Genereren van pseudo-random getallen

Sinem Ertem

1. Pseudotoevalsgenerator

De pseudotoevalsgenerator[1] (= pseudorandom number generator (PRNG)) is een algoritme voor het generen van pseudotoevalsgetallen. Dat wil zeggen een opeenvolging van ogenschijnlijk willekeurige getallen zonder enige samenhang. Ook wel bekend als deterministisch willekeurige bit-generator. De output die de generator geeft is dus **niet** willekeurig, omdat hij volledig wordt bepaald door een relatief kleine verzameling van initiële waarden.

Het wordt pseudo genoemd omdat het genereren van 'willekeurige' getallen met een bekend model feitelijk niet helemaal willekeurig is. Het is soort van random maar toch niet helemaal. 'Uniform distribution = equally likely to occur ', Een dobbelsteen heeft 1/6 kans op elk getal dus uniform.

1.2 Linear Congruential Generator

Een groot aantal generatoren maakt gebruik van *lineaire congruentie*[2]. In het model wat ik ga maken zal ik gebruik maken van de recursieve formule waarop deze generatoren zijn gebaseerd:

$$Xn+1 = (a\,Xn + c) \bmod m$$

De modules ***m*** *{2,3,4. . .}*
De vermenigvuldigingsfactor ***a*** *{1, . . ., m-1}*
De toename ***c*** *{1, . . ., m – 1}*

De vooraf vastgestelde variabelen:

- num_steps = 5
- previous = if previous is None (in het begin), x=1, else x = previous
- a = 2
- c = 3
- m = 5

x1 = 2 * 1 + 3 (mod 5)
= 0
x2 = 2 * 0 + 3 (mod 5)
= 3
x3 = 2 * 3 + 3 (mod 5)
= 4
x4 = 2 * 4 + 3 (mod 5)
= 1

In het begin van de berekening wordt er voor Xn,1 genomen omdat er natuurlijk nog geen getal is berekend. Als we x1 hebben berekend komt er 0 uit en dit getal nemen we mee voor de volgende berekening. De reeks die hieruit voortkomt is dan ook: [0,3,4,1] en dit zal zich steeds gaan herhalen zolang mod 5 niet wordt veranderd. Stel we veranderen alleen mod naar 6, dan zal de reeks veranderen naar: [5, 1, 5, 1, 5]. Zie hier hoe de getallen alleen bestaan uit 5 en 1. Dit is geen goede uitkomst. Dit heeft te maken met de regel dat voor m een priemgetal wordt genomen omdat deze de beste uitkomsten geeft.

[1] Bron: https://nl.wikipedia.org/wiki/Pseudotoevalsgenerator
[2] Bron: https://nl.wikipedia.org/wiki/Lineaire-congruentiegenerator

2. Kolmogorov-Smirnov Test of Randomness

Nu we met lineaire congruentie verschillende nummers hebben gegenereerd moeten we gaan testen of deze nummers uniform zijn. De *Kolmogorov-Smirnovtoets* is een toets uit de statistiek die gebaseerd is op een maat voor het verschil in twee verdelingen. In de vorm van één steekproef is het een aanpassingstoets waarmee onderzocht wordt of de verdeling (waaruit de steekproef getrokken is) afwijkt van een bekende verdeling zoals: *de normale verdeling* of *uniforme verdeling*. De reeks *X1,…,Xn* die is voortgekomen uit de pseudotoevalsgenerator wordt gezien als een aselecte steekproef. Hierop wordt weer de nulhypothese getoetst tegen de alternatieve hypothese:

H0: generated data might have been sampled from UD
H1: generated data is unlikely to have been sampled from UD

De *Kolmogorov-Smirnov test* kijkt dus wat de grootste deviatie is uit de reeks gegenereerde pseudo random getallen. Als de deviatie groter is dan Dalpha, wordt H0 afgewezen

