Generiranje naključnih števil

↓↓↓
Blaž Stojanovič

→ **Leta 1946 so v Los Alamosu fiziki, ki so v tem času izdelovali jedrsko bombo, prvič pognali simulacije z metodo Monte Carlo [1]. Dandanes algoritma ne uporabljajo le fiziki, temveč se ta na široko uporablja tudi v financah, biologiji in računalniški grafiki. Organizacija IEEE je Metropolisov algoritem, različico metode Monte Carlo, uvrstila med deset najpomembnejših algoritmov dvajsetega stoletja. Ključnega pomena tako za Monte Carlo algoritem kot še za mnoge druge pa je generiranje naključnih števil. Toda kljub veliki potrebi po dobrih generatorjih naključnih števil smo nanje čakali kar nekaj časa. Še leta 1988, 42 let po iznajdbi Monte Carlo metode, je bil objavljen članek z naslovom *Generatorji naključnih števil: dobre je težko najti* (*Random generators: good ones are hard to find* [2]).**

Pred iznajdbo računalnikov je bilo iskanje naključnih števil zelo mučno. Sir Francis Galton, znan angleški polihistor, je leta 1890 v slavno revijo *Nature* napisal, da ne obstaja boljše naprave za naključno izbiranje števil kot igralne kocke [3]. Da bi postopek vsaj malce pospešili, so na začetku dvajsetega stoletja, ko je potreba po takšnih številih narasla, nastale dolge tabele polne naključnih števil. Prvo je leta 1927 objavil Leonard Tippett, vsebovala pa je 41600 števk. Tippett je števila naključno izbral iz cenzusnih registrov.

Prvi računalnik, ki je lahko generiral prava naključna števila, je bil *Ferranti Mark 1*, ki je 20 bitna naključna števila ustvarjal s pomočjo električnega šuma. Tudi danes lahko prava naključna števila dobimo z meritvijo fizikalnih sistemov, za katere pričakujemo, da so naključni. Pridobivamo jih npr. z merjenjem atmosferskega in termičnega šuma ali pa z meritvijo kakšnih kvantnih pojavov. Za veliko uporab pravzaprav ni ključnega pomena, da so števila zares naključna, zadostuje da so *statistično* naključna, kar pomeni, da zaporedja takih števil ne vsebujejo nobenih vzorcev ali regularnosti in jih za praktične potrebe ne moremo ločiti od zares naključnih. Ta, t. i. *psevdonaključna* števila, lahko brez večjih težav generiramo z računalniki, kar je mnogo hitreje kot pridobivanje zares naključnih števil.

Princip delovanja generatorjev psevdonaključnih števil

Princip delovanja generatorja psevdonaključnih števil je preprost. Začnemo z začetnim številom s_0, ki ga imenujemo seme (ang. *seed*), iz tega s pomočjo prehodne funkcije f izračunamo novo število. Tako generiramo zaporedje števil z zaporednim apliciranjem funkcije f kot $s_1 = f(s_0)$, $s_2 = f(s_1)$ oz. v splošnem

- $$s_i = f(s_{i-1}). \qquad (1)$$

Najpogosteje je seme s_0 določeno kar z računalnikovo uro, da je tudi samo do neke mere naključno, lahko pa ga seveda nastavimo tudi sami. Zaporedje števil, ki ga tako pridobimo po enačbi (1), se po določenem številu korakov začne ponavljati. To število korakov imenujemo *perioda* generatorja in jo označimo s p. Perioda generatorja naključnih števil je pomembna lastnost in želimo si, da bi bila perioda našega generatorja čim večja. Kadarkoli uporabljamo generatorje psevdonaključnih števil pa se moramo zavedati, da števila niso zares naključna, saj novo število iz starega dobimo po nekem determinističnem, vnaprej predpisanem postopku. Znan je citat slavnega matematika in fizika Johna von Neumanna [4], ki je opozarjal na takšno »zlorabo« generatorjev naključnih števil:

Kdor se spogleduje z uporabo aritmetičnih postopkov za generiranje naključnih števk, je v grehu. Kajti, kot je bilo večkrat poudarjeno, naključna števila sama po sebi ne obstajajo – obstajajo le metode, ki ustvarjajo naključna števila, in dosleden aritmetični postopek gotovo ni ena izmed njih.

Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin. For, as has been pointed out several times, there is no such thing as a random number – there are only methods to produce random numbers, and a strict arithmetic procedure of course is not such a method.

Citat ne pomeni, da je bil von Neumman proti uporabi psevdonaključnih števil, temveč je želel le opozoriti na pravilno uporabo.

Generiranje z rezanjem robnih števk

Prav John von Neumann je leta 1949 predlagal *metodo srednjega kvadrata*, ki spada v širšo skupino generatorjev, ki generirajo števila s pomočjo rezanja. Algoritem je zelo preprost: začnemo z n mestnim semenom, ki ga kvadriramo, in dobimo neko kvečjemu $2n$ mestno število, ki mu spredaj napišemo dovolj ničel, da je dolgo točno $2n$. To število obrežemo z leve in z desne, tako da spet dobimo n mestno število. Kot zgled si poglejmo prvi korak metode kvadriranja, če je seme 13:

- $s_0 = 13$
  $s_0^2 = 0169$
  $s_1 = \emptyset 16 \emptyset = 16$

Največja težava metod, ki uporabljajo rezanje števk, je, da se hitro ujamejo v kratke cikle ali pa naletijo na ničlo, in tako vračajo le še nič. Poglejmo nadaljevanje zgornjega zaporedja:

- $16 \to 25 \to 62 \to 84 \to 5 \to 2 \to 0 \to 0 \ldots$

V 50-ih letih je Metropolis pokazal, da za 20-bitna števila metoda s kvadriranjem lahko zaide v trinajst različnih ciklov, najdaljši izmed njih pa je dolg 143 številk [5]. Malce boljša je metoda z množenjem, kjer se naključno število pomnoži z drugim naključnim številom, a ima tudi ta enake hibe kot metoda srednjega kvadrata.

V programskem jeziku Python lahko metodo srednjega kradrata implementiramo v eni sami vrstici. Naš generator bo vračal štirimestna števila, brez težav pa ga lahko bralec spremeni, da bo vračal števila s poljubno mesti.

```
def generate(s):
    return int(str(s*s).zfill(8)[2:6])
```

Število s, podano kot parameter, najprej kvadriramo, potem ga spremenimo v niz s funkcijo `str`. Če ima niz manj kot osem elementov, ga dopolnimo z ničlami s pomočjo metode `zfill` in potem vzamemo števke od vključno drugega do vključno petega mesta, pri čemer začnemo šteti mesta z 0.

Oglejmo si delovanje generatorja s sliko naključnih točk v ravnini. Če hočemo generirati dvodimenzionalne točke, potrebujemo dve semeni. Standardno je, da generator psevdonaključnih števil vrača vrednosti na intervalu od 0 do 1, zato vrednosti ki jih dobimo po kvadriranju, delimo z največjim številom, ki ga lahko generator vrne. Funkcija `zaporedje_stevil` vrne zaporedje N psevdonaključnih števil s semenom s. Pokličemo jo dvakrat in tako dobimo x in y koordinate točk.

```
m = 9999
def zaporedje_stevil(N, s):
    rand = []
    for i in range(N):
        s = generate(s)
        rand.append(s/m)
        return rand

x = zaporedje_stevil(1000, 5412)
y = zaporedje_stevil(1000, 1143)
```

Ustvarjene točke so narisane na sliki 1 levo. Jasno je vidna pomanjkljivost generatorja, prej ali slej se ujame v cikel in vrednosti se začnejo ponavljati. Za zgoraj izbrana semena se to zgodi veliko prej kot v tisoč iteracijah in posledično je različnih točk na sliki precej malo.

Linearni kongruentni generatorji

Prvo pravo izboljšavo je predlagal Lehmer, ko je predlagal *linearne kongruentne generatorje*. Generatorji take vrste so še danes zelo priljubljeni: v programskem jeziku C jo uporablja funkcija `rand`, še vedno se uporablja tudi v programskem jeziku Java.[1] Generator deluje tako, da poleg semena izberemo še tri števila:

