YOCH

Zeste de savoir

Tirages aléatoires non équiprobables

24 mai 2021

Table des matières

1. Aperçu des différentes approches courantes (ou pas) 1
   - 1.1. Approche discrète 2
   - 1.2. Fréquences cumulées 2
   - 1.3. Lancer de fléchettes 2
   - 1.4. Méthode Alias 3
   - 1.5. Arbre binaire 4

Bonjour à tous,

Je vous propose de découvrir quelques algorithmes intéressants sur le thème des tirages aléatoires pondérés, sujet relativement courant en simulation, mais quelquefois méconnu.

Dans les cas qui suivent, on considérera le problème suivant: soit une liste de N éléments, ayant chacun une probabilité p_k, comment tirer efficacement un élément à partir de cette distribution?

Avant tout, on va supposer avoir à notre disposition deux «outils» fondamentaux:

- un générateur aléatoire permettant de tirer un entier entre 0 et N ;
- un générateur aléatoire permettant de tirer un réel entre 0 et 1.

Ces deux générateurs nous permettent déjà de gérer les cas les plus simples:

- dans le cas d'une distribution uniforme, le générateur d'entiers suffit pour choisir un élément dans une liste donnée;
- s'il existe seulement deux tirages possibles (mettons 'A' et 'B'), avec respectivement une probabilité p et $1-p$, le générateur de réels suffit pour procéder au tirage: on tire une valeur x entre 0 et 1, si $x \leq p$ alors on choisit 'A', sinon 'B'.

1. Aperçu des différentes approches courantes (ou pas)

Cette partie s'inspire largement de cette excellente page sur le sujet, n'hésitez pas à y jeter un œil.

1.1. Approche discrète

Une façon de faire, lorsque c'est possible, est d'utiliser un tableau de valeurs correspondant à ladite distribution. Par exemple, admettons que les probabilités de tirage soient les suivantes: 70% de 'A', 10% de 'B' et 20% de 'C'. On peut dans ce cas employer un tableau de 10 éléments, soit 7 'A' 1 'B' et 2 'C', puis choisir aléatoirement une valeur dans le tableau avec notre générateur d'entiers de 1 à 10.

On fait ici le compromis d'utiliser un peu d'espace mémoire supplémentaire afin de procéder simplement au tirage. Malheureusement, cette approche n'est pas envisageable dans tous les cas, et va dépendre de l'aspect de notre distribution. Toutefois, d'autres approches plus rusées permettent d'étendre cette méthode pour restreindre largement l'espace nécessaire avec un léger surcoût en temps[1].

1.2. Fréquences cumulées

Avec notre tableau de probabilités, on peut résoudre le problème avec un seul tirage dans $[0..1]$, moyennant une recherche linéaire dans la distribution. Soit P le tableau des probabilités cumulées ($P_0 = 0, P_1 = P_0 + p_1, \ldots, P_n = P_{n-1} + p_n$), on génère un réel x entre 0 et 1, puis on cherche dans P l'indice k qui satisfait $P_{k-1} \leq x < P_k$.

Dans sa version naïve, la complexité de la recherche de k est en $O(N)$, ce qui est particulièrement inefficace. Afin d'améliorer la complexité moyenne, on peut trier les valeurs de P en ordre décroissant, ainsi la recherche aura une meilleure probabilité de terminer plus tôt, mais on peut faire encore mieux.

Une optimisation intéressante est d'effectuer une recherche dichotomique pour trouver le bon indice dans le tableau de fréquences cumulées. L'avantage de cette technique est qu'elle fournit une garantie de complexité de $O(\log N)$.

i

Cette méthode est très couramment employée, c'est d'ailleurs l'algorithme utilisé par python pour `random.choices` .

1.3. Lancer de fléchettes

Il est possible de représenter le tableau de probabilités comme une cible sur lesquelles on lancerait des fléchettes «aléatoires» (i.e. ayant une probabilité égale d'atterrir n'importe où sur la cible). On recommence à lancer tant que l'on n'a pas atteint une portion de la cible correspondant à un tirage valide. On peut montrer que cette procédure correspond au tirage souhaité.

1. Marsaglia, G., Tsang, W. W., & Wang, J. (2004). Fast generation of discrete random variables. Journal of Statistical Software, 11(3), 1–11.

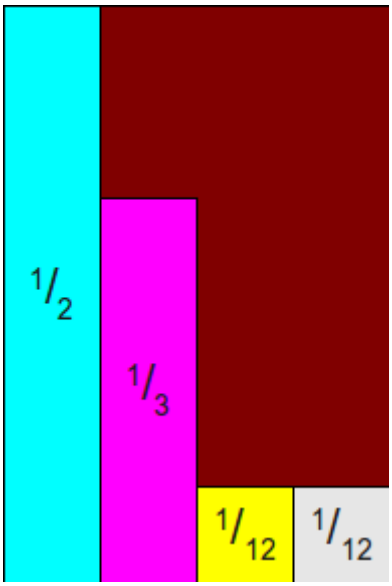

FIGURE 1.1. – Exemple. source : http://www.keithschwarz.com/darts-dice-coins/

Cet algorithme requiert deux tirages : le premier avec le générateur d'entiers pour choisir une colonne, et un second avec le générateur de réels pour savoir si notre fléchette est bien dans la zone valide.

