Monopoly sous l'œil de Markov : Dé Rapide et probabilité d'atterrissages

ABISOUROUR BASSMA n°18359

Juillet 2024

Table des matières

1 Introduction

Ce document présente un commentaire détaillée du code Python utilisé pour effectuer l'analyse des probabilités des différentes cases dans le jeu de

Monopoly, en utilisant les lancers de dés classiques et du dé rapide.

2 Modules Importés

Les modules `numpy`, `scipy.linalg`, et `pandas` sont importés pour la manipulation des matrices, le calcul des vecteurs propres et l'exportation des données dans un fichier Excel.

```
# Importation du module numpy pour la manipulation des matrices
import numpy as np
# Importation du module scipy.linalg pour les op rations sur
    les matrices (le calcul des vecteurs propres)
import scipy.linalg as la
# Importation du module pandas pour la manipulation et l'
    exportation des donn es dans un fichier Excel
import pandas as pd
```

Listing 1 – Importation des modules

3 Liste des Cases

Les noms des cases du jeu de Monopoly sont stockés dans une liste.

```
nom_cases = [
    'Case D part', 'Belleville', 'Caisse de Communaut ', '
    Lecourbe', 'Imp ts',
    'Gare Montparnasse', 'Vaugirard', 'Carte Chance', '
    Courcelles', 'R publiques',
    'Simple Visite/Prison', 'La Villette', 'Cie    lectricit   ',
     'Neuilly',
    'Paradis', 'Gare de Lyon', 'Mozart', 'Caisse de Communaut
    ', 'Saint-Michel',
    'Pigalle', 'Parc Gratuit', 'Matignon', 'Carte Chance', '
    Malesherbes',
    'Henri-Martin', 'Gare du Nord', 'Saint-Honor ', 'La Bourse
    ', 'Cie des Eaux',
    'La Fayette', 'Allez en Prison', 'Breteuil', 'Foch', '
    Caisse de Communaut ',
    'Capucines', 'Gare Saint-Lazare', 'Carte Chance', 'Champs-
      lyses  ', 'Taxe de Luxe',
    'La Paix'
]
```

Listing 2 – Liste des noms des cases

4 Calcul des Probabilités des Lancers de Dés

Les fonctions suivantes permettent de calculer les probabilités des résultats possibles pour deux dés standards (une partie classique de Monopoly) et avec un dé rapide supplémentaire (une partie rapide) en tenant compte de la face "Mr. Monopoly".

```
def proba_des():
    """
    Calcule les probabilit s des r sultats possibles pour
    deux d s standards.
    :return: Liste des probabilit s.
    """
    return [x / 36 for x in [1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1]]
```

Listing 3 – Calcul des probabilités des dés (Partie Classique)

Dans cette analyse, nous avons choisi comme stratégie de jeu pour le dé rapide de toujours se déplacer selon deux dés blancs lorsqu'on obtient la face du "Bus".

FIGURE 1 – Faces "Mr. Monopoly" et "Bus" du dé rapide. Référence : eBay

```
def proba_des_rapide():
    """
    Calcule les probabilit s des r sultats possibles pour une
     partie rapide (uniquement les 3 faces num rot es (un,
    deux et trois) et la face "Bus" selon la strat gie fixe
    choisie pr ablement).
    :return: Liste des probabilit s non incompl te.
    """
```

```
    return [(x * (4 / 6)) / 144 for x in [1, 3, 6, 10, 14, 18,
    20, 20, 18, 14, 10, 6, 3, 1]]
```

Listing 4 – Calcul des probabilités des dés (Partie Rapide)

