

# Modélisation statistique des matchs de football par la loi de Poisson

Positionnements thématiques :

- Mathématiques
- Informatique pratique

Code cnc : CA217M

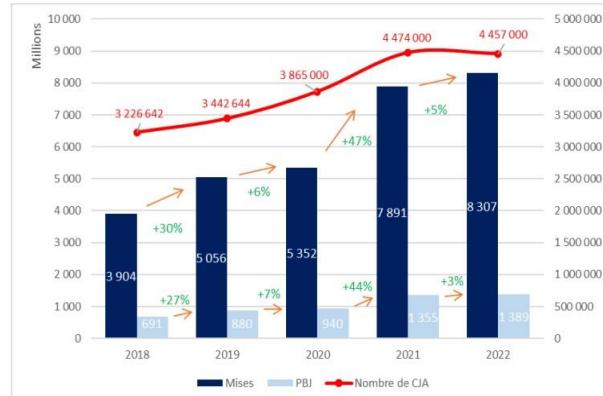
**Oudrhiri Idrissi**

Safwane





Graphique n°14 : évolution des principaux indicateurs économiques en pari sportif en ligne entre 2018 et 2022 (en M€)



## Image 1 et 2 : paris sportifs

REGRAGUI :  
« ON  
A SIMPLEMENT  
FAIT DÉJOUER  
LES XG ET LES  
TRUCS  
D'HARVARD »

à lire sur [SOFOOT.com](http://SOFOOT.com)

A composite image featuring a portrait of a man with a beard and a soccer ball in the background, followed by a quote from SOFOOT.com about a coach named REGRAGUI.

Image 3 : avis d'un coach  
sur les statistiques dans le football



## échauffement

### *Visualisation mathématique du football :*

- ✓ Loi de Bernoulli
- ✓ Loi binomiale
- ✓ De la loi binomiale à la loi de Poisson

## 1ère mi-temps

### *Une première modélisation :*

- ✓ Un lambda pas si lambda
- ✓ Comment simuler un championnat
- ✓ Un premier résultat
- ✓ Une piste d'amélioration

## 2ème mi-temps

### *Une modélisation plus poussé*

- ✓ Optimisation
- ✓ Un comparatif
- ✓ Un programme pratique

# Échauffements

## Visualisation mathématique du football:

**Objectif :**

**Trouver un modèle mathématique fiable pour modéliser le football**

## Visualisation mathématique du football:

### Loi de Bernoulli



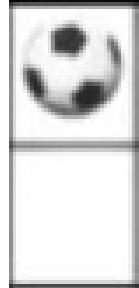
But = réussite:

$$P(X_n = \text{but}) = p$$



Pas but = échec:

$$P(X_n = \text{pas but}) = 1 - p$$



**Proposition 1:**

**La variable aléatoire  $X_n$  donnant le nombre de buts marqués sur une période n suit une loi de Bernoulli de paramètre p  $X \sim \mathcal{B}(p)$**

**Hyothèses:**

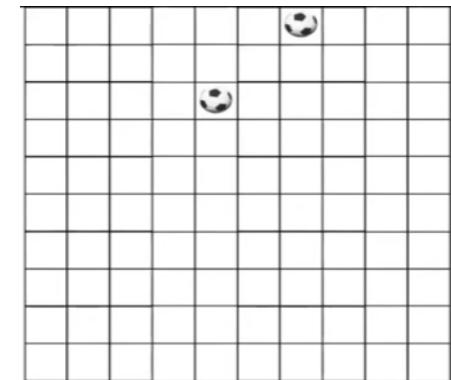
- ❖ p est constant
- ❖ un unique but peut etre marqué sur chaque période

**Proposition 2:**

**La variable aléatoire X donnant le nombre de buts au cours du match suit une loi binomiale de paramètre n et p     $X \hookrightarrow \mathcal{B}(n, p)$**

**Contraintes :**

- Plusieurs buts pourraient être marqués sur une même période
- Choix de la durée d'une période



**Exemple 1:** match de 100 périodes



**Image 4 :** 2 buts coup sur coup

**Solution :**

**Avoir une infinité de petites périodes**

**Théorème :**

**Soit ( $X_n$ ) une suite de variables aléatoires telles que:**

$$\forall n \in \mathbb{N}; \quad X_n \sim \mathcal{B}(n, p)$$

**Si  $p \rightarrow 0$ :**

**Alors ( $X_n$ ) converge en loi vers une variable aléatoire de Poisson de paramètre  $np$ , c-à-d:**

$$\forall k \in \mathbb{N}; \quad P(X_n = k) \rightarrow \frac{e^{-\lambda} \lambda^k}{k!}$$



# Coup d'envoi

## *Une première modélisation :*

**Objectifs :**

**Détermination du lambda et  
première simulation d'un  
championnat**

**Une première modélisation :**

Un lambda pas si lambda

**Proposition 1:**

**Soit  $\lambda > 0$  et X une variable aléatoire telle que :  $X \hookrightarrow \mathcal{P}(\lambda)$**

**Alors :  $E(X) = \lambda$**

**Conséquence:**

$\lambda$  représente la valeur prise par X en moyenne .

Une première modélisation :

Un lambda pas si lambda

$$\lambda = \frac{\text{Total buts marqués}}{\text{Arsenal} + \text{Arsenal} + \text{Manchester City} + \text{Chelsea} + \dots}$$

## Une première modélisation :

## Un premier résultat

```
def remplir_matrice_X(data, equipes):
    nb_equipes = len(equipes)
    X = np.zeros((nb_equipes, nb_equipes))
    for index, row in data.iterrows(): #.iterrows()
        equipe_domicile = row['HomeTeam'] # Le nom
        equipe_exterieure = row['AwayTeam'] #celui
        buts_domicile = row['FTHG']
        #on récupère les indices des équipes dans l
        i = equipes.index(equipe_domicile)
        j = equipes.index(equipe_exterieure)
        X[i][j] = buts_domicile
    return X

#La matrice Y dont le coefficient de la ième ligne
def remplir_matrice_Y(data, equipes):
    nb_equipes = len(equipes)
    Y = np.zeros((nb_equipes, nb_equipes))
    for index, row in data.iterrows():
        equipe_domicile = row['HomeTeam']
        equipe_exterieure = row['AwayTeam']
        buts_domicile = row['FTAG']
        i = equipes.index(equipe_domicile)
        j = equipes.index(equipe_exterieure)
        Y[i][j] = buts_domicile
    return Y
```

