PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

BAKALÁŘSKÁ PRÁCE

Náhodná čísla

2013 Martin Polák

Anotace

Náhodná čísla jsou v dnešní době velmi důležitá. Uplatňují se často při počítačovém zpracování problémů z různých oblastí lidské činnosti. V práci jsou představeny některé z generátorů pseudonáhodných čísel, jsou popsány jejich výhody a nevýhody. Dále jsou popsány metody získání náhodných čísel pozorováním fyzikálních jevů. V závěru jsou představeny některé metody testování náhodnosti číselných posloupností. V rámci práce byly také implementovány představené algoritmy v programovacím jazyku C a vznikl program pro testování náhodnosti posloupností.

Chtěl bych poděkovat vedoucímu práce RNDr. Miroslavu Kolaříkovi, PhD. za zajímavé téma a cenné připomínky, a také mé rodině za trpělivost.

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

Od nepaměti se lidé během svého života setkávali s jevy, jejichž výskyt byl, a nebo se alespoň zdál být náhodný. U některých, jako je třeba střídání úplňku a novu u Měsíce, ovšem záhy odhalili zákonitosti a cyklus střídání a ze zdánlivě náhodného jevu se stal jev předvídatelný. U jiných, například průlet komet kolem Země, trvalo staletí, než pokrok vědy a poznání umožnil popsat chování komet a předpovědět jejich návrat. Stále ovšem existuje velké množství jevů, jejichž výskyt je pro nás, ať už skutečně nebo zdánlivě, náhodný. Náhodná čísla se uplatňují například při počítačových simulacích fyzikálních jevů, v kryptografii nebo i k zábavě v rámci počítačových her. Pomocí náhodných čísel můžeme také vybrat náhodný vzorek z velké množiny dat k analýze. Je tedy třeba získat co nejkvalitnější zdroj náhodných dat, protože jinak by simulace fyzikálních jevů nebyla kvalitní, v případě kryptografie bychom i při použití vhodných algoritmů nedostali dobré výsledky.

Ve 20. letech 20. století se objevily tabulky s náhodnými čísly. V roce 1927 publikoval L.H.C. Tippett tabulky s více než 40 tisíci náhodnými čísly, která získal z dat ze sčítání lidu [1]. Poté se započalo s konstrukcí přístrojů, které by umožňovali generování náhodných čísel automatizovaně. Počítač Ferranti Mark I, poprvé uvedený do provozu v roce 1951, obsahoval instrukci nazvanou /W, jež vložila do 20 nejméně významných bitů akumulátoru náhodné hodnoty [3]. Používal se k tomu hardwarový generátor šumu, který byl součástí počítače. Další možností jsou speciální zařízení určená přímo ke generování náhodných čísel. Jedním z takových je zařízení ERNIE (Electronic Random Number Indicator Equipment), které je používané od roku 1957 k losování výher v loterii britských dluhopisů. V současnosti se používá už 4. generace.

Při postupném rozšiřování počítačů se začalo řešit získávání náhodných čísel. Tabulky náhodných čísel nebyly v tehdejší době řešením, neboť počítače měly velice omezené paměťové možnosti. Hardwarové generátory, jako například ten v počítači Ferranti Mark I, způsobovaly problém s laděním programu, neboť při každém průchodu programem jednotka dodávala jiné hodnoty a navíc při poruše tohoto generátoru je obtížné poruchu detekovat. Proto se začalo s hledáním metod, jak generovat náhodná čísla čistě matematickými metodami, které pro stejné vstupy generují stejné výsledky, chovají se tedy deterministicky.

V práci se nejprve zabývám čistě matematickými metodami 2. pro generování posloupnosti náhodných čísel. Popisuji metodu middle square 2.2., včetně jejích nevýhod. Navrhl jsem vylepšení této metody, které odstraňuje některé nevýhody a umožní i snadnější použití v praxi. Dále popisuji generátory založené na lineární kongruentní metodě 2.2., kde jsou rozebrány volby parametrů a vlastnosti vygenerovaných posloupností. Nakonec je v matematických metodách popsán generátor Blum Blum Shub 2.3.2., který při dodržení určitých podmínek, může sloužit jako zdroj kvalitních náhodných čísel pro použití v kryptografii.

Další část práce se zabývá dvěma možnostmi, jak generovat posloupnosti náhodných čísel pomocí pozorování fyzikálních jevů. Nejprve se věnuji popisu generátoru Lavarand 3.1., který vznikl v 90. letech 20. století ve společnosti Silicon Graphics. Potom v kapitole 3.2. navrhuji metodu získávání posloupnosti náhodných čísel pomocí sledování difuze v kapalinách.

Následuje část, kde se seznámíme s možnostmi, jak poznat, zda je určitá posloupnost čísel náhodná. Představíme si především dvě metody - χ^2 test 4.2., což je základní statistický test a jako druhý je π test 4.3., který jsem navrhl a určuje náhodnost posloupnosti pomocí aproximace čísla π.

V poslední části práce, v kapitole 5., pak popisuji program VENGEANCE, který jsem v rámci této práce vyvinul, a který implementuje testy náhodnosti posloupnosti čísel s uniformním rozložením mezi 0 a 1. Program lze použít buď jako GUI aplikaci na počítačích firmy Apple s operačním systémem OS X nebo jako verzi pro příkazovou řádku, kde je program multiplatformní a může fungovat na všech platformách, kde je k dispozici kompilátor jazyka C. V rámci práce vznikla také multiplatformní knihovna, kde jsou implementovány metody, popsané v této práci.

2. Matematické metody

V této části se budeme zabývat matematickými metodami, které umožňují generování pseudonáhodných čísel. U náhodných čísel musíme vždy mluvit v kontextu, neboť těžko můžeme o jednom číslu bez dalšího prohlásit, že je náhodné. Místo toho mluvíme o jisté posloupnosti čísel, kde jednotlivé členy posloupnosti mezi sebou nemají (ideálně) žádnou závislost. V případě generování náhodné posloupnosti pomocí deterministického algoritmu hovoříme o pseudonáhodných číslech, jelikož mezi členy posloupnosti závislost je (je dána použitým algoritmem), ale v případě kvalitního algoritmu je tato závislost netriviální a posloupnost pseudonáhodných čísel se blíží posloupnosti zcela náhodných čísel. Říkáme, že posloupnost vypadá jako náhodná. Pseudonáhodná posloupnost bývá často definována rekurentním předpisem obecně ve tvaru:

$$x_{n+1} = f(x_n). \tag{1}$$

Zde x_n značí předchozí člen posloupnosti (počáteční hodnota x_0 se označuje jako semínko – seed), x_{n+1} je nově získaný člen posloupnosti a f je funkce, generující posloupnost. Pokud budeme pracovat v oboru přirozených čísel s 0, značíme $\mathbb{N}_0$, mohli bychom generující funkci popsat jako následující zobrazení:

$$f : (\mathbb{N}_0)^n \to \mathbb{N}_0. \tag{2}$$

Jedním ze základních vstupních parametrů je tzv. semínko (seed), což je startovací hodnota, od které se potom dále, podle funkčního předpisu, odvíjí posloupnost pseudonáhodných čísel. Jako seed se volí nějaká nesnadno zjistitelná

hodnota – často se používá aktuální čas nebo, pokud jsou dostupné příslušné senzory, teplota procesoru, případně kombinace těchto veličin.

Dále nás u posloupnosti pseudonáhodných čísel zajímají určité podposloupnosti. Pokud se totiž tyto podposloupnosti ve výstupu objeví opakovaně, mluvíme pak o cyklu posloupnosti. Je-li tento cyklus krátký, je to nežádoucí.

Definice 1. *Máme-li posloupnost* $(x_i)_{i=1}^n$ *čísel a dále ryze monotónní rostoucí posloupnost přirozených čísel* $(a_k)_{k=1}^m$, $m \leq n$, *pak posloupnost* $(x_{a_k})_{k=1}^m$ *nazveme podposloupností posloupnosti* $(x_i)_{i=1}^n$.

Pokud existuje podposloupnost, která se v původní posloupnosti periodicky opakuje, nazveme toto opakování cyklem posloupnosti.

Definice 2. *Mějme posloupnost* $(x_i)_{i=1}^n$ *a její podposloupnost* $(x_{a_k})_{k=1}^m$. *Existuje-li* $c \in \mathbb{N}$ *takové, že* $(x_{a_k})_{k=1}^m = (x_{a_k+bc})_{k=1}^m$, $b \in \mathbb{N}$, *pak* $(x_{a_k})_{k=1}^m$ *je cyklem posloupnosti* $(x_i)_{i=1}^n$ *s délkou* m.

Příklad 1. *Je dána posloupnost* $(x_n) = 1, 4, 3, 5, 8, 5, 9, 7, 0, 1, 9, 7, 0, 1, \ldots, 9, 7, 0, 1$. *Cyklem je podposloupnost* $(y_m) = 9, 7, 0, 1$ *a délka cyklu je* 4.

Je třeba tedy věnovat pozornost tomu, aby délka případného cyklu byla v posloupnosti co největší a aby nedošlo v krátké době k degeneraci náhodné posloupnosti na určitou hodnotu, například na 0, která se poté periodicky opakuje. Vznikne tím cyklus s délkou 1. Délka cyklu není jedinou důležitou veličinou udávající kvalitu náhodné posloupnosti. Existují posloupnosti délky m a délkou cyklu také m, tedy žádné číslo se v nich neopakuje, ale přesto o nich nemůžeme prohlásit, že by byly náhodné. Například posloupnost $1, 2, 3, 4, \ldots, m-1, m$, tedy posloupnost přirozených čísel od 1 do m, kde $x_{n+1} = x_n + 1$. Jistě se zde žádná hodnota neopakuje, ale snadno vidíme jaká hodnota bude následovat a můžeme okamžitě říci, jaká hodnota je například na 1000. místě posloupnosti.

2.1. Uniformní rozdělení náhodných čísel

Pro praktické použití je vhodné mít pseudonáhodná čísla v nějakém standardním tvaru, abychom s nimi mohli snadněji pracovat. Obvykle se používá uniformní rozdělení čísel mezi 0 a 1 [1]. Požadované číslo u_n získáme pomocí vztahu:

$$u_n = \frac{x_n}{m}, \tag{3}$$

kde x_n je vygenerované číslo v posloupnosti a m je maximální číslo, které se může objevit v posloupnosti. U metod, používajících modulární aritmetiku je to hodnota modulu, u metody 2.2. je to číslo $b^d - 1$, kde b je základ číselné soustavy a d je počet číslic generovaných náhodných čísel.

2.2. Metoda middle square

Někdy kolem roku 1946 navrhl John von Neumann metodu, která umožňuje snadno algoritmicky generovat posloupnosti pseudonáhodných čísel. Metoda se nazývá middle square, což poměrně přesně vystihuje její podstatu. U této metody mohou nastat některé problémy, které zde popisuji a navrhuji jejich řešení.

Pokud chceme získat následující náhodné číslo v posloupnosti, vezmeme aktuální číslo, spočítáme jeho druhou mocninu a jako výsledek použijeme prostřední číslice.

Příklad 2. *Číslo $x_n = 4896$, druhá mocnina $x_n^2 = 23970816$ a tedy následující číslo v posloupnosti je $x_{n+1} = 9708$.*

Algoritmus 1 nám ukazuje možnou implementaci postupu neformálně popsaného výše.

