Modélisation du capital d'un casino et calcul de la probabilité de ruine

Jordan Touil, Mathis Jouve, Antoine Rapold

14 janvier 2024

Table des matières

1 Introduction

La gestion du capital d'un casino est un défi complexe qui nécessite une compréhension approfondie des processus stochastiques et des lois probabilistes régissant les gains et pertes. Dans le cadre de notre étude, nous avons élaboré un modèle visant à représenter l'évolution du capital du casino en fonction du temps, en tenant compte de différentes lois de probabilité pour les gains des joueurs. Nous avons introduit le concept de temps d'arrêt N_t, déterminé par un processus de Poisson, pour modéliser le nombres de joueurs remportant des gains à l'instant t. Ces gains, représentés par des variables aléatoires $(X_i)_{i\geq 1}$, ont été envisagés selon différentes lois de probabilité. Pour illustrer l'impact de ces paramètres sur le processus Y_t, nous avons créé des graphiques mettant en évidence les variations du capital du casino.

Par la suite, nous avons simulé la probabilité de ruine du casino dans différents scénarios. Grâce à ces résultats nous avons pu tirer des conclusions quand à l'importance du choix du paramètre α. En utilisant ces observations, nous avons cherché à étendre notre modèle en explorant un nouveau cadre basé sur le jeu de la roulette. Cette extension vise à diversifier notre approche en intégrant

d'autres éléments caractéristiques des casinos, avec l'objectif final d'améliorer la gestion du capital dans un contexte plus réaliste.

Ainsi, notre exploration combine une modélisation rigoureuse, des simulations numériques et une réflexion sur les implications pratiques pour la gestion d'un casino. Nous vous invitons à suivre notre parcours à travers ces différentes étapes, qui offrent un aperçu approfondi des défis liés à la gestion financière dans l'industrie du jeu.

2 Modélisation du processus Y_t

2.1 Création du temps d'arrêt N_t

Les gains des joueurs ont lieux aux instants de saut d'un processus de Poisson $(N_t)_{t\geq 0}$ de paramètre 1. Plus précisément, considérons $(\xi_k)_{k\geq 1}$ une suite de variables aléatoires indépendantes identiquement distribuées (ce que l'on notera i.i.d. dans la suite) de loi exponentielle de paramètre 1 et posons : $\forall i \in \mathbf{N}^*, T_i = \xi_1 + ... + \xi_i$; *et* $\forall t \in \mathbf{R}_+, N_t = max\{i \in \mathbf{N} : T_i \leq t\}$.

Ces gains sont donnés par une suite de variables aléatoires strictement positives $(X_i)_{i\geq 1}$ i.i.d, et indépendantes du processus $(N_t)_{t\geq 0}$. Pour cela nous avons défini la fonction N_t qui prend en argument le temps (t) et qui renvoie donc N_t correspondant au nombre de joueurs ayant eu un gain à l'instant t.

```
def Nt(t):
    cpt=0
    T=0
    while T<=t:
        cpt +=1
        T +=expo(1,1)
    return cpt
```

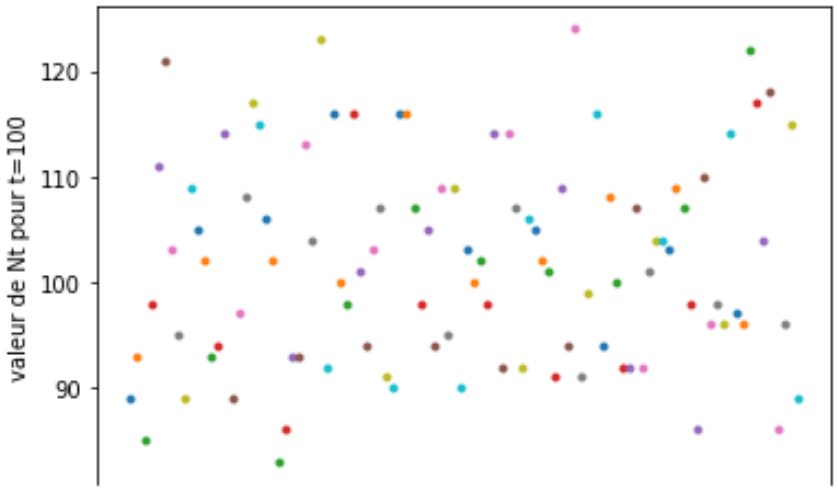

FIGURE 1 – Représentation du processus Nt pour t=100

Sur le graphique ci-dessus nous avons simuler 100 fois la valeur du processus N_t pour t=100. Grâce à cela on remarque bien que le temps d'arrêt N_t est un processus aléatoire.

2.2 Représentation du processus Y_t

On suppose désormais que X_1 admet un moment d'ordre 2 et on note : $E[X] = \mu$ et $V[X] = \sigma^2$. Pour tout $t \in \mathbf{R}_+$, on note Y_t le capital du casino à l'instant t. On a alors, si $Y_0 \geq 0$ est le capital initial du casino, pour tout instant $t \geq 0$,

$$Y_t = Y_0 + \alpha t - \sum_{i=1}^{N_t} X_i$$

Pour ce faire nous avons commencé par définir la fonction perte qui calcule la somme des gains des joueurs. Nous avons décidé que les gains $(X_i)_{i\geq 0}$ suivraient une loi exponentielle de paramètre

λ ou une loi uniforme sur [0 ;gain-max] où gain-max est le gain maximale que peut remporter un joueur. Voici le code obtenu pour une loi exponentielle :

```
def perte_loi_expo(la,t):
    nt=Nt(t)
    s=0
    for i in range(nt+1):

        s=s+expo(1,la)
    return s
```

Par la suite, nous avons donc défini la fonction capital qui calcule l'évolution du capital du casino de l'instant 0 à un instant tmax fixé.

```
def capital_loi_expo(y0,a,tmax,la):
    c=np.zeros(tmax+1)
    x=np.linspace(0,tmax,tmax+1)
    for i in range(tmax+1):
        c[i]=y0+a*i-perte_loi_expo(la,i)
    plt.plot(x,c,label=f'alpha={a}')
    plt.title("  volution  du capital du casino")
    plt.xlabel('temps')
    plt.ylabel("capital du casino  l'instant t")
    plt.legend()

    return c
```

Cette fonction prend en argument Y_0, a qui correspond au paramètre α, tmax ainsi que λ qui correspond au paramètre de la loi exponentielle.

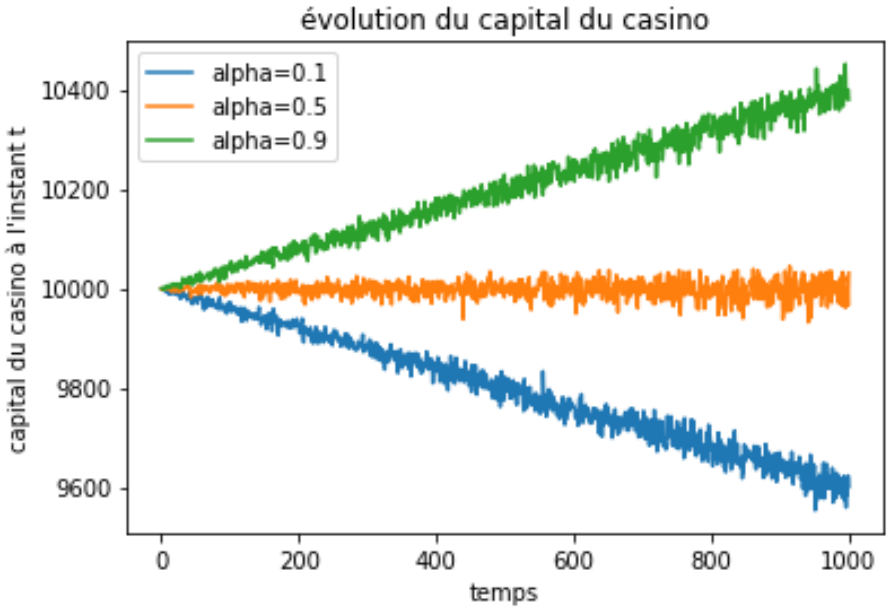

FIGURE 2 – Modélisation de l'évolution du capital pour plusieurs valeurs de α et les X_i qui suivent une loi exponentielle

Sur le graphique ci-dessus nous avons modélisé l'évolution du capital pour différentes valeurs de α sachant que les X_i suivent une loi exponentielle de paramètre 2 ($E[X] = 0.5$).

