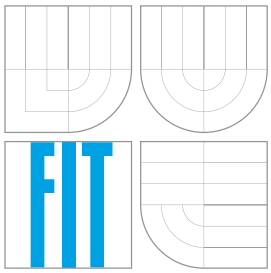
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

GENEROVÁNÍ PSEUDONÁHODNÝCH ČÍSEL

GENERATING THE PSEUDO-RANDOM NUMBERS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — VOJTĚCH ŠTULÍR
AUTHOR

VEDOUCÍ PRÁCE — Ing. LUKÁŠ GRULICH
SUPERVISOR

BRNO 2008

Abstrakt

Tato bakalářská práce pojednává o generování pseudonáhodných čísel pro použití v počítačové technice. Nastiňuje možnosti a způsoby generování posloupností čísel, popisuje různé druhy generátorů a jejich princip činnosti. Dále pak pojednává o transformaci rovnoměrného rozložení generovaných posloupností na rozložení jiná. Jedna z částí se zabývá testováním a určováním kvality výsledné posloupnosti čísel. Poslední část práce ukazuje vlastní implementaci některých popsaných částí v textu.

Klíčová slova

Generátor pseudonáhodných čísel, transformace rozložení, PRNG, LCG, statistiský test, hypotéza

Abstract

This bachelor's thesis is about pseudorandom generators which are used for computers technology. It is written in different kinds of these generators and their working principle. And also this thesis deals with analysis transformation of generated number sequence on others. One part is studied testing and define consequent quality of number sequence. Last part of this thesis is shown my own implementation of something part in text.

Keywords

Pseudo-random numbers generator, transformation, PRNG, LCG, statistical test, hypothesis

Citace

Vojtěch Štulír: Generování pseudonáhodných čísel, bakalářská práce, Brno, FIT VUT v Brně, 2008

Generování pseudonáhodných čísel

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Lukáše Grulicha. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vojtěch Štulír
6. května 2008

Obsah

Kapitola 1

Úvod

S rozvíjející se počítačovou technikou a její výkonností se zvyšuje i její nasazení ve výzkumném a komerčním odvětví. Často je při tom potřeba simulovat události či jevy, které v reálném světě nelze jednoduše napodobit. Jde především o situace trvající příliš krátkou, nebo právě naopak velmi dlouhou dobu. V těchto případech nelze zachytit jednoduše jednotlivé stavy rychle probíhajícího procesu anebo by pozorování trvalo příliš dlouho, aby studie přinesly efektivní výsledky. Někdy dokonce ani není v lidských silách dosáhnout výsledku. Proto se tyto jevy začaly zkoumat pomocí počítačové techniky zvané modelování a simulace reálného světa.

Tyto techniky napodobují s větší či menší přesností reálnou událost podle zadaných kritérií a požadavků. Protože v reálném světě se často vyskytují náhodné jevy, nebo naopak lze náhodnými jevy popsat rozmanitost reálného světa, vznikl požadavek na jejich generování. Nejčastěji se generují v číselné podobě, jež označuje v kontextu stav věcí či událostí.

Jedním z oborů využívající technik generování náhodných čísel, je právě obor modelování a simulace, pomocí něhož se snažíme předpovědět přibližný stav či vývoj situace, množství, rozšíření aj. K tomu je potřeba znát buď již podobné výsledky z minulosti, nebo je nutné je určitým způsobem odsimulovat. Protože v reálném světě nelze počítat s přesným numerickým či analytickým chováním, je potřeba do simulace předpovídající statistiku, vnést nahodilost některých situací. A jsme zpátky u nutnosti získávání náhodných jevů.

Jiná odvětví, kde se využívají náhodná čísla v informačních technologiích, jsou síťové komunikace a jejich zabezpečení. Široké rozšíření elektronické komunikace sebou nese i riziko jejího zneužití a rušení od jiných zdrojů. Lze poměrně snadno odchytávat takové přenosy v rozsáhlejších sítích a především v síti Internet, která je v dnešní době tou největší. Internet je užíván širokou škálou různých osob, proto vzniká nutnost tyto komunikace zabezpečit. Tím se zabývají obory kryptografie, využívající v hojné míře právě náhodné hodnoty.

V těchto a zajisté i v mnoha jiných případech, je tedy nutností pracovat s posloupnostmi náhodných čísel. Ty se musí určitým mechanismem generovat a testovat na kvalitu takto vzniklých sekvencí čísel. Problematika je popisována na dalších stranách práce.

Text práce je rozdělen podle logických celků do několika kapitol v pořadí daném postupem řešení problému témata. V první kapitole jsou popsány obecné požadavky na generování čísel, možnosti způsobu generování a s tím spojené druhy generátorů. Následující kapitola stručně nastiňuje problém pravděpodobnostního rozložení a druhy typů rozložení.

Na tuto část těsně navazuje v téže kapitole oddíl, zabývající se transformacemi rovnoměrného rozložení na rozložení jiné, požadované.

Další kapitola teoretického rozboru práce rozebírá podrobněji požadavky na kvalitu generované posloupnosti pseudonáhodných čísel a určení splnění těchto požadavků pomocí různých statistických testů. V poslední kapitole je popsána praktická část této práce a to především návrh a implementace některých druhů generátorů a výsledky jejich zkoumání pomocí navržených testů.

Kapitola 2

Náhodné a pseudonáhodné generátory čísel

Ve skutečnosti musíme rozlišovat dva poměrně rozdílné pojmy. Je to generování *náhodných* čísel a generování *pseudonáhodných* čísel. V této kapitole si tyto dva pojmy vysvětlíme a dále si rozdělíme různé typy generátorů.

2.1 Generátory náhodných čísel

Opravdu náhodnou sekvenci čísel lze získat pouze pomocí fyzikálních, mechanických či chemických generátorů. Jde vždy o nedeterministické generátory, které pracují na principu přírodního jevu s náhodným chováním. Využívá se určité vlastnosti s náhodným chováním v daném fyzikálním jevu. Jako nejpoužívanější příklad fyzikálních generátorů v praxi lze uvést šumové generátory založené na vlastnostech polovodičových přechodů.

Zajímavé jsou také radioaktivní zářiče s detektorem, kdy se jako náhodnosti využije počet vyzářených elektronů.

V běžném životě se člověk nejčastěji setkává s trochu odlišnými druhy mechanických generátorů. Jsou to například obyčejná hrací kostka, nebo mince, pokud ji používáme pro rozhodování podle hodu. Propracovanější mechanizmy jsou v aplikacích sázkových her. Tyto generátory jsou teoreticky deterministické, ale díky mnoha neznámým vstupním parametrům a počátečnímu stavu, lze je jen s velkými obtížemi a nebo vůbec opakovat [19].

Dříve byly pro jednouché manuální výpočty užitečnou pomůckou tzv. tabulky náhodných čísel. K jejich vytvoření se využívalo rozsáhlých statistických souborů dat získaných v jiném odvětví. Příkladem by mohl být počet obyvatel v městech seřazených a vybraných jistým filtrem.

2.1.1 Výhody a nevýhody používání

Generované posloupnosti čísel jsou opravdu náhodné a nelze je opakovat. To je sice z hlediska náhodnosti dobrá zpráva, avšak pro další použití je to i jistá nevýhoda. Vznikají potíže s implementací algoritmů, pro něž jsou čísla generována, a jejich testováním. Protože se posloupnost náhodných čísel nikdy neopakuje, nelze snadno ladit vznikající aplikace a testovat jejich správnost.

Další nevýhodou je udržování stability těchto generátorů. Ta je velmi závislá na okolních vlivech a poměrně těžce udržitelná. Malá změna okolí může vyvolat nečekně rozsáhlé od-

chylky ve výsledném výstupu generátoru. Je tedy zapotřebí speciální technické vybavení, které může stát nemalé částky.

V neposlední řadě je zde problém počítačového zpracování výsledků generování. Pokud je obtížné data dále zpracovávat počítačovou technikou, jsou pro simulaci či jiné použití nevhodné a nepoužitelné.

Proto se ve velké převaze v informačních technologiích používají generátory pseudonáhodných čísel.

2.2 Generátory pseudonáhodných čísel PRNG

Generátorem pseudonáhodných čísel je vždy určitý algoritmus. Algoritmus je jednoznačná posloupnost konečného počtu elementárních kroků, vedoucí k řešení daného problému. Z toho vyplývá, že tyto generátory jsou deterministické a lze při zachování vstupních podmínek jejich činnost opakovat se stejným výsledkem. Generující algoritmy využívají rekurze, tzn. pro výpočet nové hodnoty používají předchozí hodnoty a aktuálního stavu algoritmu. Vidíme tak, že posloupnost není čistě náhodná, ale generuje se podle určitých pravidel a můžeme ji opakovat.

Označují se jako aritmetické generátory a často se objevují pod zkratkou PRNG, což vychází z anglického označení Pseudo-Random Number Generator.

Zvláštním poddruhem těchto generátorů jsou, jak už bylo zmíněno i v úvodu, bezpečnostní generátory pseudonáhodných čísel používaných v kryptografii. Jsou na ně kladeny jiné požadavky než na generátory určené pro simulace. Označujeme je CSPRNG a hlavním kritériem je jejich rezistence na různé druhy útoků a snahu o prolomení jejich bezpečnosti.

2.2.1 Výhody a nevýhody používání

Pseudonáhodné generátory se tedy řídí podle algoritmu. Ten nám zaručuje determinismus získaných hodnot. U těchto generátorů avšak musíme vzít na vědomí, že posloupnost generovaných čísel se po nějaké době bude opakovat – mají svou periodu. Zvláště u jednodušších generátorů se musí dát pozor, aby perioda nebyla příliš krátká. Pokud generátor narazí na číslo již vygenerované, budou se opakovat všechny následující hodnoty stejně jako v předešlém průchodu a perioda bude kratší, než jsme požadovali.

Vždy se snažíme o co nejdelší volenou periodu, proto musí být generátor navrhnut tak, aby prošel celou periodou aniž by se dostal do nějaké dřívější, menší smyčky. Z důvodu dosažení co nejdelší periody, vznikají rozmanité druhy generátorů.

2.2.2 Požadované vlastnosti

Z předešlé části vyplývají požadované vlastnosti na generátory pseudonáhodných čísel. Pokud si je popíšeme pomocí rekurentního zápisu:

$$x_i = F(x_{i-1}, x_{i-2}, \ldots, x_{i-m})$$

, lze požadavky na algoritmus generátorů shrnout takto:

- perioda generátoru (označovaná často p taková, že $x_{i+p} = x_i$, kde p je nejmenší možné číslo) je co nejdelší

- rozložení x_i je v požadovaném intervalu $\langle a, b\rangle$ rovnoměrné, což značíme $\mathrm{R}(a, b)$. Většinou požadujeme u těchto generátorů normované rovnoměrné rozložení v intervalu $\langle 0, 1\rangle$ zapsané jako $\mathrm{R}(0, 1)$
- páry (x_i, x_{i+1}), trojice (x_i, x_{i+1}, x_{i+2}), ..., jsou nekorelované
- páry $(f_0(x_i), f_1(x_{i+1}))$, trojice $(f_0(x_i), f_1(x_{i+1}), f_1(x_{i+2}))$, ..., funkcí jsou nekorelované
- rychlý výpočet

Výše uvedené požadované vlastnosti jsou obecně požadované u všech generátorů. Podrobněji jsou různé vlastnosti celých generátorů popsány v podkapitole 4.1 kapitoly 4

Vedle velmi jednoduchých způsobů vytváření pseudonáhodných posloupností, jsou známy i propracovanější techniky používající se úspěšně v praxi. Dvěma nejpoužívanějšími typy generátorů jsou generátory s posuvným registrem a kongruentní generátor. Dále existují složitější generátory jako například Mersenne Twister, Blum Blum Shub a další.

2.2.3 Jednoduché techniky generování pseudonáhodných čísel

V této kapitole se pouze stručně zmíníme o starších a velmi primitivních principech generování čísel. Tyto jednoduché techniky příliš nevyhovují požadavkům na kvalitní generátor a ukážeme si je zde pouze okrajově. Protože však některé z těchto způsobů můžou být základem pro složitější odvozené metody, popíšeme si alespoň jejich základní myšlenku:

1. Čísla s dostatečným počtem číslic vynásobíme konstantou a z výsledku vypíšeme k prostředních číslic. Takto získané čísla lze použít pro vytvoření k-místných pseudonáhodných čísel z intervalu $\langle 0, 1\rangle$.
   Např.: Po vynásobení konstantou dostaneme výsledek $v = 376425$ a z něj vybereme prostřední čtyři číslice, tedy 7642. Výsledné číslo je pak $x_0 = 0,7642$.
2. Číslo s dostatečným počtem číslic umocníme na druhou, mocninu vydělíme dostatečně velkým prvočíslem. Získaný zbytek považujeme za pseudonáhodné číslo z intervalu $\langle 0, 1\rangle$. Pracujeme podle vzorce $x_{i+1} = x_i^2 (\mathrm{mod}\ p)$.
3. Využijeme cifer Ludolfova čísla π jako zdroje náhodných čísel. Tento způsob generování vyplývá ze zjištění, že π je tzv. normální číslo. Názvem normální číslo označujeme číslo, v němž frekvence výskytu každé skupiny k číslic ($k \in \mathbb{N}$), v nekonečné posloupnosti číslic za desetinou čárkou se asymptoticky blíží číslu 10^{-k}.

