Chaîne de Lettres

Sommaire

Chaîne de Lettres

Modèle Informatique

Variables essentielles

Acheteurs potentiels : $\boldsymbol{P_{tot}}$

Génération : $\boldsymbol{g}$

Acheteurs à la génération $\boldsymbol{g}$: $\boldsymbol{P(g)}$

Nombre d'exemplaires de la lettre à la génération $\boldsymbol{g}$: $\boldsymbol{Z_g}$

$$\mathbb{P}_{\geq 1}(g) = \mathbb{P}_{\geq 1}(0)\frac{P(g)}{P_{tot}}$$

$$\mathbb{P}_{\geq 2}(g) = \mathbb{P}_{\geq 1}(g)^2$$

$$\mathbb{P}_0(g) = 1 - \mathbb{P}_{\geq 1}(g)$$

$$\mathbb{P}_2(g) = \mathbb{P}_{\geq 2}(g)$$

$$\mathbb{P}_1(g) = 1 - \mathbb{P}_2(g) - \mathbb{P}_0(g)$$

Chaîne de Lettres

Explication de Code

```
import random


def sold(p1, p2):
    p0 = 1-p1-p2
    return(random.choices((0,1,2),(p0,p1,p2))[0])
```

On note **p1** et **p2** les probabilités de vendre 1 ou 2 lettres, ce programme renvoie soit 0, soit 1 soit 2 avec les probabilités correspondantes

```
def update_letter(pop, i):
    letter = pop[i][2]
    first = letter[0]
    if first is not None:
        pop[first][1] += 50
    return(letter[1::]+[i])
```

La liste **letter** de 12 indices correspondant aux précédents acheteurs de la lettre
Les 12 noms qui étaient à l'origine sur la liste sont remplacés par **None**

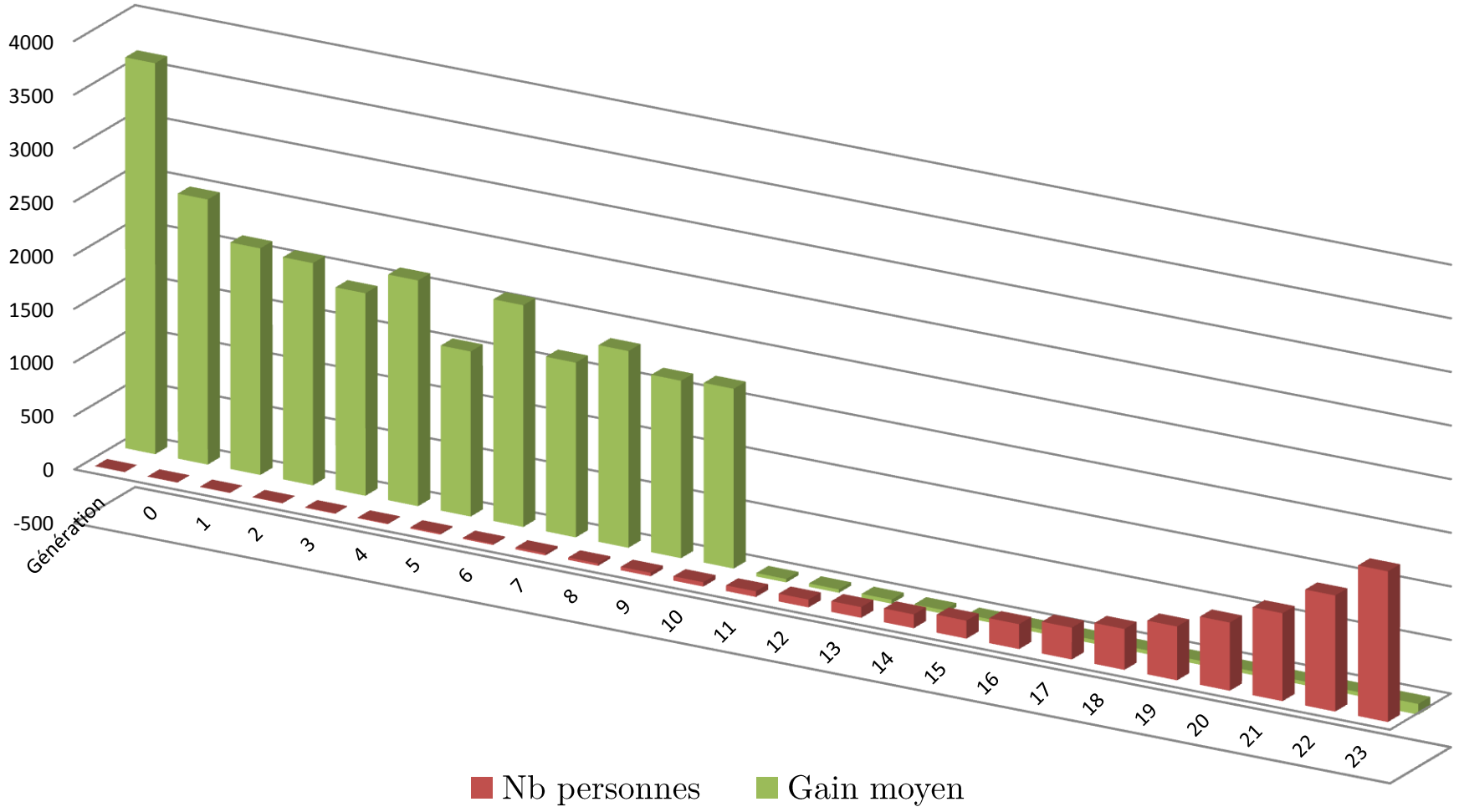
```
def pop_gen2(p1, p2, population):
    g = 0
    pop = [[g, -100, [None]*12]]
    indice = 0
    while len(pop)-indice > 0:
        g += 1
        l = len(pop)
        pop_dispo = population-l
        p1_g = p1*(pop_dispo)/population
        p2_g = p2*(pop_dispo)/population
        for i in range(indice,l):
            s = sold(p1_g, p2_g)
            pop[i][1] += 50*s
            for j in range(s):
                pop.extend([[g, -100, update_letter(pop, i)]])
        indice = l
    return(pop)
```

On note **p1_g** et **p2_g** les probabilités à la génération **g** de vendre respectivement 1 et 2 lettres.

Chaîne de Lettres

Exploitation de la Simulation

Majorité


4000
3500
3000
2500
2000
1500
1000
500
0
-500
Génération
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
Nb personnes
Gain moyen

Affichage

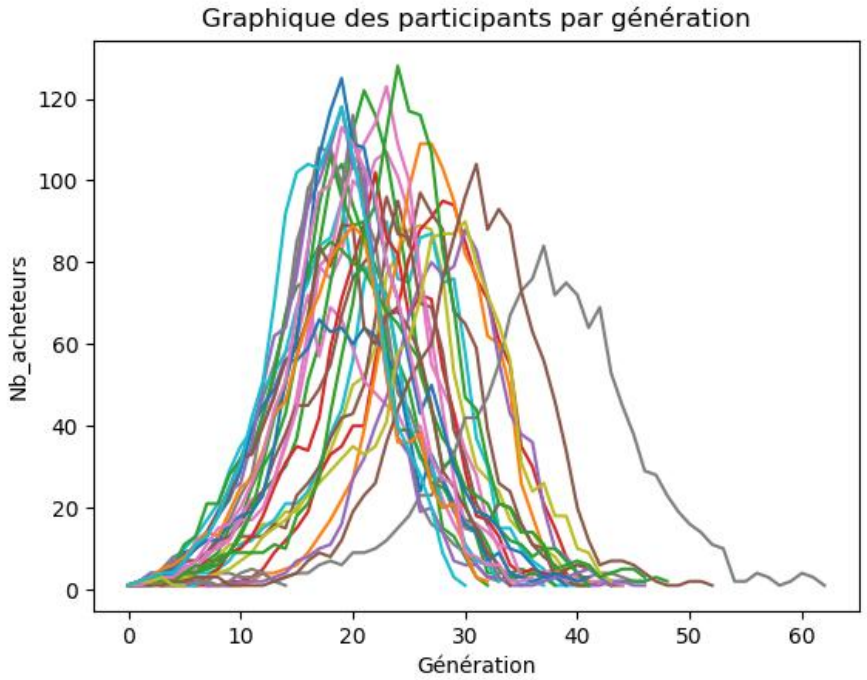
```
>>> t = pop_gen2(.5, .4, 3000)

>>> avg_display(t)
0 1 -50
1 1 -100
```

```
>>> t = pop_gen2(.5, .4, 3000)

>>> avg_display(t)
0 1 1750
1 1 2350
2 1 3350
3 2 2050
4 3 1683
5 3 2016
6 3 2166
7 5 1150
8 7 678
9 11 318
10 13 250
11 20 102
12 25 48
13 36 1
14 48 -15
```