(1) Middle square

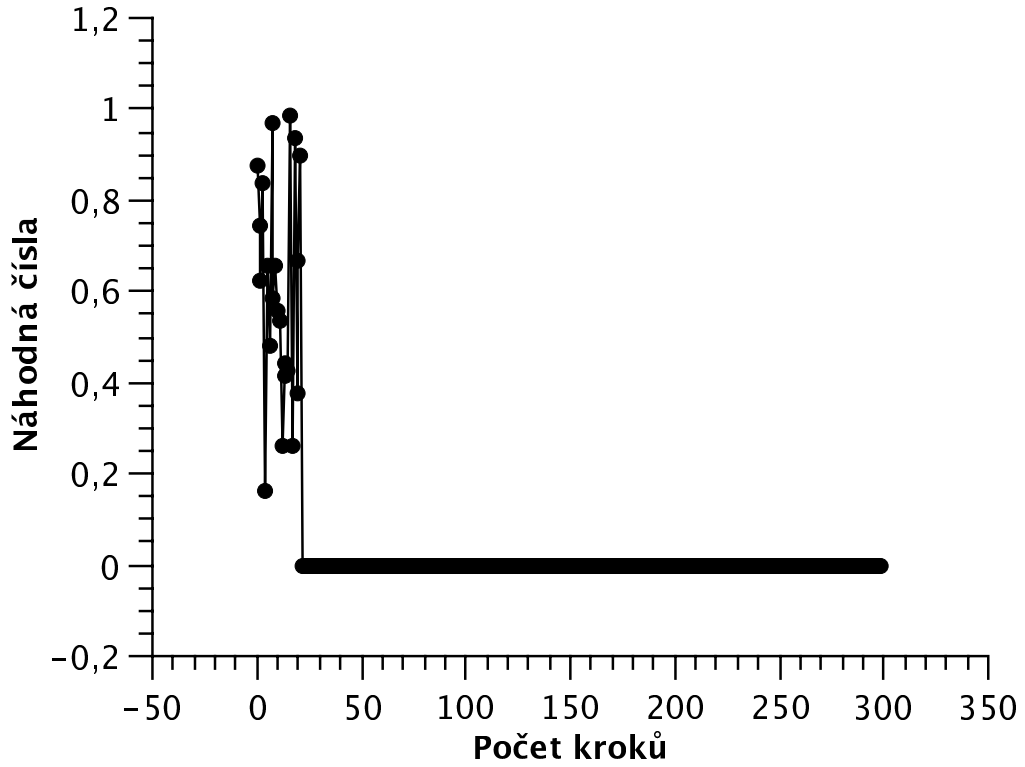
```
1: procedure MIDDLE-SQUARE(previous, digits)          ▷ Další náhodné číslo
2:     base ← 10                                      ▷ Pracujeme v dekadické soustavě
3:     start ← (3 * digits)/2
4:     end ← digits/2
5:     remainder ← previous^2 mod base^start
6:     result ← remainder/base^end
7:     return result
8: end procedure
```

Tato metoda není v praxi příliš využitelná. Problémem je především vytváření cyklů malé délky, což pseudonáhodnou posloupnost znehodnotí. Vzhledem k tomu, že základem generující funkce je druhá mocnina, vzniknou problémy u čísel, jejichž druhá mocnina má prostřední číslice stejné jako umocňované číslo (například 50, 60, 3792, 7600 nebo 2500) nebo čísla 0 a 1. Pokud se takové číslo dostane na vstup této metody, další členy posloupnosti již budou pouze toto číslo. Vznikne tedy cyklus s délkou 1. Další problém nám nastíní Věta 1. Označme počet číslic, jež má obsahovat výstupní číslo jako d.

Věta 1. *Pokud je $\frac{1}{2}d$ a více zleva nejvýznamnějších číslic vstupního čísla $x_n \in \mathbb{N}_0$ rovno 0, pak posloupnost v konečném počtu kroků zdegeneruje na hodnotu 0.*

Důkaz. Definujeme:

- b základ číselné soustavy,
- s pozice počáteční číslice středu čtverce x_n^2 zleva,
- e pozice koncové číslice středu čtverce x_n^2 zleva.



Obrázek 1.: Náhodná čísla generovaná metodou MIDDLE SQUARE s výchozí hodnotou 87564.

$$s = \frac{\log_b[(b^d)^2]}{2} + \frac{d}{2} = \frac{\log_b[(b^d)^2] + d}{2} = \frac{2d + d}{2} = 1\frac{1}{2}d \tag{4}$$

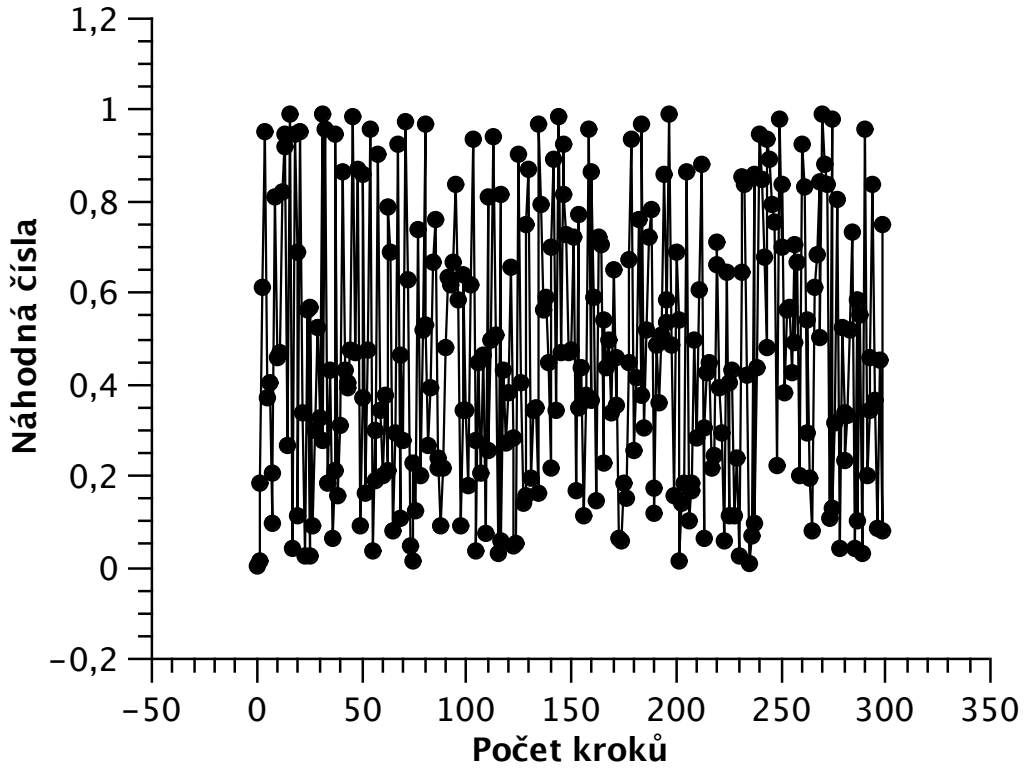
$$e = \frac{\log_b[(b^d)^2]}{2} - \frac{d}{2} = \frac{\log_b[(b^d)^2] - d}{2} = \frac{2d - d}{2} = \frac{1}{2}d \tag{5}$$

Dále je zřejmě $d = s - e$. Protože platí $\log_b(x_n^2) < \log_b(b^d)$, je také $x_n^2 < b^d$. Jelikož $s = 1\frac{1}{2}d$, tak $x_n^2 \bmod b^s = x_n^2$. Máme-li tedy $x_{n+1} = \frac{x_n^2 \bmod b^s}{b^e}$, plyne z toho, že $x_{n+1} = \frac{x_n^2}{b^e}$. Tedy $x_{n+1}^2 < x_n^2$ a tedy i $x_{n+1} < x_n$. Dostaneme-li tedy jako vstup číslo dle Věty 1, další členy posloupnosti budou vždy menší než předchozí a po konečném počtu kroků již budou výstupy z funkce vždy 0 (Obr. 1.). □

Ovšem problém nastane i v případě, že se nulové číslice objeví na konci vygenerovaného čísla.

Věta 2. *Je-li v d-místném čísle $x_n \in \mathbb{N}_0$ na $\frac{1}{2}d$ nejméně významných číslicích zprava číslice 0, je dalším členem v posloupnosti číslo, jež má na $\frac{1}{2}d$ nejméně významných číslicích zprava opět číslici 0.*

Důkaz. Mějme d-místné číslo x_n v soustavě o základu b, které má na $\frac{1}{2}d$



Obrázek 2.: Náhodná čísla generovaná metodou MIDDLE SQUARE IMPROVED s výchozí hodnotou 87564.

nejméně významných číslicích zprava 0. Existuje číslo y_n tak, že:

$$x_n = y_n b^s, \quad s = \frac{1}{2}d. \tag{6}$$

Číslo y_n se tedy od x_n liší o b^s násobek a zároveň má číslo y_n $\frac{1}{2}d$ číslic. Vzhledem k tomu, že číslo b^s je mocnina základu číselné soustavy, je pro všechna d-místná čísla x_n konstantou. Počet navzájem různých čísel x_n, která můžeme vytvořit je tedy shodný s počtem navzájem různých čísel y_n. Tím dochází v posloupnosti k tvorbě krátkých cyklů, protože se vytváří pouze náhodná čísla v rozsahu $b^{\frac{1}{2}d}$ a menším. □

Příklad 3. *Jako výchozí hodnotu použijme $x_n = 5600$ a pokračujeme v generování posloupnosti dále $x_{n+1} = 3600, x_{n+2} = 9600, x_{n+3} = 1600, \ldots$. Všimněme si, že ke každému x_n existuje i číslo y_n tak, že $x_n = y_n 10^2$ ($y_n = 56, y_{n+1} = 36, \ldots$).*

Podívejme se nyní, jak bychom mohli tyto problémy řešit. Po vygenerování dalšího členu posloupnosti budeme muset testovat, zda tento člen není problematický ve smyslu výše zmíněném. Musíme tedy vyzkoušet především, zda výstup není shodný se vstupem, zda nevyhovuje Větě 1, a nebo Větě 2. Pokud je jedna

z podmínek splněna, musíme použít pomocnou funkci, která výsledek upraví a zavoláme znovu generující funkci. Asi nejkvalitnější řešení by bylo oznámit žadateli o náhodná čísla, že je třeba dodat nové semínko, nicméně v tom případě bychom byli závislí na vnějším zásahu, což není vhodné. Navrhneme si tedy jednoduchou funkci, která toto bude řešit. Výhodou tohoto přístupu je, že pro stejné vstupy dává vždy stejné výstupy (je tedy deterministický), nevýhodou je, že náš problém nemusí řešit kvalitně.

Nejprve si tedy algoritmem 2 upravíme původní algoritmus 1. Přidáme podmínku, která testuje problematické hodnoty a pokud na nějakou narazí, zavolá rekurzivně algoritmus 2 s tím, že parametr *previous* je upraven algoritmem 3. Algoritmus 3 ke vstupní hodnotě přičte *digits*, což je počet číslic požadovaného výstupu z algoritmu 2. Tím se řeší problém popsaný Větou 2. Výsledek potom násobíme 2^{digits}, což řeší problém Věty 1.