Man city

Arsenal	[0. 0. 3. 2. 2. 3. 5. 5. 2. 2. 3. 2. 1. 3. 4. 2. 5. 2. 0. 2.]
Aston Villa	[1. 0. 3. 3. 6. 3. 2. 3. 4. 3. 3. 3. 1. 1. 1. 4. 1. 0. 4. 2.]
Bournemouth	[0. 2. 0. 1. 3. 2. 0. 1. 2. 3. 0. 4. 0. 2. 2. 1. 2. 0. 1. 1.]
Brentford	[0. 1. 2. 0. 0. 3. 2. 1. 1. 0. 1. 3. 1. 1. 2. 3. 2. 2. 3. 1.]
Brighton	[0. 1. 3. 2. 0. 1. 1. 4. 1. 1. 2. 4. 0. 0. 3. 1. 1. 4. 1. 0.]
Burnley	[0. 1. 0. 2. 1. 0. 1. 0. 0. 2. 0. 1. 0. 0. 1. 1. 5. 2. 1. 1.]
Chelsea	[2. 0. 2. 0. 3. 2. 0. 2. 6. 1. 1. 3. 4. 4. 3. 0. 2. 2. 5. 2.]
Crystal Palace	[0. 5. 0. 3. 1. 3. 1. 0. 2. 0. 1. 1. 2. 4. 2. 0. 3. 1. 5. 3.]
Everton	[0. 0. 3. 1. 1. 1. 2. 1. 0. 0. 2. 1. 1. 0. 3. 2. 1. 2. 1. 0.]
Fulham	[2. 1. 3. 0. 3. 0. 0. 1. 0. 0. 1. 1. 0. 0. 0. 5. 3. 3. 5. 3.]
Liverpool	[1. 3. 3. 3. 2. 3. 4. 0. 2. 4. 0. 4. 1. 0. 4. 3. 3. 4. 3. 2.]
Luton	[3. 2. 2. 1. 4. 1. 2. 2. 1. 2. 1. 0. 1. 1. 1. 0. 1. 1.]
Man City	[0. 4. 6. 1. 2. 3. 1. 2. 2. 5. 1. 5. 0. 3. 1. 2. 2. 3. 3. 5.]
Man United	[0. 3. 0. 2. 1. 1. 2. 0. 2. 1. 2. 1. 0. 0. 3. 3. 4. 2. 3. 1.]
Newcastle	[1. 5. 2. 1. 1. 2. 4. 4. 1. 3. 1. 4. 2. 1. 0. 1. 5. 4. 4. 3.]
Nott'm Forest	[1. 2. 2. 1. 2. 1. 0. 3. 0. 2. 0. 2. 2. 0. 0. 2. 0. 2. 2.]
Sheffield United	[0. 0. 1. 1. 0. 1. 2. 0. 2. 3. 0. 2. 1. 1. 0. 1. 0. 0. 2. 2.]
Tottenham	[2. 1. 3. 3. 2. 2. 1. 3. 2. 2. 2. 0. 2. 4. 3. 2. 0. 1. 1.]
West Ham	[0. 1. 1. 4. 0. 2. 3. 1. 0. 0. 2. 3. 1. 2. 2. 3. 2. 1. 0. 3.]
Wolves	[0. 1. 0. 0. 1. 1. 2. 1. 3. 2. 1. 2. 2. 3. 2. 1. 1. 2. 1. 0.]

**Exemple 2 :** correspondance entre X et équipes

## Une première modélisation :

## Un premier résultat

```
def somme_buts_marqués(X, Y, i):
    tot_buts = 0
    for j in range(len(X)):
        if j != i:
            tot_buts += X[i][j] + Y[j][i]
    return tot_buts

#calcule le nombre de buts marqués en moyenne par une
def calcul_lambda(X,Y,n):
    lamda=np.zeros((n,1))
    for i in range(n):
        tot_buts_i=somme_buts_marqués(X,Y,i)
        tot_matchs= (n-1)*2
        lamda[i][0]=tot_buts_i/tot_matchs
    return lamda
```

```
def simu_match_1(lamda,i, j):
    team1_goals = np.random.poisson(lamda[i][0])
    team2_goals = np.random.poisson(lamda[j][0])

    if team1_goals > team2_goals:
        result = 'Team 1 wins'
    elif team1_goals < team2_goals:
        result = 'Team 2 wins'
    else:
        result = 'Draw'

    return team1_goals, team2_goals, result
```

```
#simulation complète 1
def simu_champ_1(data):
    equipes= sorted(set(data[ 'HomeTeam' ]))
    X=remplir_matrice_X(data,equipes)
    Y=remplir_matrice_Y(data, equipes)
    lamdas= calcul_lambda(X,Y,len(equipes))
    print('classement espéré:')
    classement_esp=classement_1(lamdas,equipes, 10000)
    print(classement_esp)
```

3/

## Une première modélisation :

## Un premier résultat

```
def classement_1(lamda, equipes, n):
    num_simulations = n
    total_points = {team: 0 for team in equipes} #dictionnaire qui va contenir le nombre total de points pour chaque équipe
    points_esp = {team: 0 for team in equipes} #dictionnaire qui va contenir le nombre de points par équipe pour la simulation

    for _ in range(num_simulations):
        points = {team: 0 for team in equipes}

        for i in range(len(equipes)):
            for j in range(len(equipes)):
                if i != j:
                    team1_goals, team2_goals, result = simulate_match_1(lamda, i, j)
                    if team1_goals > team2_goals:
                        points[equipes[i]] += 3
                    elif team1_goals < team2_goals:
                        points[equipes[j]] += 3
                    else:
                        points[equipes[i]] += 1
                        points[equipes[j]] += 1

        for team in equipes:
            total_points[team] += points[team]
    for team in equipes:
        points_esp[team]=total_points[team]/num_simulations

    # conversion du dictionnaire des points en moyenne en DataFrame pour un tri facile et une analyse ultérieure
    df_classement = pd.DataFrame.from_dict(points_esp, orient='index', columns=['Points'])
    df_classement = df_classement.sort_values(by='Points', ascending=False)

    return df_classement
```

```
#simulation complète 1
def simu_champ_1(data):
    equipes= sorted(set(data['HomeTeam']))
    X=remplir_matrice_X(data,equipes)
    Y=remplir_matrice_Y(data, equipes)
    lamdas= calcul_lambda(X,Y,len(equipes))
    print('classement espéré:')
    classement_esp=classement_1(lamdas,equipes, 10000)
    print(classement_esp)
```

## Une première modélisation :

## Un premier résultat

	Points		Points
<b>Man City</b>	91	Man City	73.3459
<b>Arsenal</b>	89	Arsenal	70.6318
<b>Liverpool</b>	82	Liverpool	67.9901
<b>Aston Villa</b>	68	Newcastle	67.4121
<b>Tottenham</b>	66	Chelsea	62.7062
<b>Chelsea</b>	63	Aston Villa	62.1597
<b>Newcastle</b>	60	Tottenham	60.8343
<b>Man United</b>	60	West Ham	51.7053
<b>West Ham</b>	52	Crystal Palace	49.8482
<b>Crystal Palace</b>	49	Man United	49.7536
<b>Brighton</b>	48	Brentford	49.0098
<b>Everton</b>	48	Fulham	48.5034
<b>Bournemouth</b>	48	Brighton	48.4078
<b>Fulham</b>	47	Bournemouth	47.6998
<b>Wolves</b>	46	Luton	46.2814
<b>Brentford</b>	39	Wolves	44.9190
<b>Nott'm Forest</b>	36	Nott'm Forest	44.2838
<b>Luton</b>	26	Burnley	38.2936
<b>Burnley</b>	24	Everton	37.5610
<b>Sheffield United</b>	16	Sheffield United	33.6780

