Komputery i los (I)

Jerzy Karczmarczuk
Zakład Informatyki, Uniwersytet w Caen, Francja

Bóg nie gra w kości.
(Albert Einstein)

Każdy, kto sądzi, że można generować liczby losowe za pomocą algorytmów arytmetycznych, jest, oczywiście, w stanie grzechu.
(John von Neumann)

1. Wstęp

Rys. 1. Fresk z Pompei: gracze w kości

Niniejsze opracowanie jest poświęcone generowaniu obiektów losowych na komputerze. „Obiekt" oznacza tutaj liczby (całkowite albo rzeczywiste), ale także wektory i inne konstrukcje wielowymiarowe, funkcje losowe, permutacje lub inne obiekty kombinatoryczne, kolory, przedziały czasowe itp. Jest to bardzo ważna dziedzina, o olbrzymiej liczbie zastosowań, więc mimo iż temat nie dotyczy bezpośrednio fizyki tylko jej metod rachunkowych, może się przydać niektórym czytelnikom. Nas oczywiście interesują zastosowania związane z fizyką, np. symulacja zjawisk fizycznych lub obliczenia statystyczne, ale nie należy zapominać, że zakres zastosowań liczb losowych jest szerszy i obejmuje kryptografię, szereg metod algorytmicznych (całkowanie Monte-Carlo, optymalizację itp.), symulację systemów komunikacyjnych, syntezę obrazów i gry komputerowe, oraz przetwarzanie i analizę tekstów w językach naturalnych.

Świat rzeczywisty, a także świat gier symulacyjnych, jest „nieokreślony", nieregularny i nieprzewidywalny i programy komputerowe należy móc dostosować do zagadnień z tym związanych. Problem jest jednak skomplikowany, gdyż tak „naprawdę", to nie wiemy, co to znaczy: „losowy" [1]. Słowo ***los*** w naszym języku posiada kilka kompletnie sprzecznych ze sobą znaczeń, a jedno z nich oznacza całkowite przeciwieństwo nieokreśloności, a mianowicie: przeznaczenie... Jednak zarówno filozoficzne *Fatum*, które (jeśli ktoś chce...) determinuje, co się z nami stanie, jak i kompletny bałagan, mają jedną

cechę wspólną: nie jesteśmy w stanie przewidzieć, co będzie za chwilę, jaką drogę wybierze elektron albo człowiek, przyszłość jest przed nami ukryta, mimo iż w prostych przypadkach równania ruchu sprawdzają się dobrze[1].

Tak więc, możemy chwilowo zastąpić słowo „losowy" przez „nieprzewidywalny". Nie odpowiada to na pytanie, dlaczego czegoś nie da się przewidzieć, i co można racjonalnie wywnioskować z danych „losowych". Pozwala jednak zauważyć, że „losowość" danych uwidacznia się, gdy dysponujemy całym ciągiem oraz pozwala stwierdzić, że analiza statystyczna jest potężnym narzędziem, które określa ścisłe i dokładne parametry tych ciągów: średnie, dyspersje, rozkłady spektralne, korelacje statystyczne itp., co pozwoliło np. określić związki pomiędzy termodynamiką a mechaniką statystyczną. Dzięki niej stworzyliśmy skuteczne narzędzia obliczeniowe takie jak metody Monte-Carlo, bez których uprawianie fizyki byłoby trudne.

Dziedzina „losowości świata" pozostaje fascynująca i przyprawia o ból głowy zarówno filozofów jak i matematyków. Można spotkać się z twierdzeniem, że losowość wynika z niemożliwości ustalenia przyczyn jakiegoś zdarzenia. Jest to raczej nieścisłe, typowe zachowania nieregularne w fizyce wynikają głównie z dwóch powodów: niestabilność równań opisujących procesy oraz z jednoczesnego wpływu wielu konkurujących przyczyn. (Dodajmy do siebie wiele różnie wyglądających funkcji, prawdopodobnie otrzymamy okropny bałagan, funkcję wyglądającą na losowy szum...) Obie kategorie: niestabilność i mieszanie pozwalają symulować bardzo dobrze procesy losowe na komputerze, z pełną świadomością, że jest to tylko modelowanie. Nie będziemy zajmować się prawdziwymi liczbami losowymi, generowanymi przez procesy fizyczne.

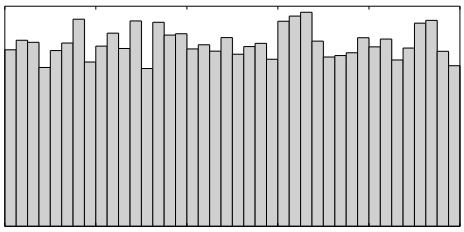
Artykuł jest przeznaczony dla Czytelnika niebojącego się prostych wzorów z analizy i mającego pewne pojęcie o własnościach prawdopodobieństw; probabilistyka jest podstawową teorią pozwalającą mówić o zmiennych losowych. Nie podajemy programów komputerowych, są one proste i krótkie.

2. Co to jest liczba losowa?

Liczba 0,75000 = 3/4 intuicyjnie „nie zasługuje" na miano liczby losowej. Ale czy 0,415926535897932384626... można nazwać liczbą losową (powiedzmy, w zakresie 0–1)? Wygląda ona „byle jak", bez widocznych na pierwszy rzut oka regularności. Można jednak wypowiedzieć tezę, że pytanie o losowość jednej liczby nie ma sensu. Jak stwierdziliśmy, losowość uwidacznia się, gdy możemy mówić o rozkładzie prawdopodobieństwa, gdy mamy do czynienia z całym ciągiem, np. że ciąg (skończony, np. 100 000) liczb: ..., 0,53951, 0,80829, 0,28072, 0,41163, 0,43648, 0,48126, 0,01840, 0,67538, ... itd. spełnia pewne

[1] Nie twierdzimy, że równania w skomplikowanych przypadkach się nie sprawdzają, ale dla skomplikowanych równań dla wielu ciał trajektorie są niestabilne, nawet najmniejsze błędy narastają lawinowo i *dokładne* sprawdzenie czegokolwiek jest niezwykle trudne...

kryteria statystyczne. Sprawdzamy, że średnia $\langle x \rangle = \frac{1}{N}\sum_{k=1}^{N} x_k$ jest bliska 0,5, że wariancja $\langle x^2 \rangle - \langle x \rangle^2$ wynosi ok. 1/12, co odpowiada rozkładowi równomiernemu na odcinku 0–1, że rozrzut tych liczb wygląda na losowy (co pokazuje histogram na rys. 2), i to fizykowi na ogół wystarczy. Może użyć tego ciągu do symulacji lub do Monte-Carlo. (Niektóre zastosowania liczb, wektorów i funkcji losowych zostaną omówione później). To, czy ciąg jest „rzeczywiście" nieprzewidywalny, utworzony dzięki jakiemuś mechanizmowi fizycznemu opartemu, np. o szum termiczny, czy jest stablicowany, czy jest wynikiem prostego algorytmu, nie ma w zasadzie znaczenia. Testy odróżniające dobre generatory od złych są zwykle oparte o zdrowy rozsądek. Ba, o ile w kryptografii zależy nam, aby ukryć mechanizm generacji i maksymalnie utrudnić przewidzenie następnych elementów ciągu, w technikach obliczeniowych fizyki korzystnym jest, aby móc dokładnie powtórzyć daną sekwencję losową na innym komputerze, w innym języku itp., co pozwoli mieć więcej zaufania do stabilności i rzetelności użytych metod numerycznych.



