Billards Dispersifs

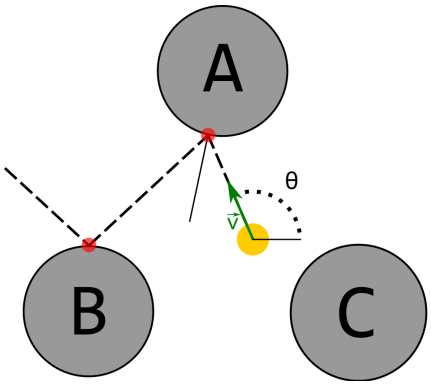
Théo Denorme, Thomas Morand, Alexandre Poncin
Encadré par Damien Thomine

2021

Sommaire

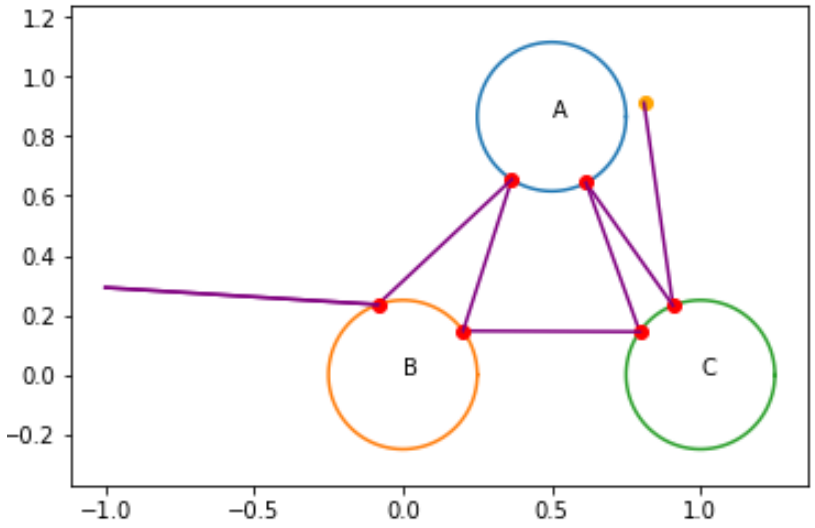
1 Introduction du système étudié

Le sujet de ce projet est d'étudier le comportement d'une particule évoluant dans Q, ensemble fermé de $\mathbb{R}^2$ comportant 3 obstacles circulaires A, B et C de rayon identique a (voir [GR89]). La particule se déplace à vitesse unitaire, constante à l'intérieur de Q et obéit aux lois de la réflexion lorsqu'elle percute ∂Q.



L'espace des phases pour notre particule est donc $\mathrm{P} = \{(q, \vec{v})\}$, dont la position des points est située à l'extérieur des disques, c'est à dire vérifiant : $(x - x_i)^2 + (y - y_i)^2 \geq a^2$, si l'on note (x_i, y_i) le centre du i-ième cercle. La vitesse de notre particule étant unitaire, la vitesse est donnée par $\theta \in]-\pi, \pi]$, tel que $v_x = \cos\theta$ et $v_y = \sin\theta$. Le système est donc à 3 degrès de libertés.

Pour comprendre notre système, on va s'intéresser aux moments où la particule percute les obstacles. Posons alors $\Omega := \{x \in P \,|\, q \in \partial Q,\, \langle \vec{v}, \vec{n}(q) \rangle \geq 0\} = \bigcup_{C \in \{A,B,C\}} \partial C \times [-\pi/2, \pi/2]$ l'ensemble des phases après que la particule ait percuté un des obstacles, et qui est équivalent à la donnée d'un point du bord et d'un angle mesuré par rapport à la normale, dont on note $\vec{n}(q)$ le vecteur normal à ∂Q au point q unitaire rentrant dans Q.

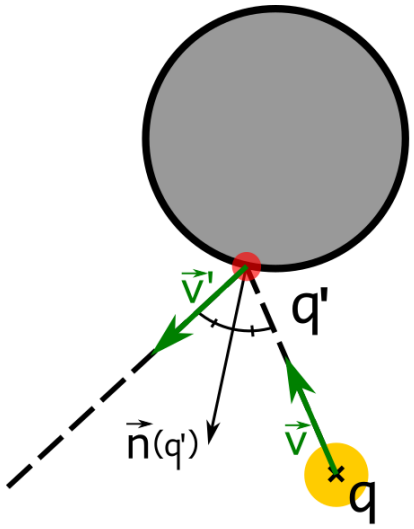


Simulation de rebonds sur les obstales avec python (Voir code A).

Définissons ensuite l'application $\tau : \begin{array}{ccc} \Omega & \to &]0,+\infty] \\ x=(q,v) & \mapsto & \inf\{s>0 : q+s\vec{v}\in\partial Q\} \end{array}$ qui donne la distance parcourue avant la collision avec le prochain obstacle.Nous pouvons définir ensuite sur un sous-ensemble de Ω l'application T qui à une configuration après un choc associe, si elle existe, la configuration au choc suivant :

$$T : \begin{array}{ccc} F & \to & \Omega \\ x=(q,\vec{v}) & \mapsto & (q',v') \end{array} \text{ avec } q' = q+\tau(x)\vec{v} \text{ et } \vec{v'} = \vec{v} - 2\langle\vec{v},\vec{n}(q')\rangle\,\vec{n}(q')$$

Avec $F := \tau^{-1}(]0,+\infty[)$ le sous-ensemble où T est bien défini.



$T(q,\vec{v}) = (q',\vec{v'})$

L'ensemble des trajectoires est alors représenté par les suites $\left(T^k(x)\right)_{m(x)\le k\le M(x)}$ avec $m(x) := \inf\{k\in\mathbb{Z} : T^k(x) \text{ est bien défini}\}$ et $M(x) := \sup\{k\in\mathbb{Z} : T^k(x) \text{ est bien défini}\}$

On définit alors l'espace récurrent K comme le sous ensemble de F des phases pouvant effectuer un nombre infini de rebonds dans le passé comme dans le futur :

$$K := \{x\in F \,|\, m(x) = -\infty \text{ et } M(x) = +\infty\}$$

Définition 1. *Condition de non eclipsité:*
Aucune droite n'intersecte plus de 2 obstacles.

Par la suite, nous nous plaçons dans cette condition de non éclipsité.

Définition 2. *On définit l'ensemble F_C inclus dans Ω avec C un obstacle comme $F_C = \{(q=(q_1,q_2),\vec{v}=(v_1,v_2))\in F$ tel que $|v_2\times(c_1-q_1) - v_1\times(c_2-q_2)| \le a$ et $v_1\times(c_1-q_1) + v_2\times(c_2-q_2)\ge 0\}$*

Proposition 1. *Pour tout obstacle C, pour tout $x\in F$, $x\in F_C$ si et seulement si le prochain obstacle rencontré par la phase x est l'obstacle C.*

Preuve. Soit $(q, \vec{v}) = ((q_1, q_2), (v_1, v_2)) \in F$.
La phase $(q, \vec{v})$ peut rencontrer au plus un obstacle. Elle rencontre l'obstacle C si et seulement si la paramétrisation de sa droite rencontre le bord de C dans un temps futur.
On paramétrise sa trajectoire rectiligne par $t \in \mathbb{R} \mapsto q + t\vec{u}$, ce qui nous permet d'obtenir une condition nécessaire pour avoir une collision avec C :

la droite induite par la trajectoire de la phase $(q, \vec{v})$ rencontre l'obstacle C
$\iff \exists t \in \mathbb{R},\ (c_1 - q_1 - t \times v_1)^2 + (c_2 - q_2 - t \times v_2)^2 = a^2$
$\iff \exists t \in \mathbb{R}, t^2(v_1^2 + v_2^2) - t \times 2((c_1 - q_1)v_1 + (c_2 - q_2)v_2) + (c_1 - q_1)^2 + (c_2 - q_2)^2 - a^2 = 0$
$\iff \exists t \in \mathbb{R}, t^2 - t \times 2((c_1 - q_1)v_1 + (c_2 - q_2)v_2) + (c_1 - q_1)^2 + (c_2 - q_2)^2 - a^2 = 0$ (car $v_1^2 + v_2^2 = 1$)
$\iff (2((c_1 - q_1)v_1 + (c_2 - q_2)v_2))^2 - 4 \times ((c_1 - q_1)^2 + (c_2 - q_2)^2 - a^2) \geq 0$
$\iff (c_1 - q_1)^2(v_1^2 - 1) + (c_2 - q_2)^2(v_2^2 - 1) + 2v_1v_2(c_1 - q_1)(c_2 - q_2) + a^2 \geq 0$
$\iff a^2 \geq (c_1 - q_1)^2 v_2^2 + (c_2 - q_2)^2 v_1^2 - 2v_1v_2(c_1 - q_1)(c_2 - q_2)$ (car $v_1^2 + v_2^2 = 1$)
$\iff a^2 \geq ((c_1 - q_1)v_2 - (c_2 - q_2)v_1)^2$
$\iff a \geq |(c_1 - q_1)v_2 - (c_2 - q_2)v_1|$

La phase rencontrera effectivement l'obstacle C si et seulement si la droite induite par la trajectoire de la phase $(q, \vec{v})$ rencontre l'obstacle C dans le futur (avec la paramétrisation définie ci dessus).
Or comme $q \in Q$ (il n'est pas à l'intérieur de l'obstacle C), si le polynome précédent admet 2 racines, elles seront soient toutes les 2 positives, soient toutes les deux négatives. On peut donc se contenter de calculer la moyenne de ces deux racines pour en obtenir le signe.
On trouve alors la condition $((c_1 - q_1)v_1 + (c_2 - q_2)v_2) \geq 0$. □

Remarque. Cette caractérisation de F_C, pour C un obstacle, nous montre, en particulier, que ces espaces sont 2 à 2 disjoints.

Lemme 1. *Pour tout obstacle C, pour tout $(q, \vec{v}) \in F_C$, $\tau(q, \vec{v})$ est égale à la plus petite racine de $(c_1 - q_1 - tu_1)^2 + (c_2 - q_2 - tu_2)^2 = a^2$, c'est donc une fonction continue sur F_C*

Preuve. Soit C un obstacle, soit $(q, \vec{v}) \in F_C$, $\tau(q, \vec{v}) = \inf\{s > 0 : q + s\vec{v} \in \partial Q\}$, c'est donc la plus petite racine de $(c_1 - q_1 - tu_1)^2 + (c_2 - q_2 - tu_2)^2 = a^2$ qui est $t = ((c_1 - q_1)v_1 + (c_2 - q_2)v_2) - \sqrt{(q_{n,1}, q_{n,2}, u_{n,1}, u_{n,2})^2 - (c_1 - q_1)^2 + (c_2 - q_2)^2 - a^2}$ c'est donc une fonction continue en les variables $(q_{n,1}, q_{n,2}, u_{n,1}, u_{n,2})$ □

Proposition 2. *L'ensemble F est fermé*

Preuve. Soit $(q_n, \vec{u}_n)_{n\in\mathbb{N}} \in F^{\mathbb{N}}$ tel que $(q_n, \vec{u}_n) \underset{n\to+\infty}{\longrightarrow} (q, \vec{u}) \in \Omega$
Quitte à extraire une sous suite, comme le nombre d'obstacle est fini, on peut supposer qu'il existe un obstacle C, tel que pour tout $n \in \mathbb{N}$, $(q_n = (q_{n,1}, q_{n,2}), \vec{u}_n = (u_{n,1}, u_{n,2})) \in F_C$.
Si on note $c = (c_1, c_2)$ le centre de C, alors pour tout $n \in \mathbb{N}$, $\tau(q_n, \vec{v}_n)$ converge vers t_∞ (car τ est continue sur F_C) qui est une racine positive de $(c_1 - q_1 - tu_1)^2 + (c_2 - q_2 - tu_2)^2 = a^2$ donc $(q, \vec{u}) \in F_C \subset F$.
F est donc fermé. □

Lemme 2. *Pour tout obstacle C, F_C est compact.*

Preuve. • Commençons par montrer que F_C est un fermé de F
Soit un obstacle C, $F_C = \{(q = (q_1, q_2), \vec{v}) \in F$ tel que $|v_2 \times (c_1 - q_1) - v_1 \times (c_2 - q_2)| \leq R$ et $v_1 \times (c_1 - q_1) + v_2 \times (c_2 - q_2) \geq 0\} = f^{-1}([0, R]) \bigcap g^{-1}([0, +\infty])$ avec $f(q, \vec{v}) = |v_2 \times (c_1 - q_1) - v_1 \times (c_2 - q_2)|$ et $g(q, \vec{v}) = v_1 \times (c_1 - q_1) + v_2 \times (c_2 - q_2)$, deux fonctions continues sur F.

- F est un fermé et pour tout obstacle C, F_C est un fermé de F. Donc F_C est un fermé pour tout obstacle C.

- Ω est borné donc F_C est un compact pour tout obstacle C.

□

Lemme 3. *τ est continue sur F*

Preuve. τ est continue sur F_C pour tout obstacle C.
Or les espaces F_C sont disjoints et compacts donc ils sont à distance strictement positive les uns des autres.
Donc τ est continue sur F. □

Proposition 3. *T est une fonction continue et bijective sur son image.*

Preuve. • Soit $(q, \vec{v}), (q', \vec{u}) \in F$ tel que $T(q, \vec{v}) = T(q', \vec{u})$. On pose $(q", \vec{w}) = T(q, \vec{v}) = T(q', \vec{u})$
Notons $s = \tau(q, \vec{v})$ et $t = \tau(q', \vec{u})$.
Alors $\vec{w} = \vec{v} - 2\langle \vec{v}, \vec{n}(q") \rangle \vec{n}(q") = \vec{u} - 2\langle \vec{u}, \vec{n}(q") \rangle \vec{n}(q")$.
Donc $\langle \vec{w}, \vec{n}(q") \rangle = -\langle \vec{v}, \vec{n}(q") \rangle = -\langle \vec{u}, \vec{n}(q") \rangle$
Et $\langle \vec{w}, \vec{n}(q")^{\perp} \rangle = \langle \vec{v}, \vec{n}(q")^{\perp} \rangle = \langle \vec{u}, \vec{n}(q")^{\perp} \rangle$ (car $\vec{n}(q")$ est unitaire).
Donc $\vec{v} = \vec{u}$ (car leurs coordonnées sont égales dans la base $(\vec{n}(q"), \vec{n}(q")^{\perp})$).
Et $q" = q + s\vec{v} = q' + t\vec{u}$.
Or $\vec{v} = \vec{u}$ donc $q - q' = (t - s)\vec{u}$.
Supposons $s \neq t$, sans perdre de généralité, prenons $t > s$.
Or par définition de $\tau(y, \vec{u})$, $t = \inf\{\tilde{s} > 0 : q' + \tilde{s}\vec{u} \in \partial Q\}$.
Mais $q' + (t - s)\vec{u} = q \in \partial Q$ et $t - s < t$, c'est donc absurde, ainsi $s = t$ et donc $q = q'$.
T est donc bijective sur son image.

- Soit $(q, \vec{v}) \in F$
$T(q, \vec{v}) = (q', \vec{v'})$ avec $q' = q + \tau(x)\vec{v}$ et $\vec{v'} = \vec{v} - 2\langle \vec{v}, \vec{n}(q') \rangle \vec{n}(q')$
T est continue comme produit et composition de fonctions continues car par le lemme 3, τ est continue.

□

Proposition 4. *L'ensemble récurrent K est métrisable et compact.*

Preuve. K est un sous-ensemble de Ω donc il est métrisable.
Montrons à présent que K est fermé dans Ω:
Soit $(x_n)_{n \in \mathbb{N}} \in K^{\mathbb{N}}$ tel que :

$x_n \underset{n\to\infty}{\longrightarrow} x \in \Omega$
Supposons que $M(x) = k \in \mathbb{N}$, ainsi $T^{k+1}(x)$ n'est pas défini.
T étant continue sur F, elle l'est aussi sur K et donc T^k est continue sur K, ainsi:
$T^k(x_n) \in F \underset{n\to\infty}{\longrightarrow} T^k(x) \in F$ car F est fermé.
Donc $T^{k+1}(x)$ est bien défini, ce qui contredit le fait que $M(x) = k$.
Alors on a bien $M(x) = +\infty$. Supposons maintenant que $m(x) = l \in \mathbb{Z}_-$, ainsi $T^{l-1}(x)$ n'est pas défini.
Pour $x = (q, \vec{v})$, on pose:
$\forall n \in \mathbb{N}, x_n := (q_n, \vec{v}_n)$
$\forall n \in \mathbb{N}, x'_n := (q_n, -\vec{v}_n + 2\langle \vec{v}_n, \vec{n}(q_n)\rangle \vec{n}(q_n))$ (on "renverse le temps")
Nous avons alors ici:
$x'_n \underset{n\to\infty}{\longrightarrow} x' = (q, -\vec{v} + 2\langle \vec{v}, \vec{n}(q)\rangle \vec{n}(q)) \in \Omega$ (par continuité de $x \to x'$)
De plus on a bien :
$\forall n \in \mathbb{N}, T(x_n) = T^{-1}(x'_n)$, ainsi:
$T^{-l}(x'_n) \in F \underset{n\to\infty}{\longrightarrow} T^{-l}(x') \in F$ car F est fermé et T^{-l} continue.
Donc $T^{1-l}(x') = T^{l-1}(x)$ est bien défini, ce qui contredit le fait que m(x)=l.
Alors $m(x) = -\infty$, d'où x appartient à K.
Ainsi K est un fermé de Ω qui est borné dans $\mathbb{R}^3$, donc enfin K est compact.

□

2 Sous-décalages de type fini

Dans la partie suivante, nous allons nous ramener à l'étude des sous-décalges de type fini. Pour cela, on introduit dans un premier temps le vocabulaire des langages, puis la topologie de l'espace des suites indexées par $\mathbb{N}$, on introduit la notion de matrice de transition et on finit en généralisant les résultats aux suites indexées par $\mathbb{Z}$.

Nous allons maintenant étudier la théorie des sous-décalges de type fini (voir [Rob95]) afin de préparer la partie suivante où nous regarderons le système (K, T) à travers Σ_A.

2.1 Espace métrique Σ_p^+

Le problème requiert d'étudier quelques notions de dynamique symbolique. On introduit donc la notion de sous-décalage (voir [Che11]).

Définition 3. *Soit $\mathcal{A}$ un ensemble fini. On dit que $\mathcal{A}$ est l'**alphabet** et que les **lettres** sont les éléments de cet alphabet. Un **mot** de $\mathcal{A}$ est une suite finie de lettres et on note $|w|$ la longueur d'un mot. On note $\mathcal{A}^*$ l'ensemble des mots de $\mathcal{A}$ de longueur finie et $\mathcal{A}^+$ l'ensemble $\mathcal{A}^*$ dépourvu du mot vide. On définit la **concaténation** comme l'opération interne sur $\mathcal{A}^*$, notée comme un produit, qui à $a = a_1a_2...a_k$ et $b = b_1b_2...b_{k'}$ associe $ab = a_1a_2...a_kb_1b_2...b_{k'}$.*

Définition 4. *On munit les espaces $\mathcal{A}^{\mathbb{N}}$ de la topologie produit, $\mathcal{A}$ étant muni de la topologie discrète.*

Soit $i_1 < i_2 < ... < i_k$ des entiers et $a_1, a_2, ..., a_k$ des éléments de $\mathcal{A}$. Le cylindre $[(i_1, a_1), (i_2, a_2), ..., (i_k, a_k)]$ est la partie de $\mathcal{A}^{\mathbb{N}}$ dont les éléments s vérifient $s_{i_1} = a_k, ..., s_{i_k} = a_k$. Si $w = w_1...w_k \in \mathcal{A}^*$, on note $[w, n]$ le cylindre $[(n, w_1), (n+1, w_2), ..., (n+k-1, w_k)]$ et $[w]$ désigne $[w, 0]$.

Lemme 4. *1. Les cylindres sont ouverts et fermés*

2. Les cylindres forment une base de la topologie de $\mathcal{A}^{\mathbb{N}}$.

Preuve. 1. Un cylindre $[(i_1, a_1), (i_2, a_2), ..., (i_k, a_k)]$ est de la forme $\mathcal{A} \times ... \times \mathcal{A} \times \{a_1\} \times \mathcal{A} \times ... \times \mathcal{A} \times \{a_k\} \times \mathcal{A} \times ...$ avec $\mathcal{A}$ et les $\{a_i\}$ ouverts et fermés pour la topologie discrète, donc ce cylindre est ouvert et fermé par définition de la topologie produit.

2. Une base de la topologie produit est $\{\mathcal{A}^{k_1} \times U_1 \times \mathcal{A}^{k_2} \times ... \times U_{n-1} \times \mathcal{A}^{k_n} \times U_n \times \mathcal{A}^{\mathbb{N}}, n, k_1, k_2, .., k_n \in \mathbb{N}, U_1, U_2, U_2, ...U_n \subset \mathcal{A}\}$. Comme les cylindres sont inclus dans cet ensemble (en prenant des singletons pour les U_i), il ne reste qu'à montrer que cette base est engendrée par les cylindres. Si $U = \mathcal{A} \times ... \times U_1 \times ... \times U_n \times ... \times \mathcal{A} \times ...$ avec U_i à l'emplacement k_i dans le produit, alors $U = \bigcup_{(w_1, w_2, ..., w_n) \in \prod_{i=1}^{n} U_i} [(k_1, w_1), ..., (k_n, w_n)]$ est bien une union de cylindres. □

Proposition 5. *Cet espace est métrisable et on peut le munir d'une distance ultramétrique définie pour tout $s, t \in \mathcal{A}^{\mathbb{N}}$ par :*

$$d(s,t) = 2^{-\inf\{k \in \mathbb{N} : s_k \neq t_k\}} \text{ avec } \inf \varnothing = +\infty$$

Preuve. Montrons que d est une distance sur $\mathcal{A}^{\mathbb{N}}$.