(2) Middle square improved

```
1:  procedure MIDDLE-SQUARE(previous, digits)          ▷ Další náhodné číslo
2:      base ← 10                                      ▷ Pracujeme v dekadické soustavě
3:      start ← (3 * digits)/2
4:      end ← digits/2
5:      remainder ← previous^2 mod base^start
6:      result ← remainder/base^end
7:      if (result = previous) || (result^2 < base^(digits−1)) || (result mod base^end =
    0) then                                            ▷ || Disjunkce (OR)
8:          renewed ← RENEW(previous, digits)
9:          result ← MIDDLE-SQUARE(renewed, digits)
10:     end if
11:     return result
12: end procedure
```

(3) Renew for middle-square

```
1:  procedure RENEW(previous, digits)          ▷ Výstupem je upravené číslo
2:      sum ← previous + digits
3:      result ← sum <<^digits      ▷ << Bitový posun vlevo o určený počet bitů
    (digits)
4:      return result
5:  end procedure
```

Na Obr. 1. je vidět vznik cyklu s délkou 1. Všimněme si, že už asi po 20 krocích je výstupem hodnota 0, která se pak již bude opakovat pořád. Tento problém odstranila metoda middle square improved, jak vidíme na Obr. 2., kde při stejných vstupních parametrech tento cyklus nevzniknul.

2.3. Lineární kongruentní metoda

Metoda generování pseudonáhodných čísel poprvé představena D. H. Lehmerem v roce 1949. Můžeme ji popsat vztahem

$$x_{n+1} = (ax_n + c) \bmod m, \quad n \in \mathbb{N}_0, \tag{7}$$

kde

- x_{n+1} je další člen posloupnosti,
- x_n je aktuální člen posloupnosti; speciálně x_0 je počáteční člen posloupnosti – semínko,
- $a \in \mathbb{N}_0$ je multiplikativní konstanta,
- $c \in \mathbb{N}_0$ je aditivní konstanta,
- $m \in \mathbb{N}_0$ je modul.

Na volbě x_0, a, c, m závisí kvalita výsledné posloupnosti. Pokud si zvolíme například $a = 1$ a $c = 1$, tak výsledkem bude posloupnost $x_0, x_0{+}1, x_0{+}2, \ldots, x_0{+}m$, což sice je posloupnost s maximálním možným cyklem o délce m, ale náhodná jistě není.

Původní Lehmerova metoda nebere v úvahu aditivní konstantu, tzn. $c = 0$ (i když Lehmer zmiňoval i možnost $c > 0$), takže vztah je jednodušší:

$$x_{n+1} = ax_n \bmod m, \quad n \in \mathbb{N}_0. \tag{8}$$

Vztah (8) nám pěkně ukazuje, že pokud má být generovaná posloupnost náhodná, je třeba volit $a \geq 2$. Dále u (8) například nikdy nedosáhneme maximální délky cyklu m. Což je zřejmé, neboť $c = 0$, a proto by se nemělo vyskytnout $x_n = 0$.

2.3.1. Volba konstant

Konstanta m, by měla být dostatečně velká. Uvědomme si, že posloupnost nemůže mít cyklus delší než m, pokud tedy zvolíme příliš malé m, bude mít posloupnost krátký cyklus a nebude příliš náhodná. Přirozeně se nabízí možnost volby m, jako maximální velikosti datového typu. Pak můžeme operaci dělení modulo m počítat snadno pouze pomocí bitových operací. Problém je ale v tom, že číslice na pravé straně čísla jsou méně náhodné, než číslice nalevo. V počítači totiž pracujeme s čísly v binární soustavě a maximální velikost celočíselného datového typu je vždy ve tvaru 2^n. Pokud bude d dělitelem čísla m, a zároveň bude platit

$$y_n = x_n \bmod d, \tag{9}$$

potom

$$y_{n+1} = (ay_n + c) \bmod d. \tag{10}$$

Vytváří se lineární kongruentní posloupnost s délkou cyklu maximálně d. Budeme-li pracovat s čísly v soustavě o základu 2 a budeme mít $m = 2^{32}$ a $d = 2^3$, budou tři nejméně významné bity zleva tvořit posloupnost s cyklem nejvýše 8, protože $y_n = x_n \bmod 8$ a platí (10), viz Př. 4. Pokud bychom použili $m = 2^{16} - 1$, problém s méně náhodnými číslicemi by nastal pouze v případě, že bychom do (10) dosadili za d hodnoty $3, 5, 17, 257$, čili čísla, jež dělí modul. Pro praktické použití je tedy lépe použít hodnotu $m = 2^n \pm 1$, případně největší možné prvočíslo $p < 2^n$. Samozřejmě, že v některých případech nám nemusí menší náhodnost číslic v pravé části čísla vadit. Pak můžeme použít hodnotu $m = 2^n$ a výpočet bude snazší.

Příklad 4. *Zvolíme* $a = 3$, $c = 0$, $m = 2^5$, $d = 2^8$, $x_0 = 35$, *pak je*

$$\begin{aligned} x_{n+1} &= (3 \cdot 35) \bmod 32 = 9 \\ y_{n+1} &= (3 \cdot 35) \bmod 8 = 1 \\ y_{n+1} &= x_{n+1} \bmod 8 = 9 \bmod 8 = 1. \end{aligned} \tag{11}$$

Pro volbu dalších konstant je užitečné znát prvočíselný rozklad příslušných hodnot m (Tab. 1.). Hodnoty jsem získal z [1], kde je v tabulce na straně 14 kompletní přehled. Pro naše potřeby budou stačit uvedené hodnoty (Tab. 1.).

$2^n - 1$	n	$2^n + 1$
$3 \cdot 5 \cdot 17 \cdot 257$	16	65537
$3 \cdot 5 \cdot 17 \cdot 257 \cdot 65537$	32	$641 \cdot 6700417$
$3 \cdot 5 \cdot 17 \cdot 257 \cdot 641 \cdot 65537 \cdot 6700417$	64	$274177 \cdot 67280421310721$

Tabulka 1.: Prvočíselný rozklad nejpoužívanějších čísel mocniné řady 2.

Nyní potřebujeme zvolit další konstanty a, c, x_0 tak, aby cyklus v posloupnosti měl nejdelší možnou délku, tedy m. Taková posloupnost existuje vždy, jak ukazuje příklad volby $a = 1$ a $c = 1$, posloupnost je $x_0, x_0 + 1, x_0 + 2, \ldots, x_0 + m$. My však budeme potřebovat stanovit obecnější pravidla pro konstanty, abychom získali náhodnou posloupnost.

Věta 3. *[1] Lineární kongruentní posloupnost má cyklus délky m tehdy a pouze tehdy, když*

1. *c je nesoudělné s m;*
2. *$(a - 1)$ je násobek p pro každé prvočíslo p, které dělí m;*
3. *$(a - 1)$ je násobek 4, pokud m je násobek 4.*

Důkaz Věty 3 je uveden v [1]. Nyní ještě dva zajímavé postřehy.

Délky cyklu m můžeme dosáhnout tehdy a pouze tehdy, když se každé celé číslo x, $0 \leq x < m$ vyskytne v cyklu pouze jednou. Cyklus délky m v posloupnosti je tedy možný tehdy a pouze tehdy, pokud je cyklus délky m v posloupnosti, která má $x_0 = 0$. Máme-li potom obecný tvar pro k-tý člen lineární kongruentní posloupnosti [1]

$$x_{n+k} = (a^k x_n + \frac{(a^k - 1)c}{a - 1}) \bmod m, \quad k \geq 0, n \geq 0 \tag{12}$$

a můžeme předpokládat, že $x_0 = 0$, dostaneme [1]

$$x_n = (\frac{a^k - 1}{a - 1})c \bmod m. \tag{13}$$

Jestliže by tedy c bylo soudělné s m, nikdy by nemohlo nastat $x_n = 1$. Proto je ve Větě 3 první podmínka nezbytná[1].

Vrátíme-li se zpět k případu (8), tedy $c = 0$, neměla by se objevit hodnota $x_n = 0$, jak jsme si již ukázali. Dále pokud d bude dělitel m a objeví se x_n, které bude násobkem d, další členy posloupnosti $x_{n+1}, x_{n+2}, \ldots$ budou opět násobky d. Pro dostatečně náhodnou posloupnost tedy budeme chtít, aby všechna x_n byla nesoudělná s m. Tím je maximální délka cyklu omezena na $\varphi(m)$, kde $\varphi(m)$ je Eulerova funkce, která vyjadřuje počet čísel v intervalu $0 \ldots m$, jež jsou nesoudělná s m.

Definice 3. *Nechť a je nesoudělné s m, potom nejmenší přirozené číslo k nazveme řádem čísla a modulo m, pokud platí $a^k \equiv 1 \pmod{m}$.*

Definice 4. *Číslo a nazveme primitivním prvkem modulo m, pokud má řád modulo m roven $\varphi(m)$. Má tedy maximální možný řád modulo m.*

Následující věta nám potom řekne, jaké jsou podmínky dosažení maximální délky cyklu v případě lineární kongruentní posloupnosti, kde $c = 0$ (8).

Věta 4. *[1] Lineární kongruentní posloupnost, kde $c = 0$ (8), má maximální délku cyklu tehdy, když*

1. *x_0 je nesoudělné s m;*

2. *a je primitivním prvkem modulo m.*

V případě, že $m = 2^n$, je a primitivním prvkem modulo m tehdy, když $a \equiv 3 \pmod{m}$ nebo $a \equiv 5 \pmod{m}$. Potom je maximální délka cyklu $m/4$ [1].

[1]Pokud jsou c a m nesoudělná, nemůže naopak nastat $x_n = 0, \quad n > 0$, což nám ale nevadí, protože $x_0 = 0$.

2.3.2. RANDU

Metoda generování náhodných čísel založená na lineární kongruenci. Obecný tvar je stejný jako u Lehmerovy metody (8). Volba konstant je $a = 65539$ a $m = 2^{31}$, jako semínko se používá hodnota x_0, kde x_0 je liché číslo. Posloupnost je dána předpisem

$$x_{n+1} = 65539 \cdot x_n \bmod 2^{31}, \quad n \in \mathbb{N}_0. \tag{14}$$

Protože x_0 je liché číslo, $65539 \cdot x_0$ je také liché. Výsledkem je tedy vždy liché číslo. Z toho vyplývá, že maximální délka cyklu nemůže být větší než $2^{31}/2$.

Pokud se podíváme na volbu konstanty a a hodnotu x_0, vidíme, že $65539 \equiv 3 \pmod 8$ a tedy 65539 je primitivním elementem modulo 2^{31} (viz Věta 4) a cyklus v posloupnosti dosáhne maximální možnou délku $2^{31}/4$.

Tato metoda byla populární v 60. a na začátku 70. let 20. století. Popularita metody byla daná i tím, že na (binárním) počítači lze tuto metodu implementovat velice snadno i bez použití násobení nebo dělení. Stačí k tomu bitové operace a sčítání, jak vidíme v algoritmu 4.

