Matematicko-fyzikální fakulta Univerzity Karlovy

NMAG166 Zkoušková práce

# Generátory pseudonáhodných čísel

Jiří Škrobánek

Praha
2019

# Obsah



# 1 Úvod

Tento článek popisuje způsoby generování pseudonáhodných čísel a některá využití náhodnosti a těchto generátorů.
Hned z kraje uveďme, co budeme rozumět generátorem náhodných čísel. Definice mohou být v závislosti na využití odlišné, vždy se však jedná o nějaký druh zobrazení.

**Definice 1** (Generátor náhodných čísel)**.** Nechť $f : \mathbb{N} \to [0, 1)$ je funkce a $\{f(n)\}$ posloupnost jejích funkčních hodnot. Řekneme, že $f$ je generátor náhodných čísel, pokud $\{f(n)\}$ je *náhodná posloupnost*.

Termín *náhodná posloupnost* je třeba zatím chápat intuitivně, bude rozveden později.
Je patrné, že takováto definice je pro praktické využití nevhodná. V souladu s [11] definujme vyčíslení.

**Definice 2** (Algoritmus vyčíslující číslo)**.** Algoritmus, jež, obdržeje kladné celé číslo $c$, vyprodukuje $c$ cifer čísla $r \in [0, 1)$ po řádové čárce v binárním zápisu a zastaví, vyčísluje číslo $r$.
Číslo $r \in [0, 1)$, které vyčísluje nějaký algoritmus, nazveme vyčíslitelné.

Algoritmů existuje pouze spočetně mnoho, rozumíme-li jimi všechny platné vstupy pro univerzální Turingův stroj. [1] Téměř jistě (s pravděpodobností 1) tedy nebudou funkční hodnoty generátoru náhodných čísel vyčíslitelné. Definujeme proto ještě diskrétní verzi generátoru.

**Definice 3** (Diskrétní generátor náhodných čísel)**.** Nechť $m \in \mathbb{N}$ a nechť $F : \mathbb{N} \to \{0, \ldots, m\}$ je funkce a $\{F(n)\}$ posloupnost jejích funkčních hodnot. Řekneme, že $F$ je diskrétní generátor náhodných čísel, pokud existuje generátor náhodných čísel $f$, takový že $\forall n \in \mathbb{N} : (m + 1) \cdot f(n) \in [F(n), F(n) + 1)$.

Čili diskrétní generátor v jistém smyslu pracuje se zaokrouhlenými hodnotami spojitého generátoru. Z důvodů, které budou patrné po upřesnění vlastností náhodné posloupnosti, je v praxi obtížné (nemožné) získat pravý generátor náhodných čísel, proto používáme generátor *pseudo*náhodných čísel, který svým chováním pro naše potřeby v dané situaci dostatečně aproximuje vlastní generátor náhodných čísel.
Potřebujeme-li pouze členy posloupnosti pro konkrétní indexy (je-li například nutno opětovně rekonstruovat stejnou situaci), hodí se ještě následující alternativní definice. Uveďme ji proto:

**Definice 4** (Alternativa náhodného generátoru pro abecedy)**.** Nechť $\Sigma$ a $\Psi$ jsou abecedy[2], $n \in \mathbb{Z}^+$ a $\varphi : \Sigma^* \to \Psi^n$. Řekneme, že $\varphi$ je náhodné, pokud existují diskrétní generátor náhodných čísel pro $m + 1$ hodnot $F$, bijekce $\alpha : \Sigma^* \to \mathbb{N}$ a $\beta : \Psi^n \to \{0, \ldots, m\}$ splňující:

$$\forall \gamma \in \Sigma^* : \beta(\varphi(\gamma)) = F(\alpha(\gamma))$$

[1] Podle Churchovy teze [2, kapitola 2] by nemělo záviset na zvoleném způsobu reprezentace čísel, ani na formální reprezentaci algoritmu.

[2] Abeceda je zde pojem z teorie formálních jazyků, jedná se de facto o neprázdnou konečnou množinu. $\Sigma^*$ značí množinu všech konečných posloupností nad abecedou $\Sigma$.

Všimněme si tedy, že náhodný generátor pro abecedy můžeme dle naší definice získat z jednoduše diskrétního generátoru tak, že konečné řetězce nad abecedou $\Sigma$ očíslujeme přirozenými čísly a za $m$ zvolíme $|\Psi|^n$.