Rys. 2. Histogram rozkładu równomiernego

Stanisław Ulam (1909–1984), wybitny amerykański matematyk polskiego pochodzenia, wprowadził do fizyki metody Monte-Carlo

Precyzyjne metody algorytmiczne, pozwalające generować długie ciągi liczb losowych umożliwiają zwykle przerwanie generacji i jej wznowienie od punktu przerwania, co jest też ważnym elementem statystycznej jakości ciągu, gdy program musi być wykonywany na raty. Tak więc, **w większości przypadków, w ciągach liczb losowych używanych przez fizyków nie ma nic „naprawdę" losowego!** Wyglądają one na losowe, i to wszystko. Zwykle nazywa się je pseudolosowymi, i tylko nimi będziemy się dalej zajmować.

Zanim jednak przejdziemy do prostych algorytmów generacji tych ciągów, uczciwość nakazuje przeanalizowanie pewnego paradoksu: stwierdziliśmy, że mówienie o losowości pojedynczych liczb ma mało sensu. Jednak 0,75 i 0,4159265358979... wyglądają bardzo różnie, ta druga „może wyglądać" na losową. Pierwsza nie, w dwójkowym układzie pozycyjnym ten ułamek ma kształt 0,1100000000... co na zdrowy rozsądek z losowością nie ma nic wspólnego. Jednak wyrażenie $\sin 0{,}75 = 0{,}68163876002333412\ldots$ jest bardziej nieregularne, cyfry są rozrzucone bezładnie. Nasza poprzednia przykładowa liczba losowa nie została wymyślona, jest to $10(\pi - 3{,}1)$. Możemy więc rozważać losowość pojedynczych liczb rzeczywistych, traktowanych jako *ciągi cyfr*,

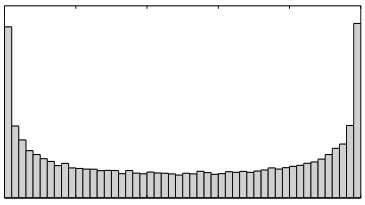
w dowolnym układzie pozycyjnym. O ile liczby wymierne w dowolnym systemie pozycyjnym są ułamkami okresowymi, liczby przestępne (np. π), są „losowymi" ciągami cyfr, spełniającymi szereg testów statystycznych. Okazuje się, że praktycznie wszystkie wielkości pomiarowe (oraz wynikające z obliczeń) w fizyce, mają taki charakter. Świat liczb w przyrodzie wygląda na „losowy".

W teorii liczb losowych spotyka się „definicję" (nieścisłą) ciągów losowych jako takich, których nie da się skompresować. Żadna transformacja, żaden algorytm *odwracalny* (bezstratny!) nie skróci ciągu losowego. O ile metody kompresji tekstów czy obrazów dają bardzo dobre wyniki, pozwalające na kilkukrotne zmniejszenie długości dokumentów na dysku lub w sieci, po kompresji dokument wygląda jak szum i skrócić go powtórnie już się nie da. Algorytmy kompresji funkcjonują tylko dla ciągów charakteryzujących się dużą regularnością. Tym niemniej, ciągi otrzymywane przez proste algorytmy są statystycznie akceptowalne, mimo że sam algorytm, zwykle krótki, może być traktowany jako „kompresja" generowanego ciągu. Tak więc należy odróżnić stosowalność liczb losowych do symulacji, optymalizacji itp., od pewnych własności teoretycznych generowanych ciągów. Fizyk chcący posługiwać się generatorami algorytmicznymi musi wykazać pewną nonszalancję i oderwanie od faktu, że jego generatory dają wyniki w pełni przewidywalne.

Niektórzy filozofowie nauki zastanawiają się czy wartości bezwymiarowych stałych przyrody, np. ładunku elektronu, są wynikiem „przypadku" (a jeśli tak, to *jakiego*?) czy są dane przez wzór matematyczny. Jednak jakakolwiek byłaby odpowiedź na to pytanie, liczba ta prawdopodobnie przejdzie przez testy losowości (czego nie zrobiono, gdyż znamy ją niezbyt dokładnie).

3. Metody kongruencyjne generowania liczb losowych

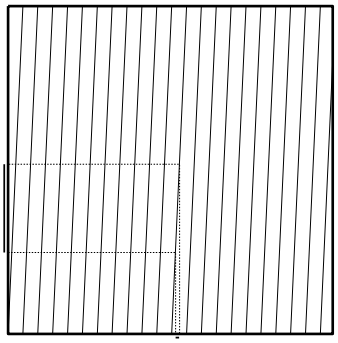
W rachunkach także spotykamy liczby o charakterze losowym! Wiele ciągów $x_0, x_1, x_2, x_3, \ldots, x_n, \ldots$ zadanych wzorami typu $x_n = f(x_{n-1})$, zachowuje się w ten sposób. Niektóre równania ewolucji z dyskretnym czasem, np. tzw. odwzorowanie logistyczne: $x_{n+1} = \lambda x_n(1 - x_n)$ dla $\lambda = 4$ generują chaos, kolejne wartości x są rozrzucone bezładnie między 0 a 1. Ten model był już dyskutowany w *Fotonie*. Nie służy on jako (typowy) generator liczb losowych, gdyż rozkład częstości liczb przezeń generowanych nie jest równomierny (lub jednorodny). (Dobrym, ambitnym ćwiczeniem jest wykazanie, że proces logistyczny dla $\lambda = 4$ daje rozkład $p(x) = 1/\left(\pi\sqrt{x(1-x)}\right)$ dla x między 0 a 1).



Rys. 3. Rozkład wyników równania logistycznego

Jednak od dawna wiadomo, a w latach 50. XX wieku Lehmer opracował to dokładnie, że prosta arytmetyka operacji na skończonym przedziale (reszt):

$x_{n+1} = R \cdot x_n \bmod M$, generuje zupełnie niezłe ciągi losowe, i ta metoda, zwana kongruencyjną, stanowi podstawę większości używanych generatorów na świecie, gdyż jest bardzo szybka. (Metody dające lepsze generatory są powolniejsze). Wystarczy wziąć dużą liczbę *R*, np. 987171,1679, wartość początkową *x* między 0 a 1, pomnożyć *R* przez *x*, a następnie przez wyniki operacji i odrzucać za każdym razem część całkowitą wyniku. Tak otrzymaliśmy nasz pierwszy ciąg losowy w rozdziale 2. Dlaczego to działa?