Voorbeeld:
Stel we hebben de gegenereerde random nummers: 0.44, 0.81, 0.14, 0.05 en 0.93 N=5

i	1	2	3	4	5
R(i)	0.05	0.14	0.44	0.81	0.93
i/N	(1/5) = 0.25	(2/5) = 0.40	(3/5) = 0.60	(4/5) = 0.80	(5/5) = 1
i/N – R(i) D+ max	0.25 – 0.05 = 0.15	0.40 - 0.14 = 0.26	0.60 – 0.44 = 0.16	- (negatief)	1 – 0.93 = 0.07
R(i) – (i-1)/N D- max	0.05 – (1-1) / 5 = 0.05	- (negatief)	0.44 – (3-1) / 5 = 0.04	0.81 – (4-5) / 5 = 0.21	0.93 – (5- 1) / 5 = 0.13

D+ max {0.15,0.26,0.16,0.07} = 0.26
D- max {0.05,0.04,0.21,0.13} = 0.21
D = max (D+, D-)
(0.26,0.21)

Dalpha = 0.05 = 0.565 (check *Kolmogorov-Smirnov Critical Values*)
0.26 < 0.565 → H0 wordt niet verworpen.

Hiermee kunnen we dus concluderen dat de gegenereerde random getallen in het voorbeeld uniform zijn omdat H0 niet wordt verworpen.

2.2 Distribution

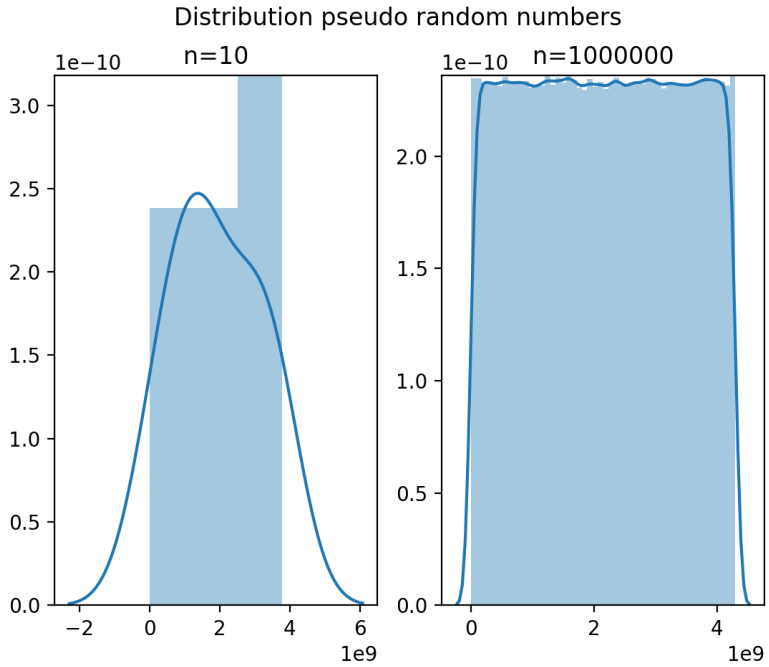
Met behulp van de *Seaborn* library kunnen we de gegenereerde data gaan visualiseren om een beter beeld te krijgen van de uniformiteit.

De *continue uniforme verdeling*[3] is een verdeling op een interval met constante kansdichtheid, wat inhoudt dat er dus geen voorkeur is voor waarden uit het interval.

[3] Bron: https://nl.wikipedia.org/wiki/Uniforme_verdeling_(continu)

Net als een dobbelsteen heeft elke zijde even veel ‘kans’ en zal dus bij elke gooi een uniforme uitkomst hebben.

Figuur 1: plot uit de code *main.py*



In de eerste plot worden 10 verschillende getallen gegenereerd met parameters:
num_steps = 10
previous = None
a = 214013
c = 2531011
*m = 2**32*

In de tweede plot worden dezelfde parameters gebruikt maar dan met *num_steps* = 1.000.000. Dit houdt dus in dat we na het genereren van één miljoen pseudo random getallen beter kunnen zien dat de getallen ook daadwerkelijk uniform zijn.

3. Pi benaderen

Hoe precies kunnen we met 1.000.000 random gegenereerde getallen tussen 0 en 1 pi (= 3.141592..) gaan benaderen? We zullen hier m = 1 gebruiken en als previous(seed) = 0.00001.

Volgens het *Monte Carlo Approach* [4] kunnen we als het ware één miljoen getallen gebruiken als coördinaat(x,y). Zo neem ik de eerste helft van mijn lijst als de x-coördinaat en de andere helft als y-coördinaat.