## 2 Motivace

Potřeba náhodných hodnot se objevuje v mnoho oblastech průmyslové, vědecké i jiné činnosti. V některých případech je možné extrahovat náhodné hodnoty z prostředí, nelze to však zdaleka vždy, proto se využívají různé algoritmy. Oblasti opírající se o náhodné vstupy mimo jiné jsou:

- **Simulace** Podstatné jsou zejména modely v jaderné fyzice. Kvantové jevy se jeví jako náhodné, proto jsou k jejích simulaci potřebné náhodné hodnoty. Ale důležité jsou rovněž pro makroskopické simulace, kde se například náhodou může řídit rozhodování agentů.
- **Randomizované algoritmy** V určitých případech se jako efektivní ukazují nedeterministické algoritmy, které pracují asymptoticky rychle pouze v průměrném případě. Pro jejich efektivitu je klíčové mít k dispozici dostatečně náhodné hodnoty. Příklady mohou být Quicksort pro třídění a Kargerův algoritmus pro hledání minimálního řezu.
- **Probabilistické algoritmy** Rozdíl mezi těmito algoritmy a randomizovanými je ten, že u těchto algoritmů není zaručeno, zda bude výsledek správný. Výsledek je správný pouze s určitou (vysokou) přesností, nebo je výsledek algoritmu pouze v určitém okolí skutečného výsledku. Typicky se vyzkouší pouze nějaké malé množství možností z celkového množství (numerická integrace metodou Monte Carlo). Náhodná čísla jsou důležitá pro volbu těchto možností. Mnoho prvočíselných testů je pouze probabilistických.
- **Testování softwaru** Ověřit, že se program chová správně na všech svých vstupech není reálně možné, proto je velmi důležité dokázat vybrat malý reprezentativní vzorek testovacích dat, u nichž by se s velkou pravděpodobností chyby projevily.
- **Kryptografie** Volba klíčů pro šifrovací protokoly je prováděna pomocí pseudonáhodných generátorů. Některé protokoly navíc obsahují ve zprávě místo vyčleněné pro náhodné bity.
- **Rekreace** Digitalizované varianty klasických karetních a deskových her spoléhají na pseudonáhodné generátory. Čistě digitální hry pak užívají prvku náhody ve zhruba stejném měřítku.
- **Právo a volební systémy** Ačkoliv vzácně, státní systémy vyžadují rozhodnutí náhodným losem v určitých situacích (shoda v počtu hlasů). Ve starověkých Athénách se ale volby rozhodovaly čistě náhodným losem.

## 3 Náhodnost formálně

Nyní se pokusíme zpřesnit již zmíněný termín náhodná posloupnost (čísel z intervalu $[0, 1)$). Možná nepřekvapivě, se ukazuje, že intuitivní představa náhodné posloupnosti (Nejčastěji chceme, aby nebylo možné určit následující člen z předešlých.) se velmi špatně formalizuje. Většina jednoduchých definic vede k tomu, že buďto žádná náhodná posloupnost neexistuje, nebo že existuje mnoho posloupností splňujících takovou definici, které se ale z jiných důvodů nedají považovat za náhodné.
V tomto textu se budeme držet formalizace podle [1, svazek druhý, strany 149-179]. Dále uvažujme už jen posloupnosti na intervalu $[0, 1)$. Rozumný základní požadavek na náhodnou posloupnost je bezesporu následující.

**Definice 5** (Rovnoměrně (uniformně) rozdělená posloupnost)**.** Posloupnost $\{f(n)\}$ nazveme rovnoměrně rozdělenou, pokud splňuje:

$$\forall a, b \in [0, 1], a < b : \mu([a, b)) = |a - b| = \lim_{n\to\infty} \Pr\left[f(i) \in [a, b) | i \in \{0, \ldots, n\}\right]$$

Jinak řečeno podíl hodnot z každého intervalu odpovídá velikosti tohoto intervalu. Tento požadavek je ovšem nedostačující. Na lokální úrovni může být posloupnost značně předvídatelná. Je-li $\{g(n)\}$ náhodná posloupnost, pak je náhodná i $\{h(n)\}$, kde $h(n) = g(\lfloor n/2 \rfloor)$. Takže v $\{h(n)\}$ jsou po sobě jdoucí dvojice stejných čísel. Toto ovšem s intuitivní definicí náhodnosti není v souladu. Je třeba požadavky zpřísnit.

**Definice 6** ($k$-rozdělená posloupnost)**.** Pro kladné celé číslo $k$, nazveme posloupnost $\{f(n)\}$ $k$-rozdělenou, pokud splňuje:

$$\forall a_1, \ldots, a_k, b_1, \ldots, b_k \in [0,1], a_i < b_i : |a_1 - b_1| \cdots |a_k - b_k| =$$
$$= \lim_{n\to\infty} \Pr\left[f(i) \in [a_1, b_1) \& \ldots \& f(i+k-1) \in [a_k, b_k) | i \in \{0, \ldots, n\}\right]$$

Jako pozorování uveďme, že rovnoměrně rozdělená posloupnost je 1-rozdělená.