(4) RANDU

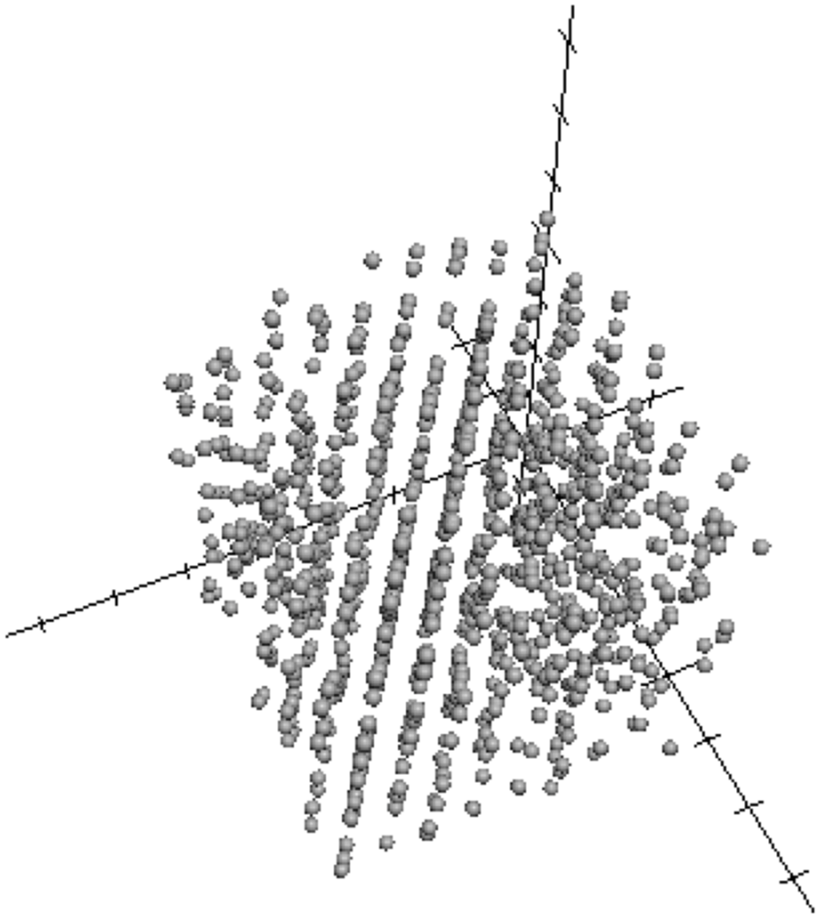
```
1: procedure RANDU(previous)          ▷ Výstupem je další číslo posloupnosti
2:     mask ← 2^31 − 1
3:     result ← (previous <<^16) + previous + (previous <<^1)   ▷ << Bitový
   posun vlevo o určený počet bitů
4:     result ← result & mask                          ▷ & Bitová operace AND
5:     return result
6: end procedure
```

Algoritmus je tedy velice rychlý a dá se implementovat i na počítačích s omezenými zdroji, ale v roce 1968 ukázal George Marsaglia, že pokud použijeme vygenerovaná náhodná čísla jako souřadnice bodů v jednotkové krychli s dimenzí ≥ 3, body nebudou v prostoru rozptýleny náhodně, ale seřadí se do několika rovin [4], viz Obr. 3. V (15) můžeme vidět, jak jsou na sobě závislé vždy 3 po sobě následující prvky posloupnosti generované algoritmem RANDU. Pokud pracujeme v aritmetice modulo 2^{31}, pak

$$\begin{aligned} x_{n+1} &= (2^{16}+3) \cdot x_n \\ x_{n+2} &= (2^{16}+3) \cdot x_{n+1} = (2^{16}+3)^2 \cdot x_n = \\ &= (2^{32} + 2^{16} \cdot 6 + 9)x_n = (6 \cdot 2^{16} + 9)x_n = \\ &= [6(2^{16}+3) - 9]x_n = 6(2^{16}+3)x_n - 9x_n = \\ &= 6x_{n+1} - 9x_n. \end{aligned} \tag{15}$$

Obecně můžeme x_{n+j} prvek posloupnosti RANDU popsat jako

$$x_{n+j} = a^j x_n \bmod 2^{31}, \tag{16}$$



Obrázek 3.: Uspořádání náhodných čísel generovaných algoritmem RANDU do paralelních rovin.

protože ale máme $a = 65539 = 2^{16} + 3$ a tedy

$$\begin{aligned} x_{n+j} &= x_n(2^{16}+3)^j \bmod 2^{31} = \\ &= x_n \cdot \sum_{k=0}^{j} \binom{j}{k} (2^{16})^{j-k} \cdot 3^k \bmod 2^{31}. \end{aligned} \tag{17}$$

Jelikož je $2^{31+n} \equiv 0 \pmod{2^{31}}, n \in \mathbb{N}_0$, všechny členy polynomu, kde je $j-k \geq 2$ budou rovny 0. Takový problém by nenastal, pokud by modul byl zvolen například jako největší možné prvočíslo menší než 2^{31}.

2.4. Metoda Blum Blum Shub

Tato metoda je vhodná jako zdroj kvalitních pseudonáhodných čísel pro kryptografické účely. Naopak se příliš nehodí jako obecná funkce pro generování pseu-

donáhodných čísel. Generující funkce je dána vztahem

$$x_{n+1} = x_n^2 \bmod m, \tag{18}$$

kde x_{n+1} je další člen posloupnosti, x_n je aktuální člen posloupnosti a m je modul. Výstupem jedné iterace je nejméně významný bit čísla x_{n+1}. Získaný bit se potom připojí zprava k výslednému číslu. Pokud tedy potřebujeme získat například 64 bitové pseudonáhodné číslo, musíme vygenerovat 64 členů posloupnosti dané předpisem (18). Tento generátor je vhodný především ke kryptografickým účelům, protože generování kvalitní posloupnosti je poměrně časově náročné. Dále je také problém, že délka cyklu v posloupnosti je kratší, než hodnota modulu m. Není tedy snadné převést členy posloupnosti na uniformní rozdělení.

(5) Blum Blum Shub

```
1: procedure BLUM-BLUM-SHUB(previous, m, bits)  ▷ Výstupem je další číslo posloupnosti
2:     result ← 0
3:     x_n ← previous
4:     for i ← 1, bits do
5:         x_n ← x_n^2 mod m
6:         result ← (result <<1)|(x_n&1)  ▷ & Bitová operace AND, << bitový posun o určený počet bitů, | Bitová operace OR
7:     end for
8:     return (result, x_n)
9: end procedure
```

Při volbě semínka x_0, bychom se měli vyhnout hodnotám 0 a 1. To je zřejmé neboť jejich druhá mocnina je opět stejné číslo. Dále by x_0 mělo být nesoudělné s m. Parametr m je součin dvou prvočísel p, q takových, že $p, q \equiv 3 \pmod 4$ a zároveň $p \neq q$.

Vysvětlit detailně problém, díky němuž je tento generátor považovaný za kryptograficky bezpečný, je nad rámec této práce. Zjednodušeně bychom mohli problém popsat: „Máme-li složené číslo n, určete odmocninu čísla x, kde x je kvadratické reziduum modulo n.“ Kvadratické reziduum modulo n je pak takové číslo $a \in \mathbb{Z}_n$[2], ke kterému existuje číslo $b \in \mathbb{Z}_n$ tak, že $b^2 \equiv a \pmod n$. K vyřešení tohoto problému je třeba znát prvočíselný rozklad čísla n, jehož zjištění pomocí faktorizace je pro velká čísla výpočetně složitá úloha. Tedy rozlišit posloupnost bitů, jež jsou generovány touto metodou (za předpokladu vhodně zvolených parametrů) od čistě náhodných bitů je minimálně stejně výpočetně náročné, jako faktorizace velkých čísel.

[2] $\mathbb{Z}_n$ značí množinu zbytkových tříd dělení modulo n.

3. Fyzikální jevy jako zdroj náhodných čísel

Jako zdroj náhodných čísel můžeme také využít některých fyzikálních jevů. Můžeme použít dva základní přístupy. Fyzikální jev může sloužit pro inicializaci generátoru náhodných čísel, jako například v 3.1. nebo můžeme použít přímo tento jev jako zdroj náhodných čísel 3.2.. Existují i další fyzikální jevy, které lze použít pro generování náhodných čísel. Například kvantové optické jevy, které jsou základem pro komerčně dostupný hardware pro generování náhodných čísel viz http://www.idquantique.com/random-number-generators/products.html. Také lze použít sledování rozpadu radiaktivních částic, což používá generátor HotBits viz http://www.fourmilab.ch/hotbits/. Z praktických důvodů (bohužel nemám k dispozici patřičné vybavení pro sledování rozpadu částic a jiných jevů) jsem navrhl generátor 3.2., který generuje náhodná čísla pomocí sledování difuze v kapalinách a který jsem mohl i implementovat.

3.1. Lavarand

Tato metoda využívá jako zdroj pro semínko generátoru náhodných čísel obrázek lávové lampy. Metoda vznikla v roce 1996 ve firmě Silicon Graphics, Inc. Lávová lampa je dekorační doplněk, který pomocí fyzikálního jevu, kdy hustější kapalina vytváří bubliny v méně husté kapalině, tvoří náhodné obrazce. Takto bylo u sebe umístěných 6 lávových lamp, které byly snímány kamerou. V pravidelných intervalech byly pořizovány obrázky, které slouží jako základ pro vygenerování semínka [5].

Každý snímek obsahoval více než 900 tisíc bytů a bylo třeba získat z něj semínko, které má 140 bytů. K tomu byla použita hašovací funkce SHA-1. Z jednoho obrázku se vygenerovalo 7 hašů délky 160 bitů (hašoval se vždy každý 7. byte) a výsledkem bylo 140 bytové semínko. Toto semínko se použilo jako vstup pro Blum Blum Shub generátor (2.3.2.), kde modul byl číslo:

```
53607511146220112360879790227353067135925376599474552853531481818526455500785383194936281698585494806464721601253386562653528822554332256253291788639699229111681587721849555921168837796146614508574462019262321980150281379240551466508660971979094060899184914133568666470293429076893274969016410915119621659901793350665953,
```

což je součin dvou prvočísel. Vzhledem k vlastnostem Blum Blum Shub generátoru a způsobu získání semínka, se jednalo jistě o kvalitní zdroj náhodných čísel pro kryptografické účely. Bohužel kvůli problémům a restrukturalizaci společnosti Silicon Graphics, Inc. již stránky věnované tomuto generátoru neexistují. Rekonstrukce tohoto generátoru je nad rámec této práce.

3.2. Difuze v kapalinách

Difuze je fyzikální jev, kdy se částice jedné látky rozptylují v látce jiné. Pohyb každé částice je náhodný, ovšem postupně dochází k přesunu částic z míst z vyšší koncentrací do míst s nižší koncentrací. Je sice nad rámec této práce popsat přesně tento jev, popíšeme si ale způsob, jakým lze tohoto jevu využít ke generování posloupnosti náhodných čísel.

Vybereme si difuzi v kapalině, kvůli rychlosti a relativně snadnému zachycení tohoto jevu. Mějme tedy průhlednou nádobu s vodou a jinou kapalinu, barevně výrazně odlišnou. Například inkoust. Ve chvíli, kdy začne probíhat difuze, začneme digitálním fotoaparátem pořizovat snímky. Tyto snímky potom převedeme do černobílé palety (Obr. 4.). Z takto upraveného snímku je ještě vhodné oříznout řádky, na kterých je pouze jedna barva, tzn. řádky které jsou buď celé bílé nebo celé černé.

