(Lab)Exercise: Random Number Generators

Rein van den Boomgaard

April 2014

1 Rekenen met stochasten

Laat zien dat $E(X+Y) = EX + EY$. Hierbij mag je er *niet* vanuit gaan dat X en Y onafhankelijk zijn. Je moet uitgaan van de gezamenlijke kansfunctie $p_{XY}(x,y)$. De eerste regel van het bewijs is niets anders dan de definitie van de verwachting:

$$E(X+Y) = \sum_x \sum_y (x+y) p_{XY}(x,y)$$

d.w.z. sommeer de uitkomst $(x+y)$ maal de kans op die uitkomst $(p_{XY}(x,y))$ over alle mogelijke uitkomsten (vandaar de dubbele sommatie).

Dit bewijs geldt strict genomen alleen voor discrete stochasten. Het is echter eenvoudig te bewijzen dat (pas altijd op bij deze woorden, "its trivial to prove..." is vaak een opmaat tot vele uren puzzelplezier...) dit ook geldt voor continue stochasten. Dit bewijs hoef je niet te geven.

2 Uniform verdeelde stochasten

Laat X en Y twee onafhankelijke uniform verdeelde stochasten zijn $X, Y \sim \text{Uniform}(0,1)$.

1. Gebruik de standaard random number generator uit Numpy om uniform verdeelde getallen te genereren. Genereer N paren (X,Y) en plot deze punten in het platte vlak (je kunt de matplotlib `plot` functie gebruiken waarbij je geen verbindende lijn tekent maar een marker plot).

2. In de jaren 60 van de vorige eeuw, in de tijd dat IBM de computermarkt domineerde, was er op de IBM mainframe computers een subroutine (lees functie) RND beschikbaar welke een reeks random getallen x_k genereerde aan de hand van de volgende formule:

$$x_{k+1} = (ax_k + c) \bmod m :$$

met $a = 65539$, $c = 0$, and $m = 2^{31}$. Omdat $a = 2^{16} + 3$ kan de vermenigvuldiging worden gedaan met een shift en een optelling. Op een 32 bits machine is de modulo 2^{31} ook eenvoudig en snel uit te voeren. Deze efficientie overwegingen waren in die jaren van trage computers—mijn mobiele telefoon is sneller dan een mainframe van toen—van groot belang.

Schrijf een Python/Numpy programma om random numbers te genereren volgens de IBM methode. Herschaal de getallen zodat een reel getal in het interval van 0 tot 1 wordt gegenereerd. Herhaal de 1e opdracht maar nu gebruikmakend van de IBM random number generator. Zie je enig verschil tussen beide random number generatoren?

3. De IBM generator heeft ook nadelen. Niet alleen de verzameling van gegenereerde getallen van belang is maar ook de volgorde waarin die worden gegenereerd is van belang. Opeenvolgende trekkingen dienen (zo veel mogelijk) onafhankelijk van elkaar te zijn.

   Dat is waar het mis gaat bij de IBM generator. Bewijs dat voor deze generator:

   $$x_{k+2} = 6x_{k+1} - 9x_k$$

   Kortom als je twee opeenvolgende trekkingen observeert kun je alle volgende getallen simpelweg voorspellen.

   Als je met de IBM generator punten in 3D zou genereren en de x,y en z coordinaten zijn opeenvolgende trekkingen dan geldt dus dat $z = 6y - 9x$ (wel modulo 2^{31} blijven rekenen!). Als je dan veel punten in een 3D scatterplot zet dan zie die vlakken (vanwege de modulo rekening) terugkomen. Niet bepaald een random generator dus.

4. De IBM generator behoort tot de klasse van de "linear congruential generators". Die zijn makkelijk te implementeren en efficient. Helaas (en het kan echt beter dan de IBM instantiatie) zijn deze generatoren niet echt random. Dus niet goed bruikbaar voor online gambling (bij online gokken gaat heel wat geld om en je zou niet willen dat hackers in staat zijn om te achterhalen hoe de RNG werkt en daarmee de uitkomsten kunnen voorspellen) en ook niet voor security toepassingen. Voor games en voor vele wetenschappelijke toepassingen (stochastische simulaties) zijn ze vaak wel bruikbaar.

   Zoek uit wat voor RNG wordt gebruikt in Numpy. Waarom is die beter dan een linear congruential generator? (NB het is hier NIET vereist om de werking van de generator tot in detail te begrijpen).

   Als je meer wilt weten over random number generatoren—een onderwerp met veel meer informatica aspecten dan je op het eerste gezicht zou denken—kijk dan op de website random.org. Wikipedia is ook hier een nuttige kennisbron.

Je zou misschien denken dat een 'echte' random number generator dus niet kan bestaan. We kunnen zo'n generator inderdaad niet in rekenregels vatten: programmatisch komen we niet verder dan "pseudo random number generators." Wel kunnen we de natuur gebruiken waarin vele processen 'echt' random zijn. Aan zo'n proces kun je metingen verrichten en op die manier 'echte' random getallen genereren. Hardware random generators zijn voor een paar honderd dollar te koop als USB device. De website random.org bevat ook een webservice waarin je 'echte' random getallen kan opvragen.

3 De exponentiële verdeling

In simulaties van wachtrijen moet bijna altijd een reeks van events worden gegenereerd (bijvoorbeeld de aankomsttijden van klanten in een winkel, van auto's op een snelweg, van jobs bij een server). De verdeling van de tijdintervallen tussen twee klanten, twee auto's en twee jobs laat zich goed modeleren met de *exponentiele verdeling*. Laat X een stochast zijn die exponentieel is verdeeld, $X \sim \text{Exp}(\lambda)$ met als kansdichtheidsfunctie:

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x} & : x \geq 0 \\ 0 & : x < 0 \end{cases}$$

Zoek deze verdeling op in Wikipedia.

1. Wat zijn de verwachting en de variantie?
2. Wat is de betekenis van de λ parameter?
3. Hoe schat je de parameter λ gegeven een aantal trekkingen uit deze verdeling?
4. Maak een random number generator (in Python) voor de exponentiële verdeling uitgaande van de beschikbare generator voor de uniforme verdeling. Gebruik hierbij de "inverse transform sampling" methode.
5. Maak een histogram van een groot aantal trekkingen uit deze verdeling (met $\lambda = 1$).
6. Schat uit die trekkingen ook de λ parameter.