ÉNONCÉ

Nous essayons de trouver la meilleure stratégie pour le jeu de deux joueurs:
À chaque manche, ceux-ci donnent en même temps un nombre entre 1 et 10. Si le nombre de l'un est le suivant de l'autre, il prend cinq points à son adversaire. Sinon, il lui en donne un.

INTRODUCTION

Le principe de toute stratégie pour ce jeu est donc de donner un nombre suffisamment grand pour être immédiatement après le nombre de l'adversaire. Tout en restant suffisamment petit pour ne pas avoir un trop gros écart, ce qui ferait perdre la manche.

La solution en elle-même ne présente en réalité que peu d'intérêt étant donné que le jeu a été inventé pour l'exercice. Cependant, au-delà de la démarche et de la réflexion, il est intéressant de noter que tous les jeux ayant les mêmes caractéristiques, que nous définirons tout de suite, peuvent reprendre le même raisonnement. Le plus connu d'entre tous étant sans doute le « pierre-feuille-ciseaux ». Ainsi, de nombreux exemples seront faits avec ce jeu afin de les rendre plus appréhendables.

Puisque les manches sont indépendantes les unes des autres, que les deux joueurs révèlent leur coup au même moment, et qu'il n'y a pas de condition sur la sélection desdits coups, le jeu est appelé « symétrique ». Ce qui implique que les deux joueurs sont en théorie interchangeables et que la stratégie choisie par l'un peut être également choisie par l'autre.
De plus, le fait que les points d'un joueur soient obligatoirement pris à son adversaire permet de qualifier le jeu de « à somme nulle ». Impliquant que la meilleure stratégie que celui-ci puisse choisir est, du point de vue de l'adversaire, la pire et inversement. Puisque la victoire de l'un signifie nécessairement la défaite de l'autre.

Ces propriétés nous permettent ainsi de voir plus clair sur la nature de ce que nous cherchons puisque nous pourrions en déduire que la meilleure stratégie existe, est unique, ne perds jamais mais ne gagne pas forcément. Ceci peut se démontrer avec des théorèmes assez compliqués de dualité ou d'équilibre de Nash. Il est en revanche à portée de raisonner sur la dernière conclusion. En effet, la symétrie du jeu fait que l'adversaire, qui joue contre ladite stratégie peut aussi l'utiliser avec tout autant d'efficacité. Or, par le principe de la somme nulle, il ne peut pas y avoir deux gagnants, ni deux perdants résultant donc en une égalité. La stratégie optimale ne garantit donc pas la victoire du moment que l'adversaire la connaît. Il pourrait également y avoir une stratégie moins bonne de manière générale, mais qui soit suffisamment bien choisie pour contrer la stratégie optimale, et qui résulterait aussi en une égalité.

Une stratégie peut être représentée sous la forme d'un vecteur $x(x_1,x_2,...)$ composé d'une distribution de probabilités pour chaque coup.
Lorsque tout le poids de la distribution est sur un seul coup, à l'instar de (0,1,0), la stratégie est dite « pure ». Dans le cas contraire, par exemple ($^1/_3$, $^2/_3$, 0), elle est appelée « mixte ».

RECHERCHE NAÏVE

Nous pouvons commencer par estimer le résultat à l'aide d'une méthode de recherche que nous pouvons qualifier de naïve. Et dont le fonctionnement est le suivant:

- On part de stratégies perfectibles.
- On garde la meilleure d'entre elles en les faisant jouer des milliers de fois les unes contre les autres.
- On fait un nouvel ensemble de stratégies qui dérivent de la gagnante et on recommence.

Exemple avec l'affichage du programme pour les quelques premières générations :

```
Stratégies de départ: [[0, 0, 0, 0, 0, 0, 0, 0, 0, 1.0], [0, 0, 0, 0, 0, 0, 0, 0, 1.0, 0], [0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0, 0], [0, 0, 0, 1.0, 0, 0, 0, 0, 0, 0], [0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0], [0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0], [1.0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

En train de chercher une stratégie... Génération actuelle: 0
Meilleure stratégie actuelle: (0, 1.0, 0, 0, 0, 0, 0, 0, 0, 0)

En train de chercher une stratégie... Génération actuelle: 1
Meilleure stratégie actuelle: (0.0, 0.5, 0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)

En train de chercher une stratégie... Génération actuelle: 2
Meilleure stratégie actuelle: (0.0, 0.375, 0.375, 0.25, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)

En train de chercher une stratégie... Génération actuelle: 3
Meilleure stratégie actuelle: (0.0, 0.328125, 0.328125, 0.34375, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)
```

Les stratégies de départ sont dans ce cas toutes les stratégies pures possibles.