Podrobnější informace lze hledat v knize panů Fabiána a Kluibera [4].

2.2.4 Generátory s posuvným registrem LFSR

Principem a jádrem tohoto druhu generátorů je využití funkce posuvných registrů s lineární zpětnou vazbou. Proto se snadno implementují jako hardwarové generátory, ale lze je implementovat i jako algoritmy. Jsou označovány zkratkou LFSR z anglického Linear Feedback Shift Register.

Jak už bylo výše napsáno, používají tyto generátory pro svou činnost funkci posuvného registru a funkci zvolené bitové operace. Bitovou operací bývá nejčastěji funkce bitový

exkluzivní OR neboli XOR. Operaci použijeme s vybranými vstupními bity předešlého čísla a její výsledek pomocí posuvné funkce zapíšeme do posuvného registru a tak získáme výsledné číslo posloupnosti.

Konkrétními příklady LFSR generátorů, lišící se způsobem použití funkce XOR, přesněji jejich *pořadím* použití na dané bity, jsou:

- Fibonacciho LFSR generátor
- Galoisův LFSR generátor
- Geefeho LFSR generátor
- Self – shrinking LFSR generátor
- Bit – search LFSR generátor
- ...

Blíže jsou tyto druhy generátorů popsané například v práci [14].

Čísla bitů sloužící pro vstupy logických členů XOR těchto generátorů, jsou určeny pomocí tabulky „zpětnovazebních“ polynomů pro maximální LFSR generátor. Tuto tabulku můžeme vidět například v publikaci [20].

Tyto generátory dokáží vytvářet dlouhé pseudonáhodné sekvence s dobrým statistickým rozložením. Často jich bývá využito k vytváření složitějších generátorů jako jedna z jeho částí, například tak, že jich použijeme více v nelineární kombinaci.

2.2.5 Lineární kongruentní generátory LCG

Jsou nejjednodušším typem generátorů navržené D. H. Lehmerem. Zkratka vynikla z anglického Linear Congruential Generator. Vycházejí z rekurze závislé pouze na jednom předešlém členu posloupnosti. Jejich činnost je založena na vytváření posloupnosti čísel podle jednoduché funkce dané předpisem:

$$x_{i+1} = (a\, x_i + b) \,(\mathrm{mod}\ m)$$

, kde operace mod znamená zbytek po celočíselném dělení, konstanta a je multiplikační, konstanta b aditivní, někdy také označována jako inkrementační, a m představuje modul. Při inicializaci generátoru musíme ještě určit počáteční hodnotu x_0, označovanou jako semínko. Volí se v různých případech jako nezáporné celé číslo. Někdy je i požadavek, aby bylo liché nebo nula.

Pomocí těchto hodnot nastavujeme délku periody p generátoru a rozložení výsledné posloupnosti. Modul m volíme co největší, protože nám označuje největší možnou periodu. Nejčastěji se volí jako mocnina 2, tedy 2^n , kde n je počet bitů používaného datového typu. To nám s využitím implicitních vlastností při výpočtech celočíselných typů ušetří při implementaci časově náročnou operaci modulo. V dnešní době se stále častěji objevují generátory s modulem m sníženým o určitou hodnotu, jako například $2^n - 6$ a podobně. Více v [9].

Konstanty a a b se také volí s ohledem na co největší periodu generátoru. Ideálně tedy tak, aby perioda byla přímo m. To je však velmi složité a často se spíše volí z praxe vyzkoušené hodnoty. Základními vlastnostmi by však mělo být dodržení následujících požadavků [16],[8] :

- c a m jsou navzájem nesoudělná čísla
- $a - 1$ je dělitelné všemi děliteli m, jež jsou prvočíslem
- $a - 1$ je dělitelné čtyřmi pokud m je dělitelné čtyřmi

Algoritmus pracující podle výše uvedeného vzorce, však generuje pseudonáhodná čísla v intervalu $\langle 0, m-1 \rangle$, proto je potřeba ještě pro získání rozložení posloupnosti $\mathrm{R}(0,1)$ dělit výsledek modulem m. Pak výsledek dostaneme po provedení následujícího vzorce:

$$R_i = \frac{x_i}{m}$$

Zvláštní případy (typy) LCG vznikají při vynechání jedné z konstant nebo úpravou vzorce. Zatímco v plném znění vzorce, jak je uveden výše, mluvíme o kongruentní smíšené metodě, můžeme se setkat i s následujícími obměnami:

1. Aditivní neboli Fibonacciův generátor

$$x_{i+1} = (x_i + x_{i-1}) \,(\mathrm{mod}\ m)$$

2. Multiplikační neboli Lehmerův generátor

$$x_{i+1} = (a\, x_i) \,(\mathrm{mod}\ m)$$

Kongruentní generátory jsou implementačně jednoduché a dosti rychlé. Nevýhodou je obtížná volba parametrů, kdy při špatném zvolení konstant dojde k podstatnému zhoršení vlastností generované posloupnosti. Pro využití v systémech s velkými nároky na generátor je dále nevýhodou závislost po sobě jdoucích hodnot. To se projeví především pokud potřebujeme využívat n-tice náhodných čísel. Problém nastává i při požadavku na generování celých čísel, protože kongruentní generátory mají málo náhodné nejméně významné bity generovaných čísel.

Příklad implementace je na straně 29 v kapitole 5.1.1.

2.2.6 Generátory využívající celulární automaty

Celulární automaty jsou již poměrně staré a známé systémy, kterými se zabývalo a zabývá v oblasti informační techniky celá řada odborníků a specialistů. Využití těchto systémů a jejich vlastností jako metod pro generování pseudonáhodných čísel, začalo být aktuální až v nedávné době. Celulární automaty jsou složené z pole buněk, kde buňka představuje základní element (atomický prvek). V čase pak tyto buňky určitým způsobem mění svůj stav a vyvíjí se. Chování automatu (obsažených buněk) se řídí podle pravidel, která na základě stavu buňky a okolí této buňky určují nový výsledný stav. Jde o systémy, řídící se podle svého aktuálního stavu, který odvodí ze stavu svého okolí a množinou definovaných pravidel. Svůj stav odvozují podle stavů buněk ve svém o okolí a je to jediná informace, která v daný moment určuje tento stav.

Pro pseudonáhodné generátory se využívá druh stochastických celulárních automatů. Mluvíme zde o vlastnosti chaotického chování celulárního automatu, patřící do tzv. 3. třídy celulárních automatů.

Celá teorie celulárních automatů je velmi rozsáhlá a dosti složitá, proto se nadále tímto druhem možného generování v práci nebudu zabývat, i když věřím že by šlo o zajímavé

odvětví řešení problému generování pseudonáhodných čísel.

Následující podkapitoly popisují složitější generátory s netriviálními generujícími algoritmy. Tyto složité, ale kvalitní generátory využívají různých přístupů pro svou práci a s mnohem sofistikovanějšími metodami generování.

2.2.7 Mersenne twister

Jeden z nejznámějších a také nejkvalitnějších generátorů je Mersenne twister. Jde o generátor navrhnutý pány Takuji Nishimura a Makoto Matsumoto v roce 1997 [13], [2].

V dnešní době je považován díky své velmi velké periodě, která se udává jako hodnota $2^{19937} - 1$, a kvalitě za nejspolehlivější a nejvýkonnější generátor v oblasti simulace. Jeho práce spočívá na lineární rekurenci na matici nad konečným binárním polem. V dnešní době se používají dva typy tohoto druhu generátoru, lišící se délkou periody a to podle periody určené délkou datového typu. Přesněji 32 nebo 64 bitové druhy, přičemž každý generuje jiné posloupnosti čísel.

Mersenne twister se nehodí pro kryptografii, protože po určitém daném počtu iterací lze zjistit další následující a lze tedy šifrování komunikace prolomit. Jako řešení se jeví použití na výstupní posloupnost generátoru hashovací funkce, která nám zajistí potřebnou bezpečnost, avšak tím se snižuje výkonnost a ztrácí tak své kvality.

Jak již bylo řečeno, tento generátor s hodně velkou periodou je velmi kvalitní a splňuje drtivou většinu statistických testů. Další výhodou je vícedimenzionální nezávislost, kdy při zkoumání i dlouhých n-tic nepozorujeme žádnou závislost. Co se týče rychlosti, jde o pomalejší generátor, nedosahující v rychlosti kvalit LCG. Nevýhodou může být poměrně dlouhý počáteční čas ustálení generátoru. Jde zde o problém s počátečními vstupními hodnotami. Výstupní hodnoty při startu generátoru nesplňují svou kvalitou statistické testy. Prototo se obvykle používají k inicializaci (k určení počátečních hodnot) jednodušší a rychlejší generátory jako např. lineární kongruentní generátor.

Ukázka implementace i s jejím popisem se nachází na straně 29 v kapitole 5.1.2.

2.2.8 Blum Blum Shub

Dalším z generátorů je Blum Blum Shub navržený roku 1986 pány Lenore Blum, Manuel Blum a Michael Shub. Ke svému výpočtu používá problému kvadratických zbytků. Zjednodušený vzorec popisující tento jev je:

$$x_{i+1} = (x_i^2) \,(\mathrm{mod}\ m)$$

, kde m je násobek dvou dostatečně velkých prvočísel. Tento generátor není příliš vhodný pro využití v simulačních aplikacích vzhledem ke své nevelké rychlosti práce. Nicméně je velmi vhodný pro kryptografii, kde díky jeho neobvykle silné bezpečnosti našel uplatnění [7], [1].

2.2.9 Další druhy generátorů

Za další druhy generátorů lze považovat kombinace předešlých generátorů, spojených paralelně či sériově do složitějších zapojení. Někdy se použije jednoho generátoru pro získání počátečních hodnot pro inicializaci jiného generátoru.

Také existují i různé variace a derivace předešlých druhů generátorů a další nové se testují a připravují pro použití. Jako příklad ještě uvedu generátor Yarrow – 160 a Linux – PRNG a smíšené generátory jako filtr von Neumanna, iterační filtr Y. Perese nebo techniku xorování (viz. [19]).

Nelze obsáhnout všechny druhy a typy generátorů pseudonáhodných čísel a ani nelze jeden a každý popsat tak podrobně, jak je to při všech informacích uvedených v dostupných publikacích možné. Vidíme, že je velké množství druhů, typů a variací generátorů a lze si tedy podle požadavků vybrat ten nejvhodnější s jeho výhodami i nevýhodami.

Kapitola 3

Transformace typů rozložení

Výstupem uvedených generátorů z předešlých kapitol je pseudonáhodná posloupnost číslic s rovnoměrným rozložením. Někdy však požadujeme generování jiného typu rozložení. V praxi tyto generátory vytváříme za pomoci generátorů pseudonáhodných čísel s rovnoměrným rozložením (někdy označované jako primární generátory) a následně tuto posloupnost transformujeme do podoby požadovaného rozložení. Ty můžou být jak spojité tak diskrétní.

Nejčastěji používaná rozložení:

- spojitá
  - rovnoměrné – uniformní
  - exponenciální
  - normální – Gaussovo
  - Pearsonovo
  - studentovo
  - Laplaceovo
  - Maxwellovo
  - logistické
  - Weibullovo
  - ...
- dikrétní
  - Poissonovo
  - binomické
  - geometrické
  - logaritmické
  - Pascalovo
  - ...

Druhů rozložení je opravdu velké množství a více se o nich můžeme dozvědět v různých publikacích nebo na Internetu.

3.1 Pravděpodobnostní rozložení a jejich druhy

Náhodná (tedy i pseudonáhodná) proměnná veličina je definována svým rozložením pravděpodobnosti. Rozložení pravděpodobnosti je funkce rozdělující pravděpodobnost událostem nebo tvrzením. Nejobvyklejší je popsat rozložení pomocí funkce hustoty rozložení pravděpodobnosti nebo distribuční funkce.

Parametry těchto funkcí jsou střední hodnota a rozptyl, obecně dány vzorci:

- pro diskrétní náhodnou veličinu X
  - střední hodnota
    $$E(X) = \sum_{x_i \in M} x_i f(x_i)$$
  - rozptyl
    $$D(X) = \sum_{x_i} [x_i - E(X)]^2 f(x_i)$$

  , kde $f(x)$ je frekvenční funkce a M je obor hodnot.