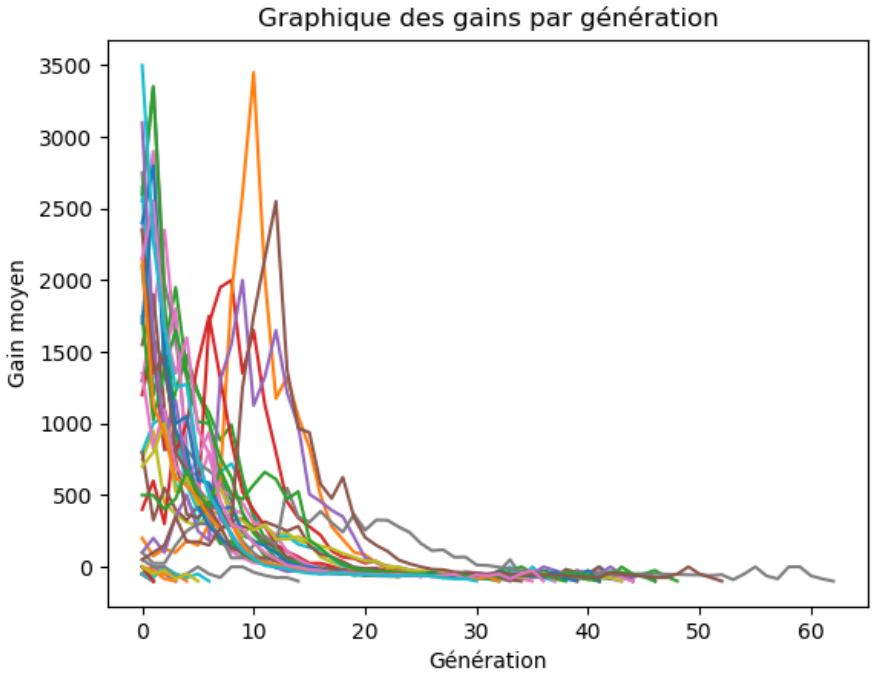
Gen	Pop	$
15	67	-27
16	83	-32
17	104	-38
18	124	-46
19	131	-54
20	118	-59
21	98	-60
22	79	-56
23	71	-61
24	56	-67
25	38	-68
26	25	-70
27	15	-50
28	15	-70
29	9	-73
30	5	-70
31	3	-17
32	5	-90
33	1	-100

Sur deux simulations on affiche la moyenne des gains en fonction de la génération

```
>>> coord = coord_gen2(50, .5, .4, 3000)
>>> coord_plot(coord, 1)
>>> plot_display(1)
```

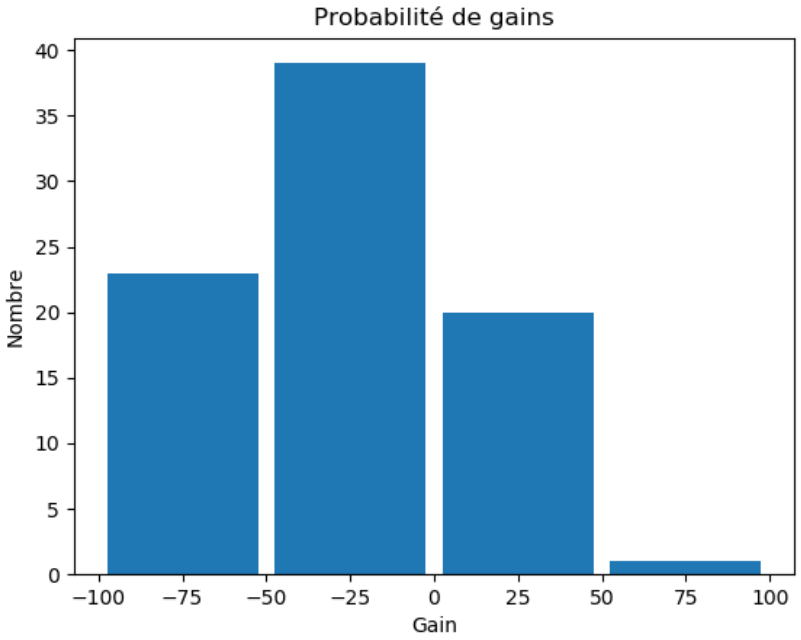



```
>>> coord_plot(coord, 2)
>>> plot_display(2)
```



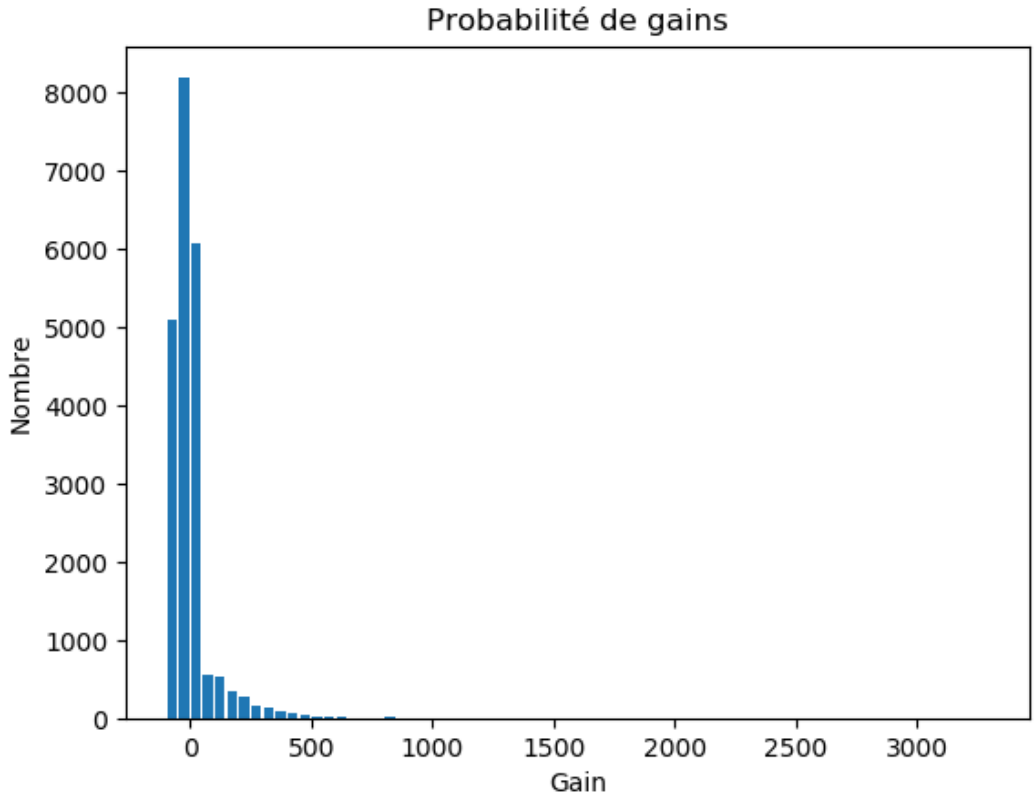
Distribution

```
>>> h = histo_data(.5, .4, 3000)
>>> histo_bar(h, 17)
>>> histo_display()
```



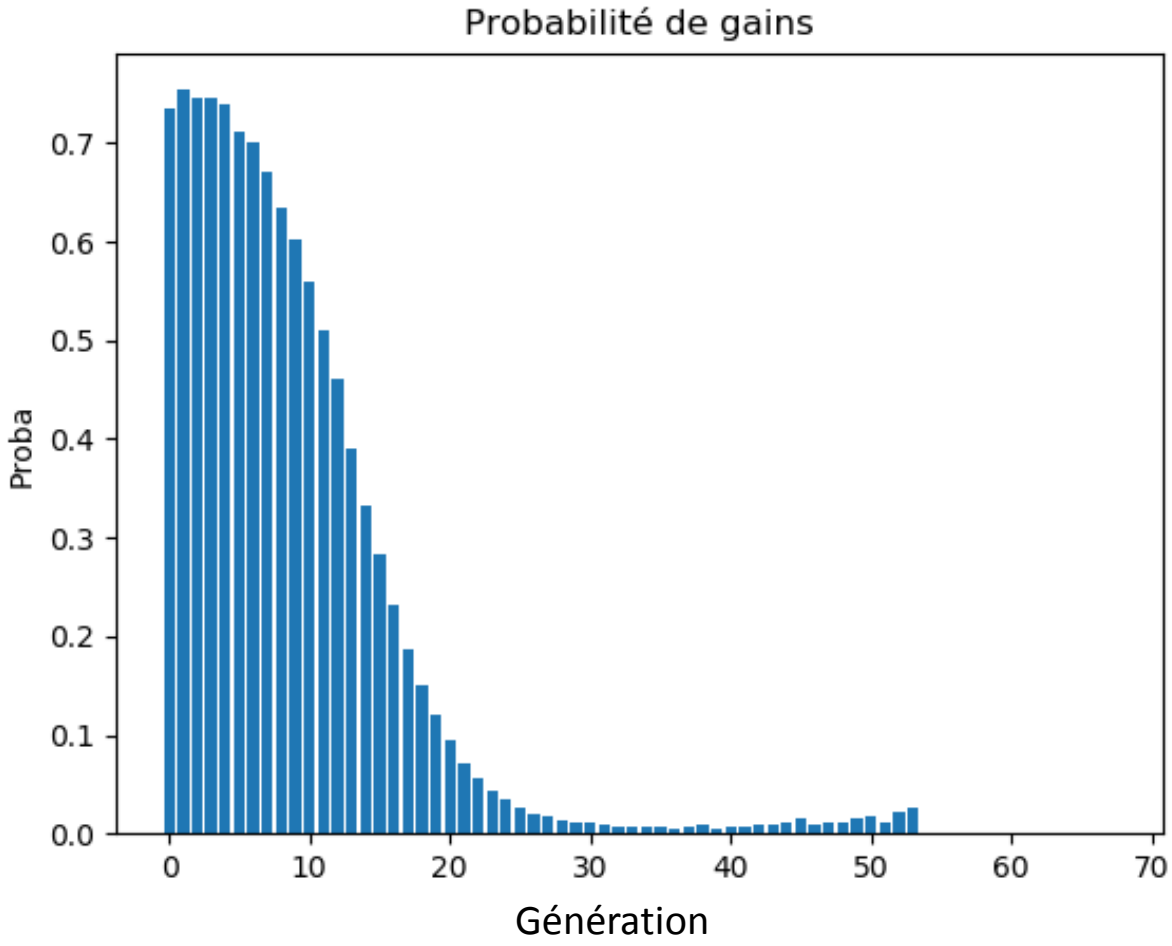
Les données de la génération **17** pour une simulation de paramètres
p1 = .5, p2 = .4, pop = 3000

```
>>> prob_bar(500, .5, .4, 3000, 17)
>>> histo_display()
```