- m, modulus; $m > 0$
  a, multiplikator; $m > a \geq 0$
  c, inkrement; c

Nova števila dobimo s prehodno funkcijo

- $f(s_i) = (a \cdot s_i + c) \bmod m$,

kjer mod$m$ označuje ostanek pri celoštevilskem deljenju z m. Pri izbiri naših čarobnih števil m, a in c moramo biti pazljivi, če želimo, da je naš generator čim boljši. Tipično sta c in m tuji, a pa je izbran tako, da za vsak $x \in \mathbb{N}$ velja, da $a \cdot x$ ni deljiv z m. Jasno je, da dolžina cikla nikoli ne bo presegla števila m, saj so ostanki pri deljenju z m med 0 in $m - 1$. Izkaže pa se, da jo lahko maksimiziramo, če so izpolnjeni naslednji pogoji:

- c in m sta tuji,
- $a - 1$ je večkratnik vseh praštevil, ki so delitelji m,
- $a - 1$ je večkratnik 4, kadar je m večkratnik 4.

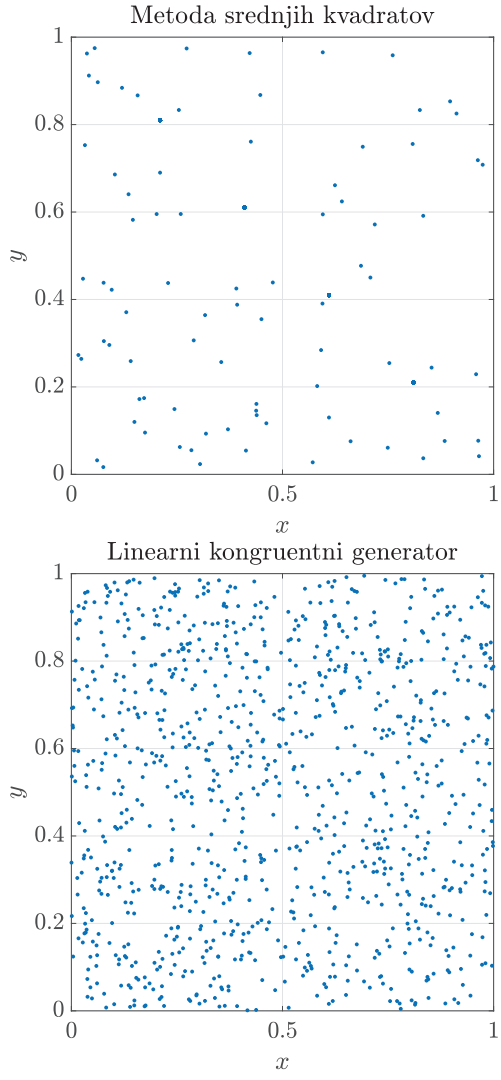
Bralci, ki jih zanima dokaz, ga lahko najdejo v Knuthovi slavni knjigi [5]. Implementacija je še bolj preprosta kot pri metodi srednjih kvadratov:

```
def generate(m, a, c, s):
    return (a*s + c) % m
```

Seveda je treba pametno izbrati čarobna števila. Modulus je ponavadi potenca 2, ker lahko računalnik ostanke pri deljenju s potencami 2 izračuna hitreje, zato izberemo $m = 2^{32}$. Ostali števili sta izbrani ustrezno po zgornjem predpisu, da je perioda generatorja $m - 1$, npr. $c = 1013904223$ in $a = 1664525$.

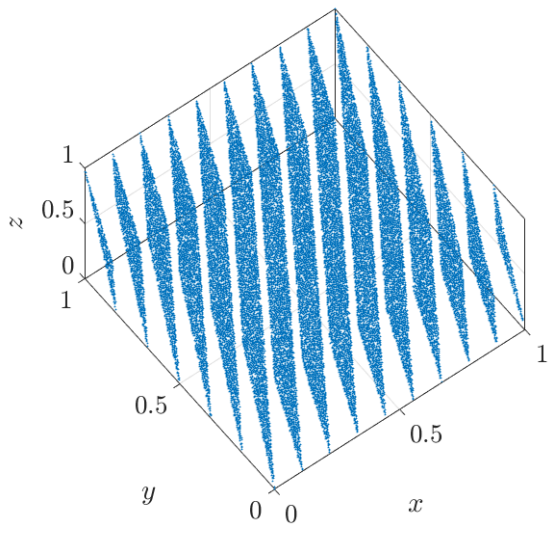
Na sliki 1 vidimo, da je opisana metoda veliko močnejša kot metoda srednjih kvadratov.

