dice battle

un projet de programmation linéaire

Lisa Lam · ******* **Léane Salais** · ********

Un jeu de dés sans mise oppose deux joueurs. L'un gagne, l'autre perd ; le jeu est à somme nulle.
De ce jeu, deux saveurs seront ici proposées.
Dans une version séquentielle, le joueur 1 commence. Avec l'aide du hasard, il s'agit d'atteindre **N** points en un nombre non borné de coups. Ayant **D** fixé, chaque joueur lance entre 1 et **D** dés ; il espère qu'aucun d'entre eux ne tombe sur 1, sans quoi il ne marque qu'un point ; s'il a plus de chance, il marque la somme des **D** dés. Les tours se suivent jusqu'à ce que la somme des tirages d'un joueur atteigne **N**. A chaque tour, les scores actuels sont affichés.
Dans une variante simultanée, les deux joueurs n'ont droit qu'à un seul coup (on étudiera plus loin le cas général) et doivent immédiatement maximiser leur bénéfice. En gardant la même forme, celle d'un jeu à somme nulle, il suffit maintenant de faire mieux que son adversaire pour gagner. La simultanéité implique que l'information sur le score de l'adversaire n'est plus disponible au départ d'un tour.

Qu'apporte la programmation linéaire dans la résolution d'une telle situation ?

1 modélisation

Calculer la probabilité **P(d, k)** d'obtenir **k** points en lançant **d** dés requiert de séparer deux états : (1) **k** est la somme du tirage, (2) **k** vaut 1. La première phase du calcul se fait sous la condition qu'aucun des **d** dés n'est tombé sur 1, ce qu'on représente par une probabilité **Q(d, k)** : on s'y restreint donc à l'état (1).
Pour **Q(d, k)**, le premier cas I est donc celui où le premier dé obtient 2, le second, II, celui où ce même dé obtient 3, et ainsi de suite. Chacun des cas I à V est décomposé en sous-cas de la même façon : I contient le cas I' où le deuxième dé obtient 2, puis le cas II' où le deuxième dé obtient 3, etc. La décomposition se poursuit jusqu'à ce que tous les dés aient varié, ce qui se traduit par la relation de récurrence suivante :

$$\boldsymbol{Q(d,k)} \quad = \quad \sum_{\boldsymbol{j=2}}^{\boldsymbol{6}} \boldsymbol{Q(d-1,k-j)} * \boldsymbol{\pi(j)} \quad = \quad \sum_{\boldsymbol{j=2}}^{\boldsymbol{6}} \frac{\boldsymbol{Q(d-1,k-j)}}{\boldsymbol{5}}$$

où **π(j)** est la probabilité d'obtenir un nombre de points **j** au dé variable, à savoir $\frac{1}{5}$ s'il y a cinq possibilités.
L'initialisation correspond aux cas où **d** vaut 1, où le calcul est immédiat :
Q(1, j) = $\frac{1}{5}$ avec **j** entre 2 et 6, **Q(1, j) = 0** si **j** vaut 1 (tirage impossible dans les conditions de définition de **Q**).

Expliqué dans l'énoncé, le passage à **P(d, k)** – qui va maintenant inclure les tirages évalués à 1 – implique de lever la restriction et de se replacer dans l'univers original. On doit pondérer la probabilité calculée en **Q(d, k)** par les chances d'être effectivement dans l'état (1). La formule générale est la suivante :

$$\boldsymbol{P(d,k)} = \boldsymbol{p}(\textit{obtenir autre chose que } 1 \textit{ à chacun des dés } \wedge \textit{ obtenir k points auxdits dés}) = \left(\frac{\boldsymbol{5}}{\boldsymbol{6}}\right)^{\boldsymbol{d}} \boldsymbol{Q(d,k)}$$

D'un autre côté, pour tout **d**, **P(d, 1) = 1** $-\left(\frac{5}{6}\right)^{d}$; c'est la probabilité d'être dans l'état (2).
Il est impossible d'obtenir un score compris entre **2** et **2d – 1**, et pour tous ces cas **P(d, k) = 0** – de fait, on ne peut gagner $\mathbf{k} \neq \mathbf{1}$ que si tous les dés génèrent des valeurs supérieures à 2 (ce qui fait un score ajouté de **2d** minimum) ; il est évidemment aussi impossible d'obtenir plus de **6d** points à un lancer (**d** dés à six faces).