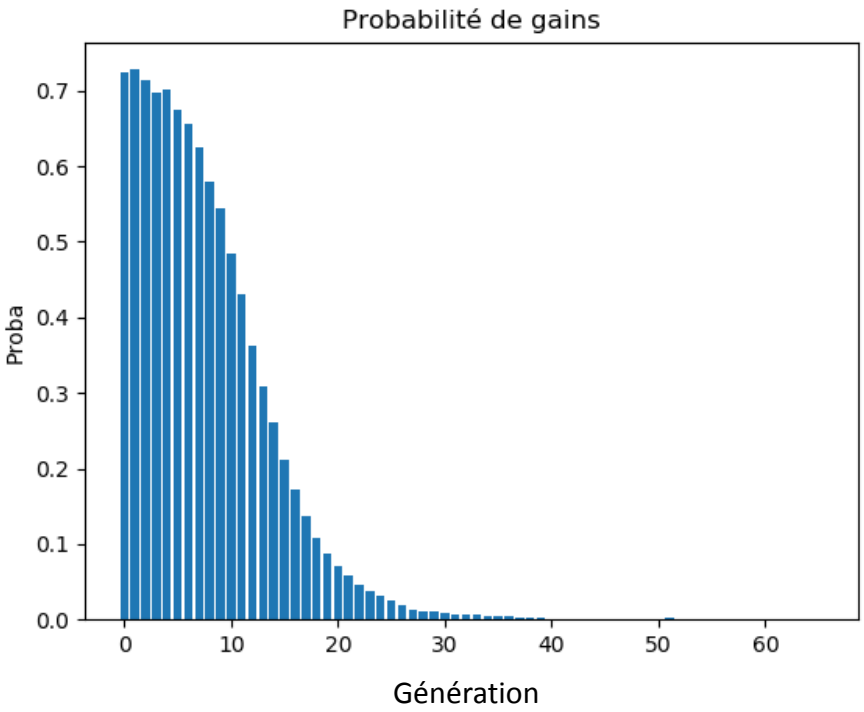


On moyenne les données de la génération **17** sur **500** simulations pour les paramètres **p1 = .5, p2 = .4, pop = 3000**

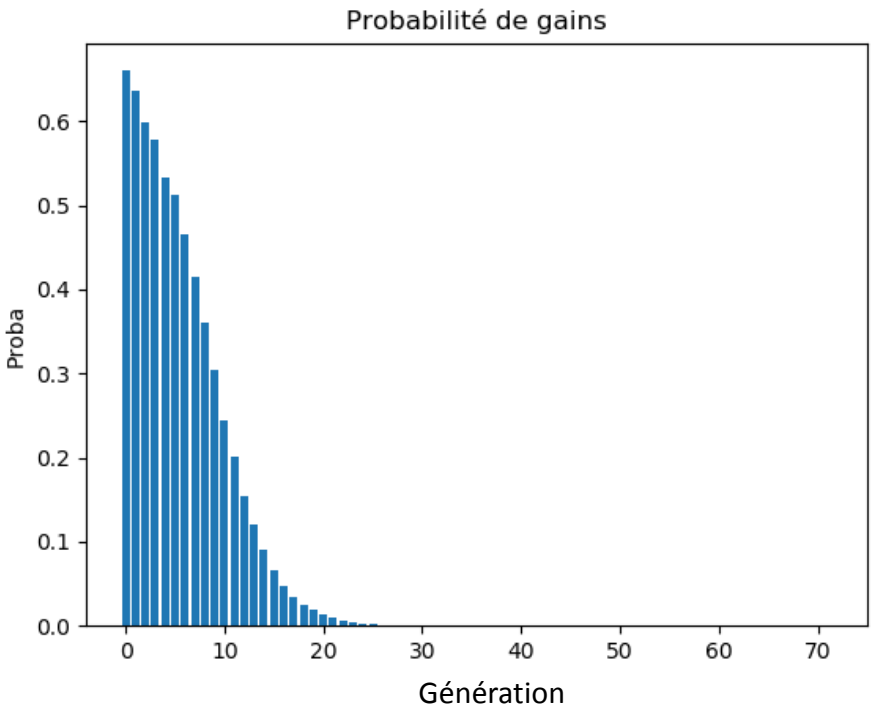
```
>>> dat = prob_gain(500, 0, .5, .4, 3000)

>>> bar_display(dat)
```



La probabilité de gagner plus que **0$** en fonction de la génération, moyennée sur **500** simulations



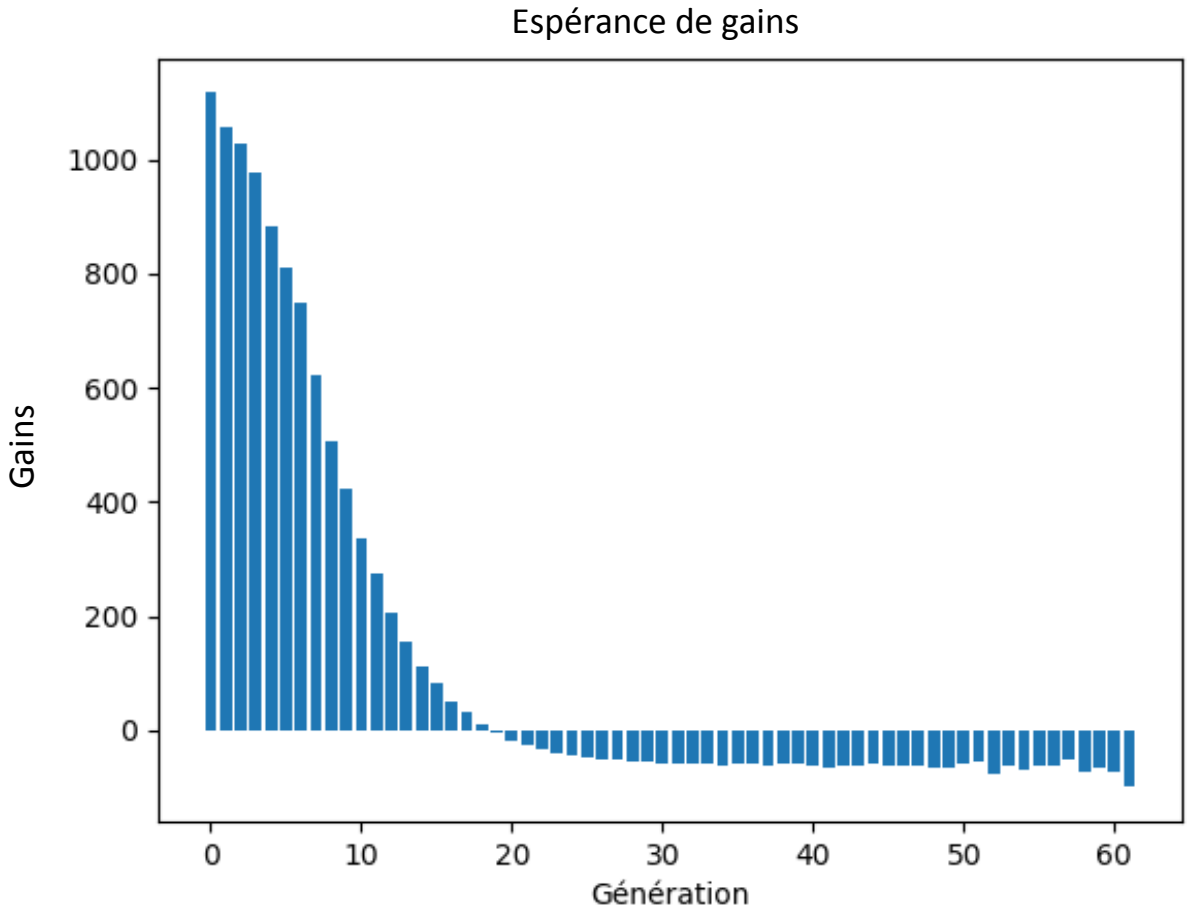
Pour un gain strictement supérieur à 100$