Classement réel

Classement simulé

## Une première modélisation :

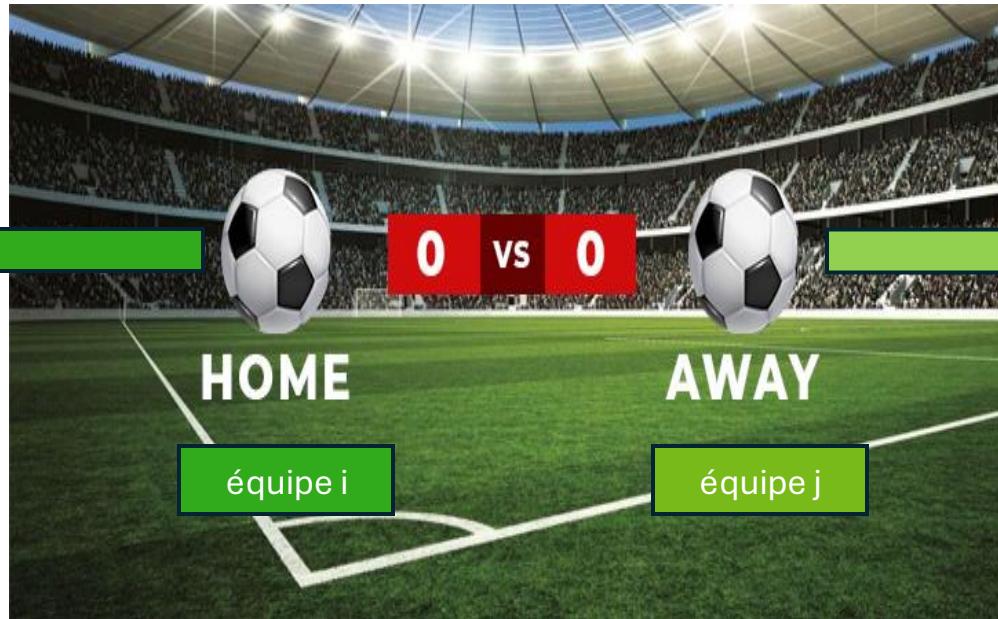
## Une piste d'amélioration

	points	buts_marques	buts_encaisses	diff_buts
<b>Man City</b>	91	96	34	62
<b>Arsenal</b>	89	91	29	62
<b>Liverpool</b>	82	86	41	45
<b>Aston Villa</b>	68	76	61	15
<b>Tottenham</b>	66	74	61	13
<b>Chelsea</b>	63	77	63	14
<b>Newcastle</b>	60	85	62	23
<b>Man United</b>	60	57	58	-1
<b>West Ham</b>	52	60	74	-14
<b>Crystal Palace</b>	49	57	58	-1
<b>Brighton</b>	48	55	62	-7
<b>Everton</b>	48	40	51	-11
<b>Bournemouth</b>	48	54	67	-13
<b>Fulham</b>	47	55	61	-6
<b>Wolves</b>	46	50	65	-15
<b>Brentford</b>	39	56	65	-9
<b>Nott'm Forest</b>	36	49	67	-18
<b>Luton</b>	26	52	85	-33
<b>Burnley</b>	24	41	78	-37
<b>Sheffield United</b>	16	35	104	-69

Classement avec les buts

## Une première modélisation :

Une piste d'amélioration



Variable aléatoire  
donnant le nombre  
de buts marqués par  
l'équipe i contre  
l'équipe j

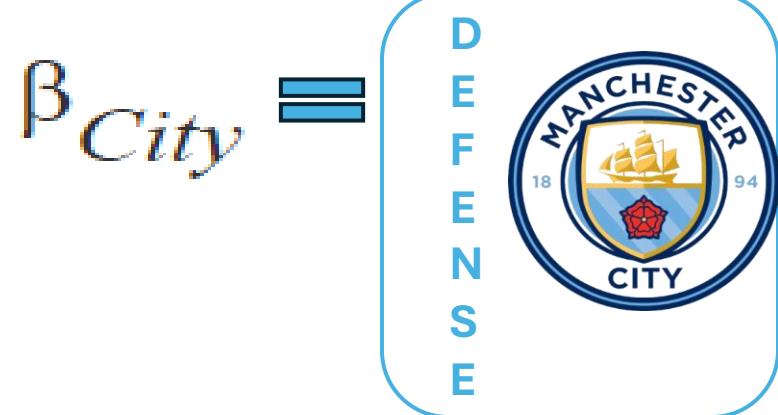
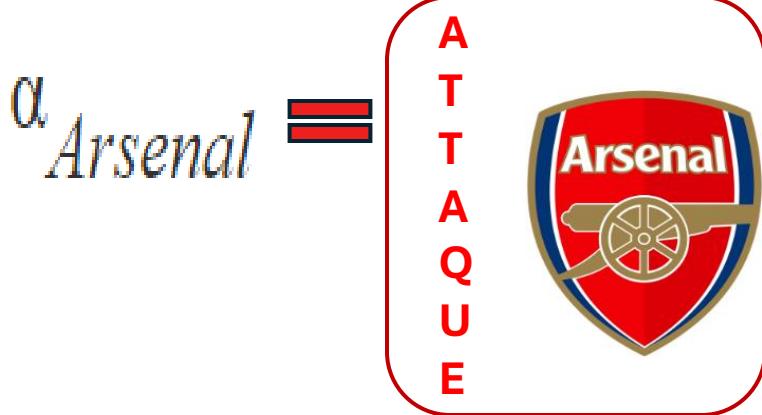
$$X_{i,j} \hookrightarrow P(\lambda_{i,j})$$

Variable aléatoire  
donnant le nombre  
de buts marqués par  
l'équipe j contre  
l'équipe i

$$Y_{i,j} \hookrightarrow P(\mu_{i,j})$$

Hyothèses:

- ❖  $X_{i,j}$  et  $Y_{i,j}$  sont indépendantes
- ❖  $\lambda_{ij} = \alpha_i \beta_j$  et  $\mu_{ij} = \alpha_j \beta_i$



**Une première modélisation :**

Une piste d'amélioration

**conséquence:**

$$P(X_{i,j} = x, Y_{i,j} = y) = P(X_{i,j} = x) \ P(Y_{i,j} = y)$$

## *Une meilleure modélisation :*

**Objectif :**

- Déterminer les coefficients  $\alpha$  et  $\beta$
- Effectuer une nouvelle modélisation

*Une meilleure modélisation :*

Optimisation

Problème :

**Comment déterminer ces coefficients**

Solution :

**La vraisemblance**

**Une meilleure modélisation :**

Optimisation

Dans notre cas :

Fonction de vraisemblance ;

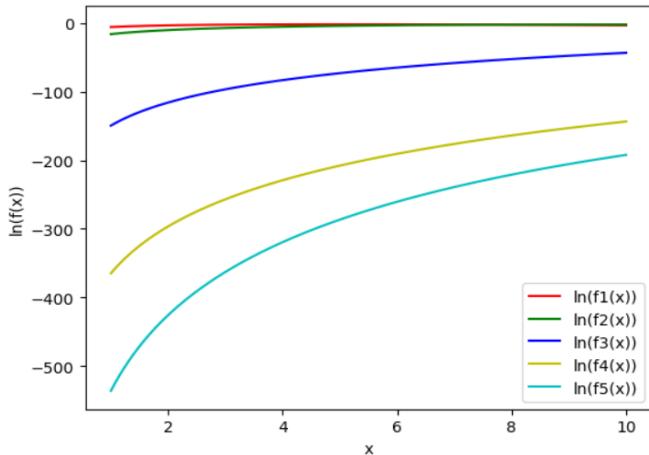
$$F : \mathbb{R}^{2n} \rightarrow \mathbb{R}$$
$$\left( \alpha_1, \dots, \alpha_n, \beta_1, \dots, \beta_n \right) \mapsto \prod_{1 \leq i \neq j \leq n} P(X_{ij} = x_{ij} \cap Y_{ij} = y_{ij})$$

Problème :

Comment maximiser la vraisemblance ?