Rys. 4. Funkcja $y = R \cdot x \bmod 1$

Jak widzimy na wykresie tej funkcji na rys. 4, (ze znacznie mniejszym *R*, w przeciwnym wypadku linie byłyby praktycznie pionowe i kwadrat byłby wypełniony do nieczytelności), kwadrat 0–1 jest dość gęsto wypełniony bardzo stromymi liniami. Bardzo wąski przedział *x* jest odwzorowany w znacznie szerszy, funkcja jest *niestabilna*. Nawet maleńkie odchylenie od jakiejś wartości (także błędy zaokrąglenia itp.) narastają lawinowo, a ponieważ obrazem funkcji jest zawsze skończony przedział 0–1, przedział ten jest iteracyjnie „zwijany" do odcinka 0–1 i szatkowany na mnóstwo fragmentów. W wyniku wielu iteracji dostajemy rozrzut wyglądający na chaotyczny. Jest to nienajgorszy generator, bardzo prosty do zaprogramowania.

Nie jest on jednak wykorzystywany „zawodowo", gdyż arytmetyka na liczbach rzeczywistych (zmiennoprzecinkowych) nie jest łatwa do porównywania na różnych komputerach. Liczba miejsc znaczących i algorytmy zaokrąglania mogą się różnić, więc taki generator nie byłby przenośny i trudniejszy do analizy. Możemy źle wybrać czynnik mnożący i dostać generator nie nadający się do niczego. (Historycznie operacje rzeczywiste były także o wiele powolniejsze niż całkowite). Standardem więc stały się metody kongruencyjne oparte na liczbach całkowitych. Należy wybrać (w miarę możności dużą) liczbę całkowitą *M* zwaną modułem generatora, jakąś wartość początkową X_0 z przedziału $0 : M - 1$, a następnie iterować $X_{m+1} = A \cdot X_m + C$. Aby uzyskać liczbę rzeczywistą z przedziału 0–1, wystarczy wyniki dzielić przez *M*. Zero będzie należeć do generowanego przedziału, a dokładnie 1 – nie.

Algorytm ten może oczywiście wygenerować tylko *M* różnych liczb, potem będą się one powtarzać.

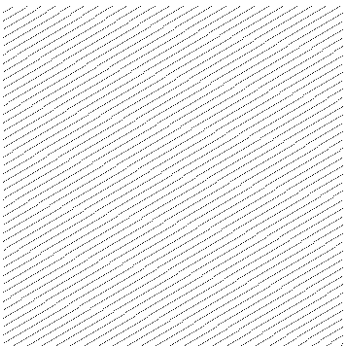
Stałe *A* i *C* muszą spełniać pewne kryteria oparte na teorii liczb, w przeciwnym wypadku generator będzie powtarzał krótkie, nieużyteczne serie. Weźmy $M = 100,\ A = 27,\ C = 17,\ X_0 = 0$. Ciąg liczb dla tego generatora to 17, 76, 69, 80, 77, 96, 9, 60, 37, 16, 49, 40, 97, 36, 89, 20, 57, 56, 29, 0, po czym seria się powtarza. Ciąg zawiera tylko 20 różnych wartości, a można było trafić jeszcze gorzej. Dla $A = 30$, ciąg wynikowy to 17, 27, po czym liczba 27 powtarza się w nieskończoność. Nie można bez zastanowienia wybrać byle jakich para-

metrów. Dla prostego generatora kongruencyjnego parametry powinny spełniać następujące kryteria (patrz książka Knutha):

- *M* i *C* winny być względnie pierwsze (nie posiadać wspólnego podzielnika), wyjątkiem jest $C = 0$ omówione poniżej.
- Definiujemy $B = A - 1$. Dla każdej liczby p, będącej podzielnikiem (pierwszym) *M*, musi ona być także podzielnikiem *B*.
- Jeżeli *M* dzieli się przez 4, także i *B* musi być wielokrotnością 4.

Ponieważ $100 = 2 \cdot 2 \cdot 5 \cdot 5$, liczba *B* musi dzielić się przez 20. $A = 61$ jest więc dobrym parametrem, dostaniemy 100 różnych liczb w ciągu. Ale jakość generatora zależy od tego co chcemy z nim zrobić! Po podzieleniu przez *M* dostaniemy *M* liczb rzeczywistych, które można wykorzystać do symulacji, gier, itp. W żadnym jednak wypadku nie można wykorzystać ciągu *X* do generacji losowych bitów przez wzięcie reszty z dzielenia przez 2. Dostaniemy zawsze 0, 1, 0, 1, 0,... i podobnie rzecz będzie się miała z parami bitów – resztami z dzielenia przez 4.

Często, ze względu na wygodę, przyjmuje się $C = 0$, co wymaga innych kryteriów spełnianych przez pozostałe parametry. Dla czytelnika nieobeznajmionego z teorią liczb, kryteria te będą niezrozumiałe, więc ich nie podamy. Chętni winni skorzystać ze stablicowanych propozycji, dostępnych w książkach i Internecie. Jednym z typowych zestawów parametrów jest $M = 2^{31} - 1 = 2147483647$, $A = 7^5 = 16807$. *M* jest tzw. liczbą pierwszą Mersenne'a. Ciąg generowany przez taki generator ma długość maksymalną $M - 1$. (Oczywiście nie można zaczynać od $X_0 = 0$). Ogólnie, okres jest maksymalny jeśli *M* jest liczbą pierwszą. Często jednak wykorzystuje się $M = 2^K$, co nie gwarantuje najlepszej jakości statystycznej, ale jest szybkie, gdyż reszta z dzielenia przez *M* jest otrzymywana tanio (odrzuca się bity wyższych rzędów), a gdy *K* jest długością słowa maszynowego przechowującego liczby całkowite, np. $K = 31$, wtedy za darmo: ignoruje się bity nadmiaru przy operacjach arytmetycznych. Dla tej wartości *M*, *A* musi spełniać warunek, że reszta z dzielenia przez 8 jest równa 3 lub 5. Okres generatora jest wtedy równy 2^{K-2}.



Rys. 5. Dwuwymiarowy rozkład utworzony przez generator kongruencyjny

Może także (lecz nie musi) wystąpić dodatkowy kłopot, a mianowicie *korelacje* między sąsiednimi liczbami trudniejsze do wykrycia na pierwszy rzut oka. Weźmy generator z $M = 2^{16}$ i z rozsądnym *A*. Ciąg generowanych liczb wygląda rozsądnie, ale jeśli utworzymy pary kolejnych liczb i potraktujemy je jako współrzędne w kwadracie 0–1, powstały obraz jest regularny. Tworząc wektory 3- lub więcej wymiarowe,

możemy otrzymać obrazy jeszcze regularniejsze, a ponieważ rozkłady wielowymiarowe w fizyce są bardzo potrzebne, proste generatory dostępne w standardowych bibliotekach różnych implementacji większości języków programowania są mało przydatne. Problemowi można zaradzić na kilka sposobów, np. przez użycie wielu generatorów z różnymi parametrami, albo nawet tylko dwóch generatorów: głównego, który wypełnia tablicę (kilkadziesiąt lub kilkaset elementów) i pomocniczego, który losuje jeden z elementów tej tablicy. Po dostarczeniu tego elementu jako wyniku, generator główny „zapełnia lukę". W ten sposób sąsiednie wyniki generatora głównego są od siebie oddalone i pomieszane.