**Tvrzení 1.** Je-li $\{f(n)\}$ $(k+1)$-rozdělená posloupnost pro celé $k > 1$, pak je $k$-rozdělená.

*Důkaz:* Nechť $\{f(n)\}$ je $(k+1)$-rozdělená posloupnost pro celé $k > 1$. Z definice

$$\forall a_1, \ldots, a_k, a_{k+1}, b_1, \ldots, b_k, b_{k+1} \in [0,1], a_i < b_i : |a_1 - b_1| \cdots |a_k - b_k| \cdot |a_{k+1} - b_{k+1}| =$$
$$= \lim_{n\to\infty} \Pr\left[f(i) \in [a_1, b_1) \& \ldots \& f(i+k-1) \in [a_k, b_k) \& f(i+k) \in [a_{k+1}, b_{k+1}) | i \in \{0, \ldots, n\}\right]$$

Volbou $a_{k+1} = 0, b_{k+1} = 1$ dostaneme

$$\forall a_1, \ldots, a_k, b_1, \ldots, b_k \in [0,1], a_i < b_i : |a_1 - b_1| \cdots |a_k - b_k| \cdot |1 - 0| =$$
$$= \lim_{n\to\infty} \Pr\left[f(i) \in [a_1, b_1) \& \ldots \& f(i+k-1) \in [a_k, b_k) \& f(i+k) \in [0, 1) | i \in \{0, \ldots, n\}\right] =$$
$$= \forall a_1, \ldots, a_k, b_1, \ldots, b_k \in [0,1], a_i < b_i : |a_1 - b_1| =$$
$$= \lim_{n\to\infty} \Pr\left[f(i) \in [a_1, b_1) \& \ldots \& f(i+k-1) \in [a_k, b_k) | i \in \{0, \ldots, n\}\right]$$

Což je přesně definice $k$-rozdělené posloupnosti, tedy posloupnost je $k$-rozdělená **Q.E.D.**

**Definice 7** ($\infty$-rozdělená posloupnost)**.** Posloupnost $\{f(n)\}$ nazveme $\infty$-rozdělenou, pokud je $k$-rozdělená pro každé kladné celé $k$.

Vhodné zpřísnění požadavků na náhodnou posloupnost je, aby byla $\infty$-rozdělená. Takto předejdeme označení $\{h(n)\}$ za náhodnou. Zároveň každá $\infty$-rozdělená posloupnost už splňuje některé běžné testy, které se užívají ke kontrole toho, zda se posloupnost chová náhodně.
Stále však nemůžeme být plně spokojeni, protože $\infty$-rozdělená posloupnost stále může vykazovat vlastnosti, které nepovažujeme za náhodné.

**Tvrzení 2.** Nechť $\{v(n)\}$ je $\infty$-rozdělená posloupnost. A $\{u(n)\}$ je posloupnost definovaná vztahem

$$u(n) = \begin{cases} v(n) & n \text{ není prvočíslo,} \\ 0 & n \text{ je prvočíslo} \end{cases}.$$

Pak $\{u(n)\}$ je $\infty$-rozdělená posloupnost.

*Důkaz:* Prvočísla mají z prvočíselné věty asymptotickou hustotu 0, to jest
$\lim_{n\to\infty} \Pr\left[x \text{ je prvočíslo} | x \in \{0, 1, ..., n\}\right] = 0$. Pro libovolné celé kladné $k$ proto platí

$$\lim_{n\to\infty} \Pr\left[\text{množina } \{x, x+1, \ldots, x+k-1\} \text{ obsahuje prvočíslo} | x \in \{0, 1, \ldots, n\}\right] = 0$$

Ať je tedy vnitřní predikát pro pravděpodobnost po dosazení prvočísla za některý z indexů splněn, či nikoliv, neovlivní to asymptotickou pravděpodobnost. Platí

$$\lim_{n\to\infty} \Pr\left[u(i) \in [a_1, b_1) \& \ldots \& u(i+k-1) \in [a_k, b_k) | i \in \{0, \ldots, n\}\right] =$$
$$= \lim_{n\to\infty} \Pr\left[v(i) \in [a_1, b_1) \& \ldots \& v(i+k-1) \in [a_k, b_k) | i \in \{0, \ldots, n\}\right]$$

Takže i $\{u(n)\}$ je $(k)$-rozdělená posloupnost pro každé celé kladné $k$, proto je $\infty$-rozdělená. **Q.E.D.**
Přímočarý způsob jak toto napravit by byl požadovat, aby každá podposloupnost náhodné posloupnosti byla $\infty$-rozdělená. Žádná posloupnost s touto vlastností ovšem neexistuje.