- pro spojitou náhodnou veličinu X
  - střední hodnota
    $$E(X) = \int_{-\infty}^{\infty} x f(x) \mathrm{d}x$$
  - rozptyl
    $$D(X) = \int_{-\infty}^{\infty} [x_i - E(X)]^2 f(x) \mathrm{d}x$$

  , kde $f(x)$ je funkce hustoty pravděpodobnosti. Náhodná veličina se spojitým rozložením je tedy definována intervalem hodnot, které nabývá, a příslušnou hustotou.

Distribuční funkce $F(x)$ popisuje rozložení pravděpodobnosti mezi náhodné jevy a má následující vlastnosti:

- distribuční funkce udává pravděpodobnost s jakou nastoupí náhodná veličina X, která nabývá hodnoty menší nebo rovnu hodnotě x
  $$F(x) = P(X \leq x)$$
- pro každé reálné x je distribuční funkce neklesající
- $F(x)$ nabývá hodnot z intervalu $\langle 0, 1 \rangle$ pro každé reálné x
- $\lim_{x \to -\infty} F(x) = 0 \quad \text{a} \quad \lim_{x \to \infty} F(x) = 1$

Funkce hustoty pravděpodobnosti $f(x)$ je derivace spojité distribuční funkce $F(x)$. Platí pro ni následující pravidla:

- $f(x) \geq 0$ pro každé reálné x
- $\int_{-\infty}^{\infty} f(x) \mathrm{d}x = 1$

V následujících kapitolách jsou naznačeny nejčastěji požadovaná pravděpodobnostní rozložení. Z důvodů velkého množství nelze v této práci všechny druhy rozložení uvést a následující jsou zde spíše ukázány jako příklad, než jako plnohodnotný text zabývající se podrobněji teorií pravděpodobnosti a jejího rozložení.

3.1.1 Rovnoměrné rozložení pravděpodobnosti

Rovnoměrné rozložení pravděpodobnosti je nejjednodušším typem spojitého rozložení. Takto popisovaný model konkrétní situace je dosti nepřesný, protože v reálném prostředí se vyskytuje poměrně málo případů s tímto rozložením.

Za rovnoměrné rozložení pravděpodobnosti veličiny X, označíme situaci, pokud nabývá X hodnot z intervalu $\langle a, b\rangle$ konečné délky a všechny hodnoty z tohoto intervalu jsou stejně pravděpodobné. Označujeme jej $\mathrm{R}(a, b)$. Používané je tzv. normované rovnoměrné rozložení $\mathrm{R}(0, 1)$.

Charakteristiky rozložení:

- Distribuční funkce:
$$F(x) = \begin{cases} 0 & \text{pro} \quad x < a \\ \frac{x-a}{b-a} & \text{pro} \quad a \leq x \leq b \\ 1 & \text{pro} \quad x > b \end{cases}$$
- Funkce hustoty:
$$f(x) = \begin{cases} 0 & \text{pro} \quad x < a \\ \frac{1}{b-a} & \text{pro} \quad a \leq x \leq b \\ 0 & \text{pro} \quad x > b \end{cases}$$
- Střední hodnota :
$$E(x) = \frac{a+b}{2}$$
- Rozptyl:
$$D(x) = \frac{(b-a)^2}{12}$$

3.1.2 Exponenciální rozložení pravděpodobnosti

Spojitá náhodná veličina X má toto rozložení tehdy, pokud hustota pravděpodobnosti má tvar exponenciální funkce. Exponenciální rozložení nám udává dobu mezi nastoupením dvou jevů.

Charakteristiky rozložení:

- Distribuční funkce:
$$F(x) = \begin{cases} 1 - \mathrm{e}^{-\frac{1}{A}(x-x_0)} & \text{pro} \quad x \geq x_0 \\ 0 & \text{pro} \quad x < x_0 \end{cases}$$
- Funkce hustoty:
$$f(x) = \frac{1}{A}\mathrm{e}^{-\frac{1}{A}(x-x_0)} \quad \text{pro } x \geq x_0$$
- Střední hodnota :
$$E(x) = x_0 + A$$
- Rozptyl:
$$D(x) = A^2$$

3.1.3 Normální (Gaussovo) rozložení pravděpodobnosti

Toto rozložení pravděpodobnosti aproximuje řadu jiných pravděpodobnostních rozložení a to jak spojitých tak i diskrétních. Značí se N(μ,σ^2), kde μ je střední hodnota tohoto rozložení a σ^2 je rozptyl a jednoznačně jej určují.

Normální rozložení, nazývané taky Gaussovo rozložení, se vztahuje i k odhadu chyb. Například určení chyby při měření, které je způsobeno vzájemně nezávislými chybami. Pokud máme dostatečný počet hodnot, jejich odchylka od ideálního výsledku často aproximuje s grafem normálního rozložení. Má tedy velký význam v teorii pravděpodobnosti a matematické statistiky. Normované normální rozložení značíme N(0, 1).

Charakteristiky rozložení:

- Distribuční funkce:
$$F(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} \mathrm{e}^{-\frac{(\xi-\mu)^2}{2\sigma^2}} \mathrm{d}\xi$$
- Funkce hustoty:
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}(x-x_0)}$$
- Střední hodnota :
$$E(x) = \mu$$
- Rozptyl:
$$D(x) = \sigma^2$$

Generování čísel s rovnoměrným rozložením jsme si popsali v předešlých kapitolách. Při modelování potřebujeme ale i jiná rozložení než rovnoměrná. Proto se budeme zabývat metodami transformace rovnoměrného rozložení posloupnosti čísel na rozložení jiné.

Při vytváření libovolného rozložení postupujeme takovým způsobem, kdy nejprve generujeme rovnoměrné rozložení a následně pomocí určitých metod převádíme (transformujeme) na rozložení jiné. Pro převod používáme různé metody transformace s tím, že každá metoda má svá omezení pro použití. Musíme tedy tuto metodu, s ohledem na tvar výstupního rozložení, vhodně zvolit.

Přehled používaných transformačních metod:

- inverzní
- vylučovací
- kompoziční
- speciální (aproximační, přímé pomocí vzorce)

3.2 Inverzní metoda transformace

Máme-li posloupnost čísel s normovaným rovnoměrným rozdělením R$(0, 1)$ a distribuční funkci $F(x)$ požadované náhodné veličiny, potom pokud má tato distribuční funkce inverzní funkci $F_{-1}(x)$ a je *stále rostoucí*, platí:

$$R = F(x) \Rightarrow X = F_{-1}(R)$$

Metoda je použitelná pro rozložení, pro která je možno snadno vypočítat inverzní funkci distribuční funkce. Typické je například exponenciální rozložení. Pokud distribuční funkce daného rozložení nemá inverzní funkci a nelze použít úspěšně jinou metodu transformace, lze použít aproximace této funkce.

3.3 Vylučovací metoda transformace

Pokud hustota pravděpodobnosti $f(x)$ je ohraničena v intervalu (x_1, x_2) tedy $x \in \langle x_1, x_2 \rangle$ a $f(x) \in \langle 0, M \rangle$, kde M je maximální hodnota $f(x)$, pak generujeme postupně body $x = \mathrm{R}(x_1, x_2)$ a $y = \mathrm{R}(0, M)$. Pokud je $y \leq f(x)$, označíme x jako hodnotu požadované náhodné veličiny X, jinak musíme generování opakovat. Méně formálně řečeno je princip založen na generování náhodných bodů s rovnoměrným rozložením v *ploše* obdélníku (x, y). Pokud se nachází tento bod v ploše pod funkcí hustoty $f(x)$ požadovaného rozložení, označíme x za nově vygenerované náhodné číslo. Jinak generujeme další bod a opakujeme postup.

Efektivita této metody je určena poměrem plochy obdélníku $(x_2 - x_1)M$ a plochy integrálu funkce $\int_{x_1}^{x_2} f(x)\mathrm{d}x$. Tedy:

$$\text{efektivita metody} = \frac{(x_2 - x_1)M}{\int_{x_1}^{x_2} f(x)\mathrm{d}x}$$

Metodou lze řešit i transformace na rozložení se složitou distribuční funkcí a její inverzní funkcí. Problém nastává v případě malého poměru plochy pod funkcí hustoty ku ploše obdélníka v němž generujeme náhodné body. V takovém případě prudce klesá efektivita této metody a musíme volit metodu jinou.

3.4 Kompoziční metoda transformace

Kompoziční metoda rozkládá složitou funkci hustoty pravděpodobnosti na více jednodušších funkcí ([17]). Pokud tedy $f(x)$ si označíme hustotu pravděpodobnosti jednoduše generovatelného rozdělení a p_i i-té náhodně generované číslo, pak hustota původního rozdělení je dána:

$$f(x) = \sum_{i=1}^{k} p_i f_i(x)$$

Horní hranice funkce k někde bývá nahrazena symbolem nekonečna. Nic to nemění na platnosti vzorce, protože stále platí pokud $i > k$ pak $p_i = 0$. Totéž můžeme říct i o rozkladu složité distribuční funkce na její jednodušší části, jestliže je to výhodnější než využívat funkce hustoty ([12]). Píšeme pak ekvivalentně:

$$F(x) = \sum_{i=1}^{k} p_i F_i(x)$$

Často se používá ve spojení s jednodušší inverzní nebo vylučovací metodou. Po rozložení složité funkce lze najít např. pro všechny části inverzní funkci k distribuční funkci, nebo lze jednotlivé části efektivně počítat vylučovací metodou.

3.5 Speciální metody transformací

Při nevhodnosti předešlých transformačních metod musíme sáhnout po metodách složitějších, které například vycházejí z vícenásobných transformací, kdy zadanou náhodnou veličinu získáme na základě rozložení, jež je také výsledkem jiné transformace. To je poměrně náročná a složitá operace, proto jsou tyto transformace také časově a realizačně náročné.

3.5.1 Metody využívající aproximace

Jedním typů speciálních transformací jsou metody aproximační, kdy některou z charakteristických funkcí rozložení aproximujeme vhodnější funkcí a aplikujeme jednu z předešlých metod. Aproximace nám zavádí odchylku a tudíž určitou chybu. Tyto metody jsou poměrně jednoduché a často vyhovují požadované přesnosti.

Příkladem může být *aproximace omezených rozložení po částech rovnoměrným rozložením* (převzato z [5]): Princip spočívá v nahrazení určité omezené části rovnoměrným rozložením. Převedeme složité rozložení na posloupnost několika po sobě jdoucích rovnoměrných rozložení. Celkově si po tomto „rozřezání“ a aproximací částí lze představit po částech rovnoměrné rozložení – histogram. Toto rozdělení lze generovat pomocí modifikované metody inverzní transformace tak, že je vygenerována hodnota z diskrétního rozložení odpovídajícího relativním četnostem hodnot jednotlivých částí. Pokud dále budeme považovat distribuční funkci schodovitého grafu za relaci, lze k ní vytvořit relaci inverzní a vhodnou úpravou získat i inverzní funkci. Poté vygenerovaná hodnota určuje svoje zařazení do dané části. Pak lze pro tuto část vygenerovat hodnotu z intervalu odpovídajícímu jejím hranicím. Takto dostaneme požadované výsledné rozložení.

3.5.2 Metody používající konkrétního vzorce

Jiným druhem speciálních metod jsou takové, kdy pomocí matematických vzorců lze přímo vypočítat a transformovat rovnoměrné rozložení na rozložení požadované. Nejčastěji se s nimi setkáváme u transformací na častá teoretická rozložení, jako jsou normální, rovnoměrné, exponenciální atd. Někdy jde o zaužívané vzorce odvozené pomocí metody inverzní transformace. Jako příklad můžou být uvedeny vzorce pro transformaci na normální rozložení[12]:

$$X_1 = \sqrt{-2\ln(R_1)}\sin(2\pi R_2)$$

$$X_2 = \sqrt{-2\ln(R_1)}\cos(2\pi R_2)$$

Kapitola 4

Testy a testování generátorů

Algoritmické generátory můžou vytvářet pouze *pseudonáhodnou* posloupnost čísel. Jak však ověřit „kvalitu náhodnosti“ nebo chcete-li „míru náhodnosti“? Už z rozdílného přístupu v generování a z různých možných hodnot konstant vyplývá, že výsledné posloupnosti nebudou stejné a ani stejně kvalitní. Jak tedy určit použitelnost toho či onoho generátoru pro určitý problém?

Jedna z možností je prověřit výstup generátoru sadou statistických testů a na základě jejich výsledků rozhodnout, zda generátor splňuje řadu předpokladů. Pak teprve můžeme rozhodnout o vhodnosti či nevhodnosti daného generátoru pro řešený problém.