Nous pouvons observer qu'en fonction du choix de paramètre α la probabilité de ruine est plus ou moins élevée. Nous allons voir par la suite que le choix de ce paramètre est intimement lié à la probabilité de ruine.

En réitérant le même procédé avec cette fois-ci une loi uniforme sur [0,1], on remarque que l'évolution est similaire à celle d'une loi exponentielle à condition que les espérances soit égales entre elles (voir Annexe 1).

3 Probabilité de ruine

3.1 Introduction de la probabilité de ruine

L'objectif principal d'un gérant de casino est sans nul doute d'évaluer la probabilité qu'il soit un jour ruiné en fonction de son investissement initial Y. C'est pourquoi nous introduisons désormais r(Y) tel que :

$$r(Y) = P(\exists t \in \mathbf{R}_+ \; t.q \; Y_t < 0 \mid Y_0 = Y)$$

Bien évidemment, de son point de vue l'idéal est que cette probabilité soit la plus petite possible. Malheuresement pour lui ce n'est pas toujours le cas.

3.2 Cas $\alpha \leq \mu$

Dans cette partie nous nous plaçons dans le cas où $\alpha \leq \mu$ et nous allons démontrer à l'aide de méthodes de modélisation et de simulations que $\forall\, Y > 0$, $r(Y) = 1$. L'objectif est de montrer $\exists k > 1 \; t.q \;\; P(X_1 + ... + X_k > \alpha t + Y) > 0$. Pour cela nous avons fait appel à une méthode de Monte Carlo classique ou nous approchons $P(X_1 + ... + X_k > \alpha t + Y)$ par $E[1_{[X1+...+Xn>\alpha t+Y]}]$. En effet, en utilisant la loi fort des grands nombres on approche la valeur de la probabilité avec la moyenne empirique de l'espérance ci-dessus. Le code qui suit va donc nous servir à calculer la valeur de la probabilité de ruine dans le cas $\alpha \leq \mu$ tel que les X_i suivent une loi exponentielle de paramètre λ .

```
def MC(a,la,tmax,n,y0,k):
    r=np.zeros(n)
    for i in range(n):
        r[i]=(np.sum(expo(k,la))>=a*tmax+y0)
    m=np.mean(r)
    std=np.std(r)
    em=1.96*std/np.sqrt(n)
    return (m,std,m-em,m+em)
```

Ce code prend en argument α, λ, tmax qui représente le temps auquel on veut calculer la probabilité de ruine, n qui représente le nombre d'itération et k le nombre de loi exponentielle à simuler. Il renvoie par la suite l'approximation de la probabilité à calculer. À l'aide de ce code, nous obtenons la représentation graphique suivante. Nous avons pris $\alpha = 0.5$ et $\mu = 1$. Pour des raisons de rapidité de calcul nous choisi des valeurs d'investissement initial faibles afin de ne pas avoir à prendre un k trop grand.

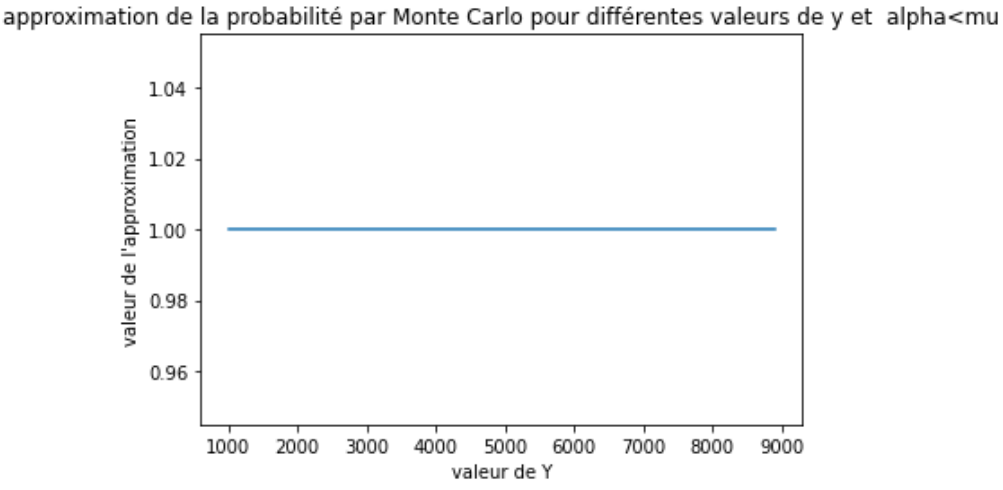

FIGURE 3 – Approximation de la probabilité de ruine pour Y allant de 1000 à 9000 avec $\alpha = 0.6$ et $\mu = 1$

Sur la figure 3 on voit bien que $\forall\, Y \in [1000; 9000]$, r(Y)=1. Cette assertion peu être généralisée pour toute valeur de Y.

De plus, lorsque $\alpha < \mu$, on a $\frac{Y_t}{t}$ tend vers α - μ. Ce qui implique r(Y)=1.

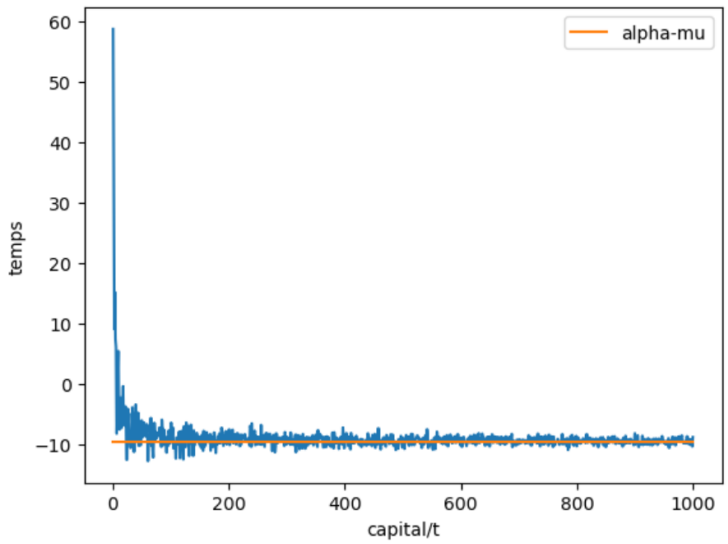

FIGURE 4 – Approximation de $\frac{Y_t}{t}$

Et dans le cas $\alpha = \mu$: On a $S_n = \sum_i^n (X_i\text{-}\alpha\xi_i)$ une marche aléatoire. Si S_n tend vers l'infini, alors r(Y)=1. Nous pouvons bien observer sur la figure 5 que lorsque n tend vers l'infini S_n tend également vers l'infini.

```
def Sn(n):
    S=np.zeros(n)
    S[0]=-np.log(np.random.rand())+np.log(np.random.rand())
    for i in range(n):
        S[i]=S[i-1]-np.log(np.random.rand())+np.log(np.random.rand())
    sup = np.maximum.accumulate(S)
    return(sup)
```