**Tvrzení 3.** Nechť $\{v(n)\}$ je rovnoměrně rozdělená posloupnost. Existuje rostoucí posloupnost nezáporných celých čísel $\{i(n)\}$ taková, že $\{v(i(n))\}$ je rostoucí posloupnost.

*Důkaz:* Ukážeme, že pro každé nezáporné celé $t$ existuje celé $s > t$, že $v(t) < v(s)$. Toto ukážeme sporem. Je-li $\forall i \in \{t+1, t+2, \ldots\} : v(i) \leq v(t)$, pak $1 - v(t) = \mu([v(t), 1)) > 0 =$

$$= \lim_{n\to\infty} \Pr\left[v(i) \in [v(t), 1) | i \in \{t, \ldots, n\}\right] = \lim_{n\to\infty} \Pr\left[v(i) \in [v(t), 1) | i \in \{1, \ldots, n\}\right]$$

Toto je ve sporu s definicí rovnoměrně rozdělené posloupnosti.
Začneme tedy podposloupnost od libovolného prvku v $\{v(n)\}$ a přitom víme, že následníka můžeme vždy zvolit, protože vhodný prvek v posloupnosti existuje **Q.E.D.**

**Tvrzení 4.** Žádná rovnoměrně rozdělená posloupnost, jejíž každá podposloupnost by byla rovnoměrně rozdělená, neexistuje.

*Důkaz:* Podle předchozího tvrzení každá rovnoměrně rozdělená posloupnost obsahuje rostoucí podposloupnost. Žádná rostoucí posloupnost $\{r(n)\}$ není rovnoměrně rozdělená, protože

$$\lim_{n\to\infty} \Pr\left[r(i) \in [r(0), r(1)) | i \in \{0, \ldots, n\}\right] = 0 < |r(1) - r(0)|$$

.
Našli jsme tedy podposloupnost, která není rovnoměrně rozdělená. Žádná kýžená posloupnost tedy neexistuje **Q.E.D.**
Ve chvíli, kdy známe hodnoty v posloupnosti, dokážeme vždy najít velmi nenáhodnou podposloupnost. Proto formulujeme následující definici.

**Definice 8** (Náhodná posloupnost)**.** Posloupnost $\{f(n)\}$ nazveme náhodnou, pokud pro každou totální rostoucí parciálně rekurzivní funkci jednoho argumentu $r$ je posloupnost $\{f(r(n))\}$ $\infty$-rozdělená.

Definice třídy parciálně rekurzivních funkcí [12] není přímočará, zde ji nebudeme uvádět. Vzhledem k Curchově tezi můžeme ale místo parciálně rekurzivní funkce v definici použít výstupy nějakého Turingova stroje, který se zastaví pro libovolný vstup, pro dané přirozené číslo na vstupu.
Dá se ukázat, že některé posloupnosti jsou náhodné podle předchozí definice. Ani tato definice náhodné posloupnosti ve skutečnosti ale není zcela vhodná, bylo by vhodné ji ještě zesílit, nicméně budeme ji užívat. Blíže opět v [1].

# 4 Kvalita generátoru

K určení toho, jak moc aproximuje posloupnost náhodného generátoru skutečně náhodnou posloupnost, byly vyvinuty testy, například Kolmogorovův-Smirnovův test [9], $\chi^2$-test (chí-kvadrát test) [10], spektrální testy [1, svazek druhý, strana 93], sledování délek monotónních úseků a další.
Pro zhodnocení, jak moc se daná posloupnost chová náhodně, je vhodné používat více různých druhů testů. Dobrý výsledek v jednom testu zdaleka není postačující.
Zejména v kryptografii je pro bezpečnost generátoru navíc zcela zásadní, aby nebylo možné určit pokračování posloupnosti (parametry generátoru) z několika jejích známých členů. Odhalování chyb generátorů je přitom značně netriviální proces.

# 5 Typy generátorů

Uvažme diskrétní generátor náhodných čísel $F$. Existuje-li nějaká parciálně rekurzivní funkce, jejíž funkční hodnoty pro všechny argumenty odpovídají hodnotám $F$, pak můžeme sestrojit jinou rostoucí parciálně rekurzivní funkci, která bude určovat konstantní podposloupnost hodnot generátoru $F$. Existuje-li spojitý generátor $f$ z definice diskrétního generátoru, jeho posloupnost nezaokrouhlených hodnot není náhodná, neboť umíme najít podposloupnost, jejíž všechny členy leží v nějakém vlastním podintervalu $[0, 1)$. To je spor s $\infty$-rozděleností této podposloupnosti, proto původní parciální rekurzivní funkce nemůže existovat.