4.1 Požadované a testované vlastnosti generátorů

Nejčastější druhy požadavků jsou [6]:

1. **výpočetní** – požadavky na kvalitu a způsob výpočtu. I když jsou dnešní počítače a mikrokontroléry na velmi vysoké úrovni za poměrně nízké náklady, musíme i přesto šetřit výpočetními prostředky. Proto musíme požadovat i výpočetní kvalitu používané aplikace (generátoru).

    - **efektivita** – zcela jistě chceme, aby byl generátor rychlý a nezabíral si pro sebe příliš mnoho zdrojů. Aby pro svůj chod spotřebovával co nejméně paměti a požadoval pouze minimum procesorového času a to především u rozsáhlých simulací. Ovlivnit efektivitu lze vhodnou volbou typu generátoru.

      Pokud se bavíme už o určitém druhu, pak při jeho implementaci je nutné volit používané operace a vnitřní strukturu aplikace s ohledem právě na výše popsané omezující faktory.
    - **přenositelnost** – jinými slovy nezávislost na druhu platformy na níž generátor běží. Algoritmus generátoru by měl být snadno instalovatelný a fungovat stejně na různých platformách, ať už hardwarových či softwarových. Často se využívá implicitního ořezávání datových typů, což vede k tomu, že na platformě, kde je tento typ jiného rozsahu, bude funkce generátoru jiná. Na druhou stranu tímto způsobem můžeme zvýšit efektivitu kódu. Proto si musíme dobře promyslet otázky jak, na jaké platformě bude aplikace pracovat a kým bude používána.

      Musíme volit vždy určitý kompromis, protože příliš robustní program použitelný na libovolné platformě, může být například pomalý nebo zbytečně rozsáhlý a paměťové náročný. Naopak velmi rychlý algoritmus, který není dostatečně

ošetřen a pracuje pouze na jediném druhu platformy, nebude taktéž příjmaný s kladnými ohlasy (vyjímkou snad může být některá specializovaná a jednoúčelná aplikace).

- **opakovatelnost** – především pro účely ladění je důležitá možnost opakování generování shodné pseudonáhodné posloupnosti. Dosáhneme toho možností nastavování počáteční hodnoty posloupnosti tzv. semínka a to tedy alespoň v části ladění simulace. Tato možnost usnadňuje zajisté čas práce na takovéto aplikaci a může být užitečná při samotném využití, kdy například v simulaci ověřujeme správnost určitých dat a znalostí.

2. **strukturní** – požadavky na strukturu výsledné posloupnosti.

   - **délka periody** – vždy se snažíme o co nejdelší neopakující se posloupnost pseudonáhodných čísel. Proto je dobré volit délku periody generátoru mnohonásobně větší, než je perioda posloupnosti. Požadavek na délku periody je v mnoha případech zásadní a měli bychom proto věnovat velkou pozornost volbě správných konstant používaných v aplikaci, protože ty nejvíce ovlivňují omezení zvolené a požadované periody.
   - **rovnoměrnost rozložení** – důležitou podmínkou je také rovnoměrné rozložení čísel v generovaném intervalu. Transformace pravděpodobnostního rozložení počítají na vstupu s rovnoměrným rozložením a proto se vždy snažíme o co možná nejkvalitnější rozložení. Nekvalitní vstup transformace může významně ovlivnit i její výstup.

3. **požadavek složitosti** – požadavky na teoretickou složitost výsledné posloupnosti

   - **neodhadnutelnost následujícího členu (nepředvídatelnost)** – myslí se tím nevypočitatelnost následujícího členu posloupnosti na základě doposud vygenerované posloupnosti. Aspoň tedy ne v rozumném čase, kdy by mohlo zjištění dalšího členu poškodit uživatele. Je důležitá především pro kryptografii, kdy se snažíme ochránit data, které by mohla získat nepovolaná osoba nebo dokonce zkušený útočník.

4. **statistické** – požadavky kvality výsledné posloupnosti

   - **úspěšnost v empirických testech** – generátor podrobujeme různým empirickým testům a ověřujeme platnost tzv. nulové hypotézy H_0, která značí testovanou hypotézu. Ku příkladu hypotézu, že vygenerované hodnoty jsou nezávislá náhodná čísla z intervalu $\langle 0, 1\rangle$ rovnoměrného rozložení $\mathrm{R}(0, 1)$, a jiné vlastnosti. Ověřujeme si Pomocí těchto testů si ověřujeme požadavky na kvalitu vlastností uvedených výše a na základě vyhodnocení testů, můžeme rozhodnout o tom, zda je pro nás daný generátor přípustný a vhodný.

4.2 Rozdělení druhů testů

Testy lze klasifikovat podle různých kritérií a jejich vlastností. Několik příkladů rozdělení je pro názornost uvedeno v následujícím textu.

Pomocí testů a testování posloupnosti pseudonáhodných čísel se snažíme dokázat či vyvrátit určité tvrzení. Podle něj přijmeme respektive odmítneme danou hypotézu. Jednou z možností jak dělit testy je podle zaměření, kterou vlastnost testované posloupnosti ověřujeme. Téměř vždy jde o kvalitu rozložení a náhodnost rozložení prvků posloupnosti a to jak primární rozložení posloupnosti (důraz na rovnoměrnost) tak i na transformované (především odpovídající tvar rozložení).

Testy se také budou lišit od sebe podle toho k jakému účelu je testovaná posloupnost určena. Zcela jistě jsou požadavky na posloupnost pro kryptografii jiné než pro obecně simulační využití. Tomu se musí přizpůsobit i druhy testů.

Podle pánů Lawa a Keltona [12] dělíme testy ještě z pohledu druhu testování na empirické a teoretické.

Empirické testy jsou obvykle druhem statistického testu a jsou založeny na aktuálně vygenerované sekvenci čísel.

Teoretické testy nejsou myšleny ve smyslu statistického testování, ale pracují s číselnými parametry generátorů k určení vlastností globálně bez výstupní posloupnosti generátoru.

V neposlední řadě se testy můžou třídit i podle baterie testů v nichž se nachází. Existují testy obecné, využívané většinou známých sad testů, ale jsou i testy přímo specifické objevující se pouze u jedné ze sady testů. I společné testy jsou v různých distribucích jiné. Z nejznámějších a nejpoužívanějších testovacích baterií můžeme jmenovat DIEHARD, FIPS, NIST, CRYPT – XS.

4.3 Vyhodnocování statistických testů

Statistickými testy si dále můžeme ověřit vlastnosti souboru čísel pseudonáhodné posloupnosti, zda opravdu odpovídá požadovanému rozložení nebo dokonce nalézt jeho rozložení či jemu podobné. Někdy se také využívají k určení parametrů rozložení, aby mu vzorek odpovídal.

4.3.1 Testování statistických hypotéz

Při testování náhodnosti jdeme cestou tzv. testovaní statistických hypotéz a postupujeme většinou podle následujícího scénáře [11]:

1. **Zavedeme nulovou hypotézu H_0 a k ní alternativní hypotézu H_a:**
   Nejprve si zavedeme tzv. nulovou hypotézu H_0. Ta značí pravdivost testované vlastnosti. U testování v oblasti pseodonáhodných posloupností to bude zcela jistě hypotéza, že daná posloupnost je *náhodná* nebo *rovnoměrná*. Negací hypotézy H_0 získáme opačnou hypotézu, což je alternativní hypotéza H_a a označuje např. *nenáhodnost* posloupnosti.

   Na začátku přijmeme za pravdivou hypotézu H_0 a v průběhu testování se snažíme tuto hypotézu vyvrátit a přijmout tak H_a. Pokud se nám to nepodaří, musíme přijmout H_0 jako pravdivou.

2. **Zvolíme hladinu významnosti α:**
   Hladina významnosti α je předem zvolená pravděpodobnost s dostatečně malou hodnotou pro zamítnutí hypotézy o rozložení daného náhodného čísla. Značí s jakou

pravděpodobností bude označena dobrá náhodná posloupnost jako nenáhodná. Podrobněji se touto hodnotou zabývá text níže.

3. **Určíme kritickou hodnotu:**
   Jsou formulovány dvě disjunktní hypotézy H_0 a H_a. Stejně je rozdělen na dva taktéž disjunktní obory hodnot celý obor možných hodnot testovacího kritéria. Obor na nějž odkazuje H_0 označuje obor přijetí a H_a značí druhý jako obor odmítnutí (kritický obor). Hraniční hodnoty mezi těmito dvěma obory označujeme jako kritické hodnoty. Kritická hodnota se počítá pomocí matematických metod z teoretického rozložení testovaných dat.

4. **Výpočet statistické hodnoty testu:**
   Během provádění statistického testu je počítána jeho hodnota z dat testované posloupnosti. Výsledná hodnota se poté porovná s kritickou hodnotou. Výpočtem této hodnoty se prakticky zabývá od podkapitoly 4.4 celý zbytek této kapitoly.

5. **Rozhodnutí o přijetí hypotézy:**
   Po porovnání vypočtené hodnoty a hodnoty kritické, určíme do kterého ze dvou oborů výsledná hodnota padla. Na základě výsledného oboru konstatujeme zda danou hypotézu, kterou každý obor značí, přijmeme či odmítneme. Nachází-li se výsledek v oboru přijetí a výsledná statistická hodnota testu nepřekročila kritickou hodnotu, přijmeme nulovou hypotézu a prohlásíme, že posloupnost můžeme na základě daného testu označit za náhodnou. V opačném případě odmítneme H_0, přijmeme H_a a považujeme danou posloupnost za nenáhodnou.

Při rozhodování o přijetí či nepřijetí dané hypotézy se můžeme dopustit chyby. Tento stav nám ilustruje tabulka 4.1. Vidíme zde dva typy chyb:

1. U první se dopouštíme chyby tím, že nulovou hypotézu zamítneme u skutečně náhodných dat a označíme je za nenáhodná. Tato chyba se označuje za chybu I. typu (či pokud chcete I. druhu) nebo také jako hladina významnosti testu a značí se α. Říká nám s jakou pravděpodobností budou v konečné klasifikaci označena skutečně náhodná data za nenáhodná.

2. Chybu II. typu zapisujeme symbolem β. Udává s jakou pravděpodobností nastane označení špatných nenáhodných dat jako dat náhodných a tedy vyhovujících. β nabývá hodnoty z širokého intervalu a je mnohem těžší ji určit oproti hodnotě α. Někdy se udává místo hodnoty β její doplněk tzv. *sílu testu* $(1 - \beta)$.

Oba typy chyb jsou na sobě závislé a to nepřímou úměrou. Snažíme-li se jednu z nich minimalizovat, druhá nám roste. Ve vztahu mezi těmito hodnotami ještě vystupuje proměnná n, která nám udává velikost testované posloupnosti dat.

Snahou je vždy co nejvíce omezit chybu II. typu, protože výhodnější je odmítnout dobrý generátor, než přijmout špatný. To by totiž mohlo způsobit velké komplikace a hrozivé následky jeho práce při dalším použití. V praxi se často volí v testech velikost testovaného vzorku dat a hladina významnosti α, z nichž se určí pomocí kritického bodu hodnota β.

4.3.2 $P - hodnota$

$P-hodnota$ (v angličtině označovaná $P-value$ nebo $P-level$) nám udává pravděpodobnost s jakou dokonalý generátor vygeneruje posloupnost čísel méně náhodnou, než je posloup-

Skutečná situace	Závěr	
	Příjmutí H_0	Příjmutí H_a (Odmítnutí H_0)
Posloupnost **je** náhodná	Není chyba	Chyba I. typu
Posloupnost **není** náhodná	Chyba II. typu	Není chyba

Tabulka 4.1: Typy chyb

nost čísel testovaných dat. Zavádíme $P - hodnotu$ hypotézy, což je přesně nejmenší pozorovaná hladina významnosti, na které může být nulová hypotéza zamítnuta [11]. Čím více se hodnota blíží k 1, tím je testovaná posloupnost kvalitnější, což znamená v tomto případě náhodnější. Pokud se hodnota blíží k 0, jde o posloupnost nekvalitní. V krajních případech, kdy $P - hodnota = 1$ a $P - hodnota = 0$, jde o perfektní náhodnou posloupnost a o zcela nenáhodnou sekvenci čísel.

Jak ale rozhodneme zda je testovaná sekvence náhodná a přijmeme tak nulovou hypotézu? Odpověď je jednoduchá. Porovnáme $P - hodnotu$ s hladinou významnosti α. Platí:

$$\begin{aligned} P - hodnota \leq \alpha &\Rightarrow H_a \\ P - hodnota > \alpha &\Rightarrow H_0 \end{aligned}$$

Při použití $P - hodnoty$ v testování postupujeme obdobně jako v klasickém postupu testování. Rozdíl se projeví až při rozhodnování o přijetí hypotézy. Na základě vypočtené statistické hodnoty testu (viz kapitola 4.3.1 bod 4) vypočteme $P - hodnotu$. V následujícím kroku rozhodneme o přijetí nulové hypotézy: je-li $P - hodnota$ menší nebo rovna hladině významnosti přijímáme alternativní hypotézu. V opačném případě přijímáme nulovou hypotézu a alternativní zamítáme.