En pratique:

On étudie:  $\ln(F(\alpha, \beta)) = \sum_{\substack{1 \leq i, j \leq n \\ i \neq j}} x_{ij} \ln(\lambda_{ij}) - \lambda_{ij} + y_{ij} \ln(\mu_{jj}) - \mu_{ij} - \ln(x_{ij}! y_{ij}!)$



**Figure 1 :**  
concavité de la fonction  $\ln \circ F$

Une meilleure modélisation :

Optimisation

Remarque :

**$\ln(F)$  est concave**

Conséquence:

**Elle atteint un maximum en un point si et seulement si sa différentielle y est nulle**

$$\frac{\partial(\ln F)}{\partial \alpha_i} = 0, \text{ pour } i = 1, \dots, n$$

$$\frac{\partial(\ln F)}{\partial \beta_i} = 0, \text{ pour } i = 1, \dots, n$$

Une meilleure modélisation :

## Optimisation

$\alpha_{Arsenal}$



2

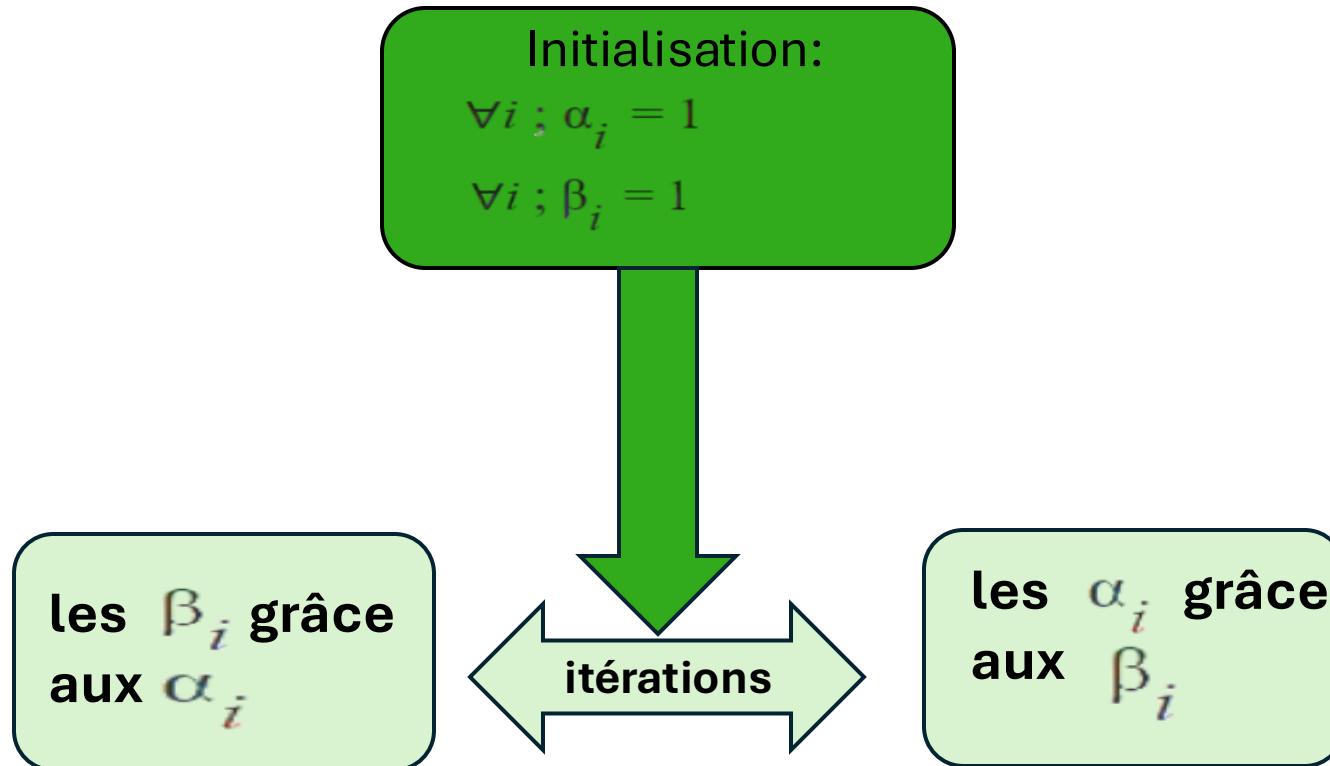


$\beta_{City}$



2



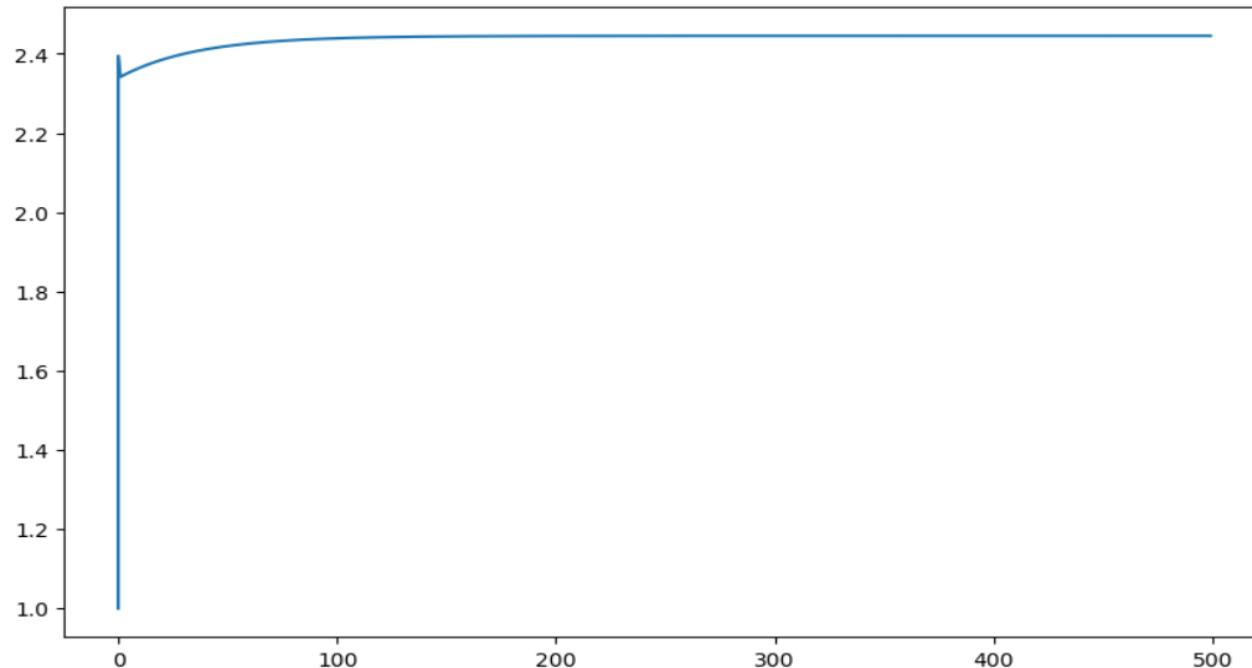


**Schéma 1 :** algorithme de résolution

```
def calcul_coeffs(X, Y):
    nb_equipes = len(X)
    alpha = np.ones((nb_equipes, 1))
    beta = np.ones((nb_equipes, 1))
    for a in range(500):
        for i in range(nb_equipes):
            if i!=16:
                S2 = somme_coeffs(beta, i)
                S1 = somme_buts_marqués(X, Y, i)
                if S2 != 0:
                    alpha[i] = S1 / S2
            for i in range(nb_equipes):
                S4 = somme_buts_encaissés(X, Y, i)
                S3 = somme_coeffs(alpha, i)
                if S3 != 0:
                    beta[i] = S4 / S3
    return alpha, beta
```

Une meilleure modélisation :