»»»» code : classe mère Strategy, fonctions Q(d,k), P(d,k) et allP(d,k)

variante séquentielle

Les deux adversaires se voient et jouent à tour de rôle. Le joueur étiqueté « 1 » commence, et chacun choisit sa stratégie comme bon lui semble parmi celles que nous allons définir.

2

stratégie aveugle

La stratégie aveugle est une réponse saine dans le seul cas où l'on ignore la situation de l'adversaire et la valeur de l'objectif **N.** Sans horizon fixé, il s'agit de maximiser le nombre de points potentiellement obtenus à chaque lancer, en équilibrant le risque d'obtenir 1 et les chances de marquer un score conséquent. Il suffit de chercher

$$\boldsymbol{argmax_d \;\; EP(d) = 4d\left(\frac{5}{6}\right)^d + 1 - \left(\frac{5}{6}\right)^d}$$

Il semble, selon l'exécution du code, que la stratégie la moins absurde dans ce cas soit de jouer six dés si la borne **D** le permet, et un maximum **D** de dés si **D** est inférieur à 6.

»»»»» *code : classe Sequential, fonctions scoremax() et blind_strategy(i,j)*

3

Malheureusement, la stratégie aveugle ne donne pas de très bons résultats sur la durée, et peut même faire perdre celui qui s'y fie dans des cas limite assez courants. Soit **N** = 10. Posons qu'il soit permis de jouer **D** = 4 dés. Si le premier joueur – dont c'est le tour – a 8 points, et l'autre 9, il serait rationnel pour lui de jouer seulement un ou deux dés : s'il n'obtient pas immédiatement 2 points, il est certain de perdre. Or, lancer les quatre dés comme le préconise la stratégie aveugle maximise le risque d'obtenir 1, et donc de perdre.
Avec quatre dés, le joueur 1 aura une probabilité de $\left(\frac{5}{6}\right)^4 = 0.48$ d'obtenir au moins 2,
soit seulement la moitié de celle qu'il a s'il ne lançait qu'un seul dé (0.83).

On décide donc de s'orienter vers une stratégie dite « optimale », qui prend à la fois en compte la distance à l'objectif et les décisions possibles de l'adversaire.

4

programmation dynamique

Dans un jeu séquentiel, il est sage de supposer que l'adversaire est intelligent. Il fera tout pour gagner au prochain coup, en fonction de la situation (de score) dans laquelle il est mis. La détermination de la stratégie optimale va donc nécessairement passer par un calcul d'espérance : quelles sont mes chances de gain, celles de l'adversaire ? Même s'il s'agit d'un jeu de hasard, à quoi puis-je m'attendre sur un grand nombre de tirages, et comment tirer profit de cette information ?

Dans un jeu à somme nulle, un joueur gagne ce que l'autre perd : les espérances de gain sont opposées. Mon espérance de gain **EG** dépend donc des espérances de gain de l'adversaire pour chaque valeur de score réalisable avec mon choix de dés. Ce choix étant censé être optimal, cela signifie qu'il n'y a qu'un seul **d = d*** retenu pour définir **EG** : on va pouvoir supposer, dans la formule, que c'est un nombre fixé.
Si je joue ce nombre **d*** de dés, l'autre aura telle chance de gagner par la suite si le hasard génère tel score **k**, et encore telle chance de gagner pour tel autre **k**, etc. L'estimation va donc prendre en compte toutes les possibilités de score avec un poids qui évoque leurs chances de survenir (leur **P(d*, k)**). Et comme la somme de tout cela équivaut à mon risque total de perdre, trouver mon **EG** revient à prendre l'opposé de cette grandeur. Plus lisiblement, pour un état de score **(i, j)** donné et à supposer que les deux joueurs jouent optimalement, on a :

$$\boldsymbol{EG(i,j) = \sum_{k=1}^{6d*} P(d*,k) * -EG(j, i+k) \equiv \sum_{k=1}^{6d*} P(d*,k) * EG(i+k, j)}$$

