Samppa Hynninen

Satunnaislukujen generointi havaitun aineiston pohjalta

Tietotekniikan
kandidaatintutkielma
26. elokuuta 2012

Jyväskylän yliopisto

Tietotekniikan laitos

Jyväskylä

Tekijä: Samppa Hynninen
Yhteystiedot: sajusahy@student.jyu.fi
Työn nimi: Satunnaislukujen generointi havaitun aineiston pohjalta
Title in English: Generating random numbers from observed data
Työ: Tietotekniikan kandidaatintutkielma
Sivumäärä: 28
Tiivistelmä: Satunnaislukujen generointia hyödynnetään laajalti tietotekniikan alan eri sovellusalueilla, kuten kryptografiassa. Tässä tutkielmassa käsitellään eri tavoin jakautuneiden satunnaislukujen generointia ja erityyppisten generaattoreiden ominaisuuksia. Lisäksi tutkitaan satunnaislukugeneraattoreiden tilastollista testaamista ja havaintoaineiston hyödyntämistä satunnaislukuja generoitaessa.
Abstract: Random number generation is widely used in different fields of information technology, such as cryptography. This thesis examines the generation of random numbers from different distributions and features of various kinds of random number generators. Furthermore, statistical testing of random number generators is studied. Besides these, this thesis covers how to use observed data when generating random sequences.
Avainsanat: satunnaisluvut, satunnaislukugeneraattorit, tilastolliset testit
Keywords: random numbers, random number generators, statistical tests

Sisältö

1 Johdanto

Kandidaatintutkielmani aiheena on satunnaislukujen generointi. Aihe valikoitui tutkielman kohteeksi, sillä satunnaislukujen sovelluskohteita on nykyään enemmän kuin koskaan aikaisemmin. Tämän lisäksi satunnaislukuja käytetään monissa kriittisissä sovelluksissa, joissa satunnaislukugeneraattoreiden tuottamien satunnaislukujen laatu on erittäin keskeisessä asemassa. Näitä moderneja sovelluskohteita ovat esimerkiksi erilaiset salaukset ja muut kryptografiset sovellukset, uhkapelit sekä simulointi ja mallinnus muilla tieteenaloilla. Tarkoituksena on tutkia, mitä satunnaisluvut ovat, miten niitä generoidaan ja erityisesti sitä, miten havaitusta datasta voidaan generoida satunnaislukuja.

Kandidaatintutkielma alkaa luvussa 2 siitä, että selvitetään, mitä satunnaisluvuilla tietotekniikan alalla ylipäänsä tarkoitetaan. Tämän jälkeen käydään läpi näennäisesti satunnaisten ja todella satunnaisten lukujen erot. Kolmannessa kappaleessa käsitellään tasajakautuneiden satunnaislukujen generointia. Ensin käsitellään mitä tasajakautuneisuudella tarkoitetaan ja tämän jälkeen tutkitaan erilaisia generaattoriluokkia, joilla tasajakautuneita satunnaislukuja voidaan generoida. Neljännessä luvussa perehdytään muiden jakaumien satunnaislukujen generointiin. Tätä varten esitellään pääasialliset menetelmät, joissa hyödynnetään myös tasajakauman satunnaislukuja. Luku 5 luo katsauksen tämän päivän standardeihin satunnaislukujen generoinnissa. Luvussa tutkitaan uusimpien satunnaislukugeneraattoreiden ominaisuuksia sekä niiden toimintaperiaatteita. Kuudes luku keskittyy siihen, kuinka satunnaislukugeneraattoreita voidaan testata tilastollisesti ja mitä erilaisilla testeillä saavutetaan. Tämän lisäksi luvussa käsitellään erilaisia testikirjastoja sekä vertaillaan eri generaattoreiden hyvyyttä testien valossa. Viimeinen luku vastaa kysymykseen, kuinka havaintoaineistosta saadaan muodostettua todennäköisyysjakauma.

Tutkielma on toteutettu kirjallisuuskatsauksena. Kirjallisuutta on etsitty IEEE-Exploresta, ACM Digital Librarysta sekä Google Scholar -tietokannoista. Näiden verkon tietokantojen lisäksi tutkielmassa hyödynnettiin Jyväskylän Yliopiston kirjaston kokoelmia. Tutkielmassa on pyritty hyödyntämään mahdollisimman tuoretta tutkimustietoa yhdessä alan perusteosten kanssa. Suurin osa aihepiirin kirjallisuudesta on englanninkielistä, jonka myötä myös tutkielma pohjautuu lähes kokonaan ulkomaiseen tutkimustietoon.

2 Satunnaisluvuista

Puhuttaessa satunnaislukujen generoimisesta tietokoneella vastaan tulee hyvin nopeasti kysymys siitä, mitä lukujen satunnaisuudella tarkoitetaan. Tämä todetaan myös kirjassa Numerical Recipes in C [2], sillä kaikkien tietokoneohjelmien tuottama sisältö on aina täysin ennustettavissa. Ratkaisuna tähän perustavanlaatuiseen ongelmaan on käytetty satunnaislukugeneraattoreita, joita yleensä kutsutaan edellämainitusta syystä pseudosatunnaislukugeneraattoreiksi, eli ne tuottavat näennäisesti satunnaisia lukuja (engl. pseudo random). Käytännön sovelluksissa monet fysikaalisten prosessien tuottamat todella satunnaiset luvut ovat sellaisenaan oikeastaan hyödyttömiä, sillä niiden hyödyntämiseksi sovelluksissa täytyy lähes aina tuntea havaittujen arvojen jakauma [7]. Näistä voidaan kuitenkin tietyillä muunnoksilla tuottaa tasajakautuneita satunnaislukuja. Esimerkkinä tällaisesta keinosta on ns. "von Neumann unbiasing" [3], jota voidaan soveltaa jonoon tuntemattomasta jakaumasta poimittuja riippumattomia binäärilukuja $\{x_i\}_{i=1}^n$, missä $p = Pr[x_i = 0] \neq 1/2$, $q = Pr[x_i = 1]$. Näistä luvuista saadaan muodostettua tasajakautuneita riippumattomia lukuja ilman p:n tuntemista siten, että jaetaan jono $\{x_i\}_{i=1}^n$ pareihin. Tämän jälkeen hylätään parit, joissa kummatkin bitit ovat samat. Sitten korvataan kaikki 01-parit bitillä 1 ja 10-parit bitillä 0. Tällä menetelmällä saadaan n kappaleesta tuntemattoman jakauman bittejä npq kappaletta harhattomia bittejä.

Erään määritelmän tietokoneiden tuottamien lukujen satunnaisuudesta antaa Numerical Recipes in C [2], jossa määritellään, että deterministisen tietokoneohjelman, joka tuottaa satunnaisia jonoja lukuja, tulisi olla kaikin tavoin erillinen ja riippumaton ohjelmasta, joka käyttää näitä tuotettuja satunnaislukuja. Satunnaislukugeneraattoreita voidaan testata useilla erilaisilla tilastollisilla testeillä, jolloin niiden vertaileminenkin onnistuu. Monesti onkin täysin ohjelman käyttötarkoituksesta ja ohjelmoijasta kiinni, mikä on riittävän hyvä, eli riittävän satunnainen satunnaislukugeneraattori. Hyvänä ideana pidetäänkin, ettei yhtä generaattoria tai edes saman tyyppisiä generaattoreita käytettäisi kaikissa sovelluksissa, vaan kannattaa käyttää jopa yhdessä ohjelmassa eri generaattoreita ja kaikille generaattoreille vielä eri alkuarvoja [7].

Edellämainituilla tilastollisilla testeillä voidaan olennaisesti testata eri satunnaislukugeneraattoreiden tuottamia lukujonoja. Monet sovellukset vaativatkin käytetyiltä satunnaisluvuilta esimerkiksi tarpeeksi suurta syklin pituutta tai niiden jakautumista riippumattomasti tietylle välille toimiakseen, ja tilastollisilla testeillä voidaan testata, täyttääkö tietty generaattori vaaditut ominaisuudet. Tällaisia testejä on äärettömän paljon. Tästä erilaisten testien määrästä johtuen ei voida määritellä

mitään kokoelmaa testejä, joka olisi täydellinen [9].