Z definice náhodné posloupnosti výše tedy vyplývá, že každá posloupnost nad množinou $\{0, \ldots, m\}$, kde $n$-tý člen je dodán algoritmem, nemůže být náhodná. Proto tedy pseudonáhodné generátory.
Většinou se v tomto kontextu používají binární slova, v tomto textu ale budeme ve formulacích používat vektory. Uveďme nyní některé typy pseudonáhodných generátorů.

## Lineární kongruenční generátor (LCG)

Mějme čtyři přirozená čísla:

$$\begin{aligned} m,&\quad \text{modulo,} && 0 \neq m,\\ a,&\quad \text{násobitel,} && a < m,\\ c,&\quad \text{inkrement,} && c < m,\\ C(0),&\quad \text{počáteční hodnota,} && C(0) < m. \end{aligned}$$

Pak diskrétní $m$-ární generátor $C$ [4] definujeme předpisem

$$C(n+1) = (a \cdot C(n) + c) \mod m.$$

Každý generátor $C$ je nutně periodický, ale při vhodně zvolených hodnotách se dá dosáhnout i toho, že $m$ po sobě jdoucích hodnot je různých. Pokud se tohoto podaří dosáhnout, není už počáteční hodnota příliš významná, určuje pouze posunutí posloupnosti, a může být volena libovolným vnějším způsobem.
Bývá výhodné, a proto se často volí $m$ jako mocnina 2, zejména $2^{32}$, nebo $2^{64}$, protože se jedná o velikost použitého datového typu a modulení je prováděno automaticky s každým výsledkem.
Je důležité, aby čísla $m$, $a$, $c$, byla nesoudělná. Pro $c$ je například vhodná hodnota 1. Je-li $a$ zvoleno příliš blízko 0 ($< m/100$) nebo příliš blízko $m$ ($> 99m/100$), vyskytují se v posloupnosti úseky, které nejsou příliš náhodné. Jsou-li tyto podmínky dodrženy, posloupnost vykazuje dobré pseudonáhodné rysy.
Velká výhoda kongruenčního generátoru je jeho jednoduchost a tudíž rychlost. LCG se zároveň stal základem pro další obměněné generátory.

### Lehmerův generátor

Lehmerův generátor [5] $L$ je speciální případ kongruenčního generátoru (s $c = 0$) pracující nad multiplikativní grupou mod $p$, kde $p$ je prvočíslo nebo jeho mocnina. Nevyužívá tedy obvykle efektivního hardwarového modulení $2^k$.
Lehmerův generátor je definovám předpisem

$$L(n+1) = a \otimes L(n),$$

kde $L(0)$ je zvoleno libovolně nesoudělně s $p$, $a$ je zvoleno jako prvek vysokého řádu. [3]

## Permutovaný kongruenční generátor (PCG)

Permutovaný kongruenční generátor [3] je relativně nový přístup vycházející z LCG. $2^t$-ární PC generátor $P$ sestrojíme následujícím způsobem. Vyjít můžeme z LCG $C$ s parametry $m = 2^{2^k}, a, c, C(0)$, musí platit $2^k \geq t$, typicky $2^k = 64$ nebo 128.

**Definice 9** (Posun)**.** Posun o $p$ míst je operace pracující s vektorem $u = (u_1, u_2, \ldots, u_d)^T$, $d \geq p$. Výsledkem je vektor $u \twoheadrightarrow p = (0, 0, \ldots, 0, u_1, u_2, \ldots, u_{d-p-1}, u_{d-p})^T$, tedy $p$ nul je přidáno na začátek vektoru a posledních $p$ složek je odstraněno.

**Definice 10** (Rotace)**.** Rotace o $q$ míst je operace pracující s vektorem $u = (u_1, u_2, \ldots, u_d)^T$, $d \geq q$. Výsledkem je vektor $u \circlearrowleft q = (u_{1+q \mod d}, u_{2+q \mod d}, \ldots, u_q)^T$.

Hodnotu $C(n)$ budeme interpretovat jako vektor prostoru $\mathbb{F}_2^{2^k}$, $u = (u_1, u_2, \ldots, u_k)^T$. PCG poté s hodnotou provádí nějaké vektorové operace. Některé z vhodných operací jsou následujících:

[3] $\otimes$ značí násobení v grupě.

- **RR** *Náhodná rotace* Je-li na vstupu vektor $u$, s $2^k$ složkami, vezmeme jeho prvních $k-1$ složek. Tento vektor dimenze $k-1$ interpretujeme jako číslo v binární soustavě $b$, dalších $2^{k-1}$ složek $u$ bude tvořit výstupní vektor s $2^{k-1}$ složkami, nejdříve se s nimi ale provede rotace (podle definice 10) o $b$ míst. Zbylých $2^{k-1}-k+1$ složek zůstane nevyužito.

