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 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

3.1 Partie test

3.1.1 Algorithmes généraux

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. Cet algorithme a donc pour but de créer une liste contenant 1 000 000 d'entiers représentant 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. Cet algorithme a donc pour but de créer un dictionnaire calculant l'occurence de chaque nombre.

3.1.2 Algorithmes pour chi2

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. Cet algorithme a donc pour but de créer un dictionnaire de probabilités associant chaque valeur possible avec la probabilité d'obtenir cette valeur

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. Cet algorithme a donc pour but de calculer le Kr.

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 cette 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. Cet algorithme a donc pour but d'effectuer le test de chi2 pour les différentes valeurs d'alpha.

3.1.3 Algorithmes pour Gap

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. Cet algorithme a donc pour but de créer un dictionnaire associant les différentes longueurs que l'on peut obtenir avec la probabilités d'obtenir cette longueur.

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. Cet algorithme a donc pour but d'exécuter le test du gap avec un certain intervalle.

3.1.4 Algorithmes pour Poker

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. Cet algorithme a donc pour but de retourner le nombre de Striling correspondant au k et r et modifie dict_stir pour rendre la fonction plus performante.

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. Cet algorithme a donc pour but de créer un dictionnaire associant le nombre d'intervalle dans lesquelles on trouve au moins une valeur à la probabilité d'obtenir ce nombre d'intervalle.

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. Cet algorithme a donc pour but d'exécuter le test du poker.

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. Cet algorithme a donc pour but de créer un nombre entre [0, 1[avec decimal_number décimales.

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. Cet algorithme a donc pour but de créer, avec le module random de python, un nombre entre [0, 1[decimal_number décimales.

4 Choix personnels

Pour la fonction test_gap, expliquons le fait que si pi est vrai et si b est plus grand que 0.9 (ce b fait référence à celui utilisé en argument dans la fonction test_gap), on pose b comme étant égal à 0.9. Ceci vient du fait que lorsque l'on effectue le test avec les décimales de pi (ce qui est le cas car le booléen pi est vrai dans ce cas-ci), on effectue en réalité le test avec les décimales de pi divisées par 10 (afin qu'elles se trouvent entre 0 et 1), les différentes valeurs possibles sont donc de la forme 0.k où k est un entier entre 0 et 9. Les valeurs étudiées sont donc en réalité entre 0 et 0.9, il n'est donc pas possible d'avoir une valeur plus grande que 0.9, ceci explique pourquoi on limite b à la valeur 0.9 dans le cas présent.

Expliquons maintenant le fait que l'on considère compt comme valant 10 s'il est supérieur à 10. Cela vient du fait que passer une certaine valeur, la probabilité d'obtenir une longueur (et donc compt) plus grande que cette valeur devient très proche de 0 ce qui peut engendrer des erreurs de calculs, c'est pour cela que l'on a décidé de regrouper les cas où la longueur est supérieure à 10 avec le cas où la longueur vaut 10. Le choix de la valeur 10 est purement arbitraire, nous avons constaté que cette valeur était suffisamment grande pour que l'on n'inclut pas les longueurs ayant une probabilité relativement éloignée de 0 et suffisamment petite pour inclure les cas où la probabilité d'obtenir la longueur soit proche de 0.

Parlons maintenant du calcul de proba si pi est vrai. Rappelons-nous d'abord que cela signifie qu'on travaille avec des valeurs entre 0 et 0.9 et qu'il n'y a en réalité que 10 valeurs possibles. Remarquons que si a et b ont plus d'une décimale, chercher des valeurs entre a et b revient à chercher les valeurs entre le plus petit nombre ayant au maximum une décimale plus grand que a et le plus grand nombre ayant au maximum une décimale plus petit que b (ce qui est également évidemment la cas si a et b n'ont qu'une seule décimale). Considérons désormais a et b comme étant des nombres ayant au plus une décimale (si ce n'était pas le cas dès le départ il suffit de prendre les nombres ayant au maximum une décimale comme décrit précédemment), dans ce cas il y a $10 * b - 10 * a - 1$ nombres strictement entre a et b (par exemple pour a valant 0.2 et b valant 0.5, il y a deux valeurs qui sont strictement entre a et b sur les 10 possibles qui sont 0.3 et 0.4), de plus les valeurs de a et b sont évidemment entre a et b, il y a donc $10 * b - 10 * a + 1$ nombres entre a et b. Il suffit de diviser cette valeur par 10 (le nombre de valeurs possibles) pour obtenir la probabilité d'être entre a et b. Ceci conclut l'explication du calcul de proba dans le cas présent ainsi que les choix personnels relatifs à test_gap.

Pour la fonction test_poker, expliquons l'opération effectuée si poker est vrai. On est donc dans le cas où on fait un test avec un générateur. Il y a donc potentiellement un grand nombre de valeur possible (10 exposant le nombre de décimale) ce qui fait que le test peut mal fonctionner (cela peut revenir à lancer un nombre relativement faible de dés ayant un nombre relativement élevé de faces, il serait donc extrêmement improbable d'obtenir un poker). Pour limiter ce problème on va en fait faire en sorte que les nombres dans la liste passée en argument puissent prendre seulement 10 valeurs. Pour ce faire, comme on sait que les valeurs de cette liste sont entre 0 et 1, on va effectuer une opération que va faire en sorte qu'une valeur se trouvant dans [0 ; 0.1[soit considérée comme valant 0, une valeur dans [0.1 ;0.2[comme 0.1, etc. On a donc qu'il n'y a plus que 10 valeurs possibles dans la liste qui sont de la forme 0.k avec k un entier entre 0 et 9. Ceci permet d'éviter le problème décrit plus haut.