Tavallisesti satunnaislukujen generointi jakaantuu kahteen osaan. Tavallisin tapaus on välille $(0, 1)$ tasajakautuneiden satunnaislukujen generointi. Näitä lukuja tarvitaan myös, kun generoidaan muiden haluttujen jakaumien lukuja. Käytännössä ensin generoidaan lukuja välille $(0, 1)$, joita sitten skaalataan halutulle välille, kuten myöhemmin esitetään [7].

3 Tasajakautuneet satunnaisluvut

Tasajakautuneilla $(0, 1)$-satunnaisluvuilla tarkoitetaan reaalilukuja U_n, jotka ovat jakautuneet tasaisesti reaaliakselin välille $(0, 1)$. Tietokoneiden tarkkuuden, jolla ne esittävät reaaliluvut, ollessa äärellinen, saadaan $[0, 1)$-välin satunnaislukuja generoitua, kun generoidaan luonnollisia lukuja X_n väliltä $[0, 2^s)$, missä s on yleensä sananpituus tietokoneessa. Tämän jälkeen tästä saadaan $[0, 1)$-välin reaaliluku skaalaamalla 2^s:llä, eli $U_n = X_n/2^s$ [1]. Yksi vanhimmista ja yleisimmistä algoritmeista tasajakautuneiden satunnaislukujen generointiin on Linear Congruential Method[1] (myöh. LCM), jonka kehitti D. H. Lehmer vuonna 1949. Generaattori tunnetaan myös nimellä Lehmer-generaattori.

LCM tarvitsee toimiakseen neljä muuttujaa, joiden valinta jää käyttäjän tehtäväksi. LCM:n toimivuus riippuukin hyvin pitkälti siitä, onko seuraavat muuttujat valittu hyvin:

- m, modulus
- a, kertoja
- c, lisäys
- X_0, alkuarvo.

Nyt näistä saadaan halutunpituinen jono satunnaislukuja $\langle X_n \rangle$ asettamalla

$$X_{n+1} = (aX_n + c) \bmod m.$$

3.1 Yhdistelmägeneraattorit

Yhdistelmägeneraattoreissa ideana on käyttää useampaa lyhyen syklin generaattoria. Esimerkiksi Wichmann-Hillin [5] generaattorissa käytetään kolmea yksinkertaista generaattoria, joista jokaisen modulus on alkuluku. Olkoon näillä syklit m_1, m_2 ja m_3. Kun näillä generaattoreilla tuotetaan jonot X_i ja määritellään $U_i = X_i/m_i$ saadaan tuloksena jono

$$U = U_1 + U_2 + U_3 \bmod 1.$$

Vuonna 1988 L'Ecuyer [10] esitti oman versionsa yhdistetystä generaattorista, jossa hän yhdisti k kappaletta generaattoreita, joilla on alkulukumodulot m_j siten, että

[1] Algoritmien nimet jätetty suomentamatta

$(m_j - 1)/2$ ovat keskenään jaottomia alkulukuja ja kertoimet siten, että ne tuottavat täydet syklit. Määritellään nyt i:nnen generaattorin sykliksi $x_{i,1}, x_{i,2}, x_{i,3}, ...$ Nyt saadaan j:s satunnaisluku jaksossa asettamalla

$$x_j \equiv \sum_{s=1}^{k} (-1)^{s-1} x_{s,j} \bmod (m_1 - 1).$$

Vuonna 1997 L'Ecuyer ja Andres esittelivät vielä pidemmälle kehitetyn version edellämainitusta yhdistetystä generaattorista. Tässä generaattorissa ideana oli satunnaislukugeneraattori, joka pohjautuu neljän lineaarisesti kongruentin satunnaislukugeneraattorin yhdistämiseen. Tämän esitellyn generaattorin syklinpituudeksi saatiin 2^{121} [11], kun vanhemman, vuonna 1988 julkaistun generaattorin sykli oli vain 2^{61} [10].

Gentle mainitsee kirjassaan [7], että generaattoreiden yhdistäminen voi todella parantaa niiden satunnaisuutta, mutta pahimmassa tapauksessa, johtuen generaattoreiden äärellisestä määrästä tiloja, voi tilanne ollakin se, että eri generaattoreiden erilliset huonot ominaisuudet vain korostuvat yhdistettäessä niitä. Esimerkiksi seuraavien jonojen yhdistäminen antaisi lopputuloksena syklin pituudeksi 1:

$$x_1, x_2, x_3, ..., x_k, ...$$

$$x_k, x_{k-1}, x_{k-2}, ..., x_2, x_1, ...$$

Yhdistettäessä satunnaislukugeneraattoreita tulee myös huomioida mahdollisuus, että yhdistelmägeneraattori saattaa jopa huonontaa kummankin yhdistettävän generaattorin tuloksien laatua eikä yhdistäminen takaa lainkaan generaattorien tuottamien tulosten poikkeuksien poistumista [7].

3.2 Sekoitetut generaattorit

Sekoitetut generaattorit toimivat siten, että poimitaan jollakin generaattorilla X taulukosta alkio ja lasketaan alkiolle arvo toisella generaattorilla Y. Näissä sekoitetuissa generaattoreissa jokainen tulos siis vaatii kaksi satunnaislukua. Tällöin saadaan tuloksena pidempi sykli satunnaislukujen jonolle kuin yksittäisellä generaattorilla ja vähennetään peräkkäiskorrelaatiota. Yhtenä vaihtoehtona on myös käyttää vain yhtä generaattoria, jolla poimitaan taulukon arvo ja lasketaan alkiolle arvo. Numerical Recipes in C esittelee erään sekoitetun generaattorin, jonka ovat kehittäneet Park ja Miller [8]. Kyseinen generaattori läpäisee tilastolliset testit, kun syklin pituus on luokkaa $m < 20 \cdot 10^8$ [2]. Kyseinen generaattori käyttää sekoitusalgoritminaan Bays–Durham-sekoitusta, jonka ideana on käyttää yhtä ja samaa generaattoria

sekoittamaan itsensä. Käytännössä Bays–Durham-sekoitus toimii seuraavasti generoitaessa jono y_i [7]:

1. Määritetään generaattorilla taulukko T, jossa alkiot $x_1, x_2, ..., x_k, i = 1$.
2. Generoidaan x_{k+i} ja asetetaan $y_i = x_{k+i}$.
3. Generoidaan j y_i:stä (käyttäen $\mathrm{mod} k$:ta).
4. $i = i + 1$.
5. Asetetaan $y_i = T(j)$.
6. Generoidaan x_{k+i} ja päivitetään $T(j)$:hin x_{k+i}. Palataan kohtaan 3.

Tämä Bays–Durham-sekoitus pidentää sykliä, jos $k! > s$, missä k on taulukon alkioiden määrä ja s alkuperäisen generaattorin syklin pituus [13],[7].

4 Muiden jakaumien satunnaisluvut

Kaksi tärkeintä menetelmää muiden kuin tasajakautuneiden $(0, 1)$-satunnaislukujen generointiin ovat käänteismuunnosmenetelmä ja hyväksymis–hylkäys-menetelmä. Nämä menetelmät eroavat siinä, että käänteismuunnosmenetelmässä täytyy tuntea satunnaismuuttujan x kertymänfunktion F_x käänteisfunktio F_x^{-1}. Hyväksymis–hylkäys-menetelmässä sen sijaan riittää, että tunnetaan jokin funktio $g(x) > f(x)$ ja tässä $f(x)$ on halutun satunnaismuuttujan tiheysfunktio.

