UDC: 004.421.5 Info M str. 32-38

STATISTIČKA ANALIZA GENERATORA PSEUDO-SLUČAJNIH BROJEVA
STATISTICAL ANALYSIS OF PSEUDORANDOM NUMBER GENERATORS

Aleksandar Keleč, Nikola Obradović, Igor Dujlović, prof. dr Zoran Mitrović

REZIME: U radu je opisan koncept slučajnih brojeva i njihova višestruka primjena. Opisani su tipovi generatora pseudo-slučajnih brojeva i glavne karakteristike koje treba da ima dobar generator. Pored toga, opisani su mehanizmi i statistički testovi kojima se utvrđuje kvalitet generisanih pseudo-slučajnih sekvenci koje produkuju pojedini algoritmi. Analizirani su testovi koji imaju za cilj da pokažu koliko su generisane sekvence slučajne i uniformne. U okviru praktičnog dijela rada kreiran je testni scenario koji ima za cilj da pokaže koliko su kvalitetne sekvence koje generišu pojedini poznati algoritmi, čije su implementacije date kroz standardne biblioteke programskog jezika Python. Analizirani su algoritmi *uniform*, *randint* i *betavariate* iz standardnih Python biblioteka. Dodatno, u okviru rada je implementiran algoritam *rand_22* za koji je pokazano da generiše sekvence koje prate uniformnu raspodjelu, te se može ravnopravno koristiti u primjenama koje zahtijevaju dugačke sekvence pseudo-slučajnih brojeva sa uniformnom raspodjelom. U okviru testnog scenarija su provedeni i dokumentovani sljedeći testovi: *Chi-square* test, Grafički test, Poker test i *Runs* test. Grafički test je na vizuelan način pokazao da algoritmi *uniform*, *randint* i *rand_22* generišu uniformo raspoređene slučajne brojeve, dok se to ne može reći za algoritam *betavariate*. Sa druge strane, ostali testovi su kroz numeričke rezultate potvrdili ovu činjenicu.

KLJUČNE REČI: slučajni brojevi, pseudo-slučajni brojevi, generatori pseudo-slučajnih brojeva, statistički testovi, uniformnost, slučajnost

ABSTRACT: The paper describes the concept of random numbers and their multiple applications. The types of pseudorandom number generators are described as well as the main characteristics that a good generator should have. In addition, the mechanisms and statistical tests that determine the quality of generated pseudorandom sequences produced by individual algorithms are described. Furthermore, the paper deals with statistical tests that show how random and uniform the generated sequences are. As the practical work of the paper, a test scenario was created that aims to show the quality of the sequences generated by several known algorithms, whose implementations are given by standard Python libraries. The algorithms *uniform*, *randint*, and *betavariate* from standard Python libraries were analyzed. Additionally, the algorithm *rand_22* is implemented, which has been proven to generate sequences that follow a uniform distribution and can be used equally in applications that require long sequences of pseudorandom numbers with a uniform distribution. The following tests were conducted and documented within the test scenario: Chi-square test, Graphic test, Poker test, and Runs test. The graphical test visually showed that the *uniform*, *randint*, and *rand_22* generate uniformly distributed random numbers, while the same cannot be said for the *betavariate* algorithm. On the other hand, other tests confirmed this fact through numerical results.

KEY WORDS: Random numbers, Pseudorandom numbers, Pseudorandom number generators, Statistical tests, Uniformity, Randomness

1. UVOD

Slučajni brojevi (eng. *random numbers*) predstavljaju važan segment u različitim primjenama, kao i oblastima ljudskog djelovanja. Unazad nekoliko decenija, slučajni brojevi se koriste u okviru raznih tehnika statističkog uzorkovanja, pri čemu se oni generišu na različite načine i iz različitih izvora. Brojne su oblasti u kojima primjenu nalaze slučajni brojevi, poput kriptografije, igara na sreću, statistike, računarske simulacije, numeričke analize i sl [1].