5 Création des Matrices de Transition

La fonction `creer_matrice` remplit une matrice colonne par colonne en utilisant une liste de probabilités.

```
def creer_matrice(proba_list, matrice, compteur_colonne, dim):
    """
    Fonction r cursive qui permet de remplir une matrice
    colonne par colonne.
    :param proba_list: Liste des probabilit s.
    :param matrice: Matrice de transition    remplir.
    :param compteur_colonne: L'indicateur r cursif.
    :param dim: Dimension de la matrice.
    """
    if compteur_colonne == dim:
        return matrice
    dernier_element = proba_list.pop()  # Retire le dernier
      lment   de la liste
    proba_list = [dernier_element] + proba_list  # Ajoute cet
      lment   au d but de la liste
    matrice[:, compteur_colonne] = proba_list  # Remplit la
    colonne actuelle de la matrice
    creer_matrice(proba_list, matrice, compteur_colonne + 1,
    dim)  # Appel r cursif pour la colonne suivante
```

Listing 5 – Création des matrices de transition

5.0.1 Construction de la Matrice d'une Partie Classique

Initialisation la construction de `matrice_des` , la matrice de transition d'une partie classique (lancer de deux dés blancs) de Monopoly.

```
# Construire matrice_des qui regroupe simplement les
    probabilit s de d placement selon les lancers de d s
    classiques
proba_ligne_case_depart = [0] * 40  # Initialise une ligne de
    probabilit s avec des z ros
proba_ligne_case_depart[2:13] = proba_des()  # Remplit la ligne
     avec les probabilit s des d s standards
matrice_des = np.zeros((40, 40))  # Cr e une matrice de z ros
     de 40x40
```

```
matrice_des[:, 0] = proba_ligne_case_depart  # Remplit la
    premi re colonne avec les probabilit s
creer_matrice(proba_ligne_case_depart, matrice_des, 1, 40)  #
    Remplit le reste de la matrice
```

Listing 6 – Initialisation de la matrice `matrice_des`

5.0.2 Construction de la Matrice d'une Partie Rapide

Initialisation la construction de `matrice_des_rapide` , la matrice de transition d'une partie rapide (lancer de deux dés blancs et du dé rapide).

```
# Construire matrice_des_rapide qui regroupe les probabilit s
    de d placement selon les lancers de d  s rapide
# y compris celui de Mr Monopoly
proba_ligne_case_depart_rapide = [0] * 40  # Initialise une
    ligne de probabilit s avec des z ros
proba_ligne_case_depart_rapide[2:16] = proba_des_rapide()  #
    Remplit la ligne avec les probabilit s des d  s rapides
matrice_des_rapide = np.zeros((40, 40))  # Cr e une matrice de
     z ros de 40x40
matrice_des_rapide[:, 0] = proba_ligne_case_depart_rapide  #
    Remplit la premi re colonne avec les probabilit s
creer_matrice(proba_ligne_case_depart_rapide,
    matrice_des_rapide, 1, 40)  # Remplit le reste de la
    matrice
```

Listing 7 – Initialisation de la matrice `matrice_des_rapide`

Incrémentons les probabilités de la face "Mr Monopoly" du Dé Rapide dans chaque colonne, qui permet au joueur d'avancer jusqu'à la prochaine propriété. [1]

```
# Impl menter Mr Monopoly
for i in range(40):
    matrice_des_rapide[monsieur_monopoly_value(i), i] += 2 / 6
     # Ajoute les probabilit s de Mr Monopoly

propriety = [1, 3, 5, 6, 8, 9, 11, 12, 13, 14, 15, 16, 18, 19,
    21, 23, 24, 25, 26, 27, 28, 29, 31, 32, 34, 35, 37, 39]

def monsieur_monopoly_value(pos):
    """
```

1. D'après le livret des règles du jeu Monopoly, le joueur bouge selon le score total des deux dés blancs et joue comme dans une partie classique, puis avance jusqu'à la prochaine propriété n'appartenant à aucun joueur. Dans le cas échéant, le joueur paye le loyer de la prochaine propriété. HASBRO/PARKER BROTHERS https ://www.hasbro.com/common/instruct/00009.pdf

```
    D termine la prochaine propri t  de Mr Monopoly en
    fonction de la position actuelle. Cette fonction sera
    employ  plus tard dans la matrice du lancer de d s d'une
    partie RAPIDE.
    :param pos: Position actuelle.
    :return: Prochaine position.
    """
    if (pos == 39):
        return 1
    else:
        for x in propriety:
            if x > pos:
                return x
```