- **RS** *Náhodný posun* Je-li na vstupu vektor $u$, s $2^k$ složkami, vezmeme jeho prvních $k-3$ složek. Tyto interpretujeme jako $k-3$ cifer čísla v binární soustavě $b$. Složky s indexy $b+k-3$ až $2^{k-1}+b+k-4$ budou tvořit výstupní vektor dimenze $2^{k-1}$.

  Ilustrujme toto na příkladovém vektoru ($k=5,\ b=10_2$):

$$(\mathbf{1,0},1,1,1,0,0,0,1,1,0,1,1,0,1,0,1,0,0,0,1,1,0,1,1,0,1,0)^T \rightarrow$$

$$\rightarrow (1,0,0,0,1,1,0,1,1,0,1,0,1,0,0,0)^T$$

  Tučná část určuje posun. Černé pozice jsou výstupní. Tato operace je z hlediska pseudonáhodnosti slabší než RR, ale zato je jednodušší.

- **XSH** *XOR-posun* Je-li $u$ vstupní vektor, výstupní vektor je $u + (u \twoheadrightarrow c)$. Kde $c$ je nějaká konstanta zvolená s ohledem na ostatní operace. (Používáme značení podle definice 9.)

- **XSL** *Zjednodušený XOR-posun* Je-li $u \in \mathbb{F}_2^{2^k}$ vstupní vektor, výstupní vektor je $u + (u \twoheadrightarrow 2^{k-1})$.

- **RXS** *Náhodný XOR-posun* Je-li $u$ vstupní vektor, výstupní vektor je $u + (u \twoheadrightarrow f(u))$. Kde $f$ je nějaká jednoduchá funkce. Posun tedy není o konstantní počet pozic.

- **M** *Násobení* V kontextu vektorů se jedná o násobení čtvercovou maticí.

Možné funkční kombinace jsou například XSH-RR (na $C(n)$ se nejprve provede XSH, na výsledek se pak použije RR), XSH-RS (na $C(n)$ se nejprve provede XSH, na výsledek se pak použije RS).
Vychází výhodné mít 64-bitový vnitřní LCG a 32-bitový výstup nebo 128-bitový vnitřní LCG a 64-bitový výstup. PCG jsou dokonalejší než LCG, ale jejich implementace funguje o něco pomaleji.

### Linear-feedback shift register – LFSR *(Posuvný registr s lineární zpětnou vazbou)*

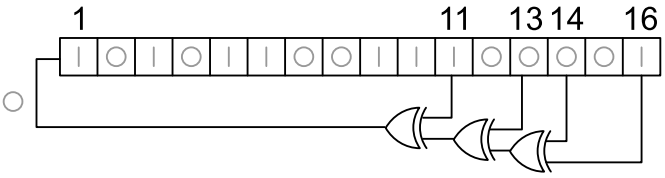
LFSR generátor pracuje nad tělesem charakteristiky 2. Významným případem je $\mathbb{F}_2$, čemuž odpovídá binární posloupnost, následující prvek je určen z několika předcházejících prvků (stavu generátoru) [6].
V praxi můžeme tyto generátory rozdělit do 2 kategorií: Fibonacciho a Galoisovy, rozdíl je však pouze technický. Posloupnosti generované Galoisovy LFSR jsou právě ty generované Fibonacciho LFSR [7, kapitola 2]. Intuitivnější je Fibonacciho generátor, kterému se věnuje další odstavec. Kvůli použití bitových operací je ale implementačně výhodnější používat Galoisův LFSR.

#### Fibonacciho LFSR

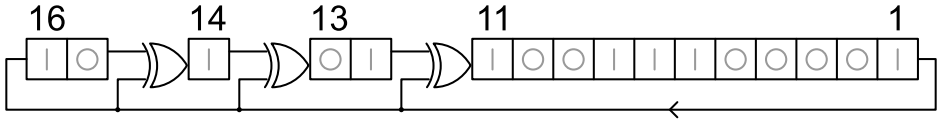
Fibonacciho generátor $F$ řádu $k$ je zadán rekurentním vztahem:

$$F(n) = a_1 \cdot F(n-1) + a_2 \cdot F(n-2) + \ldots + a_k \cdot F(n-k),$$

kde $a_i \in \mathbb{F}_2$ pro každé $i$. V takovém případě řekneme, že $F$ je zadán zpětným polynomem $a_k x^k + a_{k-1}x^{k-1} + a_1 x + 1$.
Počáteční stav (Počátečních $k$ prvků) by neměl být výhradně nulový, jinak se zřejmě $F(i) = 0$, pro všechna $i$. Maximální možná délka periody posloupnosti potom tedy je $2^k - 1$, protože existuje tolik nenulových stavů generátoru.
Dá sa ukázat, že posloupnost má maximální periodu, právě když je její charakteristický zpětný polynom (*feedback polynomial*) primitivní. Na obrázku 1 je Fibonacciho LFSR generátor řádu 16 maximální periody daný polynomem $x^{16} + x^{14} + x^{13} + x^{11} + 1$.