Soit $s, t \in \mathcal{A}^{\mathbb{N}}$ tel que $d(s,t) = 0$. Alors $\{k \in \mathbb{N} : s_k \neq t_k\} = \varnothing$ (sinon $2^{-\inf\{k \in \mathbb{N} : s_k \neq t_k\}} > 0$), donc $s = t$. d est bien définie.

Soit $s, t \in \mathcal{A}^{\mathbb{N}}$. Alors $d(s,t) = 2^{-\inf\{k \in \mathbb{N} : s_k \neq t_k\}} = d(t,s)$. Donc d est symétrique.

Montrons maintenant l'inégalité ultratriangulaire :

Soit $u, s, t \in \mathcal{A}^{\mathbb{N}}$. Si $s = t$, alors $d(s,t) = 0 \leq \max(d(s,u), d(u,t))$. Sinon si $s \neq t$, on note $k_0 = \inf\{k \in \mathbb{N} : s_k \neq t_k\}$, tel que $s_{k_0} \neq t_{k_0}$. Mais alors soit $s_{k_0} \neq u_{k_0}$, soit $u_{k_0} \neq t_{k_0}$. Donc soit $\inf\{k \in \mathbb{N} : s_k \neq u_k\} \leq k_0$, soit $\inf\{k \in \mathbb{N} : u_k \neq t_k\} \leq k_0$. On a alors bien $d(s,t) = 2^{-\inf\{k \in \mathbb{N} : s_k \neq t_k\}} \leq \max(d(s,u), d(u,t))$.

Montrons maintenant que la topologie métrique induite par d est la topologie produit. Soit $B(s,r)$ avec $s \in \mathcal{A}^{\mathbb{N}}$ et $r > 0$ une boule ouverte de $\mathcal{A}^{\mathbb{N}}$. Alors

$$\begin{aligned} B(s,r) &= \{t \in \mathcal{A}^{\mathbb{N}} | 2^{-\inf\{k \in \mathbb{N} : s_k \neq t_k\}} < r\} \\ &= \{t \in \mathcal{A}^{\mathbb{N}} | \inf\{k \in \mathbb{N} : s_k \neq t_k\} > -\log_2(r)\} \\ &= \{t \in \mathcal{A}^{\mathbb{N}} | \forall k \leq -\log_2(r), s_k = t_k\} \\ &= [w] \text{ avec } w = s_0 s_1 ... s_{\lfloor -\log_2(r) \rfloor} \end{aligned}$$

Comme les cylindres forment une base de $\mathcal{A}^{\mathbb{N}}$, les ouverts de la topologie métriques sont des ouverts pour la topologie produit.
Soit $C = [(i_1, a_1), (i_2, a_2), ..., (i_k, a_k)]$ avec $i_1 < i_2 < ... < i_k$ et $a_1, ..., a_k$ dans $\mathcal{A}$ un cylindre de $\mathcal{A}^{\mathbb{N}}$. Soit $s \in C$. Alors $B(s, 2^{-i_k}) = \{t \in \mathcal{A}^{\mathbb{N}} | d(s,t) < 2^{-i_k}\} = [s_0 s_1 ... s_{i_k}] \subset C$ donc C est ouvert pour la topologie métrique.

Les deux topologies sont donc équivalentes. □

Proposition 6. *$\mathcal{A}^{\mathbb{N}}$ est compact.*

Preuve. Il s'agit du théorème de Tychonov sur le produit de compacts $(\mathcal{A})_{n\in\mathbb{N}}$ □

Définition 5. *Soit $p \geq 2$. On définit l'espace Σ_p^+ comme l'ensemble $\mathcal{A}^{\mathbb{N}}$ lorsque $\mathcal{A} = \{1, ..., p\}$.*
On munit Σ_p^+ de σ, définie par $\sigma(s) = t$ tel que pour tout $k \in \mathbb{N}$ $t_k = s_{k+1}$.

Proposition 7. *σ est 2-lipschitzienne. En particulier, σ est continue.*

Preuve. Soit $s, t \in \Sigma_+^p$.
Si $s = t$, alors pour tout $k \in \mathbb{N}, \sigma(s)_k = s_{k+1} = t_{k+1} = \sigma(t)_k$. Ainsi $\sigma(s) = \sigma(t)$. Donc $d(\sigma(s), \sigma(t)) = 0 \leq 2 \times d(s,t) = 0$
Sinon $s \neq t$, on note alors $k_0 = \inf\{k \in \mathbb{N} : s_k \neq t_k\}$.
Donc pour tout $k \leq k_0 - 2$, $\sigma(s)_k = s_{k+1} = t_{k+1} = \sigma(t)_k$ (car $k + 1 < k_0$).
Alors $\inf\{k \in \mathbb{N} : \sigma(s_k) \neq \sigma(t_k)\} \geq k_0 - 1$, donc $d(\sigma(s), \sigma(t)) = 2^{-\inf\{k\in\mathbb{N}:\sigma(s_k)\neq\sigma(t_k)\}} \leq 2^{-(k_0-1)} \leq 2 \times 2^{-k_0} \leq 2 \times 2^{-\inf\{k\in\mathbb{N}:s_k\neq t_k\}} \leq 2d(s,t)$ □

2.2 Décalages unilatères

Définition 6. *1. Une matrice de taille $p \times p$ est une **matrice de transition** si ses coefficients sont dans $\{0, 1\}$, si pour tout $i \in \{1, ..., p\}, \sum_{j=1}^p a_{i,j} \geq 1$ (pour chaque état i, il existe au moins une transition autorisée par A de cet état vers un état j, c'est à dire tel que $a_{i,j} = 1$) et si pour tout $j \in \{1, ..., p\}, \sum_{i=1}^p a_{i,j} \geq 1$ (pour chaque état j, il existe au moins une transition autorisée par A d'un état i vers cet état, c'est à dire tel que $a_{i,j} = 1$).*

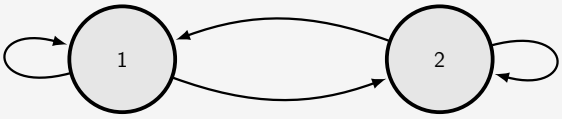
*2. Un mot $s = s_0 s_1 \dots s_n$ de $\{1, \dots, p\}^*$ est dit **admissible** ou **autorisé par A** si $\forall k \in \{0, \dots, n-1\}, a_{s_k, s_{k+1}} = 1$.*

3. Soit $A = (a_{i,j})_{(i,j)\in\{1,...,p\}^2}$ une matrice de transition de taille $p \times p$, on définit alors l'ensemble Σ_A^+ par $\{s \in \Sigma_p^+ : \forall k \in \mathbb{N}, a_{s_k, s_{k+1}} = 1\}$.

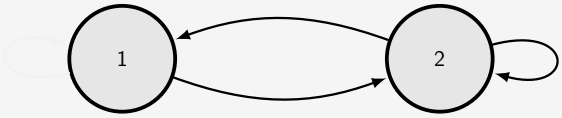
Remarque. L'intérêt de la condition $\sum_{j=1}^p a_{i,j} \geq 1$ dans 1. est de pouvoir compléter par la droite un mot fini autorisé par A en un mot infini de Σ_A, la condition $\sum_{i=1}^p a_{i,j} \geq 1$ permet de compléter un mot par la gauche.

Exemple. $A_1 = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$, $A_2 = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$ et $A_3 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$ sont des matrices de transition. La matrice A_2 est appelée matrice de Fibonacci, et la matrice A_3 est la matrice associée à notre système.

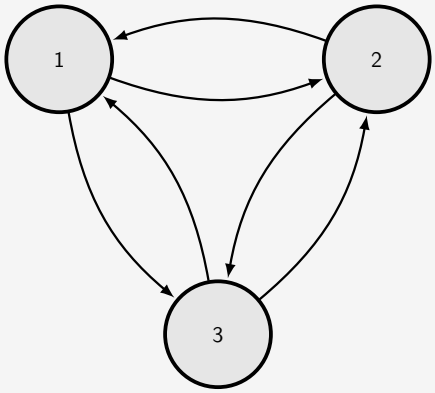
Remarque. Les matrices de transitions se représentent sous la forme d'un graphe dont les sommets sont les états $\{1, ..., p\}$ et les arêtes (orientées) sont les transitions autorisées par A :



Représentation graphique de A_1



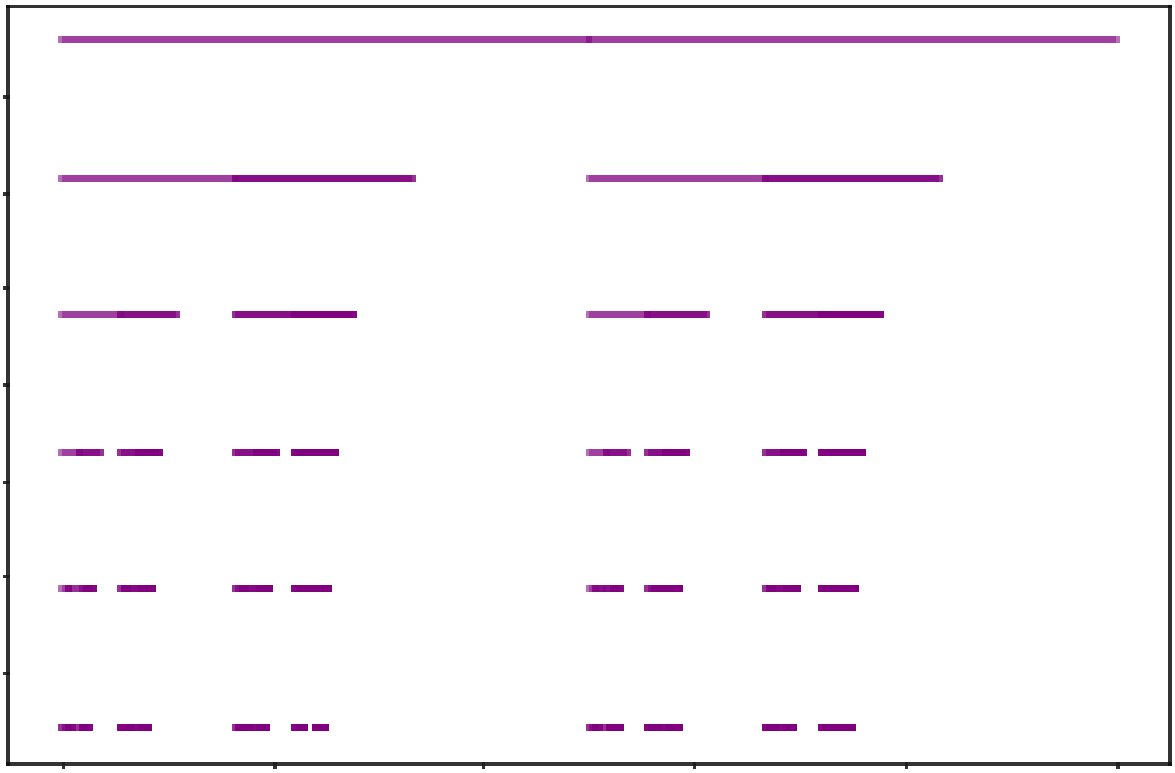
Représentation graphique de A_2



Représentation graphique de A_3

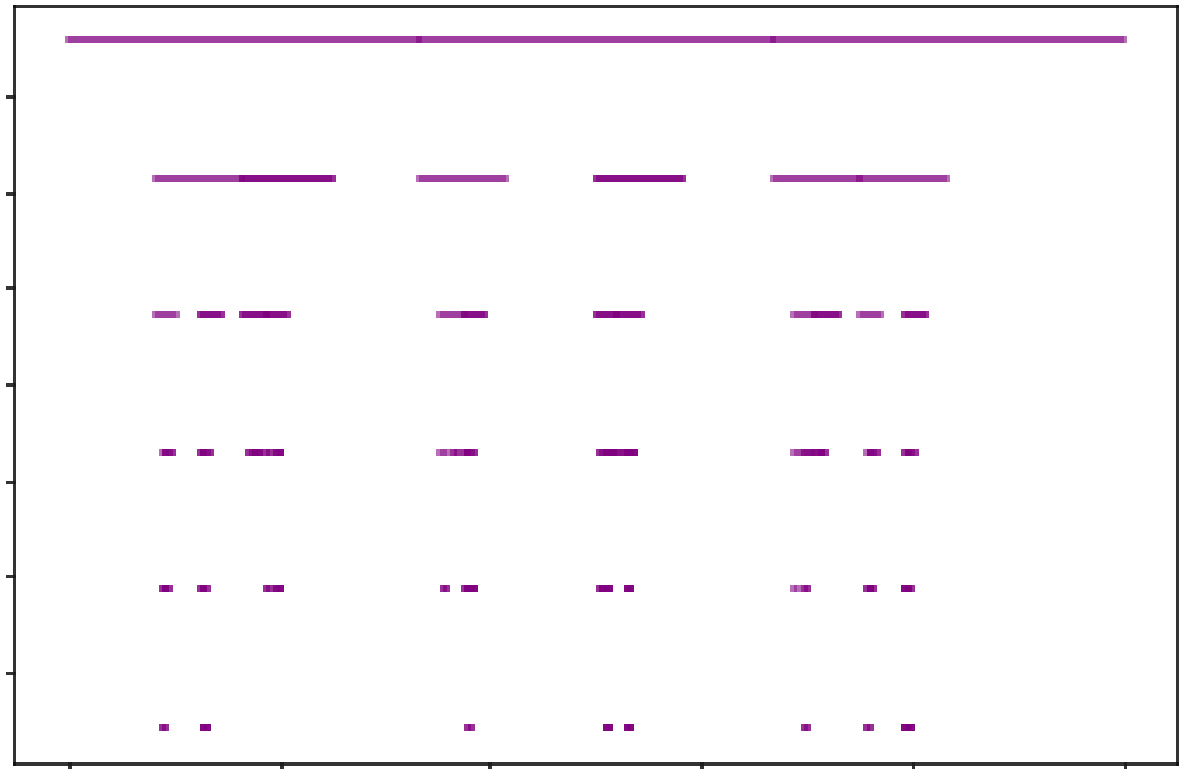
Nous pouvons aussi représenter Σ_A avec un ensemble de Cantor C_A, inclus dans le segment $[0, 1]$. En effet, si A est une matrice de dimension $p \times p$, un élément $(s_n)_{n\in\mathbb{N}}$ de Σ_A peut être codé par la décomposition p-adique d'un élement de $[0, 1]$, autrement dit on définit:

$$C_A = \left\{ \sum_{k=0}^{\infty} \frac{(s_k - 1)}{(p+1)^{k+1}} \,\middle|\, (s_k)_{k\in\mathbb{N}} \in \Sigma_A \right\}$$



Construction de l'espace de Cantor C_{A_1}

Construction de l'espace de Cantor C_{A_2}



Construction de l'espace de Cantor C_{A_3}

Proposition 8. *1.* Σ_A^+ *est fermé dans* Σ_p^+

2. $\sigma(\Sigma_A^+) = \Sigma_A^+$

Preuve. 1. Soit $(s^n)_{n\in\mathbb{N}} \in (\Sigma_A^+)^{\mathbb{N}}$ tel que $\lim\limits_{n\to+\infty} s^n = s \in \Sigma_p^+$. Montrons que $s \in \Sigma_A^+$.
Soit $k \in \mathbb{N}$, il existe $n \in \mathbb{N}$ tel que $d(s^n, s) < 2^{-(k+1)}$ car $\lim\limits_{n\to+\infty} s^n = s$.
Alors en particulier $s_k^n = s_k$ et $s_{k+1}^n = s_{k+1}$ donc $a_{s_k,s_{k+1}} = a_{s_k^n,s_{k+1}^n} = 1$.
Ainsi $s \in \Sigma_A^+$.

2. Soit $t \in \sigma(\Sigma_A^+)$, alors il existe $s \in \Sigma_A^+$ tel que $t = \sigma(s)$.
Donc pour tout $k \in \mathbb{N}, a_{t_k,t_{k+1}} = a_{\sigma(s)_k,\sigma(s)_{k+1}} = a_{s_{k+1},s_{k+2}} = 1$ donc $t \in \Sigma_A^+$.
Réciproquement, soit $s \in \Sigma_A^+$.
Par définition d'une matrice de transition, $\sum_{i=1}^p a_{i,s_0} \geq 1$, donc il existe $i_0 \in \{1, .., p\}$ tel que $a_{i_0,s_0} = 1$.
Posons $t \in \Sigma_A^+$ tel que $t_0 = i_0$ et pour tout $k \in \mathbb{N}^*$, $t_k = s_{k-1}$. t appartient bien à Σ_A^+ car $a_{t_0,t_1} = a_{i_0,s_0} = 1$ et pour tout $k \in \mathbb{N}^*$, $a_{t_k,t_{k+1}} = a_{s_{k-1},s_k} = 1$.
De plus, pour tout $k \in \mathbb{N}$, $\sigma(t)_k = t_{k+1} = s_k$. Donc $\sigma(t) = s$

□

Définition 7. *On définit $\sigma_A : \Sigma_A^+ \to \Sigma_A^+$ comme la restriction de σ à Σ_A^+.*

Proposition 9. *Σ_A^+ est compact*

Preuve. Σ_A^+ est fermé et inclus dans Σ_p^+ compact donc Σ_A^+ est compact. □

2.3 Quelques propriétés sur les matrices de transition

On peut obtenir de l'information sur Σ_A à partir de certaines propriétés de A. On énonce donc ici quelques propositions sur la matrice A pour se familiariser avec l'espace Σ_A. Ces propositions permettent aussi de motiver le choix de la définition de matrice de transition.

Lemme 5. *Supposons que $(A^k)_{ij} = n$. Alors il y a n mots autorisés par A de longueur $k+1$ commençant par i et finissant par j.*

Preuve. Notons $c(k,i,j)$ le nombre de mots de longueur $k+1$ de i à j. Pour i,j dans $\{1,...,p\}$, il suffit donc de montrer par récurrence sur $k \in \mathbb{N}$, $H(k) : \forall i,j \in \{1,...,p\}, c(k,i,j) = (A^k)_{ij}$. Si $k = 1$, alors $c(k,i,j)$ vaut 1 si la transition de i à j est autorisée (le seul mot est ij) et 0 sinon (pas de mot). On a bien $c(k,i,j) = (A^k)_{ij}$. Supposons que la propriété est vraie au rang $k-1 \geq 1$. Alors par calcul matriciel,

$$\begin{aligned}(A^k)_{ij} &= \sum_l (A^{k-1})_{il} A_{lj} \\ &= \sum_l c(k-1,i,l) \times c(1,l,j) \\ &\quad \text{(par hypothèse de récurrence H(k) et aussi H(1))} \\ &= c(k,i,j).\end{aligned}$$ □

Expliquons la dernière égalité : notons $C(k,i,j)$ l'ensemble des mots de longueur k + 1. Alors $C(k,i,j)$ est en bijection avec $\bigsqcup_l C(k-1,i,l) \times C(1,l,j)$ (en associant $w_0 w_1 ... w_k$ à $(w_0 w_1 ... w_{k-1}, w_{k-1} w_k)$), d'où l'égalité des cardinaux.

Proposition 10. *Le nombre de points fixes de σ_A^k est égal à la trace de A^k.*

Preuve. Soit $s \in \Sigma_A$ un point fixe de σ_A^k. Il vérifie donc $\sigma_A^k(s) = s$ et s'écrit donc sous la forme $s = i s_1 s_2 s_3 ... s_{k-1} i s_1 s_2 s_3 ... s_{k-1} ....$ L'ensemble des points fixes de σ_A^k est donc en bijection avec les mots autorisés par A de longueur k + 1 commençant et terminant par la même lettre. Le résultat suit du lemme précédent :

$$\#Fix(\sigma_A^k | \Sigma_A) = \sum_i c(k,i,i) = \sum_i (A^k)_{ii} = Tr(A^k).$$ □

.

Définition 8. *1. Soit $s \in (\Sigma_A^+)$, on dit que s est* ***périodique*** *de période k si et seulement si $\sigma_A^k(s) = s$.*

2. Si s admet une période, alors on appelle la ***période primitive*** *K, la plus petite période de s.*

Proposition 11. *On se place dans Σ_A dans le cas où l'on a :*

$$A = \begin{pmatrix} 0 & 1 & \dots & 1 & 1 \\ 1 & 0 & \dots & 1 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & \dots & 0 & 1 \\ 1 & 1 & \dots & 1 & 0 \end{pmatrix}$$

Dans cette configuration, on étudie donc les suites dans $\{1, \dots, p\}$ avec $p \geq 1$, tel que deux termes consécutifs sont distincts. Alors le nombre de suites périodiques de période $k \geq 1$ de Σ_A est $(p-1)^k + (-1)^k(p-1)$.