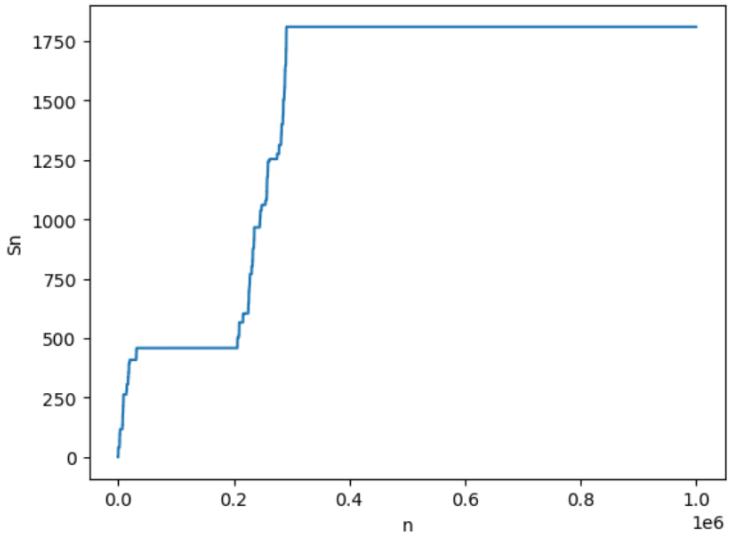

FIGURE 5 – Approximation de S_n

Enfin, lors de la modélisation du processus Y_t nous avons pu mettre en évidence que dès lors que $\alpha < \mu$ le capital chute drastiquement jusqu'à atteindre 0 (voir Annexe 2 et Annexe 3).

3.3 Cas $\alpha > \mu$

Après avoir traité le cas $\alpha \leq \mu$, nous devons traiter le cas $\alpha > \mu$, ce cas est plus complexe à simuler avec une méthode de Monte Carlo classique. En effet, il faudrait un nombre d'itérations

trop important pour trouver une valeur de la probabilité non nulle. Pour cela nous allons essayer d'estimer $P(X_1+...+X_k > \alpha t+Y)$ en lui appliquant la méthode de Monte Carlo par échantillonnage préférentiel (étant donné que dans les deux autres cas la probabilité est égale à 1). Lorsque T tend vers l'infini on a $\frac{N_t}{t} \to 1$. Ainsi N_t est équivalent à 1 quand t est trés grand. Alors on a :

$$P(X_1 + ... + X_k > \alpha t + Y) = P(X_1 + ... + X_t > \alpha t + Y) = P(X_1 + ... + X_t - Y > \alpha t)$$

De plus, quand t tend vers l'infini, $X_1 + ... + X_t - Y \simeq X_1 + ... + X_t$ et $\alpha > \mu$ ainsi, $\alpha = \mu + \epsilon$ avec $\epsilon \in \mathbf{R}_+$. Alors dans ce cas, nous pouvons approcher $P(X_1 + ... + X_k > \alpha t + Y)$ par $P(S_t \geq (\mu + \epsilon)t)$ avec $S_t = \sum_0^t X_i$. Le code ci-dessous renvoie donc l'approximation de Monte Carlo par échantillonnage préférentiel de la probabilité de ruine pour $\alpha > \mu$. Dans le cas où les X_i suivent une loi exponentielle de paramètre 1.

```
def MC_2(N,n,eps):
    result = np.zeros(N)
    theta = eps/(1+eps)
    for k in range(N):
        sum_xi_theta = np.sum(-(1/(1-theta))*np.log(np.random.rand(n)))
        if (sum_xi_theta>= n*(1+eps)):
            result[k] = ((1/(1-theta))**n)*np.exp(-theta*sum_xi_theta)
        else:
            result[k]= 0
    mean = np.mean(result)
    std = np.std(result)
    icsize = 1.96*std/np.sqrt(N)
    return np.array([mean,std,mean-icsize,mean+icsize])
```

Grâce à cette méthode nous avons pu observer que $\forall\ Y > 0$, $r(Y) > 0$. En revanche plus ϵ est grand, plus l'écart entre α et μ est grand et donc plus la probabilité de ruine tend vers 0 mais sans jamais l'atteindre. Nous pouvons observer ce phénomène sur le graphique et le tableau de données suivants :

TABLE 1 – Résultats de l'approximation Monte Carlo pour plusieurs valeurs de ϵ

Epsilon	**Valeurs de l'approximation**
0.1	0.0010829
0.2	1.22E-09
0.3	1.8E-18
0.4	8.3E-30
0.5	2.2E-43

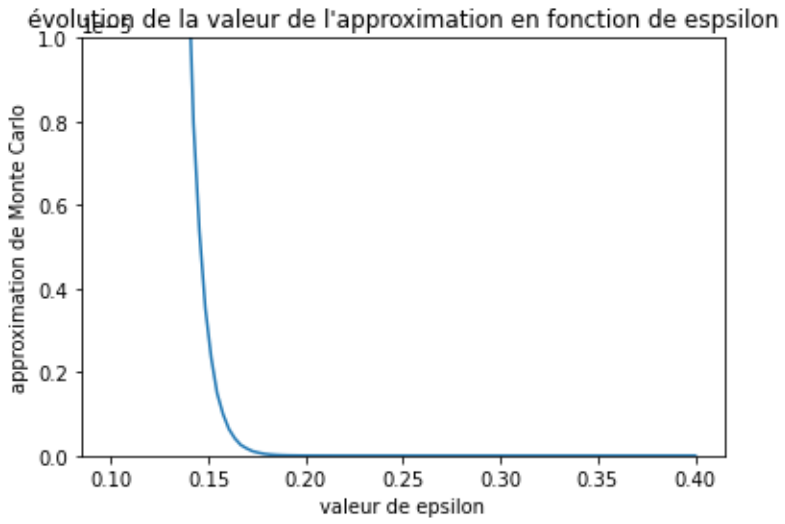

FIGURE 6 – Évolution de la valeur de l'approximation de Monte Carlo en fonction de espsilon

3.4 Calcul de la valeur exacte de r(Y) dans le cas de gains de lois exponentielles

Nous avons pu voir dans les sections passées que la probabilité de ruine est souvent difficile à calculer. En revanche, celle-ci peut être calculée explicitement lorsque les $(X_i)_{i\geq 1}$ sont distribués suivant une loi exponentielle de paramètre $\gamma = \frac{1}{\mu} > \frac{1}{\alpha}$. Dans ce cas on a :

$$r(Y) = \frac{e^{-(\gamma - 1/\alpha)Y}}{\alpha\gamma}$$

Nous pouvons observer ci-dessous l'évolution de la valeur de la probabilité de ruine lorsque $\mu = 1$.

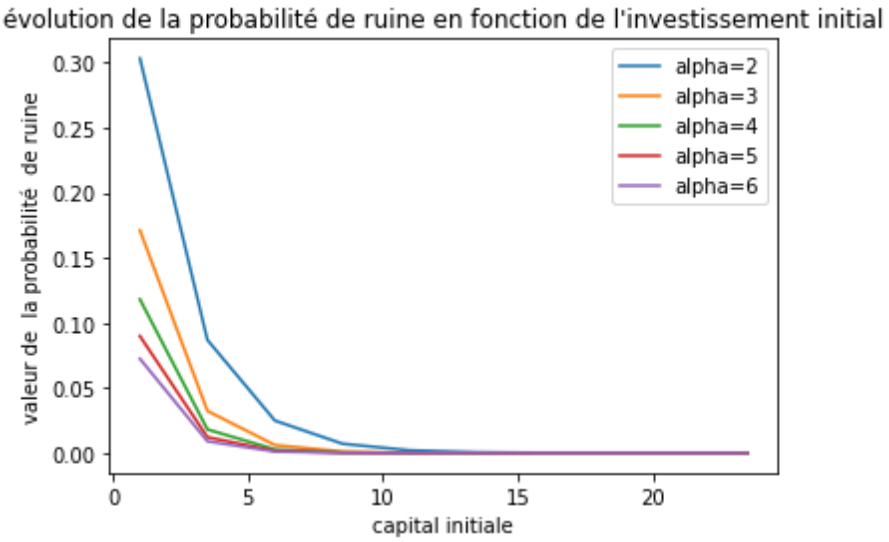

FIGURE 7 – Évolution de la probabilité de ruine en fonction de l'investissement initial pour $\mu = 1$

On peut observer que plus α est grand plus la probabilité de ruine tend vers 0 lentement, de plus, pour un investissement initial faible, plus α est petit plus la probabilité de ruine est grande.