Optimisation



**Figure 2 :** convergence de l'algorithme

## Une meilleure modélisation :

## Optimisation

```
#calcule du Lambda et du mu associés au match de i cont
def lamda(alpha_beta, i, j):
    alpha_i = alpha_beta[i][0]
    beta_j = alpha_beta[j][1]
    return alpha_i * beta_j

def mu(alpha_beta, i, j):
    beta_i = alpha_beta[i][1]
    alpha_j = alpha_beta[j][0]
    return alpha_j * beta_i

#on réunit tous les Lambda et les mu dans respectivemer
def matrices(alpha_beta):
    L = np.zeros((20, 20))
    M = np.zeros((20, 20))
    for i in range(20):
        for j in range(20):
            if j != i:
                L[i][j] = lamda(alpha_beta, i, j)
                M[i][j] = mu(alpha_beta, i, j)
    return L, M
```

```
def simu_champ_2(data):
    equipes = sorted(set(data['HomeTeam']))
    X = remplir_matrice_X(data, equipes)
    Y = remplir_matrice_Y(data, equipes)
    alpha, beta = calcul_coeffs(X, Y)
    alpha_beta = np.concatenate((alpha, beta), axis=1)
    L, M = matrices(alpha_beta)
    classement_final=classement_2(L, M, equipes, 10000)
    print('le classement espéré est')
    print(classement_final)
```

## Une meilleure modélisation :

## Optimisation

	alphas	betas	Points
Man City	2.591135	0.535715	<b>Arsenal</b> 88.5302
Arsenal	2.445504	0.454846	<b>Man City</b> 87.1299
Newcastle	2.349245	0.969503	<b>Liverpool</b> 78.5916
Liverpool	2.334471	0.640827	<b>Newcastle</b> 65.7663
Chelsea	2.129188	0.978406	<b>Aston Villa</b> 61.6101
Aston Villa	2.097825	0.946424	<b>Chelsea</b> 61.0918
Tottenham	2.042436	0.944800	<b>Tottenham</b> 60.6101
West Ham	1.673433	1.133198	<b>Crystal Palace</b> 52.2456
Crystal Palace	1.568080	0.885325	<b>Man United</b> 52.1020
Man United	1.568080	0.885325	<b>Fulham</b> 49.3677
Brentford	1.549632	0.991616	<b>Brighton</b> 48.7735
Brighton	1.518039	0.944939	<b>Brentford</b> 47.8343
Fulham	1.516760	0.929661	<b>West Ham</b> 45.6413
Bournemouth	1.496679	1.020479	<b>Bournemouth</b> 45.5123
Luton	1.463380	1.293326	<b>Everton</b> 44.5823
Wolves	1.383217	0.986607	<b>Wolves</b> 44.0133
Nott'm Forest	1.357774	1.016179	<b>Nott'm Forest</b> 42.1188
Burnley	1.146215	1.175471	<b>Luton</b> 35.6009
Everton	1.093311	0.767354	<b>Burnley</b> 31.7280
Sheffield United	1.000000	1.560418	<b>Sheffield United</b> 19.1805

Une meilleure modélisation :

## Comparatif

	points		Points		Points
<b>Man City</b>	91	Arsenal	88.5302	Man City	73.3459
<b>Arsenal</b>	89	Man City	87.1299	Arsenal	70.6318
<b>Liverpool</b>	82	Liverpool	78.5916	Liverpool	67.9901
<b>Aston Villa</b>	68	Newcastle	65.7663	Newcastle	67.4121
<b>Tottenham</b>	66	Aston Villa	61.6101	Chelsea	62.7062
<b>Chelsea</b>	63	Chelsea	61.0918	Aston Villa	62.1597
<b>Newcastle</b>	60	Tottenham	60.6101	Tottenham	60.8343
<b>Man United</b>	60	Crystal Palace	52.2456	West Ham	51.7053
<b>West Ham</b>	52	Man United	52.1020	Crystal Palace	49.8482
<b>Crystal Palace</b>	49	Fulham	49.3677	Man United	49.7536
<b>Brighton</b>	48	Brighton	48.7735	Brentford	49.0098
<b>Everton</b>	48	Brentford	47.8343	Fulham	48.5034
<b>Bournemouth</b>	48	West Ham	45.6413	Brighton	48.4078
<b>Fulham</b>	47	Bournemouth	45.5123	Bournemouth	47.6998
<b>Wolves</b>	46	Everton	44.5823	Luton	46.2814
<b>Brentford</b>	39	Wolves	44.0133	Wolves	44.9190
<b>Nott'm Forest</b>	36	Nott'm Forest	42.1188	Nott'm Forest	44.2838
<b>Luton</b>	26	Luton	35.6009	Burnley	38.2936
<b>Burnley</b>	24	Burnley	31.7280	Everton	37.5610
<b>Sheffield United</b>	16	Sheffield United	19.1805	Sheffield United	33.6780

Classement réel

optimisé

initial

## Une meilleure modélisation :

## Un programme pratique

```
: def coeffs(matchs_vus, equipes):
    X= remplir_matrice_X(matchs_vus, equipes)
    Y= remplir_matrice_Y(matchs_vus, equipes)
    return calcul_coeffs(X,Y)

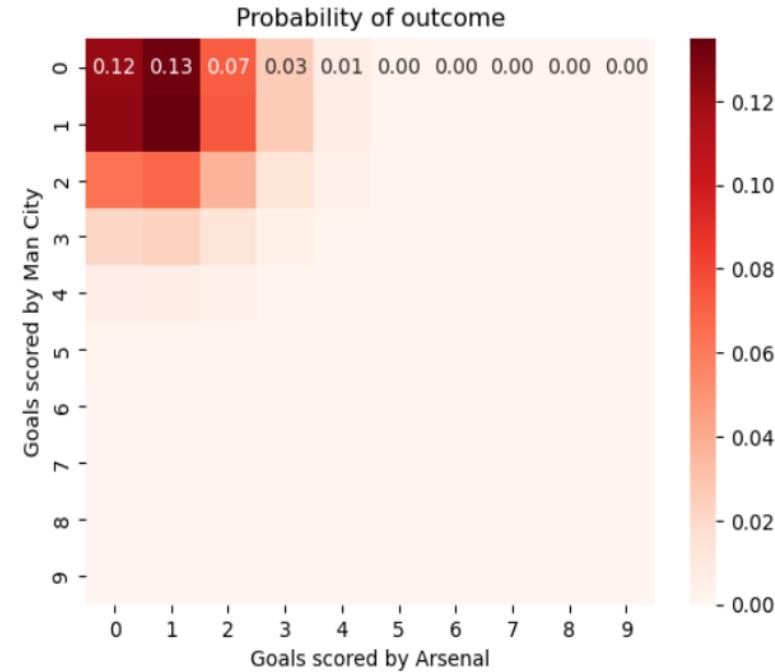
: def simu(data, equipe_dom, equipe_ext, n):
    equipes=sorted(set(data[ 'HomeTeam' ]))
    journée=n-1
    matchs_joués=10*journée
    matchs_vus = data.iloc[:matchs_joués]
    alpha, beta= coeffs(matchs_vus, equipes)
    alpha_beta= np.concatenate((alpha, beta), axis=1)
    gamm= gamma(matchs_vus)
    L, M=matrices(alpha_beta)
    i=equipes.index(equipe_dom)
    j=equipes.index(equipe_ext)
    lmda=L[i][j]*gamm
    mu=M[i][j]
    probas=proba(lmda,mu,10)
    seab.heatmap(probas, annot=True, fmt='.2f', cmap="Reds", cbar=True)

#configuration des labels
    plt.xlabel(f'Goals scored by {equipe_ext}')
    plt.ylabel(f'Goals scored by {equipe_dom}')
    plt.title('Probability of outcome')
#affichage de la heatmap
    plt.show()

equipe_dom=input('entrez l'équipe qui joue à domicile')
equipe_ext=input('entrez l'équipe qui joue à l'extérieur')
data=pd.read_csv(input('entrez un fichier csv contenant les données du championnat'))
journée=int(input('quelle journée voulez vous simuler'))
simu(data, equipe_dom, equipe_ext, journée)
```