Preuve. Soit k ≥ 1. Posons $\Sigma_A^{(k)} = \{s \in \Sigma_A | s$ de période k$\}$ Alors on a:

$$f : \begin{array}{ccc} \Sigma_A^{(k)} & \to & \{1, \dots, p\}^{k+1} \\ s & \mapsto & (s_0, s_1, \dots s_{k-1}, s_0) \end{array}$$

f est une bijection sur son image, ainsi compter le nombre de suite périodique de période k tel que deux termes consécutifs soient différents revient à compter le nombre de mot de l'alphabet $\{1, \dots p\}$ de longueur $k+1$, terminant et commençant par la même lettre, et tel que deux lettres consécutives sont toujours différentes. On a :

$$f(\Sigma_A^{(k)}) = \bigsqcup_{i=1}^{p} \{(a_0, \dots, a_k) \in \{1, \dots, p\}^{k+1} \,|\, a_i \neq a_{i+1} \text{ et } a_0 = a_k = i\}$$

Ainsi:

$Card(\Sigma_A^{(k)}) = Card(f(\Sigma_A^{(k)}))$

$Card(\Sigma_A^{(k)}) = \sum_{i=1}^{p} Card(\{$Mots de longueur k + 1 commençant et terminant par la même lettre i$\})$

D'après le lemme 5 on a:

$Card(\{$Mots de longueur k commençant et terminant par la même lettre i$\}) = (A^k)_{i,i}$

D'où:

$$Card(\Sigma_A^{(k)}) = \sum_{i=1}^{p} (A^k)_{i,i} = Tr(A^k)$$

Calculons à présent A^k, on pose:

$$J := \begin{pmatrix} 1 & 1 & \dots & 1 & 1 \\ 1 & 1 & \dots & 1 & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & 1 & \dots & 1 & 1 \\ 1 & 1 & \dots & 1 & 1 \end{pmatrix}$$

On a :

$$J^2 = p \times J$$

Donc:

$$\forall n \in \mathbb{N}^*, J^n = p^{n-1} \times J$$

De plus comme J commute avec I_p, on utilise la formule de Newton :

$$\begin{aligned} A^k = (J - I_p)^k &= \sum_{l=0}^{k} \binom{k}{l} (-1)^{k-l} J^l \\ &= \sum_{l=1}^{k} \binom{k}{l} (-1)^{k-l} p^{l-1} J + (-1)^k I_p \\ &= \frac{1}{p} (\sum_{l=0}^{k} \binom{k}{l} (-1)^{k-l} p^l) \times J - \frac{(-1)^k}{p} J + (-1)^k I_p \\ &= \frac{1}{p} (p-1)^k J - \frac{(-1)^k}{p} J + (-1)^k I_p \end{aligned}$$

D'où finalement:

$$\begin{aligned} Card(\Sigma_A^{(k)}) &= Tr(A^k) \\ &= \frac{1}{p} (p-1)^k Tr(J) - \frac{(-1)^k}{p} Tr(J) + (-1)^k Tr(I_p) \\ &= \frac{1}{p} (p-1)^k \times p - \frac{(-1)^k}{p} \times p + (-1)^k \times p \\ &= (p-1)^k - (-1)^k + (-1)^k \times p \\ &= (p-1)^k + (-1)^k (p-1). \end{aligned}$$

□

Remarque. Dans le cas de notre billard, où $A = A_3$, on a donc $2^k + 2(-1)^k$ suites périodiques de période k.

Définition 9. *Une matrice $n \times n$ à coefficients dans $\{0,1\}$ est **réductible** s'il existe une paire i,j telle que $(A^k)_{ij} = 0$ pour tout $k \geq 1$. Visuellement, cela signifie qu'il existe dans le graphe associé une paire de sommets i, j telle que i ne peut être relié à j.*

*Une matrice $n \times n$ à coefficients dans $\{0,1\}$ est donc **irréductible** si pour toute paire*

i,j , il existe $k(i,j) \geq 1$ tel que $(A^{k(i,j)})_{ij} > 0$. Sur le graphe associé, cela veut dire que pour tous sommets i, j il existe un mot autorisé par A de i vers j.

Proposition 12. *Soit A une matrice de transition. Supposons que $s^* \in \Sigma_A$ est d'orbite dense dans Σ_A et n'est pas périodique. Alors A n'est pas une matrice de permutation et pour tout $k \geq 1$, $\sigma_A^k(s^*)$ est d'orbite dense dans Σ_A.*

Preuve. Soit $k > 1$. Montrons d'abord que l'orbite de $\sigma_A^k(s^*)$, $\{\sigma_A^{k+j}(s^*)\,;\, j \in \mathbb{N}\}$, est dense dans $\Sigma_A \setminus \{\sigma_A^i(s^*)\,;\, 0 \leq i < k\}$. Soit s dans $\Sigma_A \setminus \{\sigma_A^i(s^*); 0 \leq i < k\}$ et s^n une suite de l'orbite de $\sigma_A(s^*)$ tel que s^n converge vers s. Par l'absurde, supposons qu'à partir d'un certain rang, pour tout n, s^n appartienne à $\{\sigma_A^i(s^*); 0 \leq i < k\}$. Alors il existe i_0 tel qu'une infinité de s_n soit égal à $\sigma_A^{i_0}(s^*)$. Donc pour tout n dans $\mathbb{N}$, il existe k_n tel que $s^{k_n} = \sigma_A^{i_0}$ et $d(s^{k_n}, s^*) \leq 2^{-n}$, donc les n premières lettres de $\sigma_A^{i_0}(s^*)$ et s sont identiques. Donc $s = \sigma_A^{i_0}(s^*) \notin \Sigma_A \setminus \{\sigma_A^i(s^*); 0 \leq i < k\}$. Contradiction. Donc s est bien dans l'adhérence de l'orbite de $\sigma_A^k(s^*)$.

Soit $s \in \Sigma_A$ et $\epsilon > 0$. s^* a un antécédent par σ_A (d'après la définition de matrice de transition) qu'on note t^*. L'orbite de t^* est dense dans Σ_A (comme $\sigma_A(t^*) = s^*$ et que s^* est d'orbite dense). Il existe donc $n_1 \geq 0$ tel que $d(s, \sigma_A^{n_1}(t^*)) < \frac{\epsilon}{2}$. Or t^* n'est pas dans l'orbite de s^*, sinon t^* est périodique et donc s^* l'est aussi. Donc t^* appartient à $\Sigma_A \setminus \{\sigma_A^i(s^*)\,;\, 0 \leq i < k\}$. Donc par densité de l'orbite de $\sigma_A^k(s^*)$ dans $\Sigma_A \setminus \{\sigma_A^i(s^*)\,;\, 0 \leq i < k\}$, il existe $n_2 \geq k$ tel que $d(t^*, \sigma_A^{n_2}(s^*)) < \frac{\epsilon}{2}$. Mais alors $d(\sigma_A^{n_1}(t^*), \sigma_A^{n_1+n_2}(s^*)) < \frac{\epsilon}{2}$. Donc $d(s, \sigma_A^{n_1+n_2}(s^*)) \leq d(s, \sigma_A^{n_1}(t^*)) + d(\sigma_A^{n_1}(t^*), \sigma_A^{n_1+n_2}(s^*)) < \epsilon$. L'orbite de $\sigma_A^k(s^*)$ est bien dense dans Σ_A. □

Proposition 13. *Les deux propositions suivantes sont équivalentes :*

1. *A est irréductible*

2. *σ_A possède une orbite dense dans Σ_A.*

Preuve. Montrons d'abord 1. $\implies$ 2. . Montrons qu'on peut construire une suite s^* contenant tous les mots autorisés par A de longueur finie. Pour n dans $\mathbb{N}$ indexons les mots de longueurs n : notons les $v_{n,1}, v_{n,2}, ..., v_{l_n,n}$ avec l_n le nombre de mots de tailles n. Comme A est irréductible, pour tout i dans $\{1, ..., l_n - 1\}$, il existe un mot $t_{n,i}$ autorisé par A reliant la dernière lettre de $v_{n,i}$ et la première lettre de $v_{n,i+1}$. On peut donc construire le mot $w_n = v_{n,1}t_{n,1}v_{n,2}t_{n,2}...t_{n,l_n-1}v_{n,l_n}$ contenant tous les mots de longueur n. Ensuite, toujours par irréductibilité de A, pour tout n dans $\mathbb{N}$, il existe un mot $t_n^{'}$ autorisé par A reliant la dernière lettre de w_n et la première lettre de w_{n+1}. Alors la suite $s^* = w_1 t_1^{'} w_2 t_2^{'} w_3...$ est bien dans Σ_A et contient bien tous les mots autorisés par A de longueur finie. Soit $u \in \Sigma_A$. Soit $n \in \mathbb{N}$. Alors il existe un k dans $\mathbb{N}$ tel que $\sigma_A^k(s^*)$ coïncide avec u sur les n premiers termes. Mais alors $d(u, \sigma_A^k(s^*)) \leq 2^{-n}$. L'orbite de s^* est donc dense dans σ_A.

Montrons maintenant 2. $\implies$ 1. . Si A est une permutation, A est irréductible (on peut tracer le graphe d'une permutation pour vérifier qu'on peut aller d'un sommet à l'autre). Supposons donc que A n'est pas une permutation. Soit $s^* \in \Sigma_A$ dans une orbite dense de σ_A. Comme A n'est pas une permutation, s^* n'est pas périodique. Soit i,j dans $\{1, ..., p\}$. Il existe

$a \in \Sigma_A$tel que $a_0 = i$ par construction de A et Σ_A (cf rem. déf. 4.1.). Par densité de l'orbite de s^*, il existe t dans cette orbite tel que $d(a,t) < 1$, ce qui implique $t_0 = a_0$. Or il existe $k_1 \in \mathbb{N}$ tel que $\sigma_A^{k_1}(s^*) = t$, donc $s^*_{k_1} = a_0 = i$. Par le même raisonnement, il existe $b \in \Sigma_A$ tel que $b_0 = j$. En utilisant la densité de l'orbite de $\sigma_A^{k_1+1}(s^*)$ (par la proposition précédente comme s^* n'est pas périodique), il existe $k_2 > k_1$ tel que $\sigma_A^{k_2}(s^*)_0 = b_0$, donc $s^*_{k_2} = j$. Alors le mot $s_{k_1}s_{k_1+1}...s_{k_2}$ est autorisé par A et relie les sommets i et j, donc A est irréductible. □

Lemme 6. *Si A est une matrice de transition irréductible et qu'il existe i_0 tel que $\sum_j a_{i_0,j} \geq 2$ alors pour tout i, il existe $k_i \in \mathbb{N}$ tel que $\sum_j (A^{k_i})_{i,j} \geq 2$*

Preuve. Puisque A est irréductible, il existe un mot $w = is_1s_2...s_{k-1}i_0 \in \Sigma_A$. Posons k la longueur de w. Il y a donc k - 1. Alors $c(k-1,i,i0) \geq 1$. Après i_0, il y a donc 2 choix possibles. Donc $\sum_j (A^k)_{ii_0} \geq \sum_j (A^{k-1})_{i_0j} \times A_{i_0j} = \sum_j c(k-1,i,i_0)A_{i_0j} \geq 2$. □

Définition 10. *Un espace topologique est **parfait** s'il est fermé et qu'il ne possède pas de points isolés.*

Proposition 14. *Si A est une matrice de transition irréductible et qu'il existe i_0 tel que $\sum_j a_{i_0,j} \geq 2$ alors Σ_A^+ est parfait.*

Preuve. Soit $s \in \Sigma_A^+$, soit $\epsilon > 0$, alors il existe $N \in \mathbb{N}$ tel que $\epsilon \geq \frac{1}{2^{N+1}}$ et il existe $k_{s_n} \in \mathbb{N}$ tel que $\sum_j (A^{k_{s_n}})_{s_n,j} \geq 2$ (par le lemme précedent). Posons $t \in \Sigma_A^+$ tel que $\forall n \in \{0,...,N\}, t_n = s_n$ et $t_{N+m} \neq s_{N+m}$ pour un m compris entre 1 et k_{s_n}. On peut choisir un tel t car il y a plus de 2 mots différents de longueur k_{s_n} commençant par s_n. Ainsi $d(s,t) \leq 2^{-\inf\{k \in \mathbb{N} : s_k \neq t_k\}} \leq 2^{-(N+1)} \leq \epsilon$ et $s \neq t$. □

2.4 Décalages bilatères

Définition 11. *On définit de la même manière que pour $\mathcal{A}^{\mathbb{N}}$ les espaces $\mathcal{A}^{\mathbb{Z}}$ (voir définition 3 et définition 4)*

Proposition 15. *$\mathcal{A}^{\mathbb{Z}}$ est métrisable et on peut le munir d'une distance ultramétrique définie pour tout $s,t \in \mathcal{A}^{\mathbb{Z}}$ par:*

$$d(s,t) = 2^{-\inf\{|k| \in \mathbb{N} : s_k \neq t_k\}} \text{ avec } \inf \varnothing = +\infty$$

Proposition 16. *$\mathcal{A}^{\mathbb{Z}}$ est compact.*

Définition 12. *Soit $p \geq 2$. On définit l'espace Σ_p comme l'ensemble $\mathcal{A}^{\mathbb{Z}}$ lorsque $\mathcal{A} = \{1,...,p\}$.*
On munit Σ_p de σ, définie par $\sigma(s) = t$ tel que pour tout $k \in \mathbb{Z}$ $t_k = s_{k+1}$.

Remarque. Nous avons noté de la même manière, σ, le décalage unilatéral (sur $\mathcal{A}^{\mathbb{N}}$) et le décalage bilatéral (sur $\mathcal{A}^{\mathbb{Z}}$) mais le contexte permet d'éviter toute ambiguïté.

Proposition 17. *σ est un homéomorphisme sur Σ_p*

Preuve. On peut aussi munir Σ_p de σ^{-1}, définie par $\sigma^{-1}(s) = t$ tel que pour tout $k \in \mathbb{Z}$ $t_k = s_{k-1}$.
Par construction, σ^{-1} est l'inverse de σ et ces deux applications sont continues car 2-lipschitziennes. □

Définition 13. *Soit $A = (a_{i,j})_{(i,j)\in\{1,\dots,p\}^2}$ une matrice de transition de taille $p \times p$, on définit alors l'ensemble Σ_A par $\left\{ s \in \Sigma_p | \, \forall k \in \mathbb{Z}, a_{s_k,s_{k+1}} = 1 \right\}$.*

Proposition 18. *1. Σ_A est fermé dans Σ_p*

2. $\sigma(\Sigma_A) = \Sigma_A$

Définition 14. *On définit $\sigma_A : \Sigma_A \to \Sigma_A$ comme la restriction de σ à Σ_A.*

Proposition 19. *Σ_A est compact*

Remarque. Les preuves des théorèmes et propositions précedentes sur $\mathcal{A}^{\mathbb{Z}}$ se démontrent de la même manière que sur $\mathcal{A}^{\mathbb{N}}$, on ne les détaillera donc pas.

3 Homéomorphisme entre K et Σ_A

3.1 Détermination des suites périodiques

Par la suite, on se donne un mot w^* admissible de taille k dans $\mathbb{N}$ et on essaie de trouver une trajectoire de K k-périodique suivant w^*, c'est-à-dire x dans K tel que $(T^n(x))_{n\in\{0,\dots,k-1\}} = w^*$.

Définition 15. *Posons $\Omega_{w^*} := \prod_{i=0}^{k-1} C_{w_i^*}$.*
On définit $\mathcal{E}$: $\begin{array}{ccc} \Omega_{w^*} & \to & \mathbb{R}^+ \\ Y = (Y_0, \dots, Y_{k-1}) & \mapsto & \sum_{i\in\mathbb{Z}/k\mathbb{Z}} ||Y_{i+1} - Y_i|| \end{array}$ *l'action associée à w^*.*

Théorème 1. *1. $\mathcal{E}$ est lisse .*

2. $\mathcal{E}$ admet un minimum et il est atteint pour une unique trajectoire périodique de K.

Preuve. Montrons d'abord que $\mathcal{E}$ est lisse. Pour tout $(Y_0, \dots, Y_{k-1})$ dans Ω_{w^*} on a :
$\forall i \in \mathbb{Z}/k\mathbb{Z}, Y_i \neq Y_{i+1}$ (car w^* est admissible), or $x \to ||x||$ est $\mathcal{C}^\infty$ sur $\mathbb{R}^2\backslash\{0\}$ or :
$\mathcal{E}(Y_0, \dots, Y_{k-1}) = \sum_{i \in \mathbb{Z}/k\mathbb{Z}} ||f_i(Y_0, \dots, Y_{k-1})||$ où $\forall i \in \mathbb{Z}/k\mathbb{Z}, f_i(Y_0, \dots, Y_{k-1}) = Y_i - Y_{i+1} \neq 0$.
Les f_i étant bien $\mathcal{C}^\infty$ et à valeurs différentes de 0, on en déduit que $\mathcal{E}$ est elle aussi $\mathcal{C}^\infty$.

Montrons maintenant que $\mathcal{E}$ possède un minimum global sur Ω_{w^*}.
$\forall i \in \mathbb{Z}/k\mathbb{Z}, C_{w_i^*}$ est un fermé borné de $\mathbb{R}^2$ (c'est un cercle du plan), c'est donc un compact de $\mathbb{R}^2$.
Ainsi Ω_{w^*} est un compact comme produit fini de compact.
$\mathcal{E}$ est continue sur Ω_{w^*} compact donc elle y admet un minimum global.

Montrons que le minimum de $\mathcal{E}$ est une trajectoire périodique.
Soit $(Y_0, Y_1, \dots, Y_{k-1}) \in \underset{(Y_0,Y_1,\dots,Y_{k-1}) \in \Omega_{w^*}}{\operatorname{argmin}} \mathcal{E}(Y_0, Y_1, \dots, Y_{k-1})$
On définit pour i dans $\mathbb{Z}/k\mathbb{Z}$:
$(q_i, \vec{v_i}) := \left(Y_i, \frac{Y_{i+1}-Y_i}{||Y_{i+1}-Y_i||}\right)$

Montrons d'abord que $T(q_i, \vec{v_i})$ est bien défini, c'est à dire que pour tout i dans $\mathbb{Z}/k\mathbb{Z}$ $(q_i, \vec{v_i})$ est un élement de F, ce qui est équivalent à dire que tout segment $[Y_i, Y_{i+1}]$ n'intersecte les obstacles qu'à ses extremités, ou encore que pour tout $i \in \mathbb{Z}/k\mathbb{Z}, Card([Y_i, Y_{i+1}] \cap \partial Q) = 2$.
Raisonnons par l'absurde en supposant qu'il existe $i \in \mathbb{Z}/k\mathbb{Z}$ tel que $Card([Y_i, Y_{i+1}] \cap \partial Q) > 2$
Autrement dit $]Y_i, Y_{i+1}[$ s'intersecte avec au moins un obstacle et plus précisement avec $C_{w_i^*}$ ou $C_{w_{i+1}^*}$ par condition de non eclipsité.
On considère sans perdre de généralités que $C_{w_i^*} \cap]Y_i, Y_{i+1}[$ est non vide , soit $Y_i' \in C_{w_i^*} \cap]Y_i, Y_{i+1}[$, alors on a :

$$||Y_i - Y_{i+1}|| = ||Y_i - Y_i'|| + ||Y_i' - Y_{i+1}|| \text{ et}$$
$$||Y_i' - Y_{i-1}|| \leq ||Y_i - Y_i'|| + ||Y_i - Y_{i-1}||$$

Donc :

$||Y_i' - Y_{i-1}|| + ||Y_i' - Y_{i+1}|| \leq ||Y_i - Y_i'|| + ||Y_i - Y_{i-1}|| + ||Y_i - Y_{i+1}|| - ||Y_i - Y_i'||$ d'où:
$||Y_i' - Y_{i-1}|| + ||Y_i' - Y_{i+1}|| \leq ||Y_i - Y_{i-1}|| + ||Y_i - Y_{i+1}||$

De plus comme Y_{i-1}, Y_i, Y_{i+1} ne sont pas alignés on a : $||Y_i' - Y_{i-1}|| \neq ||Y_i - Y_{i-1}||$ d'où:
$||Y_i' - Y_{i-1}|| + ||Y_i' - Y_{i+1}|| < ||Y_i - Y_{i-1}|| + ||Y_i - Y_{i+1}||$
Ainsi enfin :
$\mathcal{E}(Y_0, \dots, Y_i', \dots, Y_{k-1}) < \mathcal{E}(Y_0, \dots, Y_i, \dots, Y_{k-1})$
Ce qui est exclu car $(Y_0, \dots, Y_i, \dots, Y_{k-1})$ est un minimum global de $\mathcal{E}$.
Donc $(q_i, \vec{v_i}) \in F$

On va montrer que pour tout i dans $\mathbb{Z}/k\mathbb{Z}$, $T(q_i, \vec{v_i}) = (q_{i+1}, \vec{v}_{i+1})$, de sorte que $(T^n(q_0, \vec{v}_0))_{n \in \mathbb{Z}}$ est une trajectoire périodique vérifiant $(T^n(q_0, \vec{v}_0))_{\{n \in \{0,\dots,k-1\}} = w^*$.

Montrons à présent que pour i dans $\mathbb{Z}/k\mathbb{Z}$, $T(q_i, \vec{v_i}) = (q_{i+1}, \vec{v}_{i+1})$
Soit $p_1(x, y) \to x$ la projection d'un vecteur de Ω sur sa première coordonnée. Par construc-

tion on a déjà que $p_1(T((q_i, \vec{v}_i))) = q_{i+1}$.

Montrons que $p_2(T((q_i, \vec{v}_i))) = \vec{v}_{i+1}$.

On peut voir $\mathcal{E}$ comme la restriction d'une fonction J de $\mathbb{R}^{2k}$ dans $\mathbb{R}$ de variable $(x_0, y_0, x_1, y_1, \ldots, x_{k-1}, y_{k_1})$. Minimiser $\mathcal{E}$ revient alors à minimiser J sous contraintes, où les contraintes sont, pour tout i dans $\{0, \ldots, k-1\}, (x_i, y_i) \in C_{w_i^*}$. Le problème de minimisation est alors $\min_{g=0} J$ en posant $g = (g_0, \ldots, g_{k-1})$ et pour tout i dans $\{0, \ldots, k-1\}, g_i : (x_0, y_0, x_1, y_1, \ldots, x_{k-1}, y_{k_1}) \mapsto (x_i - c_{w_i^*,1})^2 + (y_i - c_{w_i^*,2})^2 - a_{w_i^*}^2$.