Les cas de base sont ceux où l'un ou l'autre des deux scores (**i** ou **j**) atteint ou dépasse l'objectif **N**.
Pour **i ≥ N**, **∀ j**, **EG(i, j) = 1**. Le joueur 1 a tout simplement gagné.
Pour **j ≥ N**, **∀ i ∈ [0,N[** , **EG(i, j) = -1**. Le second joueur a atteint **N** points avant le premier.

En quoi consiste alors la stratégie optimale ? La tactique ne dépend que d'un seul paramètre, le choix **d** du nombre de dés. Le reste est une question de chance. Pour jouer intelligemment, il faudra juste déterminer le nombre **d*** de dés à lancer qui maximise l'espérance de gain précédemment calculée :

$$\boldsymbol{d^* = \max_{d\in[1,D]} \sum_{k=1}^{6d} P(d,k) * -EG(j, i+k)}$$

Du point de vue du programmeur, si les **EG(i, j)** sont calculées et stockées pour tous les **d** possibles, le calcul de **d*** revient à chercher l'indice **d** qui maximise cette liste avec la bibliothèque *numpy*.

»»»» code : classe Sequential, fonctions eg(i,j), allEG() et optimal_strategy(i,j)

La définition des règles du jeu sans possibilité de score nul nous facilite la vie. Dans la formule précédente, la certitude de toujours augmenter le score **i** d'une certaine quantité permet à la récursion d'avancer. De fait, nous avons défini l'initialisation pour les valeurs maximales de **i** et de **j** (près de **N**). Le calcul « descend » donc de ces **i** et **j** maximaux vers des **i** et **j** de plus en plus petits : et on compte sur le fait qu'à chaque appel récursif, les valeurs d'**EG(j, i')** sont déjà calculées pour les **i' = i** augmenté de **k**, $\forall\ \mathbf{k} \in [1, 6\mathbf{d}]$.
Mais la récursion risque de stagner si la quantité ajoutée **k** a une chance de valoir 0 : l'espérance vaut alors

$$\boldsymbol{EGN(i,j) = EG(i,j) + [\ P(d,0) * -EGN(j,i)\ ]}$$

Or par définition du jeu à somme nulle, **EGN(j, i) = −EGN(i, j)**, et c'est cette propriété qui permettait jusqu'alors de définir **EG(j, i)**… en allant chercher l'opposé d'**EG(i, j)** ; mais dans ces nouvelles circonstances, **EG(i, j)** n'existe pas encore. Il est donc impossible d'obtenir l'une ou l'autre de ces deux valeurs, puisqu'elles sont interdépendantes. Même si la formule est toujours vraie, il faudrait changer de paradigme de codage.

Heureusement, nous n'avons pas à traiter un cas pareil. La formule récursive donnée plus haut suffit.
Le stockage des valeurs d'**EG** dans une matrice à chaque étape du calcul récursif permet un accès en **O(1)** aux valeurs déjà traitées. On parle de mémoïsation. C'est le principe de la programmation dynamique, qui permet d'optimiser le temps de calcul, autrement conséquent dans un tel contexte : à partir d'un score donné, modéliser tous les chemins possibles jusqu'à la victoire de l'un des joueurs aurait pris longtemps.