4.3.3 Chí-kvadrát χ^2

Často se v testech setkáváme s univerzálním statistickým kritériem χ^2, kterého se využívá při vyhodnocování většiny statistických testů. Nazýváme ho Pearsonovým chí-kvadrátem rozložení nebo také χ^2 testem.

Pokud si všechna náhodná čísla rozdělíme do k kategorií můžeme výslednou hodnotu chí-kvadrátu V vypočíst ze vztahu

$$V = \sum_{i=1}^{k} \frac{(y_i - np_i)^2}{np_i}$$

, kde N je počet navzájem nezávislých pokusů resp. počet čísel testované vstupní sekvence (viz [15], [17]). Symbol p_i označuje pravděpodobnost, že výsledek pokusu padne do kategorie i. y_i je počet pokusů, které opravdu do kategorie i padly. Někdy se také používá místo symbolu y_i symbol O_i jako pozorovaná empirická četnost v intervalu i a se symbolem E_i, který značí očekávanou (předpokládanou) četnost v intervalu i a je rovno np_i. Pro určení zda je hodnota chí-kvadrátu vyhovující zavádíme stupeň volnosti v a je roven:

$$v = k - 1$$

Známe-li tedy výsledek chí-kvadrátu a známe stupeň volnosti, můžeme podle statistických tabulek určit hladinu významnosti, která těmto hodnotám odpovídá [17].

4.4 Příklady statistických testů

V následujících oddílech jsou uvedeny příklady a popisy statistických testů. Jsou rozděleny podle testované vlastnosti posloupnosti na tři oddíly a to na testy rovnoměrnosti, testy náhodnosti a testy transformovaných rozložení. Testy někdy testují i více vlastností zároveň, v tom případě jsou uvedeny pouze jednou a to tam, kde je to vhodnější. Dělení podkapitol bylo převzato z [17].

Některé testy mají dvojí podobu. Jedna představuje test určený pro kryptografii a pracuje především s bitovou posloupností nul a jedniček. Druhou formou jsou testy pracující s normovanou posloupností (interval $\langle 0, 1\rangle$), tvořenou obecným desetinným číslem či celým číslem pokud pracujeme v obecném intervalu. Protože se v této práci zaobíráme především generátory pro simulaci a jejich výstupem, zajímá nás hlavně druhá forma testů. Přesto se někde pro zajímavost zmíníme i o jejich tvaru pro testování kryptografických generátorů.

Pozn.: Při práci jsem hojně využil hlavně materiálů z knihy [12], kde je podrobně popsáno mnoho testů. Protože jsem z této publikace čerpal opravdu často, nebudu ji zdůrazňovat pokaždé, ale pouze ve významných částech.

4.5 Testy rovnoměrnosti rozložení

4.5.1 Test rovnoměrnosti

Testujeme zda má posloupnost čísel na celém intervalu rovnoměrné rozložení. Rozdělíme interval na k podintervalů. Padnutí určité hodnoty do i-tého intervalu má pravděpodobnost $\frac{1}{k}$. Pomocí χ^2 potom vyhodnotíme teoretický počet generovaných veličin, které padnou do jednotlivých intervalů a jejich skutečný počet. Mluvíme tedy o aplikování χ^2 tak, jak je popsán v předešlé kapitole 4.3.3.

Pokud si do vzorce pro výpočet hodnoty χ^2 zavedeme ještě proměnou m_i značící počet hodnot v i-tém intervalu, můžeme úpravou vzorce získat zjednodušený následující tvar:

$$V = \frac{k}{n}\sum_{i=1}^{k}(m_i - \frac{n}{k})^2$$

Hodnota n zde označuje celkový počet čísel ve vstupních datech.

V kryptografii je tento test zaměřen na kontrolu jedniček a nul v binárním vyjádření vstupní posloupnosti. Spočítá poměr počtu jedniček a nul. Tento poměr by se měl blížit hodnotě 0,5. Větší odchylka od středu značí převahu jedné z hodnot.

4.5.2 Sériový test

Tento test zobecňuje test rovnoměrnosti pro více dimenzí tzn. pro vyšší počet n-tic (nejčastěji pro dvojice či trojice) tedy ve vícedimenzionálním prostoru. Takový prostor označujeme jako d-dimenzionální.

Výpočet vychází z klasického testu rovnoměrnosti (viz kapitola výše 4.5.1) s úpravami vzorce pro vyšší počet dimenzí d.

$$V(d) = \frac{k^d}{n}\sum_{i_1=1}^{k}\sum_{i_2=1}^{k}\cdots\sum_{i_d=1}^{k}(m_{i_1 i_2 \cdots i_d} - \frac{n}{k^d})^2$$

Symboly mají stejný význam jako v předešlých vzorcích χ^2. Především v již zmíněné předchozí kapitole.

4.5.3 Kolmogorov – Smirnof test

Řeší hlavní nevýhodu a problém χ^2 testu. U klasického testu rovnoměrnosti je nejtěžší specifikovat jednotlivé intervaly, tzn. určit počet intervalů a jejich velikost. Navíc χ^2 test byl navržen pro diskrétní rozložení a proto při použití na spojité rozložení pracujeme s jeho aproximací [7].

Princip Kolmogorov – Smirnof testu spočívá v porovnávání empirické distribuční funkce $F_n(x)$ a distribuční funkce předpokládaného rozložení $F'(x)$. Vstupní data se nijak neseskupují a není nutné vymezovat a určovat intervaly. Další výhodou je jeho správnost práce pro jakoukoli velikost vstupní sekvence.

Nevýhodami jsou jeho možnosti použití. Nelze jej použít všude tam, kde lze testovat pomocí χ^2 testu. Dalšími podmínkami pro správnou práci je známost všech parametrů předpokládaného rozložení a rozložení je *spojité*.

Nejprve si pro další práci definujeme znovu empirickou distribuční funkci a to následovně:

$$F_n(x) = \frac{\text{počet náhodných čísel} \leq x}{n}$$

n je velikost vstupního vzorku (celkový počet hodnot).

Při výpočtu hodnoty Kolmogorov – Smirnof testu D_n nás zajímá největší rozdíl mezi hodnotou empirické distribuční funkce náhodné proměnné $F_n(x)$ a předpokládané distribuční funkce $F'(x)$ pro všechna x. Vzorec je pak:

$$D_n = \max_x \{|F_n(x) - F'(x)|\}$$

Statistická hodnota D_n může být vypočítána pomocí:

$$D_n^+ = \max_{1 \leq i \leq n} \left\{ \frac{i}{n} - F'(X_{(i)}) \right\}$$

a

$$D_n^- = \max_{1 \leq i \leq n} \left\{ F'(X_{(i)}) - \frac{i-1}{n} \right\}$$

Výsledek získáme výběrem:

$$D_n = \max \{D_n^+, D_n^-\}$$

Nyní stačí již rozhodnout o přijetí resp. odmítnutí nulové hypotézy H_0. Obecně platí, že čím je D_n větší, tím je vstup míň kvalitní. Při konečném vyhodnocování může nastat více případů. Nejjednodušším případem je znalost všech možných parametrů. H_0 bude zamítnuta pokud bude platit:

$$\left(\sqrt{n} + 0,12 + \frac{0,11}{\sqrt{n}} \right) D_n > c_{1-\alpha}$$

, kde $c_{1-\alpha}$ je kritická hodnota při daném hladině významnosti a bere se z tabulky (viz [12], kde je i více podrobností o testu).

4.5.4 Rozložení maxima z n členů

Bereme postupně n-tice po sobě jdoucích náhodných čísel vygenerované posloupnosti $x_1, x_2, \ldots, x_n$. Vypočteme hodnoty $U = [\max(x_1, x_2, \ldots, x_n)]^n$ a takto vzniklou posloupnost podrobíme testu rovnoměrnosti viz [17].

4.6 Testy náhodnosti rozložení

Protože generátory pseudonáhodných čísel pracují podle určitého algoritmu, vždy najdeme určité souvislosti a zákonitosti výstupní posloupnosti. Proto zjišťujeme míru náhodnosti, kterou pak číselně ohodnotíme.

4.6.1 Test na intervalu

Kvalitu náhodnosti rozložení zde testujeme prověřením délky posloupnosti pseudonáhodných čísel mezi dvěma hodnotami, které padnou do stejného, předem zvoleného intervalu.

Označíme si počet hodnot v nejdelší posloupnosti nebo zvolíme maximální takovou hodnotu, po kterou nás velikost délky zajímá, symbolem h. Budeme počítat četnosti pro všechny posloupnosti délky t. t nabývá postupně hodnot $0, 1, 2 \ldots h$ [18]. Vyčíslíme počty posloupností stejných délek a zhodnotíme kritériem χ^2, kde pravděpodobnost označující posloupnost délky t je:

$$p_t = (b - a)(1 - (b - a))^t$$

S tím, že a označuje hodnotu dolní mezi intervalu a b horní mezi.

Test lze využít pouze pro zkoumání posloupnosti s normovaným rovnoměrným rozložením $\mathrm{R}(0, 1)$ nebo její podsekvence.

4.6.2 Poker test

Vygenerujeme k skupin o m hodnotách (při klasickém poker testu je $m = 5$). r označuje počet různých číslic ve skupině. Pomocí χ^2 srovnáváme očekávané a pozorované četnosti s pravděpodobností p_r, kdy:

$$p_r = \frac{d(d-1)\cdots(d-r+1)}{d^m} \left\{ {m \atop r} \right\}$$

d určuje maximální možnou hodnotu generované veličiny zvětšenou o 1, $\left\{ {m \atop r} \right\}$ je Stirlingovo číslo (viz kapitola 4.6.5). Popis tohoto testu lze najít v [17].

4.6.3 Test mezer

Pokud vezmeme tři po sobě následující čísla a, b, c z testované posloupnosti, jejich porovnáním dostaneme následující zápisy:

$$a < b < c, \quad a > b > c, \quad a > c > b, \quad a < c < b, \quad c > a > b, \quad c < a < b$$

Rovnost zde nebereme v úvahu. Pak počet výskytů těchto možností ve zkoumané posloupnosti můžeme statisticky zhodnotit χ^2. Protože je 6 možných situací je pravděpodobnost $\frac{1}{6}$ [17].

4.6.4 Test autokorelace

Tento test určuje stupeň, hladinu nezávislosti vzorků. Pokud se testová statistika ρ_j rovná nule je vzorek nezávislý a totožný s rozložením náhodné proměnné. Definovaný je takto:

$$\rho_j = \frac{C_j}{C_0}$$

, kde

$$C_j = \mathrm{cov}(X_i, X_{i+j}) = E(X_i X_{i+j}) - E(X_i)E(X_{i+j})$$

Funkce cov značí kovarianci mezi vstupy v sekvenci rozdělené hodnotou j, tzn. míru vzájemné vazby mezi dvěma náhodnými veličinami X_i a X_{i+j}. Nejzajímavější variantou je, když vstupní proměnné X_i mají normované rovnoměrné rozložení $\mathrm{R}(0,1)$ a píšeme $X_i = R_i$. Dostáváme tak konkrétní hodnoty $E(R_i) = E(R_{i+j}) = \frac{1}{2}$, $C_0 = \frac{1}{12}$ z toho získáme $C_j = E(R_i R_{i+j}) - \frac{1}{4}$. Doplněním do vzorce pro ρ_j získáme:

$$\rho_j = 12E(R_i R_{i+j}) - 3$$

V praxi nepočítáme však ρ_j z $E(R_i R_{i+j})$, ale přímo z R_1, R_{1+j}, R_{1+2j}, ...
Pak můžeme psát přímo vzorec:

$$\rho'_j = \frac{12}{h+1} \sum_{k=0}^{h} R_{1+kj} R_{1+(k+1)j} - 3$$

,kde $h = [\frac{(n-1)}{j}] - 1$. Přepokládejme nyní, že hodnoty posloupnosti s rovnoměrným rozložením R_i jsou nezávislé. Můžeme pak psát:

$$D(\rho'_j) = \frac{13h + 7}{(h+1)^2}$$

$D(\rho'_j)$ zde značí varianci neboli rozptyl.