L'expression du gradient de J est :

$$\nabla J(x_0, y_0, x_1, y_1, \ldots, x_{k-1}, y_{k-1}) = \begin{pmatrix} \vdots \\ \frac{x_i - x_{i+1}}{||Y_i - Y_{i+1}||} + \frac{x_i - x_{i-1}}{||Y_i - Y_{i-1}||} \\ \frac{y_i - y_{i+1}}{||Y_i - Y_{i+1}||} + \frac{y_i - y_{i-1}}{||Y_i - Y_{i-1}||} \\ \vdots \end{pmatrix}$$

Et celui de g_i pour tout i dans $\mathbb{Z}/k\mathbb{Z}$ est :

$$\nabla g_i(x_0, y_0, x_1, y_1, \ldots, x_{k-1}, y_{k-1}) = \begin{pmatrix} 0 \\ \vdots \\ 2(x_i - c_{w_{i1}^*}) \\ 2(y_i - c_{w_{i2}^*}) \\ \vdots \\ 0 \end{pmatrix}$$

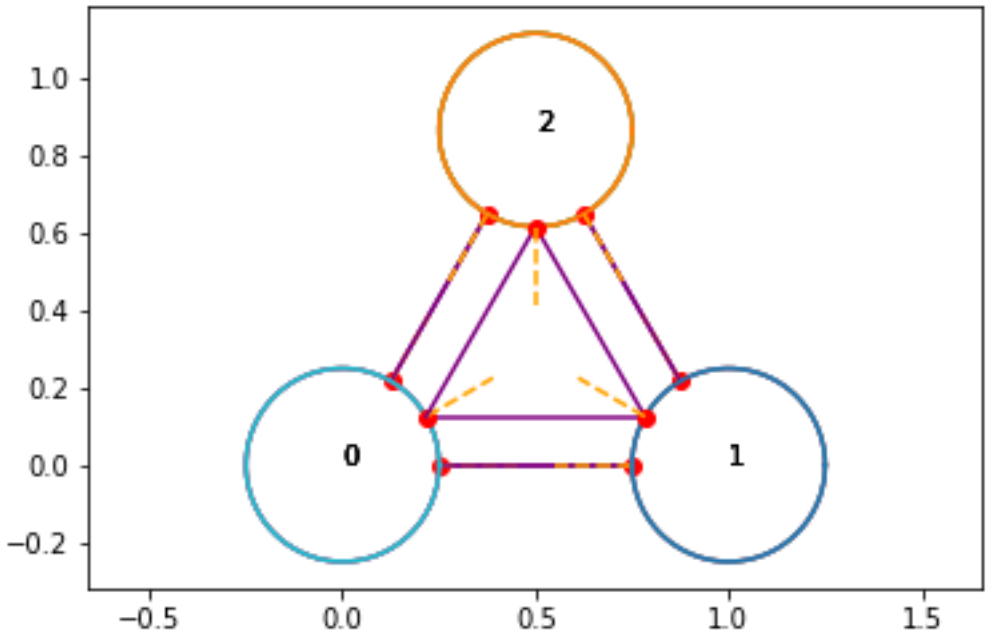
La différentielle $d_{Y^*} g$ est surjective (matrice triangulaire inférieure à coefficients non nuls sur la diagonales, les $(x_i - c_{w_{i1}^*})$ et les $(x_i - c_{w_{i1}^*})$ étant non nuls pour tout i dans $\mathbb{Z}/k\mathbb{Z}$). On peut alors appliquer le théorème des extrema liés : il existe λ dans $\mathbb{R}^k$ tel que $\nabla J(Y^*) = \langle \lambda, \nabla g(Y^*) \rangle_{\mathbb{R}^k} = \sum_{i=0}^{k-1} \lambda_i \nabla g_i(Y^*)$.

Donc pour tout i dans $\mathbb{Z}/k\mathbb{Z}$, on obtient les relations $\frac{x_{i+1} - x_i}{||Y_{i+1} - Y_i||} = \frac{x_i - x_{i-1}}{||Y_i - Y_{i-1}||} - 2\lambda_i(x_i - c_{w_{i1}^*})$ et $\frac{y_{i+1} - y_i}{||Y_{i+1} - Y_i||} = \frac{y_i - y_{i-1}}{||Y_i - Y_{i-1}||} - 2\lambda_i(y_i - c_{w_{i2}^*})$ ou vectoriellement, $\vec{v}_i = \vec{v}_{i-1} - \lambda_i \vec{n}(q_i)$ en renormalisant λ_i. Par passage à la norme et avec $\vec{v}_i$, $\vec{v}_{i-1}$ et $\vec{n}(q_i)$ unitaires, on obtient :

$$\begin{aligned} ||\vec{v}_i||^2 = ||\vec{v}_{i-1} - \lambda_i \vec{n}(q_i)||^2 &\iff 1 = 1 - 2\lambda_i \langle \vec{v}_{i-1}, \vec{n}(q_i) \rangle + \lambda_i^2 \\ &\iff \lambda_i(\lambda_i - 2\langle \vec{v}_{i-1}, \vec{n}(q_i) \rangle) = 0 \\ &\iff \lambda_i = 2\langle \vec{v}_{i-1}, \vec{n}(q_i) \rangle \qquad (\lambda_i \neq 0 \text{ car } \vec{v}_{i-1} \neq \vec{v}_i) \end{aligned}$$

On a donc bien $\vec{v}_i = \vec{v}_{i-1} - 2\langle \vec{v}_{i-1}, \vec{n}(q_i) \rangle \, \vec{n}\,(q_i)$. $\square$

Ce théorème permet de mettre en place une solution algorithmique pour trouver l'unique x dans K tel que sa trajectoire soit périodique et $(T^n(x))_{n \in \{0,\ldots,k-1\}} = w^*$, en utilisant l'algorithme de de descente de gradient projeté sur $\mathcal{E}$.



Simulation informatique des trajectoires periodiques de taille 3 et moins (Voir code B).

3.2 Conjugaison de (K, T) et $(\Sigma_{A_3}, \sigma_{A_3})$

Théorème 2. *(K, T) est conjugué à $(\Sigma_{A_3}, \sigma_{A_3})$, c'est à dire qu'il existe un homéomorphisme $h : K \to \Sigma_{A_3}$ tel que $h \circ T = \sigma_{A_3} \circ h$. En résumé, le diagramme suivant commute :*

$$\begin{array}{ccc} K & \xrightarrow{T} & K \\ \downarrow{\scriptstyle h} & & \downarrow{\scriptstyle h} \\ \Sigma_{A_3} & \xrightarrow[\sigma_{A_3}]{} & \Sigma_{A_3} \end{array}$$

Preuve. Posons dans un premier temps la fonction :

$$g : \begin{array}{ccc} K & \to & \{1,2,3\} \\ x = (q, \vec{v}) & \mapsto & i \text{ tel que } q \in C_i \end{array}.$$

Alors g est continue. En effet, $g = g_2 \circ g_1$ avec

$$g_1 : \begin{array}{ccc} K & \to & \partial Q \\ x = (q, \vec{v}) & \mapsto & q \end{array}$$

la projection selon la première coordonnée, application continue, et

$$g_2 : \begin{array}{ccc} \partial Q & \to & \{1,2,3\} \\ q & \mapsto & i \text{ tel que } q \in C_i \end{array}$$

qui est une application continue car les cercles sont 2 à 2 disjoints.

On peut ainsi définir :

$$h : \begin{array}{ccc} K & \to & \Sigma_A \\ x = (q, \vec{v}) & \mapsto & \left(g\left(T^k(x)\right)\right)_{k\in\mathbb{Z}} \end{array}$$

La continuité de h provient de celle de g et de T, montrons-le en fixant $x \in K$. Soit $n \geq 1$. Pour tout $k \in \{-n, \ldots, n\}$, par continuité de $g \circ T^k$, il existe $\delta_k > 0$ tel que $\forall y \in B(x, \delta_k), g(T^k(y)) \in B(g(T^k(x)), 1) = \{g(T^k(x))\}$. On a donc $\forall y \in B(x, \min\limits_{k\in\{-n,\ldots,n\}} \delta_k), \forall k \in \{-n, \ldots, n\}, g(T^k(y)) = g(T^k(x))$, donc $d(h(x), h(y)) = 2^{-\inf\{k\in\mathbb{N}:h(x)_k \neq h(y)_k\}} \leq 2^{-(n+1)}$.

Montrons à présent que $h \circ T = \sigma_{A_3} \circ h$, soit $x \in K$:

$$h \circ T(x) = h(T(x) = (g(T^k(T(x))))_{k\in\mathbb{Z}} = (g(T^{k+1}(x)))_{k\in\mathbb{Z}}$$

De plus:

$$\sigma_{A_3} \circ h(x) = \sigma_{A_3}(h(x)) = \sigma_{A_3}(g(T^k(x)))_{k\in\mathbb{Z}} = (g(T^{k+1}(x)))_{k\in\mathbb{Z}}$$

Donc $h \circ T(x) = \sigma_{A_3} \circ h(x)$.

Passons maintentant à la surjectivité de h.

Notons $\mathcal{P}$ le sous-ensemble des trajectoires périodiques de K. On remarque que $h(\mathcal{P})$ est l'ensemble des sous-suites périodiques de Σ_A. En effet, la première inclusion est donnée par la définition des trajectoires périodiques et pour la deuxième inclusion, soit $s = \ldots www\ldots$ une suite périodique de Σ_A. Alors la preuve du Théorème 1. pour le mot w construit x dans K tel que $h(x) = s$.

Montrons d'abord que $h(\mathcal{P})$ est dense dans Σ_A c'est-à-dire :

$$\overline{h(\mathcal{P})} = \Sigma_A \tag{1}$$

Soit s dans Σ_A. Alors pour tout $n \geq 1$, posons w périodique de période $2n+1$ coïncidant avec s sur $\{-n, \ldots, n\}$. Alors on a bien $d(s, w) \leq 2^{-n}$.

Montrons maintenant l'égalité suivante :

$$\overline{h(\mathcal{P})} \subset h\left(\overline{\mathcal{P}}\right) \tag{2}$$

C'est un résultat classique sous les hypothèses h continue et K compact. Soit s dans $\overline{h(\mathcal{P})}$. Il existe donc $(s^n)_{n\in\mathbb{N}} = (h(x_n))_{n\in\mathbb{N}}$ dans $h(\mathcal{P})^{\mathbb{N}}$ tel que $\lim s^n = s$ et où $(x_n)_{n\in\mathbb{N}}$ est une suite à valeur dans K compact. Donc il existe une sous-suite $(x_{n_k})_{k\in\mathbb{N}}$ de $(x_n)_{n\in\mathbb{N}}$ convergeant vers x dans K. Mais alors $\lim h(x_{n_k}) = h(x)$ par continuité de h et $\lim h(x_{n_k}) = \lim s^{n_k} = s$, donc $s = h(x)$ avec $x = \lim x_{n_k} \in \overline{\mathcal{P}}$. Donc $s \in h(\overline{\mathcal{P}})$.

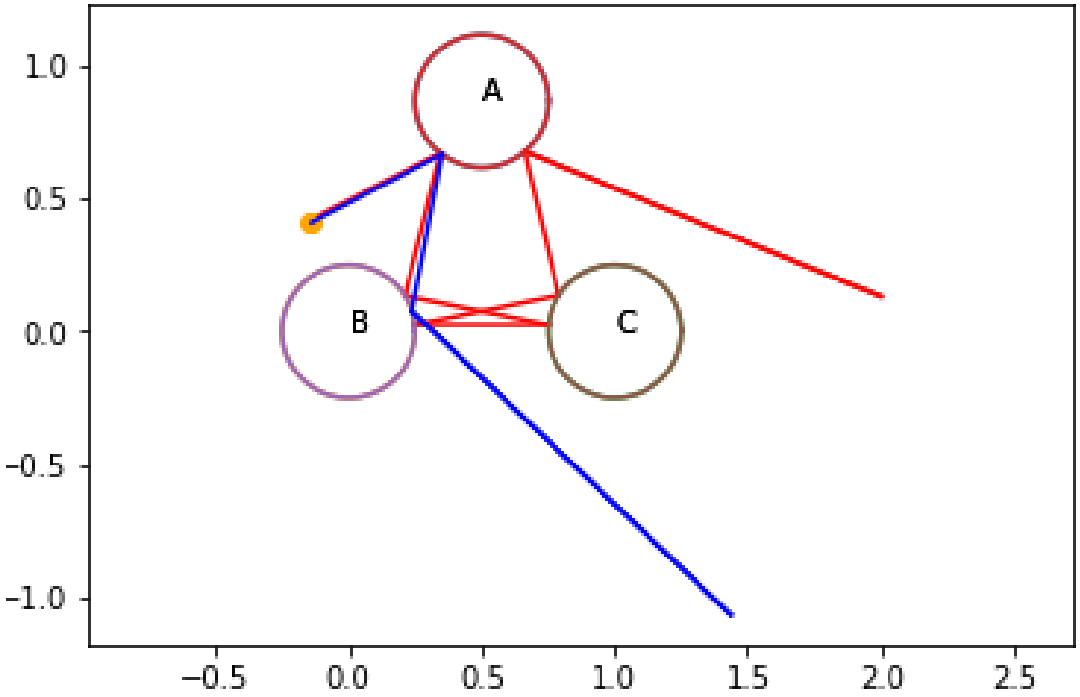
Alors en utilisant (1), (2) et l'inclusion $\overline{\mathcal{P}} \subset K$, on obtient :

$$\Sigma_A = \overline{h(\mathcal{P})} = h\left(\overline{\mathcal{P}}\right) \subset h(K) \subset \Sigma_A$$

Donc $h(K) = \Sigma_A$ et h est bien surjective.

□

3.3 Mélange topologique



Simulation informatique de deux trajectoires aux conditions initiales proches (Voir code A).

Définition 16. *Si X est un espace métrique et T une application continue, le système (X, T) est dit* ***topologiquement mélangeant*** *si pour tous les ouverts U, V non vides de X, il existe N tel que pour tout $n \geq N, U \cap T^{-n}V \neq \varnothing$.*

Proposition 20. *Les propriétés suivantes sont équivalentes :*

1. *Le système (Σ_A, σ_A) est topologiquement mélangeant*
2. *A est irréductible et il existe 2 suites périodiques de période premières entre elles*
3. *Il existe un rang N tel que pour tout $n \geq N$, pour tout i, j dans $\{1, \dots, p\}, (A^n)_{ij} > 0$.*

Preuve. 1. $\implies$ 2. : Si (Σ_A, σ_A) est topologiquement mélangeant, on peut appliquer la définition avec $U = B(s, 1)$ et $V = B(t, 1)$ avec s, t dans Σ_A. Donc il existe N tel que pour tout $n \geq N$, il existe u vérifiant la propriété :

$$u \in U \cap \sigma_A^{-n} V \qquad (P_n)$$

Soit u vérifiant la propriété (P_n) par exemple pour $n = N$. Alors $u \in B(s, 1)$ donc $u_0 = s_0$ et $\sigma_A^n(u) \in B(t, 1)$ donc $u_n = t_0$. Le mot $s_0 u_1 \dots u_{n-1} t_0 = u_0 \dots u_n$ est donc admissible. Comme on peut choisir arbitrairement s_0 et t_0, on a alors l'irréductibilité de A. Maintenant soit u et v vérifiant respectivement (P_N) et (P_{N+1}) lorsque l'on choisit nos deux suites s et t tel que $s_0 = t_0$. Alors les mots $s_0 u_1 \dots u_{N-1} t_0$ et $s_0 v_1 \dots v_N t_0$ sont admissibles. Mais alors les

suites périodiques répétant les mots $s_0u_1\dots u_{N-1}$ et $s_0v_1\dots v_N$ sont périodiques de périodes N et $N+1$, qui sont premiers entre eux.

2. $\implies$ 3. : Supposons qu'il existe s et t deux suites périodiques de Σ_A de période respectivement p et q avec $p\wedge q=1$. On va alors montrer l'assertion suivante :

$$\forall n\geq pq, \exists (u,v)\in\mathbb{N}^2, up+vq=n.$$

Soit $n\geq pq$. Alors d'après le théorème de Bézout, il existe u,v dans $\mathbb{N}$ tel que $up+vq=1$ et l'ensemble des solutions de l'équation $xp+yq=n$ dans $\mathbb{Z}^2$ est alors $\{(u_0-kq, v_0+kp)\,|\,k\in\mathbb{Z}\}$ avec $(u_0,v_0)=(nu,nv)$. On notera donc ces solutions $\{(u_k,v_k)\}_{k\in\mathbb{Z}}$. On a alors :

$$\begin{aligned} &0\leq u_k \text{ et } 0\leq v_k\\ \iff &kq\leq u_0 \text{ et } -v_0\leq kp\\ \iff &-\frac{v_0}{p}\leq k\leq \frac{u_0}{q}\end{aligned}$$

L'intervalle décrit par k est non vide car $\frac{u_0}{q}-\left(-\frac{v_0}{p}\right)=\frac{u_0p+qv_0}{pq}=\frac{n}{pq}\geq 1$ comme $n\geq pq$. Il existe donc une suite $(u_n,v_n)_{n\geq pq}$ dans $\mathbb{N}^2$ vérifiant $u_np+v_nq=n$. Posons d'abord $N_1=\sup\limits_{i,j}\inf\{\,n\,:\,\exists w=w_0\dots w_n\in\Sigma_A,(w_0,w_n)=(i,j)\,\}$ (c'est le diamètre de $\{1,\dots,p\}$ pour la distance $d(i,j)=\inf\{\,|w|\,|\,w=w_0\dots w_n\in\Sigma_A,(w_0,w_k)=(i,j)\}$ qui est bien définie par irréductibilité de A). On peut donc relier deux sommets quelconques en moins de N_1 lettres. Soit i,j dans $\{1,\dots,p\}$. Posons maintenant $s^*=s_0\dots s_{p-1}$ et $t^*=t_0\dots t_{q-1}$ le motif des suites s et t. Il existe des mots $w^{(i,s_0)}$, $w^{(s_{p-1},t_0)}$ et $w^{(t_{q-1},j)}$ reliant respectivement i à s_0, s_{p-1} à t_0 et t_{q-1} à j chacun de longueur inférieure à N_1 par définition de N_1. Alors pour tout $n\geq pq$, le mot $w^{(i,s_0)}\,(s^*)^{u_n}\,w^{(s^*_{p-1},t^*_0)}\,(t^*)^{v_n}\,w^{(t_{q-1},j)}$ est un mot admissible reliant i à j de longueur $3N_1+u_np+v_nq=3N_1+n$. On a alors bien :

$$\forall n\geq 3N_1+pq, \forall i,j\in\{1,\dots,p\}, (A^n)_{ij}>0$$

3. $\implies$ 1. : Soit U,V 2 ouverts non vides de Σ_A. Il existe donc 2 boules $B(s,2^{-k})$ et $B(t,2^{-k})$ avec $k\geq 1$ contenues respectivement dans U et V. Alors il existe $N\geq 1$ tel que pour tout $n\geq N, (A^n)_{s_k,t_{-k}}>0$ et donc il existe un mot w_n admissible de longueur n reliant s_k à t_{-k}. Posons alors $s^{(n)}$ une suite de $B(s,2^{-k})$ vérifiant $s^{(n)}=\dots s_{-k}\dots s_kw_nt_{-k}\dots t_k\dots$. Alors on a bien $s^{(n)}$ dans $B(s,2^{-k})$ et $\sigma_A^{n+2k+1}(s^{(n)})$ dans $B(t,2^{-k})$. On a alors bien :

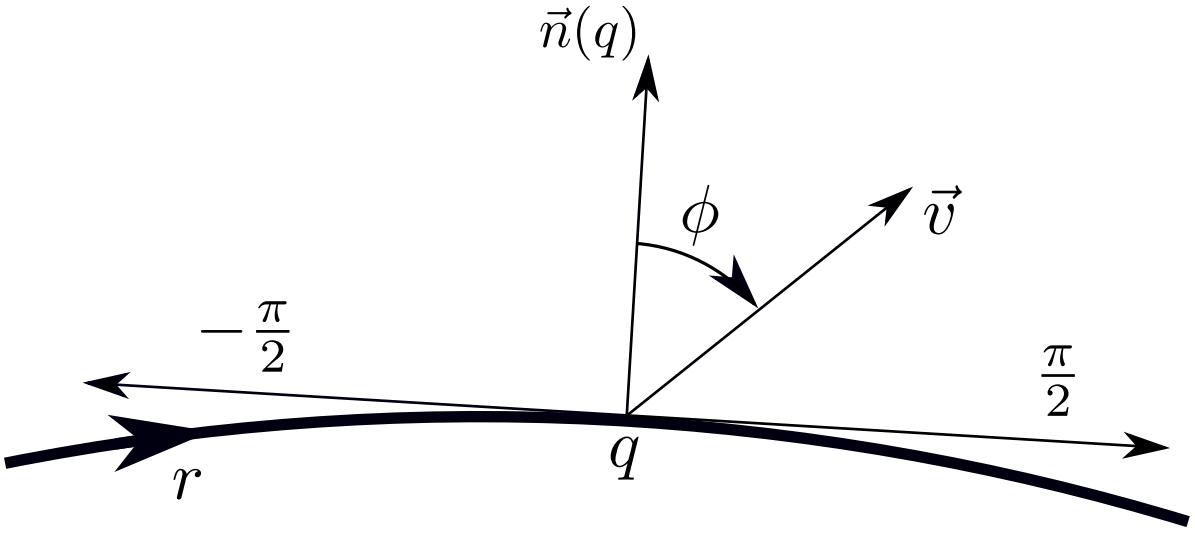
$$\forall n\geq N+2k+1, \forall i,j\in\{1,\dots,p\}, U\cap T^{-n}V\neq\varnothing$$ □

4 Coefficients de Lyapunov

4.1 Espace des phases

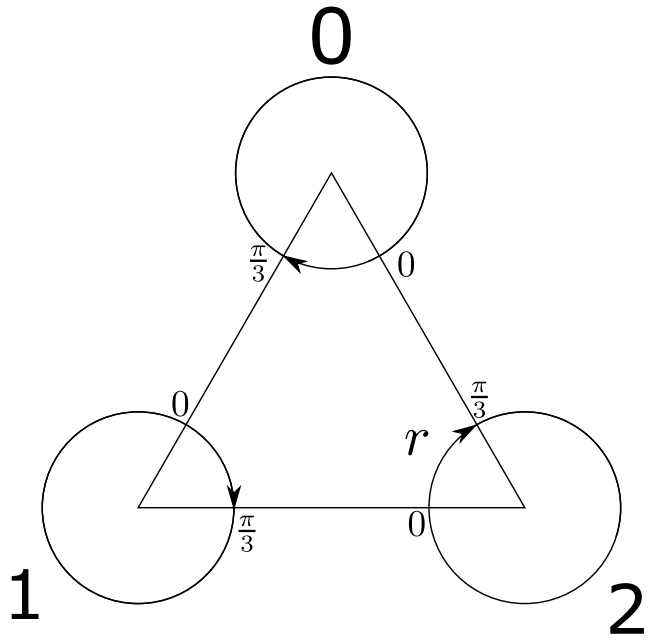
Lorsque $x=(q,\vec{v})$ avec $q\in\partial Q$, on peut écrire x en paramétrant le cercle C_i auquel q appartient, et définissant la vitesse selon la normale au point q. On pourra donc définir l'espace

des phases $\bigcup_i [a_i, b_i] \times [-\frac{\pi}{2}, \frac{\pi}{2}]$ où les $[a_i, b_i]$ sont des intervalles disjoints qui paramétrisent chacun des cercles C_i, et par la suite, on pourra aussi écrire $x = (r, \phi)$.