mise en œuvre

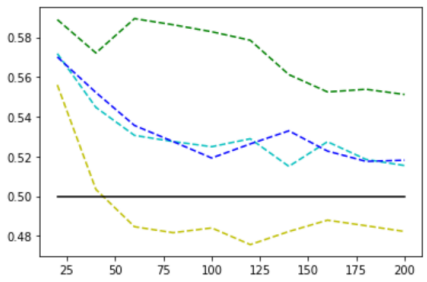
La décision majeure de la programmation a été d'utiliser les fonctionnalités objet de Python. En regroupant les stratégies dans des classes (une pour partie séquentielle, une pour partie simultanée), on permet à chaque nouveau joueur de « charger » le socle de décision qui convient à chaque partie dès sa création.
Lancer une partie séquentielle requiert d'avoir défini les conditions **D** et **N**. On peut ensuite créer un objet **Sequential(D, N)** qui sert de base de décision. Une fois cette base initialisée (calcul des variables de classe : les **P(d, k)**, la matrice complète des **EG**, la matrice complète des **d***), elle permet au joueur de choisir l'action à entreprendre selon l'*état mental* qu'il choisit : veut-il jouer à l'aveugle, et donc maximiser son score immédiat ? jouer optimalement, et donc maximiser son espérance de gain ? Ces états mentaux sont représentés par des méthodes de classe qu'il est libre d'utiliser dans chaque match. Posons qu'il ait choisi la stratégie optimale : chaque fois qu'il souhaite prendre une décision, la réponse lui vient sans effort en allant piocher dans la matrice **d*** - qui a été créée pour parer à toute éventualité de score.
Les deux adversaires se servent du socle de la même façon. Quelle que soit leur identité *réelle*, ils adoptent logiquement un point de vue de *premier joueur* (celui dont c'est maintenant le tour) à chaque phase de décision.

»»»» code : lancement de plusieurs échantillons dans la partie « Exemples et interactions »

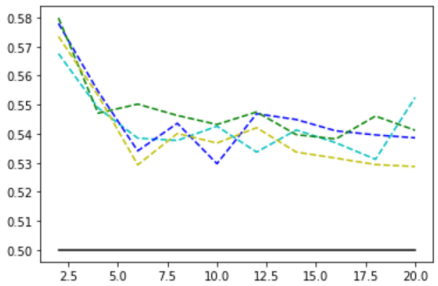
8

S'assurer de la stabilité et de l'intérêt des stratégies développées passe par un test statistique. On cherche à estimer les chances de gagner selon le choix du plan suivi. On s'affranchit du hasard en multipliant les parties jouées (loi des grands nombres), et on s'assure de ne pas être resté sur un cas de niche en variant les paramètres. L'évaluation se fait dans un tournoi en quatre temps : la stratégie aveugle rencontre la stratégie optimale, et inversement ; des matches sont aussi organisés entre les stratégies de même type. On considère alors que l'espérance de gain $G_1(a,b)$ pour la stratégie **a** contre la stratégie **b** est estimée par l'issue moyenne des matches (**1** si a gagne, **-1** sinon), sur la décision d'en jouer dix mille par type de rencontre.
Pour juger de l'efficacité d'une stratégie face à une autre, il suffira de regarder le rapport entre la ligne noire (seuil des 50% de chances de gagner) et la courbe associée. De combien creuse-t-on l'écart ?