Een cirkel bestaat uit alle punten met dezelfde afstand (straal) r tot een middelpunt M.

$$x^2 + y^2 = r^2$$

Het gebied van de cirkel is:

$$\pi r^2$$

Het gebied van de grafiek:

$$l^2$$

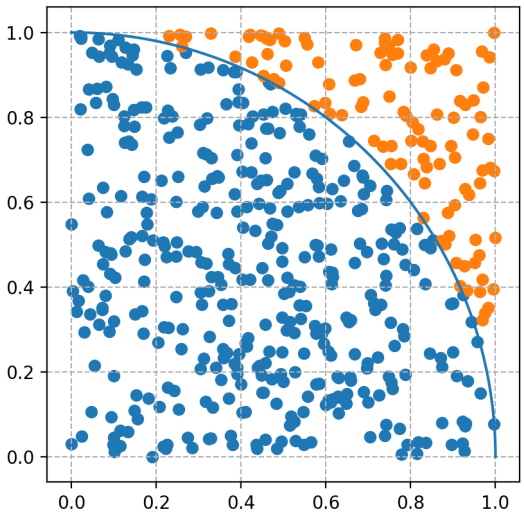
We zullen slechts een kwart van de cirkel gebruiken, daarom deel je het gebied van de cirkel door 4. Het gebied van de cirkel wordt gedeeld door het gebied van de grafiek.

$$\pi r^2 = \pi/4$$

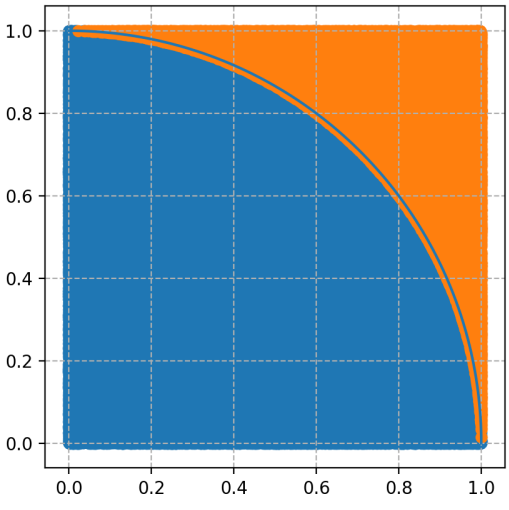
Als $\sqrt{x^2 + y^2}$ kleiner of gelijk is aan 1, dan zal het getal dus binnen de cirkel vallen. Zo niet, valt het buiten de cirkel.

Als we het aantal wat binnen de cirkel valt delen door het totaal aantal punten, krijgen we een schatting van de ratio en dit vermenigvuldigen we met 4 om de benadering van pi te krijgen.

Figuur 2: 1000 random getallen



Figuur 3: 1.000.000 random getallen



Uitkomst van de benadering: 3.140656, error van ongeveer: 0.00093665…

[4] Bron: https://academo.org/demos/estimating-pi-monte-carlo/

4. Stochastische variabelen

In hoofdstuk 1 hebben we een uniform random number generator gemaakt die pseudo random getallen genereert. In dit deel gebruik ik mijn random number generator om stochastische variabelen uit een kansverdeling te genereren.

De *Box-Muller transform*[5] door George Box en Mervin Muller is een manier om van de uniform random gegenereerde variabelen, een normaalverdeling te maken. Stel je hebt twee sets genaamd u1 en u2. Deze sets bevatten waarden van getallen tussen de 0 en 1. Door deze sets toe te passen op de onderstaande formule, genereer je normaal verdeelde getallen z1 en z2.

$$z1 = \sqrt{-2\ln(u1)}\cos(2\pi u2)$$
$$z2 = \sqrt{-2\ln(u1)}\sin(2\pi u2)$$

De normale verdeling, ook wel gaussverdeling genoemd is een kansverdeling (continu) met de twee parameters mu (verwachtingswaarde, gemiddeld genomen) en sigma (standaardafwijking) waarvan de kansdichtheid wordt gegeven door de volgende functie:

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

[5] Bron: https://en.wikipedia.org/wiki/Box–Muller_transform