Université de Mons
Faculté des Sciences
Département des mathématiques

Rapport de projet
Simulation

Professeur:
Alain Buys

Auteurs:
Loïc Dupont
Clément Durieux

Année académique 2019-2020

Table des matières

1 Introduction

Le but de ce projet est double. Dans un premier temps, nous allons étudier le caractère pseudo-aléatoire des décimales de pi. Nous allons utiliser pour se faire trois tests vus au cours (chi2, test du gap et test du poker). Dans un second temps, nous allons créer notre propre générateur aléatoire qui sera basé sur les décimales de pi et nous le comparerons ensuite au générateur aléatoire de python en utilisant les trois tests déjà évoqués ci-dessus.

2 Mode d'emploi

Pour lancer les tests, il vous suffit juste de lancer new_pi.py, ce qui lancera les tests pour les décimales de pi ou lancer randomiser.py ce qui lancera les tests mais pour notre générateur et celui du random de python. Pour le randomiser.py, si vous souhaitez changer la quantité de nombres générés, il suffit de changer le k, pour changer le nombre de décimales, il suffit de changer le n et enfin, pour changer le nombre de tests effectués, il vous suffit de changer le l. Les 3 variables se trouvent dans le début de la section main.

3 Algorithme

Dans cette section, vous allez expliquer les différents algorithmes qui vous paraissent importants pour votre projet. (Pour l'explication : son principe, les grandes lignes de comment il s'exécute, sa complexité, ...)

3.1 Partie test

Commençons par parler de la fonction get_decimal_pi. On initialise une liste liste_totale. On ouvre le fichier pi_decimal.txt puis on parcourt toutes les lignes et pour chaque élément de la ligne, on convertit l'élément en int et on le rajoute dans liste_totale si ce n'est pas un retour à la ligne (si c'est le cas on passe à la ligne suivante). Finalement, on retourne liste_finale qui est donc une liste qui contient toutes les décimales de pi.

Abordons maintenant la fonction occ_number, elle a list_value comme unique paramètre qui est une liste qui contient des valeurs. On commence par initialiser un dictionnaire dict_occ, on va ensuite parcourir les valeurs de list_value, si la valeur n'est pas dans le dictionnaire alors sa valeur associée dans le dictionnaire est initialisée à 1 sinon elle est incrémentée de 1. Cette action sert en réalité à compter le nombre de fois qu'une valeur apparait dans list_value. On retourne ensuite dict_occ.

Parlons de la fonction proba_dict_chi2 qui prend en argument r le nombre de valeur possible et dict_value un dictionnaire qui comprend toutes les valeurs possibles ainsi que le nombre de fois qu'elles apparaissent. On commence par initialiser un dictionnaire dict_proba. On pose ensuite proba comme étant égal à 1/r, vu qu'il y a r valeurs qui sont telles que la probabilité d'obtenir cette valeur est la même pour toutes les valeurs et vaut 1/r. On parcourt ensuite dict_value et on pose que la valeur correspondant à chaque élément dans dict_proba est proba. On retourne ensuite dict_proba.

Traitons maintenant de la fonction Kr qui prend comme paramètre N le nombre de valeur considérée, dict_value qui contient toutes les valeurs possibles ainsi que le nombre de fois qu'elles apparaissent et dict_proba qui comprend toutes les valeurs possibles ainsi que la probabilité d'obtenir la valeur. La fonction en elle-même n'est qu'une mise en œuvre de la formule du Kr donnée dans le cours. On retourne la valeur de Kr à la fin de la fonction.