Slučajni brojevi su u početku generisani na osnovu određenih fizičkih procesa, kao što je, na primjer, bacanje kocke ili novčića, kao i na osnovu tabela sa nasumičnim brojevima, dok se danas koriste računari za njihovo generisanje. Računarski algoritmi koji se koriste za generisanje slučajnih brojeva su zapravo deterministički, iako sekvenca koju generišu djeluje potpuno slučajna, bez bilo kakvih pravilnih ili ponavljajućih šablona koji bi se mogli uočiti. Zbog toga, ovako generisani slučajni brojevi se zovu, preciznije, pseudo-slučajni brojevi (eng. *pseudo-random numbers*) [1].

Sa razvojem modernih računarskih sistema sa velikom računarskom moći, povećao se interes za slučajnim brojevima, načinima njihovog generisanja, kao i različitim testovima koji imaju za cilj da utvrde kvalitet takvih brojeva i njihovih generatora [2].

U radu su analizirane i testirane postojeće implementacije algoritama za koje se u zvaničnoj dokumentaciji navodi da generišu uniformne sekvence, kao i implementacija algoritma koji generiše sekvence koje ne prate uniformnu raspodjelu. Cilj je da se za pomenute implementacije PRNG algoritama pokaže da li zaista generišu sekvence onako kako je to opisano u njihovoj zvaničnoj dokumentaciji i sa kojim nivoom povjerenja. Dodatno, u okviru rada je implementiran algoritam *rand_22* koji je takođe učestvovao u testnom scenariju sa ciljem da se utvrdi da li generiše sekvence koje prate uniformnu raspodjelu i da li se može ravnopravno koristiti u primjenama koje zahtijevaju velike sekvence pseudo-slučajnih brojeva sa uniformnom raspodjelom.

U drugom poglavlju opisani su tipovi generatora pseudo-slučajnih brojeva, poput kongruentnih, Fibonačijevih, *Mersenne Twister* i drugih. Opisan je koncept slučajnosti i date su glavne karakteristike koje treba da ima dobar generator pseudo-slučajnih brojeva.

U trećem poglavlju objašnjeno je na koji način se može utvrditi koliko su dobri pojedini generatori pseudo-slučajnih brojeva, odnosno, koliko kvalitetne sekvence generišu. Opisani su najčešći testovi koji se primjenjuju u ovakvim situacijama i koji pokazuju koliko su generisane sekvence slučajne i uniformne.

Četvrto poglavlje prikazuje rezultate nekoliko različitih statističkih testova koji su primijenjeni na četiri generatora pseudo-slučajnih brojeva. Korišteni su sljedeći testovi: *Chi-square* test, Grafički test, Poker test i *Runs* test, sa ciljem da se pokaže koliko su uniformne i slučajne sekvence koje generišu pojedini generatori, odnosno, koliki je stepen korelacije između elemenata sekvence.

U posljednjem poglavlju data je kratka rekapitulacija istraživanja i zaključna razmatranja, te su date smjernice za buduća istraživanja.

2. GENERATORI SLUČAJNIH BROJEVA

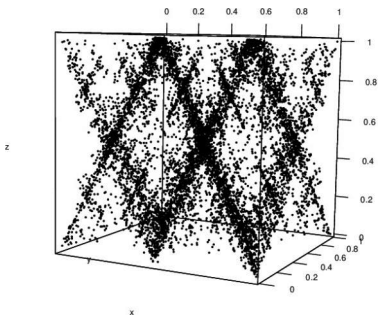
Koncept slučajnosti (eng. *randomness*) nije jednostavno objasniti kako to djeluje na prvi pogled. Kao primjer fizičkih procesa za koje se smatra da su slučajni, često se uzima bacanje novčića ili kocke. Međutim, ovi procesi se u određenoj mjeri mogu smatrati determinističkim, ako se poznaju jednačine koje opisuju kretanje novčića ili kocke i ako se poznaju određeni inicijalni uslovi [1].

Računarski algoritmi za generisanje slučajnih brojeva predstavljaju kratak opis sekvence koju produkuju, pa se zbog toga, po definiciji, ne mogu smatrati apsolutno slučajnim (eng. *truly random*), već pseudo-slučajnim, pa se i sekvenca koju generišu naziva pseudo-slučajna sekvenca. U skladu s tim, izveden je i termin „generator pseudo-slučajnih brojeva“ (eng. *Pseudo-Random Number Generator* - PRNG). Računarski algoritmi (PRNG) generišu pseudo-slučajne sekvence koje se u određenoj mjeri mogu smatrati za predvidljive i reproduktivne, što olakšava proces praćenja i debagovanja računarskih simulacija, kao i verifikaciju rezultata. Sa druge strane, kako je računarska memorija diskretne prirode i ograničenog kapaciteta, ona može da čuva konačan skup brojeva, pa se i zbog toga svaka sekvenca mora eventualno ponoviti [1].