4.1 Käänteismuunnosmenetelmä

Käänteismuunnosmenetelmässä on halutun jakauman tiheysfunktio $f(x)$. Tälle tiheysfunktiolle olkoon nyt kertymäfunktio $F_x : x \to (0, 1)$. Nyt saadaan satunnaisluvut generoitua seuraavasti [2]:

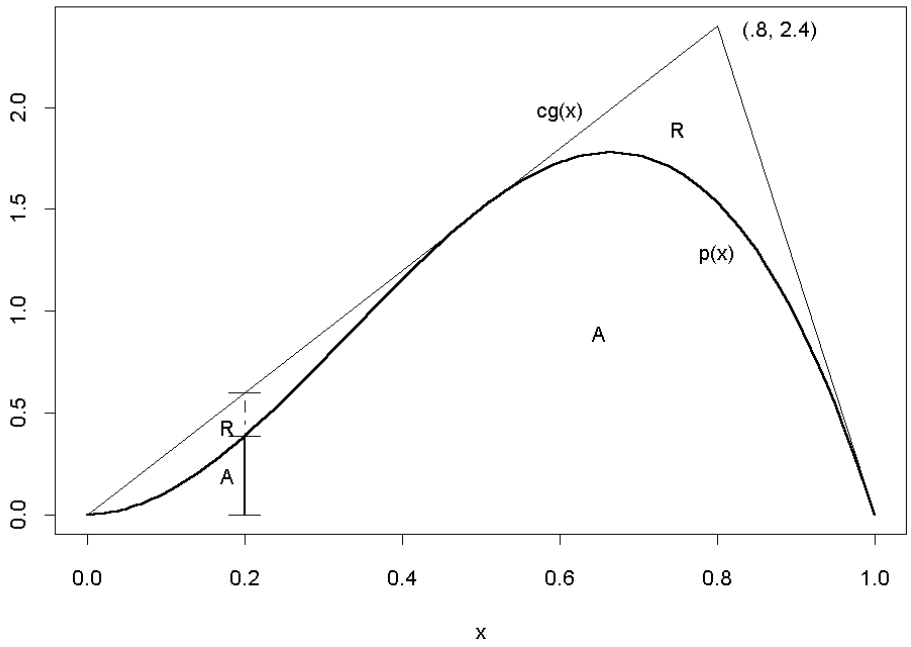
1. Johdetaan kertymäfunktion F_x käänteisfunktio F_x^{-1}, jolle $F_x^{-1}(F_x(x)) = x$.
2. Generoidaan satunnaisluku u jakaumasta $U \sim \mathrm{Tas}(0, 1)$.
3. Lasketaan $x = F_x^{-1}(u)$.
4. Nyt x on halutun jakauman satunnaisluku.

Käänteismuunnosmenetelmästä on myös erillinen versio diskreeteille jakaumille, joka poikkeaa edelläesitetystä algoritmista siinä, että diskreetissä versiossa satunnaismuuttujan x tulosvaihtoehdot on järjestetty suuruusjärjestykseen taulukkoon kertymäfunktion perusteella, eli $F_{x_1} < F_{x_2}$. Algoritmi on siis seuraava [12]:

1. Järjestetään satunnaismuuttujan x tulosvaihtoehdot $x_1, ..., x_n$ suuruusjärjestykseen siten, että $F_x(x_{i-1}) < F_x(x_i)\ \forall i \in [1, n]$.
2. Generoidaan satunnaisluku u jakaumasta $U \sim \mathrm{Tas}(0, 1)$.
3. Valitaan x_i, jolle pätee $F_x(x_{i-1}) < u \leq F_x(x_i)$.
4. Nyt valittu x_i on haluttu luku.

4.2 Hyväksymis–hylkäys-menetelmä

Jos satunnaismuuttujalle x ei tunneta kertymäfunktion käänteisfunktiota F_x^{-1}, voidaan hyväksymis–hylkäys-menetelmää käyttää generoitaessa satunnaislukuja x:n



Kuva 1: Hyväksymis–hylkäys-menetelmä [7].

jakaumasta. Nyt tarvitaan jokin satunnaismuuttuja y, jonka jakauman tiheysfunktiolle g pätee $f(y) \leq cg(y)\ \forall y$ ja jollekin c. Satunnaismuuttuja y tulee valita siten, että sen kertymäfunktio ja kertymäfunktion käänteisfunktio tunnetaan analyyttisesti. Hyväksymis–hylkäämis-menetelmä toimii nyt seuraavasti [2],[4]:

1. Generoidaan y jakaumasta $g(y)$ käyttäen esimerkiksi käänteismuunnosmenetelmää, kun y:n kertymäfunktion käänteisfunktio tunnetaan.

2. Generoidaan u jakaumasta $U \sim Tas(0, 1)$

3. Jos $u \leq \frac{f(y)}{cg(y)}$, valitaan $x = y$, muuten hylätään ja palataan alkuun.

5 Nykypäivän standardit

Nykyään satunnaislukugeneraattoreiden käyttökohteet ovat hyvin monimuotoiset ja generaattoreilta vaaditaankin paljon ominaisuuksia. Tänä päivänä käyttökohteita ja sovelluksia satunnaislukugeneraattoreille on useilla eri aloilla, kuten esimerkiksi tietokonesimuloinnissa, uhkapeleissä, kryptografiassa ja tilastollisessa satunnaisotannassa. Uusissa sovelluksissa generaattoreilta vaaditaan mm. pitkää syklin pituutta. Tästä syystä satunnaislukujen generointi kehittyy alana jatkuvasti, kun uusia satunnaislukugeneraattoreita kehitetään vastaamaan nykypäivän tarpeita. Uudet generaattorit tuovat mukanaan pidempiä syklejä ja niissä parannetaan vanhempien ratkaisuiden puutteita. Ohessa esitellään kaksi tämän hetken tunnettua satunnaislukugeneraattoria ja niiden ominaisuuksia. Yleinen toimintaperiaate uusimmille satunnaislukugeneraattoreille on seuraavanlainen [15]:

$$x_i = Ax_{i-1}$$

$$y_i = Bx_i$$

$$u_i = \sum_{l=1}^{w} y_{i,l-1} 2^{-l},$$

missä $x_i = (x_{i,0}, ..., x_{i,k-1})^T$ on k-bittinen tilavektori ja $y_i = (y_{i,0}, ..., y_{i,w-1})^T$ w-bittinen tulosvektori askeleessa i. A on $k \times k$ todennäköisyysmatriisi ja B on $w \times k$ muunnosmatriisi, k, w ovat positiivisia kokonaislukuja ja $u_i \in [0, 1)$ on tulos askeleella i.

5.1 Mersenne Twister

Yksi tunnetuimmista nykyään laajalti käytössä olevista satunnaislukugeneraattoreista on Makoto Matsumoton ja Takuji Nishimuran vuonna 1997 kehittämä Mersenne Twister. Kyseinen generaattori tuottaa syklin $2^{19937} - 1$ tietyillä parametrien valinnoilla [6]. Se tuottaa 623-ulotteisen tasajakauman, vaikka generaattorin käyttämä työmuisti on vain 624 sanaa. 623-ulotteisella jakaumalla tarkoitetaan sitä, että lasketaan todennäköisyys tapahtumalle A, jossa on 623 muuttujaa. Käytännössä tämä tarkoittaa sitä, että

$$P(A) = \int_A p(\overline{x}) \mathrm{d}\overline{x},$$

missä

$$\overline{x} = (x_1, x_2, x_3, ..., x_{623})$$

ja

$$\mathrm{d}\overline{x} = \mathrm{d}x_1, \mathrm{d}x_2, \mathrm{d}x_2, ..., \mathrm{d}x_{623}.$$

Mersenne Twister vaatii toimiakseen melko monimutkaisen alustuksen [7], joka oli alkuperäisessä [6] versiossa puutteellinen. Nopeudeltaan Mersenne Twister on samaa luokkaa kuin standardin ANSI-C -kirjaston rand(). Generaattori on kehitetty erityisesti Monte Carlo -simulointia ja muita tilastollisia simulointeja varten. Kuten monet muutkin tunnetuimmista pitkän syklin satunnaislukugeneraattoreista, perustuu Mersenne Twisterin toiminta lineaarisiin differenssiyhtälöihin $\mod 2$.

Mersenne Twisterin toiminta perustuu siihen, että se sekoittaa sisäisen tilansa ja ajaa sisäisen tilansa kaikki sanat nopean muunnosfunktion läpi tuottaen näin 624 lukua. Siis $x_{n+1} = F(x_n, x_{n-1}, x_{n-2}..., x_{n-623})$. Tällöin sillä on siis 624 sanan sisäinen tila, eli tilavektorissa on $32 \cdot 624 = 19968$ bittiä.