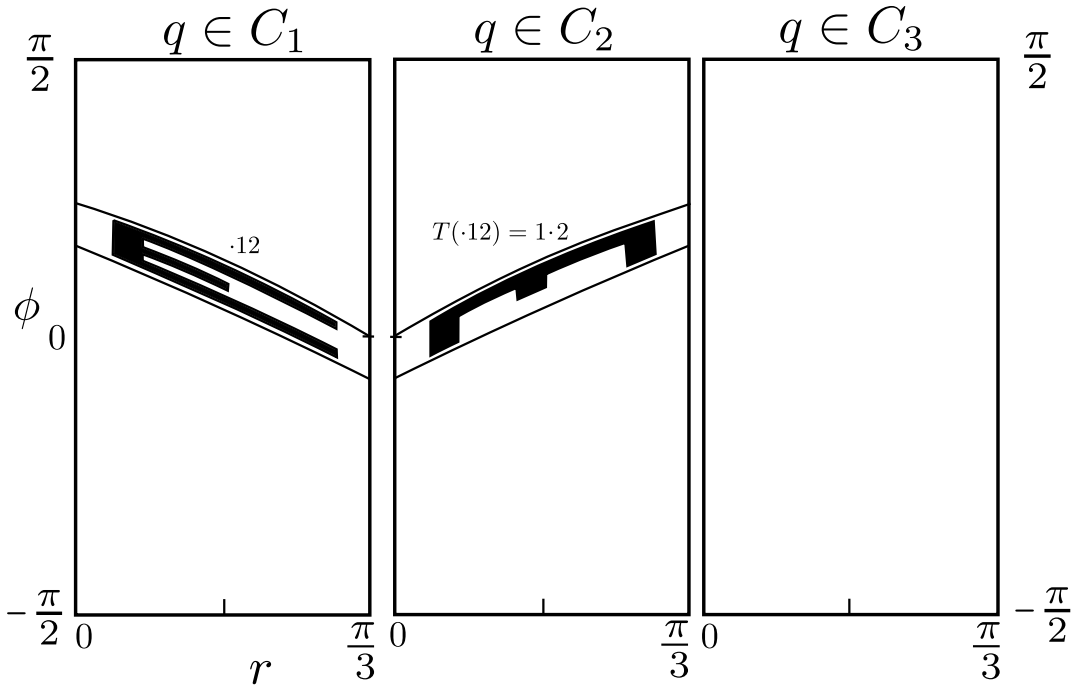
Paramétrisation de $x = (q, \vec{v})$

L'espace des phases fournit alors une nouvelle représentation de la transformation T et de l'ensemble récurrent, qui encourage à prendre la convention suivante pour les paramétisations de r :



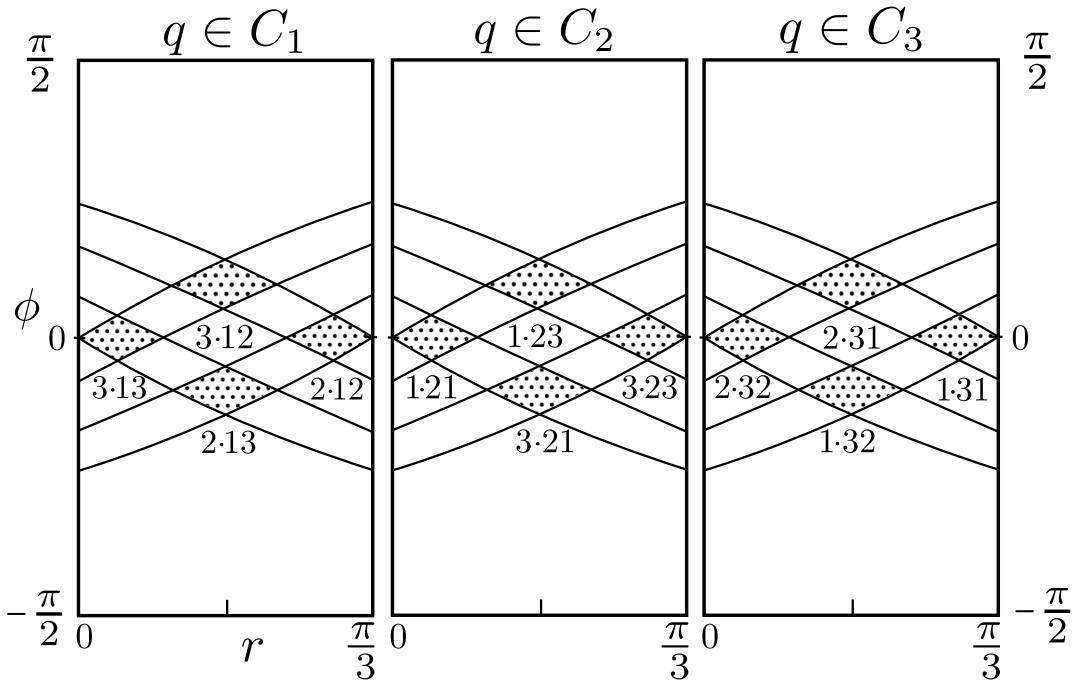
Convention prise pour r pour la représentation dans l'espace des phases

Pour comprendre comment fonctionne la transformation T dans l'espace des phases, voici un premier exemple sur $\cdot 12$, l'ensemble des suites de Σ_A telles que $s_0 = 1$ et $s_2 = 2$:



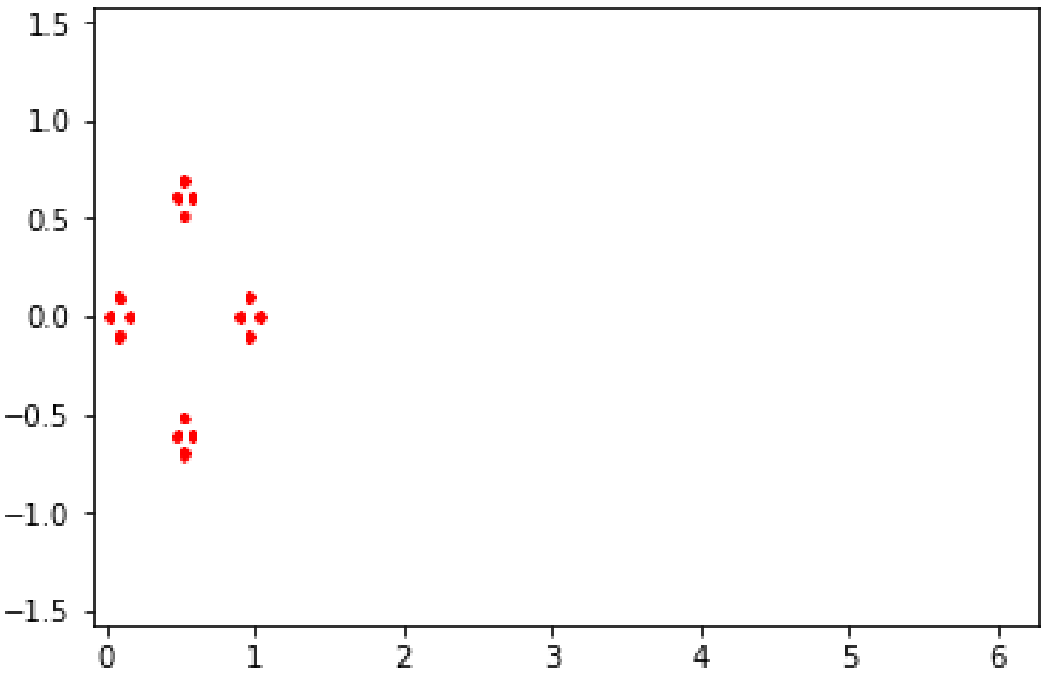
$1 \cdot 2$, image dans l'espace des phases de $\cdot 12$, ensemble des suites telles que $s_0 = 1$ et $s_1 = 2$.

On peut donc chercher à localiser l'espace récurrent en regardant les contraintes qu'il doit satisfaire en tant que sous-ensemble de la partition de Σ_A de la forme $(1 \cdot 23) \bigsqcup \ldots \bigsqcup (3 \cdot 21)$ (portions en losanges avec des poids) :



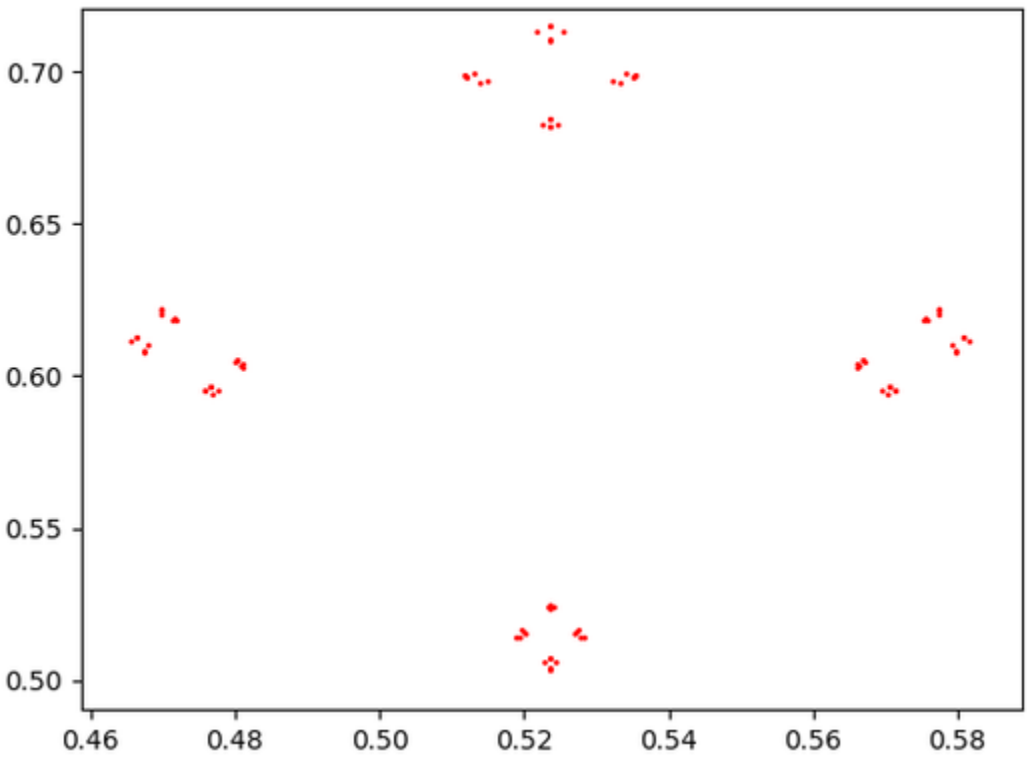
L'espace récurrent se situe dans les portions en losanges avec des poids.

Pour approcher l'espace récurrent, on peut aussi regarder l'ensemble des suites périodiques de périodes inférieures ou égales à n , voici ce que cela donne au niveau du premier obstacle dans l'espace des phases quand n=9 :



Représentation dans l'espace des phases des périodes de A_3 de taille 9 et moins (Voir code B).

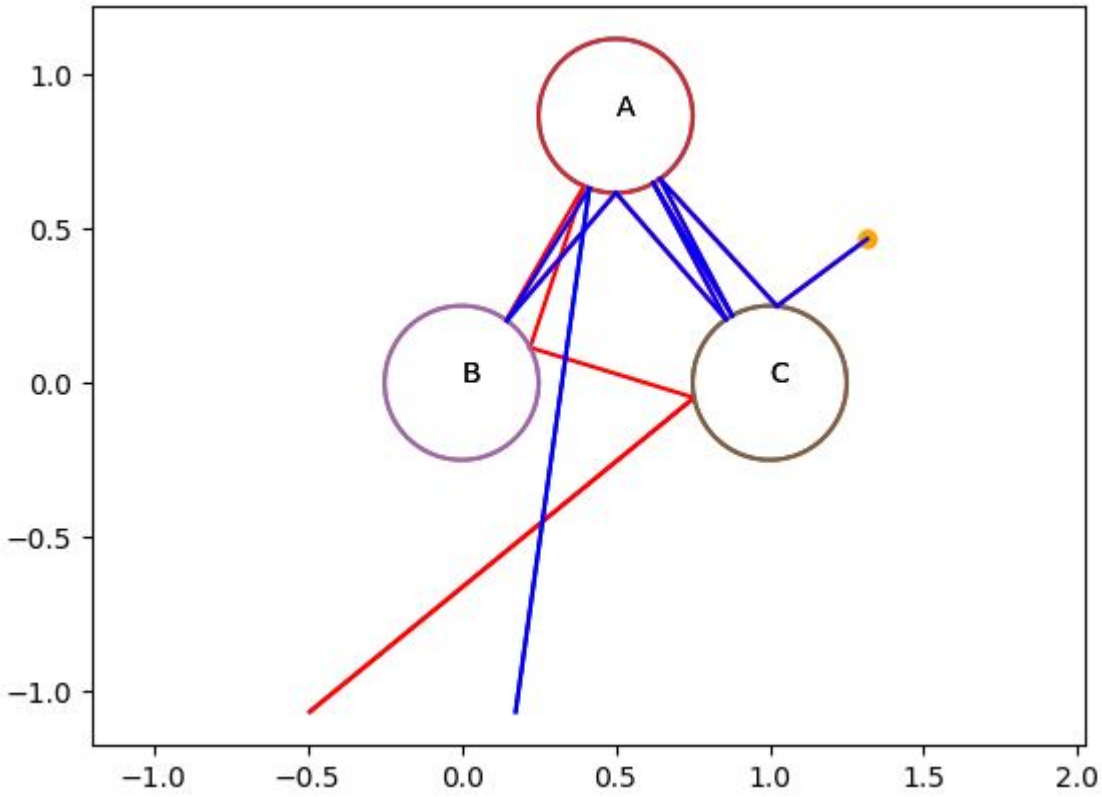
En regardant de plus près les points on remarque bien cet aspect d'espace de Cantor :



Représentation dans l'espace des phases des périodes de A_3 de taille 9 et moins lorsque l'on se rapproche d'un cluster de points (Voir code B).

4.2 Calcul des coefficients de Lyapunov

Nous allons maintenant nous intéresser aux exposants de Liapunov de notre système. Un exposant de Liapunov permet de quantifier la stabilité ou l'instabilité du sytème (voir [Wil17, BL85]). Puisque nous sommes ici dans un système chaotique, on s'attend à ce que les calculs de nos coefficients de Liapunov nous donne des résultats positifs.



Représentation de deux trajectoires ayant un écart angulaire initial variant de 10^{-7} (Voir code A).

Nous avons ici représenté en rouge une trajectoire effectuant 10 rebonds avant de partir à l'infini. La trajectoire en bleu part du même point que la trajectoire en rouge mais avec une différence angulaire de 10^{-7}. On observe que la trajectoire bleu part à l'infini après avoir suivi la trajectoire rouge sur les 8 premiers obstacles.
Grâce à cette représentation, on peut approximmer l'exposant de Liapunov grâce à la formule $d(T^n x, T^n y) \approx \exp(\lambda n) d(x, y)$ avec λ l'exposant de Liapunov du système au point x.
Les 2 trajectoires se séparent après 8 obstacles, on obtient donc que $\lambda(x) \approx \frac{1}{n} \times \ln(\frac{d(T^n x, T^n y)}{d(x,y)}) \approx \frac{7}{8} \times \ln(10) \approx 2,01$.

Nous allons maintenant nous placer dans l'espace (K, T) que nous allons munir d'une mesure μ définit sur l'epace conjugué $(\Sigma_{A_3}, \sigma_{A_3})$ par : à un obstacle $i \in \{1, 2, 3\}$, on donne une probabilité $\frac{1}{2}$ de rejoindre chacun des 2 autres obstacles pour le prochain rebond.

Proposition 21. *1.* $\lambda(x) = \lim_{n\to+\infty} \frac{1}{n} \ln \|D_x T^n\|$ $\mu - pp$

2. Le coefficient de Liapunov de notre système existe et est constant μ-presque partout.

Preuve. 1. C'est le théorème d'Oseledets que nous admettrons ici (voir [CM06]).

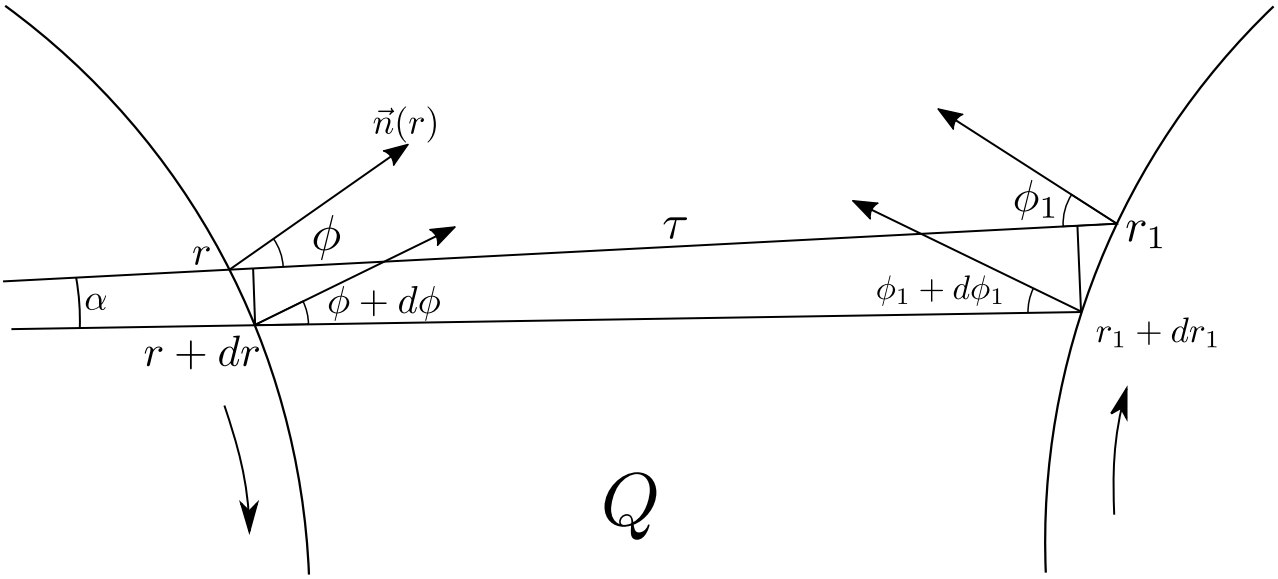
2. On déduit du premier point que λ est T invariant c'est à dire $\lambda \circ T = \lambda$. De plus μ est ergodique et T invariante donc λ est constant $\mu - pp$.

□

Proposition 22. *La différentielle de T est donnée, pour tout $x = (r, \phi)$, par :*

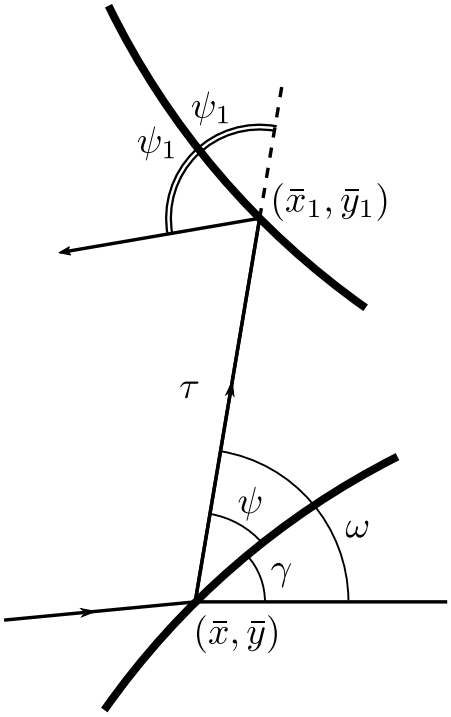
$$D_x T = \frac{-1}{\cos(\phi_1)} \begin{pmatrix} \tau\mathcal{K} + \cos(\phi) & \tau \\ \tau\mathcal{K}\mathcal{K}_1 + \mathcal{K}\cos(\phi_1) + \mathcal{K}_1\cos(\phi) & \tau\mathcal{K}_1 + \cos(\phi_1) \end{pmatrix}$$

où $\mathcal{K}$ et $\mathcal{K}_1$ sont les courbures aux points x et $T(x)$.