[1] verzija 11



SLIKA 1.
Tisoč točk ustvarjenih z metodo srednjih kvadratov in linearnim kongruentnim generatorjem

Kljub temu, da linearni kongruenti generatorji nimajo veliko očitnih težav, je ena izmed njih lepo vidna, ko generiramo točke v več dimenzijah. Generirane točke namreč ležijo v enakomerno razmaknjenih ravninah, le-teh pa je največ $(d! \cdot m)^{\frac{1}{d}}$, kjer je d število dimenzij. Če so m, a in c izbrani dovolj slabo, lahko ta pojav tudi vidimo. Tak primer je generator RANDU ($m = 2^{31}$, $a = 65539$ in $c = 0$), ki velja za enega izmed najslabših vseh časov.



SLIKA 2.
$2 \cdot 10^4$ točk v treh dimenzijah generiranih z RANDU, čarobna števila so zelo slabo izbrana.

Mersenne Twister

Leta 1997 sta Makoto Matsumoto in Takuji Nishimura iznašla generator psevdonakjučnih števil, ki je danes daleč najbolj razširjen in splošno uporabljan. Imenovala sta ga Mersenne Twister, kar v slovenščino približno lahko prevedemo kot Mersennov zvijalec. Uporablja Mersennova praštevila, imenovana po francoskem matematiku Marinu Mersennu. Poleg tega Mersenne Twister v svojem imenu tudi skriva začetnici črk avtorjev. Mersenne Twister je standarden psevdonaključni generator v programskih jezikih Python, R, Matlab, PHP, Lisp, na voljo pa je tudi v C++. Ustvarjen je bil z namenom, da odpravi slabosti psevdonaključnih generatorjev, ki so bili v rabi v tistem času. To metodi tudi uspe, saj ima različica, ki je najpogosteje uporabljena periodo $2^{19937}-1$ in opravi tudi najtežje statistične teste. Poleg tega metoda izkoristi tudi binarno strukturo računalnika, kar jo naredi zelo hitro.

Izračun π z Monte Carlo

Poglejmo si, kako bi s pomočjo naključnih števil dobili približek števila π. Kot smo omenili že prej, naključni generatorji števil vračajo števila v intervalu $[0,1)$, zato se je problema najlažje lotiti tako, da naključno izbiramo točke na enotskem kvadratu $[0,1) \times [0,1)$ in štejemo, koliko jih pade v notranjost kroga s središčem v izhodišču. Tu moramo biti malce pazljivi, saj se v enotskem kvadratu namreč nahaja le četrtina kroga, kar se lepo vidi na sliki 3. Razmerje med številom vseh generiranih točk in tistih, ki so se znašle v notranjosti kroga, nam nekaj pove o razmerju ploščin kvadrata in četrtine kroga. Vemo namreč, da za ploščino četrtine kroga $S_{krog/4}$ in ploščino kvadrata $S_{kvadrat}$ velja

- $$\frac{S_{krog/4}}{S_{kvadrat}} = \frac{\pi}{4},$$

če sta seveda stranica kvadrata in polmer kroga enaka. Verjetnost, da se bo naključno izbrana točka nahajala v krogu, je enaka ravno razmerju obeh ploščin. Od tod sledi, da je razmerje števila točk, ki se nahajajo v krogu N_{krog}, in števila vseh točk N približek za razmerje ploščin

- $$\frac{N_{krog/4}}{N} \approx \frac{S_{krog/4}}{S_{kvadrat}} = \frac{\pi}{4}.$$

Približek za π imamo tako na dlani:

- $$4\frac{N_{krog/4}}{N} \approx \pi.$$

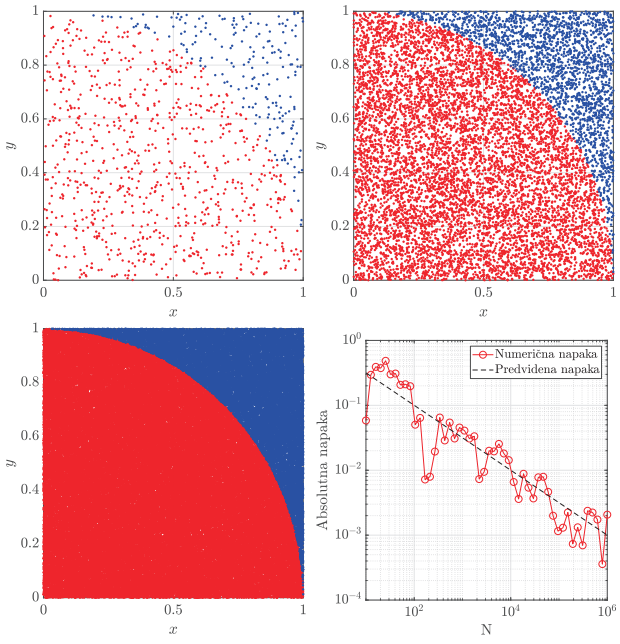
Zgornji postopek je primer metode Monte Carlo in pomembna lastnost postopka je, da napaka našega približka pada kot $\frac{1}{\sqrt{N}}$ z naraščajočim številom točk N.