Mersenne Twisteristä on kehitetty myös noin kaksi kertaa nopeampi 128-bittinen versio vuonna 2006. Tämä generaattori on nimeltään SIMD-oriented Fast Mersenne Twister [14]. SFMT on siis Mersenne Twister päivitettynä nykypäivään. SFMT:ssä on myös paremmat ominaisuudet toipumiseen tilasta, jossa tilavektorissa on paljon nollia ja vähän ykkösiä (ns. 0-excess state). Myös sen tasajakauman dimensio on korkeampi kuin alkuperäisessä Mersenne Twisterissä [14].

5.2 WELL

WELL, eli Well equidistributed long-period linear -satunnaislukugeneraattori perustuu lineaaristen differenssiyhtälöiden $\mod 2$ ratkaisemiseen, kuten myös Mersenne Twister. Kyseinen generaattori tekee enemmän bittimuunnoksia eri vaiheiden välillä kuin monet muut generaattorit, joilla on suuri syklin pituus. Monet muut generaattorit, kuten esimerkiksi juurikin edelläkäsitelty Mersenne Twister, esittävät tilansa suuren bittimäärän avulla, mutta tilasiirtymät vaativat vain muutamia operaatioita 32-bittisillä sanoilla, jolloin suurta osaa biteistä ei muokata tilamuunnoksissa lainkaan [15]. WELL on myös paremmalla tarkkuudella tasajakautunut kuin esimerkiksi edellämainittu Mersenne Twister [15]. Tästä huolimatta WELL-generaattorin syklin pituus on yhtä pitkä kuin Mersenne Twisterillä ja niiden nopeuskin on yhtäläinen.

5.3 Linux random number generator

Linuxin pseudosatunnaislukugeneraattori (myöh. LRNG) perustuu datan keräämiseen eri laiteajureilta ja muista lähteistä. Tämän generaattorin yhteydessä entropiak-

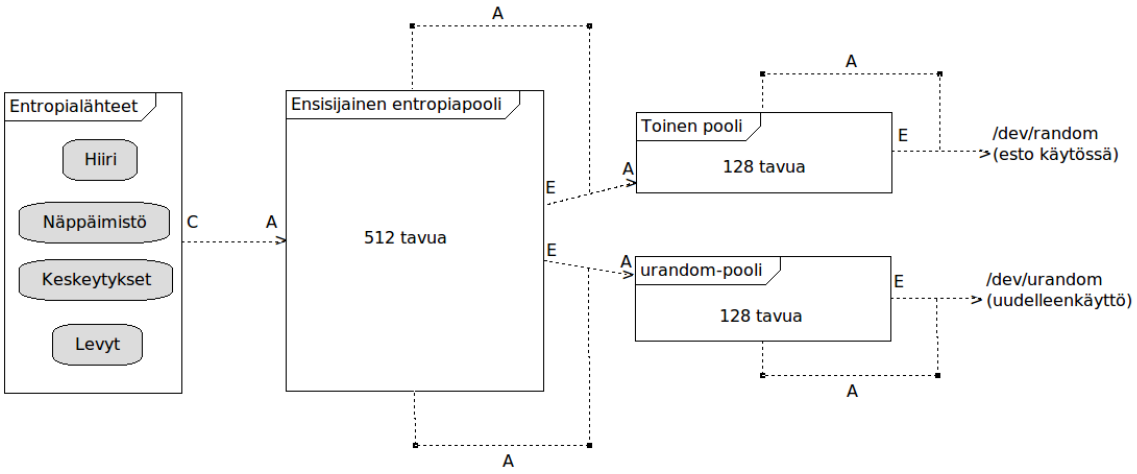
si nimitetään arvoja, jotka saadaan havaituista tapahtumista, joilla on neljä lähdettä: näppäimistö, hiiri, levyjen luku- ja kirjoitustapahtumat sekä järjestelmäkeskeytykset. Nämä entropia-arvot kerätään kahtena 32-bittisenä sanana, joista ensimmäinen mittaa havaitun tapahtuman aikaa (kulunut aika järjestelmän käynnistyksestä) ja toinen sana on sen sijaan tapahtuman arvo. Tapahtuman arvolla tarkoitetaan tässä esimerkiksi näppäinpainalluksen enkoodattua numeroarvoa [16].

LRNG:n toiminta perustuu siihen, että generaattori kerää entropia-arvoja ensimmäiseen entropiapooliin (512 tavua). Tästä poolista arvoja poimitaan sitten toiseen pooliin (128 tavua) tai urandom-pooliin (128-tavua). LRNG:n arvot luetaan sitten edellämainituista toisesta poolista tai urandom-poolista, jotka ovat Linuxissa saatavilla tiedostoista /dev/random ja /dev/urandom [17]. Kaikilla kolmella poolilla on arvio niiden sisältämän entropian määrästä, joka on kokonaislukuarvo väliltä $(0, \text{poolin koko})$. Tämän arvion arvo kasvaa, kun pooliin poimitaan lisää havaintoja ja vastaavasti sen arvo vähenee poimittaessa arvoja poolista.

Erona edellämainituilla kahdella tiedostolla /dev/random ja /dev/urandom, joista satunnaislukuja voidaan lukea, on niiden toiminta riippuen entropian määrästä pooleissa. Näistä /dev/random palauttaa luettaessa ainoastaan maksimissaan entropia-arvion verran tavuja [17]. Jos pyydetty määrä on suurempi kuin toisen poolin sisältämä arvio entropian määrästä, yrittää LRNG poimia entropiaa ensisijaisesta poolista. Jos tämä epäonnistuu, LRNG estää datan lukemisen /dev/randomista ja odottaa kunnes uusia tapahtumia havaitaan ja ensisijaisen entropiapoolin arvio sen sisältämästä entropian määrästä kasvaa [16]. Sen sijaan /dev/urandom käyttää uudestaan jo kerättyä entropiaa datan lukemisen eston sijaan, jolloin siitä voidaan aina lukea haluttu määrä dataa.

5.4 LRNG ja turvallisuusriskit

LRNG:n käytössä on tiettyjä riskejä, jotka liittyvät erityisesti sen alustukseen. LRNG käyttää alustuksessaan, järjestelmän käynnistyessä, pysyviä käyttöjärjestelmäparametrejä, aikaa, levyoperaatioita ja järjestelmätapahtumia. Tämä saattaa johtaa hyvin pieneen entropiamäärään etenkin, jos järjestelmässä ei ole kovalevyjä. Yleisesti ongelma on ratkaistu niin, että Linux-jakeluun on liitetty skripti, joka suljettaessa järjestelmää lukee 512 tavua satunnaista dataa /dev/urandomista ja kirjoittaa ne tiedostoon. Järjestelmää käynnistettäessä nämä 512 tavua luetaan tiedostosta ja kirjoitetaan /dev/urandomiin. Tässä vielä määritellään kirjoittaminen siten, että muokataan ensisijaista entropiapoolia urandom-poolin sijaan [16]. Täten siis LRNG:n turvallisuus riippuu osaltaan kolmannen osapuolen skripteistä, joiden turvallisuus,



Kuva 2: Linuxin satunnaislukugeneraattori [16]. Entropia kerätään eri lähteistä (C) ja lisätään (A) ensisijaiseen entropiapooliin. Entropiaa poimitaan ensisijaisesta poolista joko toiseen pooliin ja urandom-pooliin. Osa poimitusta entropiasta palautuu aina takaisin entropiapooliin, josta se poimittiin. Näistä saadaan sitten /dev/randomin ja /dev/urandomin tuotot (kuva muokattu artikkelista [16]).

eli tässä tapauksessa ennustettavuus, saattaa vaihdella riippuen Linux-jakelusta.

Erityiseksi ongelmaksi Gutterman, Pinkas ja Reinman [16] nostavat tietyt Linux-jakelut, kuten CD-levyltä tai flashmuistilta käynnistettävä Knoppix sekä reitittimiin tarkoitettu OpenWRT-jakelu. Näissä edellämainituissa jakeluissa ei käytetä edellä-kuvatun kaltaista skriptiä. OpenWRT:ssä ainoa entropian lähde on verkon liikenne, joka on helposti tarkkailtavissa, etenkin jos kyseessä on langaton liikenne. Täten mahdollisella hyökkääjällä on hyvin pitkälti kaikki tieto entropiasta, sillä alustettaessa LRNG OpenWRT:ssä, sen tilakin on ennustettava.