4. Inne generatory jednorodne

W 1997 roku Makoto Matsumoto i Takuji Nishimura wymyślili dość szybki generator noszący nazwę *Mersenne Twister*, o okresie $2^{19937}-1$ (ponad 10^{6000}), losowo rozrzucony w przestrzeniach aż do 623 wymiarów. Stał się on profesjonalnym standardem [3], ale w typowych zastosowaniach (zwłaszcza dla prostych symulacji oraz gier) jest on ciężki w implementacji, wymaga wewnętrznej tablicy o ponad 600 elementach i kodu zajmującego pół stronicy. Jeżeli wyniki zostaną skonwertowane do postaci całkowitej, albo zmiennoprzecinkowej 32- lub 64-bitowej, przy tak olbrzymim okresie, liczby będą się powtarzać znacznie częściej, ale powtórzenie jednej liczby nie oznacza, że cała sekwencja się powtórzy – następne wyniki będą rozrzucone zupełnie inaczej. Algorytm Mersenne Twistera jest zbyt skomplikowany, aby go tu przytaczać, niech jednak czytelnik zda sobie sprawę, że tworzenie nowych wyrafinowanych generatorów nadal postępuje! Naukowiec, czy inżynier, lub twórca gier komputerowych, który powie sobie, że przecież jego ulubiony język programowania posiada bibliotekę zawierającą jakiś generator liczb losowych i nie ma sensu uczyć się metod generacji, ryzykuje miano partacza.

4.1. Generatory oparte o uogólnienia ciągu Fibonacciego

Uogólnieniem metod kongruencyjnych są algorytmy kombinujące kilka członów wygenerowanego ciągu. Ponieważ znany jest ciąg Fibonacciego, zdefiniowany wzorem $F_0 = 0, F_1 = 1,\ F_n = F_{n-1} + F_{n-2}$, ciągi oparte o regułę $S_n = S_{n-j} + S_{n-k}$ są również zwane uogólnionymi ciągami Fibonaciego, i wzór

$$S_n = S_{n-j} + S_{n-k} \pmod{M}, 0 < j < k$$

bywa nazywany generatorem Fibonacciego. Zamiast dodawania można użyć innych działań, np. mnożenia. Ten generator wymaga dodatkowej pamięci na przechowanie poprzednich wyników, oraz inicjalizacji (innym generatorem), ale jest szybki i bywa używany w niektórych programach. Oto kilka wariantów par (j, k) podawanych w literaturze:

$(j = 24,\ k = 55),\ (j = 31,\ k = 63),\ (j = 38,\ k = 89), (j = 83, k = 258),$

Dla małych wartości parametrów okres generatora jest również niewielki.

Od Redakcji:

Ciąg Fibonacciego

Ciąg Fibonacciego jest spektakularnym dowodem na głębokie związki pomiędzy naturą i kulturą a matematyką. Ciąg Fibonacciego zaczyna się od liczb 1, 1 a każdy kolejny wyraz równa się sumie dwóch wyrazów poprzednich: **1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,…** Liczby będące wyrazami ciągu noszą nazwę liczb Fibonacciego.

Liczby ciągu Fibonacciego mają zastosowanie do opisu notowań giełdowych, do opisu budowy roślin, np. słonecznika, ananasa, szyszki sosny, do opisu wzrostu liczby królików, pszczół. Liczby Fibonacciego odnajdujemy w architekturze, w poezji, a także w muzyce. Dla przykładu: liczba płatków wielu kwiatów jest jedną z liczb Fibonacciego: jaskry mają 5, kwiaty sangwinarii 8, a astry często 21.

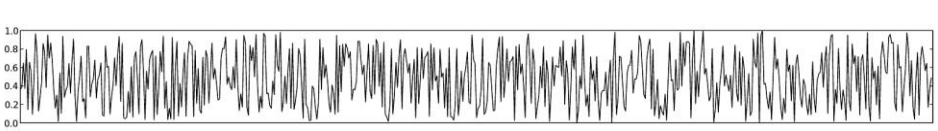
4.2. Funkcje ergodyczne

Podstawową cechą algorytmu generatora jest jego „niestabilność", następna liczba ma „mało wspólnego" z poprzednią. Możemy sobie więc wyobrazić funkcje, które nie wymagają procesu iteracyjnego (przykładania jej do poprzedniego wyniku), ale takie, które dla $x = 1, 2, 3, \ldots$ dostarczają $f(x)$ chaotycznie. Takie funkcje znajdują zastosowania w informatyce, w bazach danych, w kodowaniu itp., i noszą nazwę funkcji mieszających. Pozwalają one przekształcić ciąg liter (np. tytuł książki) na stosunkowo krótką liczbę, służącą do indeksowania tablicy tytułów, co przyspieszy szukanie (potem trzeba sprawdzić zgodność, gdyż może zajść kolizja – ten sam wynik odpowiada różnym tytułom). Fizyk nazwie takie funkcje ergodycznymi, proces ergodyczny to taki, w którym zachowanie układu zależy bardzo słabo od warunków początkowych. Oto przykład takiej funkcji, która dla $n = 1, 2, 3, \ldots$ generuje ciąg pseudo-losowy liczb 32-bitowych z przedziału 0 – 1:

$$x = (n \cdot 2^{13})\text{^}n$$

$$f = 1 - ((15731\, x^3 + 789221\, x + 1376312589)\ \&\ (2^{31} - 1))/2^{31}$$

Operator (^) oznacza alternatywę wyłączającą (różnicę symetryczną) dla każdego bitu liczb całkowitych będących argumentami: 0 jeśli dwa bity są identyczne, 1 w przeciwnym wypadku. Operator (&) jest iloczynem logicznym bitów; tutaj służy do otrzymania reszty z dzielenia przez 2^{31}. Na rys 6. widzimy wykres funkcji $f(n)$ dla kilkuset sąsiednich wartości *n*:



Rys. 6. Funkcja „ergodyczna"

Oczywiście można posłużyć się innym zestawem parametrów liczbowych, ale taki generator należy wszechstronnie przetestować przed użyciem, gdyż nie ma prostych metod teoretycznych oceny ich jakości. Ich zaletą w porównaniu z generatorami „klasycznymi" jest brak wewnętrznej pamięci. Niczego nie potrzeba pamiętać, generator jest „czystą" funkcją matematyczną, wymaga tylko jakiegoś argumentu. (Nie jest to jednak rozwiązanie naturalne, gdyż ten argument funkcji ergodycznej jest fikcyjny, pozbawiony sensu).