Jak již bylo uvedeno, značí nám nulová hypotéza nezávislost vzorku. Pokud platí $\rho = 0$ je vzorek nezávislý a příjímáme hypotézu H_0. Hodnota statistického testu je dána vzorcem:

$$A_j = \frac{\rho'_j}{\sqrt{D(\rho'_j)}}$$

a měla by se blížit normovanému normálnímu rozložení. Výsledek pak zhodnotíma na hladině významnosti α a to tak, že pokud je absolutní hodnota $|A_j|$ větší než hodnota v kritickém bodě normovaného normálního rozložení $\mathrm{N}(0,1)$ tedy v bodě $1 - \frac{\alpha}{2}$. Tyto hodnoty můžeme určit například z tabulky nacházející se v knize [12].

4.6.5 Test sběratele kupónů

Prozkoumává posloupnosti náhodných čísel a zaznamenává délky sekvencí, v nichž se objeví všechny možné členy posloupnosti z intervalu $\langle 0, d-1 \rangle$ minimálně jednou. Hodnotu d nastavujeme z množiny {0,1,2,3} vždy však maximálně 3, protože dále ve výpočtu jsou použita Stirlingova čísla druhého druhu, která nabývají velmi rychle vysokých hodnot [7], [18]. Pro ohodnocení testu se používá opět χ^2 testu, pro nějž počítáme pravděpodobnost, že posloupnost s počtem hodnot m neobsahuje všechna možná čísla [17].

$$p_r = \frac{d!}{d^m} \left\{ {m \atop d} \right\}$$

$\left\{ {m \atop d} \right\}$ je Stilingovo číslo druhého druhu. Určuje se například z tabulky. Více o Stirlingových číslech je napsáno v [21] i s dalšími odkazy.

4.6.6 Run test

Test je určen pouze pro určení náhodnosti posloupnosti. Kontroluje v testované posloupnosti nepřerušenou posloupnost stále rostoucí, respektive stále klesající sekvenci hodnot. Podle toho dělíme tento test na „run-up“ kontrolující rostoucí sekvenci a na test kontrolující klesající sekvenci „run-down“.

Nejprve je při výpočtu nutné spočítat počet po sobě jdoucích čísel, splňujících výše popsaná pravidla. Počet posloupností stejné délky značíme jako run_i, kde i značí právě délku posloupnosti a platí $1 \leq i \leq 6$. Máme tak pouze run_1, run_2, ...,run_6. Posloupnosti delší než 6 se počítejí k run_6. Tuto hodnotu určujeme postupně průchodem vstupními hodnotami [3].

Například máme-li posloupnost čísel $X = (2, 4, 7, 3, 1, 9, 6, 8)$. Výsledkem průchodu může být pole $RUN = (3, 1, 2, 2)$, kde hodnoty v poli označují po sobě jdoucí čísla v stále rostoucí posloupnosti a získáme tak hodnoty $run_1 = 1$, $run_2 = 2$ a $run_3 = 1$. Pro $i = 4, 5, 6$ platí $run_i = 0$.

Výsledná hodnota testu se spočítá následovně:

$$V = \frac{1}{n} \sum_{i=1}^{6} \sum_{j=1}^{6} a_{ij}(run_i - nb_i)(run_j - nb_j)$$

Konstanta a_ij je hodnota vybrána z matice 6×6. Přesné hodnoty najdeme v knize pana Knutha [10]. Konstanta b_i je hodnota z vektoru uvedeném tamtéž, n je počet vzorků. Pro doporučovaný počet vzorků $n > 4000$ se bude V blížit k χ^2 rozložení se stupněm volnosti $v = 6$. Pak můžeme potvrdit nulovou hypotézu [12].

V kryptografických testech se ve vstupních datech hledá nepřerušená sekvence jedné z binárních hodnot (tedy 0 nebo 1), která je z obou stran ohraničena bitem s opačnou hodnotou. i označuje počet těchto uzavřených bitů stejné hodnoty. Určujeme zda je počet těchto sekvencí skutečně náhodný [11].

4.7 Testy transformovaných rozložení

I po provedení testů rovnoměrného rozložení, musíme po transformaci posloupnosti ještě otestovat, zda výsledné rozložení opravdu odpovídá požadovanému a zda se neobjevila dosud neodhalená chyba.

4.7.1 Test dobré shody χ^2

Pro testování transformovaných rozložení se používá test dobré shody, který testuje správnost parametrů rozložení i jeho tvar. Postup provádění testu je následující:

1. Vygenerování kontrolovaného souboru hodnot náhodné veličiny s požadovaným rozložením.
2. Získání srovnávacího souboru se správnými parametry i tvarem.
3. Srovnání obou souborů výpočtem hodnoty χ^2.

4. Zhodnocení výsledků testů na základě určité hladiny významnosti.

Existuje ještě mnoho dalších testů s různými principy práce. Jsou popsané více či méně v mnoha publikacích. Nejčastěji se s novými testy setkáváme v implementacích testových baterií, kde jsou popsány v manuálech těchto aplikací. Tato velká škála testů je velmi široká a nelze se všemi testy zabývat a všechny je podrobně popsat. Testy zde uvedené patří mezi ty nejrozšířenější a nejpoužívanější.

Kapitola 5

Praktická část – rozbor implementace

V této předposlední kapitole bude popsána praktická část bakalářské práce. Skládá se z návrhu a popisů implementací vybraných generátorů. Transformací rovnoměrného rozložení na jiný druh a testování výsledné posloupnosti. V závěru této kapitoly otestujeme vygenerované posloupnosti vytvořenými testy a podle výsledků zhodnotíme implementované generátory.

5.1 Návrh generátorů

Pro implementaci jsem si vybral hlavní druhy generátorů používané v modelování a simulacích. Jsou to lineární kongruentní generátor a Mersenne Twister generátor. Oba se vyznačují trochu jinými vlastnostmi a nacházejí uplatnění v různých aplikací.

Protože jde většinou o aplikace, používající jednoduché funkce, zvolil jsem pro implementaci jednoduchý, ale výkonný programovací jazyk C/C++ integrovaný v prostředí Borland C++ Builderu. Algoritmy generátorů mají sice jednoduchou strukturu s malým počtem funkcí, ale s nárokem na rychlost velkého počtu prováděných operací (jako jsou bitové násobení, sčítání, posun nebo operace dělení a modulo), čemuž jazyk C/C++ plně vyhovuje.

Z důvodů velkého rozšíření 32 – bitových počítačových platforem jsem se rozhodl z hlediska rychlosti, použít v aplikacích 4B bezznaménkovému celočíselnému datovému typu (v jazyku C/C++ je odpovídající datový typ `unsigned long`). To umožňuje používat implicitní operaci modulo a šetřit čas. Vnáší to však omezení na použitou platformu.

V další části textu naznačím vnitřní strukturu a implementaci algoritmů představující vybrané generátory. Nebudu je zde rozebírat teoreticky (to je popsáno v kapitole 2.2), ale ukážu a popíšu zde důležité části kódu.

5.1.1 Lineární kongruentní generátor – implementace

LCG generátor má velmi jednoduchou vnitřní strukturu a jádro se skládá pouze z jedné generující funkce a definování konstant. Právě hodnoty konstant se musí volit velmi pečlivě. Většinou se využívají ověřené a už odzkoušené konstanty. Nejčastěji používané hodnoty konstant ukazuje tabulka 5.1.

Právě tyto konstanty je možné zvolit i ve vytvořené aplikaci.

Č.	Multiplikační a	Aditivní b	Modul m
1	69 069	1	2^{32}
2	1 664 525	1 013 904 223	2^{32}
3	907 633 385	129	2^{32}
4	715 136 305	2 147 001 325	2^{32}
5	1 103 515 245	12 345	2^{32}

Tabulka 5.1: Tabulka používaných hodnot konstant LCG generátoru

Následující kód ukazuje stručně hlavní části implementace LCG generátoru.

```
// Inicializace konstant
static unsigned long xi = seed;   //počáteční hodnoty
static unsigned long a = 69069;   //multiplikační konstanta
static unsigned long b = 1;       //aditivní konstanta

// Funkce generování čísel
double Random(void)
{
    xi = (xi * a + b);            //implicitní operace modula
    return xi / ((double) ULONG_MAX + 1);
}
```

Na začátku jsou definované konstanty a počáteční hodnota (semínko) následované generující funkcí.

5.1.2 Mersenne twister – implementace

Implementace generátoru Mersenne Twister už není tak jednoduchá, jako implementace LCG generátoru. I když samotná struktura se skládá pouze ze tří krátkých funkcí obsahují tyto funkce složité výpočty za použití především bitových operací.

Před samotnou implementací funkcí, musíme definovat často používané konstanty a deklarovat pomocné pole, se kterým se pracuje:

```
// Definice velikosti pole a pomocných masek
#define N 624
#define UPPER_MASK 0x80000000UL
#define LOWER_MASK 0x7fffffffUL

// Deklarace pole
unsigned long MerTwi[N];
```

Jako první je nutné inicializovat na počátku pole čísel pro další práci. Tato funkce očekává (pokud neurčíme implicitně) jako parametr počáteční hodnotu tzv. semínko:

```
void initArray(unsigned long seed)
```

```
{
  MerTwi[0]=seed;
  for(unsigned int i = 1; i < N; i++)
  {
    MerTwi[i] = 1812433253UL * (MerTwi[i-1] ^ (MerTwi[i-1] >> 30)) + i;
  }
}
```

Nyní již může začít pracovat funkce pro generování čísel pomocí vstupních hodnot z inicializovaného pole. Vygenerované čísla zapisuje zpátky na danou pozici pole.

```
void generate(void)
{
  unsigned long y;
  for(unsigned int i = 0; i < N; i++)
  {
    y = (UPPER_MASK & MerTwi[i]) | (LOWER_MASK & MerTwi[(i+1) % N]);

    if(( y % 2) != 0)
    {
      MerTwi[i] = MerTwi[(i + 397) % N] ^ (y >> 1);
    }
    else
    {
      MerTwi[i] = MerTwi[(i + 397) % N] ^ ( y>> 1) ^ 0x9908b0df;
       }
     }
}
```

Tato funkce se musí volat, vždy když vyčerpáme (vybereme) všechny čísla z pole. Poté je nutné, aby se nám sekvence čísel neopakovala a perioda nebyla pouze `N`, znovu vygenerovat celé nové pole.

Abychom však získali určité číslo z tohoto pole, nemůžeme ho jen tak přečíst přímo z dané pozice. Je nutné si toto číslo na dané pozici vyextrahovat a zabarvit pomocí poslední pomocné funkce, kterou voláme z hlavní funkce.

```
double extract(unsigned long position)
{
  unsigned long i = position % N;
  if(i == 0)
  {
    generate();
  }
  unsigned long y = MerTwi[i];
  y ^= (y >> 11);
  y ^= (y << 7) & 0x9d2c5680UL;
  y ^= (y << 15) & 0xefc60000UL;
  y ^= (y >> 18);
```

```
  return y / ((double)ULONG_MAX +1);

}
```

V této funkci je parametrem `position`, což nám označuje pořadí generovaného čísla. Tento parametr se mění od nuly po číslo požadovaného počtu generovaných čísel. Na začátku je vidět, že tato funkce si generující funkci volá sama vždy, když vyčerpá celé pole pseudonáhodných čísel `MerTwi[]` .

V popisované aplikaci jsem navíc umožnil i volbu implicitního generátoru použitého prostředí. Hlavně z důvodů porovnání kvality generování a výsledků testů s vlastními implementovanými generátory. Umožnil jsem tak získání referenčních výsledků generátorem, vytvořeným a zdokonaleným vývojáři a odborníky firmy Borland. Pro využití jsem však musel výstup generátoru upravit. Implicitně totiž tento generátor produkuje náhodná čísla v rozsahu datového typu *znaménkového* `integeru`. Já však pracuji s vygenerovanými kladnými čísly v rozsahu od nuly po konstantu `ULONG_MAX`. Proto jsem nastavil rozsah implicitního generátoru mezi nulou a maximální hodnotou znaménkového čísla. Abych získal vyšší čísla, násobím výstup generátoru dvěmi. To sice vnáší chybu do výstupních hodnot generátoru na pozici nejnižších bitů zaznamenávaného čísla, ale protože jsou tyto bity na konci transformace ořezány, můžu celou tuto operaci považovat za přijatelnou.

Výstupem všech těchto generátorů jsou celá čísla v intervalu od nuly po maximální hodnotu zvoleného datového typu s rovnoměrným rozložením. Pro transformaci na jiná rozložení, nebo posun podle požadovaných parametrů, jsem použil transformační funkce popsané v následující části.

5.2 Návrh transformací

Protože bychom si nevystačili v aplikacích simulující a modelující určitou situaci nebo vývoj pouze s celočíselnou posloupností s rovnoměrným rozložením, implementoval jsem v mé aplikaci transformační funkce. Ty buď posunují a upravují rovnoměrné rozložení na jeho jiný tvar, nebo převádí toto rozložení na požadované rozložení s danými parametry.