Listing 8 – Implémentation des probabilités de "Mr Monopoly" du Dé Rapide

5.1 Construction de la Matrice des Règles du Jeu

5.1.1 Cartes Chance et Caisse de Communauté

Quand le joueur atteint la case « Chance » ou « Caisse de Communauté » Le joueur prend alors la première carte du paquet indiqué et, après s'être conformé aux indications qui y sont imprimées, il remet la carte, dos en dessus, sous le paquet. [2] On s'intéresse aux cartes qui modifient la position du joueur.

```
# Construire matrice_regles qui inclut les effets des cartes
    Chance et Caisse de Communaut , ainsi que la prison
matrice_regles = np.eye(40)  # Cr e une matrice identit  de
    40x40
chance_cases = [7, 22, 36]  # Les emplacements des cases chance
for x in chance_cases:
    matrice_regles[:, x] = [0] * 40  # Initialise la colonne de
     la case Chance avec des z ros
    matrice_regles[[0, 10, 11, 15, 24, 39, x - 3], x] = 1 / 16
     # Ajoute les probabilit s des cartes Chance
    matrice_regles[x, x] = 9 / 16  # Probabilit  de rester sur
     la case

caisse_cases = [2, 17, 33]  # Les emplacements des cases caisse
     de communaut
gares_caisse_cases = [5, 25, 35]  # Les emplacements des gares
    correspondantes
for i, x in enumerate(caisse_cases):
```

2. D'après le livret des règles du jeu Monopoly, HASBRO/PARKER BROTHERS https ://www.hasbro.com/common/instruct/00009.pdf

```
    matrice_regles[:, x] = [0] * 40  # Initialise la colonne de la case Caisse de Communaut  avec des z ros
    matrice_regles[[0, 1, 10, gares_caisse_cases[i]], x] += 1 / 16  # Ajoute les probabilit s des cartes Caisse de Communaut
    matrice_regles[x, x] += 12 / 16  # Probabilit  de rester sur la case
```

Listing 9 – Inititalisation de `matrice_regles`

5.1.2 Aller en prison

Le joueur est emprisonné lorsque son jeton atterit sur la case « Aller en Prison »

```
matrice_regles[10, 30] = 1  # Aller en prison
matrice_regles[30, 30] = 0  # Sortir de prison
```

Listing 10 – Implémenter la règle de la case « Aller en Prison »

6 Vérification de la Matrice de Transition

La fonction suivante vérifie si chaque ligne d'une matrice de transition a une somme égale à 1, en tenant compte d'une marge d'erreur.

```
def est_matrice_de_transition(matrice, erreur=1e-6):
    """
    V rifie si chaque ligne d'une matrice de transition a une somme  gale  1,
    en tenant compte d'une marge d'erreur.
    :param matrice: La matrice  v rifier.
    :param erreur: La marge d'erreur acceptable.
    :return: True si chaque ligne a une somme  gale  1, False sinon.
    """
    sommes_ligne = np.sum(matrice, axis=0)  # Calcule la somme de chaque colonne
    return np.all(np.abs(sommes_ligne - 1) < erreur)  # V rifie si les sommes sont  gales  1 dans la marge d'erreur
```

Listing 11 – Vérification de la matrice de transition

7 Calcul de la Distribution Stationnaire des Probabilités

La stratégie du joueur de cette modélisation markovienne consiste à quitter la prison immédiatement après son prochain tour (Leave Jail Rule-LJ) [3] Ci-dessous les fonctions qui affiche les valeurs des probabilités long terme du jeu Monopoly selon les versions du jeu, dans un tableur Excel.