Osobine koje karakterišu dobar generator slučajnih brojeva su sljedeće [1]:

- Slučajna sekvenca (eng. *random pattern*). Generator treba da prođe testove vezane za slučajnost.
- Dug period (eng. *long period*). Generator treba da generiše najdužu moguću sekvencu koja se ne ponavlja.
- Efikasnost (eng. *efficiency*). Generator treba da bude brz i zahtijeva što manje memorijskih resursa, s obzirom da većina računarskih simulacija radi sa milionima slučajnih brojeva.
- Ponovljivost (eng. *repeatability*). Generator treba da produkuje uvijek istu sekvencu, ako se generisanje započne pod istim inicijalnim uslovima.
- Prenosivost (eng. *portability*). Generator treba da se može izvršavati na različitim tipovima računara i da je sposoban da na svakom produkuje iste sekvence.

Veoma je teško zadovoljiti sve navedene karakteristike u okviru jednog generatora slučajnih brojeva. Na primjer, veliki broj generatora produkuje sekvence kod kojih se može identifikovati visok stepen korelacije, što znači da kod takvih generatora preovladava odsustvo slučajnosti [3]. Korelacija između slučajnih sekvenci se najlakše utvrđuje testovima grafičkog prikaza tačaka u 3D ravni, kao na slici 1. Dakle, ukoliko se tačke koncentrišu na nekoliko zajedničkih ravni, umjesto nasumičnog rasporeda u 3D prostoru, to je pokazatelj da dati PRNG pada na testu slučajnosti.



Slika 1 – *Korelacija slučajnih sekvenci kod nekih PRNG algoritama [4]*

Svi PRNG algoritmi se mogu opisati pomoću jednačine (1)[5]. Ideja koja stoji iza ovih algoritama bazirana je na generisanju sekvence brojeva x_1, x_2, x_3, … pomoću rekurzivne funkcije, oblika:

$$x_i = f(x_{i-1}, x_{i-2}, \dots, x_{i-n}), \quad (1)$$

pri čemu je potrebno n inicijalnih brojeva za započinjanje rekurzije. Inicijalni brojevi se nazivaju i „sjeme“ algoritma (eng. *seed*). Razlika između pojedinih PRNG algoritama se ogleda u izboru funkcije f koja je zadužena da generiše slučajne brojeve.

2.1. *Kongruentni generatori*

Kongruentni generatori predstavljeni su sljedećom jednačinom:

$$x_{i-1} = (ax_i + c) \bmod m, \quad (2)$$

pri čemu je *seed* vrijednost, veliki cijeli broj koji određuje period generatora, što znači da generator produkuje vrijednosti između 0 i $m - 1$. Parametar a ($0 \le a < m$) se naziva multiplikator, a parametar c ($0 \le c < m$) je inkrement. Vrijednosti a, c, x_0 i m u značajnoj mjeri utiču na ponašanje i kvalitet generatora [5].

Kongruentni generatori su veoma brzi i koriste minimalne memorijske resurse. Međutim, njihov period je ograničen i zavisi od izbora vrijednosti m. Za standardne linearne kongruentne generatore, važi da je $m \sim 2^{32}$, što znači da oni mogu da generišu oko 10^9 slučajnih brojeva. Na današnjim računarima, takve slučajne sekvence se „potroše“ za nekoliko sekundi. Iz tog razloga, preporuka je da se ovakvi generatori nikada ne koriste za potrebe numeričkih simulacija. Sa druge strane, oni su prihvatljivi za generisanje *seed* sekvenci za druge PRNG algoritme. Još jedan nedostatak ovih algoritama je taj što ih je veoma teško paralelizovati [5].

