L. DIDIER & R. CABANE

Un problème de surréservation

Présentation et objectifs

Dans le programme (spécialité Terminale)

Contenus	Capacités attendues	Algorithmes
Épreuve de Bernoulli, loi de Bernoulli. Schéma de Bernoulli : répétition de n épreuves de Bernoulli indépendantes. Loi binomiale $B(n,p)$: loi du nombre de succès. Expression à l'aide des coefficients binomiaux.	Modéliser une situation par un schéma de Bernoulli, par une loi binomiale. Utiliser l'expression de la loi binomiale pour résoudre un problème de seuil, de comparaison, d'optimisation relatif à des probabilités de nombre de succès. Dans le cadre d'une résolution de problème modélisé par une variable binomiale X, calculer numériquement une probabilité du type $P(X=k)$, $P(X\leq k)$, $P(-k\leq X\leq k)$, en s'aidant au besoin d'un algorithme ; chercher un intervalle I pour lequel la probabilité $P(X\in I)$ est inférieure à une valeur donnée α, ou supérieure à $1-\alpha$.	Problème de la surréservation. Étant donné une variable aléatoire binomiale X et un réel strictement positif α, détermination du plus petit entier k tel que $P(X>k)\leq\alpha$.

Positionnement du problème

Dans certains domaines et notamment dans le transport aérien, pour un vol donné, un certain nombre de passagers ayant procédé à une réservation ne se présentent pas à l'embarquement (maladie, retard, etc.).

Le taux de personnes qui ne se présentent pas semble se situer en moyenne autour de 5 %. Ce taux était plus important il y a quelques années car on pouvait annuler sa réservation sans pénalités.

Ce taux est confidentiel car ce taux de non-présentation est intimement lié à un autre, celui de la surréservation.

Pour chacun de leurs vols, afin d'améliorer le taux de remplissage de l'avion et donc la rentabilité du vol, les compagnies proposent un nombre de réservations supérieur au nombre de places de l'avion : c'est la surréservation, ou surbooking.

Dans cette pratique le risque est évident : que certains voyageurs ne puissent pas embarquer en raison d'un manque de places dans l'avion.

Les victimes de ce fonctionnement seront dédommagées financièrement.

Comment est déterminé le nombre de places en surréservation sachant que cela coûte de l'argent à la compagnie ? (dédommagement, prise en charge des frais d'hôtel, de restaurant, etc.)

À faire…

Étudions un exemple :

Un avion de la compagnie TI-Airline en provenance de Maths-en-ville et à destination de Lumière-ville possède 300 places à bord. Les années précédentes, la compagnie a observé un taux de non présentation à l'embarquement de 4 % sur ce vol pour les personnes ayant effectué une réservation.

Niveau : spécialité maths Terminale

Un problème de surréservation

L. DIDIER & R. CABANE

Toutes les personnes ayant procédé à une réservation ont la même probabilité de ne pas se présenter à l'enregistrement et on suppose que leurs comportements sont indépendants les uns des autres. On note α le pourcentage de risque accepté par la compagnie pour que certains voyageurs soient en situation de surbooking. Combien de places doit vendre la compagnie aérienne TI-Airline sur ce vol pour respecter le risque accepté par la compagnie mais aussi remplir au maximum l'avion ?

Buts à atteindre

1. Écrire une fonction qui prend en paramètres deux entiers naturels n et k et qui renvoie le coefficient binomial associé.
2. Écrire une fonction qui prend en paramètres trois nombres (deux entiers naturels n et k et un nombre décimal p compris entre 0 et 1) et qui renvoie $P(X=k)$, où X suit une loi binomiale de paramètres n et p.
3. Écrire une fonction qui permet de répondre à l'exemple étudié en supposant que la compagnie aérienne ne souhaite pas manquer de places dans l'avion dans plus de 1 % de ses vols.

L. DIDIER & R. CABANE

Fiche méthode

Étapes de résolution

▶ Objectif 1 :

Voir la fiche Triangle de Pascal qui présente diverses manières de calculer les coefficients binomiaux. Nous proposons ici une manière (efficace) parmi toutes celles présentées.

Si $k>n$ on considère que le coefficient est nul.

On peut utiliser un argument de symétrie sur les coefficients pour gagner en efficacité.