Parlons maintenant de la fonction test_chi2 qui prend les paramètres suivants : r le nombre de valeurs possibles, list_value une liste qui comprend toutes les valeurs considérées, dict_value un dictionnaire qui comprend toutes les valeurs possibles ainsi que le nombre de fois qu'elles apparaissent, dict_proba qui comprend toutes les valeurs possibles ainsi que la probabilité d'obtenir cet valeur et deg qui est le degré de liberté. On commence par initialiser une liste list_win qui nous servira à retourner les résultats des tests avec différentes valeurs d'alpha. On calcule ensuite Kr. On crée ensuite une liste list_alpha qui comprend les différentes valeurs d'alpha. Pour chaque valeur d'alpha on prend la valeur critique (valeur dans le tableau slide 39) correspondante. On compare ensuite la valeur critique et Kr, si Kr est plus petit que la valeur critique alors le test est réussi sinon il est raté. On met le résultat de chaque test dans list_win et on retourne list_win.

On va maintenant parler de la fonction proba_dict_gap qui prend en paramètre p qui est la probabilité de tomber entre a et b et dict_value un dictionnaire qui comprend toutes les valeurs possibles (ici les longueurs entre deux valeurs qui se situent entre a et b) ainsi que le nombre de fois qu'elles apparaissent. On commence par initialiser un dictionnaire dict_proba. On calcule ensuite, pour toutes les longueurs dans dict_value, la probabilité d'obtenir cette longueur suivant la méthode du cours sauf si la longueur vaut 10, il suffit de regarder dans la fonction test_gap pour se rendre compte que l'on ne peut pas avoir une valeur plus grande que 10 cependant ce cas correspond à avoir une longueur valant au moins 10 (cela comprend donc aussi les longueurs valant 11, 12, etc) la probabilité dans ce cas-ci vaut donc 1 auquel on a retiré la somme des probabilités d'obtenir une longueur allant de 0 à 9. On met ensuite toutes les longueurs et leur probabilité dans dict_proba. On retourne ensuite dict_proba.

On va maintenant aborder la fonction test_gap qui prend en paramètre les valeurs a et b qui sont les valeurs utilisées dans le test du gap (données par l'utilisateur), list_value la liste des valeurs considérées et pi un booléen qui est faux si on fait un test avec les générateurs et vrai si on fait le test avec les décimales de pi. On commence par lancer une erreur si les valeurs de a et b ne conviennent pas à nos critères. Si pi est vrai et b est plus grand que 0.9 on dit que b vaut 0.9 (cette action est expliquée dans la section choix personnels). Ensuite, on initialise une valeur k à 0 et on va incrémenter k jusqu'à ce que le k ème élément de list_value soit entre a et b. On initialise ensuite une liste list_gap et un variable compt (à 0). On va ensuite parcourir list_value et incrémenter compt tant que l'on passe par des valeurs qui ne sont pas comprises entre a et b et lorsque l'on passe par une valeur qui se trouve entre a et b on ajoute compt à list_gap sauf si compt est plus grand que 10, dans ce cas on rajoute 10 à list_gap (explications dans choix personnels), on réinitialise ensuite compt à 0. On crée ensuite un dictionnaire qui est le résultat de l'application de occ_number sur list_gap. On calcule ensuite proba qui est la probabilité d'être entre a et b, proba vaut b − a si pi est faux, le calcul de proba si pi est vrai est expliqué dans la partie choix personnels. Puis, on crée un autre dictionnaire qui est le résultat de proba_dict_gap avec proba et dict_value comme argument et on pose deg comme étant la longueur de dict_value − 1. On lance ensuite un test de chi2.

Parlons maintenant de la fonction stirling_number qui prend en paramètre dict_stir, un dictionnaire qui contient les nombres de stirling déjà calculé et les valuers k et r qui sont nécessaires pour calculer le nombre de stirling que l'on veut. Cette fonction n'est qu'une application de la formule donnée dans le cours, elle retourne le nombre de stirling déterminé avec k et r et le stocke dans dict_stir.

Abordons désormais la fonction proba_dict_poker qui prend en paramètre les valeurs k et d qui sont les valeurs utilisées dans le test du poker. On commence par initialiser deux dictionnaires dict_proba et dict_stir. On calcule ensuite, pour tous les entiers i entre 1 et d, la probabilité d'obtenir une valeur dans i intervalles suivant la méthode du cours et on met i et la probabilité d'avoir des valeurs dans i intervalles dans dict_proba. On retourne ensuite dict_proba.