Cet algorithme est très efficace lorsque les probabilités sont proches[2], car la surface valide de la cible est alors importante. En revanche, dans certains cas, l'algorithme demeure assez peu efficace. Mais il en existe une variante particulièrement efficace que nous allons voir.

1.4. Méthode Alias

Il est possible de redistribuer les couleurs sur la cible de manière à ce que l'ensemble de cette dernière soit couverte, avec au maximum deux couleurs par colonne. On peut montrer qu'une telle construction existe toujours.

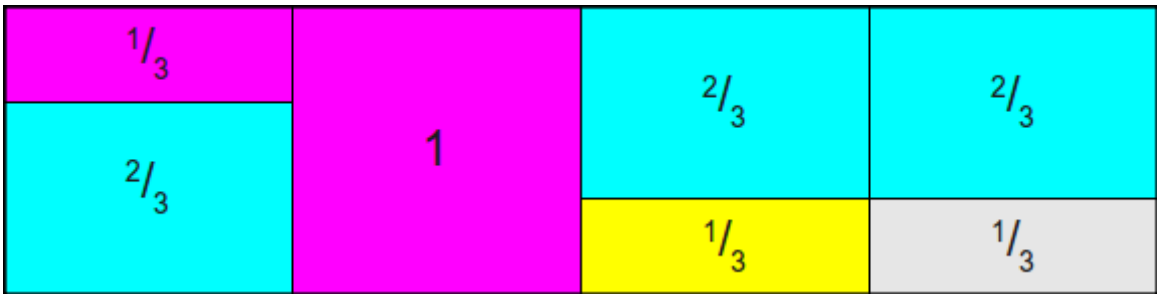

FIGURE 1.2. – Exemple. source : http://www.keithschwarz.com/darts-dice-coins/

Ainsi, il existe un algorithme en $O(1)$ pour générer une valeur pour toute distribution non équiprobable. Par ailleurs, il existe un algorithme de complexité linéaire pour la construction de la table d'alias[3].

1.5. Arbre binaire

Pratiquement toutes les méthodes vues ci-dessus (sauf celle du lancer de fléchettes) supposent que la distribution aléatoire ne change pas au cours du temps, et admettent un temps de construction linéaire, qui se révèle prohibitif si la distribution change souvent (dans le pire des cas, entre chaque tirage).

En faisant l'hypothèse que les probabilités peuvent changer mais N reste fixe, on peut proposer un algorithme qui permet à la fois de modifier une probabilité et de réaliser un tirage en $O(\log N)$. Pour cela, on ne va plus parler de probabilités, mais plutôt de poids attribués à chaque tirage possible (autrement dit on n'impose pas que $\sum_{k=1}^{N} p_k = 1$), ce qui revient finalement au même.

Cet algorithme a originellement été proposé[4] pour réaliser des tirages sans remise, ce qui correspond à assigner $p_k \leftarrow 0$ pour chaque k tirée, mais l'algorithme se généralise sans peine.

Il s'agit de construire un arbre binaire quasi-complet, avec comme feuilles les poids p, et chaque nœud ayant pour valeur la somme des valeurs de ses fils. Ensuite:

1. pour tirer une valeur aléatoire k en accord avec notre distribution, on tire x dans $[0..1]$ à l'aide d'un générateur de réels, puis on multiplie x par la valeur du nœud racine, et on descend le long des branche jusqu'à une feuille en prenant la branche de gauche si $x \leq$ gauche et la branche de droite si $x <$ gauche (dans ce cas, on doit aussi faire $x \leftarrow x -$ gauche);
2. pour modifier un poids, on part de la feuille correspondante et on répercute la modification de bas en haut dans l'arbre.

Voila, j'espère que ce petit tour d'horizon vous aura plu.

Comme j'ai essayé de le montrer, il est difficile de départager ces différentes approches, qui sont chacune plus ou moins adaptée en fonction du profil d'usage.

Je n'ai pas vraiment eu le courage d'entrer plus avant dans les explications, mais vous pouvez bien sûr me faire savoir si le texte vous semble accessible, ou si au contraire certains passages sont vraiment trop mal faits.

2. Lipowski, A., & Lipowska, D. (2012). Roulette-wheel selection via stochastic acceptance. Physica A : Statistical Mechanics and its Applications, 391(6), 2193–2196.
3. Vose, M. D. (1991). A linear algorithm for generating random numbers with a given distribution. IEEE Transactions on software engineering, 17(9), 972–975.
4. Wong, C. K., & Easton, M. C. (1980). An efficient method for weighted sampling without replacement. SIAM Journal on Computing, 9(1), 111–113.