Après avoir étudié le processus Y_t, simuler la probabilité de ruine dans différents cas de figures ; nous avons décider de nous intéresser à un autre modèle plus concret qui nous permettrait de mieux comprendre le choix des paramètres, et à établir des liens significatifs avec les résultats théoriques antérieurs. Par conséquent, nous allons désormais nous intéresser à un modèle où le casino connait ses gains ou ses pertes grâce au jeu de la roulette. L'objectif étant de savoir comment les caractéristiques uniques de ce jeu influent-elles sur la probabilité de ruine du casino ?

4 Approche simplifiée avec le jeu de la Roulette

4.1 Introduction

La roulette est un jeu de hasard emblématique, souvent trouvé dans les casinos du monde entier. Elle se compose d'une roue numérotée et d'un plateau de paris. La roue comporte 37 ou 38 numéros, selon la variante de roulette choisie. Le nombre de numéros est un choix crucial pour les casinos, comme nous le verrons par la suite.

Les joueurs peuvent décider de miser sur plusieurs types de jeux, tels que la couleur, la parité du numéro, le numéro, etc. Il existe une mise minimale et maximale pour chaque type de pari.

4.2 Création de la roulette

Pour modéliser la richesse d'un casino, nous avons initialement considéré que le casino ne proposait qu'un seul jeu, la roulette. Dans une première phase, nous avons tenté de recréer le

jeu de la roulette authentique, c'est-à-dire de simuler un tirage aléatoire d'un numéro compris entre 0 et 36. Ensuite, à chaque rotation de la roulette, nous avons simulé des paris pour un joueur sélectionné de manière aléatoire, avec une mise également choisie de manière aléatoire. Pour simuler la roulette, nous avons créé plusieurs fonctions représentant chaque type de pari possible. Pour mieux comprendre ces codes, chaques fonctions qui simulent les différents paris prennent en argument la mise du joueur et le numéro tiré lors du tour de la roulette, mise à part certaines. Celles où les paris peuvent se faire sur un groupe de numéro prennent en argument la grille de la roulette. Après avoir élaboré l'intégralité des paris, nous avons simulé un tour de roulette, en tenant compte des mises maximales pour chaque type de pari. Une fois la simulation de la roulette achevée, nous avons développé une fonction permettant de modéliser l'évolution de la richesse du casino au fil du temps. Plus précisément, à chaque unité de temps, nous avons procédé à un tirage aléatoire pour déterminer le nombre de joueurs, reproduisant ainsi fidèlement les conditions d'un casino. Cette fonction prend en argument le capital de départ du casino et le temps sur lequel la modélisation se fait. Toutes les fonctions qui ont permis de créer cette modélisation sont décrites en Annexe 4. Malheureusement, les résultats graphiques n'ont pas été ceux escomptés. Comme on peut le voir ci-dessous :

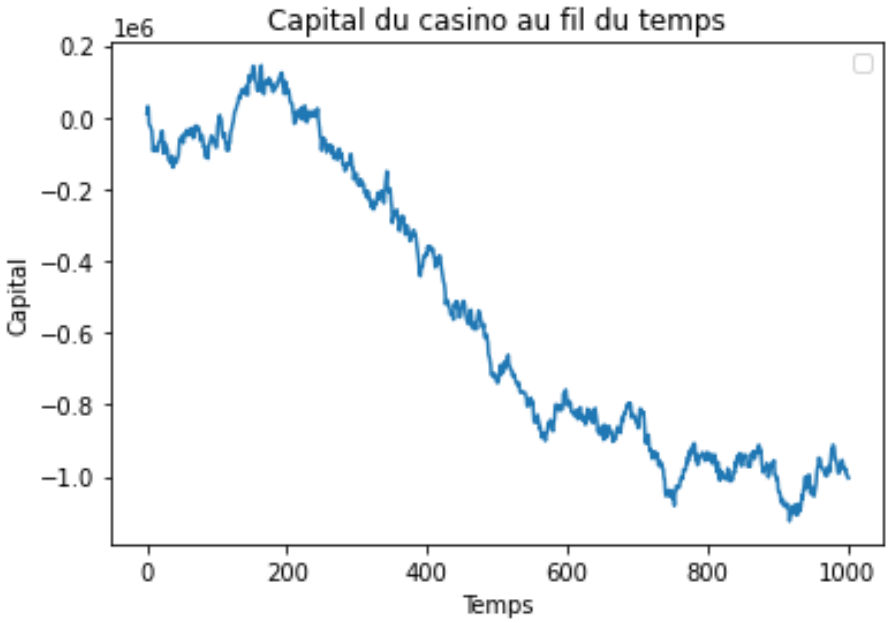

FIGURE 8 – Évolution du capital d'un casino proposant uniquement la roulette

On constate clairement qu'une erreur est présente, et la courbe devrait normalement suivre une tendance croissante. L'origine de cette erreur pourrait résulter d'un code trop volumineux en termes d'appels de fonctions, d'un potentiel problème au niveau des calculs, voire même d'un disfonctionnement des choix aléatoire.

Afin de remédier à cette erreur, nous avons pris la décision de revoir notre modèle en ajustant l'ensemble des fonctions liées aux paris potentiels. Cette révision intègre désormais automatiquement les probabilités de gains des joueurs grâce à l'utilisation d'une simulation de la loi uniforme $[0\,;1]$.

Une fois ces fonctions mises en place, la structure du code de la roulette et de la richesse du casino reste inchangée (voir Annexe 5). La seule modification apportée est l'ajout d'un paramètre permettant de définir le nombre de numéros de la roulette.

Désormais, les résultats graphiques sont cohérents, comme illustré ci-dessous :

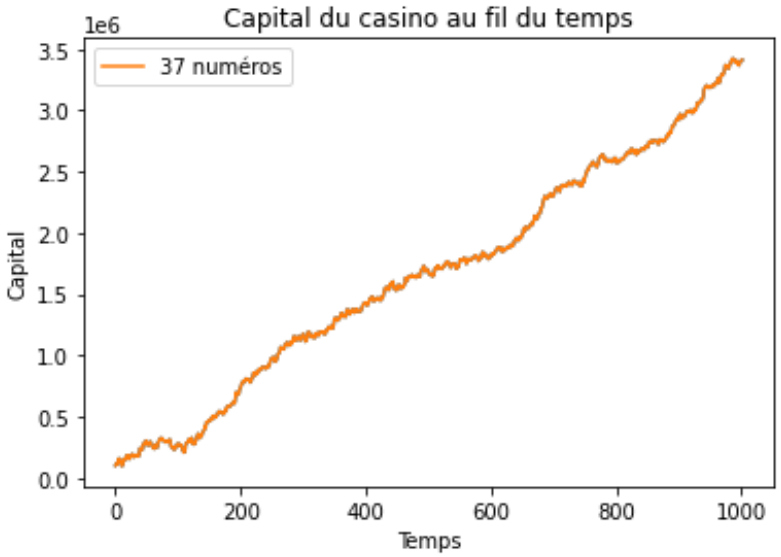

FIGURE 9 – Évolution du capital d'un casino proposant uniquement la roulette

4.3 Relation avec le modèle initial

Dans la version initiale du modèle, la variable Y_t était influencée à la fois par un paramètre α déterministe qui est le multiplicateur des gains du casino et par $\sum_{i=1}^{N_t} X_i$ qui est la somme des gains des joueurs, où les X_i sont des composantes aléatoires. Dans la version actuelle, le paramètre α et les X_i sont directement déterminés par le nombre de numéros existants. Notablement, l'espérance de gains des joueurs et le multiplicateur des gains du casino affichent une symétrie dans leur relation. Autrement dit, les ajustements apportés au modèle ont permis d'établir une connexion plus claire entre le paramètre α, le nombre de numéros, et la distribution des gains entre les joueurs et le casino.