Obrázek 1: Fibonacciho 16-bitový LFSR generátor s maximální periodou




Obrázek 2: Galoisův 16-bitový LFSR generátor s maximální periodou, produkuje stejnou posloupnost jako generátor z obrázku 1.

### Maticová reprezentace LFSR

Následující stav LFSR se dá vyjádřit přehledně maticově. Označíme-li si stav Fibonacciho LSFR $F$ po vygenerování $n$ hodnot $\phi(n) = (F(n-k), \ldots, F(n-1), F(n))^T$. Stav po vygenerování dalšího prvku v posloupnosti $F$ v dalším kroku získáme pro vhodnou matici $A$ a $\phi(n)$ vektor předchozích $k$ hodnot (stav) výpočtem $\phi(n+1) = A \cdot \phi(n)$.
Zpětnému polynomu $a_k x^k + a_{k-1} x^{k-1} + a_1 x + 1$ by odpovídala matice $A$.

$$A = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_1 & a_2 & a_3 & \cdots & a_k \end{bmatrix}$$

Někdy je potřeba přeskočit velké množství hodnot v posloupnosti, proto se hodí, že získáváme vyjádření $n$-tého členu totiž

$$\phi(n) = A^n \cdot \phi(0)$$

a rychlejší způsob, jak $n$-tý člen počítat – pomocí rychlého umocňování matic. Zde $\phi(0)$ reprezentuje původní stav.

### Bezpečnost LFSR

Jak vidíme z následujícího tvrzení, využití generátoru (zde LFSR) v konkrétní situaci je třeba vždy zvážit. Z $2n$ po sobě jdoucích ukořistěných hodnot posloupnosti dokážeme získat celý cyklus.

**Tvrzení 5.** LFSR generátor $Q$ řádu $n$ s periodou $2^n - 1$ je jednoznačně určen hodnotami $Q(i), Q(i+1), \ldots, Q(i+2n-1)$ pro libovolné celé nezáporné $i$.

Z následující soustavy rovnic dokážeme určit vektor koeficientů $(a_1, a_2, \ldots, a_n)^T$:

$$\left[\begin{array}{cccc|c} Q(i+n-1) & Q(i+n-2) & \cdots & Q(i) & Q(i+n) \\ Q(i+n) & Q(i+n-1) & \cdots & Q(i+1) & Q(i+n+1) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ Q(i+2n-2) & Q(i+2n-3) & \cdots & Q(i+n-1) & Q(i+2n-1) \end{array}\right]$$

Protože pouze jeden polynom může generovat maximální posloupnost, získali jsme vektor koeficientů.

## Příklad nerekurentního algoritmu

Velice často není praktické mít přístup k předchozím členů posloupnosti náhodného generátoru, čili $n$-tý člen posloupnosti musí být snadno zjistitelný pouze z $n$. Budeme totiž chtít členy posloupnosti

pro vybrané indexy (řetězce nad abecedou). Zde už se dostáváme k blízce souvisejícímu konceptu hašovacích funkcí.
Opusťme vektorový pohled, konstruujeme generátor $G$ pro abecedy. Pro zjednodušení uvážíme abecedu $\Omega$, jejíž znaky budou odpovídat jednotlivým řetězcům původní výstupní abecedy délky $n$. Budeme mít funkci dvou argumentů $w$, prvním argumentem bude prvek $\Omega$, druhým bude prvek vstupní abecedy $\Sigma$. Hodnotu, kterou přiřadíme řetězci $\sigma \in \Sigma^*$ bude odpovídat $w(\ldots w(w(\omega, \sigma_1), \sigma_2) \ldots, \sigma_{|\sigma|})$.
Možná lépe toto zapíšeme jako $G(\sigma) = x(\sigma, |\sigma|)$, přitom

$$x(\sigma, i) = \begin{cases} w(x(\sigma, i-1), \sigma_i) & i > 0 \\ \omega & i = 0 \end{cases},$$

$\omega \in \Omega$. Prázdnému řetězci proto zřejmě náleží $\omega$. [4] Pěknější pohled na generování bychom dostali s použitím konečných automatů.
Funkci $w$ musíme však volit obezřetně, abychom získali dostatečně kvalitní a bezpečný generátor.