2.2. *FIBONAČIJEVI GENERATORI*

Fibonačijevi generatori generišu realne (eng. *floating-point*) brojeve na intervalu [0,1), pri čemu se za generisanje neke

vrijednosti koristi razlika, suma ili proizvod prethodnih vrijednosti. Tipičan primjer Fibonačijevog generatora dat je u sljedećoj jednačini [1]:

$$x_k = x_{k-17} - x_{k-15} \quad (3)$$

Fibonačijevi generatori su zamišljeni kao poboljšanje kongruentnih generatora i većina ovih generatora prolazi standardne empirijske testove generatora slučajnih brojeva. Zahtijevaju više memorijskih resursa u odnosu na kongruentne generatore, što, uz današnje računarske sisteme, ne predstavlja poseban nedostatak. Osim toga, ovi generatori su veoma brzi, imaju veoma dug period i mogu se paralelizovati na multiprocesorskim računarskim sistemima [5].

2.3. *Mersenne Twister generator*

Mersenne Twister generator je razvijen 1997. godine i predstavlja najkorišteniji PRNG algoritam opšte namjene [6]. Period algoritma je dat pomoću *Mersenne*-ovog prostog broja $M_n = 2^{n-1}$, $n \in N$. Implementacija ovog algoritma, MT19937, koja je dio većine programskih jezika i biblioteka (Python, MatLab, R, C, Boost), ima period $\rho = 2^{19937} - 1 \approx 10^{6001}$. Algoritam je veoma brz i generiše vrlo kvalitetne slučajne sekvence [7].

2.4. *OSTALI GENERATORI*

Od ostalih PRNG algoritama, važno je spomenuti WELL generatore, kao i *ran* i *drand48* familije algoritama.

WELL (eng. *Well Equidistributed Long-period Linear*) generatori su razvijeni s ciljem da obezbijede bolju uniformnost (eng. *equidistribution*), kao i osobinu miksanja bita (eng. *bit-mixing*), zadržavajući ekvivalentnu dužinu perioda i brzinu rada kao kod *Mersenne Twister* generatora [9].

Familija algoritama *ran* sastoji se od algoritama *ran0*, *ran1* i *ran2*, pri čemu se prva dva ne preporučuju za namjenu generisanja slučajnih brojeva, jer ne prolaze sve statističke testove i imaju kratak period (2^{32}). Sa druge strane, *ran2* ima duži period (10^{18}) i zadovoljava sve statističke testove. Međutim, *ran2* je prilično spor i zbog toga ga treba koristiti samo za generisanje *seed* vrijednosti za druge algoritme [5,8].

Unix-ova familija generatora, *drand48*, je bazirana na linearnom kongruentnom generatoru sa 48-bitnom numeričkom aritmetikom. Slučajni brojevi se generišu na osnovu jednačine (2), pri čemu je $a = 25214903917$, $c = 11$ i $m = 2^{48}$. Međutim, zbog poznatih nedostataka kongruentnih generatora, *drand48* nije preporučljivo koristiti za potrebe generisanja kvalitetnih pseudo-slučajnih sekvenci [5].

3. TESTIRANJE PRNG ALGORITAMA

S obzirom da su PRNG algoritmi deterministički, postavlja se pitanje da li su, i u kojoj mjeri, sekvence koje oni generišu slučajne. Osim toga, za pojedine PRNG algoritme poželjno je utvrditi stepen uniformnosti slučajnih sekvenci koje oni proizvode. Da bi se ustanovio kvalitet pojedinih PRNG algoritama, koriste se statistički testovi. Danas postoji na desetine statističkih testova razvijenih sa ciljem da pokažu da li PRNG algoritmi daju uniformne i slučajne sekvence. Treba imati u vidu da nijedan test nije univerzalan i da činjenica da je neki PRNG algoritam zadovoljio neki test ne znači nužno da je taj algoritam dovoljno dobar, ali se povećava povjerenje. Zbog toga se, prilikom analize PRNG algoritama, oni obično propuštaju kroz više različitih testova, kombinovanih u testne pakete (eng. *test suits*), o kojima će biti riječi u ovom poglavlju. Osim toga, ukoliko jedna generisana sekvenca zadovolji određeni test, to ne znači da će ga i naredna sekvenca, generisana od strane istog PRNG generatora, zadovoljiti. U nastavku su opisani neki od poznatijih statističkih testova [1,2,5,10, 19,20,21,22].