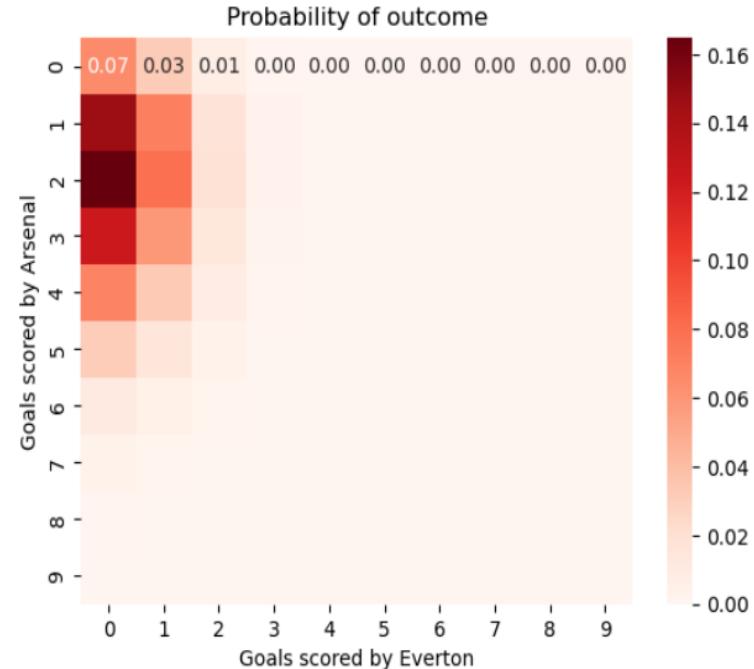
Une meilleure modélisation :

Un programme pratique



Man City - Arsenal

1      1



Arsenal - Everton

2      1

# Conclusion

## Positifs

- ✓ Les résultats sont concluants .
- ✓ Le programme est simple à utiliser .

# Conclusion

## Négatifs

Beaucoup de paramètres  
n'ont pas été pris en compte.

# Annexe

## Programme python 1ère simulation

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as seab
import math

#la matrice X dont le coefficient de la ième ligne et jème colonne contiendra le nombre de buts marqués par i contre j avec i jouant à domicile
#la ième équipe est l'équipe ayant l'indice i dans la liste équipe qui est ordonné selon l'ordre alphabétique
def remplir_matrice_X(data, equipes):
    nb_equipes = len(equipes)
    X = np.zeros((nb_equipes, nb_equipes))
    for index, row in data.iterrows():#.iterrows renvoie des tuples contenant l'indice d'une ligne et la ligne elle même qui est sous forme d'une sorte
        equipe_domicile = row['HomeTeam'] # le nom de l'équipe jouant à domicile
        equipe_exterieure = row['AwayTeam'] #celui de celle jouant à l'extérieur
        buts_domicile = row['FTHG']
        #on récupère les indices des équipes dans la liste équipes
        i = equipes.index(equipe_domicile)
        j = equipes.index(equipe_exterieure)
        X[i][j] = buts_domicile
    return X

#la matrice Y dont le coefficient de la ième ligne et jème colonne contiendra le nombre de buts marqués par j contre i avec j jouant à l'extérieur
def remplir_matrice_Y(data, equipes):
    nb_equipes = len(equipes)
    Y = np.zeros((nb_equipes, nb_equipes))
    for index, row in data.iterrows():
        equipe_domicile = row['HomeTeam']
        equipe_exterieure = row['AwayTeam']
        buts_domicile = row['FTAG']
        i = equipes.index(equipe_domicile)
        j = equipes.index(equipe_exterieure)
        Y[i][j] = buts_domicile
    return Y
```

# Annexe

## Programme python 1ère simulation

```
#on calcule le nombre total de buts marqués par la ième équipe
def somme_buts_marqués(X, Y, i):
    tot_buts = 0
    for j in range(len(X)):
        if j != i:
            tot_buts += X[i][j] + Y[j][i]
    return tot_buts

#calcule le nombre de buts marqués en moyenne par une équipe et qui est à la fois le paramètre de la loi de Poisson
def calcul_lambda(X,Y,n):
    lamda=np.zeros((n,1))
    for i in range(n):
        tot_buts_i=somme_buts_marqués(X,Y,i)
        tot_matchs= (n-1)*2
        lamda[i][0]=tot_buts_i/tot_matchs
    return lamda

#simule un match en faisant un tirage aléatoire des buts selon la loi de Poisson
def simulate_match_1(lamda,i, j):
    team1_goals = np.random.poisson(lamda[i][0])
    team2_goals = np.random.poisson(lamda[j][0])

    if team1_goals > team2_goals:
        result = 'Team 1 wins'
    elif team1_goals < team2_goals:
        result = 'Team 2 wins'
    else:
        result = 'Draw'

    return team1_goals, team2_goals, result
```

# Annexe

## Programme python 1ère simulation

```
#simulation du classement obtenu en moyenne à partir des données
def classement_1(lamda, equipes, n):
    num_simulations = n
    total_points = {team: 0 for team in equipes}#dictionnaire qui va contenir le nombre total de points de chaque équipe sur toutes les simulations
    points_esp = {team: 0 for team in equipes}#dictionnaire qui va contenir le nombre de points gagnés en moyenne

    for _ in range(num_simulations):
        points = {team: 0 for team in equipes}

        for i in range(len(equipes)):
            for j in range(len(equipes)):
                if i != j:
                    team1_goals, team2_goals, result = simulate_match_1(lamda, i, j)
                    if team1_goals > team2_goals:
                        points[equipes[i]] += 3
                    elif team1_goals < team2_goals:
                        points[equipes[j]] += 3
                    else:
                        points[equipes[i]] += 1
                        points[equipes[j]] += 1

        for team in equipes:
            total_points[team] += points[team]
    for team in equipes:
        points_esp[team]=total_points[team]/num_simulations

    # conversion du dictionnaire des points en moyenne en DataFrame pour un tri facile
    df_classement = pd.DataFrame.from_dict(points_esp, orient='index', columns=['Points'])
    df_classement = df_classement.sort_values(by='Points', ascending=False)

    return df_classement
```

## Annexe

### Programme python 1ère simulation

```
def simu_champ_1(data):
    equipes= sorted(set(data[ 'HomeTeam' ]))
    X=remplir_matrice_X(data,equipes)
    Y=remplir_matrice_Y(data, equipes)
    lamdas= calcul_lambda(X,Y,len(equipes))
    print('classement espéré:')
    classement_esp=classement_1(lamdas,equipes, 10000)
    print(classement_esp)
```

# Annexe

## Programme python 2ème simulation

```
#somme des buts encaissés par l'équipe i
def somme_buts_encaissés(X, Y, i):
    tot_buts = 0
    for j in range(len(X)):
        if j != i:
            tot_buts += X[j][i] + Y[i][j]
    return tot_buts
```