En visualisant ces ajustements à travers un graphique, nous pourrons mieux appréhender la dynamique résultante entre le paramètre alpha, le nombre de numéros, et la répartition des gains entre les joueurs et le casino. (Voir ci-dessous)

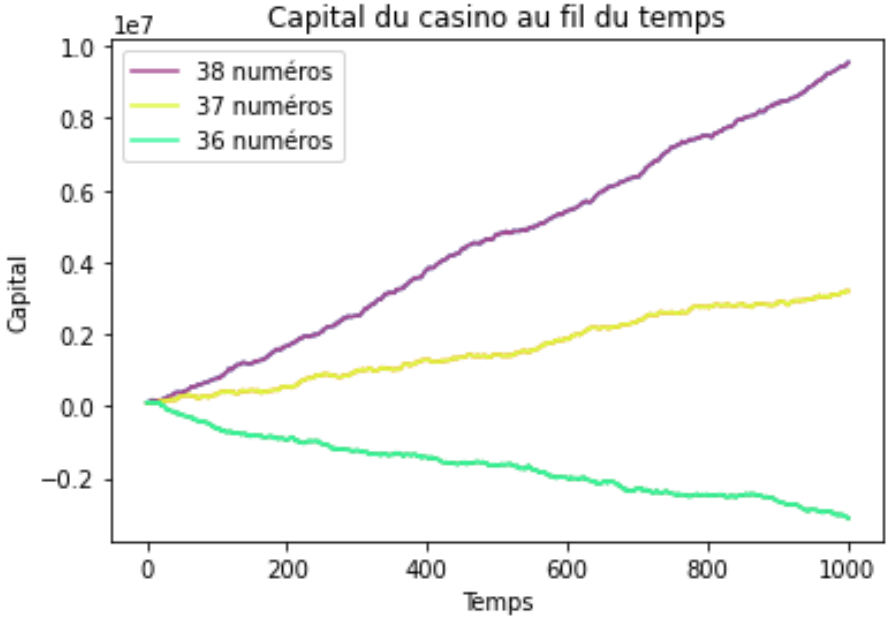

FIGURE 10 – Évolution du capital d'un casino proposant 3 types de Roulettes différentes

Nous pouvons remarquer que le casino peut utiliser le nombre de numéros en tant que levier pour faire évoluer son capitale d'une manière plus significative. C'est pourquoi, il existe différentes roulettes notemment l'européenne et l'americaine qui ont respectivement 37 et 38 numéros.

Prenons l'exemple de la Roulette Européenne, qui compte 37 numéros offrant aux joueurs la possibilité de parier sur la couleur (rouge ou noire). En cas de tirage du numéro 0 lors d'un tour de roulette, le casino s'empare de la moitié de la mise du joueur. Dans tous les autres cas, le casino remporte la totalité de la mise du joueur en cas de perte du joueur, ou reverse le montant de la mise si le joueur gagne.

D'un point de vue probabiliste, le casino démontre une probabilité de $\frac{19}{37}$ de remporter la manche, tandis que le joueur détient une probabilité de $\frac{18}{37}$. Il est intéressant de noter que dans cette configuration, $\alpha > \mu$ (avec $E[X_i] = \mu$) nous sommes en cohérence avec le modèle initial.

En supposant maintenant que le casino décide de retirer le 0 de la roulette, réduisant ainsi le nombre de numéros à 36, nous pouvons réexaminer le cas du joueur misant exclusivement sur la couleur. À ce stade, le casino et le joueur présentent chacun une probabilité de $\frac{1}{2}$ de remporter la manche. Cette modification de la configuration de la roulette souligne l'impact direct sur la distribution des probabilités et illustre comment les ajustements structurels influent sur les dynamiques de jeu. De plus, ici le paramètre $\alpha = 0.5$ est bien égale à μ, ce qui va induire un r(y)=1 (qui est la probabilité de ruine d'un casino) que l'on peut voir sur la Figure 10.

5 Conclusion

Cette étude se concentre sur une analyse approfondie de la gestion du capital dans l'industrie du jeu, avec une orientation spécifique vers la compréhension de la dynamique financière d'un casino. En utilisant des modèles sophistiqués basés sur des processus stochastiques et des lois probabilistes, notre démarche vise à approfondir la compréhension des mécanismes sous-jacents à la gestion du capital dans le contexte des établissements de jeu.

Au cœur de notre méthodologie réside le temps d'arrêt N_t déterminé par un processus de Poisson, qui modélise le nombre de joueurs remportant des gains à un instant t. Cette approche dynamique nous permet d'observer les fluctuations du capital du casino au fil du temps.

Les simulations de la probabilité de ruine, réalisées dans divers scénarios, mettent en valeur l'importance cruciale du choix judicieux des paramètres, en particulier α, dans la minimisation du risque de ruine. Ces résultats apportent des éclairages précieux pour une compréhension approfondie de la gestion financière d'un casino.

L'extension de notre modèle pour inclure le jeu de la roulette enrichit notre approche en tenant compte d'éléments caractéristiques spécifiques des casinos. Les ajustements apportés, en considérant le nombre de numéros dans la roulette, permettent une représentation plus réaliste des dynamiques financières des casinos.

Annexe 1

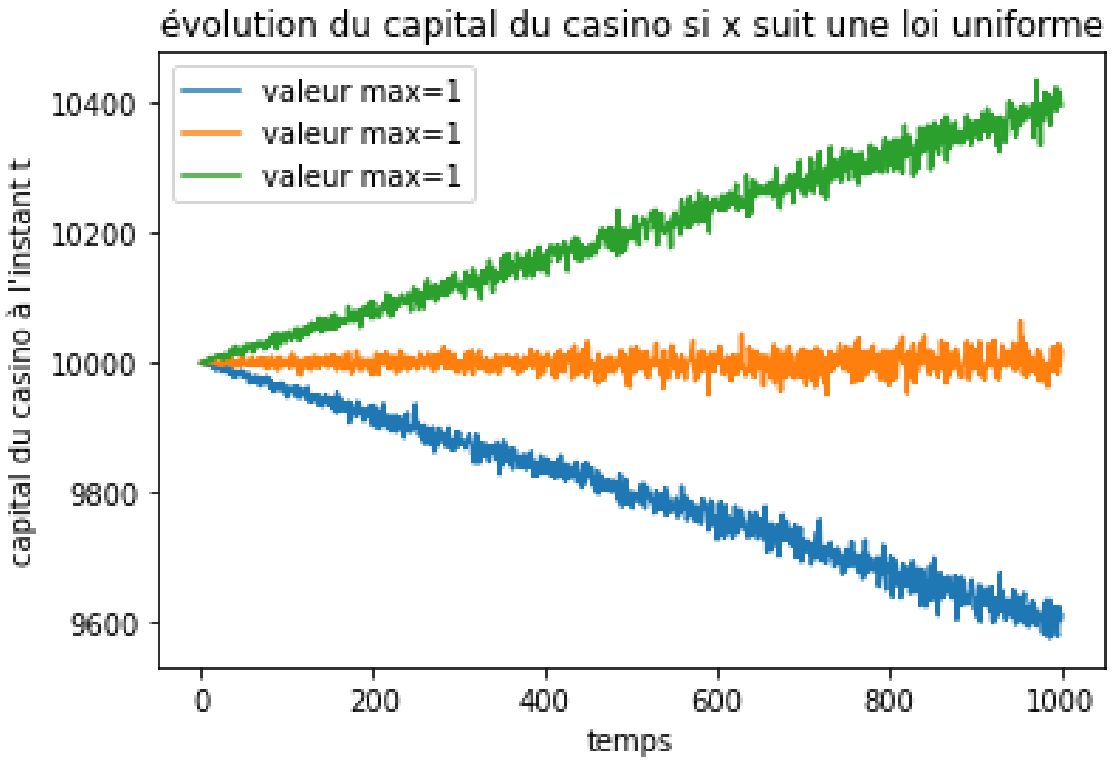

FIGURE 11 – Modélisation de l'évolution du capital pour plusieurs valeurs de α et les X_i qui suivent une loi uniforme