Variation de l'objectif N



Variation du nombre de dés D



optimale × optimale
aveugle × aveugle
optimale × aveugle
aveugle × optimale

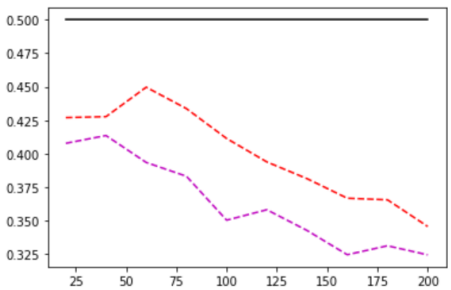
Il semble que la multiplication des **D** ait un effet chaotique. Un avantage est donné au premier joueur sans distinction d'esprit tactique. On se concentre donc sur le premier graphique pour l'analyse.
Lorsque les deux joueurs choisissent des stratégies similaires, on observe systématiquement un biais en faveur du premier joueur (≈ +4% d'espérance de gain) : il s'agit d'une course de vitesse, or il ouvre le jeu avec une chance d'augmenter son score avant son adversaire. Cet avantage s'estompe quand **N** grandit : cette chance supplémentaire devient peu significative comparée au nombre de tours nécessaire pour gagner.
La stratégie optimale accentue ce biais (≈+7% d'espérance de gain) lorsqu'elle joue en premier contre la stratégie aveugle, bien qu'elle perde aussi naturellement du terrain sur de grands **N**.
Par contre, si le joueur aveugle ouvre la partie, c'est plus compliqué : sur de petits **N**, l'avantage naturel au premier joueur fait effet sans qu'elle ne puisse rien contre lui. De fait, l'optimisation suppose qu'il joue intelligemment, et ne permet pas de s'imaginer qu'il lancerait tant de dés aussi vite ; il prend donc de l'avance et gagne. Cela dit, pour de plus grands **N**, la stratégie optimale a le temps de moduler son raisonnement en fonction de la distance à l'horizon, ce dont l'autre n'a absolument pas conscience. Il commet des erreurs, en plus de voir s'atténuer l'avantage de l'ouverture, et la stratégie optimale reprend sa couronne.

»»»»» *code : tests déroulés et graphiques affichés dans la partie « Tests statistiques »*

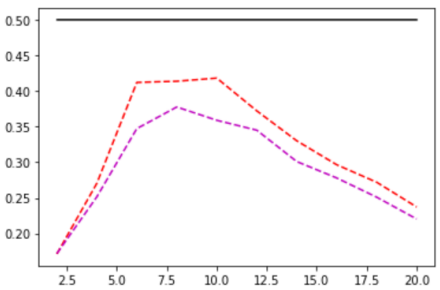
9

Peu inspirées par des modes de jeu aléatoires (pourquoi penser aussi mal quand une stratégie optimale est disponible ?), nous n'avons testé que la stratégie la plus stupide qui soit, celle de tirer un nombre de dés au hasard avant de jouer. Sans surprise, elle est largement dominée par la stratégie optimale, et même par la stratégie aveugle, quoiqu'on lui donne l'avantage de jouer en premier. En gardant le même procédé de test :

Variation de l'objectif **N**



Variation du nombre de dés **D**



aléatoire × optimale
aléatoire × aveugle

variante simultanée

La condition de simultanéité revient à supprimer l'information donnée au second joueur sur le score du premier au temps **t** (surprise !). Ils jouent en même temps et en partant du même point. Ils ont la même distance à parcourir pour arriver à l'objectif **N** : qui ira le plus loin ?
Sans relation d'ordre, le biais en faveur du premier joueur est évidemment supprimé. En supposant qu'ils optimisent leurs choix de la même façon, les deux sont exactement égaux face au hasard.

10

jeu en un coup

Lorsque les actions $\mathbf{d_1}$ et $\mathbf{d_2}$ sont fixées, un joueur gagne (gagne 1) dans les cas où son score dépasse celui de l'adversaire sans que l'autre ne dépasse le sien. On doit donc sommer les chances que cela arrive pour tout score (**k**,**l**) potentiel, quand [**k** varie jusqu'à **6*d**$_1$] et [**l** reste inférieur]. En termes probabilistes, l'intersection des événements notés entre crochets se traduit par une multiplication, les probabilités requises étant récupérées dans la matrice des **P(d, k)** définie plus haut.
Un joueur perd (gagne -1) dans le cas contraire.
Le fait de gagner ou de perdre constitue une partition de l'univers du jeu, on considère donc l'espérance de gain de chacun comme la probabilité de l'union disjointe de deux événements, d'où la somme :

$$\boldsymbol{EG_1(d_1, d_2) = 1 * \sum_{k=1}^{6d_1} \sum_{l=1}^{k-1} P(d_1, k) * P(d_2, l) + (-1) * \sum_{k=1}^{6d_2} \sum_{l=1}^{k-1} P(d_2, k) * P(d_1, l)}$$