ÉDITEUR : SURBOOKI
LIGNE DU SCRIPT 0001

```
def coeff(n,k):
    if k>n:
        return 0
    e=1
    if 2*k>n:
        k=n-k
    for i in range(1,k+1):
        e=(e*(n-k+i))//i
    return e
```

▶ Objectif 2 :

On crée un fonction `binomiale` qui prend en paramètres trois nombres (deux entiers naturels n et k et p un nombre décimal compris entre 0 et 1) et qui renvoie $P(X=k)$, où X suit une loi binomiale de paramètres n et p. Le cas $k>n$ renvoie bien une probabilité nulle car `coeff(n,k)` renvoie 0 dans ce cas.

On calcule ici : $\binom{n}{k}p^k(1-p)^{n-k}$.

ÉDITEUR : SURBOOKI
LIGNE DU SCRIPT 0021

```
def binomiale(n,p,k):
    prob=p**k*(1-p)**(n-k)
    return coeff(n,k)*prob
```

▶ Objectif 3 :

Dans un premier temps on va créer la fonction de répartition de la fonction binomiale appelée `binomFrep` qui prend comme arguments deux entiers naturels k et n et qui renvoie $P(X\leq k)$ où X suit une loi binomiale de paramètres n et k.

Il s'agit ici de calculer une somme $\sum_{i=0}^{k} P(X=i)$ à l'aide d'une boucle.

ÉDITEUR : SURBOOKI
LIGNE DU SCRIPT 0023

```
def binomFrep(n,p,k):
    S=0
    for i in range(k+1):
        S=binomiale(n,p,i)+S
    return S
```

Dans un second temps, nous créons une fonction `surbooking` qui renvoie le nombre de places répondant au problème posé et prend en paramètres :

- `alpha`, un nombre décimal qui représente le pourcentage de risque accepté en effectuant du surbooking (ici 0,01) ;
- `n`, entier représentant le nombre de places dans l'avion (ici 300) ;
- `p`, nombre décimal représentant la probabilité qu'une personne ayant effectué une réservation se présente (ici 0,96).

ÉDITEUR : SURBOOKI
LIGNE DU SCRIPT 0040

```
def surbooking(alpha,n,p):
    proba = 0
    i=n
    while proba<alpha:
        proba= 1-binomFrep(i,p,n)
        i = i+1
    return i-2
```

Soit X une variable suivant une loi binomiale de paramètres n et p=0,96. Le problème revient donc à chercher un entier n maximal tel que $P(X>300)\leq 0{,}01$. On crée donc une boucle `while` qui fait appel à la fonction `binomFrep` pour répondre au problème posé.

PYTHON SHELL

```
>>> surbooking(0.01,300,0.96)
305
```

On cherche ici à déterminer le plus grand entier naturel i tel que si $X=Bin(i,p)$, $P(X\leq 300)\geq 1-\alpha$, ce qui est équivalent à : $1-P(X\leq 300)\leq \alpha$. On commencera la recherche avec $i=n$ (au minimum lorsqu'on vend autant de billets que de places).

Pour aller plus loin

Prolongement possible

La compagnie, en faisant du surbooking, cherche à optimiser son chiffre d'affaires. Cette optimisation est complexe au vu des paramètres mis en jeu, nous allons étudier cela sur un exemple simplifié.

Maintenant, sur ce même vol, la compagnie TI-Airline espère vendre le billet aller 500 €. Elle envisage de verser un dédommagement de 1000 € (remboursement de 500 € + prime de 500 €) aux personnes n'ayant pu embarquer.

Travail de groupe

La classe se répartit en plusieurs petits groupes, qui essayent de répondre à la question suivante :

Quel nombre n ($n \geq 300$) de billets doit vendre la compagnie afin d'optimiser son chiffre d'affaires ?

Chaque groupe devra présenter sa réponse à la classe en justifiant.

Éléments de réponse :

On pose n le nombre de billets vendus, X_n le nombre de personnes qui se présentent, et Y_n le nombre de personnes en surbooking ($Y_n = \max(X_n - 300, 0)$). On note G_n le chiffre d'affaires en euro : $G_n = 500n - 1000Y_n$.

