemp2, y = ytemp2)
  xtemp3 = X69[1, 1:N, i]
  ytemp3 = X69[2, 1:N, i]
  dftemp3 = data.frame(walk = walk, x = xtemp3, y = ytemp3)
  df67 = rbind(df67, dftemp1)
  df68 = rbind(df68, dftemp2)
  df69 = rbind(df69, dftemp3)
}


df67$pop = 1
df68$pop = 2
df69$pop = 3
df84= rbind(df67,df68,df69)


plot84 = ggplot(df84, aes(x = x, y = y, color = pop)) + geom
    _path() + labs(color = "Population") + scale_colour_
    gradientn(colors = c("red","green","blue"))  + theme_
    minimal() + geom_segment(aes(x = 0, y = 0, xend = K, yend
     = 0)) + geom_segment(aes(x = 0, y = K, xend = K, yend =
    K)) + geom_segment(aes(x = 0, y = 0, xend = 0, yend = K))
     + geom_segment(aes(x = K, y = 0, xend = K, yend = K))+
    labs (title = "Toutes les populations" ,x="Ouest / Est" ,
     y = "Sud / Nord")


plot67 = ggplot(df67, aes(x = x, y = y, color = walk)) +
    geom_path() + labs(color = "Time") + scale_colour_
    gradientn(colors = gradient) + theme_minimal() + geom_
    segment(aes(x = 0, y = 0, xend = K, yend = 0)) + geom_
    segment(aes(x = 0, y = K, xend = K, yend = K)) + geom_
    segment(aes(x = 0, y = 0, xend = 0, yend = K)) + geom_
    segment(aes(x = K, y = 0, xend = K, yend = K))+ labs (
    title = "Population A" ,x="Ouest / Est" , y = "Sud / Nord
    ")
```

```
#plot4 = plot4 + geom_density2d(color = "purple")

plot68 = ggplot(df68, aes(x = x, y = y, color = walk)) +
    geom_path() + labs(color = "Time") + scale_colour_
    gradientn(colors = gradient) + theme_minimal() + geom_
    segment(aes(x = 0, y = 0, xend = K, yend = 0)) + geom_
    segment(aes(x = 0, y = K, xend = K, yend = K)) + geom_
    segment(aes(x = 0, y = 0, xend = 0, yend = K)) + geom_
    segment(aes(x = K, y = 0, xend = K, yend = K))+ labs (
    title = "Population␣B" ,x="Ouest␣/␣Est" , y = "Sud␣/␣Nord
    ")
#plot5 = plot5 + geom_density2d(color = "purple")

plot69 = ggplot(df69, aes(x = x, y = y, color = walk)) +
    geom_path() + labs(color = "Time") + scale_colour_
    gradientn(colors = gradient) + theme_minimal() + geom_
    segment(aes(x = 0, y = 0, xend = K, yend = 0)) + geom_
    segment(aes(x = 0, y = K, xend = K, yend = K)) + geom_
    segment(aes(x = 0, y = 0, xend = 0, yend = K)) + geom_
    segment(aes(x = K, y = 0, xend = K, yend = K))+ labs (
    title = "Population␣C" ,x="Ouest␣/␣Est" , y = "Sud␣/␣Nord
    ")
#plot4 = plot4 + geom_density2d(color = "purple")

grid.arrange(plot67, plot68, plot69, ncol=3)
plot84
```

Code source de la Figure 4.3.3.1 : (Retour à la page)

```
#Population par population
for(i in 1:(individu)){
  C = MARpop(P, Q, R, S, alpha, beta, gamma, delta, N,
      xdebut, ydebut)
  P = C$P
  Q = C$Q
  R = C$R
  S = C$S
  X47[1:2, 1:N, i] = C$Mar
}

U = C$R
V = C$S

for(i in 1:(individu)){
  C = MARpop(R, S, U, V, alpha, beta, gamma, delta, N,
      xdebut1, ydebut1)
  R = C$P
  S = C$Q
  U = C$R
  V = C$S
  X48[1:2, 1:N, i] = C$Mar
}

for(i in 1:(individu)){
  C = MARpop(U, V, R, S, alpha, beta, gamma, delta, N,
```