```
#somme des beta ou des alpha
def somme_coeffs(beta, i):
    tot_beta = 0
    for j in range(len(beta)):
        if j != i:
            tot_beta += beta[j]
    return 2 * tot_beta
```

```
#calcule des matrices colonnes contenant les capacités offensives et défensives de manière itérative
def calcul_coefficients(X, Y):
    nb_equipes = len(X)
    alpha = np.ones((nb_equipes, 1))
    beta = np.ones((nb_equipes, 1))
    for a in range(500):
        for i in range(nb_equipes):
            if i != 16:
                S2 = somme_coefficients(beta, i)
                S1 = somme_buts_marqués(X, Y, i)
                if S2 != 0:
                    alpha[i] = S1 / S2
                for i in range(nb_equipes):
                    S4 = somme_buts_encaissés(X, Y, i)
                    S3 = somme_coefficients(alpha, i)
                    if S3 != 0:
                        beta[i] = S4 / S3
    return alpha, beta
```

# Annexe

## Programme python 2ème simulation

```
#calcule du Lambda et du mu associés au match de i contre j à partir d'une matrice à 2 colonnes contenant les alpha et beta
def lamda(alpha_beta, i, j):
    alpha_i = alpha_beta[i][0]
    beta_j = alpha_beta[j][1]
    return alpha_i * beta_j

def mu(alpha_beta, i, j):
    beta_i = alpha_beta[i][1]
    alpha_j = alpha_beta[j][0]
    return alpha_j * beta_i
```

```
#on réunit tous les lambda et les mu dans respectivement une matrice L et une matrice M
def matrices(alpha_beta):
    L = np.zeros((20, 20))
    M = np.zeros((20, 20))
    for i in range(20):
        for j in range(20):
            if j != i:
                L[i][j] = lamda(alpha_beta, i, j)
                M[i][j] = mu(alpha_beta, i, j)
    return L, M
```

# Annexe

## Programme python 2ème simulation

```
#on simule le championnat grace à cette 1ère amélioration
def simulate_match_2(L, M, i, j):
    team1_goals = np.random.poisson(L[i][j])
    team2_goals = np.random.poisson(M[i][j])
    if team1_goals > team2_goals:
        result = 'Team 1 wins'
    elif team1_goals < team2_goals:
        result = 'Team 2 wins'
    else:
        result = 'Draw'
    return team1_goals, team2_goals, result

def classement_2(L, M, equipes, n):
    num_simulations = n
    total_points = {team: 0 for team in equipes}
    points_esp={teams: 0 for teams in equipes}
    for _ in range(num_simulations):
        points = {team: 0 for team in equipes}
        for i in range(len(equipes)):
            for j in range(len(equipes)):
                if i != j:
                    team1_goals, team2_goals, result = simulate_match_2(L, M, i, j)
                    if team1_goals > team2_goals:
                        points[equipes[i]] += 3
                    elif team1_goals < team2_goals:
                        points[equipes[j]] += 3
                    else:
                        points[equipes[i]] += 1
                        points[equipes[j]] += 1
        for team in equipes:
            total_points[team] += points[team]
    for team in equipes:
        points_esp[team]=total_points[team]/num_simulations

#conversion du dictionnaire des points en moyenne en DataFrame pour un tri facile
df_classement = pd.DataFrame.from_dict(points_esp, orient='index', columns=['Points'])
df_classement = df_classement.sort_values(by='Points', ascending=False)

return df_classement
```

# Annexe

## Programme python 2ème simulation

```
#simulation complète 2
def simu_champ_2(data):
    équipes = sorted(set(data['HomeTeam']))
    X = remplir_matrice_X(data, équipes)
    Y = remplir_matrice_Y(data, équipes)
    alpha, beta = calcul_coeffs(X, Y)
    alpha_beta = np.concatenate((alpha, beta), axis=1)
    L, M = matrices(alpha_beta)
    classement_final=classement_2(L, M, équipes, 10000)
    print('le classement espéré est')
    print(classement_final)
```

# Annexe

## Programme python 3ème simulation

```
#le gamma qui représente l'avantage de jouer à domicile
def gamma(data):
    matchs = sorted(list(zip(data['HomeTeam'], data['AwayTeam'], data['FTHG'], data['FTAG'])))
# Initialiser un dictionnaire pour stocker les buts marqués à domicile et à l'extérieur par chaque équipe
    buts_par_equipe = {}
    facteurs=[]

    for equipe_domicile, equipe_exterieur, buts_domicile, buts_exterieur in matchs:
        # Ajouter les buts marqués à domicile à l'équipe à domicile
        if equipe_domicile not in buts_par_equipe:
            buts_par_equipe[equipe_domicile] = {"domicile": 0, "exterieur": 0}
            buts_par_equipe[equipe_domicile]["domicile"] += buts_domicile
        # Ajouter les buts marqués à l'extérieur à l'équipe à l'extérieur
        if equipe_exterieur not in buts_par_equipe:
            buts_par_equipe[equipe_exterieur] = {"domicile": 0, "exterieur": 0}
            buts_par_equipe[equipe_exterieur]["exterieur"] += buts_exterieur
    for equipe, buts in buts_par_equipe.items():
        facteurs.append(buts['domicile']/buts['exterieur'])
    somme=0
    for i in range(20):
        somme+=facteurs[i]
    gamma=somme/20
    return gamma
```

# Annexe

## Programme python 3ème simulation

```
#simulation avec cette 2ème amélioration
def simulate_match_3(L, M, i, j, gamma):
    team1_goals = np.random.poisson(L[i][j] * gamma)
    team2_goals = np.random.poisson(M[i][j])
    if team1_goals > team2_goals:
        result = 'Team 1 wins'
    elif team1_goals < team2_goals:
        result = 'Team 2 wins'
    else:
        result = 'Draw'
    return team1_goals, team2_goals, result

def classement_3(L, M, data, equipes, n, gamma):
    num_simulations = n
    total_points = {team: 0 for team in equipes}
    points_esp = {team: 0 for team in equipes}
    for _ in range(n):
        points = {team: 0 for team in equipes}
        for index, row in data.iterrows():
            equipe_dom= row['HomeTeam']
            equipe_ext=row['AwayTeam']
            i= equipes.index(equipe_dom)
            j= equipes.index(equipe_ext)
            equipe_dom_buts, equipe_ext_buts, result = simulate_match_3(L, M, i, j, gamma)
            if equipe_dom_buts > equipe_ext_buts:
                points[equipes[i]] += 3
            elif equipe_dom_buts < equipe_ext_buts:
                points[equipes[j]] += 3
            else:
                points[equipes[i]] += 1
                points[equipes[j]] += 1
        for team in equipes:
            total_points[team] += points[team]
    for team in equipes:
        points_esp[team]=total_points[team]/num_simulations

    # Convertir le dictionnaire des points totaux en DataFrame pour un tri facile
    df_classement = pd.DataFrame.from_dict(points_esp, orient='index', columns=['Points'])
    df_classement = df_classement.sort_values(by='Points', ascending=False)

    return df_classement
```

# Annexe

## Programme python 3ème simulation

```
#simulation complète 3

def simu_champ_3(data):
    équipes = sorted(set(data['HomeTeam']))
    X = remplir_matrice_X(data, équipes)
    Y = remplir_matrice_Y(data, équipes)
    alpha, beta = calcul_coeffs(X, Y)
    alpha_beta = np.concatenate((alpha, beta), axis=1)
    gamma_val = gamma(data)
    L, M = matrices(alpha_beta)
    classement_final=classement_3(L, M, data, équipes, 10000, gamma_val)
    print('le classement espéré est')
    print(classement_final)
```

# Annexe

## Programme python simulation match