Représentation des petites variations

Preuve. Soit (r, ϕ) avec r l'abscisse curviligne et ϕ l'angle à la normale, soit (r_1, ϕ_1) l'image de (r, ϕ) par T. Soit $\mathcal{K}$(r) la courbure d'un cercle (celle-ci étant l'inverse de son rayon).
Soit $(\bar{x}, \bar{y})$ et $(\bar{x}_1, \bar{y}_1)$ les coordonnées cartésiennes respective de (r, ϕ) et (r_1, ϕ_1). Les angles ω, γ et ψ sont quant à eux définis sur le dessin ci dessous:



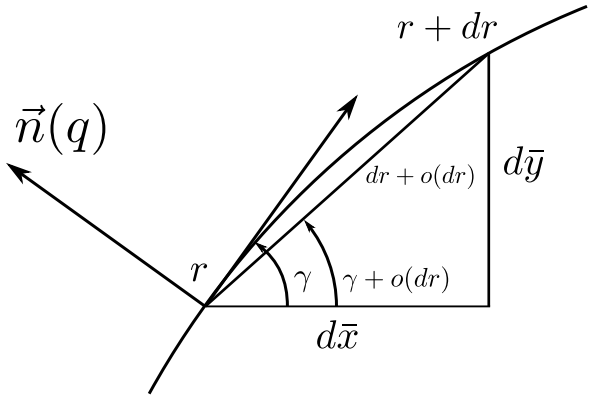
Introduction de $(\bar{x}, \bar{y}), (\bar{x}_1, \bar{y}_1), \psi, \psi_1, \omega, \gamma$

On a tout d'abord les relations :

$$\bar{x}_1 = \bar{x} + \tau \cos(\omega) \tag{3}$$
$$\bar{y}_1 = \bar{y} + \tau \sin(\omega) \tag{4}$$
$$\psi = \frac{\pi}{2} - \phi \tag{5}$$

On a aussi (voir dessin ci dessous) :

$$d\bar{x} = \cos(\gamma) dr \tag{6}$$
$$d\bar{y} = \sin(\gamma) dr \tag{7}$$



Calcul de $d\bar{x}$ et $d\bar{y}$

Et par la caractérisation de la courbure :

$$d\gamma = -\mathcal{K}dr \tag{8}$$

Par définition de γ et ψ on a aussi que:

$$\gamma + \psi = \omega = \gamma_1 - \psi_1$$

En différenciant et en utilisant (8) on a :

$$-\mathcal{K}dr + d\psi = d\omega = -\mathcal{K}_1 dr_1 - d\psi_1 \tag{9}$$

En differenciant les équations obtenues en (3) et (4) on a :

$$d\bar{x}_1 = d\bar{x} + \cos(\omega)\tau \tag{10}$$
$$d\bar{y}_1 = d\bar{y} + \sin(\omega)\tau \tag{11}$$

En utilisant les relations (6) et (7) cela donne:

$$\cos(\gamma_1)dr_1 = \cos(\gamma)dr + \cos(\omega)d\tau - \tau\sin(\omega)d\omega$$
$$\sin(\gamma_1)dr_1 = \sin(\gamma)dr + \sin(\omega)d\tau + \tau\cos(\omega)d\omega$$

Donc:

$$\cos(\gamma_1)dr_1 - \cos(\gamma)dr = \cos(\omega)d\tau - \tau\sin(\omega)d\omega \tag{12}$$
$$\sin(\gamma_1)dr_1 - \sin(\gamma)dr = \sin(\omega)d\tau + \tau\cos(\omega)d\omega \tag{13}$$

A présent en faisant $\cos(\omega) \times (13) - \sin(\omega) \times (12)$ et utilisant la relation (9) on a :
Dans le membre de gauche :

$$-\cos(\omega)\sin(w-\psi)dr + \sin(\omega+\psi_1)\cos(\omega)dr_1 + sin(w)\cos(w-\psi)d - \sin(w)\cos(w+\psi_1)dr_1$$
$$=(\cos(\omega)\sin(w+\psi_1) - \sin(w)cos(w+\psi_1))dr_1 + (\sin(w)\cos(w-\psi) - \sin(\omega-\psi)\cos(\omega))dr$$
$$=\sin(w+\psi_1-\omega)dr_1 + \sin(w-(w-\psi))dr$$
$$=\sin(\psi_1)dr_1 + \sin(\psi)dr$$

Dans le membre de droite:

$$\cos(\omega)\sin(\omega)d\tau + \tau\cos^2(\omega)d\omega - \sin(\omega)\cos(\omega)d\tau + \tau\sin^2(\omega)d\omega = \tau d\omega$$

Cela nous donne alors:

$$\sin(\psi_1)dr_1 + \sin(\psi)dr = \tau d\omega \tag{14}$$

On utilise ensuite les égalités obtenue en (9) :

$$sin(\psi_1)dr_1 + sin(\psi)dr = -\tau\mathcal{K}dr + \tau d\psi \tag{15}$$
$$\iff cos(\phi_1)dr_1 + cos(\phi)dr = -\tau\mathcal{K}dr - \tau d\phi \tag{16}$$
$$\iff cos(\phi_1)\mathcal{K}_1 dr_1 = (-\tau\mathcal{K}_1\mathcal{K} - \cos(\phi)\mathcal{K}_1)dr - \mathcal{K}_1\tau d\phi \tag{17}$$
$$\iff \cos(\phi_1)(d\phi_1 + \mathcal{K}dr + d\phi) = (-\tau\mathcal{K}_1\mathcal{K} - \cos(\phi)\mathcal{K}_1)dr - \mathcal{K}_1\tau d\phi \tag{18}$$
$$\iff -cos(\phi_1)d\phi_1 = (\tau\mathcal{K}\mathcal{K}_1 + \mathcal{K}\cos(\phi_1) + \cos(\phi)\mathcal{K}_1)dr + (\mathcal{K}_1\tau + cos(\phi_1))d\phi \tag{19}$$

Enfin l'équation (16) et l'équation (19) nous donnent:

$$D_xT = \frac{-1}{\cos(\phi_1)} \begin{pmatrix} \tau\mathcal{K} + \cos(\phi) & \tau \\ \tau\mathcal{K}\mathcal{K}_1 + \mathcal{K}\cos(\phi_1) + \mathcal{K}_1\cos(\phi) & \tau\mathcal{K}_1 + \cos(\phi_1) \end{pmatrix}$$

□

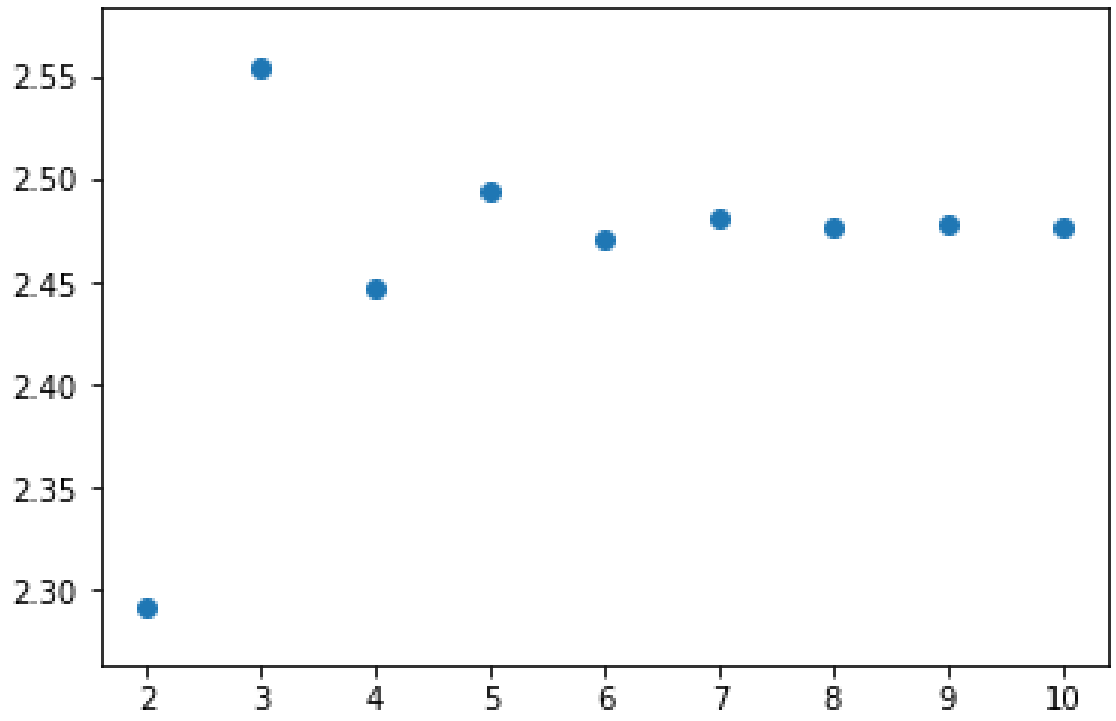
Sachant que l'ensemble des suites périodiques est dense dans K, une bonne estimation du coefficient d Lyapunov est de calculer :

$$\frac{1}{K} \sum_{\text{x point périodique de période n}} \ln(\lambda_+(x)) \tag{20}$$

Où $\lambda_+(x)$ est la plus grande valeur propre de de $DT^n(x)$ et K est le nombre de points périodiques de période n.
Pour x périodique de période n, on sait aussi que $\lambda_+(x)$ est le même pour les $T^i(x)$ pour i allant de 0 à p, p étant la longueur de la période primitive associée à l'orbite de x . Cela vient du fait que les matrices $D_{T^i(x)}T^n$ sont conjuguées entre elles. Ainsi au lieu de parler de $\lambda_+(x)$ on peut parler directement de $\lambda_+(s)$ où s est une période.
Ainsi on peut estimer le coefficient de Lyapunov avec la formule suivante :

$$\frac{1}{K} \sum_{\text{s : période de longueur n}} \ln(\lambda_+(x)) \times l(p) \tag{21}$$

Où $l(p)$ est la longueur de la période primitive associé au mot engendré par s. Au plus n est grand, au plus la précision de l'estimation du coefficient de Lyapunov est grande, c'est précisement comme cela que nous estimons le cofficient de liapunov pour notre système :



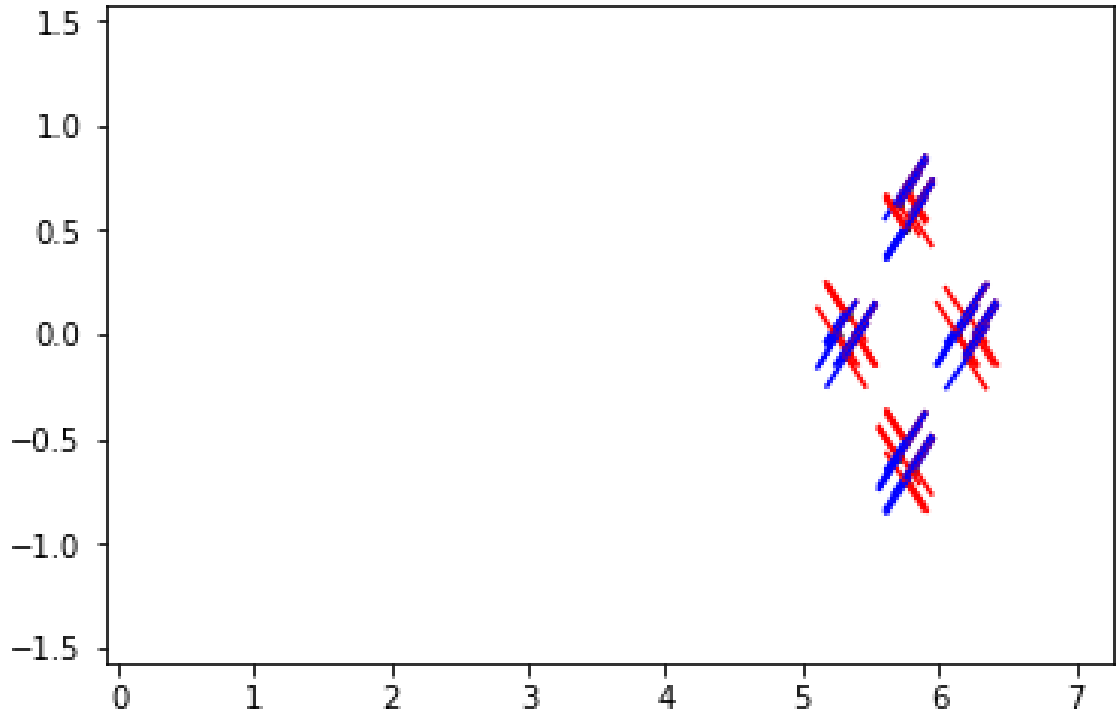
Exposant de Lyapunov pour les périodes 2 à 10 (Voir code C).

Nous pouvons maintenant tracer la courbe de l'exposant (trouvé avec la formule proposée) en fonction de la période fixée n des suites étudiées et regarder sa limite suffit pour determiner l'exposant de liapunov. En effet si on note E_n l'ensemble des suites de période n on a nécessairement:

$$\bigcup_{k=1}^{n} E_n \subset E_{n!} \tag{22}$$

S'intéresser donc au coefficient limite trouvé avec la moyenne sur les suites périodiques de périodes inférieures à n est équivalent à s'intéresser à la limite trouvée sur celle de période exactement n.
Notre exposant de Liapunov de notre système est environ égal à 2,47. ce qui est assez proche du 2,0 estimé précédemment.

Enfin pour x un point de K, D_xT^n indique comment T transforme l'espace. Les vecteurs propres associés à la plus grande et la plus petite valeur propre de D_xT^n sont respectivement une direction d'expansion et une autre de contraction. En récupérant et en traçant ces vecteurs dans l'espace des phases d'un obstacle, on peut se faire une idée de la déformation de K par T :



Représentation dans l'espace des phases des périodes de A_3 de taille 7 avec les vecteurs propres associés à la plus grande et la plus petite valeur propre de $DT^n(x)$, respectivement en rouge et en bleu (Voir code C).