On peut répondre à l'aide d'un tableur, simulant les diverses variables aléatoires dans des colonnes[1] et répétant le tout sur autant de lignes successives que nécessaire pour « stabiliser » les fréquences et pouvoir ainsi observer les résultats. Le tableau reproduit ci-dessous montre que l'optimum semble se réaliser avec une vente de 310 ou 311 billets vendus.

n	X(n)	Y(n)	G(n)	Moyenne nb places manquantes	Moyenne du gain pour n places	n	Moyenne du gain sur 10 000 simulations
315	304	4	144000	2,95	145316	300	144012
	303	3	145500			301	144475
	300	0	150000			302	144943
	305	5	142500			303	145473
	305	5	142500			304	145898
	308	8	138000			305	146386
	306	6	141000			306	146852
	298	0	149000			307	147255
	298	0	149000			308	147574
	302	2	147000			309	147813
	304	4	144000			310	147838
	301	1	148500			311	147807
	303	3	145500			312	147487
	304	4	144000			313	146894
	302	2	147000			314	146194
	302	2	147000			315	145305
	299	0	149500			316	144218
	301	1	148500			317	143113
	303	3	145500			318	141821

1 Il faut employer la fonction CRITERE.LOI.BINOMIALE combinée avec la fonction ALEA.

Une réponse probabiliste est aussi envisageable ; elle consisterait à faire un calcul d'espérance mathématique, et ajuster n de sorte que l'espérance de G_n soit maximale, mais c'est assez compliqué.

Une réponse algorithmique consiste à simuler la situation suffisamment de fois pour que la moyenne des valeurs simulées de G_n se stabilise autour de l'espérance de G_n (loi des grands nombres), avant d'ajuster n pareillement.

Nspire CX Le nombre de calculs requis est assez élevé, nécessitant une bonne puissance de calcul ; c'est une situation où une machine rapide comme la TI-Nspire™ CX II-T permet d'obtenir une réponse dans un délai raisonnable. Le script Python correspondant est montré ci-contre.

Quelques remarques sur la programmation : l'essentiel est de simuler une variable aléatoire suivant une loi binomiale. Pour ce faire, on commence la simulation `bern` d'une variable de Bernoulli, ce qui oblige à employer la fonction `random` du module standard `random`. Pour simuler une variable de Bernoulli de paramètre 0,96 on pourrait aussi bien faire appel à `randint(0,96)/100`. Dès lors, la simulation de la variable binomiale revient à répéter des appels à la fonction `bern` autant de fois que nécessaire.

On peut dès lors faire la simulation du gain sur une répétition de `r` vols avec pour chacun `n` billets vendus ($n>n_0=300$ places), et une probabilité `p`=0,96 pour chacun des passagers de se présenter à l'embarquement. À la suite, on teste le gain simulé avec une série d'hypothèses portant sur `n`, évoluant entre 300 et 319.

Le test réalisé ici sur 5000 vols simulés suggère plutôt de vendre 312 billets. Que cette réponse soit évolutive provient du fait que les simulations ne donnent pas toujours les mêmes valeurs (d'autant que le nombre de répétitions n'est pas énorme).

SSurbook.py

```
from random import random
def bern(p):
    if random()<p:
        return 1
    return 0
def binomiale(n,p):
    s=0
    for _ in range(n):
        s = s + bern(p)
    return s

def gain(n,p,n0,p1,p2,r):
#n  = nb de billets vendus
#n0 = nb de places
#p1 = prix de vente des billets
#p2 = dédommagement / surbook
#r  = répétitions
    g = 0 # gain total
    for _ in range(r):
        m = binomiale(n,p) # alea
        g = g + m * p1 # ventes
        if m > n0: # ! surbooking
            g = g - (m - n0) * p2
    return int(g/r) # moyenne

L,p=[],0.96 # initialisation
for k in range(300,320):
    L.append(gain(k,p,
    300,500,1000,5000))
m=max(L)
print("p=",round(p,2),
" optim : ",m," places : ",
L.index(m)+300)
```

`max(L)` permet de calculer la valeur maximum dans la liste `L`.

`L.index(m)` renvoie l'index du premier terme de la liste `L` ayant la valeur `m`.

1.1 | 1.2 | Ssurbook

Shell Python

```
>>>#Running SSurbook.py
>>>from SSurbook import *
p= 0.96  optim :  148624  places :  312
```

L. DIDIER & R. CABANE