Pro převod jsem použil různé druhy metod, hlavně inverzní metodu a speciální sčítací metodu. Pro jiný tvar rovnoměrného rozložení je nutný pouze posun a nastavení intervalu. Pro transformaci na exponenciální rozložení jsem použil vzorec pro inverzní transformaci, rozšířený o upravující konstanty pro splnění zadaných parametrů. Nejnáročnější transformací je změna rovnoměrného rozložení na normální.

5.2.1 Rovnoměrné rozložení – implementace

Z předešlého víme, že implementované generátory nám dávají výstup s rovnoměrným rozložením v intervalu od 0 do maxima datového typu, tedy $\mathrm{R}(0, \texttt{ULONG\_MAX})$. Většinou však nechceme tak široký rozsah hodnot. Proto je nutné pomocí vzorce dostat tyto hodnoty do tvaru vyhovujícího uživateli. Vzorec pro transformaci je:

$$X = \mathrm{R_n}(b - a) + a$$

$\mathrm{R_n}$ značí číslo s normovaným rovnoměrným rozložením, a a b jsou dolní a horní hranice výsledného intervalu.

Hodnoty a a b jsou zadány uživatelem jako parametr tohoto rozložení. Pro použití této transformace tedy potřebujeme už jen pouze číslo s normovaným rovnoměrným rozložením. Posloupnost s rovnoměrným rozložením máme a stačí je pouze vydělit maximální hodnotou použitého datového typu, abychom získali normované rozložení. Po úpravě výše uvedeného vzorce s novými poznatky, vypadá výsledný vzorec následovně:

$$X = \frac{\mathrm{R}}{\texttt{ULONG\_MAX} + 1}(b - a) + a$$

V tomto tvaru je i použit ve funkci aplikace.

5.2.2 Exponenciální rozložení – implementace

K transformaci na exponenciální rozložení jsem použil velmi výhodné inverzní metody transformace. Protože exponenciální funkce má jednoduchý průběh, snadno nalezneme inverzní funkci, kterou je zcela jistě funkce logaritmu. Transformace se řídí vzorcem:

$$X = x_0 + (-A \ln(\mathrm{R_n}))$$

, kde x_0 značí posun, A intenzitu příchodů a oba jsou parametry zadávanými uživatelem. $\mathrm{R_n}$ je opět číslo z $\mathrm{R}(0, 1)$. Získání $\mathrm{R_n}$ jsem popsal, již v předchozí kapitole, nebo jej lze získat voláním funkce pro transformaci rovnoměrného rozložení v intervalu od 0 do 1.

5.2.3 Normální rozložení – implementace

U transformace na normální rozložení vycházíme ze známých vlastností tohoto rozložení. Důležitým poznatkem, který jsem využil při řešení transformační funkce, je vlastnost, že sčítáním více nezávislých náhodných hodnot libovolného rozložení získáme rozložení normální. Tato metoda vychází z centrální limitní věty, kdy rozdělení střední hodnoty n nezávislých náhodných hodnot s libovolným rozložením, se při zvětšování n blíží k normálnímu rozložení [17]. Při použití $n = 12$ použijeme vzorec:

$$X = (\sum_{i=1}^{12} \mathrm{R_n} - 6)\sigma + \mu$$

σ, μ jsou opět uživatelem zadané parametry. Pro zvýšení kvality by mohlo být použito $n = 24$, ale na druhou stranu by byla metoda pomalejší.

Pro získání jednoho čísla z posloupnosti musí být vygenerováno 12 čísel s normovaným rovnoměrným rozložením. Pro výslednou posloupnost o velikosti např. 10 000 čísel s normálním rozložením pravděpodobnosti výskytu je zapotřebí 120 000 čísel normovaného rovnoměrného rozložení. Aplikace řeší tento problém tak, že nejprve nechá zvolený generátor vygenerovat požadovaný počet čísel s rovnoměrným rozložením. Tyto hodnoty pak postupně použije jako semínka pro vygenerování 12 nových čísel s taktéž rovnoměrným rozložením. Tato čísla již použije ve výše uvedeném vzorci. Voláme sice generétor jen jednou pro vytvoření semínek a pak pokaždé když vytváříme výsledné číslo, ale paměťová náročnost se zvýší pouze o 12 čísel. Ztrácíme sice procesorový čas, ale zato šetříme často důležitější paměťový prostor než kdybychom měli naráz generovat dvanáckrát více čísel najednou a držet je v paměti. Narozdíl od obou předešlých metod je tato metoda ztrátová a vnáší do transformace vlastní chybu.

V aplikaci se požadované rozložení i se svými parametry nastavuje na příslušné záložce. Nejprve uživatel vybere druh rozložení a poté hodnoty parametrů. Přednastaveny jsou parametry pro normované druhy rozložení a uživatel může navíc zvolit možnost generování pouze celých čísel bez desetinné části.

Po této volbě už může být spuštěno generování, během nějž jsou vygenerovány pseudonáhodná čísla s rovnoměrným rozložením v intervalu rozsahu hodnot použitého datového typu. Ihned po vygenerování je spuštěna transformační funkce podle zadaných údajů. Výstupem je soubor s hodnotami, odpovídajícími požadavkům uživatele.

5.3 Návrh testů

Pro implementaci jsem vybral test rovnoměrnosti, který testuje správnou rovnoměrnost rozložení testované posloupnosti. Ostatní testy zkoumají náhodnost hodnot vstupních dat. Mezi tyto testy patří test mezer a run test, který je rozdělen na run – up test a na run – down test.

Testování rovnoměrnosti dává smysl pouze u sérií čísel s rovnoměrným rozložením. Ostatní testy můžeme použít pro libovolné rozložení.

Do této podkapitoly zařadím i implementaci histogramu ,který se zobrazuje v jedné ze záložek pokud je požadován.

5.3.1 Test rovnoměrnosti – implementace

Tento jednoduchý test nejprve zjistí nejnižší a nejvyšší hodnotu testované posloupnosti a určí podle ní rozsah intervalu. Podle uživatelem zadaného čísla rozdělí celý interval na příslušný počet podintervalů. V nich spočítá všechny vzorky spadající právě do příslušného podintervalu. Takto získané hodnoty se vyhodnotí pomocí χ^2.

Nakonec se porovná výsledná hodnota s tabulkovou kritickou hodnotou a rozhodne se o přijetí nulové hypotézy o rovnoměrnosti.

Funkce pro výpočet hodnoty testu rovnoměrnosti si nejprve celé pole testované posloupnosti seřadí podle velikosti a v dalším průchodu počítá postupně počet hodnot spadajících do podintervalu. Protože jsou hodnoty seřazeny, prochází jednoduše od prvního až po poslední podinterval. Seřazením hodnot posloupnosti ji znehodnotím a nemohl bych dále provádět testy náhodnoti čísel, proto tento test volám vždy až jako poslední.

5.3.2 Test mezer – implementace

Funkce při průchodu vstupní posloupností zjišťuje typ jedné z možných kombinací vztahů mezi třemi po sobě následujícími čísly viz 4.6.3. Po rozhodnutí, o kterou kombinaci jde, si poznamená výsledek a posune se dále na nově vzniklou trojici čísel. Po zjištění počtů výskytů všech kombinací v celé posloupnosti, se takto získané pole hodnot opět zpracuje funkce χ^2. Konečný výsledek vyhodnotíme porovnáním s kritickou hodnotou z tabulky.

5.3.3 Run test – implementace

V aplikaci jsou implementovány obě varianty run testu run – up i run – down test. Obě pracují obdobně tak, že postupným průchodem zjišťují délky stále rostoucích resp. stále klesajících podposloupností. V průběhu průchodu posloupností se sčítá počet nalezených podsekvencí stejné délky. Získané pole šesti hodnot výsledných součtů zhodnotí vzorcem podle předpisu v kapitole 4.6.6.

Po porovnání s hodnotou kritického bodu, můžeme rozhodnout o přijmutí či zamítnutí hypotézy náhodnosti.

Všechny výsledky testů jsou zobrazovány v odpovídající záložce a můžou být i tisknuty do csv nebo xml souboru podle jejich pravidel. Při zobrazení výsledků testů je zároveň ukázána hodnota kritického bodu, výsledné rozhodnutí o splnění testu a informace zda byl test prováděn.

Do csv souboru se tisknou informace korektním způsobem. První řádek plní funkci hlavičky a označuje druh obsažených položek. Zbývající řádky obsahují název a výsledek testu, kritickou hodnotu a hladinu významnosti, na které byl test prováděn.

5.3.4 Histogram – implementace

Poslední ze záložek ve výsledné aplikaci obsahuje histogram testované posloupnosti. Histogram je velmi vhodný pro zobrazení a kontrolu rozložení pravděpodobnosti testované sekvence čísel. Nejde o přímo matematické testování, ale lze tak rychle a jednoduše zjistit vizuální kontrolou, jestli se pravděpodobnostní rozložení výskytu hodnot ve vstupních datech blíží předpokládanému.

Implementace je velmi jednoduchá a podobá se postupu testu rovnoměrnosti, kdy se v každém z podintervalů (jejich počet určuje uživatel) určí počet hodnot do něj spadajících. Následně se takto získaný soubor hodnot zobrazí v grafu, kde na vodorovné ose jsou zobrazeny hodnoty intervalů a na svislé ose počet hodnot spadajícího do daného intervalu.

V aplikaci můžeme rozsah intervalů a počet obsahujících hodnot tisknout stejně jako testy do csv a xml souborů.

5.4 Testování

Postupně jsem testoval všechny posloupnosti vygenerované navrženými generátory. Výstupní posloupnosti těchto generátorů měly rovnoměrné rozložení s intervalem $\langle 0, 1 \rangle$. Testy byly prováděny na hladině významnosti α=0,9. Pro porovnání s různě nastavenými parametry jsem zaznamenal pět údajů u každého generátoru. Měnil jsem velikost testované sekvence a počáteční semínko. Protože pracuji pořád na stejné hladině významnosti, musí být výsledek testu rovnoměrnosti a testu mezer nižší než hodnota χ^2 o této hladině významnosti a stupněm volnosti $v = 5$ tj. musí být nižší než hodnota 9,236. Run testy sice nejsou hodnoceny podle χ^2, ale pokud mají splnit nulovou hypotézu, nesmí hodnotu kritického bodu výrazně překročit.

5.4.1 Výsledky testů

V následujících tabulkách jsem zaznamenal výsledky testů s výše popsanými parametry. Každá tabulka obsahuje výsledky testů všech implementovaných generátorů při stejných počátečních podmínkách.

V tabulkách výsledků jsou LCG generátory s různými vnitřními konstantami označeny čísly, korespondující s tabulkou 5.1 na straně 30.

Generátor	Test rovnoměrnosti	Test mezer	Run – up	Run – down
LCG č. 1	2,505	0,852	4,824	2,497
LCG č. 2	4,486	1,063	3,509	6,046
LCG č. 3	12,817	8,946	8,592	8,631
LCG č. 4	5,128	0,232	1,569	4,700
LCG č. 5	0,780	4,966	5,490	3,902
Mersenne twister	4,070	4,275	4,905	5,601
Borland Builder 6.0	6,123	10,870	10,959	13,547

Tabulka 5.2: Tabulka výsledků testů při velikosti vstupních dat 10 000 a počátečním semínku 1

Generátor	Test rovnoměrnosti	Test mezer	Run – up	Run – down
LCG č. 1	3,115	2,900	7,081	3,261
LCG č. 2	2,150	0,999	2,430	4,866
LCG č. 3	5,204	6,909	5,107	9,511
LCG č. 4	4,974	3,193	2,584	5,198
LCG č. 5	5,639	1,801	3,631	3,306
Mersenne twister	6,364	3,095	7,378	3,534
Borland Builder 6.0	2,422	4,618	6,482	6,177

Tabulka 5.3: Tabulka výsledků testů při velikosti vstupních dat 100 000 a počátečním semínku 1

Generátor	Test rovnoměrnosti	Test mezer	Run – up	Run – down
LCG č. 1	4,603	0,691	0,658	6,413
LCG č. 2	1,350	1,286	3,356	5,165
LCG č. 3	8,504	1,357	4,093	4,092
LCG č. 4	0,848	5,196	7,632	9,164
LCG č. 5	4,498	1,078	4,388	3,215
Mersenne twister	4,993	5,363	1,803	1,706
Borland Builder 6.0	7,422	3,449	4,453	12,842

Tabulka 5.4: Tabulka výsledků testů při velikosti vstupních dat 1 000 000 a počátečním semínku 1

Generátor	Test rovnoměrnosti	Test mezer	Run – up	Run – down
LCG č. 1	6,615	1,177	1,831	6,869
LCG č. 2	7,335	1,196	1,544	1,547
LCG č. 3	4,604	1,047	5,203	5,513
LCG č. 4	7,425	8,839	9,938	11,833
LCG č. 5	7,381	0,567	6,534	4,749
Mersenne twister	4,840	2,141	2,777	2,523
Borland Builder 6.0	3,228	1,462	11,911	1,063

Tabulka 5.5: Tabulka výsledků testů při velikosti vstupních dat 10 000 000 a počátečním semínku 1

Generátor	Test rovnoměrnosti	Test mezer	Run – up	Run – down
LCG č. 1	8,164	5,448	4,603	7,929
LCG č. 2	5,285	6,242	7,733	7,731
LCG č. 3	10,356	6,791	12,407	8,394
LCG č. 4	2,889	9,057	3,879	5,166
LCG č. 5	2,613	1,336	5,250	9,422
Mersenne twister	4,038	0,555	4,054	3,631
Borland Builder 6.0	1,578	2,964	3,174	4,994

Tabulka 5.6: Tabulka výsledků testů při velikosti vstupních dat 100 000 a počátečním semínku 1 241 986

5.4.2 Vyhodnocení

Ve vytvořených tabulkách 5.2, 5.3, 5.4 a 5.5 jsem porovnal výsledné posloupnosti generátorů při různě velké generované a testované posloupnosti. Porovnání tabulek ukazuje nestejné výsledné hodnoty testů generovaných posloupností při stejných počátečních hodnotách generátorů.