Pour un gain strictement supérieur à 500$

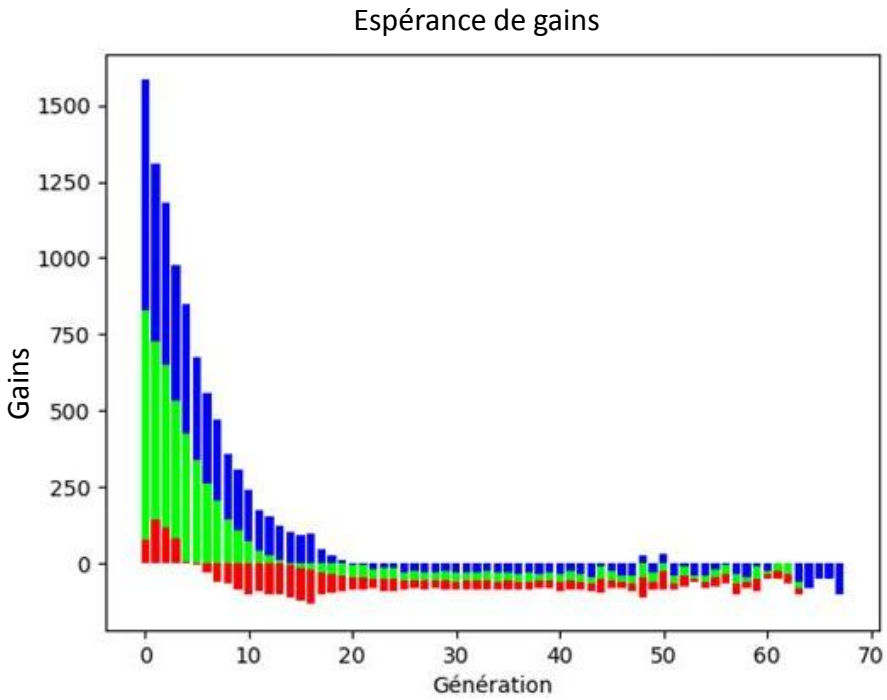
Moyenne

```
>>> dat = expect_val(500, .5, .4, 3000)
>>> val_display(dat)
```

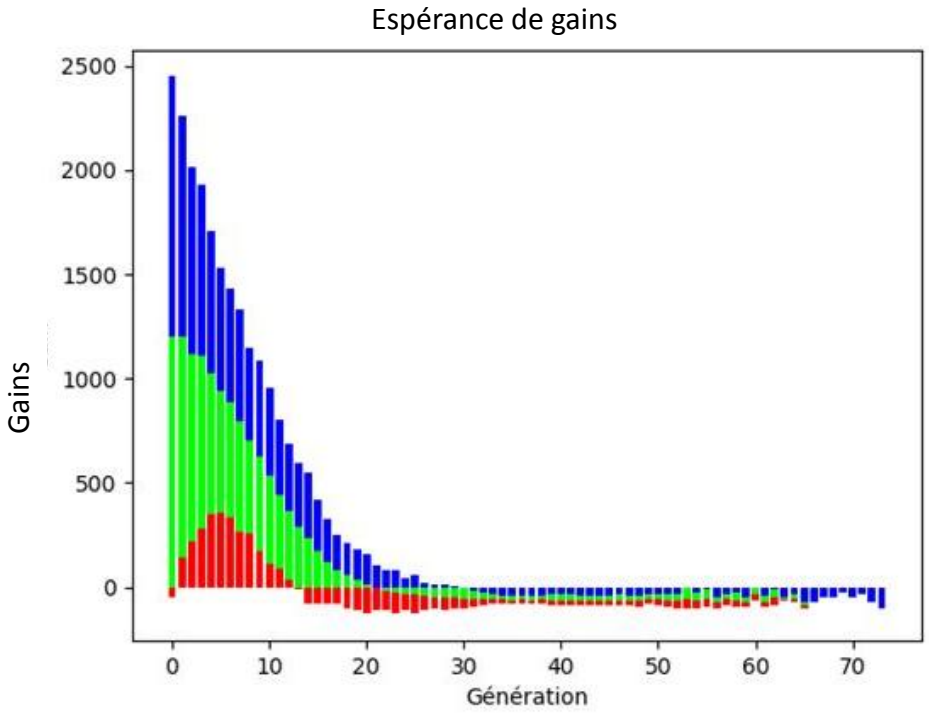


L'espérance de gains moyennée sur **500** simulations

Écart type

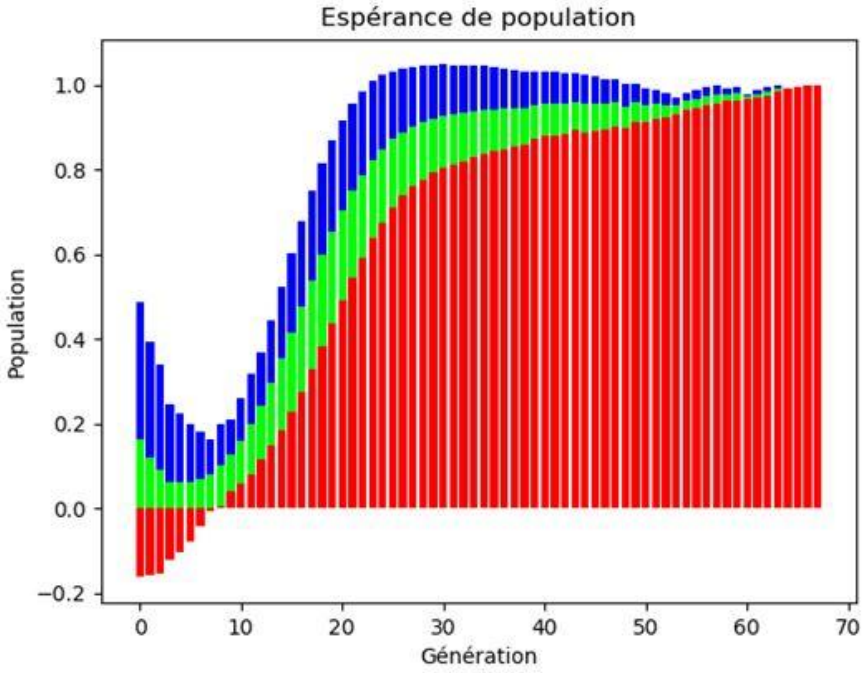


1000 personnes

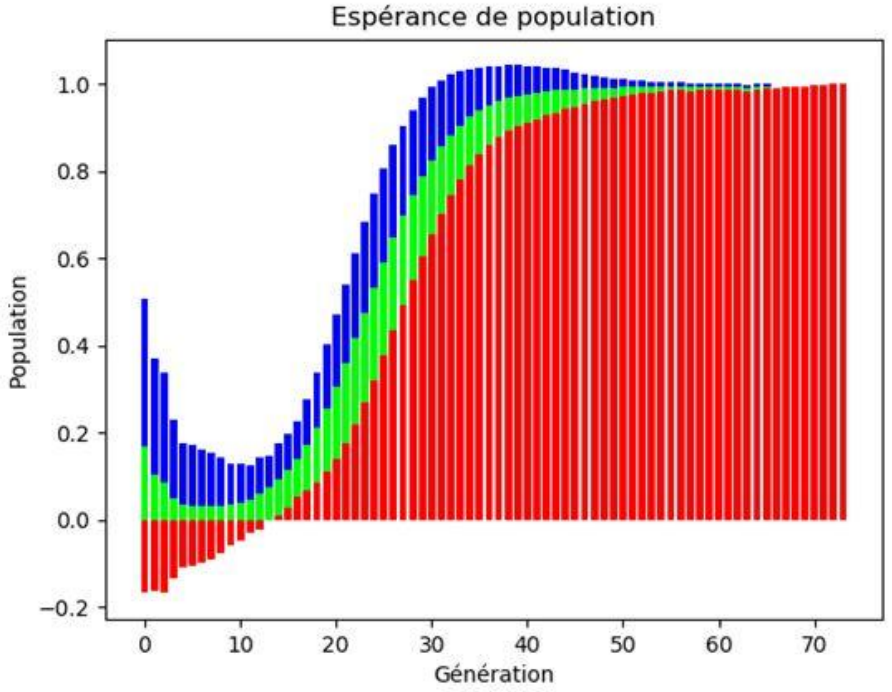


5000 personnes

On fait apparaitre l'écart type sur les courbes de population



1000 personnes



5000 personnes

On obtient un seuil $\frac{k}{N}$ de 0.36

1. L'évolution de la simulation pourrait se faire en fonction du temps plutôt que de la génération.
2. Il y a un « effet de mode » initial et on peut simuler un taux de confiance de la population en fonction des gains et des pertes des premiers participants.
3. On peut permettre aux participants d'acheter des lettres plusieurs fois, où d'acheter plusieurs lettres à la fois.
4. Les probabilités de ventes sont inhomogènes car certains habitants connaissent plus de monde.
5. Tenir compte de l'augmentation du nombre de participants potentiels en modélisant la façon dont la lettre peut atteindre de nouvelles villes.