Annexe 2

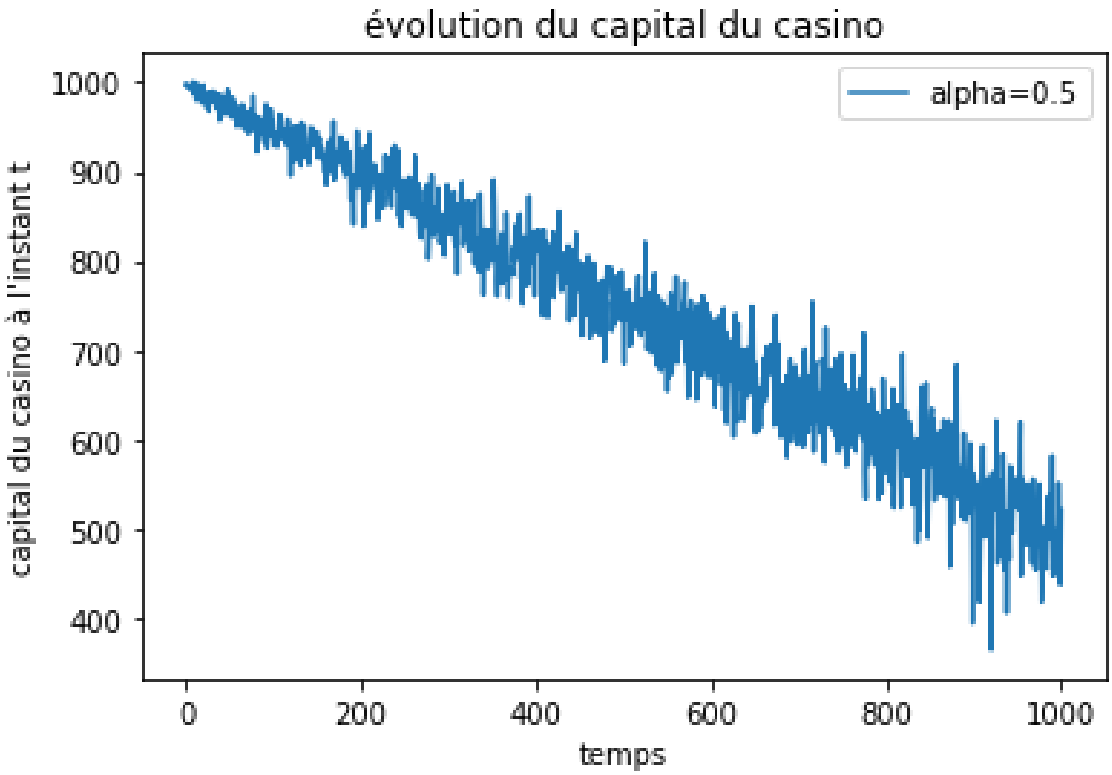

FIGURE 12 – Modélisation de l'évolution du capital pour $\alpha < \mu$ et les X_i qui suivent une loi exponentielle de paramètre 1

Annexe 3

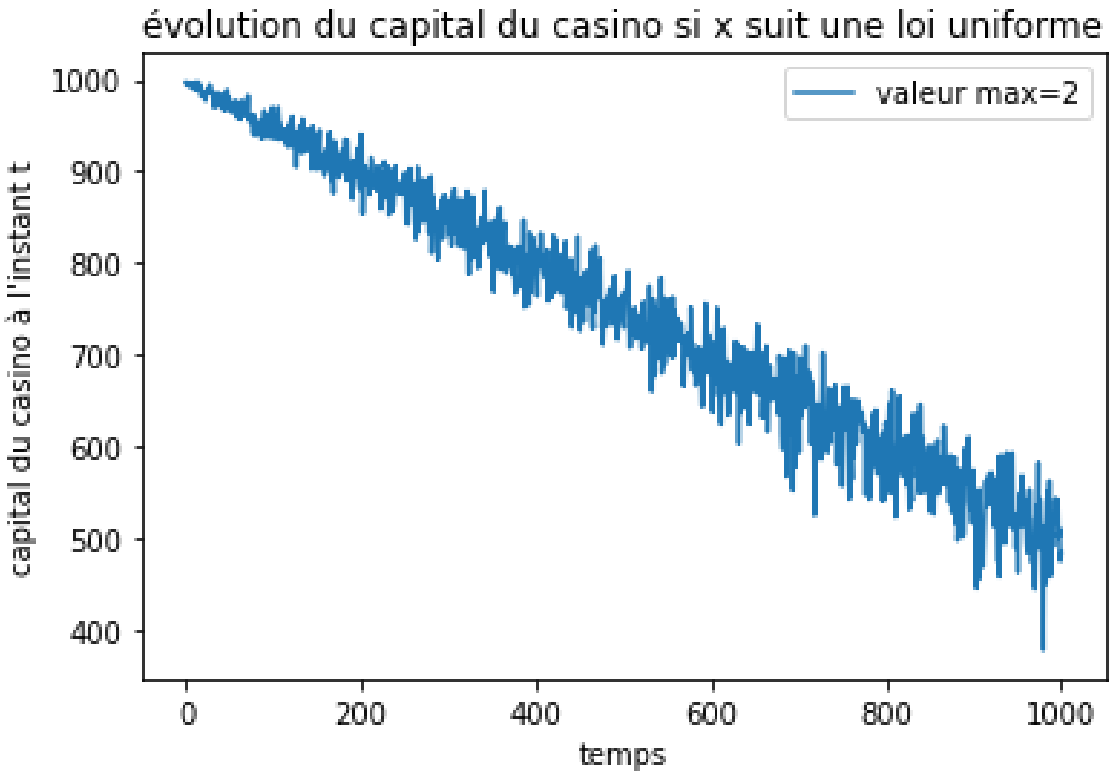

FIGURE 13 – Modélisation de l'évolution du capital pour $\alpha < \mu$ et les X_i qui suivent une loi uniforme [0,2]