3.1. *Chi-square* test

Chi-square test (χ^2) jedan od najpoznatijih statističkih testova koji predstavlja osnovu za sve empirijske testove i koristi se u kombinaciji sa mnogim drugim testovima. U osnovi, ovaj test se koristi da za neki rezultat eksperimenta kaže koliko je on vjerovatan [10].

Chi-square test treba da pokaže da li sekvenca koju je generisao neki PRNG algoritam prati uniformnu raspodjelu. Ovaj test je baziran na uspostavljanju tzv. *null* hipoteze (eng. *null hypothesis,* H_0), koja se onda prihvata ili odbacuje sa određenim nivoom povjerenja (eng. *confidence level*), u zavisnosti od rezultata testa i korištenih parametara. U slučaju PRNG algoritama, H_0 glasi ovako: „generisana sekvenca slučajnih brojeva prati uniformnu raspodjelu“. Osnovna ideja je da se porede uočene frekvencije slučajnih brojeva sa očekivanim frekvencijama.

Test se u slučaju PRNG algoritma provodi na sljedeći način:

- generiše se k sekvenci slučajnih brojeva,
- identifikuju se očekivane (e) i observirane (o) frekvencije za svaku sekvencu.

Nakon toga se izračuna χ^2 distribucija, po formuli:

$$D = \sum_{i=1}^{k} \frac{(o_i - e_i)^2}{e_i} \quad (4)$$

D predstavlja varijablu čija distribucija aproksimira χ^2 distribuciju sa stepenom slobode $k - 1$. Ako je $D < \chi^2_{1-\alpha;\, k-1}$ to znači da se hipoteza H_0 može prihvatiti sa nivoom povjerenja α, odnosno, α predstavlja vjerovatnoću da je donesena pogrešna odluka prihvatanjem hipoteze H_0. Za α se najčešće uzimaju vrijednosti 0.01, 0.05 i 0.1 (1%, 5% i 10%) [10]. Vrijednost $\chi^2_{1-\alpha;\, k-1}$ se čita iz tabele kritičnih vrijednosti *Chi-square* raspodjele, koja je data u [11].

3.2. *Frekvencijski test*

Frekvencijski test (eng. *Frequency test*) je prilično jednostavan test koji utvrđuje da li PRNG algoritam brojeve iz sekvence distribuira uniformno na jediničnom intervalu. Test

se provodi tako što se izabrani interval podijeli na K pod-intervala i onda se generiše N slučajnih brojeva, pri čemu se računa koliko se puta neki broj pojavio u svakom od pod-intervala. Dalje se, pomoću *Chi-square* testa, utvrđuje da li dati PRNG algoritam generiše uniformne sekvence.

3.3. *Serijski test*

Serijski test (eng. *Serial test*) predstavlja „višedimenzionalnu“ verziju frekvencijskog testa. Ovim testom se utvrđuje da li su parovi susjednih brojeva (ili trojke, četvorke, ... n-torke) uniformno raspoređeni na izabranom intervalu. Da bi se ovo odredilo, potrebno je izbrojati koliko puta se par $(Y_{2j}, Y_{(2j+1)}) = (q, r)$ pojavljuje, za svako $0 \leq j < n$ (n je dužina slučajne sekvence). Ovo treba uraditi za svaki par brojeva (q, r), pri čemu je $0 \leq q, r < d$ (moguće vrijednosti slučajnih brojeva su u skupu [1, d]). To znači da u tom slučaju postoji d^2 kategorija za χ^2 test, pri čemu svaka kategorija ima pridruženu vjerovatnoću od $1/ d^2$ [10].