A Codage de T

```
import matplotlib.pyplot as plt
import numpy as np
import random as rd

###Les centres des obstacles A, B, C ainsi que le rayon de chacune des ces boules

A=(0.5,3**0.5/2)
B=(0,0)
C=(1,0)
Rayons=[0.25,0.25,0.25]

Noms=['A','B','C']
obstacles=[A,B,C]
### Definition de la zone d'observation
Centre=(0.5,3**0.5/4)
u,v=Centre
Rlim=1.5

##Tirage d'un point de depart et son angle de vitesse aleatoire qui ne se trouve pas dans les disques obstacles



def test_tirage(x0,y0,obstacles,Rayons):# Teste si le point (x0,y0) se trouve dans un des obstacles
    #(x0,y0) est le point a tester
    #obstacles est la liste des centres des obstacles
    #Rayons est la liste des rayons respectifs des obstacles
    for i in range(len(obstacles)) :
        if np.linalg.norm(np.array(obstacles[i])-np.array((x0, y0)))<Rayons[i]:
            return True
    return False



def tirage_traj_general(Rlim,obstacles,Centre,Rayons):
    #Centre est le centre du domaine d'observation (c'est un carre de cote 2*Rlim et de "centre" Centre !)
    #Rlim est la demi longueur d'un cte du carre d'observation
    #obstacles est la liste des centres des obstacles
    #Rayons est la liste des rayons respectifs des obstacles
    u,v=Centre
    a=rd.uniform(u-Rlim,v+Rlim)
```

```
    b=rd.uniform(u-Rlim,v+Rlim)
    alpha=rd.uniform(-np.pi,np.pi)

    while test_tirage(a,b,obstacles,Rayons) :
        a=rd.uniform(u-Rlim,v+Rlim)
        b=rd.uniform(u-Rlim,v+Rlim)

    return a,b,alpha

##Fonction de traage des obstacles la fonction trace un cercle de centre (x1,x2) et de rayon r

def trace_cercle(x1,x2,r):
    #(x1,x2) le centre du cercle a tracer
    #r son rayon
    theta=np.linspace(0,2*np.pi,100)
    xx1=x1+r*np.cos(theta)
    xx2=x2+r*np.sin(theta)
    plt.plot(xx1,xx2)

##Calcul de l'intersection de la trajectoire avec la zone limite d'observation

def calculator_infini(x0,y0,alpha,Centre,Rlim):
    #(x0,y0) sont les coordonnees du point de depart
    #alpha est l'angle de la trajectoire (en radian dans  ]-pi,pi])
    #Centre est le centre du domaine d'observation (c'est un carre de cote 2*Rlim et de "centre" Centre !)
    #Rlim est la demi longueur d'un cte du carre d'observation
    u,v=Centre
    intersec=[]
    x1=(v+Rlim-y0)/np.tan(alpha)+x0
    if x1>=u-Rlim and x1<=u+Rlim :
        intersec.append((x1,v+Rlim))
    x2=(v-Rlim-y0)/np.tan(alpha)+x0
    if x2>=u-Rlim and x2<=u+Rlim :
        intersec.append((x2,v-Rlim))
    y3=np.tan(alpha)*(u+Rlim-x0)+y0
    if y3>=v-Rlim and y3<=v+Rlim :
         intersec.append((u+Rlim,y3))
    y4=np.tan(alpha)*(u-Rlim-x0)+y0
    if y4>=v-Rlim and y4<=v+Rlim :
         intersec.append((u-Rlim,y4))
    bordx0,bordy0=intersec[0]
    ps=np.cos(alpha)*(bordx0-x0)+np.sin(alpha)*(bordy0-y0)
    if ps>=0:
        return intersec[0]
```

```
85      return intersec[1]
86
87  ##Fonction qui trace un segment entre deux points
88
89  def trace_segment(point1,point2,color1):
90      #point1 est la premiere extremite du segment
91      #point2 est la seconde extremite du segment
92      a,b=point1
93      c,d=point2
94      plt.plot(np.array((a,c)),np.array((b,d)),color=color1)
95
96  #Fonction qui trace les obstacles
97
98  def trace_obstacles(obstacles,Noms,Rayons):
99      #obstacles est la liste des centres des obstacles
100     #Noms est la liste des noms respectifs des obstacles
101     #Rayons est la liste des rayons respectifs des obstacles
102     for i in range(len(obstacles)):
103         trace_cercle(*obstacles[i],Rayons[i])
104         plt.text(*obstacles[i],Noms[i],color='black')
105     plt.axis('equal')
106
107 ##Fonction qui remet un angle dans ]-pi,pi]
108
109 def remet_le(angle):
110     #angles est un angle en radian
111     while angle<=-np.pi:
112         angle+=2*np.pi
113     while angle>np.pi:
114         angle-=2*np.pi
115     return angle
116
117
118 ##Calcul de l'angle de rebond sur un obstacle circulaire
119
120 def angle_reflexion(a1,a2,xc,yc,alpha):
121     #(a1,a2) est le centre du cercle
122     #(xc,yc) est un point sur le cercle
123     #alpha est l'angle de la trajectoire (en radian dans  ]-pi,pi] )
124     return (remet_le(np.pi+2*np.arctan((a2-yc)/(a1-xc))-alpha))
125
126 ##Fonction qui calcule l'intersection entre la trajectoire et un des
    ↪  obstacles(elle renvoie aussi quel est l'obstacle), elle renvoie "infini"
    ↪  si il n'y en a pas
127
128 def calculator(obstacles,Rayons,x0,y0,alpha,delta):
129     #obstacles est la liste des centres des obstacles
```

```
130     #Rayons est la liste des rayons respectifs des obstacles
131     #(x0,y0) sont les coordonnees du point de depart
132     #alpha est l'angle de la trajectoire (en radian dans  ]-pi,pi] )
133     #delta est un parametre qui est propre a cette fonction et a la fonction
        ↪  suivante
134     #Centre est le centre du domaine d'observation (c'est un carre de cote
        ↪  2*Rlim et de "centre" Centre !)
135     #Rlim est la demi longueur d'un cte du carre d'observation
136     contact="infini"
137     record="infini"
138     boule="Pas de boule touchee"
139     for i in range(len(obstacles)) :
140         a1,a2=obstacles[i]
141         if abs(np.sin(alpha)*(a1-x0)-np.cos(alpha)*(a2-y0)) < Rayons[i] and
            ↪  np.cos(alpha)*(a1-x0)+np.sin(alpha)*(a2-y0)>delta:
142             a=1+np.tan(alpha)**2
143             b=-2*a1-2*(np.tan(alpha)**2)*x0+2*np.tan(alpha)*(y0-a2)
144             c=a1**2+(x0**2)*np.tan(alpha)**2-2*np.tan(alpha)*x0*(y0-a2)+(y0-
                ↪  a2)**2-Rayons[i]**2
145             det=b**2-4*a*c
146             racine1=(-b-np.sqrt(det))/(2*a)
147             racine2=(-b+np.sqrt(det))/(2*a)
148             y1=np.tan(alpha)*(racine1-x0)+y0
149             y2=np.tan(alpha)*(racine2-x0)+y0
150             if np.linalg.norm(np.array([x0,y0])-np.array([racine1,
                ↪  y1]))<np.linalg.norm(np.array([x0,y0])-np.array([racine2,
                ↪  y2])):
151                 contact=(racine1,y1)
152             else :
153                 contact=(racine2,y2)
154             if record=="infini" :
155                 record=contact
156                 boule=obstacles[i]
157             elif np.linalg.norm(np.array((x0,y0))-
                ↪  np.array(record))>np.linalg.norm(np.array((x0,y0))-
                ↪  np.array(contact)):
158                 record=contact
159                 boule=obstacles[i]
160             #print(record)
161     return record,boule
162
163
164 ##Fonction qui trace le segment entre le depart et l'intersection avec un
    ↪  obstacle ou la limite de la zone d'observation
165
166 def avance(obstacles,Rayons,x0,y0,alpha,delta,Centre,Rlim,color):
167     #obstacles est la liste des centres des obstacles
```

```
168     #Rayons est la liste des rayons respectifs des obstacles
169     #(x0,y0) sont les coordonnees du point de depart
170     #alpha est l'angle de la trajectoire (en radian dans  ]-pi,pi] )
171     #delta est un parametre qui est propre a cette fonction et a la fonction
        ↪  suivante
172     contact,_=calculator(obstacles,Rayons,x0,y0,alpha,delta)
173     if contact=='infini' :
174        trace_segment((x0,y0),calculator_infini(x0,y0,alpha,Centre,Rlim),
           ↪  color)
175
176     else:
177         trace_segment((x0,y0),contact,color)
178         #plt.plot(*contact,color='red',marker='o')
179
180 ##Fonction qui affiche la trajectoire d'un point sur le billard a partir d'un
    ↪  point et une direction de depart aleatoire
181
182 def rebond(obstacles,Rayons,Noms,Centre,Rlim):
183     #obstacles est la liste des centres des obstacles
184     #Rayons est la liste des rayons respectifs des obstacles
185     #Noms est la liste des noms respectifs des obstacles
186     #Centre est le centre du domaine d'observation (c'est un carre de cote
        ↪  2*Rlim et de "centre" Centre !)
187     #Rlim est la demi longueur d'un cte du carre d'observation
188     trace_obstacles(obstacles,Noms,Rayons)
189     x0,y0,alpha=tirage_traj_general(Rlim,obstacles,Centre,Rayons)
190     print(x0,y0,alpha)
191     plt.plot(x0,y0,color='red',marker='o')
192     u,boule=calculator(obstacles,Rayons,x0,y0,alpha,0)
193     n=0
194     avance(obstacles,Rayons,x0,y0,alpha,0,Centre,Rlim)
195     while u != 'infini' and n<20 :
196         a1,a2=boule
197         x0,y0,alpha=*u,angle_reflexion(a1,a2,*u,alpha)
198         u,boule=calculator(obstacles,Rayons,x0,y0,alpha,1e-6)
199         avance(obstacles,Rayons,x0,y0,alpha,1e-6,Centre,Rlim)
200         print("trajectoire:   x = ",u[0]," y = ",u[1]," alpha = ",u[2])
201         n+=1
202     avance(obstacles,Rayons,x0,y0,alpha,0,Centre,Rlim)
203     plt.show()
204
205 ##Fonction qui affiche la trajectoire d'un point sur le billard a partir d'un
    ↪  point et une direction de depart choisie
206
207 def rebond_deterministe(x0,y0,alpha,obstacles,Rayons,Noms,Centre,Rlim,color):
208     #(x0,y0) sont les coordonnees du point de depart
209     #alpha est l'angle de la trajectoire (en radian dans  ]-pi,pi] )
```

```
    #obstacles est la liste des centres des obstacles
    #Rayons est la liste des rayons respectifs des obstacles
    #Noms est la liste des noms respectifs des obstacles
    #Centre est le centre du domaine d'observation (c'est un carre de cote
    ↪  2*Rlim et de "centre" Centre !)
    #Rlim est la demi longueur d'un cte du carre d'observation
    trace_obstacles(obstacles,Noms,Rayons)
    plt.plot(x0,y0,color='orange',marker='o')
    u,boule=calculator(obstacles,Rayons,x0,y0,alpha,0)
    n=0
    avance(obstacles,Rayons,x0,y0,alpha,0,Centre,Rlim,color)
    while u != 'infini' and n<20 :
        a1,a2=boule
        x0,y0,alpha=*u,angle_reflexion(a1,a2,*u,alpha)
        #print('on repart   :',x0,y0,alpha)
        u,boule=calculator(obstacles,Rayons,x0,y0,alpha,1e-10)
        avance(obstacles,Rayons,x0,y0,alpha,1e-10,Centre,Rlim,color)
        #print(u)
        n+=1
    avance(obstacles,Rayons,x0,y0,alpha,0,Centre,Rlim,color)


##Fonction qui affiche la trajectoire d'un point sur le billard a partir d'un
↪  point et une direction de depart aleatoire parmi celles qui
↪  s'intersectent au moins une fois

def rebond_bon(obstacles,Rayons,Noms,Centre,Rlim,color):
    #alpha est l'angle de la trajectoire (en radian dans  ]-pi,pi] )
    #obstacles est la liste des centres des obstacles
    #Rayons est la liste des rayons respectifs des obstacles
    #Noms est la liste des noms respectifs des obstacles
    #Centre est le centre du domaine d'observation (c'est un carre de cote
    ↪  2*Rlim et de "centre" Centre !)
    #Rlim est la demi longueur d'un cte du carre d'observation
    trace_obstacles(obstacles,Noms,Rayons)
    x0,y0,alpha=tirage_traj_general(Rlim,obstacles,Centre,Rayons)
    u,boule=calculator(obstacles,Rayons,x0,y0,alpha,0)
    while u=='infini':
        x0,y0,alpha=tirage_traj_general(Rlim,obstacles,Centre,Rayons)
        u,boule=calculator(obstacles,Rayons,x0,y0,alpha,0)
    plt.plot(x0,y0,color='orange',marker='o')
    print("trajectoire:   x = ",x0," y = ",y0," alpha = ",alpha)
    n=0
    avance(obstacles,Rayons,x0,y0,alpha,0,Centre,Rlim,color)
    while u != 'infini' and n<20 :
        a1,a2=boule
        x0,y0,alpha=*u,angle_reflexion(a1,a2,*u,alpha)
```

```
253         u,boule=calculator(obstacles,Rayons,x0,y0,alpha,1e-6)
254         avance(obstacles,Rayons,x0,y0,alpha,1e-6,Centre,Rlim,color)
255         #print(u)
256         n+=1
257     avance(obstacles,Rayons,x0,y0,alpha,0,Centre,Rlim,color)
258     plt.show()
259 
260 ##Fonction qui affiche la trajectoire d'un point sur le billard a partir d'un
    ↪  point et une direction de depart aleatoire parmi celles qui
    ↪  s'intersectent au moins n fois
261 
262 #Fonction de comptage de rebonds
263 
264 def compte_rebonds(x0,y0,alpha,obstacles,Rayons):
265     #(x0,y0) sont les coordonnees du point de depart
266     #alpha est l'angle de la trajectoire (en radian dans  ]-pi,pi] )
267     #obstacles est la liste des centres des obstacles
268     #Rayons est la liste des rayons respectifs des obstacles
269     u,boule=calculator(obstacles,Rayons,x0,y0,alpha,0)
270     n=0
271     while u != 'infini' and n<100:
272        a1,a2=boule
273        x0,y0,alpha=*u,angle_reflexion(a1,a2,*u,alpha)
274        u,boule=calculator(obstacles,Rayons,x0,y0,alpha,1e-6)
275        n+=1
276     return n
277 
278 
279 ##Fonction qui trouve et trace une trajectoire avec au moins n rebonds
280 
281 def n_rebonds_svp(obstacles,Noms,Rayons,n,Centre,Rlim,color):
282     #obstacles est la liste des centres des obstacles
283     #Rayons est la liste des rayons respectifs des obstacles
284     #Noms est la liste des noms respectifs des obstacles
285     #Nombre minimal de rebonds sur les obstacles desires
286     #Centre est le centre du domaine d'observation (c'est un carre de cote
    ↪  2*Rlim et de "centre" Centre !)
287     #Rlim est la demi longueur d'un cte du carre d'observation
288     trace_obstacles(obstacles,Noms,Rayons)
289     x0,y0,alpha=tirage_traj_general(Rlim,obstacles,Centre,Rayons)
290     while compte_rebonds(x0,y0,alpha,obstacles,Rayons)<n:
291         x0,y0,alpha=tirage_traj_general(Rlim,obstacles,Centre,Rayons)
292     print("  x = ",x0," y = ",y0," alpha = ",alpha)
293     print("nb_rebond",compte_rebonds(x0,y0,alpha,obstacles,Rayons))
294     plt.plot(x0,y0,color='orange',marker='o')
295     u,boule=calculator(obstacles,Rayons,x0,y0,alpha,0)
296     n=0
```

```
    avance(obstacles,Rayons,x0,y0,alpha,0,Centre,Rlim,color)
    while u != 'infini' and n<20 :
        a1,a2=boule
        x0,y0,alpha=*u,angle_reflexion(a1,a2,*u,alpha)
        #print('on repart   :',x0,y0,alpha)
        u,boule=calculator(obstacles,Rayons,x0,y0,alpha,1e-10)
        avance(obstacles,Rayons,x0,y0,alpha,1e-10,Centre,Rlim,color)
        #print(u)
        n+=1
    avance(obstacles,Rayons,x0,y0,alpha,0,Centre,Rlim,color)



#n_rebonds_svp(obstacles,Noms,Rayons,6,Centre,Rlim)


###Rajoutons des obstacles


o_1=(-0.5,3**0.5/2)
o_2=(0.5,3**0.5/2)
o_3=(-1,0)
o_4=(0,0)
o_5=(1,0)
o_6=(-0.5,-3**0.5/2)
o_7=(0.5,-3**0.5/2)

Noms2=[str(i) for i in range(1,8)]
obstacles2=[o_1,o_2,o_3,o_4,o_5,o_6,o_7]
Centre2=(0,0)
Rlim2=1.5
Rayons2=[0.25 for  i in range(7)]
Rayons3=[0.3,0.35,0.3,0.4,0.2,0.25,0.15]




#n_rebonds_svp(obstacles2,Noms2,Rayons3,23,Centre2,Rlim2)


#Le point de depart est en orange
#Le point de sortie de la trajectoire est l'extremite non marquee

#rebond_deterministe(-0.14771195289118022,0.41724651296907145,
↪  0.47435283021459895,obstacles,Rayons,Noms,Centre,Rlim)
```

```
#rebond_deterministe(-0.14771195289118022,0.41724651296907145,0.48,
↪  obstacles,Rayons,Noms,Centre,Rlim)
#Pour lancer un point (x0,y0) avec un angle alpha donne

print("Voici une trajectoire d'au moins 8 rebonds : " )
n_rebonds_svp(obstacles,Noms,Rayons,8,Centre,Rlim,'red')
plt.figure()
plt.show()

##fonction qui trace deux trajectoires l'une en bleue l'autre en rouge

def trace_rebonds_deterministe(a,b,c,d,e,f,obstacles,Rayons,Noms,Centre,
↪  Rlim):
    rebond_deterministe(a,b,c,obstacles,Rayons,Noms,Centre,Rlim,"red")
    print("1   :", compte_rebonds(a,b,c,obstacles,Rayons))
    rebond_deterministe(d,e,f,obstacles,Rayons,Noms,Centre,Rlim,"blue")
    print("2   :", compte_rebonds(d,e,f,obstacles,Rayons))
    plt.show()




print("Voici deux trajectoires proches l'une de l'autre :  ")

trace_rebonds_deterministe(0.667482841472288,  0.10452278926045411,
↪  0.10098578721700369,0.667482841472288,  0.10452278926045411,
↪  0.10098578721700369+10**(-5),obstacles,Rayons,Noms,Centre,Rlim)
plt.figure()
```

B Détermination des trajectoires périodiques

```python
import numpy as np
import numpy.random as rnd
import matplotlib.pyplot as plt

##Le but ici est d'effectuer un descente de gradient projette afin de trouver les trajectoires associees a n'importe quel mot admissible

mot=[0,1,2]
obstacles0=np.array([(0,0),(1,0),(0.5,3**0.5/2)])
rayons0=np.array([0.25,0.25,0.25])
matrice_transition0=np.array([[0,1,1],[1,0,1],[1,1,0]])


##Fonction qui verifie si le mot est admissible pour la matrice de transition associee au systeme

def admissible(mot,matrice_transition,long_mot):
    #mot est une liste d'elements de obstacles qui forme un mot admissible pour la matrice de transition
    #matrice_transition est la matrice de transition associee au systeme
    #long_mot est la longueur de la liste mot
  for i in range(long_mot):
    if matrice_transition[mot[i%long_mot]][mot[(i+1)%long_mot]]==0:
      return False
  return True

##Fonction qui projette un vecteur Y sur l'ensemble produit des cercles mot[i]

def proj_vecteur(mot,Y,obstacles,rayons,long_mot):
    #mot est une liste d'elements de obstacles qui forme un mot admissible pour la matrice de transition
    #Y est le vecteur a projeter
    #obstacles est la liste des centres des obstacles
    #rayons est la liste des rayons respectifs des obstacles
    #long_mot est la longueur de la liste mot
  for i in range(long_mot):
    x=proj_couple(np.array((Y[2*i],Y[2*i+1])),np.array(obstacles[mot[i]]), rayons[mot[i]])
    Y[2*i]=x[0]
    Y[2*i+1]=x[1]
  return Y

def proj_couple(x,centre,rayon):
```

```
    if np.array(x)[0]==np.array(centre)[0] and
    ↪  np.array(x)[1]==np.array(centre)[1]:
        theta=2*rnd.random()*np.pi
        return centre+rayon*np.array([np.sin(theta),np.cos(theta)])
    else:
        return centre+rayon*(x-centre)/np.linalg.norm(x-centre)

##Fonction gradient de l'efficacite

def grad_E(Y,long_mot):
    #Y est le parametre de la fonction gradient
    #long_mot est la longueur du mot
  grad=[]
  for i in range (long_mot):
    i0=(i-1)%long_mot
    i1=i
    i2=(i+1)%long_mot
    x0=Y[2*i0]
    y0=Y[2*i0+1]
    x1=Y[2*i1]
    y1=Y[2*i1+1]
    x2=Y[2*i2]
    y2=Y[2*i2+1]
    N0=np.sqrt((x0-x1)**2+(y0-y1)**2)
    N1=np.sqrt((x1-x0)**2+(y1-y0)**2)
    grad.append((x1-x2)/N1+(x1-x0)/N0)
    grad.append((y1-y2)/N1+(y1-y0)/N0)
  return np.array(grad)

##Calcul d'une constante du pas qui optimise la complexite de la descente de
↪  gradient

def distance_minmax_entre_obstacles(obstacles,rayons,matrice_transition):
    #matrice_transition est la matrice de transition du systeme
    #obstacles est la liste des centres des obstacles
    #rayons est la liste des rayons respectifs des obstacles
    record_max=np.linalg.norm(np.array(obstacles[0])-
    ↪  np.array(obstacles[1]))+rayons[0]+rayons[1]
    record_min=np.linalg.norm(np.array(obstacles[0])-
    ↪  np.array(obstacles[1]))-rayons[0]-rayons[1]
    p=len(obstacles)
    for i in range(p):
        for j in range(p):
            mot=[i,j]
            if admissible(mot,matrice_transition,2) :
```

```
                distance=np.linalg.norm(np.array(obstacles[mot[0]])-
                ↪  np.array(obstacles[mot[1]]))-rayons[mot[0]]-
                ↪  rayons[mot[1]]
                if distance<record_min:
                    record_min=distance
                elif distance > record_max:
                    record_max=distance
    return record_min, record_max


def constante_de_lip(obstacles,rayons,matrice_transition,k):
    #matrice_transition est la matrice de transition du systeme
    #obstacles est la liste des centres des obstacles
    #rayons est la liste des rayons respectifs des obstacles
    record_min, record_max=distance_minmax_entre_obstacles(obstacles,rayons,
    ↪  matrice_transition)
    return np.sqrt(k)*np.sqrt(4+(record_max**2)/record_min**2)

Lip=constante_de_lip(obstacles0,rayons0,matrice_transition0,len(mot))


##Fonction de gradient projete pour trouver un argmin de l'efficacite.

#On considere qu'on se place dans la condition de non eclipsite
def projected_gradient_descent(mot,pas=0.0001,
↪  matrice_transition=np.array([[0,1,1],[1,0,1],[1,1,0]]),
↪  obstacles=np.array([(0,0),(1,0),(0.5,3**0.5/2)]),rayons=np.array([0.25,
↪  0.25,0.25]),epsilon=1e-6,nbre_it=1000):
    #mot est une liste d'elements de obstacles qui forme un mot admissible
    ↪  pour la matrice de transition
    #pas est le pas de la descente de gradient projete
    #matrice_transition est la matrice de transition du systeme
    #obstacles est la liste des centres des obstacles
    #rayons est la liste des rayons respectifs des obstacles
    #esplison est le niveau de precision (pour la condition d'arrt)
    #nbre_it est le nombre d'iterations maximal du gradient projete (pour la
    ↪  condition d'arrt)
  long_mot=len(mot)
  if not admissible(mot,matrice_transition,long_mot):
    return False
  theta=[rnd.random()*2*np.pi for i in range (long_mot)]
  Y=[]
  for i in range(long_mot):
    Y.append(obstacles[mot[i]][0]+np.cos(theta[i])*rayons[mot[i]])
    Y.append(obstacles[mot[i]][1]+np.sin(theta[i])*rayons[mot[i]])
  compteur=0
  Y=np.array(Y)
```

```
  liste_rep=[Y]
  while compteur==0 or (compteur<=nbre_it and
  ↪  np.linalg.norm(np.array(liste_rep[-1])-np.array(liste_rep[-
  ↪  2]))>epsilon):
    Y_int=Y-(1/Lip)*grad_E(Y,long_mot)
    Y=np.copy(proj_vecteur(mot,Y_int,obstacles,rayons,long_mot))
    liste_rep.append(Y)
    compteur+=1
  #print("nombre d'iterations :",compteur)
  return Y


##Fonction recursive qui renvoie la liste des mots de longueur: 'longueur'
↪  dans un alphabet a p elements

def enum_periode(longueur,p):
    def recurs(longueurafaire,p,reponse):
        if longueurafaire==0:
            return []
        rep=[]
        for x in reponse :
            for i in range(p):
                rep.append(x+[i])
        return rep+recurs(longueurafaire-1,p,rep)
    return [[i] for i in range(p)]+recurs(longueur-1,p,[[i] for i in
    ↪  range(p)])


##Applique une rotation sur un mot

def rotate(l,n):
    return l[n:]+l[:n]

#A partir d'un mot on cree la liste de ses rotations et de sa periode
↪  primitive


def rotation2(mot,long,N):
    liste=[mot]
    if long>1:
        for i in range(1,long) :
            m=mot.copy()
            m=rotate(m,i)
            liste.append(m)
    K=N//long
    for i in range(len(liste)):
        for k in range(2,K+1):
```

```
            m=liste[i].copy()
            m*=k
            liste.append(m)
    return liste


##Tri la liste de enum periode
def tri(liste_totale,matrice_transition,N):
    cpt=0
    l=len(liste_totale)
    while cpt<l:
        mot=liste_totale[cpt]
        long=len(mot)

        liste=rotation2(mot,long,N)

        for i in range(len(liste)):
            if liste[i] in liste_totale and liste[i]!=mot  :
                liste_totale.remove(liste[i])
        cpt+=1
        l=len(liste_totale)
    listerep=liste_totale.copy()

    for mot1 in liste_totale :

        if not(admissible(mot1,matrice_transition,len(mot1))):
            listerep.remove(mot1)

    return listerep

def liste_div(n):
    l=[]
    for i in range(2,n//2+1):
        if n%i==0:
            l.append(i)
    return l


def repere_periode_primitive(mot):
    n=len(mot)
    if n<=3:
        return mot
    liste_diviseurs=liste_div(n)
    for k in liste_diviseurs :
        if (n//k)*mot[:k]==mot:
            return mot[:k]
```

```
208     return mot
209 
210 
211 ##Fonction de traage de cercle
212 
213 def trace_cercle(x1,x2,r):
214     #(x1,x2) le centre du cercle a tracer
215     #r son rayon
216     theta=np.linspace(0,2*np.pi,100)
217     xx1=x1+r*np.cos(theta)
218     xx2=x2+r*np.sin(theta)
219     plt.plot(xx1,xx2)
220 
221 #Fonction qui trace les obstacles
222 
223 def trace_obstacles(obstacles,Rayons):
224     #obstacles est la liste des centres des obstacles
225     #Rayons est la liste des rayons respectifs des obstacles
226     for i in range(len(obstacles)):
227         trace_cercle(*obstacles[i],Rayons[i])
228         plt.text(*obstacles[i],str(i),color='black')
229     plt.axis('equal')
230 
231 ##Fonction qui trace un segment entre deux points
232 
233 def trace_segment(point1,point2):
234     #point1 est la premiere extremite du segment
235     #point2 est la seconde extremite du segment
236     a,b=point1
237     c,d=point2
238     plt.plot(np.array((a,c)),np.array((b,d)),color='purple')
239 
240 ##Fonction qui trace un segment entre deux points
241 
242 def trace_segment3(a,b,c,d,colori):
243     #point1 est la premiere extremite du segment
244     #point2 est la seconde extremite du segment
245     plt.plot(np.array((a,c)),np.array((b,d)),color=colori)
246 
247 ##Fonction qui trace un segment entre deux points en pointille
248 
249 def trace_segment2(point1,point2):
250     #point1 est la premiere extremite du segment
251     #point2 est la seconde extremite du segment
252     a,b=point1
253     c,d=point2
254     plt.plot(np.array((a,c)),np.array((b,d)),'r--',color='orange')
```

```
255
256
257 ##Fonction qui trace la periode
258
259
260 def trace_periode(liste_rep,obstacles,Rayons):
261     #liste_rep est sous la forme [x_0,y_0,...,x_k,y_k]
262     #obstacles est la liste des centres des obstacles
263     #Rayons est la liste des rayons respectifs des obstacles
264     if type(liste_rep)==type(True):
265         #print( 'mot non admissible')
266         return False
267     trace_obstacles(obstacles,Rayons)
268     long=len(liste_rep)
269     for i in range(int(long/2)):
270         trace_segment((liste_rep[2*(i%int(long/2))],liste_rep[2*(i%
            int(long/2))+1]),(liste_rep[2*((i+1)%int(long/2))],liste_rep[2*
            ((i+1)%int(long/2))+1]))
271         plt.plot(liste_rep[2*(i%int(long/2))],liste_rep[2*(i%int(long/2))+
            1],color='red',marker='o')
272         n1=np.array((liste_rep[2*((i-1)%int(long/2))],liste_rep[2*((i-1)%
            int(long/2))+1]))-np.array((liste_rep[2*(i%int(long/2))],
            liste_rep[2*(i%int(long/2))+1]))
273         n2=np.array((liste_rep[2*((i+1)%int(long/2))],liste_rep[2*((i+1)%
            int(long/2))+1]))-np.array((liste_rep[2*(i%int(long/2))],
            liste_rep[2*(i%int(long/2))+1]))
274         normal=np.linalg.norm(n1+n2)**(-1)*(n1+n2)
275         trace_segment2((liste_rep[2*(i%int(long/2))],liste_rep[2*(i%
            int(long/2))+1]),np.array((liste_rep[2*(i%int(long/2))],
            liste_rep[2*(i%int(long/2))+1])+0.2*normal))
276
277
278
279
280 ##Fonction qui trace toutes les periodes de longueur N et inferieur
281
282 def trace_periode_N(N,obstacles,rayons,matrice_transition):
283     #obstacles est la liste des centres des obstacles
284     #rayons est la liste des rayons respectifs des obstacles
285     #matrice_transition est la matrice de transition du systeme
286     p=len(obstacles)
287     liste_totale=enum_periode(N,p)
288     liste_totale=tri(liste_totale,matrice_transition,N)
289     for mot in tri(liste_totale,matrice_transition,N):
290         #long=len(mot)
```

```
        liste_rep=projected_gradient_descent(mot,pas=0.001,
        ↪  matrice_transition=matrice_transition0,obstacles=obstacles0,
        ↪  rayons=rayons0,epsilon=1e-6,nbre_it=10000)
        trace_periode(liste_rep,obstacles,rayons)
    plt.show()

##Fonction qui trace toutes les periodes de longueur N exactement

def trace_periode_N_exact(N,obstacles,rayons,matrice_transition):
    #obstacles est la liste des centres des obstacles
    #rayons est la liste des rayons respectifs des obstacles
    #matrice_transition est la matrice de transition du systeme
    p=len(obstacles)
    liste_totale=enum_periode(N,p)
    for mot in tri(liste_totale,matrice_transition,N):
        long=len(mot)
        if long==N:
            liste_rep=projected_gradient_descent(mot,pas=0.001,
            ↪  matrice_transition=matrice_transition0,obstacles=obstacles0,
            ↪  rayons=rayons0,epsilon=1e-6,nbre_it=10000)
            trace_periode(liste_rep,obstacles,rayons)
    plt.show()



print("Trace des trajectoires de periode 3 et moins ")

trace_periode_N(3,obstacles0,rayons0,matrice_transition0)
plt.figure()

print("Trace des trajectoires de periode 5 ")

trace_periode_N_exact(5,obstacles0,rayons0,matrice_transition0)
plt.figure()
```