Annexe 4

Script de la premiere simulation de la Roulette

```
import numpy as np
import matplotlib.pyplot as plt
def roul() :
    return np.random.randint(0,36)

def manque_passe(mise,numero_tire):
    liste_choix = ["manque","passe"]
    choix= np.random.randint(0,2)

    if numero_tire ==0 :
        gain = mise/2
    else:
        if liste_choix[choix]  == "passe":
            if (numero_tire>=19):
                gain= -mise
            else:
                gain= mise
        else :
            if(numero_tire<=18):
                gain=-mise
            else:
                gain=mise
    return gain

def pair_impair(mise,numero_tire):
    liste_choix= ["pair","impair"]
    choix= np.random.randint(0,2)

    if numero_tire == 0 :
        gain = mise/2
    else :
        if liste_choix[choix] == "pair" :
            if numero_tire%2 == 0:
                gain = -mise
            else :
                gain = mise
        else :
            if numero_tire%2 != 0:
                gain =-mise
            else:
                gain = mise
    return gain

def couleur(mise,numero_tire):
    liste_choix= ["rouge","noir"]
    choix= np.random.randint(0,2)
    numero_tire=roul()
    rouge=np.array([1,3,5,7,9,12,14,16,18,21,23,25,27,28,30,32,34,36])
    noir=np.array([2,4,6,8,10,11,13,15,17,19,20,22,24,26,29,31,33,35])
    if numero_tire == 0 :
        gain = mise/2
    else :
        if numero_tire in rouge:
            if liste_choix[choix]== "rouge":
                gain=-mise
            else:
                gain=mise
```

```
        else:
            if liste_choix[choix]== "noir":
                gain=-mise
            else:
                gain=mise
    return gain

def double(mise,numero_tire):
    choix=np.random.randint(1,4)
    if (choix==1):
        jeu=manque_passe(mise,numero_tire)
    elif(choix==2):
        jeu=pair_impair(mise,numero_tire)
    else:
        jeu=couleur(mise,numero_tire)
    return jeu

def tier(mise,numero_tire):
    choix=np.random.randint(1,4)
    if numero_tire==0 :
        gain=mise/2
    else:
        if choix==1 :
            if 1<=numero_tire<=12 :
                gain=-mise*2
            else:
                gain=mise
        elif choix==2 :
            if 13<=numero_tire<=24 :
                gain=-mise*2
            else:
                gain=mise
        else:
            if 25<=numero_tire<=36 :
                gain=-mise*2
            else:
                gain = mise
    return gain

def generer_groupes_carre(matrice):
    groupes = []
    lignes, colonnes = matrice.shape
    for i in range(lignes - 1):
        for j in range(colonnes - 1):
            groupe = [matrice[i, j], matrice[i, j+1], matrice[i+1, j], matrice[i+1, j
            groupes.append(groupe)
    groupes.append([0,1,2,3])
    return groupes

def selectionner_groupe_carre(groupe):
    if len(groupe) == 1:
        choix= groupe[0]
    else:
        choix = groupe[np.random.randint(0, len(groupe) - 1)]
    return choix

def generer_groupe_cheval(matrice) :
    groupes = []
    lignes, colonnes = matrice.shape
    for i in range(lignes) :
        for j in range (colonnes) :
```

```
            if j < colonnes-1 :
                groupedroite = [matrice[i,j],matrice [i,j+1]]
                groupes.append(groupedroite)
    for j in range (colonnes) :
        for i in range (lignes) :
            if i % 2 & i != 0 :
                groupehaut = [matrice[i,j],matrice[i-1,j]]
                groupebas = [matrice[i,j],matrice[i+1,j]]
                groupes.append(groupehaut)
                groupes.append(groupebas)
    groupes.append([0,1])
    groupes.append([0,2])
    groupes.append([0,3])
    return groupes

def selectionner_groupe_cheval(groupe) :
    choix = groupe[np.random.randint(0, len(groupe) - 1)]
    return choix

def selectionner_groupe_transversal(matrice):
    lignes , colonnes = matrice.shape
    groupes =[]
    for i in range(colonnes):
        groupes.append(matrice[:,i])
    choix = groupes[np.random.randint(0, len(groupes))]
    return choix

def selectionner_groupe_colonne(matrice):
    lignes , colonnes = matrice.shape
    groupes =[]
    for i in range(lignes):
        groupes.append(matrice[i,:])
    choix = groupes[np.random.randint(0, len(groupes))]
    return choix

def selectionner_groupe_sixain(matrice):
    lignes, colonnes = matrice.shape
    groupes = []
    for i in range(colonnes - 1):
        groupe_cote_a_cote = np.concatenate((matrice[:, i], matrice[:, i+1]), axis=0)
        groupes.append(groupe_cote_a_cote)
    choix = groupes[np.random.randint(0, len(groupes) - 1)]
    return choix

def roulette2() :
    mise = np.random.randint(1,1001)
    grille = np.array([[3,6,9,12,15,18,21,24,27,30,33,36],[2,5,8,11,14,17,20,23, 26,2
    numero_tire = roul()
    if mise <= 30 :
        choix = np.random.randint(1,9)
        if choix == 1 : #toutes les jeux double gains
            gain = double(mise,numero_tire)
        elif choix == 2 : # mises douzaine
            gain = tier(mise,numero_tire)
        elif choix == 3 : #jeu carr
            groupe = selectionner_groupe_carre(generer_groupes_carre(grille))
            if numero_tire in groupe :
                gain = -8*mise
            else:
                gain = mise
        elif choix == 4: #jeu cheval
```

```
            groupe = selectionner_groupe_cheval(generer_groupe_cheval(grille))
            if numero_tire in groupe :
                gain = -17 *mise
            else:
                gain = mise
        elif choix == 5 :  #jeu num ro plein
            numero_mise = roul()
            if numero_mise == numero_tire :
                gain=-35*mise
            else :
                gain = mise
        elif choix == 6 : #jeu transversale
            groupe = selectionner_groupe_transversal(grille)
            if numero_tire in groupe :
                gain = -10*mise
            else :
                gain = mise
        elif choix == 7 : #jeu sixain
            groupe = selectionner_groupe_sixain(grille)
            if numero_tire in groupe :
                gain = -4*mise
            else:
                gain = mise
        elif choix ==8 : #jeu colonne
            groupe = selectionner_groupe_colonne(grille)
            if numero_tire in groupe :
                gain = -2*mise
            else :
                gain = mise
    elif 30 <mise <= 60:
        choix = np.random.randint(1,8)
        if choix == 1 : #toutes les jeux double gains
            gain = double(mise,numero_tire)
        elif choix == 2 : # mises douzaine
            gain = tier(mise,numero_tire)
        elif choix == 3 : #jeu carre
            groupe = selectionner_groupe_carre(generer_groupes_carre(grille))
            if numero_tire in groupe :
                gain = -8*mise
            else:
                gain = mise
        elif choix == 4: #jeu cheval
            groupe = selectionner_groupe_cheval(generer_groupe_cheval(grille))
            if numero_tire in groupe :
                gain = -17 *mise
            else:
                gain = mise
        elif choix == 5 : #jeu transversale
            groupe = selectionner_groupe_transversal(grille)
            if numero_tire in groupe :
                gain = -10*mise
            else :
                gain = mise
        elif choix == 6 : #jeu sixain
            groupe = selectionner_groupe_sixain(grille)
            if numero_tire in groupe :
                gain = -4*mise
            else:
                gain = mise
        elif choix ==7 : #jeu colonne
            groupe = selectionner_groupe_colonne(grille)
```

```
            if numero_tire in groupe :
                gain = -2*mise
            else :
                gain = mise
    elif 60 < mise <= 100 :
        choix = np.random.randint(1,7)
        if choix == 1 : #toutes les jeux double gains
            gain = double(mise,numero_tire)
        elif choix == 2 : # mises douzaine
            gain = tier(mise,numero_tire)
        elif choix == 3 : #jeu carre
            groupe = selectionner_groupe_carre(generer_groupes_carre(grille))
            if numero_tire in groupe :
                gain = -8*mise
            else:
                gain = mise
        elif choix == 4 : #jeu transversale
            groupe = selectionner_groupe_transversal(grille)
            if numero_tire in groupe :
                gain = -10*mise
            else :
                gain = mise
        elif choix == 5 : #jeu sixain
            groupe = selectionner_groupe_sixain(grille)
            if numero_tire in groupe :
                gain = -4*mise
            else:
                gain = mise
        elif choix ==6 : #jeu colonne
            groupe = selectionner_groupe_colonne(grille)
            if numero_tire in groupe :
                gain = -2*mise
            else :
                gain = mise
    elif 100 < mise <=120 :
        choix = np.random.randint(1,6)
        if choix == 1 : #toutes les jeux double gains
            gain = double(mise,numero_tire)
        elif choix == 2 : # mises douzaine
            gain = tier(mise,numero_tire)
        elif choix == 3 : #jeu carre
            groupe = selectionner_groupe_carre(generer_groupes_carre(grille))
            if numero_tire in groupe :
                gain = -8*mise
            else:
                gain = mise
        elif choix == 4 : #jeu sixain
            groupe = selectionner_groupe_sixain(grille)
            if numero_tire in groupe :
                gain = -4*mise
            else:
                gain = mise
        elif choix ==5 : #jeu colonne
            groupe = selectionner_groupe_colonne(grille)
            if numero_tire in groupe :
                gain = -2*mise
            else :
                gain = mise
    elif 120 < mise <= 250 :
        choix = np.random.randint(1,5)
        if choix == 1 : #toutes les jeux double gains
```

```
            gain = double(mise,numero_tire)
        elif choix == 2 : # mises douzaine
            gain = tier(mise,numero_tire)
        elif choix == 3 : #jeu sixain
            groupe = selectionner_groupe_sixain(grille)
            if numero_tire in groupe :
                gain = -4*mise
            else:
                gain = mise
        elif choix ==4 : #jeu colonne
            groupe = selectionner_groupe_colonne(grille)
            if numero_tire in groupe :
                gain = -2*mise
            else :
                gain = mise
    elif 250 < mise <= 500:
        choix = np.random.randint(1,4)
        if choix == 1 : #toutes les jeux double gains
            gain = double(mise,numero_tire)
        elif choix == 2 : # mises douzaine
            gain = tier(mise,numero_tire)
        elif choix ==3 : #jeu colonne
            groupe = selectionner_groupe_colonne(grille)
            if numero_tire in groupe :
                gain = -3*mise
            else :
                gain = mise
    elif 500 < mise <=1000 :
            gain = double(mise,numero_tire)
    return gain

def fortune2(k_depart,temps):
    t= np.linspace(0,temps+1,temps+1)
    k_final= np.zeros(temps+1)
    k_final [0] = k_depart
    for i in range(1 , temps+1):
        nombre_joueur= np.random.randint(1,1001)
        gain=0
        for j in range(nombre_joueur):
            gain+=roulette2()
        k_final[i] = k_final[i-1] + gain
    plt.plot(t,k_final);plt.ylabel('Capital')
    plt.xlabel('Temps');plt.title("Capital du casino au fil du temps")
    plt.legend()
    plt.show()
    return k_final
```