```
#loi de poisson
def poisson(lamda,k):
    return (math.exp(-lamda)*((lamda)**k))/math.factorial(k)
```

```
#matrice cintenant les différentes probabilités de résultat d'un match donné
def proba(lamda,mu,n):
    probas=np.zeros((10,10))
    for i in range(10):
        for j in range(10):
            probas[i][j]=poisson(lamda,i)*poisson(mu,j)
    return probas
```

```
#calcul des capacités offensives et défensives après une certaine journée de championnat
def coeffs(data, journée, équipes):
    matchs_joués=0
    matchs_joués+=10*journée
    matchs_vus=data.iloc[:matchs_joués]
    X= remplir_matrice_X(matchs_vus, équipes)
    Y= remplir_matrice_Y(matchs_vus, équipes)
    return calcul_coeffs(X,Y)
```

# Annexe

## Programme python simulation match

```
#résultats probables d'un match à partie des matchs précédents
def simu(data, equipe_dom, equipe_ext, n):
    équipes=sorted(set(data['HomeTeam']))
    journée=n-1
    alpha, beta= coeffs(data,journée, équipes)
    alpha_beta = np.concatenate((alpha, beta), axis=1)
    gamm= gamma(data)
    L, M=matrices(alpha_beta)
    i=équipes.index(equipe_dom)
    j=équipes.index(equipe_ext)
    lmda=L[i][j]*gamm
    mu=M[i][j]
    probas=proba(lmda,mu,10)
    seab.heatmap(probas, fmt=' .2f ', cmap="Reds", cbar=True)

#configuration des labels
plt.xlabel(f'Goals scored by {equipe_ext}')
plt.ylabel(f'Goals scored by {equipe_dom}')
plt.title('Probability of outcome')
#affichage de la heatmap
plt.show()

équipe_dom=input('entrez l'équipe qui joue à domicile')
équipe_ext=input('entrez l'équipe qui joue à l'extérieur')
data=pd.read_csv(input('entrez un fichier csv contenant les données du championnat'))
journée=int(input('quelle journée voulez vous simuler'))
simu(data, équipe_dom, équipe_ext, journée)
```

# Annexe

## Programme python figure 1

```
# Définir la plage de valeurs pour x
x = np.linspace(1, 10, 1000)
# Calculer les valeurs de ln(x)
y1 = np.zeros((1000,1))
y2= np.zeros((1000,1))
y3= np.zeros((1000,1))
y4 = np.zeros((1000,1))
y5=np.zeros((1000,1))
for i in range(1000):
    y1[i]=np.log(f1(x[i]))
    y2[i]=np.log(f2(x[i]))
    y3[i]=np.log(f3(x[i]))
    y4[i]=np.log(f4(x[i]))
    y5[i]=np.log(f1(x[i])*f2(x[i])*f3(x[i])*f4(x[i]))

plt.plot(x, y1, 'r', label='ln(f1(x))')
plt.plot(x, y2, 'g', label='ln(f2(x))')
plt.plot(x, y3,'b', label='ln(f3(x))')
plt.plot(x, y4, 'y',label='ln(f4(x))')
plt.plot(x,y5, 'c', label='ln(f5(x))')

# Ajouter des titres et des Légendes
plt.xlabel('x')
plt.ylabel('ln(f(x))')
plt.legend()

# Afficher le graphique
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
import math
def f1(x):
    exp=math.exp(-x)
    fact=math.factorial(5)
    return ((x**5)*exp)/fact

def f2(x):
    exp=math.exp(-x)
    fact=math.factorial(10)
    return ((x**10)*exp)/fact

def f3(x):
    exp=math.exp(-x)
    fact=math.factorial(50)
    return ((x**50)*exp)/fact

def f4(x):
    exp=math.exp(-x)
    fact=math.factorial(100)
    return ((x**100)*exp)/fact
```

# Annexe

## Programme python pour figure 2

```
def calcul_coeffs(X, Y):  
    nb_equipes = len(X)  
    alpha = np.ones((nb_equipes, 1))  
    beta = np.ones((nb_equipes, 1))  
    x=np.zeros(501)  
    y=np.zeros(501)  
    y[0]=1  
    for a in range(500):  
        for i in range(nb_equipes):  
            if i!=16:  
                S2 = somme_coeffs(beta, i)  
                S1 = somme_buts_marqués(X, Y, i)  
                if S2 != 0:  
                    alpha[i] = S1 / S2  
                y[a+1] = alpha[0][0]  
                x[a+1]=a  
        for i in range(nb_equipes):  
            S4 = somme_buts_encaissés(X, Y, i)  
            S3 = somme_coeffs(alpha, i)  
            if S3 != 0:  
                beta[i] = S4 / S3  
    return alpha, beta, x, y  
  
alpha, beta, x, y= calcul_coeffs_test(X,Y)
```

```
plt.figure(figsize=(10,6))  
plt.plot(x,y)  
plt.show()
```

# Annexe

## Obtention des équations

$$F(\alpha, \beta) = \prod_{\substack{1 \leq i, j \leq n \\ i \neq j}} \frac{\lambda_{ij}^{x_{ij}} e^{-\lambda_{ij}}}{x_{ij}!} \times \frac{\mu_{ij}^{p_{ij}} e^{-\mu_{ij}}}{y_{ij}!}$$

$$\ln(F(\alpha, \beta)) = \sum_{\substack{1 \leq i, j \leq n \\ i \neq j}} x_{ij} \ln(\lambda_{ij}) - \lambda_{ij} + y_{ij} \ln(\mu_{jj}) - \mu_{ij} - \ln(x_{ij}! y_{ij}!)$$

$$\begin{aligned}\frac{\partial \lambda_{ij}}{\partial \alpha_k} &= \begin{cases} 0, & \text{si } i \neq k \\ \beta_j, & \text{si } i = k \end{cases} \\ \frac{\partial \mu_{ij}}{\partial \alpha_k} &= \begin{cases} 0, & \text{si } j \neq k \\ \beta_i, & \text{si } j = k \end{cases}\end{aligned}$$

$$\frac{\partial \ln(F)}{\partial \alpha_k} = \sum_{\substack{j=1 \\ j \neq k}}^n (x_{kj} + y_{jk}) \times \frac{1}{\alpha_k} - 2\beta_i$$

## Annexe

### Obtention des équations

$\frac{\partial(\ln F)}{\partial \alpha_i} = 0$ , pour  $i = 1, \dots, n$  donne

$$\alpha_i = \frac{\sum_{j \neq i} (x_{i,j} + y_{j,i})}{2 \sum_{j \neq i} \beta_j},$$

De même,  $\frac{\partial(\ln F)}{\partial \beta_i} = 0$ , pour  $i = 1, \dots, n$  donne

$$\beta_i = \frac{\sum_{j \neq i} (x_{j,i} + y_{i,j})}{2 \sum_{j \neq i} \alpha_j},$$

## Annexe

Démonstration du théorème

$$\begin{aligned} P(X=k) &= C_n^k p^k (1-p)^{n-k} \\ &= \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k} \\ &= \frac{n(n-1)\dots(n-k+1)}{k!} p^k (1-p)^{n-k} \\ &\simeq \frac{n^k}{k!} p^k e^{(n-k)\ln(1-p)} \\ &\simeq \frac{(np)^k}{k!} e^{-np} \end{aligned}$$