Obrázek 4.: Snímek difuze v kapalině, převedený do černobílé palety.

Získané snímky potom použijeme k získání náhodných čísel. Pro praktickou implementaci převedeme snímek do formátu XPM (X PixMap), což je grafický formát, který se používá především v X Window systému. Data obrázku jsou zde uložena v zdrojovém kódu jazyka C, je tedy možné tento soubor vložit přímo do programu v jazyku C. Na začátku jsou uloženy rozměry, počet barev a definice barvové palety. Pak již následují samotná obrazová data. U černobílého snímku jsou reprezentována dvěma znaky – „ “ (mezera) jako černý bod a „.“ jako bílý bod. Náhodná čísla budeme generovat pouze z obrazových dat. Z každého řádku získáme jedno číslo náhodné posloupnosti, takže délka posloupnosti získané z jed-

noho snímku je shodná s vertikálním rozlišením snímku. Pokud je třeba delší posloupnost, musíme použít více snímků.

Náhodné číslo z vybraného řádku získáme pomocí hašovací funkce (19).

$$g(x) = h(x) \bmod 2^{32}, \tag{19}$$

kde x je číslo řádku a $h(x)$ definujeme jako

$$h(x) = \sum_{k=1}^{r} p^k \cdot asc(k-1), \tag{20}$$

kde r je horizontální rozlišení snímku, $p = 37$ je prvočíslo a $asc(k)$ je funkce, která pro znak v řetězci na pozici k, vrátí jeho hodnotu v tabulce ASCII.

(6) Diffusion

```
1: procedure DIFFUSION(row[])          ▷ row[] je pole s obrazovými daty řádku
2:     result ← 0
3:     p ← 37
4:     p_k ← 1
5:     i ← 0
6:     ch ← 1
7:     while (ch ← 1) ≠ 0 do
8:         p_k ← (p_k · p) mod 2^32
9:         result ← (result + ch · p_k) mod 2^32
10:        i ← i + 1
11:    end while
12:    return result mod 2^32
13: end procedure
```

V případě Obr. 4. vznikne posloupnost 236 náhodných čísel. Po převedení na uniformní rozdělení pomocí 2.1., vidíme část posloupnosti v Tab. 2.

Ještě bych se zastavil u hašovací funkce (19). Používám zde operace v modulu 2^{32}. Narozdíl například od lineární kongruentní metody 2.2., kde maximální délka cyklu v posloupnosti je dána velikostí modulu, zde tomu tak není. U deterministické metody, která generuje další číslo v posloupnosti na základě předchozího prvku, stačí, aby se některý prvek objevil v posloupnosti podruhé a další prvky se již budou také opakovat. U této metody toto neplatí.

4. Testování kvality náhodných posloupností

Pokud máme k dispozici nějakou posloupnost čísel, je dobré mít k dispozici také aparát, který nám umožní posoudit, jak moc je tato posloupnost náhodná.

1.	$0,7039153126$
2.	$0,7489043886$
3.	$0,4302932111$
4.	$0,7928773211$
5.	$0,9961468480$
⋮	⋮
236.	$0,3798935058$

Tabulka 2.: Posloupnost náhodných čísel získaných z difuze.

Člověk totiž není schopen objektivně zhodnotit, zda je určitá posloupnost čísel náhodná. Pokud někomu například dáme za úkol napsat náhodně 100 čísel, můžeme počítat s tím, že příliš náhodná nebudou. Naopak pokud bude mít člověk k dispozici posloupnost 100 náhodných čísel, je pravděpodobné, že v nich uvidí určité vzory [1]. Dokonce dle [8] dva studenti prováděli výzkum, kde ukázali, že lze poměrně přesně určit lidi na základě posloupnosti náhodných čísel, která napsali.

Testování posloupností náhodných čísel může být důležité v různých oblastech lidského konání. V případě voleb lze třeba určit, zda nedošlo k manipulaci s výsledky. V íránských prezidentských volbách v roce 2009, lze v počtech hlasů narazit na nesrovnalosti. Pokud vezmeme přesné počty odevzdaných hlasů v jednotlivých volebních obvodech, měly by být výsledkem náhodná čísla. Tedy na místě poslední číslice by se mělo vyskytovat všech možných 10 číslic přibližně stejně často. Ve skutečnosti se ale například číslice 7 vyskytovala v 17 % případů, číslice 5 zase pouze ve 4 % případů. To působí podezřele a je pravděpodobné, že výsledky někdo upravoval. U výsledků voleb v jiných státech se takové zvláštnosti neobjevují [8].

4.1. Benfordův zákon

Další zajímavý fenomén vzniká, pokud vezmeme řadu čísel, která jsou pozorováním nějaké veličiny. Například ceny akcií na burze. Pokud se podíváme na první číslice každé z cen, předpokládali bychom, že každá z číslic $1 \ldots 9$ se v posloupnosti bude průměrně vyskytovat se stejnou četností – $11,1$ %. Ve skutečnosti tomu tak ovšem není. Pokud bude cen alespoň 100 a budou v rozsahu alespoň tří řádů, dostaneme výsledky podobné těm v Tab 3. Stejně tak se toto rozdělení projeví u náhodné posloupnosti, kde jednotlivá čísla nejsou rovnoměrně rozdělena v určitém intervalu, ale jedná se o násobky, které jsou rozděleny přes několik řádů.

Četnost výskytů jednotlivých čísel z intervalu $1 \ldots 9$ na místě první číslice se řídí pravidlem nazvaným *Benfordův zákon* (21). Pravděpodobnost, že jako první

1	$30,103\ \%$
2	$17,6091\ \%$
3	$12,4939\ \%$
4	$9,691\ \%$
5	$7,91812\ \%$
6	$6,69468\ \%$
7	$5,79919\ \%$
8	$5,11525\ \%$
9	$4,57575\ \%$

Tabulka 3.: Četnosti výskytu prvních číslic dle Benfordova zákona.

číslice daného čísla v desítkové soustavě bude číslo D, je daná jako:

$$P_D = \log_{10}(1 + \frac{1}{D}). \tag{21}$$

Pokud budeme předpokládat, že toto pravidlo platí, pak by mělo platit bez ohledu na zvolené měřítko. Máme-li tedy například vzdálenosti, je jedno zda je máme uvedeny v metrech nebo milimetrech. Rozdělení by mělo být neměnné. Vezmeme nyní funkci $p(x)$, která označuje hustotu rozdělení pravděpodobnosti veličiny a která je pro nás neznámá. Pokud je rozdělení neměnné, pro konstantu k platí, že [9]

$$p(kx) = f(k)p(x). \tag{22}$$

Protože pro hustotu rozdělení platí $\int p(x)\,dx = 1$, platí $\int p(kx)\,dx = \frac{1}{k}$. Z toho plyne, že $f(x) = \frac{1}{k}$. Budeme-li derivovat (22) vzhledem k proměnné k, tak pro $k = 1$ dostaneme diferenciální rovnici [9]

$$x \cdot p'(x) = -p(x) \tag{23}$$

a po po substituci $y = p(x)$

$$\begin{aligned} y' &= -\frac{1}{x}y \\ \int \frac{1}{y}\,dy &= -\int \frac{1}{x}\,dx \\ \ln(y) &= -\int \frac{1}{x}\,dx + \ln(C) \\ y &= C \cdot e^{-\int \frac{1}{x}\,dx} \\ y &= C \cdot e^{-\ln(x)} \\ y &= \frac{C}{x} \end{aligned} \tag{24}$$

dostaneme $p(x) = \frac{1}{x}$ za předpokladu, že integrační konstanta $C = 1$. Nyní potřebujeme vyjádřit pravděpodobnost, že číslo začínající číslicí D padne do intervalu mezi D a $D + 1$. Do tohoto intervalu padne každé číslo, které má jako první číslici D. V desítkové soustavě je počet všech možností dán jako

$$\int_1^{10} p(x)\,dx \tag{25}$$

a počet příznivých možností, kdy číslo padne do určeného intervalu je pak

$$\int_D^{D+1} p(x)\,dx. \tag{26}$$

Pravděpodobnost je potom

$$P_D = \frac{\int_D^{D+1} p(x)\,dx}{\int_1^{10} p(x)\,dx} = \frac{\ln(D+1) - \ln(D)}{\ln(10) - \ln(1)} = \log_{10}(D{+}1) - \log_{10}(D) = \log_{10}(1{+}\frac{1}{D}), \tag{27}$$

což je (21).

Poprvé se o tomto fenoménu zmínil v roce 1881 matematik a astronom Simon Newcomb bez dalšího komentáře. V roce 1938 tento jev popisuje podrobně fyzik Frank Benford, který si všiml, že logaritmické tabulky jsou více ohmatané na stranách s čísly, která začínají číslem 1. V dnešní době používají některé státy Benfordova zákona například k odhalování falešných daňových přiznání [8].

Nyní se zaměříme na to, jakým způsobem určit, zda je posloupnost čísel náhodná. Existuje více postupů, jak otestovat náhodnost posloupnosti. Já jsem v této práci zvolil dva. Nejprve si představíme χ^2 test (4.2.), což je základní statistický test. Poté popíšeme π test (4.3.), který jsem navrhl v této práci, kde náhodnost posloupnosti určíme pomocí aproximace čísla π.

4.2. χ^2 test

χ^2 test (chí-kvadrát test), také označovaný jako test dobré shody, je základním statistickým testem, který umožňuje určit, jak se pozorování jevů liší od předpokladu. Můžeme si to ukázat na házení šestibokou hrací kostkou. Máme-li klasickou hrací kostku, výsledkem hodu je číslo 1 až 6. Pokud není kostka upravená, je pravděpodobnost, že padne konkrétní číslo $P = \frac{1}{6}$. Provedeme-li 36 pokusů, mělo by každé číslo padnout 6 krát. Všech možností, jak lze 36 krát za sebou hodit hrací kostkou je 6^{36} a všechny možnosti jsou stejně pravděpodobné. I to, že 36 krát za sebou padne stejné číslo. Musíme mít tedy nějakou možnost poznat, kdy jsou hody opravdu náhodné a kdy se jedná o zásah zvenčí. Sice nemůžeme jednoznačně říci, zda jsou hody náhodné, ale můžeme s určitou pravděpodobností tvrdit, že by náhodné být měly. Použijeme k tomu

χ^2 test. Rozdělíme obor hodnot náhodné veličiny do r disjunktních částí. Je-li Y_i náhodná veličina s předpokládaným rozdělením, potom

$$\chi^2 = \sum_{i=1}^{r} Y_i^2 \tag{28}$$

má χ^2 rozdělení s v stupni volnosti [7]. Z (28) odvodíme

$$\chi^2 = \sum_{i=1}^{r} \frac{(x_i - np_i)^2}{np_i}, \tag{29}$$

kde x_i značí, kolikrát dané číslo i padlo, n je počet pokusů a p_i je pravděpodobnost, že jev nastane. Tedy np_i je předpokládaná hodnota x_i [1]. Dále budeme definovat $v = r - 1$, jako stupeň volnosti. Stupeň volnosti udává počet nezávisle stanovených kategorií, do kterých můžeme rozdělit výsledky všech pozorování – definiční obor. Pokud rozdělíme známý definiční obor náhodné veličiny na r částí, nezávislých je jich pouze $r - 1$. Vyjdeme-li z toho, že

$$n = \sum_{i=1}^{r} x_i, \tag{30}$$

kde n je definiční obor, $x_1, \ldots, x_r$ jsou počty pokusů v jednotlivých kategoriích a toho, že známe hodnotu n a hodnoty $x_1 \ldots x_{r-1}$, pak můžeme určit

$$x_r = n - (x_1 + \ldots + x_{r-1}). \tag{31}$$

Tedy x_r není nezávislá. Proto je $v = r - 1$.