Le calcul des EG$_1$ a été exécuté pour **D** = 3, ce qui donne une matrice pour tous les couples **(d$_1$,d$_2$)** envisageables :

$$\boldsymbol{EG_1 = \begin{matrix} 0.000 & -0.375 & -0.227 \\ 0.375 & 0.000 & -0.199 \\ 0.227 & 0.199 & 0.000 \end{matrix}}$$

On demande que les espérances de gain des deux joueurs soient égales (et donc nécessairement nulles, à mi-chemin entre -1 et 1) lorsque leurs scores sont égaux. De fait, lorsqu'ils partent du même point, ils sont soumis au hasard de la même façon. C'est vérifié.
L'existence ici tue d'un résidu d'ordre 10^{-17} sur la diagonale ne pose pas problème, il s'agit simplement d'une accumulation de petites erreurs due à des approximations préalables. Il est en effet avéré que les lignes de la matrice des **P(d, k)** ne somment pas exactement toutes à 1.

»»»»» *code : classe Simultaneous, fonctions eg1(d1,d2), gain_matrix(). tests dans les « Annexes calculatoires »*

11

Le meilleur coup en chaque situation dépend d'absolument toutes les possibilités d'action de l'adversaire. C'est la *réciprocité* de ce constat, qu'on n'avait pas dans la variante séquentielle (où quelqu'un avait déjà joué et fixé son sort), qui rend le raisonnement bien plus complexe et surtout non déterministe. On dit que la stratégie à adopter est mixte : un joueur fait face à la pénurie d'informations en appliquant une loi de probabilité arbitraire à tous les choix tactiques possibles. Il ne peut plus prendre une décision unique et universellement valable étant donné un état de score. Pour faire court, le joueur 1 (resp. 2) développe un vecteur de *probabilités* **p** (resp. **q**) de jouer *chaque* nombre **d** de dés possible.

Si le joueur 1 rendait sa stratégie mixte **p** publique, la stratégie du joueur 2 consisterait à maximiser son espérance de gain sous ces conditions. Pour définir **q**, il doit résoudre le programme linéaire suivant :

$$\begin{array}{l} \boldsymbol{\max\limits_{q} \; q^t(-EG_1)\, p} \\ \left| \begin{array}{l} \boldsymbol{\sum_{d=1}^{D} q(d) = 1} \\ \boldsymbol{q(d) \geq 0} \end{array} \right. \end{array}$$

où **-EG**$_1$ représente l'espérance du joueur 2, l'opposé d'**EG**$_1$ (la maximiser revient à minimiser EG$_1$).
Remarque : **p(d)** et **q(d)** sont notés **p1(d)** et **p2(d)** dans le sujet. Le changement de nom facilite un peu la lecture.

»»»»» *code : hors classe (non requis), voir les addenda*

12

On file le raisonnement en ajoutant une dimension supplémentaire : le joueur 1 n'a pas rendu sa stratégie optimale publique, et lui-même, pour la calculer, se fonde sur ce qu'il croit savoir de celle du joueur 2, lequel n'a lui-même rien dit… Sans la programmation linéaire, résoudre le problème serait impossible, car l'information est pour ainsi dire inexistante.

La logique n'a pas changé : le joueur 1 doit choisir un **p** qui maximise ses chances de gain, là où le joueur 2 essaie de lui tendre des pièges, et donc de choisir un **q** qui tend à les minimiser. On maximise ce minimum :

$$\max_{p} \min_{q} p^t E G_1 q$$
$$\sum_{d=1}^{D} p(d) = 1, p(d) \geq 0$$
$$\sum_{d=1}^{D} q(d) = 1, q(d) \geq 0$$

L'équilibre ne peut être atteint que pour un match nul : c'est ce que les deux espèrent.