Chaîne de Lettres

Processus de Branchement

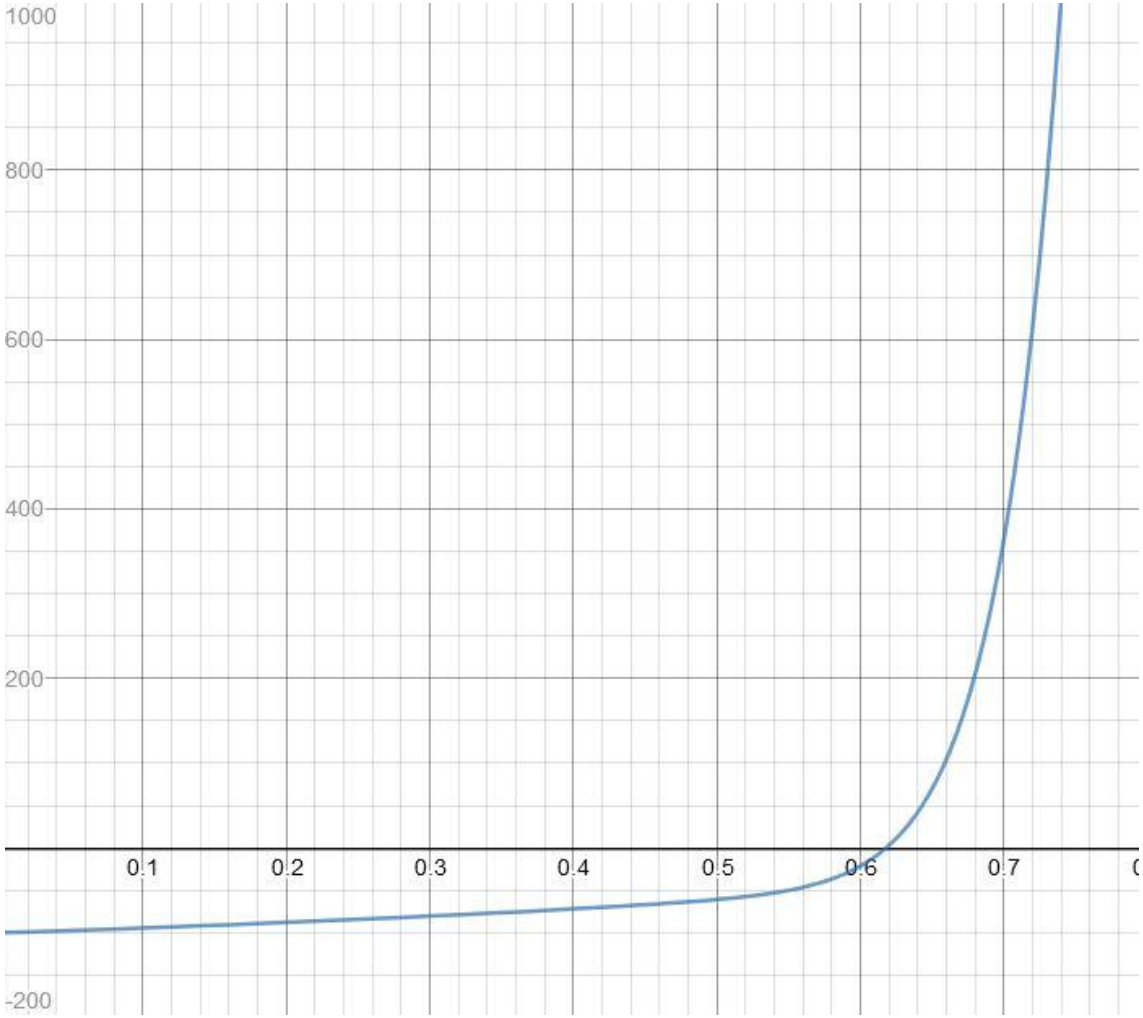
Approximation à l'origine :

$$P(g) \sim P_{tot}$$
$$\mathbb{P}_1(g) = \mathbb{P}_1(0) \equiv p_1$$
$$\mathbb{E} = \mathbb{E}(Z_1) + \mathbb{E}(Z_{12}) - 100 = 50(m + m^{12} - 2)$$
$$m = \mathbb{P}_1(g) + 2 * \mathbb{P}_2(g) = \mathbb{P}_{\geq 1}(g) + \mathbb{P}_{\geq 2}(g) = p_1(1 + p_1)$$

Condition de gain :

$$\mathbb{E} > 0 \iff m > 1 \iff p_1 > \frac{\sqrt{5} - 1}{2} \sim 0.61$$

$$\mathbb{E} = f(p_1)$$



Espérance à l'origine en fonction de la probabilité de vendre une lettre

Fonction génératrice des probabilités :

$$F_X(t) = \mathbb{E}(t^X) = \sum_{k=0}^{+\infty} \mathbb{P}(X = k)t^k$$

Lemme :

Si (Z_n) est un processus de branchement alors on a pour $n \in \mathbb{N}$

$$F_{Z_n}(t) = F_{Z_1}^{\circ(n)}(t)$$

Calcul de l'itéré de $F_{Z_1} = p_2 t^2 + p_1 t + (1 - p_1 - p_2)$

Calcul explicite de $F_{Z_1}^{\circ 3}(0)$ pour **p1=0.4, p2=0.5**

```
>>> Fn(.4,.5,3,0)
0.16851249999999995
```

Calcul de l'itéré de $F_{Z_1} = p_2 t^2 + p_1 t + (1 - p_1 - p_2)$

Calcul des coefficients de $F_{Z_1}^{\circ 3}$ pour **p1=0.4, p2=0.5**

```
>>> P = [0.1,0.4,0.5]

>>> [round(e,10) for e in Iteres(P,3)[-1]]
[0.1685125, 0.109, 0.19985, 0.175, 0.162575, 0.091, 0.06125, 0.025,
0.0078125]
```

Calcul de l'itéré de $F_{Z_1} = p_2 t^2 + p_1 t + (1 - p_1 - p_2)$

Calcul formel des coefficients de $F_{Z_1}^{\circ 2}$

```
>>> F = [1-p1-p2,p1,p2]

>>> [simplify(e) for e in Iteres(F,2)[-1]]
[-p1**2 - p1*p2 + p2*(p1 + p2 - 1)**2 - p2 + 1, p1*(p1 - 2*p2*(p1 +
p2 - 1)), p2*(p1**2 + p1 - 2*p2*(p1 + p2 - 1)), 2*p1*p2**2, p2**3]
```

Calcul des probabilités :

$$F_{Z_n}(X) = \sum_{k=0}^{+\infty} \mathbb{P}(Z_n = k) X^k$$

$\mathbb{P}(Z_n = k)$ vaut

```
>>> simplify(Iteres(F,"n")[-1]["k"])
```

Probabilité d'extinction :

$F_{Z_n}(0) = \mathbb{P}(Z_n = 0)$ qui vaut pour $n = 3$

```
>>> simplify(Iteres(F,3)[-1][0])
-2*p1*p2**2*(p1 + p2 - 1)**3 - p1*(p1 - 2*p2*(p1 + p2 - 1))*(p1 + p2
- 1) - p1*(p1 + p2 - 1) - p1 + p2**3*(p1 + p2 - 1)**4 + p2*(p1 + p2
- 1)**2*(p1**2 + p1 - 2*p2*(p1 + p2 - 1)) + p2*(p1 + p2 - 1)**2 - p2
+ 1
```

Probabilité d'extinction :

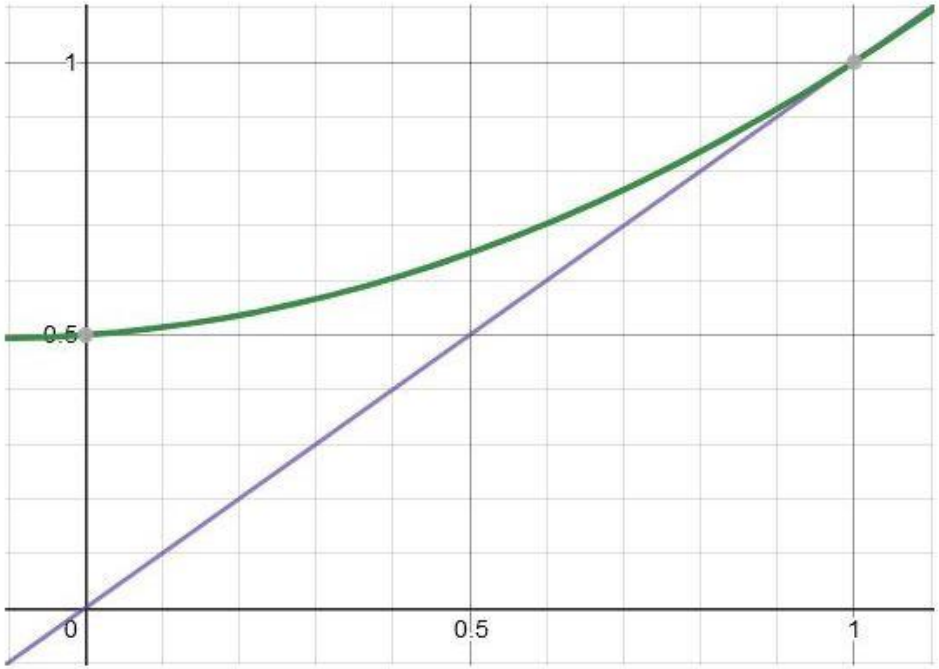
$$F_{Z_n}(0) \xrightarrow[n\to+\infty]{} \min_{[0,1]}\{x, \qquad F_{Z_1}(x) = x\}$$