Pour finir cette première partie, parlons de la fonction test_poker qui prend en paramètre list_value une liste qui comprend toutes les valeurs considérées, poker un booléen qui est vrai si on fait un test avec les générateurs et faux si on fait le test avec les décimales de pi et les valeurs k et d qui sont les valeurs utilisées dans le test du poker (d est le nombre d'intervalles et k le nombre de valeurs à avoir dans un même intervalle pour avoir un poker c'est-à-dire le nombre de valeur que l'on va considérer à la fois). On commence par effectuer une opération sur les éléments de list_value si poker est vrai (cette opération est expliquée dans la partie choix personnels). On va ensuite créer proba_dict qui est le résultat de l'appel de proba_dict_poker sur k et d. On initialise ensuite deux listes list_poker et list_test_poker. On parcoure ensuite list_value, on ajoute les éléments de list_value à list_test_poker jusqu'à ce que la longueur de cette dernière valle k. Une fois ce stade atteint, on compte ensuite le nombre de valeurs différentes dans list_test_poker et on ajoute ensuite ce nombre à list_poker, on réinitialise ensuite list_test_poker. On pose ensuite dict_value comme le résultat de l'appel de la fonction occ_number sur list_poker. On effectue ensuite un test de chi2.

3.2 Partie générateur

Commençons avec la fonction randfloat, qui prend en paramètre decimal_number qui est le nombre de décimal de la valeur que l'on va générer. On commence par initialiser un str que l'on appelle floating. On va ensuite générer un nombre time entre 0 et 999 999, on effectue ensuite une série de calcul pour au final obtenir index qui vaut un nombre entre 0 et 999 999, on considère la décimale de pi en position index et on concatène cette décimale à floating, on effectue cette opération jusqu'à avoir obtenu decimal_number nombres. On définit ensuite our_float comme étant la concaténation de '0.' et floating converti en float (équivalent à une division par 10 exposant decimal_number). On retourne ensuite our_float.

Intéressons-nous à la fonction generator_random qui prend comme paramètre decimal_number qui est le nombre de décimal de la valeur que l'on va générer. On commence par générer un nombre grâce au générateur de python et que l'on va appeler number. La deuxième instruction consiste à faire en sorte que number soit entre 0 et 1. On retourne ensuite number.

4 Choix personnel

5 Problèmes survenus

Pour les problèmes que nous avons rencontrés, nous avons eu pas mal de soucis, pour tous les tests, lors du calcul du Kr. En effet, notre premier gros problème venaient surtout de notre dictionnaire concernant les probabilités. En réalité, lorsque nous avons commencé à réfléchir et à coder notre projet, nous n'avions pas réfléchi aux nombres de paquets créés. Par exemple, lors du test de chi2, nous obtenions une longueur de 1000 pour notre dictionnaire, ce qui donne une plus grande sensibilité à une quelconque variation dans les nombres. Ceci explique donc pourquoi dans le test_chi2.txt, il y a autant de tests échoués.

Le deuxième gros problème qu'on avait, qui est lui aussi lié au Kr, concerne le dictionnaire d'occurrences pour chacun des tests. Nous avons eu quelques difficultés pour cerner le problème mais au bout d'un certain temps, nous avons compris que nous prenions également trop large. C'est-à-dire qu'encore une fois, au vu du nombre de paquets, la moindre variation fait que notre Kr peut exploser. Ceci ne se voyait pas avec les décimales de pi car on sait que les valeurs qui se trouveront dans le dictionnaire sont 0, 1, ..., 9. Mais avec nos propres nombres, on aura une plus grande longueur ce qui explique le problème. Pour régler ce fameux problème, nous avons regrouper des cas en intervalle. Par exemple, les x dans [0, 0.1] sont considérés dans un intervalle et ainsi de suite pour tous nos nombres.

6 Conclusion