3.4. *Test raspona*

Test raspona (eng. *Gap test*) podrazumijeva da se za svako U_j koje pripada nekom intervalu, odredi dužina „raspona“ između ovog broja i sljedećeg broja koji dolazi u sekvenci, a pripada tom istom intervalu. Drugim riječima, neka su α i β dva broja takva da važi $0 \leq \alpha < \beta \leq 1$. Treba naći dužinu svih sekvenci $U_j, U_{j+1}, \ldots, U_{j+r}, U_{j+r+1}$, takvih da se U_j i U_{j+r+1} nalaze između α i β, ali se ostali brojevi u sekvenci ne nalaze. Nakon toga se primijeni χ^2 test na dobijene rezultate, koristeći različite dužine „raspona“ kao kategorije, sa sljedećim vjerovatnoćama:

$$p_0 = p,\ p_1 = p(1-p),\ p_2 = p(1-p)^2,\ \ldots,\ p_k = p(1-p)^k,\ldots \quad (3)$$

pri čemu je $p = \beta - \alpha$, vjerovatnoća da se neki broj U_j nalazi između α i β [10, 12].

3.5. *Poker test*

Poker test (eng. *The Poker test*) podrazumijeva da se posmatra n grupa koje se sastoje od po pet uzastopnih brojeva , pri čemu je . Svaka od ovih grupa se smjesti u jednu od sedam kategorija (imenovanih po terminima iz kartaške igre poker):

- Sve različite: *abcde*
- Jedan par: *aabcd*
- Dva para: *aabbc*
- Tri iste: *aaabc*
- *Full house*: *aaabb*
- Četiri iste: *aaaab*
- Pet istih: *aaaaa*

Nakon toga se primijeni χ^2 test na dobijene rezultate, pri čemu se koristi ovih sedam kategorija [10, 12].

3.6. *Grafički testovi*

Poznati su i kao spektralni testovi (eng. *Spectral tests*). Kao rezultat daju grafički prikaz prostornih korelacija između slučajnih brojeva grupisanih u k-torke. Cilj im je da pokažu koliko gusto će k-torke popuniti k-dimenzioni hiperprostor. Ovi testovi se primjenjuju uglavnom na linearne kongruentne generatore pseudo-slučajnih brojeva. U [3] je pokazano da se k-torke koje generišu ovi algoritmi na grafičkom prikazu nalaze na konačnom broju paralelnih hiperravni. To znači da, ako se posmatra 2D prostor, susjedni parovi će se nalaziti na pravim linijama, dok će u 3D prostoru trojke obuhvatati konačne ravni tog prostora. Ideja je da se izmjeri maksimalno rastojanje između susjednih hiperravni i, što je rastojanje veće, to je generator lošiji. Drugim riječima, generator je bolji ako se na grafičkom prikazu ne uočavaju nikakve korelacije između iscrtanih tačaka, tj. prikaz tačaka je „haotičan“. Kod lošijih generatora, na grafičkom prikazu su vidljive određene korelacije između tačaka (slika 1).

3.7. *Runs test*

Runs test se obično izvodi na nivou sekvence bita, pri čemu se posmatraju segmenti koji se sastoje od istih bita. Drugim riječima, *run* je sekvenca od minimalno dva ista bita (0 ili 1) kojoj prethodi i koju prati bit koji je različit od bita sekvence. Na primjer, ako je data sekvenca: ε = 110001111, onda se mogu identifikovati 3 *run*-a: 11, 000 i 1111. Ovaj test ima za cilj da utvrdi frekvenciju *run* sekvenci u posmatranoj pseudo-slučajnoj sekvenci bita i pokaže da li se biti izmjenjuju prebrzo ili presporo u odnosu na potpuno slučajnu (*truly random*) sekvencu bita. U slučaju kada se posmatra sekvenca brojeva, ovaj test se izvodi tako što se posmatraju tzv. monotone sekvence. Monotone sekvence se sastoje od rastućih ili opadajućih brojeva, te se u ovom testu posmatraju dužine ovih sekvenci. Međutim, za razliku od većine prethodnih testova, ove dužine nisu međusobno nezavisne, pa zbog toga ne mogu da se koriste kao kategorije za χ^2 test. Dokaz međusobne zavisnosti proizilazi iz činjenice da su duže sekvence najčešće praćene kraćim sekvencama i obrnuto. Zbog toga se u okviru ovog testa, umjesto *Chi-square* analize koriste komplikovaniji statistički testovi, opisani u [10]. Jedan od statističkih testova koji se koriste u ovom slučaju je tzv. Z-statistika (eng. *Z-statistic*), kod koje se računa Z-vrijednost (eng. *Z-score*) i poredi sa kritičnom vrijednosti Z (eng. *Critical Value of Z*) i onda se hipoteza H_0 prihvata ili odbacuje sa određenim nivoom povjerenja [13].