Point fixe

$$\mathbb{E}(Z_1) = m, \qquad \mathbb{V}(Z_1) = \sigma^2$$

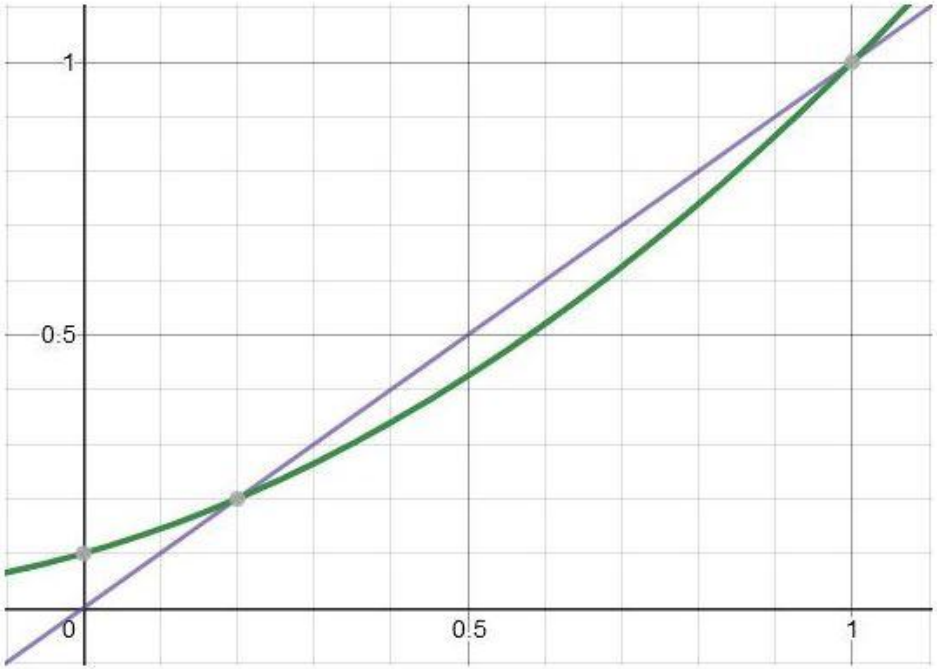
$F'_{Z_1}(1) = m \leq 1$, F'_{Z_1} croissante

$\Rightarrow l = 1$



$F'_{Z_1}(1) = m > 1, \qquad F_{Z_1}(0) = p_0 > 0$

$\Rightarrow l < 1$



Espérance :

$$\mathbb{E}(Z_n) = m^n$$

Variance :

$$\mathbb{V}(Z_n) \underset{\substack{m\neq 1\\ m=1}}{=} \begin{cases} m^{n-1}\sigma^2\left(\dfrac{1-m^n}{1-m}\right) \\ \sigma^2 n \end{cases}$$

Lemme :

1) Si $m > 1$ alors $\mathbb{P}(Z_n > 0) \underset{+\infty}{\longrightarrow} 1 - l$

2) Si $m < 1$ alors $\mathbb{P}(Z_n > 0) \sim Cm^n$

3) Si $m = 1$ alors $\mathbb{P}(Z_n > 0) \sim \frac{2}{\sigma^2 n}$

Temps d'arrêt :

$$K = \min\{k, \quad Z_k = 0\}$$

Espérance dans le cas sous-critique :

$$\mathbb{E}(K) = \sum_{k=1}^{+\infty} \mathbb{P}(Z_k > 0) = \frac{C}{1 - \mathbb{E}(Z_1)}$$

Chaîne de Lettres

Vente pyramidale Avec quota

Variables essentielles

Coût d'entrée : $\boldsymbol{c}$

Quota : $\boldsymbol{N}$

Nombre de personnes recrutées : $\boldsymbol{R}$

Gain par recrutement : $\boldsymbol{d}$

Rang d'entrée : $\boldsymbol{k}$

Condition :

$$\mathbb{E}(R)d > c$$

$$\mathbb{E}(R) > \frac{c}{d}$$

Rang d'entrée

$$X_i \sim \mathcal{B}\left(\frac{1}{i}\right)$$

$$R = \sum_{i=k}^{N-1} X_i$$

$$\mathbb{E}(R) \underset{N\to+\infty}{\sim} \ln\left(\frac{N}{k}\right)$$

Ne dépend que du quotient $\frac{N}{k}$

Condition :

$$\frac{N}{k} > \exp\left(\frac{c}{d}\right)$$

Seuil :

$$\frac{k}{N} < exp\left(-\frac{c}{d}\right) \underbrace{=}_{c=d} e^{-1} \cong 0.368$$

Espérance à l'origine :

$$\mathbb{E} = d\mathbb{E}(R) - c$$

$$\mathbb{E} \underset{N\to+\infty}{\sim} d\ln(N)$$

Probabilité de perte totale :

$$\mathbb{P}_k(0) = \prod_{i=k}^{N-1}(1 - 1/i) = \frac{(k-1)}{(N-1)}$$

Nombre de participants qui perdent tout :

$$\sum_{k=1}^{N} \mathbb{P}_k(0) = \frac{1}{N-1}(1 + \cdots + N - 1) = \frac{N}{2}$$

Lemme :

La proportion de participants qui recrutent exactement r personnes tend vers $2^{-(r+1)}$ quand $N \to +\infty$

Chaîne de Lettres

Chaîne de lettres Avec plusieurs Participations

Variables essentielles

Quota : N

Nombre de participants : k

Participants de chaque type : (U_k, V_k, W_k)

Temps d'arrêt de la chaîne : K

$$(U_0, V_0, W_0) = (u_0, 0, 0)$$

Répartition selon le nombre de lettres vendues :

$$(U_k, V_k, W_k) = (U_k, k + 2u_0 - 2U_k, U_k - u_0)$$

Règle de fonctionnement de la chaîne de Markov :

$$\mathbb{P}(U_{k+1} = u | U_k = u) = 1 - \mathbb{P}(U_{k+1} = u + 1 | U_k = u)$$

$$= 1 - \frac{V_k}{U_k + V_k} = \frac{u}{k + 2u_0 - u}$$

Effondrement :

$$k < 2N - 2u_0$$

Temps d'arrêt :

$$K = \min\{k, \qquad U_k + V_k = N\}$$

Équilibre :

$$\mathbb{P}(U_{k+1} = U_k + 1) = \frac{V_k}{V_k + U_k} \xrightarrow[k \to +\infty]{} \alpha$$

$$\frac{U_k}{k} \xrightarrow[k \to +\infty]{} \alpha$$

$$\frac{V_k}{U_k + V_k} \sim \frac{\left(1 - \frac{2U_k}{k}\right)}{1 - \frac{U_k}{k}}$$

$$\alpha = \frac{(1 - 2\alpha)}{1 - \alpha}$$

$$\alpha = \frac{\left(3 - \sqrt{5}\right)}{2} \cong 0.382$$

Contraintes :

$$\begin{cases} U_K = K\alpha \\ U_K + V_K + W_K = K \\ 2W_K + V_K = K - 1 \approx K \end{cases}$$

Proportions :

$$(0.382, 0.236, 0.382)$$

$$U_K + V_K = N \Rightarrow K = \frac{N}{(0.382 + 0.236)} = 1.62N$$

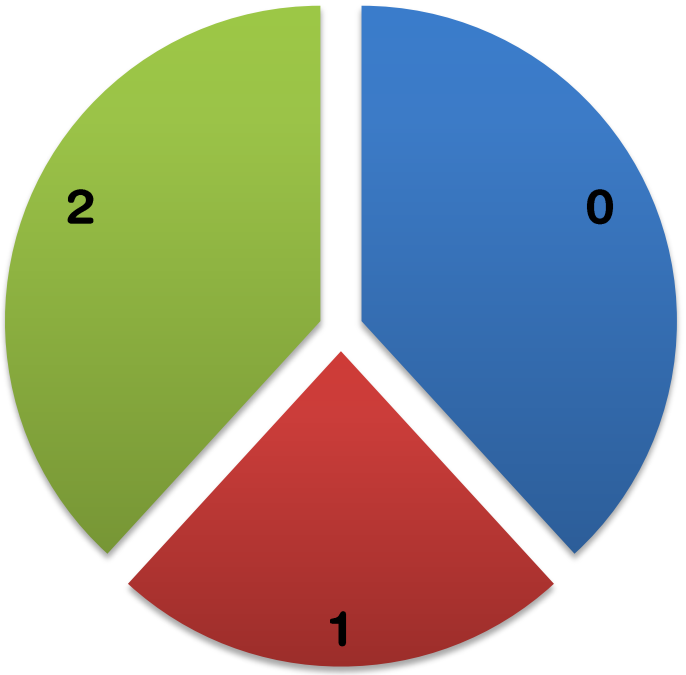
Vérification :

$$W_K = 0.62N = 0.62(U_K + V_K) \text{ car } 0.62 * (0.382 + 0.236) = 0.382$$

Proportions en fonction du nombre de lettres vendues

Proportions :