```
      xdebut2, ydebut2)
  U = C$P
  V = C$Q
  X49[1:2, 1:N, i] = C$Mar
}

walk = 1:N
x47 = X47[1, 1:N, 1]
y47 = X47[2, 1:N, 1]
df47 = data.frame(walk = walk, x = x47, y = y47)

x48 = X48[1, 1:N, 1]
y48 = X48[2, 1:N, 1]
df48 = data.frame(walk = walk, x = x48, y = y48)

x49 = X49[1, 1:N, 1]
y49 = X49[2, 1:N, 1]
df49 = data.frame(walk = walk, x = x49, y = y49)

for(i in 2:individu){
  xtemp1 = X47[1, 1:N, i]
  ytemp1 = X47[2, 1:N, i]
  dftemp1 = data.frame(walk = walk, x = xtemp1, y = ytemp1)
  xtemp2 = X48[1, 1:N, i]
  ytemp2 = X48[2, 1:N, i]
  dftemp2 = data.frame(walk = walk, x = xtemp2, y = ytemp2)
  xtemp3 = X49[1, 1:N, i]
  ytemp3 = X49[2, 1:N, i]
  dftemp3 = data.frame(walk = walk, x = xtemp3, y = ytemp3)
  df47 = rbind(df47, dftemp1)
  df48 = rbind(df48, dftemp2)
  df49 = rbind(df49, dftemp3)
}



df47$pop = 1
df48$pop = 2
df49$pop = 3
df83 = rbind(df47,df48,df49)


plot83 = ggplot(df83, aes(x = x, y = y, color = pop)) + geom
    _path() + labs(color = "Population") + scale_colour_
    gradientn(colors = c("red","green","blue"))  + theme_
    minimal() + geom_segment(aes(x = 0, y = 0, xend = K, yend
     = 0)) + geom_segment(aes(x = 0, y = K, xend = K, yend =
    K)) + geom_segment(aes(x = 0, y = 0, xend = 0, yend = K))
     + geom_segment(aes(x = K, y = 0, xend = K, yend = K))+
    labs (title = "Toutes les populations" ,x="Ouest / Est" ,
     y = "Sud / Nord")
```

```
plot47 = ggplot(df47, aes(x = x, y = y, color = walk)) +
    geom_path() + labs(color = "Time") + scale_colour_
    gradientn(colors = gradient) + theme_minimal() + geom_
    segment(aes(x = 0, y = 0, xend = K, yend = 0)) + geom_
    segment(aes(x = 0, y = K, xend = K, yend = K)) + geom_
    segment(aes(x = 0, y = 0, xend = 0, yend = K)) + geom_
    segment(aes(x = K, y = 0, xend = K, yend = K))+ labs (
    title = "Pop␣A" ,x="Ouest␣/␣Est" , y = "Sud␣/␣Nord")
#plot4 = plot4 + geom_density2d(color = "purple")

plot48 = ggplot(df48, aes(x = x, y = y, color = walk)) +
    geom_path() + labs(color = "Time") + scale_colour_
    gradientn(colors = gradient) + theme_minimal() + geom_
    segment(aes(x = 0, y = 0, xend = K, yend = 0)) + geom_
    segment(aes(x = 0, y = K, xend = K, yend = K)) + geom_
    segment(aes(x = 0, y = 0, xend = 0, yend = K)) + geom_
    segment(aes(x = K, y = 0, xend = K, yend = K))+ labs (
    title = "Pop␣B" ,x="Ouest␣/␣Est" , y = "Sud␣/␣Nord")
#plot5 = plot5 + geom_density2d(color = "purple")

plot49 = ggplot(df49, aes(x = x, y = y, color = walk)) +
    geom_path() + labs(color = "Time") + scale_colour_
    gradientn(colors = gradient) + theme_minimal() + geom_
    segment(aes(x = 0, y = 0, xend = K, yend = 0)) + geom_
    segment(aes(x = 0, y = K, xend = K, yend = K)) + geom_
    segment(aes(x = 0, y = 0, xend = 0, yend = K)) + geom_
    segment(aes(x = K, y = 0, xend = K, yend = K))+ labs (
    title = "Pop␣C" ,x="Ouest␣/␣Est" , y = "Sud␣/␣Nord")
#plot4 = plot4 + geom_density2d(color = "purple")

plot83
grid.arrange(plot47, plot48, plot49, ncol=3)
```

Code source de la Figure 4.4.1.1 : (Retour à la page)

```
#### Points d part diff rents #####
xdebut = 90
ydebut = 90
xdebut1 = 160
ydebut1 = 160