Příklad 5. *Pokud použijeme hodnoty ze skutečných 36 hodů kostkou (Tab. 4.), máme $n = 36$. Vidíme, že 1 padla 3 krát, 2 padla 10 krát, atd. Můžeme tedy definovat $x_1 = 3, x_2 = 10, x_3 = 7, x_4 = 6, x_5 = 5$. I kdybychom nevěděli, že $x_6 = 5$, mohli bychom pomocí (31) určit, že*

$$x_6 = 36 - (3 + 10 + 7 + 6 + 5) = 5. \tag{32}$$

Stejně tak bychom mohli určit libovolnou x_i, pokud bychom znali zbývající.

Číslo(i)	1	2	3	4	5	6
Skutečnost(x_i)	3	10	7	6	5	5
Předpoklad(np_i)	6	6	6	6	6	6

Tabulka 4.: Skutečný a předpokládaný výsledek 36 hodů kostkou.

Máme-li hodnoty z pokusu s hrací kostkou (Tab. 4.), můžeme spočítat hodnotu χ^2.

Řádek Skutečnost(x_i) *označuje, kolikrát padlo dané číslo, řádek Předpoklad*(np_i) *označuje, kolikrát mělo dané číslo padnout dle pravděpodobnosti.*

$$\chi^2 = \frac{(3-6)^2}{6} + \frac{(10-6)^2}{6} + \frac{(7-6)^2}{6} + \frac{(6-6)^2}{6} + \frac{(5-6)^2}{6} + \frac{(5-6)^2}{6} \doteq 4,67 \quad (33)$$

Nyní ještě potřebujeme zjistit, zda výsledek $4,67$ z Př. 5 odpovídá našim předpokladům o náhodnosti hodu kostkou. Použijeme k tomu tabulku (Tab. 5.), kde jsou vypsané hodnoty χ^2 pro vybrané hladiny významnosti a stupně volnosti. Podíváme-li se tedy do tabulky (Tab. 5.), vidíme, že pro stupeň volnosti $v = 5$

	$p = 5\%$	$p = 25\%$	$p = 50\%$	$p = 75\%$	$p = 95\%$
$v = 1$	$0,00393$	$0,1015$	$0,4549$	$1,323$	$3,841$
$v = 2$	$0,1026$	$0,5754$	$1,386$	$2,773$	$5,991$
$v = 3$	$0,3518$	$1,213$	$2,366$	$4,108$	$7,815$
$v = 4$	$0,7107$	$1,923$	$3,357$	$5,385$	$9,488$
$v = 5$	$1,1455$	$2,675$	$4,351$	$6,626$	$11,07$
$v = 6$	$1,635$	$3,455$	$5,348$	$7,841$	$12,59$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Tabulka 5.: Hodnoty χ^2 pro vybrané hladiny významnosti a stupně volnosti [1].

se v hladině významnosti $p = 95\%$ nachází hodnota $11,07$. Hladina významnosti nám říká, že pokud probíhají pozorování jevu v souladu s předpokladem, je hodnota $\chi^2 \leq 11,07$ v 95% případů. Chceme-li testovat náhodnost posloupnosti, bylo by dobré, aby se hodnota χ^2 pohybovala kolem hodnoty pro hladinu významnosti 50%. Pokud bude totiž příliš nízká, bude se blížit našemu předpokladu, což ale znamená, že posloupnost výsledků pokusu není příliš náhodná (připomeňme, že předpoklad je v našem příkladu 6 jedniček, 6 dvojek, ...). Pokud bude příliš vysoká, znamená to pravděpodobně, že ve výsledku pokusů je nějaká anomálie – některé číslo padá častěji než ostatní (při skutečném házení kostkou se pravděpodobně jedná o upravenou kostku). V Př. 5 je hodnota $\chi^2 \doteq 4,67$, hodnota je tedy blízko hodnoty pro hladinu významnosti 50% ve stupni volnosti 5. Můžeme prohlásit, že test byl pravděpodobně proveden s neupravenou pravidelnou kostkou.

Pro test náhodnosti posloupnosti budeme simulovat házení kostkou a pro výsledek pozorování spočítáme hodnotu χ^2. Pokud bude hodnota $2,675 < \chi^2 < 6,626$ (tedy v rozsahu $p = 25\%$ až $p = 75\%$ ve stupni volnosti 5, viz Tab. 5.),

prohlásíme posloupnost za dostatečně náhodnou. Pro jednodušší a rychlejší implementaci si upravíme (29) následovně

$$\chi^2 = \frac{1}{np_i} \sum_{i=1}^{r} (x_i - np_i)^2, \tag{34}$$

což můžeme udělat, neboť při házení jednou kostkou je pravděpodobnost padnutí každého čísla $\frac{1}{6}$. Pomocí (34) pak implementujeme algoritmus 7.

(7) Chi-square-test

```
1: procedure CHI-SQUARE-TEST(sequence)                    ▷ Výstupem je χ²
2:     categories[6] ← 0, 0, 0, 0, 0, 0                   ▷ Kategorie
3:     for i ← 0, size-of(sequence) − 1 do
4:         n ← sequence[i] · 6
5:         categories[n − 1] ← categories[n − 1] + 1
6:     end for
7:     result ← 0
8:     np_i ← size-of(sequence)/6     ▷ Každé číslo padne s pravděpodobností 1/6
9:     for i ← 0, 5 do
10:        x ← categories[i] − np_i
11:        result ← result + x · x
12:    end for
13:    result ← result/np_i
14:    return result
15: end procedure
```

4.3. π test

V rámci této práce jsem navrhl test, který umožňuje určit náhodnost posloupnosti čísel pomocí aproximace hodnoty čísla π. Podstatou tohoto testu je geometrická pravděpodobnost a problém Buffonovy jehly [6]. Ten můžeme formulovat například takto:

„Mějme jehlu délky a a plochu, na které jsou nakresleny rovnoběžné linky ve vzdálenosti $2a$. Jestliže na tuto plochu pouštíme jehlu, pak se poměr celkového počtu pokusů n k počtu pokusů, kdy se jehla dotýká některé linky h, blíží číslu π, nebo-li $\frac{n}{h} \approx \pi$.“

Dále v této kapitole budeme označovat:

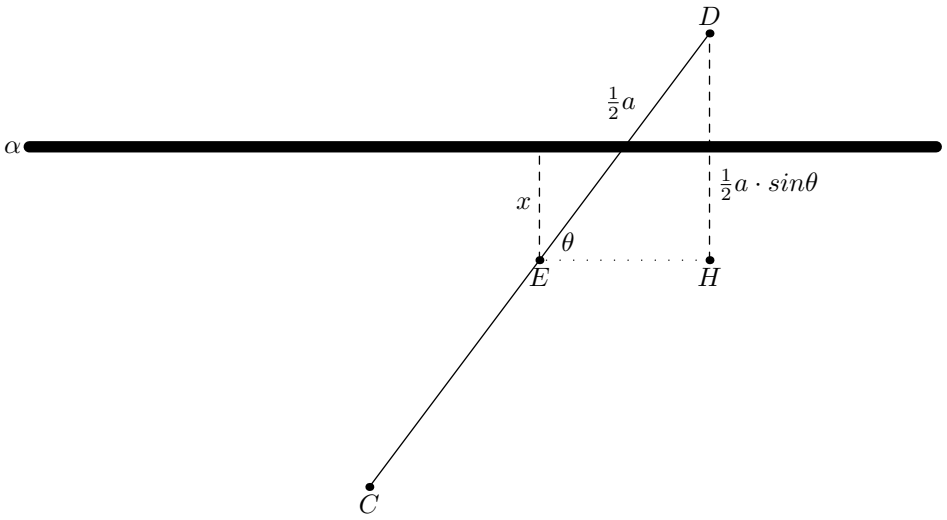
- a délka jehly,

- e vzdálenost mezi linkami ($e = 2a$),
- n počet pokusů,
- h počet pokusů, kdy se jehla dotýká linky nebo protíná linku.

Pravděpodobnost P, že jehla protne linku je

$$P = \frac{h}{n}. \tag{35}$$

Tuto pravděpodobnost si nyní vyjádříme také pomocí geometrické pravděpodobnosti, abychom mohli sestavit algoritmus pro testování posloupnosti náhodných čísel. Mějme tedy linky ve vzdálenosti $e = 2a$, jedna z linek (α) je protnuta jehlou, což je úsečka CD (Obr. 5.). Poloha jehly vzhledem k lince je dána 2 parametry. První je x, což je vzdálenost středu jehly E od nejbližší linky α a druhý je úhel θ, což je úhel, který svírá jehla s nejbližší linkou α (Obr. 5.). Střed jehly E nemůže být, při jakémkoliv dopadu jehly, ve větší vzdálenosti než



Obrázek 5.: Protnutí linky α jehlou (úsečka CD).

a vzhledem k nejbližší lince. Tedy všechny možnosti dopadu jehly jsou pak

$$\int_0^{\pi} a \, d\theta. \tag{36}$$

Aby jehla mohla protnout linku, je třeba, aby platilo $x \leq \frac{1}{2}a \cdot sin\theta$. Příznivé možnosti, kdy jehla protíná linku, můžeme vyjádřit jako

$$\int_0^{\pi} \frac{1}{2}a \cdot sin\theta \, d\theta. \tag{37}$$

Pravděpodobnost protnutí linky jehlou určujeme v rozsahu $\langle 0; \pi \rangle$. Pokud je úhel θ roven $\frac{\pi}{2}$, může x nabývat hodnot $\langle 0; \frac{a}{2} \rangle$, aby došlo k protnutí linky. Naopak, pokud je θ rovno 0 nebo π, musí být $x = 0$, jinak by k protnutí nedošlo. Pravděpodobnost je pak vyjádřena vztahem (38)

$$P = \frac{\int_0^\pi \frac{1}{2}a \cdot sin\theta \, d\theta}{\int_0^\pi a \, d\theta} = \frac{\frac{1}{2}a \int_0^\pi sin\theta \, d\theta}{a\pi} = \frac{1}{\pi}. \tag{38}$$

Přibližnou hodnotu čísla π získáme pomocí

$$\frac{h}{n} \approx \frac{1}{\pi}, \tag{39}$$

$$\pi \approx \frac{n}{h}. \tag{40}$$

Vydělíme počet všech pokusů počtem úspěšných pokusů (kdy jehla protne linku) a získáme přibližnou hodnotu π. Čím více pokusů provedeme, tím přesnější hodnotu získáme. Test je tedy založen na úvaze, že pokud budeme mít kvalitní posloupnost náhodných čísel, měli bychom se blížit k hodnotě čísla π.