5. Kilka testów jakości generatorów

Podstawowy test to sprawdzenie, czy rozkład się zgadza z teoretycznym, w przypadku rozkładu równomiernego – czy histogram jest płaski. Można to zrobić „ręcznie", wizualnie, albo sprawdzić, że średnie wysokości kolumn i ich dyspersja nie odbiegają od norm. Tutaj zaczyna się problem, którego w tym artykule nie możemy poruszyć: co to jest „norma", gdyż oczywiście będą odchylenia, a nie ma tu miejsca na omówienie testów χ^2 itp. Zainteresowani Czytelnicy winni sięgnąć do literatury dotyczącej testów. Książka Knutha [2] podaje wiele użytecznych, łatwych w zastosowaniu.

Można sprawdzić najpierw numerycznie podstawowe momenty, średnią, dyspersję, czy asymetrię rozkładu, co nie wymaga tworzenia wykresów.

To nie wystarczy, sekwencja $x_n = n \cdot \alpha \bmod 1$ dla „byle jakiego" α rzeczywistego, „nieregularnego" (tj. nie będącego prostą liczbą wymierną), przejdzie ten test. (Ten generator jest równoważny kongruencyjnemu, z $A = 1$, $C = 0$). Liczby będą równomiernie rozłożone w przedziale 0–1, ale będą silnie skorelowane.

Następnym testem będzie więc sprawdzenie, czy nie ma ewidentnych korelacji między kolejnymi liczbami. Dwuwymiarowy wykres tej liniowej sekwencji dostarczy nam kresek, jak na rys. 5. Nie możemy omówić tutaj bardziej skomplikowanych testów, np. spektralnych, korelacji w wielu wymiarach itp., ale zagadnienie jest ważne!

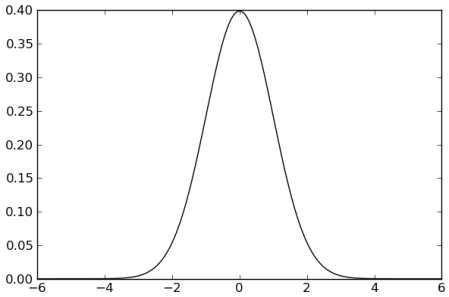
6. Rozkłady nierównomierne

Mimo, iż generatory jednorodne stanowią punkt wyjścia, bezpośrednie ich użycie do zagadnień fizycznych jest niewielkie. W fizyce takie rozkłady prawdopodobieństwa występują dość rzadko. Oto kilka innych rozkładów, które mogą nas interesować:

1. Rozkład „równomierny" w nieskończonym przedziale, np. nieograniczony ciąg zdarzeń, np. emisji cząstek przez substancję promieniotwórczą, albo przez osłabiony laser, itp. Zadana jest stała gęstość tych wydarzeń w czasie, ale nie da się tego sprowadzić do jednorodnego rozkładu w skończonym przedziale, przedział czasowy jest od zera do nieskończoności. Zwykle ten mechanizm probabilistyczny nazywa się procesem Poissona. Problem gene-

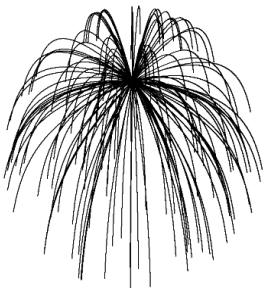
racji należy sformułować przy użyciu wielkości skończonych, np. generujemy czas oczekiwania na następne wydarzenie.

2. Fluktuacje wielkości fizycznych wokół średniej. To jest jeden z najbardziej rozpowszechnionych rozkładów, w którym małe odchylenia od średniej są znacznie prawdopodobniejsze od dużych, ale dowolne są możliwe, przedział jest nieskończony. Ten rozkład jest zadany funkcją Gaussa, $\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-(x-\langle x\rangle)^2/\,2\sigma^2\right)$, gdzie σ jest dyspersją. Dla średniej równej zeru i dyspersji równej jeden, ten rozkład nazywamy normalnym. Pakiety do obliczeń naukowych zwykle zawierają wbudowany generator tego rozkładu. Bywa on punktem wyjścia do generowania typowych funkcji losowych (szumów).



Rys. 7. Krzywa normalna Gaussa

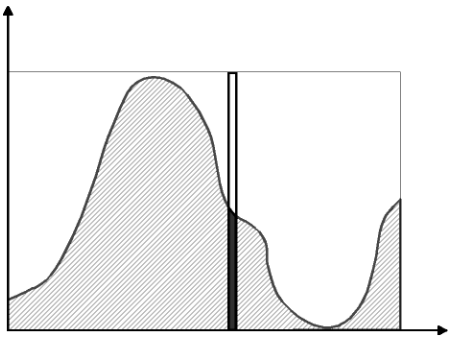
3. Wiele rozkładów o charakterze geometrycznym, zwykle wielowymiarowych. Np. jednorodny rozkład kierunków w przestrzeni (lub: jednorodny rozkład punktów na powierzchni sfery, ewentualnie wewnątrz kuli), co może służyć do symulacji i wizualizacji wybuchów, fajerwerków itp. Mimo „jednorodności", te rozkłady są odmienne od jednorodnych rozkładów w przedziale.


Rys. 8. „Sztuczne ognie"

Czasami mamy do czynienia z rozkładem prawdopodobieństwa określonym doświadczalnie, z mierzonych częstości jakiegoś zjawiska. Używamy wtedy rozkładu stablicowanego, albo przybliżonego jakąś funkcją (wielomianem, wymierną, kombinacją funkcji wykładniczych itp.). Wtedy pojawia się konieczność stworzenia algorytmu bardzo uniwersalnego, zdolnego do generowania dowolnych rozkładów. Metody generacji rozkładów dyskretnych: $x_1, x_2, \ldots\, x_k, \ldots$ dla k całkowitego, z rozkładem prawdopodobieństwa p_k, różnią się od algorytmów ciągłych, gdy gęstość rozkładu prawdopodobieństwa $p(x)$ jest funkcją, mimo iż w informatyce wszystko jest dyskretne. Rozważymy więc różne metody algorytmizacji dla tych generatorów.

6.1. Metoda eliminacji (lub odrzutu)

Przypuśćmy, że gęstość prawdopodobieństwa $p(x)$ jest zadana pewną funkcją w skończonym przedziale i bez osobliwości, tak, że można ją zamknąć w prostokącie.



Rys. 9. Metoda odrzutu

Algorytm generacji wymaga dwóch liczb losowych o rozkładzie jednorodnym. Pierwsza wybiera x między zadanymi granicami. To nie wystarczy, gdyż – jak widać na rys. 9 – liczby nieco na lewo od połowy przedziału winny pojawiać się częściej niż wartości na prawo. Teraz losujemy drugą liczbę, „pionową” r, z przedziału od dolnej do górnej granicy prostokąta zawierającego wykres. Spójrzmy na kolumnę, odpowiadającą wylosowanemu x-owi. Jeżeli r jest mniejsze od $p(x)$ dla tej wartości x, wartość ta zostaje zaakceptowana, generator zwraca ją jako wynik. Jeżeli r jest większe – wartość x zostaje odrzucona i generator powtarza całą operację.