Afin de programmer un tel algorithme, nous opérons en deux parties. Une première pour générer les différentes stratégies. Et une seconde pour les tester au travers d'un tournoi.
En ce qui concerne la génération des stratégies, nous commençons par multiplier chaque valeur par un certain coefficient, $1-1/2^n$ dans le cas de l'exemple. Ce qui libère $1/2^n$, qui est redistribué dans un second temps aux différentes valeurs du tableau, en générant ainsi une dizaine de nouvelles stratégies à tester.
La simulation des stratégies quant à elles se fait assez naturellement avec un système de point accordé aux stratégies gagnantes ou égalisantes. La fonction « choices » de la bibliothèque « random » sous python, qui prend en paramètre une liste et renvoie un élément de celle-ci pris aléatoirement selon une seconde liste de poids pour chaque élément, permet de facilement simuler le choix en lui-même et de se concentrer sur l'interprétation de ce choix.

APPROCHE VIA LA PROGRAMMATION LINÉAIRE

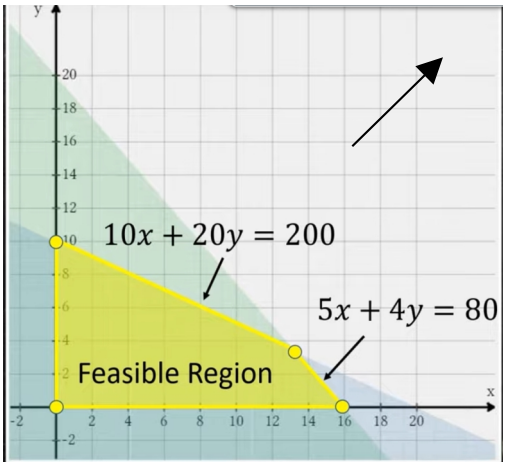
Une autre façon d'aborder la chose est celle de la programmation linéaire. Son utilité est de résoudre des problèmes de maximisation ou de minimisation de plusieurs variables en ayant des contraintes sur celles-ci.

Afin d'introduire ce concept, prenons le cas d'un menuisier cherchant une rentabilité maximale avec 200 planches de bois et 80h de main d'œuvre disponible pour fabriquer des tables et des armoires, les tables nécessitant 10 planches et 5h de travail pour 180€ là où les armoires demandent 20 planches et 4h de travail pour 200€

Ici, on cherche donc à maximiser la somme 180x + 200y

Les contraintes sont les aires définies par les droites sur le schéma. Et leur intersection marque donc la « zone de faisabilité », c'est-à-dire là où toutes les conditions sont satisfaites.

Tous les points de l'aire sont donc des candidats potentiels pour être la valeur maximale. Heureusement, nous pouvons visualiser assez facilement à l'aide de la méthode de résolution graphique « ligne ISO » que l'un des points d'intersection est obligatoirement une valeur optimale, et qu'il suffit donc de se concentrer sur eux.



Il existe plusieurs algorithmes de résolution de problèmes de programmation linéaire. Celui utilisé ici se nomme « simplex ». Sa méthode de résolution est la suivante :

- On part d'une solution potentielle. L'origine étant généralement celle par laquelle on commence.
- On se rapproche des points voisins en ajustant les variables pour veiller à rester dans la zone de faisabilité.
- Si on va dans le sens de l'objectif, représenté par la flèche noire, le point vers lequel on arrive devient la nouvelle solution optimale potentielle.
- Si au contraire, on s'éloigne de l'objectif, cela signifie que le point vers lequel on se dirige est une solution moins optimale.
- Et si peu importe le point vers lequel on se dirige, on s'éloigne de l'objectif, on a trouvé la solution optimale.

UTILISATION DE LA PROGRAMMATION LINÉAIRE

Pour comprendre comment utiliser ce principe pour nos fins, concentrons-nous sur le cas du pierre-feuille-ciseaux.

Le tableau des gains du joueur en fonction des coups de l'adversaire pour chaque manche est le suivant:

Adversaire Joueur	Pierre	Feuille	Ciseaux
Pierre	0	-1	+1
Feuille	+1	0	-1
Ciseaux	-1	+1	0

Les valeurs de ce tableau peuvent être représentés sous la forme d'une matrice M.
Soient $x(x_{pierre},x_{feuille},x_{ciseaux})$ et $y(y_{pierre},y_{feuille},y_{ciseaux})$ les stratégies mixtes du joueur et de son adversaire respectivement, l'espérance de gain du joueur en fonction de x et y est le produit xMy.