5 Problèmes survenus

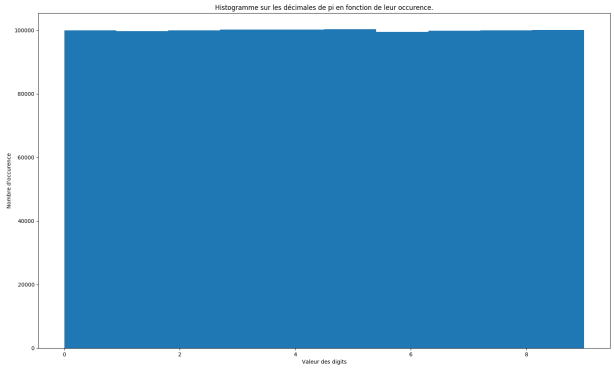
Pour les problèmes que nous avons rencontrés, nous avons eu pas mal de soucis, pour tous les tests, lors du calcul du Kr. En effet, notre premier gros problème venait surtout de notre dictionnaire concernant les probabilités. En réalité, lorsque nous avons commencé à réfléchir et à coder notre projet, 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 des 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 nous sommes rendus compte qu'il s'agissait du même problème. 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 regroupé 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.

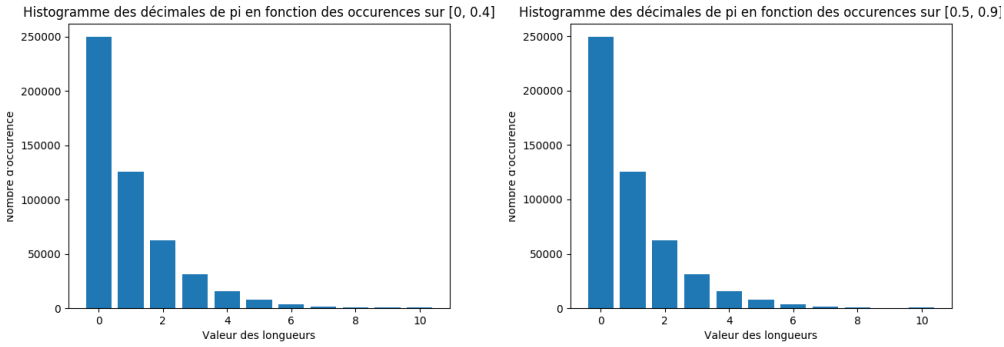
6 Conclusion

6.1 Les décimales de pi

Commençons par le caractère pseudo-aléatoire des décimales de pi. Pour ce qui est du test de chi2, il réussit avec n'importe quelle valeur d'alpha. Il est facile de voir en regardant l'histogramme que les valeurs entre 0 et 9 sont bien réparties uniformément.

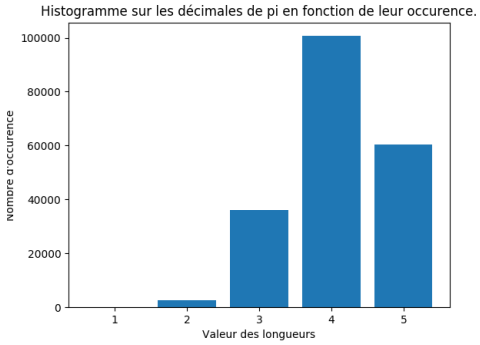


Pour ce qui est du test du gap, il marche avec la plupart des valeurs de a et b (même si il arrive que le test rate avec certaines valeurs d'alpha). Par exemple si a vaut 0 et b vaut 0.4 le test réussit pour n'importe quelle valeur d'alpha et si a vaut 0.5 et b vaut 0.9 le test réussit pour alpha valant 0.025, 0.01 et 0.001. Voici les histogrammes qui représentent les différentes longueurs entre deux éléments dans [a ; b] ainsi que le nombre de fois que ces longueurs apparaissent :



On constate en regardant les histogrammes que l'on obtient à peu près ce que l'on s'attend à avoir.

Enfin, en ce qui concerne le test du poker, le test réussit pour toute les valeurs d'alpha. Voici l'histogramme correspondant aux nombres d'intervalles différents dans lesquels on retrouve au moins une valeur ainsi que le nombre de fois que ces nombres apparaissent :



On peut donc en conclure à la vue de ces différents résultats que les décimales de pi sont plutôt bien réparties de manière uniforme et qu'elles ont donc un bon caractère aléatoire.

6.2 Les générateurs

Pour ce qui est de la partie sur les générateurs, commençons par parler du générateur de python. Sans grande surprise, il passe assez bien tous les tests et les résultats obtenus sont proches de ce à quoi on s'attend. Par exemple, si on lance mille tests avec alpha valant 0.05, on a que plus ou moins 50 tests échouent ce qui est une bonne chose car on s'attend à ce que 5% des tests échouent.

Par contre, la situation n'est pas la même avec notre générateur. En effet, même si les résultats des tests avec le test de chi2 sont plutôt proches de ce à quoi on s'attend, il arrive souvent avec le test du gap et celui du poker que le nombre de tests se solvant par un échec s'éloigne plus ou moins du nombre attendu.

Au regard de ces observations, Il est clair que le générateur de python est un bien meilleur générateur que notre générateur et que ce dernier pourrait être amélioré afin que l'on obtienne des résultats plus proche de ce qui est attendu.