6 Satunnaislukugeneraattoreiden tilastollinen testaaminen

Erilaisilla tilastollisilla testeillä voidaan testata satunnaislukugeneraattoreiden satunnaisuusominaisuuksia. Edellä on käsitelty generaattoreita, joilla on hyvin suuri syklin pituus, ja nyt näitä generaattoreita voidaan testata erilaisilla testeillä, joita on käytännössä lukemattoman paljon erilaisia [1]. Testejä tarvitaan erityisesti silloin, kun satunnaislukugeneraattoreita käytetään paikoissa, jossa niiden tuottamilla tuloksilla on kriittinen virka, kuten esimerkiksi kryptografisissa sovelluksissa [9]. Käytännössä tilastolliset testit siis antavat jonkinlaisen mitattavan suureen satunnaisuudelle. Seuraavassa käsitellään esimerkkejä erilaisista testeistä, joita voidaan soveltaa satunnaislukugeneraattorien eri ominaisuuksien testaamiseen. Knuth [1] jakaa satunnaislukugeneraattorien testaamisen kahteen osaan, teoreettisiin testeihin ja empiirisiin testeihin. Näistä kummatkin ovat siinä mielessä valideja, että niillä voidaan havaita puutteita generaattoreissa [20].

6.1 Teoreettiset testit

Satunnaislukugeneraattorien teoreettisilla testeillä tarkoitetaan a priori testejä, eli teoreettisiä tuloksia, jotka kertovat ennen generaattorien käyttöä niiden toimivuudesta [1]. Teoreettiset testit antavat enemmän ymmärrystä eri generaattoreista, kuin pelkästään yritykseen ja erehdykseen pohjautuvat empiiriset testit, joilla voidaan testata, mikäli generaattori täyttää tietyn kriteerin. L'Ecuyer ja Simard kirjoittavatkin testikirjasto TestU01:tä käsittelevässä paperissaan [19], että hyvän satunnaislukugeneraattorin suunnitteluun ei riitä jonkin satunnaisen algoritmin implementointi ja sen testaaminen empiirisesti, vaan se vaatii generaattorin syklin pituuden täsmällistä matemaattista analysointia ja sen tuottamien tulosten yhdenmukaisuuden tarkastelua. Tällaisten testien luominen on haastavaa, ja useat teoreettiset testit tuottavat tuloksia, jotka pätevät tilastollisille testeille, jotka suoritetaan koko generaattorin syklin pituudelle. Kaikilla testeillä nämä eivät toimi oikein, mutta toimiessaan ne paljastavat globaalisti jonon epäsatunnaisuutta, eli vääränlaista toimintaa todella suurilla otoksilla [1]. Teoreettiset testit voivat olla myös ainoa keino saada selville generaattorin syklin pituus, jos se on niin pitkä, ettei sitä voida selvittää empiirisesti testaamalla [20].

Knuth [1] esittää kirjassaan muutamia lauseita, jotka luovat pohjaa satunnaislukugeneraattoreiden teoreettisille testeille, kuten esimerkiksi seuraava:

Lause 1 (Knuth, 1998)**.** *Olk.* a*,* c *ja* m *s.e. ne generoivat LCM:llä jonon, jolla on maksimaalinen syklin pituus. Olk. sitten* $b = a - 1$ *ja* $d = syt(m, b)$*. Tällöin* $P(X_{n+1} < X_n) = \frac{1}{2} + r$*,*

missä $r = \frac{(2(c \bmod d)-d)}{2m}$ *ja täten* $r < \frac{d}{2m}$.

Tästä seuraa se, että käytännössä millä tahansa valinnoilla a ja c saadaan kohtuullinen todennäköisyys sille, että $X_{n+1} < X_n$.

6.2 Empiiriset testit

Empiiriset testit tarkoittavat tilastollisia testejä, joilla voidaan testata generoidun satunnaislukujonon eri ominaisuuksia. Kuten edellä mainittiin, on erilaisia empiirisiä testejä kehitetty lukemattomia erilaisia, joten tässä käydäänkin läpi muutamia tavallisimpia testejä, jotka ovat laajalti käytössä. Monet muut testit perustuvatkin seuraavassa käsiteltäviin χ^2-testiin ja Kolmogorov-Smirnov -testiin [7]. Satunnaislukugeneraattoreiden testauksessa nollahypoteesinä käytetään yleensä sitä, että annetun reaalilukujonon elementit ovat riippumattomasti ja identtisesti jakautuneet kuten tasajakaumassa välillä $(0, 1)$ [7].

χ^2-testissä verrataan havaittujen tapahtumien määrää odotusarvoon, jonka nollahypoteesi antaa. Esimerkiksi [7]: nollahypoteesinä lukujen tasajakautuneisuus välille $(0, 1)$. Otetaan satunnaismuuttujasta 100 realisaatiota. Tämän jälkeen katsotaan mille välille $(0, 0.1], (0.1, 0.2], ..., (0.9, 1[$ kukin niistä osuu. Nyt nollahypoteesin mukaan jokaiselle välille tulisi osua noin 10 lukua. Knuth [1] vie tätä menetelmää pidemmälle ja vertaa havaittujen tapahtumien ja odotusarvojen neliöitä: $V = (x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2$, missä $x_i =$ havaintojen arvot ja $y_i =$ nollahypoteesin antamat oletukset $\forall i = 1, ..., n$. Nyt mitä suurempi V on, sitä kauempana havaitut arvot ovat niiden 0-hypoteesin mukaisista odotusarvoista. Kun vielä jaetaan jokainen summattava aina arvolla y_i, saadaan ns. χ^2-tunnusluku aineistolle. Tämä jakaminen tehdään sen takia, että saadaan kaikille arvoille painotus, joka pohjautuu kyseisen arvon esiintymismääriin, eli päästään eroon ongelmasta, jossa jokaisella arvolla on yhtä suuri painoarvo riippumatta sen esiintymien määrästä, mikä vääristäisi tulosta V. Tämä voidaan vielä esittää matemaattisesti yksinkertaisemmin summalausekkeena:

$$\sum_{i=1}^{n} \frac{(x_i - y_i)^2}{y_i},$$

missä n on eri välien määrä, joille havainnot on jaettu.

Kolmogorov-Smirnov -testi soveltuu tilanteeseen, jossa ei ole mahdollista jakaa havaintoja äärelliseen määrään eri kategorioita, kuten χ^2-testissä. Poimittaessa aineistosta X n kappaletta riippumattomia havaintoja, voidaan muodostaa empiiri-

nen jakaumafunktio

$$F_n(x) = \frac{\#(havainnot\ X_1, X_2, ..., X_n \leq x)}{n}.$$

Merkitään tässä jakaumafunktiota $F(x) = P(X \leq x)$. Nyt Kolmogorov-Smirnov -testi perustuu $F_n(x)$:n ja $F(x)$:n etäisyyteen [7]. Knuth [1] antaa kaksi eri tunnuslukua, K_n^+ ja K_n^-, jotka mittaavat poikkeamia, K_n^+ silloin, kun $F_n > F$ ja K_n^-, kun $F_n < F$.

$$K_n^+ = \sqrt{n}\, max_x(F_n(x) - F(x))$$
$$K_n^- = \sqrt{n}\, max_x(F(x) - F_n(x)).$$

Koska $F_n(x)$ on määritelty äärellisen monen havainnon avulla, voidaan K_n^+ ja K_n^- kirjoittaa muotoon, jossa on läpikäytävänä vain äärellinen määrä vaihtoehtoja:

$$K_n^+ = \sqrt{n}\, max_{j \in 1,...,n} \left(\frac{j}{n} - F(X_j) \right)$$

ja

$$K_n^- = \sqrt{n}\, max_{j \in 1,...,n} \left(F(X_j) - \frac{j-1}{n} \right),$$

missä $1 \leq j \leq n$. Kolmogorov-Smirnov -testillä voidaan myös todeta jonon lukujen olevan tasajakautuneita välille $(0, 1)$ kun $F(x) = x$ ja $0 \leq x \leq 1$ [1].