Démonstration :
Espérance pour le joueur = Σ[val(X)*P(X)] avec X évènement

$= y_{pierre}*x_{feuille} + y_{feuille}*x_{ciseaux} + y_{ciseaux}*x_{pierre} - y_{pierre}*x_{ciseaux} - y_{feuille}*x_{pierre} - y_{ciseaux}*x_{feuille}$

$= y_{pierre}(x_{feuille}-x_{ciseaux}) + y_{feuille}(x_{ciseaux}-x_{pierre}) + y_{ciseaux}(x_{pierre}-x_{feuille})$

$= xMy$

Puisque les gains du joueur signifient les pertes de l'adversaire, on veut l'espérance la plus haute possible tandis que l'adversaire la veut la plus basse possible.
En réutilisant le produit, on cherche donc les solutions de ce problème:

$Obj = \max_{x}(\min_{y}(xMy))$ Soit: Maximiser avec x tandis qu'on minimise avec y, xMy

Or, en tant que tel, il n'est pas possible d'en tirer une solution car le problème est quadratique. On essaye de maximiser et de minimiser à la fois le même calcul.

Évidemment, la stratégie du joueur ne doit pas reposer sur la chance. Ni même le secret puisqu'elle est censée contrer n'importe quelle stratégie. Y compris une judicieusement choisie pour exploiter ses faiblesses.
Ainsi, on cherche une stratégie qui même connue, minimise ses points faibles.

Pour comprendre comment dépasser cette difficulté, prenons le cas de la stratégie $x=(0, {}^{2}/_{3}, {}^{1}/_{3})$

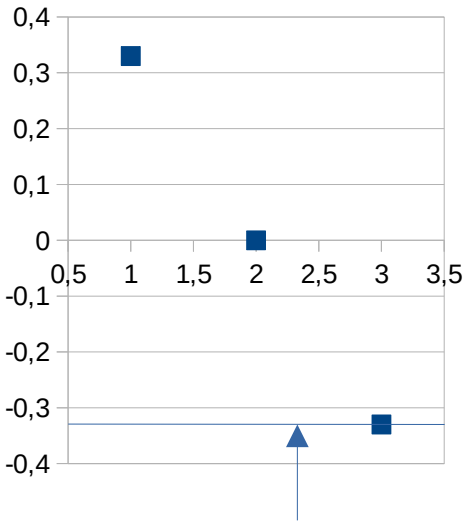
Aux yeux de l'adversaire, qui connaît notre stratégie, notre espérance pour chaque coup est le produit $xM = ({}^{1}/_{3}, 0, -{}^{1}/_{3})$.
Le coup qu'il devrait donc privilégier est là où la valeur est la plus basse. Soit « ciseaux » avec leur $-{}^{1}/_{3}$ d'espérance pour le joueur signifiant $+{}^{1}/_{3}$ pour l'adversaire. Ainsi, puisque cette espérance vaut pour chaque coup, il devrait adopter la stratégie y=(0,0,1) soit « toujours ciseaux » puisque c'est ici qu'il exploite le plus la faiblesse dans la stratégie du joueur.

De manière générale, si le joueur donne sa stratégie à l'avance, le meilleur contre pour l'adversaire sera toujours une stratégie pure exploitant la faiblesse dans celle du joueur, et il en obtiendra l'espérance associée. Ce qui veut dire que $Espérance_{adversaire} = minimum(xM)$
C'est pourquoi nous pouvons passer d'une variable y avec une infinité de possibilités à seulement « le nombre de stratégies pures » possibilités, soit les différents coups possibles dans le jeu.
Cela nous permet de réécrire le problème :

$$Obj = \max_{x} \left(\min_{y_{\text{stratégies pures}}} (xMy) \right)$$

Le problème est donc toujours le même. Cependant, avec un nombre fini de possibilités pour la variable y, nous pouvons trouver le moyen de la remplacer par une autre variable, z, qui sera le fruit d'une maximisation.

Le principe est d'augmenter z, représenté par la flèche bleue, jusqu'à atteindre la plus petite des valeurs du vecteur xM. Une fois cette valeur atteinte, nous avons trouvé le minimum des espérances du joueur, soit la stratégie optimale pour l'adversaire.



Nous pouvons ensuite utiliser ceci pour redéfinir l'objectif :

$$Obj = \max_{x} \left(\max_{z} (z \mid z < xMy \mid \text{pour tout y : stratégie pure}) \right)$$

$$= \max_{x,z} (z \mid z < xM(1,0,0),\ z < xM(0,1,0),\ z < xM(0,0,1))$$

Soit $Obj = \max(z)$
sur $x_{pierre}, x_{feuille}, x_{ciseaux}, z$
avec $z < xM(1,0,0),\ z < xM(0,1,0),\ z < xM(0,0,1)$

Maintenant, ceci est résoluble. Le même raisonnement peut s'appliquer au jeu de l'énoncé et, plus généralement, à tous les jeux symétriques à somme nulle du moment qu'on a la matrice jeu associée.