#Population par population

for(i in 1:(individu)){
  B = MARpop(P, Q, R, S, alpha, beta, gamma, delta, N,
      xdebut, ydebut)
  P = B$P
  Q = B$Q
  R = B$R
  S = B$S
  X39[1:2, 1:N, i] = B$Mar
}

for(i in 1:(individu)){
```

```
  B = MARpop(R, S, P, Q, alpha, beta, gamma, delta, N,
      xdebut1, ydebut1)
  P = B$R
  Q = B$S
  R = B$P
  S = B$Q
  X40[1:2, 1:N, i] = B$Mar
}

walk = 1:N
x39 = X39[1, 1:N, 1]
y39 = X39[2, 1:N, 1]
df39 = data.frame(walk = walk, x = x39, y = y39)

x40 = X40[1, 1:N, 1]
y40 = X40[2, 1:N, 1]
df40 = data.frame(walk = walk, x = x40, y = y40)

for(i in 2:individu){
  xtemp1 = X39[1, 1:N, i]
  ytemp1 = X39[2, 1:N, i]
  dftemp1 = data.frame(walk = walk, x = xtemp1, y = ytemp1)
  xtemp2 = X40[1, 1:N, i]
  ytemp2 = X40[2, 1:N, i]
  dftemp2 = data.frame(walk = walk, x = xtemp2, y = ytemp2)
  df39 = rbind(df39, dftemp1)
  df40 = rbind(df40, dftemp2)
}


df39$pop = 1
df40$pop = 2
df80 = rbind(df39,df40)


plot80 = ggplot(df80, aes(x = x, y = y, color = pop)) + geom
    _path() + labs(color = "Population") + scale_colour_
    gradientn(colors = c("red","green"))  + theme_minimal() +
     geom_segment(aes(x = 0, y = 0, xend = K, yend = 0)) +
    geom_segment(aes(x = 0, y = K, xend = K, yend = K)) +
    geom_segment(aes(x = 0, y = 0, xend = 0, yend = K)) +
    geom_segment(aes(x = K, y = 0, xend = K, yend = K))+ labs
     (title = "Les␣deux␣populations" ,x="Ouest␣/␣Est" , y = "
    Sud␣/␣Nord")



plot39 = ggplot(df39, aes(x = x, y = y, color = walk)) +
    geom_path() + labs(color = "Time") + scale_colour_
    gradientn(colors = gradient) + theme_minimal() + geom_
    segment(aes(x = 0, y = 0, xend = K, yend = 0)) + geom_
    segment(aes(x = 0, y = K, xend = K, yend = K)) + geom_
    segment(aes(x = 0, y = 0, xend = 0, yend = K)) + geom_
    segment(aes(x = K, y = 0, xend = K, yend = K))+ labs (
```

```
    title = "Premi re␣population" ,x="Ouest␣/␣Est" , y = "
    Sud␣/␣Nord")
#plot4 = plot4 + geom_density2d(color = "purple")

plot40 = ggplot(df40, aes(x = x, y = y, color = walk)) +
    geom_path() + labs(color = "Time") + scale_colour_
    gradientn(colors = gradient) + theme_minimal() + geom_
    segment(aes(x = 0, y = 0, xend = K, yend = 0)) + geom_
    segment(aes(x = 0, y = K, xend = K, yend = K)) + geom_
    segment(aes(x = 0, y = 0, xend = 0, yend = K)) + geom_
    segment(aes(x = K, y = 0, xend = K, yend = K))+ labs (
    title = "Deuxi me␣population" ,x="Ouest␣/␣Est" , y = "
    Sud␣/␣Nord")
#plot5 = plot5 + geom_density2d(color = "purple")

plot80
```

Code source de la Figure 4.4.2.1 : (Retour à la page)

```
#### Points d part diff rents #####
xdebut = 100
ydebut = 100
xdebut1 = 150
ydebut1 = 100
xdebut2 = 125
ydebut2 = 150



#Population par population
for(i in 1:(individu)){
  C = MARpop(P, Q, R, S, alpha, beta, gamma, delta, N,
      xdebut, ydebut)
  P = C$P
  Q = C$Q
  R = C$R
  S = C$S
  X47[1:2, 1:N, i] = C$Mar
}

U = C$R
V = C$S

for(i in 1:(individu)){
  C = MARpop(R, S, U, V, alpha, beta, gamma, delta, N,
      xdebut1, ydebut1)
  R = C$P
  S = C$Q
  U = C$R
  V = C$S
  X48[1:2, 1:N, i] = C$Mar
}

for(i in 1:(individu)){
  C = MARpop(U, V, R, S, alpha, beta, gamma, delta, N,
```