Runs-testi on Abraham Waldin ja Jacob Wolfowitzin vuonna 1940 esittelemä testi, jolla voidaan havaita epäsatunnaisuutta aineistosta [21]. Tämän testin ideana on mitata aineistosta sellaisten osien määrää tai pituutta, jossa peräkkäisten havaintojen arvot ovat joko nousevia tai laskevia, tästä nimitys run, eli ajo. Tavallinen tapa mitata kasvavan osan pituutta on etsiä pysäytyspiste, jossa arvojen kasvaminen lakkaa. Tästä saadaan sitten kasvavan osan pituus siten, että lasketaan pysäytyspisteiden väliin jäävien osien pituudet [7]. Testissä liian pieni ajojen määrä viittaa matalien ja korkeiden arvojen klusteroitumiseen ja liian suuri ajojen määrä taas näiden vuorottelemiseen [21].

Tunnusluvuksi testissä saadaan $Z = (R - R')/V$, missä R on havaintojen määrä, R' odotusarvo ja V on $\sqrt{\text{varianssi}}$.

Odotusarvo:

$$\frac{2 * N_+ * N_-}{N_+ + N_-} + 1$$

Varianssi:

$$\frac{2 * N_+ * N_- * (2 * N_+ * N_- - (N_+ + N_-))}{(N_+ + N_-)^2 * (N_+ + N_- - 1)},$$

missä N_+ on niiden indeksien lukumäärä, joissa arvot kasvavat ja N_- vastaavasti niiden indeksien lukumäärä, joissa arvot laskevat.

Näiden myötä Z:lle voidaan laskea tilastollinen merkitsevyys, jonka nojalla nollahypoteesi voidaan hyväksyä tai hylätä.

Knuth esittelee kirjassaan myös sarjatestin (engl. serial test), jossa testataan satunnaislukujonossa olevien peräkkäisten lukuparien esiintymistä n-pituisessa jonossa seuraavalla tavalla [1]:

1. Lasketaan kuinka monta kertaa pari (X_{2i}, X_{2i+1}) saa tietyn arvon (x, y), kun $0 \leq i < n$.

2. Suoritetaan sama operaatio kaikille pareille (x, y), missä $0 \leq x, y < d$ ja d voi olla mikä tahansa sopiva luku, kunhan sille voidaan soveltaa χ^2-testiä, jossa eri arvoja on n kappaletta.

3. Sovelletaan χ^2-testiä kaikkiin pareihin, joita on nyt d^2 kappaletta, joilla jokaisella on todennäköisyys $1/d^2$.

4. Jotta χ^2-testi olisi validi, tulee Knuthin mukaan [1] n:n olla suuri verrattuna d^2:een, s.e. vähintään $n \geq 5d^2$.

Tämä sama testi voidaan yleistää myös useamman alkion joukoille kuin ainoastaan pareille. Tällöin joudutaan kuitenkin pienentämään d:n arvoa, jotta vältytään liian suurelta määrältä eri kategorioita χ^2-testissä [1, 7].

6.3 Testikirjastoista

Koska erilaiset satunnaislukugeneraattoreita testaavat tilastolliset testit mittaavat erilaisia poikkeamia eri nollahypoteeseista, tarvitaan testikirjastoja, jotka kattavat erilaisia nollahypoteeseja mahdollisimman hyvin, jolloin saavutetaan mahdollisimman laajasti tieto siitä, onko tietyllä satunnaislukugeneraattorilla tuotettu aineisto satunnaista [7]. Seuraavassa käsitellään muutamia tunnetuimpia testikirjastoja ja niiden sisältämiä testejä.

DIEHARD on yksi tunnetuimmista testikirjastoista vuodelta 1995 ja se sisältää 18 erilaista testiä. Sen laatija on George Marsaglia. DIEHARD sisältää esimerkiksi runs-testin ja minimietäisyystestin, jossa ideana on mitata neliön sisään sijoitettujen pisteiden välistä etäisyyttä. Jos pisteet ovat jakautuneet tasaisesti, kahden pisteen välisen minimietäisyyden neliön tulee olla eksponentiaalisesti jakautunut s.e. odotusarvo on riippuvainen neliön sivun pituudesta ja pisteiden määrästä [22, 7]. P-arvo (engl. p-value) on todennäköisyys saada aikaan vähintään yhtä merkittävä ero tuloksessa käyttäen nollahypoteesia. DIEHARD-testien p-arvot poikkeavat tavallisista tilastollisten testien ilmoittamista p-arvoista, sillä ne ovat kumulatiivisen

jakaumafunktion arvoja, joka tarkoittaa sitä, että testien antamalla pienellä arvolla oletukset pätevät todennäköisesti, kun taas suuri arvo merkitsee pientä p-arvoa tilanteessa, jossa tarkastellaan itse funktiota, eikä sen kertymäfunktiota [7].

NIST eli National Institute of Standards & Technologyn testikirjasto on kehitetty erityisesti kryptografiset sovellukset, kuten salausavainten konstruointi, huomioiden. Myös NIST-kirjaston kaikki 15 testiä perustuvat hypoteesitestaukseen, kuten DIEHARD-testitkin. Monet näistä testeistä ovatkin hyvin samanlaisia kuin DIEHARD-kirjaston testit, kuten esimerkiksi frekvenssitesti, runs-testi ja sarjatesti. Selvittämättömiksi ongelmiksi NIST-testikirjastoon liittyvässä julkaisussa [23] määritellään tilastollisten testien riippumattomuus ja testien kattavuus. Näistä ensimmäisellä tarkoitetaan sitä, aiheutuuko suuremman kuin välttämättömän testimäärän soveltamisesta päällekkäisyyttä, eli ylimäärää. Testien kattavuudella sen sijaan tarkoitetaan ongelmaa, jossa yritetään määritellä kaikki erilaiset tavat osoittaa epäsatunnaisuutta aineistossa [23].

TestU01-kirjasto tarjoaa ANSI C:lle implementoidun testikirjaston, joka sisältää klassisten tilastollisten testien, joita löytyy monista muistakin kirjastoista, lisäksi myös uusia testejä, sekä työkaluja, joilla voidaan tutkia yhteen luokkaan kuuluvien satunnaislukugeneraattoreiden ja jonkin tietyn testin välistä toimintaa [19]. Tällä tarkoitetaan sitä, että tutkitaan kuinka suuri testin otoksen tulee olla verrattuna generaattorin syklin pituuteen, ennen kuin generaattori järjestelmällisesti epäonnistuu testissä. TestU01 myös korjaa muiden testikirjastojen puutteita, kuten sen, että DIEHARDissa testien käyttämät sekvenssit ja parametrit, kuten otosten koot, ovat kiinteitä [19]. Seuraavaan taulukkoon [19] on koottu muutamien käsiteltyjen satunnaislukugeneraattoreiden syklien pituudet (2-kantaisena logaritmina), sekä se, kuinka monen testin p-arvo osui välin $[10^{-10}, 1 - 10^{-10}]$ ulkopuolelle, eli testi selvästi epäonnistui. Näissä viiva (-) tarkoittaa sitä, että generaattori epäonnistui selvästi jo pienemmälläkin testipatterilla. Testeissä käytettiin TestU01-kirjaston eri testipattereita SmallCrush, Crush ja BigCrush.

Generaattori	Syklin pituus	SmallCrush	Crush	BigCrush
Wichmann-Hill	42.7	1	12	22
Unix-random-256	93	1	8	11
LCG(2^{24}, 16598012, 12820162)	24	14	-	-
Java.util.Random	47	1	9	21
KISS99	123	0	0	0
WELL19937a	19937	0	2	2
MT19937	19937	0	2	2

7 Todennäköisyysjakauman generointi havaintoaineistosta

Todennäköisyysjakauman generointiin havaintoaineistosta on kaksi eri mahdollisuutta. Joko aineisto voidaan sovittaa johonkin jo tunnettuun jakaumaperheeseen tai vaihtoehtoisesti aineistosta voidaan generoida sitä kuvaava ns. kokeellinen todennäköisyysjakauma, eli porrasmuotoinen tiheysfunktio havaittujen frekvenssien avulla.