Widać wyraźnie, że w pobliżu maksimum funkcji, r prawie zawsze będzie mniejsze od $p(x)$, więc prawie wszystkie wartości x zostaną zaakceptowane. W pobliżu minimum, losowane wartości „pionowe” będą na ogół większe od wartości gęstości prawdopodobieństwa i zaakceptowanych wartości x będzie znacznie mniej.

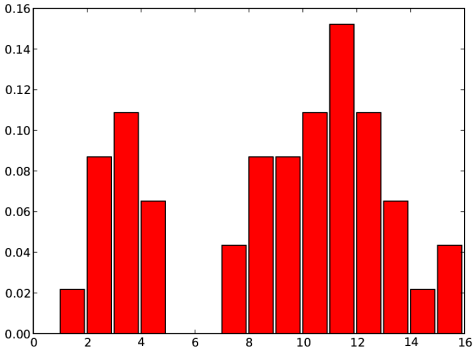
Metoda jest łatwa w zastosowaniu i często wystarczająco efektywna, mimo „strat” odrzuconych punktów. Jednak te straty mogą też być zbyt poważne. Jeśli rozkład $p(x)$ ma ostre piki, a większość powierzchni prostokąta leży nad wykresem funkcji, większość wygenerowanych punktów zostanie odrzucona.

Omówiona technika w naturalny sposób uogólnia się na rozkłady wielowymiarowe, ale tutaj straty mogą być bardzo znaczne. Nie każdy zdaje sobie sprawę, że w większej liczbie wymiarów więcej przestrzeni się „marnuje”. Na przykład, jeśli chcemy wygenerować punkty losowe w kole o średnicy 1, możemy generować punkty (x, y) z kwadratu 0–1, a następnie odrzucać wszystkie punkty na zewnątrz koła. Stracimy ok. 21 procent punktów, powierzchnia koła zajmuje 0,785 powierzchni kwadratu. Ale trójwymiarowa kula zajmuje tylko 0,52 objętości kostki jednostkowej, a w czterech wymiarach ok. 0,31, i to się pogarsza, funkcje o wykresach „jakichkolwiek”, nieregularnych, również zajmują mały procent przestrzeni. Można zadać sobie pytanie po co rozkłady w przestrzeniach o dużej, „niefizycznej” liczbie wymiarów. Jest to bardzo potrzebne fizykowi i zostanie omówione później.

6.2. Metoda odwracania dystrybuanty

Zadajmy pytanie: czy w oparciu o daną funkcję rozkładu $p(x)$ można utworzyć taką funkcję $h(r)$, że dla r losowanego z rozkładu równomiernego, którego generację mamy opanowaną, jej wynik jest rozłożony z gęstością $p(x)$. Jest to

możliwe, tylko dosyć skomplikowane, więc ta metoda daje generatory powolniejsze od jednorodnych.



Rys. 10. Dyskretny rozkład prawdopodobieństwa

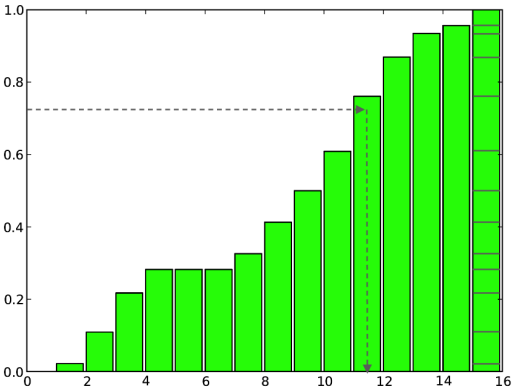
Zaczniemy od przypadku dyskretnego, od generacji liczb całkowitych k z jakiegoś przedziału. Jest to ważny przykład także dla przypadku ciągłego – pozwoli lepiej zrozumieć algorytm. Rysunek 10 przedstawia dowolny rozkład prawdopodobieństwa liczb k od 0 do 15. (Puste kolumny dla $k = 5, 6$ oznaczają prawdopodobieństwo zero). Algorytm losowania jest intuicyjnie bardzo prosty: „rzucamy" losowo punkt na ten wykres i jako wynik traktujemy współrzędną poziomą trafionej kolumny. Liczy się trafienie wnętrza prostokąta, więc liczby 0, 5 oraz 6 nie mają szans, a największe szanse ma kolumna 11. Praktyczna realizacja tego algorytmu wymaga przygotowania. W oparciu o ciąg

$$p_k = 0,\ 0{,}022,\ 0{,}087,\ 0{,}109, \ldots$$

tworzymy tzw. *dystrybuantę* tego rozkładu, albo *rozkład kumulatywny*:

$$d_k = \sum_{j=0}^{k} p_k \,.$$

Wykres tego rozkładu przedstawia rys. 11. Jest to funkcja niemalejąca, gdyż wszystkie p_k są nieujemne.



Rys. 11. Rozkład kumulatywny

Zwróćmy uwagę na ostatnią kolumnę, możemy ją interpretować jako zbiór wszystkich kolumn z rys. 10, postawionych pionowo, jedna na drugiej, z zachowaniem unormowania: suma ich wysokości równa się 1. Teraz algorytm losowania staje się prawie oczywisty. Losujemy standardową liczbę r z rozkładu jednorodnego między 0 a 1. Wylosowany segment określa wynik otrzymany przez generator. Konkretnie, realizujemy następujące operacje:

1. Losujemy r.
2. Wybieramy $k = 0$.
3. Sprawdzamy, czy $r < d_k$. W przypadku odpowiedzi pozytywnej, k jest wynikiem.
4. W przeciwnym wypadku zwiększamy k o 1 i wracamy do punktu 3.

Ponieważ dla ostatniego k z przedziału d_k jest równe 1, algorytm na pewno się zakończy. Przerywane linie na rysunku obrazują geometrię postępowania. Po wylosowaniu r na osi pionowej, kreślimy linię poziomą, aż do przecięcia się z jakąś kolumną. Widać, że kolumny 5 i 6 nie mają szans.

Jeżeli prawdopodobieństwa można przybliżyć liczbami wymiernymi wybranymi z niezbyt długiego ciągu, procedurę można przyspieszyć. Np. 16 wartości z przykładu powyżej są znormalizowanymi częstościami: [0, 1, 4, 5, 3, 0, 0, 2, 4, 4, 5, 7, 5, 3, 1, 2]. Wystarczy więc utworzyć tablicę o 46 elementach, zawierającą raz liczbę 1, 4 razy liczbę 2, ..., 7 razy liczbę 11, itd., a następnie wylosować równomiernie jeden z elementów tej tablicy.

Zauważmy, że z matematycznego punktu widzenia wybranie argumentu na osi pionowej i otrzymanie wyniku na osi poziomej jest równoważne transpozycji: obróceniu wykresu wokół przekątnej i potraktowanie go jako zwykłej funkcji (tylko dyskretnej). Tak postępuje się w przypadku ciągłym. Dystrybuanta wyrazi się całką: $d(x) = \int^x p(x')dx'$. Dolna granica całkowania jest dolną granicą przedziału. Na przykład, jeśli $p(x) = 2x$ w przedziale 0–1, wtedy $d(x) = x^2$. Algorytm generacji sprowadzi się do $y = \sqrt{r}$.