13

Ce programme linéaire est codé et résolu sous PULP, le solveur officiel Gurobi étant trop capricieux sur nos machines personnelles. Il faut définir les variables qui composent les vecteurs **q** et **p**, passer la matrice EG_1 en entrée, définir les contraintes (on traite des vecteurs de probabilités, donc toutes leurs valeurs sont positives et inférieures à 1). PULP renvoie les vecteurs suivants pour plusieurs valeurs de D :

D = 3 : **[0.000 0.000 1.000]**
D = 6, 10, 20… : **[0.000 0.176 0.053 0.000 0.770 0.000 0.000 0.000 0.000 0…]**

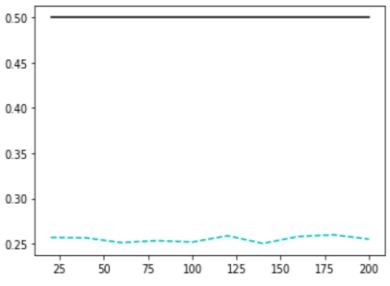
Les stratégies pures dont les p_i valent 0 sont dites *strictement dominées* : sur le deuxième exemple, le joueur ne risque *jamais* de jouer 1 ou 4 dés. Il en joue souvent 5, et très rarement 2 ou 3. Par ailleurs, le fait que les vecteurs optimaux soient les mêmes pour **D** > 5 indique que les grands nombres de dés sont jugés inutiles par le solveur.

»»»» *code : classe Simultaneous, fonctions solve_pl(D), pl_strategy(i,j). tests dans les « Annexes calculatoires »*

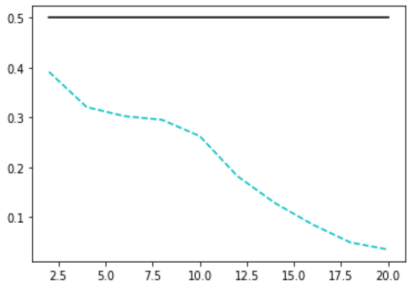
14

Pour jouer, le joueur 1 va tirer un **d** selon la distribution de probabilité **p** fraîchement calculée.
Si l'on avait voulu observer un match équitable, il aurait fallu résoudre le PL exprimé en 11 après avoir trouvé **p** – pour avoir le **q** associé : de fait, le choix de **p** suppose que **q** est aussi optimisé en fonction de ce choix, et ainsi de suite. En résolvant ce deuxième problème, la valeur de l'optimum aurait été la même (0 a-t-on dit, puisque les deux se battent pour ne pas perdre), mais la base associée aurait bien sûr été une stratégie mixte différente de **p** : **p** est une action, et **q** la réaction liée, qui en diffère.
Pour faire court, le joueur 1 ne s'attend pas à avoir devant soi un adversaire aveugle.
C'est pourtant ce qu'on lui propose d'affronter. La stratégie aveugle, déterministe, est toujours celle qui permet d'obtenir immédiatement un maximum de points – on maintient la version du mode séquentiel. La décision de celui qui l'adopte est donc fixée quelle que soit la situation (jouer six dés si **D** le permet).
Observons ci-dessous l'espérance de gain du joueur 1 (qui optimise) face à un joueur aveugle :