3.8. *Testni paketi*

Osim opisanih, postoji još nekoliko testova koji se koriste kod statističke analize PRNG algoritama, kao što su: *Coupon Collector's Test*, *Permutation Test*, *Maximum-of-t Test*, *Kolmogorov-Smirnov Test*, *Serial-correlation Test* i dr.

Da bi se izvršila sveobuhvatna analiza i detaljno testiranje PRNG algoritma, često se ovakvi testovi kombinuju u tzv. testne pakete (*test suite*), od kojih su najpoznatiji DIEHARD [14], NIST *test suite* [15], L'Ecuyer's *test suite* [16], SPRNG *test suite* [10] i sl.

4. TESTNI SCENARIO

U ovom poglavlju opisan je proces testiranja nekoliko različitih PRNG algoritama, implementiranih u vidu funkcija u programskom jeziku Python, pri čemu su korišteni neki od testova iz trećeg poglavlja, konkretno *Chi-square* test, Grafički test, Poker test i *Runs* test. Navedeni testovi su izabrani kako bi se na vizuelan način, ali i numerički, pokazalo kakva je raspodjela pseudo-slučajnih vrijednosti koje generišu izabrani PRNG algoritmi i koliko su slučajne sekvence koje oni generišu.

U testnom scenariju su korišteni sljedeći PRNG algoritmi, tj. njihove praktične implementacije:

- Funkcija *uniform*[1] iz paketa *numpy.random* programskog jezika Python, koja generiše slučajne vrijednosti koje su uniformno raspoređene na nekom intervalu,
- Funkcija *randint*[2] iz paketa *numpy.random* programskog jezika Python, koja generiše slučajne vrijednosti diskretne uniformne raspodjele iz nekog intervala,
- Funkcija *betavariate*[3] iz paketa *random* programskog jezika Python, koja generiše slučajne vrijednosti na osnovu *beta*[4] distribucije,
- Funkcija *rand_22* definisana u okviru ovog rada čiji je kod u programskom jeziku Python prikazan na slici 2.

```
cons = 0xFAACBFCC
def rand_22():
    global cons
    x = hex(round(time.time() * 1000))
    y=int(x, base=16)
    cons = (cons * y) % 0x12AB4671
    cons = cons ^ 0xFF6509
    cons = (cons*0xAB) % 4294967296
    cons = cons ^ 0xF65F42CB
    cons = cons | 0x1234567890
    cons = ((cons + 14351514) * 32) % 7777333
    return cons % 10
```

Slika 2 – *Implementacija funkcije rand_22 u programskom jeziku Python*

Chi-square test

Chi-square test je izabran za ovaj testni scenario, jer je on neizostavan statistički test koji pokazuje da li sekvenca koju je generisao neki PRNG algoritam prati uniformnu raspodjelu. Za potrebe ovog rada korištena je funkcija *chisquare*[5] iz paketa *stats.scipy*, programskog jezika Python. U ovom testu, kao i u ostalim testovima iz ovog testnog scenarija, generisano je hiljadu pseudo-slučajnih brojeva u intervalu [0,9], koji su podijeljeni u deset grupa. U ovom testu je izabrano $\alpha = 0.05$, pa je $\chi^2_{0.95;9} = 16.92$, prema [11]. U tabeli 1 prikazani su rezultati provedenog testa.