C Calcul du coefficient de Lyapunov

```
import matplotlib.pyplot as plt
import numpy as np
import random as rd

###Les centres des obstacles A, B, C ainsi que le rayon de chacune des ces boules

A=(0.5,3**0.5/2)
B=(0,0)
C=(1,0)
Rayons=[0.25,0.25,0.25]

Noms=['A','B','C']
obstacles=[A,B,C]
### Definition de la zone d'observation
Centre=(0.5,3**0.5/4)
u,v=Centre
Rlim=1.5

##Tirage d'un point de depart et son angle de vitesse aleatoire qui ne se trouve pas dans les disques obstacles



def test_tirage(x0,y0,obstacles,Rayons):# Teste si le point (x0,y0) se trouve dans un des obstacles
    #(x0,y0) est le point a tester
    #obstacles est la liste des centres des obstacles
    #Rayons est la liste des rayons respectifs des obstacles
    for i in range(len(obstacles)) :
        if np.linalg.norm(np.array(obstacles[i])-np.array((x0, y0)))<Rayons[i]:
            return True
    return False



def tirage_traj_general(Rlim,obstacles,Centre,Rayons):
    #Centre est le centre du domaine d'observation (c'est un carre de cote 2*Rlim et de "centre" Centre !)
    #Rlim est la demi longueur d'un cte du carre d'observation
    #obstacles est la liste des centres des obstacles
    #Rayons est la liste des rayons respectifs des obstacles
    u,v=Centre
    a=rd.uniform(u-Rlim,v+Rlim)
```

```
    b=rd.uniform(u-Rlim,v+Rlim)
    alpha=rd.uniform(-np.pi,np.pi)

    while test_tirage(a,b,obstacles,Rayons) :
        a=rd.uniform(u-Rlim,v+Rlim)
        b=rd.uniform(u-Rlim,v+Rlim)

    return a,b,alpha

##Fonction de traage des obstacles la fonction trace un cercle de centre (x1,x2) et de rayon r

def trace_cercle(x1,x2,r):
    #(x1,x2) le centre du cercle a tracer
    #r son rayon
    theta=np.linspace(0,2*np.pi,100)
    xx1=x1+r*np.cos(theta)
    xx2=x2+r*np.sin(theta)
    plt.plot(xx1,xx2)

##Calcul de l'intersection de la trajectoire avec la zone limite d'observation

def calculator_infini(x0,y0,alpha,Centre,Rlim):
    #(x0,y0) sont les coordonnees du point de depart
    #alpha est l'angle de la trajectoire (en radian dans  ]-pi,pi])
    #Centre est le centre du domaine d'observation (c'est un carre de cote 2*Rlim et de "centre" Centre !)
    #Rlim est la demi longueur d'un cte du carre d'observation
    u,v=Centre
    intersec=[]
    x1=(v+Rlim-y0)/np.tan(alpha)+x0
    if x1>=u-Rlim and x1<=u+Rlim :
        intersec.append((x1,v+Rlim))
    x2=(v-Rlim-y0)/np.tan(alpha)+x0
    if x2>=u-Rlim and x2<=u+Rlim :
        intersec.append((x2,v-Rlim))
    y3=np.tan(alpha)*(u+Rlim-x0)+y0
    if y3>=v-Rlim and y3<=v+Rlim :
         intersec.append((u+Rlim,y3))
    y4=np.tan(alpha)*(u-Rlim-x0)+y0
    if y4>=v-Rlim and y4<=v+Rlim :
         intersec.append((u-Rlim,y4))
    bordx0,bordy0=intersec[0]
    ps=np.cos(alpha)*(bordx0-x0)+np.sin(alpha)*(bordy0-y0)
    if ps>=0:
        return intersec[0]
```

```
85      return intersec[1]
86
87  ##Fonction qui trace un segment entre deux points
88
89  def trace_segment(point1,point2,color1):
90      #point1 est la premiere extremite du segment
91      #point2 est la seconde extremite du segment
92      a,b=point1
93      c,d=point2
94      plt.plot(np.array((a,c)),np.array((b,d)),color=color1)
95
96  #Fonction qui trace les obstacles
97
98  def trace_obstacles(obstacles,Noms,Rayons):
99      #obstacles est la liste des centres des obstacles
100     #Noms est la liste des noms respectifs des obstacles
101     #Rayons est la liste des rayons respectifs des obstacles
102     for i in range(len(obstacles)):
103         trace_cercle(*obstacles[i],Rayons[i])
104         plt.text(*obstacles[i],Noms[i],color='black')
105     plt.axis('equal')
106
107 ##Fonction qui remet un angle dans ]-pi,pi]
108
109 def remet_le(angle):
110     #angles est un angle en radian
111     while angle<=-np.pi:
112         angle+=2*np.pi
113     while angle>np.pi:
114         angle-=2*np.pi
115     return angle
116
117
118 ##Calcul de l'angle de rebond sur un obstacle circulaire
119
120 def angle_reflexion(a1,a2,xc,yc,alpha):
121     #(a1,a2) est le centre du cercle
122     #(xc,yc) est un point sur le cercle
123     #alpha est l'angle de la trajectoire (en radian dans  ]-pi,pi] )
124     return (remet_le(np.pi+2*np.arctan((a2-yc)/(a1-xc))-alpha))
125
126 ##Fonction qui calcule l'intersection entre la trajectoire et un des
    ↪  obstacles(elle renvoie aussi quel est l'obstacle), elle renvoie "infini"
    ↪  si il n'y en a pas
127
128 def calculator(obstacles,Rayons,x0,y0,alpha,delta):
129     #obstacles est la liste des centres des obstacles
```

```
130     #Rayons est la liste des rayons respectifs des obstacles
131     #(x0,y0) sont les coordonnees du point de depart
132     #alpha est l'angle de la trajectoire (en radian dans  ]-pi,pi] )
133     #delta est un parametre qui est propre a cette fonction et a la fonction
        ↪  suivante
134     #Centre est le centre du domaine d'observation (c'est un carre de cote
        ↪  2*Rlim et de "centre" Centre !)
135     #Rlim est la demi longueur d'un cte du carre d'observation
136     contact="infini"
137     record="infini"
138     boule="Pas de boule touchee"
139     for i in range(len(obstacles)) :
140         a1,a2=obstacles[i]
141         if abs(np.sin(alpha)*(a1-x0)-np.cos(alpha)*(a2-y0)) < Rayons[i] and
            ↪  np.cos(alpha)*(a1-x0)+np.sin(alpha)*(a2-y0)>delta:
142             a=1+np.tan(alpha)**2
143             b=-2*a1-2*(np.tan(alpha)**2)*x0+2*np.tan(alpha)*(y0-a2)
144             c=a1**2+(x0**2)*np.tan(alpha)**2-2*np.tan(alpha)*x0*(y0-a2)+(y0-
                ↪  a2)**2-Rayons[i]**2
145             det=b**2-4*a*c
146             racine1=(-b-np.sqrt(det))/(2*a)
147             racine2=(-b+np.sqrt(det))/(2*a)
148             y1=np.tan(alpha)*(racine1-x0)+y0
149             y2=np.tan(alpha)*(racine2-x0)+y0
150             if np.linalg.norm(np.array([x0,y0])-np.array([racine1,
                ↪  y1]))<np.linalg.norm(np.array([x0,y0])-np.array([racine2,
                ↪  y2])):
151                 contact=(racine1,y1)
152             else :
153                 contact=(racine2,y2)
154             if record=="infini" :
155                 record=contact
156                 boule=obstacles[i]
157             elif np.linalg.norm(np.array((x0,y0))-
                ↪  np.array(record))>np.linalg.norm(np.array((x0,y0))-
                ↪  np.array(contact)):
158                 record=contact
159                 boule=obstacles[i]
160             #print(record)
161     return record,boule
162
163
164 ##Fonction qui trace le segment entre le depart et l'intersection avec un
    ↪  obstacle ou la limite de la zone d'observation
165
166 def avance(obstacles,Rayons,x0,y0,alpha,delta,Centre,Rlim,color):
167     #obstacles est la liste des centres des obstacles
```

```
    #Rayons est la liste des rayons respectifs des obstacles
    #(x0,y0) sont les coordonnees du point de depart
    #alpha est l'angle de la trajectoire (en radian dans  ]-pi,pi] )
    #delta est un parametre qui est propre a cette fonction et a la fonction suivante
    contact,_=calculator(obstacles,Rayons,x0,y0,alpha,delta)
    if contact=='infini' :
       trace_segment((x0,y0),calculator_infini(x0,y0,alpha,Centre,Rlim), color)

    else:
        trace_segment((x0,y0),contact,color)
        #plt.plot(*contact,color='red',marker='o')

##Fonction qui affiche la trajectoire d'un point sur le billard a partir d'un point et une direction de depart aleatoire

def rebond(obstacles,Rayons,Noms,Centre,Rlim):
    #obstacles est la liste des centres des obstacles
    #Rayons est la liste des rayons respectifs des obstacles
    #Noms est la liste des noms respectifs des obstacles
    #Centre est le centre du domaine d'observation (c'est un carre de cote 2*Rlim et de "centre" Centre !)
    #Rlim est la demi longueur d'un cte du carre d'observation
    trace_obstacles(obstacles,Noms,Rayons)
    x0,y0,alpha=tirage_traj_general(Rlim,obstacles,Centre,Rayons)
    print(x0,y0,alpha)
    plt.plot(x0,y0,color='red',marker='o')
    u,boule=calculator(obstacles,Rayons,x0,y0,alpha,0)
    n=0
    avance(obstacles,Rayons,x0,y0,alpha,0,Centre,Rlim)
    while u != 'infini' and n<20 :
        a1,a2=boule
        x0,y0,alpha=*u,angle_reflexion(a1,a2,*u,alpha)
        u,boule=calculator(obstacles,Rayons,x0,y0,alpha,1e-6)
        avance(obstacles,Rayons,x0,y0,alpha,1e-6,Centre,Rlim)
        print("trajectoire:   x = ",u[0]," y = ",u[1]," alpha = ",u[2])
        n+=1
    avance(obstacles,Rayons,x0,y0,alpha,0,Centre,Rlim)
    plt.show()

##Fonction qui affiche la trajectoire d'un point sur le billard a partir d'un point et une direction de depart choisie

def rebond_deterministe(x0,y0,alpha,obstacles,Rayons,Noms,Centre,Rlim,color):
    #(x0,y0) sont les coordonnees du point de depart
    #alpha est l'angle de la trajectoire (en radian dans  ]-pi,pi] )
```

```
    #obstacles est la liste des centres des obstacles
    #Rayons est la liste des rayons respectifs des obstacles
    #Noms est la liste des noms respectifs des obstacles
    #Centre est le centre du domaine d'observation (c'est un carre de cote
    ↪  2*Rlim et de "centre" Centre !)
    #Rlim est la demi longueur d'un cte du carre d'observation
    trace_obstacles(obstacles,Noms,Rayons)
    plt.plot(x0,y0,color='orange',marker='o')
    u,boule=calculator(obstacles,Rayons,x0,y0,alpha,0)
    n=0
    avance(obstacles,Rayons,x0,y0,alpha,0,Centre,Rlim,color)
    while u != 'infini' and n<20 :
        a1,a2=boule
        x0,y0,alpha=*u,angle_reflexion(a1,a2,*u,alpha)
        #print('on repart   :',x0,y0,alpha)
        u,boule=calculator(obstacles,Rayons,x0,y0,alpha,1e-10)
        avance(obstacles,Rayons,x0,y0,alpha,1e-10,Centre,Rlim,color)
        #print(u)
        n+=1
    avance(obstacles,Rayons,x0,y0,alpha,0,Centre,Rlim,color)


##Fonction qui affiche la trajectoire d'un point sur le billard a partir d'un
↪  point et une direction de depart aleatoire parmi celles qui
↪  s'intersectent au moins une fois

def rebond_bon(obstacles,Rayons,Noms,Centre,Rlim,color):
    #alpha est l'angle de la trajectoire (en radian dans  ]-pi,pi] )
    #obstacles est la liste des centres des obstacles
    #Rayons est la liste des rayons respectifs des obstacles
    #Noms est la liste des noms respectifs des obstacles
    #Centre est le centre du domaine d'observation (c'est un carre de cote
    ↪  2*Rlim et de "centre" Centre !)
    #Rlim est la demi longueur d'un cte du carre d'observation
    trace_obstacles(obstacles,Noms,Rayons)
    x0,y0,alpha=tirage_traj_general(Rlim,obstacles,Centre,Rayons)
    u,boule=calculator(obstacles,Rayons,x0,y0,alpha,0)
    while u=='infini':
        x0,y0,alpha=tirage_traj_general(Rlim,obstacles,Centre,Rayons)
        u,boule=calculator(obstacles,Rayons,x0,y0,alpha,0)
    plt.plot(x0,y0,color='orange',marker='o')
    print("trajectoire:   x = ",x0," y = ",y0," alpha = ",alpha)
    n=0
    avance(obstacles,Rayons,x0,y0,alpha,0,Centre,Rlim,color)
    while u != 'infini' and n<20 :
        a1,a2=boule
        x0,y0,alpha=*u,angle_reflexion(a1,a2,*u,alpha)
```

```
253         u,boule=calculator(obstacles,Rayons,x0,y0,alpha,1e-6)
254         avance(obstacles,Rayons,x0,y0,alpha,1e-6,Centre,Rlim,color)
255         #print(u)
256         n+=1
257     avance(obstacles,Rayons,x0,y0,alpha,0,Centre,Rlim,color)
258     plt.show()
259 
260 ##Fonction qui affiche la trajectoire d'un point sur le billard a partir d'un
    ↪  point et une direction de depart aleatoire parmi celles qui
    ↪  s'intersectent au moins n fois
261 
262 #Fonction de comptage de rebonds
263 
264 def compte_rebonds(x0,y0,alpha,obstacles,Rayons):
265     #(x0,y0) sont les coordonnees du point de depart
266     #alpha est l'angle de la trajectoire (en radian dans  ]-pi,pi] )
267     #obstacles est la liste des centres des obstacles
268     #Rayons est la liste des rayons respectifs des obstacles
269     u,boule=calculator(obstacles,Rayons,x0,y0,alpha,0)
270     n=0
271     while u != 'infini' and n<100:
272        a1,a2=boule
273        x0,y0,alpha=*u,angle_reflexion(a1,a2,*u,alpha)
274        u,boule=calculator(obstacles,Rayons,x0,y0,alpha,1e-6)
275        n+=1
276     return n
277 
278 
279 ##Fonction qui trouve et trace une trajectoire avec au moins n rebonds
280 
281 def n_rebonds_svp(obstacles,Noms,Rayons,n,Centre,Rlim,color):
282     #obstacles est la liste des centres des obstacles
283     #Rayons est la liste des rayons respectifs des obstacles
284     #Noms est la liste des noms respectifs des obstacles
285     #Nombre minimal de rebonds sur les obstacles desires
286     #Centre est le centre du domaine d'observation (c'est un carre de cote
    ↪  2*Rlim et de "centre" Centre !)
287     #Rlim est la demi longueur d'un cte du carre d'observation
288     trace_obstacles(obstacles,Noms,Rayons)
289     x0,y0,alpha=tirage_traj_general(Rlim,obstacles,Centre,Rayons)
290     while compte_rebonds(x0,y0,alpha,obstacles,Rayons)<n:
291         x0,y0,alpha=tirage_traj_general(Rlim,obstacles,Centre,Rayons)
292     print("  x = ",x0," y = ",y0," alpha = ",alpha)
293     print("nb_rebond",compte_rebonds(x0,y0,alpha,obstacles,Rayons))
294     plt.plot(x0,y0,color='orange',marker='o')
295     u,boule=calculator(obstacles,Rayons,x0,y0,alpha,0)
296     n=0
```

```
    avance(obstacles,Rayons,x0,y0,alpha,0,Centre,Rlim,color)
    while u != 'infini' and n<20 :
        a1,a2=boule
        x0,y0,alpha=*u,angle_reflexion(a1,a2,*u,alpha)
        #print('on repart   :',x0,y0,alpha)
        u,boule=calculator(obstacles,Rayons,x0,y0,alpha,1e-10)
        avance(obstacles,Rayons,x0,y0,alpha,1e-10,Centre,Rlim,color)
        #print(u)
        n+=1
    avance(obstacles,Rayons,x0,y0,alpha,0,Centre,Rlim,color)



#n_rebonds_svp(obstacles,Noms,Rayons,6,Centre,Rlim)


###Rajoutons des obstacles


o_1=(-0.5,3**0.5/2)
o_2=(0.5,3**0.5/2)
o_3=(-1,0)
o_4=(0,0)
o_5=(1,0)
o_6=(-0.5,-3**0.5/2)
o_7=(0.5,-3**0.5/2)

Noms2=[str(i) for i in range(1,8)]
obstacles2=[o_1,o_2,o_3,o_4,o_5,o_6,o_7]
Centre2=(0,0)
Rlim2=1.5
Rayons2=[0.25 for  i in range(7)]
Rayons3=[0.3,0.35,0.3,0.4,0.2,0.25,0.15]




#n_rebonds_svp(obstacles2,Noms2,Rayons3,23,Centre2,Rlim2)


#Le point de depart est en orange
#Le point de sortie de la trajectoire est l'extremite non marquee

#rebond_deterministe(-0.14771195289118022,0.41724651296907145,
↪  0.47435283021459895,obstacles,Rayons,Noms,Centre,Rlim)
```

```
342 #rebond_deterministe(-0.14771195289118022,0.41724651296907145,0.48,
    ↪  obstacles,Rayons,Noms,Centre,Rlim)
343 #Pour lancer un point (x0,y0) avec un angle alpha donne
344
345 print("Voici une trajectoire d'au moins 8 rebonds : " )
346 n_rebonds_svp(obstacles,Noms,Rayons,8,Centre,Rlim,'red')
347 plt.figure()
348 plt.show()
349
350 ##fonction qui trace deux trajectoires l'une en bleue l'autre en rouge
351
352 def trace_rebonds_deterministe(a,b,c,d,e,f,obstacles,Rayons,Noms,Centre,
    ↪  Rlim):
353     rebond_deterministe(a,b,c,obstacles,Rayons,Noms,Centre,Rlim,"red")
354     print("1   :", compte_rebonds(a,b,c,obstacles,Rayons))
355     rebond_deterministe(d,e,f,obstacles,Rayons,Noms,Centre,Rlim,"blue")
356     print("2   :", compte_rebonds(d,e,f,obstacles,Rayons))
357     plt.show()
358
359
360
361
362 print("Voici deux trajectoires proches l'une de l'autre :  ")
363
364 trace_rebonds_deterministe(0.667482841472288,  0.10452278926045411,
    ↪  0.10098578721700369,0.667482841472288,  0.10452278926045411,
    ↪  0.10098578721700369+10**(-5),obstacles,Rayons,Noms,Centre,Rlim)
365 plt.figure()
```

Références

[BL85] Philippe Bougerol and Jean Lacroix. *Products of random matrices with applications to Schrödinger operators*, volume 8 of *Progress in Probability and Statistics*. Birkhäuser Boston, Inc., Boston, MA, 1985.

[Che11] Chevallier. Dcalages et systmes dynamiques associs aux suites. 2011.

[CM06] Nikolai Chernov and Roberto Markarian. *Chaotic billiards*, volume 127 of *Mathematical Surveys and Monographs*. American Mathematical Society, Providence, RI, 2006.

[GR89] Pierre Gaspard and Stuart A. Rice. Scattering from a classically chaotic repellor. *The Journal of Chemical Physics*, 90(4):2225–2241, 1989.

[Rob95] Clark Robinson. *Dynamical systems. Stability, symbolic dynamics, and chaos*. Boca Raton, FL: CRC Press, 1995.

[Wil17] Amie Wilkinson. What are Lyapunov exponents, and why are they interesting? *Bull. Am. Math. Soc., New Ser.*, 54(1):79–105, 2017.