$(0.382, 0.236, 0.382)$



Chaîne de Lettres

Systèmes De Ponzi

Variables essentielles

Dépôt initial : S_0

Flux entrant : $s(t)$

Taux nominal promis : r_p

Taux nominal d'intérêt : r_n

Taux nominal de retrait : r_w

Argent promis :

$$\frac{dS_f}{dt}(t) = (r_p - r_w)S_f(t) + s(t)$$

$$S_f(t) = e^{(r_p - r_w)t}\left(S_0 + \int_0^t e^{-(r_p - r_w)u} s(u)du\right)$$

Argent réel :

$$\frac{dS}{dt}(t) = r_n S(t) + s(t) - r_w S_f(t)$$

$$S(t) = e^{r_n t}\left(S_0 + \int_0^t e^{-r_n u}\big(s(u) - r_w S_f(u)\big)du\right)$$

Investissement :

$$s(t) = S_0(e^{r_i t} - 1)$$

Argent promis :

$$S_f(t) = S_0\left(-\frac{1}{r_w - r_p}\right) + S_0\left(\frac{1}{r_i + r_w - r_p}\right)e^{r_i t} + S_0\left(1 + \frac{1}{r_w - r_p} - \frac{1}{r_i + r_w - r_p}\right)e^{(r_p - r_w)t}$$

Argent réel :

$$S(t) = a + be^{r_n t} + ce^{r_i t} + de^{(r_p - r_w)t}$$

Calcul des constantes :

$$a = S_0\left(\frac{1}{r_n} - \frac{r_w}{r_n(r_w - r_p)}\right)$$

$$b = S_0\left(1 - \frac{1}{r_i - r_n} - \frac{1}{r_n} + \frac{r_w}{r_n(r_w - r_p)} + \frac{r_w}{(r_i + r_w - r_p)(r_i - r_n)} + \frac{r_w}{r_p - r_w - r_n}\left(1 + \frac{1}{r_w - r_p} - \frac{1}{r_i + r_w - r_p}\right)\right)$$

$$c = S_0\left(\frac{1}{r_i - r_n} - \frac{r_w}{(r_i - r_n)(r_i + r_w - r_p)}\right)$$

$$d = S_0\left(-\frac{r_w}{r_p - r_w - r_n}\right)\left(1 + \frac{1}{r_w - r_p} - \frac{1}{r_i + r_w - r_p}\right)$$

Madoff :

$$r_p = 10\%, \qquad r_n = 2.5\%, \qquad S_0 = 1$$

$$r_w = ?, \qquad r_i = ?,$$

Date de l'effondrement (années) :

r_w/r_i	0%	2%	4%	6%	8%
1%	31,3	55,6	63,6	73,9	95,8
3%	20,3	40,6	47,4	54,6	67,6
5%	16,2	34,6	41,1	47,3	57,8
7%	13,8	31,0	37,4	43,1	52,6
9%	12,2	28,6	35,0	40,4	49,2

Son escroquerie a duré 40 ans

Gains :

$$W(t) = -S_0 + \int_0^t \left(r_w S_f(u) - s(u)\right) du$$

$$W(t) = \alpha + \beta e^{r_i t} + \gamma e^{(r_p - r_w)}$$

$$\alpha = S_0 \left(-1 + \frac{1}{r_i} - \frac{r_w}{r_i(r_i + r_w - r_p)} - \frac{r_w}{r_p - r_w} \left(1 + \frac{1}{r_w - r_p} - \frac{1}{r_i + r_w - r_p} \right) \right)$$

$$\beta = S_0 \left(-\frac{1}{r_i} + \frac{r_w}{r_i(r_i + r_w - r_p)} \right)$$

$$\gamma = S_0 \left(\frac{r_w}{r_p - r_w} \right) \left(1 + \frac{1}{r_w - r_p} - \frac{1}{r_i + r_w - r_p} \right)$$

Valeurs de taux

Gain total à l'effondrement :

r_w/r_i	0%	2%	4%	6%	8%
1%	−34,0	−48,9	−17,7	142,0	2213,7
3%	−28,7	−54,7	−55,4	−38,0	90,5
5%	−32,2	−67,5	−76,0	−76,5	−49,5
7%	−45,8	−102,3	−120,9	−133,3	−139,4
9%	−121,8	−185,3	−347,3	−396,9	−469,7