Dla odwzorowania logistycznego $p(x) = \frac{1}{\pi\sqrt{x(1-x)}}$, dystrybuanta wyraża się funkcją $d(x) = \frac{1}{\pi}\arcsin(2x-1) + \frac{1}{2}$. Funkcja generująca to $y = \frac{\sin\left(\pi(r-\frac{1}{2})\right)+1}{2}$. Widać, że samo odwzorowanie logistyczne jest szybsze niż taki generator. Tak więc, dla niektórych rozkładów może się opłacić znalezienie nieliniowych odwzorowań, które je generują bezpośrednio, ale niewiele takich znamy.

6.3. Proces Poissona

Wspomnieliśmy, że pojęcie rozkładu równomiernego odnosi się do przedziałów skończonych. Jednak ważnym i ciekawym przypadkiem jest generowanie niezależnych wydarzeń w nieskończonym czasie, od zera do ∞. Może to być model emisji cząstek przez radioaktywną substancję (w mierzonym czasie nieporównywalnie krótszym niż półokres rozpadu, aby gęstość cząstek była stała w czasie), model ruchu pojazdów wjeżdżających losowo, co jakiś czas na autostradę, albo model odbiorów zleceń przez serwer obsługujący publiczne stronice internetowe itp. Gęstość „wydarzeń" na jednostkę czasu jest stała, a wydarzenia są nieskorelowane. Teoretycznie można wybrać jakiś skończony czas końcowy i potraktować model jako przypadek rozkładu równomiernego, jest to jednak

wysoce nienaturalne! Kolejne wydarzenia są uporządkowane w czasie, zaś liczby losowe w przedziale – nie.

Pytanie, na które generator tego procesu winien odpowiedzieć, brzmi: kiedy nadejdzie następne zdarzenie. Podstawową i nieco paradoksalną własnością procesu jest fakt, że odpowiedź jest niezależna od momentu, w którym zaczynamy liczyć czas. Dlaczego paradoksalną? Często zadaję moim studentom następującą łamigłówkę. W pewnym mieście autobusy nie mają rozkładu jazdy, wiadomo, że przejeżdżają przez pewien przystanek ze stałą gęstością w czasie, powiedzmy średnio co 10 minut, 6 na godzinę. Średnia odległość w czasie między autobusami wynosi więc 10 minut. Przychodzimy na przystanek o losowej porze. Jaki jest nasz średni czas oczekiwania na następny autobus?

Typowo pada odpowiedź: 5 minut, a rozumowanie jest następujące: mamy tę samą szansę przyjść w pierwszej połowie interwału między autobusami, co w drugiej. Raz będziemy czekać dwie minuty, innym razem 8... Średnio wyjdzie połowa czasu między autobusami.

Rozumowanie to jest fałszywe, gdyż przybywając losowo na przystanek, będziemy mieli większą szansę trafić w interwał dłuższy niż średnie 10 minut! Typowo poczekamy więc dłużej. Okazuje się, że nasz czas oczekiwania jest również równy 10 minut. Możemy potraktować siebie jako „dodatkowy autobus", nieskorelowany z innymi, ale należący do tego samego procesu (jedno nasze pojawienie się nie zmieni średniej odległości między zdarzeniami).

Wyprowadźmy wzór na prawdopodobieństwo czasu oczekiwania na następne zdarzenie. Wybierzmy mały odcinek czasowy Δt i zapytajmy, jaka jest szansa, że zdarzenie zajdzie właśnie podczas tego interwału. Odpowiedź jest: $\lambda\Delta t$, gdzie λ jest pewną stałą. Układ nie ma pamięci, jedynym parametrem jest gęstość statystyczna zdarzeń na jednostkę czasu, więc to wszystko. Szansa, że system „przeżyje" ten odcinek bez zdarzeń, wynosi więc $1-\lambda\Delta t$.

Szansa, że przeżyje dłuższy czas, np. T, gdzie $T = N\Delta t$, wyniesie $q = (1-\lambda\Delta t)^N$, gdyż prawdopodobieństwa zdarzeń niezależnych się mnożą, a prawdopodobieństwo przeżycia w każdym pod-interwale jest niezależne od pozostałych. Ponieważ $\Delta t = T/N$, dostaniemy $q = \left(1-\frac{\lambda \mathrm{T}}{\mathrm{N}}\right)^{\mathrm{N}}$. Granicą tego wyrażenia, gdy $N \to \infty$, jest $e^{-\lambda T}$.

Prawdopodobieństwo, że zdarzenie **zajdzie** po czasie T wynosi więc $1-e^{-\lambda T}$. To jest rozkład kumulatywny, w przedziale czasowym od zera do nieskończoności. Po długim czasie zdarzenie zajdzie na pewno (o ile w ogóle może zajść). Gęstość prawdopodobieństwa na jednostkę czasu wynosi $\frac{1}{\lambda}\, e^{-\lambda t}$ .Ten rozkład wykorzystamy w naszym generatorze procesu Poissona. Aby otrzymać jednorodny rozkład wydarzeń w czasie (nieograniczonym), użyjemy rozkładu wykładniczego, a zmienną będzie nie położenie wydarzenia na

osi czasowej, ale czas oczekiwania na następne. Wydarzenia będą generowane monotonicznie w czasie, co jest zgodne z intuicją.

Ponieważ dystrybuanta wynosi $d(t) = 1 - e^{-\lambda t}$, algorytm generacji jest prosty – generujemy $t = \frac{1}{\lambda} \ln(1 - r)$. Czytelnik śledzący uważnie tekst mógłby zaproponować uproszczenie: ponieważ r jest rozłożone równomiernie między 0 a 1, więc $1 - r$ także, i można sobie darować odejmowanie. Jednak typowe algorytmy generacji rozkładu równomiernego, oparte o liczby całkowite, zwykle dostarczają wyników *mniejszych* od 1. Zero może zostać wygenerowane, 1 – nigdy. Wyrażenie $\ln r$ może więc zakończyć się błędem i zatrzymaniem programu, $\ln(1 - r)$ jest bezpieczniejsze.

6.4. Specjalna metoda dla rozkładu Gaussa, oparta o Centralne Twierdzenie Graniczne

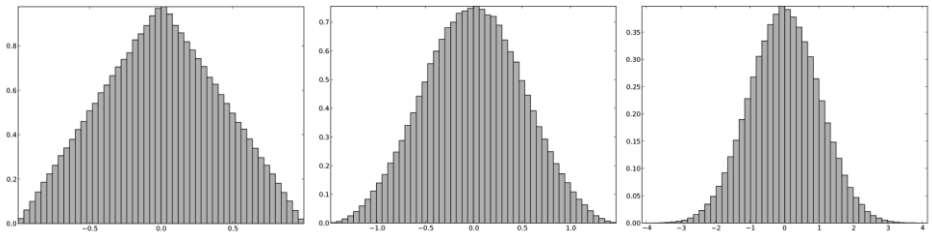
Dla rozkładu normalnego $p(x) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{x^2}{2})$ (ogólne rozkłady Gaussa wymagają tylko zmiany skali i przesunięcia średniej). Całka tej funkcji nie jest elementarna, jest to tzw. funkcja błędu erf. Znamy oczywiście jej przybliżenia numeryczne, a także aproksymacje jej funkcji odwrotnej, można więc algorytm odwracania zastosować do tego przypadku, ale będzie on powolny i niewygodny, mimo iż się go używa. Można jednak postąpić inaczej, a algorytm jest bardzo pouczający...