V algoritmu budeme potřebovat 2 parametry, x pro vzdálenost středu jehly od nejbližší linky (E na Obr. 5.) v rozsahu $\langle 0; 1 \rangle$ a úhel θ pro velikost úhlu, který svírá jehla a linka v rozsahu $\langle 0; \pi \rangle$. Použijeme 2 náhodná čísla z posloupnosti tak, že $x = x_n$ a $\theta = \pi x_{n+1}$. Potom porovnáme, zda platí $x \leq 0,5 \cdot sin\theta$. Pokud platí, jehla protíná linku α a můžeme pokus označit jako úspěšný. Jestliže platí $x > 0,5 \cdot sin\theta$, potom jehla nemůže linku α protnout a pokus je neúspěšný. Nakonec vydělíme počet všech pokusů počtem úspěšných pokusů a výsledkem je aproximace čísla π, viz algoritmus 8.

4.4. Hodnocení generátorů náhodných posloupností

Zde si stručně představíme výsledky testování posloupností náhodných čísel, generovaných jednotlivými metodami představenými v této práci. Soubory s testovacími daty jsou přiloženy na CD v adresáři `data/`. Při hodnocení budu využívat výstupu z programu Vengeance 5.2.

4.4.1. Middle square a middle square improved

Nejprve otestujeme původní metodu middle square 2.2. a potom její variantu middle square improved. Testovací data pro tuto metodu jsou uložena v souborech *AAAmiddle-square-12354871.txt*, *AAAmiddle-square-87564398.txt* a *AAAmiddle-square-53111760440.txt*.

Jako semínko byla použita hodnota $x_0 = 12354871$ a testovalo se 100000 hodnot.

(8) Pi-test

```
1: procedure PI-TEST(sequence)                    ▷ Výstupem je aproximace čísla π
2:     n ← 0                                                  ▷ Počet všech pokusů
3:     h ← 0                                                  ▷ Počet úspěšných pokusů
4:     i ← 0
5:     while i < size-of(sequence) do
6:         x ← sequence[i]
7:         θ ← π · sequence[i + 1]
8:         if x ≤ 0.5 · sin(θ) then
9:             h ← h + 1
10:        end if
11:        n ← n + 1
12:        i ← i + 1
13:    end while
14:    return n/h
15: end procedure
```

```
**********
Chi-square test

Size of sequence: 100000
Chi-squared: 98817.531201
* Value is not in an expected interval. Randomness is insufficient.

**********
Pi test

Size of sequence: 100000
Pi approximation: 1.004046
Difference: 2.137546
Difference in %: 68.040213
```

Dále byla jako semínko použita hodnota $x_0 = 87564398$ a testovalo se 100000 hodnot.

```
**********
Chi-square test

Size of sequence: 100000
Chi-squared: 69949.731549
* Value is not in an expected interval. Randomness is insufficient.

**********
```

```
Pi test

Size of sequence: 100000
Pi approximation: 1.125809
Difference: 2.015783
Difference in %: 64.164381
```

Nakonec byla jako semínko použita hodnota $x_0 = 53111760440$ a testovalo se 1000000 hodnot.

```
**********
Chi-square test

Size of sequence: 1000000
Chi-squared: 40.428474
* Value is not in an expected interval. Randomness is insufficient.

**********
Pi test

Size of sequence: 1000000
Pi approximation: 3.148050
Difference: 0.006457
Difference in %: 0.205532
```

Nyní spustíme testy s daty vygenerovanými metodou middle square improved. Vstupní údaje jsou stejné. Data jsou uložena v souborech *AAAmiddle-square-improved-12354871.txt*, *AAAmiddle-square-improved-87564398.txt* a *AAAmiddle-square-improved-53111760440.txt*.

Jako semínko byla použita hodnota $x_0 = 12354871$ a testovalo se 100000 hodnot.

```
**********
Chi-square test

Size of sequence: 100000
Chi-squared: 45.219849
* Value is not in an expected interval. Randomness is insufficient.

**********
Pi test

Size of sequence: 100000
Pi approximation: 3.166260
```

```
Difference: 0.024668
Difference in %: 0.785197
```

Dále byla jako semínko použita hodnota $x_0 = 87564398$ a testovalo se 100000 hodnot.

```
**********
Chi-square test

Size of sequence: 100000
Chi-squared: 39.407536
* Value is not in an expected interval. Randomness is insufficient.

**********
Pi test

Size of sequence: 100000
Pi approximation: 3.160057
Difference: 0.018464
Difference in %: 0.587735
```

Nakonec byla jako semínko použita hodnota $x_0 = 53111760440$ a testovalo se 1000000 hodnot.

```
**********
Chi-square test

Size of sequence: 1000000
Chi-squared: 61.476306
* Value is not in an expected interval. Randomness is insufficient.

**********
Pi test

Size of sequence: 1000000
Pi approximation: 3.138161
Difference: 0.003432
Difference in %: 0.109244
```

Jak vidíme v tabulce Tab. 6., metoda middle square improved dává obvykle lepší výsledky. To, že hodnota χ^2 je mimo očekávaný rozsah je dáno nepříliš kvalitním algorimem RENEW viz algoritmus 3. Bylo by tedy vhodné tento algoritmus vylepšit. Zajímavý výsledek pak dává poslední test, kdy je použit větší rozsah generování čísel, kdy jsou výsledky obou metod srovnatelné. To je způsobeno tím,

	middle square		middle square imp.	
	χ^2	$\pi \approx$	χ^2	$\pi \approx$
$x_0 = 12354871$	$98817,531201$	$1,004046$	$45,219849$	$3,166260$
$x_0 = 87564398$	$69949,731549$	$1,125809$	$39,407536$	$3,160057$
$x_0 = 53111760440$	$40,428474$	$3,148050$	$61,476306$	$3,138161$

Tabulka 6.: Porovnání výsledků metod MIDDLE SQUARE a MIDDLE SQUARE IMPROVED.

že v případě většího rozsahu hodnot, je menší pravděpodobnost výskytu problematického členu posloupnosti. Co je problematický člen je popsáno v části 2.2.. Pro praktické použití lze tedy doporučit spíše metodu middle square improved, i když v případě, že se použijí vhodně zvolené parametry, může i metoda middle square dávat slušné výsledky.

4.4.2. Lineární kongruentní metoda

Do rodiny lineární kongruentních patří i generátor RANDU 2.3.1., který nyní otestujeme. Test provedeme s daty v souboru *AAArandu-1.txt*. Jako semínko je zvoleno $x_0 = 1$.

```
**********
Chi-square test

Size of sequence: 100000
Chi-squared: 5.240370
* Value is in an expected interval. Randomness is sufficient.

**********
Pi test

Size of sequence: 100000
Pi approximation: 3.162755
Difference: 0.021163
Difference in %: 0.673631
```

Druhý test provedeme s daty v souboru *AAArandu-16385.txt*. Semínko je v tomto případě $x_0 = 16385$. Jedná se tedy opět o liché číslo, což je vstupní podmínka pro generátor RANDU.

```
**********
Chi-square test
```

```
Size of sequence: 100000
Chi-squared: 4.069003
* Value is in an expected interval. Randomness is sufficient.

**********
Pi test

Size of sequence: 100000
Pi approximation: 3.137058
Difference: 0.004535
Difference in %: 0.144340
```

Vidíme, že výsledky jsou dobré a algoritmus je pro určité typy úloh vhodný. Bohužel jeho vlastnost, kdy se vygenerované body v trojrozměrném prostoru uspořádávají do paralelních rovin, tyto statistické testy neodhalí. Při praktickém použití bychom měli zvážit, zda tato vlastnost neovlivní kvalitu výsledků. Ve prospěch RANDU hovoří především jednoduchá a rychlá implementace.

4.4.3. Difuze

Jako poslední otestujeme metodu získávání náhodných čísel pomocí pozorování difuze v kapalinách. Tato metoda se od předchozích liší, neboť se nejedná o pseudonáhodná čísla, generovaná matematickým předpisem. Data jsou uložena v souboru *AAAdiffusion.txt*.

```
**********
Chi-square test

Size of sequence: 1318
Chi-squared: 2.812785
* Value is in an expected interval. Randomness is sufficient.

**********
Pi test

Size of sequence: 1318
Pi approximation: 3.138095
Difference: 0.003497
Difference in %: 0.111326
```

Hodnota χ^2 je v předpokládaném rozsahu a ukazuje na dobrou náhodnost posloupnosti. Hodnota aproximace π je také velice dobrá, i když máme k dispozici pouze 1318 hodnot. Aby byla aproximace čísla π přesnější, je třeba otestovat

v ideální případě alespoň desetitisíce hodnot. Tato metoda se jeví jako vhodná k získávání náhodných čísel a při použití dokonalejší hašovací funkce by mohla být použitelná i pro kryptografii. Bylo by ovšem vhodné provést testování na větším množství dat.

5. Knihovna random a program Vengeance

5.1. Knihovna random

V rámci této práce jsem implementoval všechny popisované algoritmy v programovacím jazyku C. Tento programovací jazyk jsem zvolil z důvodu dostupnosti prakticky na všech platformách, snadnému linkování z různými programovacími jazyky a také kvůli vysokému výkonu výsledného kódu. Knihovnu by mělo být možné přeložit a používat na nejběžnějších platformách. Já jsem ji vyvíjel a testoval na systému OS X. V této kapitole jsou popsány veřejně dostupné funkce této knihovny.

Knihovna obsahuje následující funkce:

```
unsigned long long middle_square(unsigned long long previous,
    int digits)
```

Toto je implementace původní metody middle square (2.2.). Vstupem je předchozí člen posloupnosti a počet číslic výsledného čísla.

```
unsigned long long middle_square_improved(unsigned long long previous,
    int digits)
```

Tato funkce implementuje upravenou metodu middle square, jak je popsána v příslušné kapitole (2.2.). Vstupem je předchozí člen posloupnosti a počet číslic výsledného čísla.

```
unsigned long long lcm(unsigned long long m, int a, int c,
    unsigned long long x0)
```

Tato funkce implementuje obecný lineární kongruentní generátor (2.2.). Vstupem jsou parametry m – modul, a – multiplikativní konstanta, c – aditivní konstanta a předchozí člen posloupnosti.

```
unsigned int randu(unsigned int previous)
```

Tato funkce implementuje generátor RANDU (2.3.1.). I když by šel použít obecný lineární kongruentní generátor, neprojevila by se hlavní výhoda generátoru – vysoká rychlost. Vstupem je předchozí člen posloupnosti. Jako semínko se používá liché číslo.

```
unsigned int diffusion(char *image)
```

Implementace metody získávání náhodných čísel ze snímku difuze. Jako vstup se předává adresa řetězce, reprezentující horizontální řádek snímku. Formát dat je popsán v kapitole 3.2.

```
unsigned long long blum_blum_shub(unsigned long long *previous,
    unsigned long long modulus)
```

Implementace metody Blum Blum Shub (2.3.2.). Pro kvalitnější výstup by bylo vhodnější použít větší čísla. Z praktických důvodů zde používám pouze 64 bitová. Při použití větších čísel by bylo třeba použít externí knihovnu. Vstupem je předchozí hodnota a modul.

Zdrojové kódy knihovny jsou uloženy v souborech `random.h` a `random.c`. Po přeložení překladačem vznikne binární knihovna, kterou je již možno použít v programu napsaném v jazyku C. Takto ji používá program Vengeance 5.2.