Annexe 5

Script de la deuxième simulation de la Roulette

```
import numpy as np
import matplotlib.pyplot as plt
def tier (mise,n):
    proba= np.random.rand(1)
    if proba <=12/n:
        gain=-mise*2
    else :
        gain =mise
    return gain

def double(mise,n):
    proba=np.random.rand(1)
    if proba <=18/n:
        gain=-mise
    else :
        gain =mise
    return gain

def carre(mise,n):
    proba = np.random.rand(1)
    if proba <= 4/n:
        gain=-mise*8
    else :
        gain = mise
    return gain

def cheval(mise,n):
    proba = np.random.rand(1)
    if proba <= 2 / n:
        gain= -mise*17
    else:
        gain = mise
    return gain

def plein(mise,n):
    proba =np.random.rand(1)
    if proba <= 1/n:
        gain= -mise*35
    else:
        gain = mise
    return gain

def transversale (mise,n):
    proba =np.random.rand(1)
    if proba <= 3/n:
        gain = -mise*11
    else:
        gain = mise
    return gain

def sixain(mise,n):
    proba =np.random.rand(1)
    if proba <= 6/n:
        gain = -mise*11
    else:
        gain = mise
    return gain
```

```
def colonne(mise,n) :
    proba =np.random.rand(1)
    if proba <= 12/n:
        gain = -mise*2
    else:
        gain = mise
    return gain

def roulette(n) :
    mise = np.random.randint(1,1001)
    #grille = np.array([[3,6,9,12,15,18,21,24,27,30,33,36],[2,5,8,11,14,17,20,23, 26,
    #numero_tire = roul()
    if mise <= 30 :
        choix = np.random.randint(1,9)
        if choix == 1 : #toutes les jeux double gains
            gain = double(mise,n)
        elif choix == 2 : # mises douzaine
            gain = tier(mise,n)
        elif choix == 3 : #jeu carre
            gain = carre(mise,n)
        elif choix == 4: #jeu cheval
            gain = cheval(mise,n)
        elif choix == 5 :  #jeu num ro plein
            gain = plein(mise,n)
        elif choix == 6 : #jeu transversale
            gain = transversale(mise,n)
        elif choix == 7 : #jeu sixain
            gain = sixain(mise,n)
        elif choix ==8 : #jeu colonne
            gain = colonne(mise,n)

    elif 30 <mise <= 60:
        choix = np.random.randint(1,8)
        if choix == 1 : #toutes les jeux double gains
            gain = double(mise,n)
        elif choix == 2 : # mises douzaine
            gain = tier(mise,n)
        elif choix == 3 : #jeu carre
            gain = carre(mise,n)
        elif choix == 4: #jeu cheval
            gain=cheval(mise,n)
        elif choix == 5 : #jeu transversale
            gain = transversale(mise,n)
        elif choix == 6 : #jeu sixain
            gain = sixain(mise,n)
        elif choix ==7 : #jeu colonne
            gain = colonne(mise,n)

    elif 60 < mise <= 100 :
        choix = np.random.randint(1,7)
        if choix == 1 : #tous les jeux double gains
            gain = double(mise,n)
        elif choix == 2 : # mises douzaine
            gain = tier(mise,n)
        elif choix == 3 : #jeu carre
           gain = carre(mise,n)
        elif choix == 4 : #jeu transversale
            gain = transversale(mise,n)
        elif choix == 5 : #jeu sixain
            gain = sixain(mise,n)
```

```
            elif choix ==6 : #jeu colonne
                gain = colonne(mise,n)

        elif 100 < mise <=120 :
            choix = np.random.randint(1,6)
            if choix == 1 : #toutes les jeux double gains
                gain = double(mise,n)
            elif choix == 2 : # mises douzaine
                gain = tier(mise,n)
            elif choix == 3 : #jeu carre
                gain = carre(mise,n)
            elif choix == 4 : #jeu sixain
                gain = sixain(mise,n)
            elif choix ==5 : #jeu colonne
                gain = colonne(mise,n)

        elif 120 < mise <= 250 :
            choix = np.random.randint(1,5)
            if choix == 1 : #toutes les jeux double gains
                gain = double(mise,n)
            elif choix == 2 : # mises douzaine
                gain = tier(mise,n)
            elif choix == 3 : #jeu sixain
                gain = sixain(mise,n)
            elif choix ==4 : #jeu colonne
                gain = colonne(mise,n)

        elif 250 < mise <= 500:
            choix = np.random.randint(1,4)
            if choix == 1 : #toutes les jeux double gains
                gain = double(mise,n)
            elif choix == 2 : # mises douzaine
                gain = tier(mise,n)
            elif choix ==3 : #jeu colonne
                gain = colonne(mise,n)

        elif 500 < mise <=1000 :
                gain = double(mise,n)
        return gain

    def fortune(k_depart,temps,n):

        t= np.linspace(0,temps+1,temps+1)
        k_final= np.zeros(temps+1)
        k_final [0] = k_depart
        for i in range(1 , temps+1):
            nombre_joueur= np.random.randint(1,1001)
            gain=0
            for j in range(nombre_joueur):
                gain+=roulette(n)
            k_final[i] = k_final[i-1] + gain
        plt.plot(t,k_final)
        couleur = "#{:06x}".format(np.random.randint(0, 0xFFFFFF))
        plt.plot(t,k_final, label =str(n)+'numeros',color=couleur)
        plt.ylabel('Capital')
        plt.xlabel('Temps');plt.title("Capital du casino au fil du temps")
        plt.legend()
        return k_final
    plt.figure()
```