7.1 Aineiston sovittaminen tunnettuun jakaumaperheeseen

Havaintoaineistoa voidaan yrittää sovittaa tunnettuun jakaumaperheeseen ja tämä on toimiva idea etenkin, jos etukäteen on tiedossa hyvin mahdollisia jakaumia, joista aineisto voisi olla peräisin, eli voidaan määritellä jokin äärellinen joukko jakaumia testattavaksi. Aineiston sovittamiseksi jakaumaperheeseen on kaksi yleistä tapaa, pienimmän neliösumman menetelmä ja suurimman uskottavuuden estimointi. Näiden menetelmien ohella voidaan käyttää myös edelläkäsiteltyjä testejä aineiston jakauman testaamiseksi, kuten Kolmogorov-Smirnov -testiä sekä χ^2-testiä. Seuraavassa käydään läpi pääperiaatteet pienimmän neliösumman menetelmästä ja suurimman uskottavuuden estimoinnista.

Pienimmän neliösumman menetelmä (myöh. PNS-menetelmä) on matemaattinen optimointimenetelmä, jossa tavoitteena on löytää aineistolle paras sovite. PNS-menetelmän ideana on minimoida aineiston ja sovitteen pisteiden erotuksen neliöiden summaa [24]. Tämä tapahtuu seuraavalla tavalla:

1. Määritellään sisätulo $\langle u, v\rangle = \sum_{k=1}^{m} u(x_k)v(x_k)$
2. Merkitään havaintopisteitä x_i, missä $i = 1, 2, ..., m$ ja $f(x_i)$ ovat havaintoarvot
3. Käytetään sisätulon määritelmää laskettaessa normia $\|f-p\| = \sqrt{\langle (f-p), (f-p)\rangle}$
4. Tämä normi voidaan kirjoittaa auki seuraavaan muotoon $||f-p|| = \sqrt{\sum_{i=1}^{m}[f(x_k)-p(x_k)]^2}$
5. Etsitään kertoimia kehitelmään $p(x)$, joka on malli, jota sovitetaan aineistoon. Tavoitteena on minimoida $\|f-p\|$. Tässä $p(x)$ voi olla mikä tahansa sovitettava malli, kuten esimerkiksi tietyn jakauman parametrit tai jokin lineaarikombinaatio. Käytetään tässä esimerkissä määrittelyä $p(x) = \sum_{i=1}^{n} \psi_i(x)c_i$.
6. Nyt paras sovite saadaan laskettua sisätulon avulla ortogonaalisuusehtoa käyttäen, eli

7. $\langle f - p, \psi_i \rangle = \sum_{k=1}^{m} [f(x_k) - p(x_k)]\psi_i(x_k) = 0, i = 1, ..., n$

8. Tämä voidaan esittää matriisimuodossa $Ac = f$, missä c on etsittävät kertoimet.

9. Kerroinmatriisi A ja f voidaan esittää myös muodossa $A = B^T B, f = B^T \tilde{f}$, missä $b_{ij} = \psi_j(x_i), \tilde{f}_i = f(x_i)$, eli matriisissa B on i:nnellä rivillä eri muuttujien arvot i:nnessä kokeessa ja j:nnessä sarakkeessa on j:nnen muuttujan arvot kokeissa $1, ..., m$. Jos mallissa on jokin termi on vakio, tällöin sen sarakkeessa kaikki arvot ovat ykkösiä.

Esimerkiksi sovitetaan malli $p = a+bx+cx^2$ pisteistöön $(-3, 3), (0, 1), (2, 1), (4, 3)$. Nyt tulee ratkaista parametrivektori (a, b, c). Edellisestä voidaan ratkaista $c = A/f$.

$$B = \begin{bmatrix} 1 & -3 & 9 \\ 1 & 0 & 0 \\ 1 & 2 & 4 \\ 1 & 4 & 16 \end{bmatrix} \tilde{f} = \begin{bmatrix} 3 \\ 1 \\ 1 \\ 3 \end{bmatrix}$$

$$B^T B = \begin{bmatrix} 4 & 3 & 29 \\ 3 & 29 & 45 \\ 29 & 45 & 353 \end{bmatrix} B^T \tilde{f} = \begin{bmatrix} 8 \\ 5 \\ 79 \end{bmatrix}$$

Ratkaisu on nyt $x = (B^T B)^{-1} B^T \tilde{f}$, eli ratkaisemalla yhtälöryhmä

$$\begin{bmatrix} 4 & 3 & 29 \\ 3 & 29 & 45 \\ 29 & 45 & 353 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 8 \\ 5 \\ 79 \end{bmatrix}$$

saadaan ratkaisuksi arvot $a = 0.8505, b = -0.1925, c = 0.1785$, eli

$$p = 0.8505 - 0.1925x + 0.1785x^2.$$

Tässä esimerkin mallin sovituksessa ei ole kyseessä varsinaisesti jakauman sovittaminen, sillä tuloksena saatu malli $p = 0.8505 - 0.1925x + 0.1785x^2$ ei voi olla aito tiheysfunktio millään koko pisteistön sisältämällä välillä, sillä $\int_{-3}^{4} p(x)\,\mathrm{d}x > 1$.

Suurimman uskottavuuden estimoinnissa ideana on generoida todennäköisyysjakaumalle funktio, joka on mahdollisimman samankaltainen aineiston kanssa, tästä myös englanninkielinen nimi, maximum likehood estimation [25]. Tässä estimoinnissa määritellään uskottavuusfunktio $L(\theta) = c \cdot f(x; \theta)$, missä $f(x; \theta)$ määrittelee havaittavan satunnaismuuttujan X jakauman, joka riippuu jostakin parametrista θ.

Vakio c ei riipu parametrista θ. Vakio c pyritään valitsemaan s.e. L:n lausekkeesta saadaan yksinkertainen. Usein käytetäänkin funktion L logaritmia, $l(\theta) = \log L(\theta)$. Tällöin $l(\theta) = \log c + \log f(x;\theta)$, eli c ei riipu θ:sta. Suurimman uskottavuuden estimaatti $\hat{\theta}$ on parametrin θ arvo, jolla havaitun datan x todennäköisyys on suurin. Samalla se maksimoi funktiot L ja l [26]. Otetaan esimerkiksi normaalijakauma. Merkitään tässä havaintoja x_i:llä, missä $i = 1, ..., n$. Jakaumalla on tiheysfunktio

$$f(x \mid \mu, \sigma^2) = \left(\frac{1}{\sqrt{2\pi}\sigma}\right) \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

Tästä saadaan johdettua n kappaleelle riippumattomia havaintoja tiheysfunktio:

$$f(x_1, ..., x_n \mid \mu, \sigma^2) = \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n \exp\left(-\frac{\sum_{i=1}^n (x_i-\mu)^2}{2\sigma^2}\right).$$

Etsittäessä suurimman uskottavuuden estimaatteja, voidaan funktiosta käyttää sen logaritmia, jolloin myös laskut helpottuvat. Jakauman parametrien uskottavuutta, μ ja σ, voidaan maksimoida samanaikaisesti tai erikseen. Esimerkiksi μ:ta maksimoitaessa, käytettäessä funktion logaritmia, suurin uskottavuus löytyy logaritmin maksimista, kun derivoidaan μ:n suhteen:

$$D \log\left(\left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n \exp\left(-\frac{\sum_{i=1}^n (x_i-\mu)^2}{2\sigma^2}\right)\right) = 0$$

$$D\left(\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n - \frac{\sum_{i=1}^n (x_i-\mu)^2}{2\sigma^2}\right) = 0$$

$$0 - \frac{\sum_{i=1}^n 2\mu - 2x_i}{2\sigma^2} = 0$$

Tälle voidaan ratkaista $\hat{\mu}$, joka maksimoi funktion, tässä tapauksessa

$$\hat{\mu} = \sum_{i=1}^n x_i/n$$

7.2 Jakauman konstruointi havaintojen frekvenssien avulla

Havaintoaineistosta voidaan konstruoida myös ns. kokeellinen todennäköisyysjakauma. Olkoon muuttuja x diskreetti. Tällä tarkoitetaan käytännössä sitä, että luodaan havaintoarvojen frekvensseistä porrasfunktio, eli todennäköisyysjakauman tiheysfunktio, merkitään $f(x)$. Integroimalla tätä tiheysfunktiota saadaan kertymäfunktio $F(x)$, joka kuvaa satunnaismuuttujan X todennäköisyysjakauman yksikäsitteisesti ja se on määritelty kaikille $x \in \mathbb{R}$. Otetaan esimerkiksi seuraavanlaiset havainnot:

Havainto	x_1	x_2	x_3	x_4	x_5
Frekvenssi	7	4	11	2	15
Suhteellinen frekvenssi	$^7/_{39} \approx 0,179$	$^4/_{39} \approx 0,103$	$^{11}/_{39} \approx 0,282$	$^2/_{39} \approx 0,051$	$^{15}/_{39} \approx 0,385$

Näille havainnoille saadaan kertymäfunktio laskemalla $F : \mathbb{R} \to [0, 1]$ asettamalla $F(x) = P(X \leq x)$, eli tällöin

$F(x_1)$	$F(x_2)$	$F(x_3)$	$F(x_4)$	$F(x_5)$
$^7/_{39}$	$^{11}/_{39}$	$^{22}/_{39}$	$^{24}/_{39}$	1

Odotusarvo voidaan laskea todennäköisyyksien summana painottamalla jokaista mahdollista arvoa niiden todennäköisyydellä:

$$\mathbb{E}X = \sum_{i=1}^{n} x_i P(x_i), \text{ missä } n \text{ on havaintojen lukumäärä.}$$

8 Yhteenveto

Satunnaislukujen generointia on käytetty tietotekniikan sovelluksissa jo 1940-luvulta lähtien. Ajan myötä satunnaislukugeneraattoreiden sovelluskohteiden kirjo on kasvanut ja sovelluksista on tullut yhä kompleksisempia ja monissa sovelluksissa tietyt ominaisuudet omaavat satunnaisluvut ovat hyvin kriittisessä roolissa. Tämän tutkielman tavoitteena oli selvittää, kuinka satunnaislukuja voidaan generoida havaitusta aineistosta. Tutkielmassa käytiin läpi, mitä satunnaisluvut ylipäänsä ovat, perusmenetelmät niiden generointiin ja millaisia satunnaislukugeneraattoreita on olemassa. Tämän lisäksi tutkittiin, miten satunnaislukujen generoinnissa voidaan käyttää havaittua aineistoa, esimerkiksi mittausdataa ja kuinka siitä voidaan generoida satunnaislukuja. Tutkielmassa keskityttiin lähinnä näennäissatunnaislukuihin ja jätettiin fysikaalisiin prosesseihin perustuvat generaattorit tutkimuksen ulkopuolelle.

Tutkielman tavoite saavutettiin, sillä nyt on selvästi nähtävillä, mitä satunnaislukujen generoiminen havaintoaineistosta vaatii. Tutkielmassa lähdetään liikkeelle satunnaislukujen generoinnin perusperiaatteista ja päädytään siihen, kuinka havaintodatasta saadaan konstruoitua todennäköisyysjakauma, jota voidaan sitten käyttää satunnaislukuja generoidessa.

Alana satunnaislukujen generointi kehittyy edelleen jatkuvasti, sillä satunnaislukugeneraattorien tuottamien lukujen laatuun kohdistuu jatkuvasti tiukempia vaatimuksia ja uusia parempia algoritmeja kehitetään jatkuvasti. Aihealueella olisikin vielä monia mielenkiintoista tutkimusaiheita, joihin tästä voisi jatkaa. Erityisesti erilaisten uusien satunnaislukujen sovellusalueiden kartoitus ja niiden vaatimat erityisominaisuudet luvuilta voisivat tarjota mielenkiintoisen tutkimuskohteen. Toisaalta myös enemmän matemaattisesta näkökulmasta löytyisi vielä tutkimuskohteita, kuten se, miten algoritmeja voidaan vielä kehittää ja mihin suuntaan. Halutaanko generaattoreilta vain pitkää syklin pituutta vai kasvaako tarve saada niistä entistä nopeampia.

Lähteet

[1] Donald E. Knuth, *The Art Of Computer Programming Vol.2 Seminumerical Algorithms*, Third Edition, Addison-Wesley, 1998.

[2] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, *Numerical Recipes in C, The Art of Scientific Computing*, Second Edition, Cambridge University Press, 1992.

[3] John von Neumann, *Various techniques used in connection with random digits*, Applied Math Series, no. 12, 1951.

[4] The MathWorks, Inc, *Common Generation Methods - MATLAB & Simulink Example*, saatavilla WWW-muodossa <URL: `http://www.mathworks.se/help/toolbox/stats/br5k9hi-1.html`>, viitattu 23.12.2011.

[5] B. A. Wichmann, I. D. Hill, *An Efficient and Portable Pseudo-random Number Generator*, 1981.

[6] M. Matsumoto and T. Nishimura, *Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator*, ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3-30, 1998.

[7] James E. Gentle, *Random Number Generation and Monte Carlo Methods*, Second Edition, Statistics and Computing, Springer, 2003.

[8] S. K. Park, K. W. Miller, *Communications of the ACM, vol. 31*, 1988.

[9] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, James Dray, San Vo, *A Statistical test suite for random and pseudorandom number generators for cryptographic applications*, NIST Special Publication 800-22, 2001.

[10] P. L'Ecuyer, *Efficient and portable combined random number generators*, Communications of ACM, Vol. 31, Number 6, 1988.

[11] P. L'Ecuyer, T. H. Andres, *A Random Number Generator Based on the Combination of Four LCGs*, Mathematics and Computers in Simulation - Special issue: papers presented at the MSSA/IMACS 11th biennial conference on modelling and simulation, Vol. 44, Issue 1, 1997.

[12] Martin Haugh, *Generating Random Variables and Stochastic Processes*, Lecture notes, Monte Carlo Simulation, IEOR E4703, Columbia University, 2010.

[13] C. Bays, S. D. Durham, *Improving a Poor Random Number Generator*, ACM Transactions on Mathematical Software, Vol 2, No 1, March 1976.

[14] Mutsuo Saito, Makoto Matsumoto, *SIMD-oriented Fast Mersenne Twister: a 128-bit Pseudorandom Number Generator*, Monte Carlo and Quasi-Monte Carlo Methods 2006, Springer, 2008.

[15] Francois Panneton, Pierre L'Ecuyer, Makoto Matsumoto, *Improved Long-Period Generators Based on Linear Recurrences Modulo 2*, submitted to ACM TOMS, 2006.

[16] Zvi Gutterman, Benny Pinkas, Tzachy Reinman, *Analysis of the Linux Random Number Generator*, SP '06 Proceedings of the 2006 IEEE Symposium on Security and Privacy.

[17] */dev/random ja /dev/urandom -man-sivut*, saatavilla WWW-muodossa, <URL: `http://kernel.org/doc/man-pages/online/pages/man4/random.4.html`>, viitattu 18.02.2012.

[18] William N. Graham, *A Comparison of Four Pseudo Random Number Generators Implemented in Ada*, ACM SIGSIM Simulation Digest, Volume 22 Issue 2, Fall 1992.

[19] Pierre L'Ecuyer, Richard Simard, *TestU01: A C Library for Empirical Testing of Random Number Generators*, ACM Transactions on Mathematical Software (TOMS), Volume 33 Issue 4, August 2007.

[20] Agner Fog, *Chaotic Random Number Generators with Random Cycle Lengths*, December 2000.

[21] SAS Institute Inc, *Sample 33092: Wald-Wolfowitz (or Runs) test for randomness*, saatavilla WWW-muodossa <URL: `http://support.sas.com/kb/33/092.html`>, viitattu 11.03.2012.

[22] George Marsaglia, *The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness*, saatavilla WWW-muodossa <URL: `http://www.stat.fsu.edu/pub/diehard/`>, viitattu 11.03.2012.

[23] Juan Soto, *Statistical Testing of Random Number Generators*, 22nd National Information Systems Security Conference Proceedings, October 1999.

[24] Raino A. E. Mäkinen, *Numeeriset menetelmät*, luentomoniste, Jyväskylän yliopisto, 2011.

[25] In Jae Myung, *Tutorial on maximum likelihood estimation*, Journal of Mathematical Psychology 47, 2003.

[26] Erkki Liski, *Matemaattinen tilastotiede*, Matematiikan, Tilastotieteen ja Filosofian Laitos, Tampereen Yliopisto, 2004.