Skąd się bierze dzwonowy kształt tego rozkładu? Wiemy już, że ta funkcja opisuje typowe fluktuacje wielkości fizycznych, ich rozrzut wokół średniej. Zwykle jest to skutkiem wkładu licznych źródeł fluktuacji, częściowo się kompensujących. Np. proces technologiczny napełniania butelek obarczony jest błędami: zmienne ciśnienie płynu, bąbelki gazu zmieniające średnią lepkość przepływu, itp. Raz trochę więcej, raz trochę mniej, najprawdopodobniejsze będą małe fluktuacje, a bardzo duże pojawią się niezmiernie rzadko.

Teoretyczne ujęcie tego zjawiska wyraża się przez tzw. centralne twierdzenie graniczne, mówiące, że suma dużej liczby wielkości losowych o dowolnych rozkładach (ale ze skończoną wariancją), będzie miała rozkład gaussowski. Możemy to sprawdzić na prostym przykładzie rzutu kostkami.

Ponieważ prawdopodobieństwa zdarzeń niezależnych się mnożą, gdy rzucimy dwie kości, jedno elementarne zdarzenie (k_1, k_2) będzie miało prawdopodobieństwo 1/36. Jeżeli interesuje nas tylko suma oczek, prawdopodobieństwo wyniku 2 będzie 1/36 – prawdopodobieństwo zdarzenia (1,1). Ale szansa otrzymania 3 będzie już 2/36: (1,2) i (2,1). Szansa siódemki będzie 6 razy większa niż dwójki lub dwunastki. Rozkład prawdopodobieństwa dany jest krzywą trójkątną. Dla trzech kostek wzrost przy małej sumie nie jest liniowy, a kwadratowy, rozkład jest sklejeniem trzech odcinków parabol, i już wygląda jak „dzwon”. Dla większej liczby składników krzywa szybko zmierza do postaci

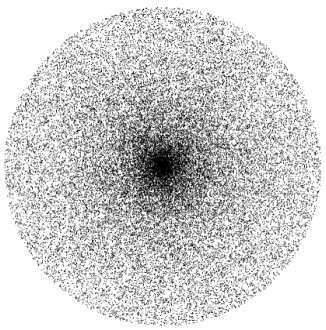
Gaussa. Jeśli zamiast kostek wziąć standardowy generator równomierny, otrzymamy wyniki jak na rys. 12.



Rys. 12. Rozkład sumy 2, 3 i 12 liczb losowych równomiernych

7. Rozkłady wielowymiarowe

Generowanie wektorów losowych bywa trudniejsze niż pojedynczych liczb losowych, gdyż składniki tych wektorów mogą być ze sobą związane skomplikowanymi zależnościami.



Rys. 13. Niejednorodny rozkład w kole.

Aby np. wygenerować losowe kierunki w dwóch wymiarach, albo – co na jedno wychodzi – punkty na okręgu, wystarczy wygenerować równomiernie kąt α między zerem a 2π, a następnie utworzyć $(x, y) = R \cdot (\cos\alpha\,, \sin\alpha)$. A co we wnętrzu koła? Można pomyśleć, że wystarczy wygenerować kąt, ale także promień i skorzystać ze wzorów powyżej. Jednak, jeśli promień jest losowany równomiernie, rozkład nie jest jednorodny, okolice środka koła są gęstsze. Powodem jest fakt, że tyle samo punktów będzie losowanych dla każdego promienia, ale w pobliżu środka te punkty rozłożą się na mniejszej powierzchni. Promień należy losować z rozkładem $p(r) \approx r$ (już wiemy jak). Formalnie można to uzasadnić patrząc na przekształcenie współrzędnych Kartezjańskich na biegunowe pod całką:

$$\int dx \cdot dy = \int r \cdot dr \cdot d\varphi\,.$$

Ale nie będziemy się nad tym zatrzymywać. Jak wygenerować losowe kierunki w przestrzeni 3D (punkty na powierzchni sfery)?

Teraz mamy dwa kąty (θ, φ): szerokość i długość geograficzna. Ponieważ

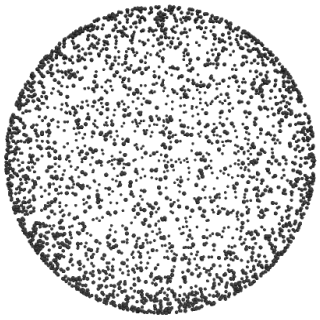
$$\int dx \cdot dy \cdot dz = \int r^2 dr \cdot \sin\theta\, d\theta \cdot d\varphi,$$

należy wygenerować φ równomiernie między 0 a 2π, oraz θ z rozkładem sinusowym; jeśli u jest liczbą losową standardową, $\theta = \arccos(2u - 1)$. (Jeżeli chcemy generować punkty we wnętrzu kuli, widzimy z jakim rozkładem losować r).

Inna metoda, mniej popularna, ale bardzo interesująca, polega na zauważeniu, że iloczyn trzech funkcji Gaussa się upraszcza:

$$e^{-x^2} \cdot e^{-y^2} \cdot e^{-z^2} = e^{-r^2}.$$

Wynik zależy tylko od promienia, a nie od kątów. Wystarczy więc wylosować trzy liczby (x, y, z) o *rozkładzie normalnym*, następnie znormalizować ten wektor, tj. podzielić składowe przez $\sqrt{x^2 + y^2 + z^2}$, i to nam dostarczy punktu na powierzchni sfery jednostkowej.



Rys. 14. Losowe punkty na powierzchni sfery

8. Zakończenie

Zagadnienie zostało omówione w sposób bardzo powierzchowny, zainteresowanych odsyłamy do literatury Internetowej, bardzo bogatej, z wieloma algorytmami gotowymi do użycia.

W drugiej części artykułu omówimy generowanie funkcji losowych (szumów), bardzo przydatnych np. w grafice, oraz przedyskutujemy kilka metod Monte-Carlo użytecznych w symulacji systemów fizycznych.

Literatura

[1] M. Kac, *Marginalia. What is random?*, American Scientist **71**(4) (1983), s. 405–406
[2] D. Knuth, *Sztuka programowania*, Tom 2: *Algorytmy seminumeryczne*, WNT, Warszawa (2002)
[3] M. Matsumoto, T. Nishimura, *Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator*, ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, (1998), s. 3–30