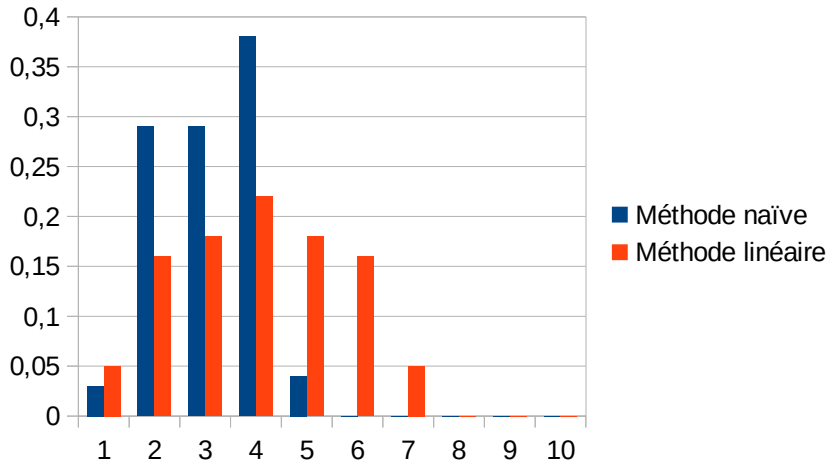
$Obj_{énoncé} = \max(z)$
sur $x_1, x_2, x_3, \ldots, z$
avec $z < xM(1,0,\ldots,0),\ z < xM(0,1,0,\ldots,0),\ \ldots,\ z < xM(0,\ldots,0,1)$

RESULTATS :

Avec l'approche naïve, le résultat est le suivant au bout de 11 générations :
La meilleure stratégie trouvée est: (0.0029230155050754547, 0.28907029841974896, 0.28907029841974896, 0.38013516953436116, 0.03831293687106552, 0.0, 0.0, 0.0, 0.00048828125, 0.0) avec 19 défaites. Sur 55

Avec l'algorithme linéaire :
x: array([5.38033395e-02, 1.57699443e-01, 1.76252319e-01, 2.24489796e-01, 1.76252319e-01, 1.57699443e-01, 5.38033395e-02, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00)

Entrées sur un graphique, nous pouvons visualiser plus facilement la façon dont les valeurs sont distribuées :



On observe ainsi une forme gaussienne sur les résultats donnés par la méthode linéaire, qui nous montre des valeurs plus étalées sur les différents coups, résultant en une forme plus harmonieuse. Aussi, pour les deux résultats, les coups entre 2 et 4 semblent être les plus intéressants.

ANNEXES :

Le code de la méthode passant par la programmation linéaire pour trouver la réponse au problème de l'énoncé :
Afin d'utiliser cette bibliothèque, il a fallu adapter les équations pour correspondre au format requis.

```
matJeu = [[0, -5, 1, 1, 1, 1, 1, 1, 1, 1], [5, 0, -5, 1, 1, 1, 1, 1, 1, 1], [-1, 5, 0, -5, 1, 1, 1, 1, 1, 1], [-1, -1, 5, 0, -5, 1, 1, 1, 1, 1], [-1, -1, -1, 5, 0, -5, 1, 1, 1, 1], [-1, -1, -1, -1, 5, 0, -5, 1, 1, 1], [-1, -1, -1, -1, -1, 5, 0, -5, 1, 1], [-1, -1, -1, -1, -1, -1, 5, 0, -5, 1], [-1, -1, -1, -1, -1, -1, -1, 5, 0, -5], [-1, -1, -1, -1, -1, -1, -1, -1, 5, 0]]

matNP = np.array(matJeu).transpose().tolist()
for i in range(len(matJeu)):    #Rajout des -z à la fin des équations
    matNP[i].append(-1)

matNP = np.array(matNP)

obj = [0] * nb_coups + [-1]     #L'objectif à maximiser, z (ici on minimise -z)
intervalles = [(0,1)]*nb_coups + [(-5, 5)]       # 0 <= [x1, x2, ...] <= 1 et -5 <= z <= 5

ineqG = np.negative(matNP)       #Les équations à optimiser
ineqD = [0] * nb_coups

eqG = [[1]*nb_coups + [0]]  #x1+x2+...+x10 (+0*z) = 1
eqD = [1]

res = linprog(c=obj, A_ub=ineqG, b_ub=ineqD,
    A_eq=eqG, b_eq=eqD, bounds=intervalles,
    method='simplex')
print(res)
```