```
      xdebut2, ydebut2)
  U = C$P
  V = C$Q
  X49[1:2, 1:N, i] = C$Mar
}

walk = 1:N
x47 = X47[1, 1:N, 1]
y47 = X47[2, 1:N, 1]
df47 = data.frame(walk = walk, x = x47, y = y47)

x48 = X48[1, 1:N, 1]
y48 = X48[2, 1:N, 1]
df48 = data.frame(walk = walk, x = x48, y = y48)

x49 = X49[1, 1:N, 1]
y49 = X49[2, 1:N, 1]
df49 = data.frame(walk = walk, x = x49, y = y49)

for(i in 2:individu){
  xtemp1 = X47[1, 1:N, i]
  ytemp1 = X47[2, 1:N, i]
  dftemp1 = data.frame(walk = walk, x = xtemp1, y = ytemp1)
  xtemp2 = X48[1, 1:N, i]
  ytemp2 = X48[2, 1:N, i]
  dftemp2 = data.frame(walk = walk, x = xtemp2, y = ytemp2)
  xtemp3 = X49[1, 1:N, i]
  ytemp3 = X49[2, 1:N, i]
  dftemp3 = data.frame(walk = walk, x = xtemp3, y = ytemp3)
  df47 = rbind(df47, dftemp1)
  df48 = rbind(df48, dftemp2)
  df49 = rbind(df49, dftemp3)
}



df47$pop = 1
df48$pop = 2
df49$pop = 3
df83 = rbind(df47,df48,df49)


plot83 = ggplot(df83, aes(x = x, y = y, color = pop)) + geom
    _path() + labs(color = "Population") + scale_colour_
    gradientn(colors = c("red","green","blue"))  + theme_
    minimal() + geom_segment(aes(x = 0, y = 0, xend = K, yend
     = 0)) + geom_segment(aes(x = 0, y = K, xend = K, yend =
    K)) + geom_segment(aes(x = 0, y = 0, xend = 0, yend = K))
     + geom_segment(aes(x = K, y = 0, xend = K, yend = K))+
    labs (title = "Toutes les populations" ,x="Ouest / Est" ,
     y = "Sud / Nord")

plot83
```

Références

[Pla03] Michael John Plank. "Cell Based Models of Tumour Angiogenesis". Thèse de doct. University of Leeds, 2003. Chap. 3. URL : https://www.math.canterbury.ac.nz/~m.plank/thesis/chapter3.pdf.

[MR06] Franz Merkl et Silke WW Rolles. "Linearly edge-reinforced random walks". In : *Dynamics & stochastics*. Institute of Mathematical Statistics, 2006, p. 66-77. URL : http://sunmaph6.ma.tum.de/foswiki/pub/M5/Allgemeines/SilkeRollesPublications/LNMS4807.pdf.

[Koz12] Gady Kozma. "Reinforced random walk". In : *arXiv preprint arXiv :1208.0364* (2012). URL : https://arxiv.org/pdf/1208.0364.pdf.

[Bou14] Élodie Bouchet. "Marches aléatoires en environnement aléatoire faiblement elliptique". Thèse de doct. Université Claude Bernard-Lyon I, 2014. URL : https://tel.archives-ouvertes.fr/tel-01057100/document.

[Le 14] Line Le Goff. "Formation spontanée de chemins : des fourmis aux marches aléatoires renforcées". Thèse de doct. Paris 10, 2014. URL : https://bdr.parisnanterre.fr/theses/internet/2014PA100180/2014PA100180.pdf.

[Zen15] Xiaolin Zeng. "Marches aléatoires renforcées et opérateurs de Schrödinger aléatoires". Thèse de doct. Université Claude Bernard-Lyon I, 2015. URL : https://tel.archives-ouvertes.fr/tel-01267712/document.

[KC22] Collet Kilian et Contamin Clément. *Code source.R*. 2022. URL : https://drive.google.com/file/d/12ro1XkfMbE_k0POxzaXicZfZgI_pCP1U/view?usp=sharing.

[Wik] Wikipedia. *Marches Aléatoires*. URL : https://fr.wikipedia.org/wiki/Marche_al%C3%A9atoire. (accessed : 26.03.2022).

[Mél22] Pierre-Loïc Méliot. *Nombre d'états visités par une marche aléatoire*. 2021-2022. URL : https://www.imo.universite-paris-saclay.fr/~meliot/agreg/range.pdf.