7.1 Partie Classique

```
def lj_des_classique():
    """
    Calcule et enregistre les probabilit s des cases pour des
    lancers de d  s  classiques.
    """
    matrice_finale = np.matmul(matrice_des, matrice_regles)  #
    Calcule la matrice finale en multipliant matrice_des par
    matrice_regles
    vecteur_propre = la.eig(matrice_finale)[1]  # Calcule les
    vecteurs propres de la matrice finale
    vecteur_etat_stable = vecteur_propre[:, 0]  # S lectionne
    le premier vecteur propre
    vecteur_etat_stable = vecteur_etat_stable / sum(
    vecteur_etat_stable)  # Normalise le vecteur propre
    vecteur_etat_stable = vecteur_etat_stable.tolist()  #
    Convertit le vecteur en liste
    vecteur_etat_stable = [float(valeur.real) for valeur in
    vecteur_etat_stable]  # Convertit les valeurs en r els
    flottants
    afficher_valeurs(vecteur_etat_stable, "lj_des_classique.
    xlsx")  # Affiche les valeurs dans un fichier Excel
```

Listing 12 – Calcul et exportation des résultats d'une partie Classique

7.2 Partie Rapide

```
def lj_des_rapide():
    """
    Calcule et enregistre les probabilit s des cases pour des
    lancers de d  s  rapides.
    """
```

3. On peut choisir d'autres stratégies de jeu concernant la règle de la prison. ROBERT B. ASH AND RICHARD L. BISHOP : Monopoly as a Markov Process : Mathematics Magazine, Vol. 45, No. 1 (1972), pp. 26-29

```
    matrice_finale = np.matmul(matrice_des_rapide, matrice_regles)  # Calcule la matrice finale en multipliant matrice_des_rapide par matrice_regles
    vecteur_propre = la.eig(matrice_finale)[1]  # Calcule les vecteurs propres de la matrice finale
    vecteur_etat_stable = vecteur_propre[:, 0]  # S lectionne le premier vecteur propre
    vecteur_etat_stable = vecteur_etat_stable / sum(vecteur_etat_stable)  # Normalise le vecteur propre
    vecteur_etat_stable = vecteur_etat_stable.tolist()  # Convertit le vecteur en liste
    vecteur_etat_stable = [float(valeur.real) for valeur in vecteur_etat_stable]  # Convertit les valeurs en r els flottants
    afficher_valeurs(vecteur_etat_stable, "lj_des_rapide.xlsx")  # Affiche les valeurs dans un fichier Excel
```

Listing 13 – Calcul et exportation des résultats d'une partie Rapide

8 Affichage et Exportation des Résultats

Les fonctions suivantes affichent les matrices de manière lisible et enregistrent les probabilités de chaque case dans un fichier Excel.

```
def afficher_joliment(matrix):
    """
    Affiche une matrice de mani re lisible.
    :param matrix: La matrice    afficher.
    """
    np.set_printoptions(threshold=np.inf)  # Configure l'affichage complet de la matrice
    np.set_printoptions(formatter={'int': '{: 4d}'.format})  # Formate les entiers pour l'affichage
    print(np.array2string(matrix, separator=' '))  # Affiche la matrice

def afficher_joliment_sur_excel(array):
    """
    Enregistre une matrice dans un fichier Excel.
    :param array: La matrice    enregistrer.
    """
    df = pd.DataFrame(array)  # Convertit la matrice en DataFrame
    df.to_excel("test.xlsx", index=False)  # Sauvegarde le DataFrame dans un fichier Excel

# Les tests stochastiques
#print(est_matrice_de_transition(matrice_regles))
```

```
#print(est_matrice_de_transition(matrice_des))
#print(est_matrice_de_transition(matrice_des_rapide))

#afficher_joliment_sur_excel(np.matmul(matrice_des,
    matrice_regles))
#lj_des_classique()
#lj_des_rapide()
```

Listing 14 – Affichage et exportation des matrices de transition