5.2. Program Vengeance

Pro testování náhodnosti posloupnosti čísel s uniformním rozdělením (2.1.), jsem v rámci této práce vytvořil program Vengeance. Umožňuje otestovat náhodnost zadané posloupnosti, případně i vygenerovat posloupnost pomocí knihovny random (5.1.). Generování posloupnosti je dostupné pouze ve verzi s GUI. Program je možno provozovat jako GUI aplikaci na operačním systému OS X nebo jako aplikaci pro textový terminál, kde by aplikace měla jít přeložit na nejběžnějších systémech s překladačem jazyka C. GUI verze aplikace je sestavena a testována na systému OS X verze 10.8.

5.2.1. Uživatelský pohled

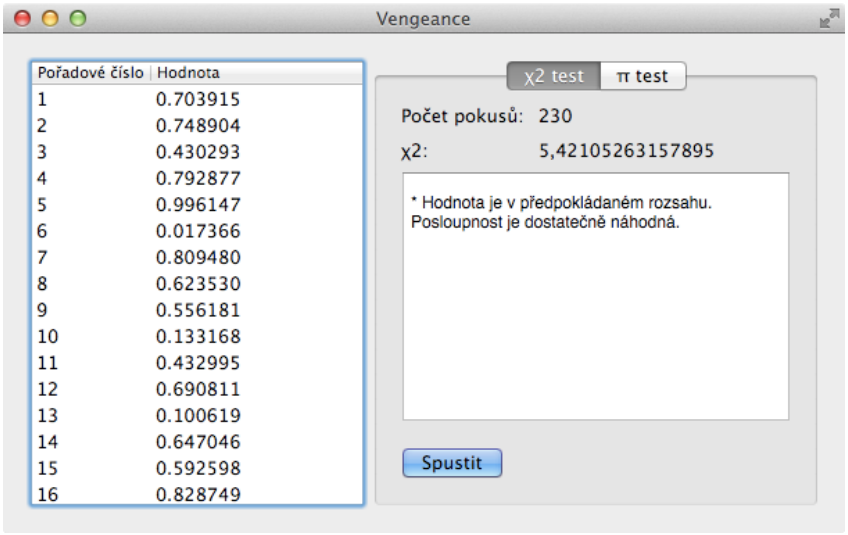
V této části si představíme aplikaci z pohledu uživatele. Nejprve se podíváme na GUI verzi a potom se zběžně zmíníme o verzi pro příkazovou řádku.

Na Obr. 6. vidíme, jak vypadá hlavní okno aplikace. V levé části je tabulka, která zobrazuje posloupnost čísel. Pravá část je vyhrazena pro zobrazení výsledků testů. Pomocí záložek se lze přepínat mezi oběma testy. Tlačítkem *Spustit* je možno spustit vybraný test.

Aplikace se dále ovládá z hlavního menu. Popíšeme si pouze položky, které se týkají přímo aplikace Vengeance, zbytek položek v menu je standardní pro každou aplikaci na systému OS X.

Nejprve si popíšeme položky v nabídce *Soubor*. Ta obsahuje tyto položky:

- *Nová middle square...* vytvoří novou posloupnost pomocí metody middle square;



Obrázek 6.: Hlavní okno aplikace VENGEANCE v GUI verzi.

- *Nová LC...* vytvoří novou posloupnost pomocí lineární kongruentní metody;
- *Nová RANDU...* vytvoří novou posloupnost pomocí metody RANDU;
- *Otevřít...* otevře soubor, který obsahuje číselnou posloupnost.

Data v souboru by měla být uspořádána tak, že každé číslo je na zvláštním řádku a jako oddělovač desetinných míst se používá znak „.“. Například:

```
0.0000305190
0.0001831097
0.0008239872
0.0032959362
```

Další položkou v menu je nabídka *Akce*, kde se nacházejí položky pro ovládání jednotlivých testů. Obsahuje tyto položky:

- *Spustit χ^2 test* vyvolá stejnou akci, jako stisknutí tlačítka *Spustit* na záložce χ^2 test;
- *Spustit π test* vyvolá stejnou akci, jako stisknutí tlačítka *Spustit* na záložce π test.

V aplikaci je také vestavěna základní nápověda, kde je stručně popsáno ovládání aplikace.

Aplikace VENGEANCE je dostupná také jako program určený pro příkazovou řádku. V této verzi nefunguje generování posloupností, je zaměřena pouze na testování již vytvořené posloupnosti ze souboru. Pokud program vyvoláme bez parametrů například příkazem `./vengeancecmd`, program vypíše nápovědu použití.

Pokud chceme provést testy, je třeba program vyvolat pomocí `./vengeancecmd jmeno-souboru` a program načte příslušný soubor a spustí oba testy. Výsledek pak vypíše na obrazovku.

5.2.2. Programátorský pohled

Výkonná část aplikace VENGEANCE je platformně nezávislá. Abychom toho dosáhli, jsou výkonné algoritmy naprogramovány v jazyku C. Tyto algoritmy jsou uloženy v souborech `test_module.h` a `test_module.c`. Funkce z tohoto modulu jsou pak volány jak z GUI aplikace, tak z aplikace pro příkazovou řádku.

Modul `test_module` obsahuje funkce:

```
int load_data(char *)
```

Načte data ze zadaného souboru do paměti. Postará se rovněž o alokaci dostatečného místa v paměti. Pokud nastane chyba, vrátí ji v návratovém kódu.

```
double chi_square(double *, int)
```

Spočítá hodnotu χ^2 pro zadanou posloupnost. Jako druhý parametr je počet čísel v posloupnosti.

```
double pi_test(double *, int)
```

Spočítá aproximaci hodnoty π pro zadanou posloupnost. Jako druhý parametr je počet čísel v posloupnosti.

```
int new_middle_square(unsigned long long, int, int)
int new_middle_square_improved(unsigned long long, int, int)
int new_lcm(unsigned long long, int, int, unsigned long long, int)
int new_randu(unsigned int, int)
```

Vygenerují příslušné posloupnosti pomocí metod, které jsou implementované v knihovně random 5.1.

Textová verze aplikace dále obsahuje v souboru `main.c` obslužný kód pro zajištění komunikace přes příkazovou řádku.

GUI verze aplikace je napsána v programovacím jazyku Objective-C a obsahuje 2 třídy, které jsou nutné pro obsluhu grafického rozhraní. Jedná se o třídy:

- *NITableViewDelegate* je delegátní třída, která má na starosti obsluhu zobrazování hodnot v tabulce, která je v levé části hlavního okna aplikace (Obr. 6.);
- *NIAppDelegate* je delegátní třída celé aplikace, kde jsou ošetřeny veškeré události v GUI.

V GUI verzi je pak ještě přítomen soubor `MainMenu.xib`, který obsahuje samotné hlavní okno aplikace, menu a propojení na metody třídy *NIAppDelegate*.

Sestavení binární aplikace je snadné. Stačí načíst příslušný projekt do vývojového prostředí *Xcode* a v menu zvolit *Product – Build*.

Závěr

Tato práce představuje problematiku získávání a testování posloupností náhodných čísel. Jsou navržena vylepšení stávající metody middle square a je také navržena nová metoda získávání náhodných čísel sledováním fyzikálního jevu difuze. V práci jsou popsány dvě metody testování náhodnosti číselných posloupností. První metoda je standardní statistický test, druhá je nově navržený test pro testování náhodnosti pomocí aproximace hodnoty π. Posloupnosti, generované pomocí metod uvedených v této práci, jsou pomocí těchto testů zhodnoceny a jsou také uvedeny doporučení pro zlepšení výsledků. Všechny metody jsou implementovány v programovacím jazyku C a byla také vytvořena aplikace Vengeance pro testování posloupností náhodných čísel.

Další rozšíření práce by se mohlo zaměřit na popis dalších metod pro generování náhodných čísel a také na přidání dalších testů do aplikace Vengeance.

Conclusions

This thesis introduces problematics of obtaining and testing of random numbers sequences. There are proposals for improvements of existing method middle square and there is also design of new method for obtaining random numbers by observing a physical phenomena called diffusion. Two methods for testing randomness of number sequences are described. First method is standard statistical test, second is newly designed test for randomness by approximation of π. Sequences generated by methods described in this thesis is tested with those tests and some recommendation for improvements are given. At last, all methods are implemented in C programming language and application Vengeance for testing of random numbers sequences was created.

Future enhancements could be focused mainly on describing more methods for generating random numbers and adding some more tests to Vengeance.

Reference

[1] Knuth, Donald E. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms.* Addison-Wesley, third edition, 1998.

[2] Kapadia, Apu; Lu, Charng-da *Random Numbers Generator.* `http://www.cs.indiana.edu/ kapadia/project2/project2.html`, Elektronická publikace. 2001 (citováno 1.2.2013).

[3] Thau, Robert S. *Alan Turing's Manual for the Ferranti Mk. I.* `http://www.computer50.org/kgill/mark1/RobertTau/turing.html`, Elektronická publikace. 2000 (citováno 1.2.2013).

[4] Marsaglia, George *Random Numbers Fall Mainly in the Planes.* In *Proc National Academy of Sciences* 61 (1). 1968. s. 25–28.

[5] Mende, Robert G.; Noll, Landon C. *How Lavarand Works.* `http://web.archive.org/web/19980521144845/ http://lavarand.sgi.com/cgi-bin/how.cgi`, Elektronická publikace. 1998 (citováno 17.3.2013).

[6] Weisstein, Eric W. *Buffon's Needle Problem.* `http://mathworld.wolfram.com/BuffonsNeedleProblem.html`, Elektronická publikace. 2013 (citováno 2.4.2013).

[7] Weisstein, Eric W. *Chi-Squared Distribution.* `http://mathworld.wolfram.com/Chi-SquaredDistribution.html`, Elektronická publikace. 2013 (citováno 11.4.2013).

[8] Ziegler, Günther M. *Matematika vám to spočítá: Příběhy královny věd.* Knižní klub, Vydání 1., Praha 2011.

[9] Weisstein, Eric W. *Benford's Law.* `http://mathworld.wolfram.com/BenfordsLaw.html`, Elektronická publikace. 2013 (citováno 30.4.2013).

A. Obsah přiloženého CD

V samotném závěru práce je uveden stručný popis obsahu přiloženého CD, tj. závazné adresářové struktury, důležitých souborů apod.

`bin/`
: Spustitelný program Vengeance ve formátu Disk Image, který je běžně používaný na platformě OS X k distribuci aplikací.

`doc/`
: Dokumentace práce ve formátu PDF, vytvořená dle závazného stylu KI PřF pro diplomové práce, včetně všech příloh, a všechny soubory nutné pro bezproblémové vygenerování PDF souboru dokumentace (v ZIP archivu), tj. zdrojový text dokumentace, vložené obrázky, apod.

`src/`
: Kompletní zdrojové texty programu Vengeance se všemi potřebnými zdrojovými texty, knihovnami a dalšími soubory pro bezproblémové vytvoření spustitelných verzí programu.

`readme.txt`
: Instrukce pro instalaci a spuštění programu Vengeance, včetně požadavků pro jeho provoz.

Navíc CD obsahuje:

`data/`
: Ukázková a testovací data použitá v práci a pro potřeby obhajoby práce.