# 6 Motivační příklad

Uvažme nějakou složitou funkci, pro kterou neumíme analyticky nalézt určitý integrál (ale tento integrál existuje). Jeden možný numerický přístup, by byl počítat integrál podle definice Riemannova integrálu, ale nedojít až k limitě. Tedy rozdělit interval na podintervaly a na každém z nich vypočítat hodnotu v prostředním bodě a předpokládat, že funkce je už na těchto podintervalech konstantní.
Alternativně můžeme v jedné dimenzi spočítat funkční hodnotu pro velký počet náhodných bodů a výsledek získat jako průměr. Podle zákona velkých čísel se budeme blížit k cíli. Tento přístup dává smysl zejména pro vícerozměrné integrály. Metoda se nazývá Monte Carlo [8].
Velice často se technika ukazuje na výpočtu konstanty $\pi$. V takovém případě se náhodně generují body z intervalu $[0, 1] \times [0, 1]$ a počítá se kolik z nich je ve vzdálenosti nejvýše 1 od bodu $(0, 0)$, tedy funkce má hodnotu 1 v této oblasti, jinde je nulová.
Spočtěme tedy nějaký určitý integrál funkce dvou reálných proměnných

$$f(x, y) = xe^y + \sin(x)\cos(y)$$

na intervalu $[0, 1] \times [0, 1]$. Výpočet provedeme v jazyku Python, stačit k tomu bude následující kód.

```
import numpy
import math
def f(x, y):
    return x * math.exp(y) + math.sin(x) * math.cos(y)
samples = 1000000
total = 0.0
for (x,y) in numpy.random.rand(samples, 2):
    total += f(x, y)
print(total / samples)
```

Dostaneme výsledek 1.24602..., což se od skutečné hodnoty

$$\int_0^1 \int_0^1 xe^y + \sin(x)\cos(y)\ dx\ dy = (e - 1 + \sin(2))/2 + \sin(1) \approx 1.245963...$$

liší velmi málo. (Využili jsme toho, že u této funkce analyticky hodnotu integrálu získáme bez problémů).
Knihovna `numpy` vnitřně používá ke generování *Mersenne twister*, což je zdokonalený generalisovaný registr se zpětnou vazbou. Algoritmus je to složitější, proto zde nebude rozebrán.
Poznamenejme ještě, že v praxi se používají mnohá vylepšení metody Monte Carlo.

[4] $|\cdot|$ pro řetězec značí jeho délku.

# 7 Závěr

Představili jsme některé základní techniky pro generování pseudonáhodných čísel. Ukázali jsme možnou formalizaci definice náhodné posloupnosti. Přehledově jsme zmínili, kde se náhodná hodnoty využívají. Jako základní aplikaci pseudonáhodných čísel jsme uvedli numerickou integraci pomocí metody Monte Carlo v bližším detailu.

# Reference


[1] KNUTH, Donald E. *The Art of Computer Programming.* Upper Saddle River, NJ: Addison-Wesley, 2011. ISBN 978-0321751041.

[2] ČERNÝ, Michal. *Výpočty.* Praha: Professional Publishing, 2011. ISBN 978-8074310492.

[3] O'NEILL, Melissa. *PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation* Harvey Mudd College, 2014. HMC-CS-2014-0905.

[4] PARK, S. K., MILLER K. W. *Random Number Generators: Good Ones Are Hard To Find. Communications of the ACM.* 31 (10): 1192–1201. 1988.

[5] PAYNE W.H.; RABUNG J.R.; BOGYO T.P. *Coding the Lehmer pseudo-random number generator. Communications of the ACM.* 12 (2): 85–86. 1969.

[6] PRESS, William; TEUKOLSKY, Saul; VETTERLING, William; FLANNERY, Brian. *Numerical Recipes: The Art of Scientific Computing, Third Edition.* Cambridge University Press. p. 386. 2007. ISBN 978-0-521-88407-5.

[7] KLEIN, Andreas. *Stream Ciphers* Springer. 2013. ISBM 978-1447150787.

[8] WEINZEIRL, Stefan. *Introduction to Monte Carlo methods* NIKHEF Theory Group. 2000. arXiv:hep-ph/0006269.

[9] SMIRNOV, N. *Table for Estimating the Goodness of Fit of Empirical Distributions. Ann. Math. Statist.* 19, no. 2, 279–281. 1948.

[10] PEARSON, Karl. *On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science.* 50:302, 157-175 1900.

[11] TURING, Alan M. *On Computable Numbers, with an Application to the Entscheidungsproblem. Proceedings of the London Mathematical Society.* 2, 42, 230-265 1937.

[12] ODIFREDDI, Piergiorgio; COOPER, S. Barry, *„Recursive Functions", The Stanford Encyclopedia of Philosophy* Winter 2016 Edition.