Ověřil jsem, že kvalitu LCG generátoru velmi ovlivňují zvolené konstanty. V tabulkách je jasně vidět závislost mezi výsledky testů a použitými konstantami.

Některé generátory dávají lepší výsledky rovnoměrnosti až pro vyšší počet výstupních hodnot (například generátor LCG číslo 3 nebo integrovaný generátor prostředí Borland Builder). Naopak u LCG generátoru číslo 1 je vidět jeho nedostatky výstupu až při vysokém počtu generovaných čísel. Pro menší počet vygenerovaných čísel dává mnohem lepší výsledky testu rovnoměrnosti než při vysokém počtu.

Test mezer ve většině případů dopadl nadmíru dobře a z nízkých hodnot je vidět, že 6 možných kombinací porovnání tří čísel nastává stejně často.

U run testů jsou znát velké výkyvy hodnot mezi jednotlivými generátory i mezi prováděnými testy. Značí to obtížnost run testů, kdy je opravdu těžké dodržet co nejkratší rostoucí resp. klesající sekvenci čísel.

Jiných zajímavých závěrů jsem dosáhl při porovnání práce generátorů s různými počátečními hodnotami. Z tabulek 5.3 a 5.6 zjišťuji značnou závislost mezi počáteční hodnotou (semínkem) generátorů a kvalitou výsledné sekvence čísel. Zcela jistě by se našla i počáteční hodnota, po které by některé generátory vyprodukovaly dokonce nevyhovující výstupy.

Podle tabulek mohu odhadovat vhodnost počátečního semínka. Zatímco testy prvních 3 druhů LCG generátorů se semínkem 1 241 986 dávaly výrazně horší výsledky, zbývající LCG generátory si naopak oproti tomu ve výsledku polepšili. Mersenne twister a implicitní generátor se zachovaly podobně v obou případech, což je ideální stav. Tak lze rozhodnout, které ze semínek je pro daný generátor lepší. Možné ideální semínko, které by *vhodně* inicializovalo většinu generátorů, se může nacházet např. mezi velkými prvočísly. Nebylo by však lehké jej nalézt a museli bychom jít metodou postupného zkoušení nebo za pomocí rozboru vnitřní struktury generátorů a jejich konstant.

Mersenne twister potvrdil své kvality i při tak jednoduché variantě, jakou je implementace, použitá v mé aplikaci. Vždy dává vyhovující výsledky a chová se dost stabilně, jak z pohledu rovnoměrnosti tak náhodnosti.

Celkově z tabulek vyplývá, že téměř ve všech případech generátory nedosahují hodnot kritického bodu a můžu tedy přijmout nulovou hypotézu pro testované vlastnosti. U testu rovnoměrnosti nulovou hypotézu o tvrzení, že daná posloupnost má rovnoměrné rozložení, a u zbývajících testů hypotézu o náhodnosti čísel. Přestože v některých z ojedinělých případů musím z výsledků odmítnout nulovou hypotézu, nejde nikdy o výrazný rozdíl mezi výslednou hodnotou a hodnotou kritického bodu.

Kapitola 6

Závěr

V teoretické části jsme se nejprve seznámili s pojmem náhodné a pseudonáhodné číslo, jeho vlastnostmi a základními požadavky, které musí splňovat, abychom je mohli použít pro další práci.

Následně jsme si ukázali jednoduché techniky generování pseudonáhodných čísel i jejich složitější generátory. Velká část tohoto oddílu popisovala nejpoužívanější generátory v oblasti modelování a simulace, jako je lineární kongruentní generátor, generátor Mersenne twister, generátory s posuvným registrem a jejich vnitřní strukturu, výhody a nevýhody. Z oblasti kryptografie byly alespoň nastíněny některé z generátorů a uveden přehled nejzmiňovanějších.

Druhá teoretická část se skládala z popisu rozložení pravděpodobnosti výskytu čísla a stručného přehledu nejznámějších druhů rozložení. Těžištěm byl popis metod transformací rovnoměrného rozložení na rozložení jiné. Především šlo o inverzní metodu, metodu vylučovací, kompoziční a jiné speciální metody.

Závěrečný oddíl teoretické části práce se zaměřil na požadavky výstupních posloupností generátorů a jejich testování. Byly zmíněny způsoby vyhodnocování testů a samozřejmě uvedeny některé z nejznámějších testů. Hlavní směrem byly testy rovnoměrnosti a náhodnoti posloupnosti.

Praktická část této práce se zabývala vlastní navrženou a naprogramovanou aplikací pro generování a testování pseudonáhodných čísel s různým rozložením jejich výskytu.

Ve třech částích zde byly postupně popsány použité metody pro generování čísel, transformace pro vytváření tří hlavních rozložení a použité testy společně s vytvářením histogramu.

Poslední část porovnávala výsledky testů jednotlivých generátorů a ohodnotila takto získané údaje. Zhodnotila tím i na základní úrovni kvalitu implementovaných generátorů a určila jejich omezení.

V budoucnosti by bylo zajímavé pokusit se výstup implementované aplikace použít v praxi zavedené simulační aplikaci a ověřit tak kvalitu vygenerovaných čísel.

Možným rozšířením práce by mohla být část, zabývající se kryptografií a kryptografickými generátory. Studováním cílených útoků na prolomení šifer, využívajících pro svou práci peudonáhodná čísla a naopak zajišťování možné obrany proti nim. Taktéž by mohly být porovnávány kvality výstupů z algoritmických generátorů a tzv. šifrovacích karet.

Literatura

[1] *Blum Blum Shub*. [online], [rev. 2008-02-19], [cit. 2008-04-16], Dostupné na URL: http://en.wikipedia.org/wiki/Blum_Blum_Shub.

[2] *Mersenne twister*. [online], [rev. 2008-04-10], [cit. 2008-04-16], Dostupné na URL: http://en.wikipedia.org/wiki/Mersenne_twister.

[3] ENTACHER, K.: *Run Tests*. [online], [rev. 2008-06-13], [cit. 2008-04-16], Dostupné na URL: http://random.mat.sbg.ac.at/tests/empirics/runs/.

[4] FABIÁN, F.; KLUIBER, Z.: *Metoda Monte Carlo a možnosti jejího uplatnění*. Prospektrum, 1998, ISBN 80-7175-058-1.

[5] GUŠTAR, M.: Generování náhodně proměnných veličin v metodě Monte Carlo. In *Rozvoj koncepcí posudku spolehlivosti stavebních konstrukcí*, Praha, ARTech, 2000, ISBN 80-02-01344-1, s. 5–8.

[6] KAHANEK, P.: *Generátory náhodných čísel v Matlabu*. [online], 2002, [rev. 2002-04-30], [cit. 2008-04-15], Dostupné na URL: http://phobos.vscht.cz/\konference_matlab/MATLAB05/prispevky/kahanek/kahanek.pdf.

[7] KAPADIA, A.: *Random Number Generators* . [online], 2001, [rev. 2001-05-30], [cit. 2008-04-15], Dostupné na URL: http://www.cs.dartmouth.edu/~akapadia/project2/.

[8] KARLEN, D.: Part III: Monte Carlo Methods. [online], 1998/99, [cit. 2008-04-15], Dostupné na URL: http://kfe.fjfi.cvut.cz/~limpouch/numet/random/Monte-karlen.pdf.

[9] Klíma, V.: Náhoda na objednávku. *Chip*, 1998: s. 58, 60, 62, [cit. 2008-04-16], Dostupné na URL: http://cryptography.hyperlink.cz/1998/1998.html.

[10] KNUTH, D. E.: *The Art of Computer Programming*. Addison-Wesley Publishing Company, 1981, vol.2.

[11] KRHOVJÁK, J.: *Problematika náhodných a pseudonáhodných sekvencí v kryptografických eskalačních protokolech a implementacích na čipových kartách*. Diplomová práce, Masarykova univerzita v Brně, Fakulta informatiky, 2005.

[12] LAW, A. M.; KELTON, W. D.: *Simulation modeling and analysis*. Konvoj, 2000, ISBN 0-07-059292-6.

[13] MATSUMOTO, M.: *What is Mersenne Twister (MT)?* [online], [cit. 2008-04-16], Dostupné na URL: http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/ewhat-is-mt.html.

[14] MÁLEK, O.: *Generování pseudonáhodných dat založené na použití LFSR.* Bakalářská práce, Masarykova univerzita v Brně, Fakulta informatiky, 2007.

[15] NIEDERREITER, H.: Testing of Pseudorandom Numbers. [online], [cit. 2008-04-15], Dostupné na URL: http://www.cimpa-icpam.org/NotesCours/PDF/2005/Niederreiter05-2.pdf.

[16] PELIKÁN, J.: Modelování a simulace náhodných jevů. In *Cesty k uplatnění pravděpodobnostního posudku bezpečnosti, provozuschopnosti a trvanlivosti konstrukcí v normativních předpisech a v projekční praxi,*, 2002, ISBN 80-02-01489-8, s. 93–98.

[17] RÁBOVÁ, Z.; PERINGR, P.; kol.: *Modelování a simulace.* Vysoké učení technické v Brně, 1992, ISBN 80-214-0480-9.

[18] SINCLAIR, B.: *Testing random().* [online], 1997, [rev. 1997-04-24], [cit. 2008-04-15], Dostupné na URL: http://www.ece.rice.edu/~bs/rng/test.html.

[19] TESAŘ, P.: *Generátory náhodných bitů – přednáškové materiály.* [online], [cit. 2008-04-15], Dostupné na URL: www.comtel.cz/files/download.php?id=2439.

[20] WARD, R.; MOLTENO, T.: *Table of Linear Feedback Shift Registers.* [online], [rev. 2007-10-26], [cit. 2008-04-16], Dostupné na URL: http://www.physics.otago.ac.nz/px/research/electronics/papers/\technical-reports/lfsr_table.pdf.

[21] WEISSTEIN, E. W.: *Stirling Number of the Second Kind.* [online], [rev. 2008-04-15], [cit. 2008-04-16], Dostupné na URL: http://mathworld.wolfram.com/StirlingNumberoftheSecondKind.html.

Příloha A

Seznam použitých symbolů

$E(X)$	střední hodnota
$D(X)$	rozptyl
$\mathrm{cov}(X_1, X_2)$	kovariance
R(0, 1), $\mathrm{R_n}$	normované rovnoměrné rozložení
R	obecný prvek rovnoměrného rozložení
r	určitý prvek rovnoměrného rozložení
$f(x)$	funkce hustoty pravděpodobnosti (z kontextu i obecně „*funkce*“)
$F(x)$	distribuční funkce (z kontextu i obecně „*funkce*“)
N(μ,σ^2)	normální rozložení pravděpodobnosti
X	obecné výsledné náhodné číslo
x	určité výsledné náhodné číslo
H	hypotéza
α	hladina významnosti (chyba I. typu)
β	chyba II. typu
χ^2	chí-kvadrát
V	hodnota chí-kvadrátu
v	stupeň volnosti
n	obecně počet hodnot (nejčastěji celková velikost dat nebo počtu po sobě jdoucích čísel)
k	obecně počet hodnot (nejčastěji počet intervalů)
m_i	obecně počet hodnot (nejčastěji počet hodnot v i-tém intervalu)
d	označení počtu dimenzí nebo maximálního dosažitelného čísla
a, b	meze intervalu
c	délka intervalu
h	obecně počet hodnot (nejčastěji počet čísel v určité sekvenci)