Oglejmo si, kako lahko takšen približek napravimo v Pythonu. V knjižnici `numpy` imamo na voljo `random.rand`, ki uporablja Mersenne Twister. Približek za π dobimo že v nekaj vrsticah.

```
import numpy as np

N = 10**4
tocke = np.random.rand(2, N) # naredimo naključne
                               točke
r2 = np.sqrt(np.square(tocke[0]) +
   np.square(tocke[1])) # kvadrat razdalje
k = r2[np.where(r2 <= 1)] # izberemo točke v krogu

print(4*len(k)/N) # izračunamo približek
```



SLIKA 3.
(a) $N = 10^3$, $\pi \approx 3{,}18240$, (b) $N = 10^4$, $\pi \approx 3{,}14818$
(c) $N = 10^5$, $\pi \approx 3{,}14029$, (d) Odvisnost napake od števila točk.
Izračun števlia π z metodo Monte Carlo.

Na sliki 3 vidimo generirana števila pri različnih N, pa tudi napako našega približka v odvisnosti od N.

Vendar pa računanje približka π še zdaleč ni vse, kar bi lahko o generiranju nakjučnih števil povedali, kajti nobena izmed naštetih metod ni dobra za kriptografsko uporabo. Kriptografsko varni generatorji psevdonaključnih števil so zanimivi tako iz matematičnega kot praktičnega pogleda. Poleg tega smo izpustili tudi razpravo o tem, kako se generatorje psevdonaključnih števil zares vrednoti; statistični testi, ki se uporabljajo v te namene, so lahko iztočnica za nadaljnje branje.

Zgodba o naključnih številih ni zanimiva le za matematike, je tudi pomemben nauk za vse, ki računalnike uporabljajo in hočejo iz njih iztisniti kar je le mogoče. Opozarja na to, da je včasih vredno pogledati pod pokrov, in se vprašati, kako stvari delujejo, ali obstaja boljši način. Donald Knuth je v svoji knjigi Umetnost računalniškega programiranja (Art of Computer Programming [5]) zapisal:

> Danes uporabljamo veliko generatorjev naključnih števil, za katere pa žal ne moremo reči, da so dobri. Premalokrat ljudje namreč nismo pripravljeni na uporabo novih metod dela, posebej, če se nam zdi, da stare delujejo. Tako je tudi v tem primeru – stare, ne več zadosti dobre metode programerji prevzemajo drug od drugega, uporabniki pa ne vedo ničesar o tem, da so pravzaprav že pomanjkljive.

> Many random number generators in use today are not very good. There is a tendency for people to avoid learning anything about such subroutines; quite often we find that some old method that is comparatively unsatisfactory has blindly been passed down from one programmer to another, and today's users have no understanding of its limitations.

Generatorji naključnih števil in njihova uporaba so se od takrat močno izboljšali, vendar zgornja trditev morda danes velja za kakšno drugo metodo, ki jo uporabljamo vsak dan.

Literatura

[1] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller in E. Teller, *Equation of state calculations by fast computing machines*, The journal of chemical physics, **21** 1953, 1087–1092.

[2] S. K. Park in K. W. Miller, *Random number generators: good ones are hard to find*, Communications of the ACM, **31** 1988, 1192–1202.

[3] F. Galton, *Dice for statistical experiments*, 1890.

[4] J. von Neumann, *Various techniques used in connection with random digits*, John von Neumann, Collected Works, **5** 1963, 768–770.

[5] D. E. Knuth, *Art of computer programming, volume 2: Seminumerical algorithms*, Addison-Wesley Professional, 2014.

×××