Chaîne de Lettres

Tableau des Constantes

Modèle	Seuil	Temps d'arrêt	Proportions à l'arrivée	Espérance à l'origine
Vente pyramidale	$1/e$ $\cong 0.368$	N	50% 0 25% 1 25% 2 +	$d \ln N$
Markov	$\frac{(3-\sqrt{5})}{2}$ $\cong 0.382$	$1.62N$	38.2% 0 23.6% 1 38.2% 2	
Branchement		$\frac{C}{1-\mathbb{E}(Z_1)}$		$50(m+m^{12}-2)$
Ponzi	$r_i > 6\%$ $r_w < 3\%$	~40 ans	$12\% > 0$ $88\% < 0$	$-120S_0$

```
# C:\Users\Victor\Desktop\TIPE Code.py
001| import random
002|
003| def sold(p1, p2):
004|     p0 = 1-p1-p2
005|     return(random.choices((0,1,2),(p0,p1,p2))[0])
006|
007| def pop_gen(g_max, p1, p2):
008|     g = 0
009|     pop = [[g, -50]]
010|     indice = 0
011|     while g < g_max and len(pop)-indice > 0:
012|         g += 1
013|         l = len(pop)
014|         for i in range(indice, l):
015|             s = sold(p1, p2)
016|             pop[i][1] += 50*s
017|             pop += [[g, -50]]*s
018|         indice = l
019|     return(pop)
020|
021| def pop_gen_repaired(g_max, p1, p2):
022|     g = 0
023|     pop = [[g, -50]]
024|     indice = 0
025|     while g < g_max and len(pop)-indice > 0:
026|         g += 1
027|         l = len(pop)
028|         for i in range(indice, l):
029|             s = sold(p1, p2)
030|             pop[i][1] += 50*s
031|             for i in range(s):
032|                 pop += [[g, -50]]
033|         indice = l
034|     return(pop)
035|
036| def total_sum(t):
037|     sum = 0
038|     for elem in t:
039|         sum += elem[1]
040|     return(sum)
041|
042| def update_letter(pop, i):
043|     letter = pop[i][2]
044|     first = letter[0]
045|     if first is not None:
046|         pop[first][1] += 50
047|     return(letter[1::]+[i])
048|
049| def pop_gen2(p1, p2, population):
050|     g = 0
051|     pop = [[g, -100, [None]*12]]
052|     indice = 0
053|     while len(pop)-indice > 0:
054|         g += 1
```

```
055|         l = len(pop)
056|         pop_dispo = population-l
057|         p1_g = p1*(pop_dispo)/population
058|         p2_g = p2*(pop_dispo)/population
059|         for i in range(indice,l):
060|             s = sold(p1_g, p2_g)
061|             pop[i][1] += 50*s
062|             for j in range(s):
063|                 pop.extend([[g, -100, update_letter(pop, i)]])
064|         indice = l
065|     return(pop)
066|
067| def data(t):
068|     return([elem[:2] for elem in t])
069|
070| def data_display(t):
071|     for elem in t:
072|         print(elem[0], elem[1])
073|
074| def avg(t):
075|     if t == []:
076|         return([])
077|     gen_max = t[-1][0]
078|     stats = []
079|     i = 0
080|     for gen in range(gen_max+1):
081|         sum = 0
082|         n = 0
083|         while i < len(t) and t[i][0] == gen :
084|             n += 1
085|             sum += t[i][1]
086|             i += 1
087|         stats += [[gen, n, sum//n]]
088|     return(stats)
089|
090| def avg_sumpop(t):
091|     if t == []:
092|         return([])
093|     gen_max = t[-1][0]
094|     stats = []
095|     i = 0
096|     for gen in range(gen_max+1):
097|         sum = 0
098|         n = 0
099|         while i < len(t) and t[i][0] == gen :
100|             n += 1
101|             sum += t[i][1]
102|             i += 1
103|         stats += [[gen, i, sum//n]]
104|     return(stats)
105|
106| def avg_display(t):
107|     if t == []:
108|         return([])
109|     gen_max = t[-1][0]
```

```
    i = 0
    for gen in range(gen_max+1):
        sum = 0
        n = 0
        while i < len(t) and t[i][0] == gen:
            n +=1
            sum += t[i][1]
            i += 1
        print(gen, n, sum//n)

def avg_sumpop_display(t):
    if t == []:
        return([])
    gen_max = t[-1][0]
    i = 0
    for gen in range(gen_max+1):
        sum = 0
        n = 0
        while i < len(t) and t[i][0] == gen:
            n +=1
            sum += t[i][1]
            i += 1
        print(gen, i, sum//n)

import matplotlib.pyplot as plt

def coord_gen(n, g_max, p1, p2):
    coord = []
    for i in range(n):
        t = pop_gen(g_max, p1, p2)
        x = [elem[0] for elem in avg(t)]
        y1 = [elem[1] for elem in avg(t)]
        y2 = [elem[2] for elem in avg(t)]
        coord += [[x,y1,y2]]
    return(coord)

def coord_gen2(n, p1, p2, population):
    coord = []
    for i in range(n):
        t = pop_gen2(p1, p2, population)
        x = [elem[0] for elem in avg(t)]
        y1 = [elem[1] for elem in avg(t)]
        y2 = [elem[2] for elem in avg(t)]
        coord += [[x,y1,y2]]
    return(coord)

def coord_plot(coord, type):
    for elem in coord:
        x = elem[0]
        y1 = elem[1]
        y2 = elem[2]
        if type == 1:
            plt.plot(x,y1)
        else :
            plt.plot(x,y2)
```

```
165|
166| def plot_display(type):
167|     plt.xlabel("Génération")
168|     if type == 1:
169|         plt.ylabel("Nb_acheteurs")
170|         plt.title("Graphique des participants par génération")
171|     else:
172|         plt.ylabel("Gain moyen")
173|         plt.title("Graphique des gains par génération")
174|     plt.show()
175|
176| def histo_data(p1, p2, population):
177|     t = pop_gen2(p1, p2, population)
178|     g_max = t[-1][0]
179|     h = [[elem[1] for elem in data(t) if elem[0] == g] for g in
range(g_max+1)]
180|     return(h)
181|
182| def histo_bar(h, g):
183|     if len(h) <= g:
184|         dat = [-2]
185|     else:
186|         dat = h[g]
187|     categories = [50*i for i in range(-2, max(dat)//50)]
188|     plt.hist(dat, categories, rwidth = 0.9)
189|
190| def histo_display():
191|     plt.xlabel("Gain")
192|     plt.ylabel("Nombre")
193|     plt.title("Probabilité de gains")
194|     plt.show()
195|
196| def prob_bar(n, p1, p2, population, g):
197|     dat = []
198|     for i in range(n):
199|         h = histo_data(p1, p2, population)
200|         if len(h) > g:
201|             dat += h[g]
202|     categories = [50*i for i in range (-2, max(dat)//50)]
203|     plt.hist(dat, categories, rwidth = 0.9)
204|
205| def prob_gain(n, cap, p1, p2, population):
206|     dat = [[]]
207|     for i in range(n):
208|         h = histo_data(p1, p2, population)
209|         diff = len(h)-len(dat)
210|         if diff > 0:
211|             for j in range(diff):
212|                 dat += [[]]
213|         for g in range(len(h)):
214|             p = len([elem for elem in h[g] if elem >
cap])/len(h[g])
215|             dat[g] += [p]
216|     for i in range(len(dat)):
217|         dat[i] = sum(dat[i])/len(dat[i])
```

```
    return(dat)

def bar_display(dat):
    plt.bar([i for i in range(len(dat))], dat)
    plt.xlabel("Gain")
    plt.ylabel("Proba")
    plt.title("Probabilité de gains")
    plt.show()

def expect_val(n, p1, p2, population):
    dat = [[]]
    for i in range(n):
        h = histo_data(p1, p2, population)
        diff = len(h)-len(dat)
        if diff > 0:
            for j in range(diff):
                dat += [[]]
        for g in range(len(h)):
            e = sum(h[g])/len(h[g])
            dat[g] += [e]
    for i in range(len(dat)):
        dat[i] = sum(dat[i])/len(dat[i])
    return(dat)

def expect_val2(n, p1, p2, population):
    dat_money = [[]]
    dat_pop = [[]]
    for i in range(n):
        h = histo_data(p1, p2, population)
        diff = len(h)-len(dat_money)
        tot = 0
        if diff > 0:
            for j in range(diff):
                dat_money += [[]]
                dat_pop += [[]]
        for g in range(len(h)):
            l = len(h[g])
            e = sum(h[g])/l
            dat_money[g] += [e]
            tot += l
            dat_pop[g] += [tot]
        for j in range(len(h)):
            dat_pop[j][-1] = dat_pop[j][-1]/tot
    for i in range(len(dat_money)):
        money_moy = sum(dat_money[i])/len(dat_money[i])
        money_var = sum([(e-money_moy)**2 for e in
dat_money[i]])/len(dat_money[i])
        dat_money[i] = (money_moy, money_var**(1/2))
        pop_moy = sum(dat_pop[i])/len(dat_pop[i])
        pop_var = sum([(e-pop_moy)**2 for e in
dat_pop[i]])/len(dat_pop[i])
        dat_pop[i] = (pop_moy, pop_var**(1/2))
    return(dat_money,dat_pop)

def money_display(dat):
```

```
271|     plt.bar([i for i in range(len(dat))], dat)
272|     plt.xlabel("Génération")
273|     plt.ylabel("Gain")
274|     plt.title("Espérance de gain")
275|     plt.show()
276|
277| def pop_display(dat):
278|     plt.bar([i for i in range(len(dat))], dat)
279|     plt.xlabel("Génération")
280|     plt.ylabel("Population")
281|     plt.title("Espérance de population")
282|     plt.show()
283|
284| def double_display(dat):
285|     plt.bar([i for i in range(len(dat[0]))], dat[0])
286|     plt.bar([i for i in range(len(dat[1]))], dat[1])
287|     plt.xlabel("Génération")
288|     plt.ylabel("Gain et population")
289|     plt.title("Espérance de gain et population")
290|     plt.show()
291|
292| def money_var_display(dat):
293|     plt.xlabel("Génération")
294|     plt.ylabel("Gain")
295|     plt.title("Espérance de gain")
296|     dat1 = [e[0]+e[1] for e in dat]
297|     dat2 = [e[0] for e in dat]
298|     dat3 = [e[0]-e[1] for e in dat]
299|     plt.bar([i for i in range(len(dat))], dat1, color=[0,0,1])
300|     plt.bar([i for i in range(len(dat))], dat2, color=[0,1,0])
301|     plt.bar([i for i in range(len(dat))], dat3, color=[1,0,0])
302|     plt.show()
303|
304| def money_var_display2(dat):
305|     plt.xlabel("Génération")
306|     plt.ylabel("Gain")
307|     plt.title("Espérance de gain")
308|     dat1 = [e[0]-e[1] for e in dat]
309|     dat2 = [e[0] for e in dat]
310|     dat3 = [e[0]+e[1] for e in dat]
311|     plt.bar([i for i in range(len(dat))], dat1, color=[1,0,0])
312|     plt.bar([i for i in range(len(dat))], dat2, color=[0,1,0])
313|     plt.bar([i for i in range(len(dat))], dat3, color=[0,0,1])
314|     plt.show()
315|
316| def pop_var_display(dat):
317|     plt.xlabel("Génération")
318|     plt.ylabel("Population")
319|     plt.title("Espérance de population")
320|     dat1 = [e[0]+e[1] for e in dat]
321|     dat2 = [e[0] for e in dat]
322|     dat3 = [e[0]-e[1] for e in dat]
323|     plt.bar([i for i in range(len(dat))], dat1, color=[0,0,1])
324|     plt.bar([i for i in range(len(dat))], dat2, color=[0,1,0])
325|     plt.bar([i for i in range(len(dat))], dat3, color=[1,0,0])
```

```
326|     plt.show()
327|
328| def Fn(p1,p2,n,x):
329|     p0 = 1-p1-p2
330|     a = x
331|     for i in range(n):
332|         a = p0+p1*a+p2*a**2
333|     return(a)
334|
335| from sympy import symbols, simplify
336|
337| p1 = symbols('p1', float = True)
338| p2 = symbols('p2', float = True)
339|
340| F = [1-p1-p2,p1,p2]
341|
342| def Somme(P,Q):
343|     S = []
344|     m = min(len(P), len(Q))
345|     for  i in range(m):
346|         S.append(P[i]+Q[i])
347|     S.extend(P[m:])
348|     S.extend(Q[m:])
349|     return(S)
350|
351| def Produit(P,Q):
352|     S = [0]*(len(P)+len(Q)-1)
353|     for i, a in enumerate(P) :
354|         for j, b in enumerate(Q):
355|             S[i+j] = S[i+j] + a*b
356|     return(S)
357|
358| def Puissances(P,n):
359|     Pn = [[1]]
360|     for i in range(n):
361|         Pn.append(Produit(Pn[-1],P))
362|     return(Pn)
363|
364| def Compose(P,Q):
365|     PQ = []
366|     for a, Qi in zip(P, Puissances(Q, len(P)-1)):
367|         PQ = Somme(PQ, Produit([a], Qi))
368|     return(PQ)
369|
370| def Iteres(P,n):
371|     L = [[0,1]]
372|     for i in range(n):
373|        L.append(Compose(L[-1],P))
374|     return(L)
375|
376| a = simplify(Iteres(F,3)[-1][0])
```