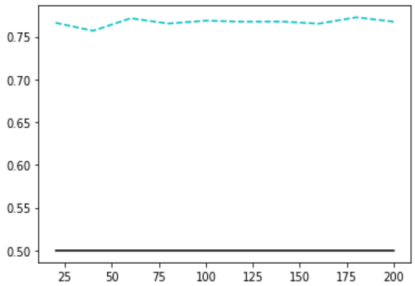
Variation de l'objectif N



Variation du nombre de dés D



! Correction des matches nuls



Le résultat est de prime abord très mauvais pour tout **N**. On constate aussi que l'optimiseur gère mal les grandes valeurs de **D**. Cela dit, c'est aussi parce que les matches nuls ne sont pas pris en compte comme des victoires de la stratégie optimale. Si on les considère comme tels, on obtient le graphe de droite, où l'espérance de gain a bondi à plus de 0.7. Cela nous permet d'estimer que les matches optimale-aveugle se concluent par environ 50% de matches nuls. Est-ce à dire que le but de l'optimisation est d'obtenir un nul ? On s'en convainc, bien qu'avec réticence, en se souvenant que c'est la valeur de l'espérance à l'optimum du PL ; dans ce cas, c'est réussi.

forme générale

15

Supposons maintenant que l'on doive jouer plusieurs tours d'affilée de ce jeu simultané. Pour déduire une stratégie optimale, il va de nouveau falloir initialiser une matrice des espérances en commençant par le coin inférieur droit. On considère les situations où le score de l'un des deux joueurs dépasse **N**, auquel cas il a gagné. S'il s'agit du joueur 1, puisqu'on calcule son espérance de gain personnelle, les cases **[N:, :N]** de la matrice (pour lequelles **i > N**) vaudront 1 ; si c'est le joueur 2, c'est que 1 a perdu : les cases **[:N, N:]** de la matrice (pour lesquelles **j > N**) vaudront donc l'opposé, -1. Si les deux dépassent **N**, on ne tranche en faveur de personne : match nul, ce qu'on traduit par une espérance de gain nulle en **[N:, N:].**

16

Cette question est l'occasion de tirer profit de nos premières erreurs dans la recherche d'EG, à la question 4, où nous n'avions pas pensé à prendre la séquentialité en compte. Découvrir à quel point la modalité séquentielle réduisait les incertitudes (en fixant et en publiant le score de l'adversaire au sortir d'un lancer) a immensément simplifié la complexité et la durée de nos calculs.
Comment peut-on calculer l'espérance de gain du joueur 1 lorsqu'il ignore à la fois son score et celui de son adversaire à l'issue du tour courant ? Il suffit de faire le calcul sur toutes les possibilités : on augmente les scores actuels de grandeurs **k** et **l** variables :

$$\boldsymbol{EG1_{d2}^{d1}(i,j)} = \sum_{k=1}^{6d_1}\sum_{l=1}^{6d_2} \boldsymbol{P(d_1,k) * P(d_2,l) * EG_1(i+k, j+l)}$$

17

La fameuse fonction est donc récupérée de nos premiers brouillons. On cherche le choix $\mathbf{d_1^*}$ qui maximise $\boldsymbol{EG1_{d2}^{d1*}(i,j)}$ pour toute possibilité de $\mathbf{d_2}$. On ne renvoie pas $\mathbf{d_2^*}$ en même temps : le joueur 2 s'en occupera en parallèle de son côté : lui cherchera son $\mathbf{d_2^*}$ qui maximise $\boldsymbol{EG1_{d2}^{d1*}(j,i)}$ pour toute possibilité de $\mathbf{d_1}$.
Au vu de toutes les pistes qu'elle a à explorer, la fonction en question est donc assez lente, mais elle est largement améliorée par mémoïsation.
Petite note : selon nos observations, si les deux sont au même score, ils agiront de la même façon. Il y a fort à parier d'ailleurs qu'ils commenceront par emprunter la stratégie aveugle, et par lancer six dés, avant que chacun voie où le hasard les mène et commence à moduler ses choix : prendre des risques, donc devenir plus agressif, si l'adversaire prend trop d'avance ; calmer le jeu et préférer s'assurer des points s'il est loin derrière, ou si l'objectif se rapproche. Le schéma global est vraiment le même que pour la variante séquentielle, à cela près que la diffusion de l'information est deux fois moins fréquente.

>>>>> *code : classe Simultaneous, fonctions egg(i,j), allEggs(), simoptimal_strategy(i,j), « Exemples et interactions »*