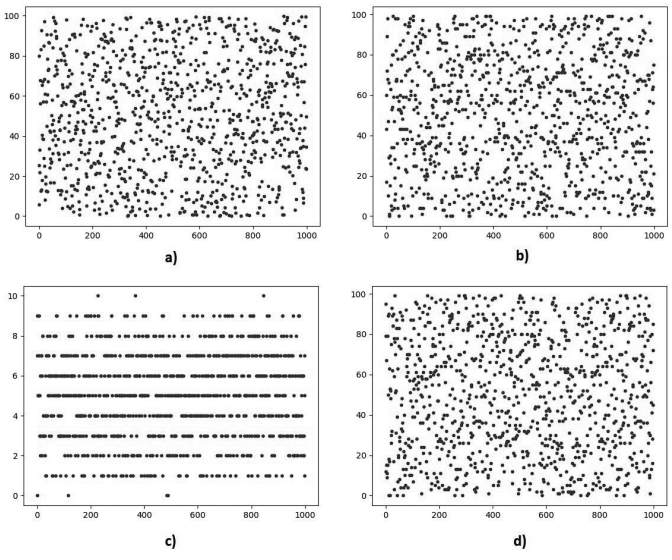
Funkcija	*Chi-square* vrijednost	$H_0(\alpha=0.05)$
uniform	13.5	Prihvaćena
randint	7.26	Prihvaćena
betavariate	180.32	Odbačena
rand_22	16.42	Prihvaćena

Tabela 1 – *Rezultati Chi-square testa*

Rezultati *Chi-square* testa pokazuju da se sa nivoom povjerenja od 95% može reći da algoritmi *uniform*, *randint* i *rand_22* generišu uniformo raspoređene slučajne brojeve, dok to ne važi za algoritam *betavariate*.

4.2. Grafički test

Grafički test je urađen kako bi se na vizuelan način prikazala raspodjela pseudo-slučajnih vrijednosti koje su generisali izabrani PRNG algoritmi. Na ovaj način se može odmah vidjeti da li postoje određene korelacije između generisanih brojeva, odnosno, da li PRNG algoritme karakteriše osobina slučajnosti. Na slici 3 jasno se može vidjeti visok stepen korelacije kod funkcije *betavariate* (c), dok se kod funkcija *uniform* (a), *randint* (b) i *rand_22* (d) ne primjećuje nikakva korelacija između generisanih brojeva, što znači da ovi algoritmi prolaze ovaj test i prema njemu daju uniformo raspoređene pseudo-slučajne brojeve.



Slika 3 – *Rezultati Grafičkog testa*

4.3. *Poker test*

Poker test je veoma pogodan u situacijama kada se želi pokazati da li se sekvenca brojeva može smatrati slučajnom ili ne. U okviru ovog rada iskorištena je implementacija Poker testa u programskom jeziku Python, data u [17]. Ova funkcija

1 https://numpy.org/doc/stable/reference/random/generated/numpy.random.uniform.html

2 https://numpy.org/doc/stable/reference/random/generated/numpy.random.randint.html

3 https://docs.python.org/3/library/random.html

4 *Beta* distribucija predstavlja kontinualnu raspodjelu vjerovatnoća na intervalu [0,1], koja zavisi od parametara α i β. U ovom radu su korištene vrijednosti $\alpha = \beta = 2$, kao i transliranje dobijenih vrijednosti na interval [0, 9] i njihovo zaokruživanje na najbliži cijeli broj.

5 https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.chisquare.html

prihvata dva argumenta, binarnu sekvencu brojeva i cjelobrojnu vrijednost koja označava stepen slobode koji se koristi kod χ^2 testa. U ovom slučaju, hipoteza H_0 se definiše na sljedeći način: $p_0 = p_1$, pri čemu je p_0 vjerovatnoća da je sljedeća cifra u sekvenci 0, dok je p_1 vjerovatnoća da je sljedeća cifra 1. Stoga, H_0 je postavljena na veoma logičan način, jer je očekivano da se kod slučajno generisane binarne sekvence pojavi 50% nula i 50% jedinica. Kako bi se ovakva implementacija Poker testa mogla primijeniti na izlaze algoritama koji se testiraju u ovom radu, ti izlazi su dodatno konvertovani u binarni brojni sistem. S obzirom na to da korištena funkcija vraća binarnu odluku, tj. da li je algoritam prošao Poker test ili ne, svaki od algoritama je podvrgnut ovom testu hiljadu puta, te je određen procenat uspješnih prolazaka testa. U tabeli 2 prikazani su rezultati testa. Rezultati Poker testa pokazuju da se sa značajnim procentom uspješnosti može reći da algoritmi *uniform*, *randint* i *rand_22* generišu sekvence koje se mogu smatrati slučajnim, dok algoritam *betavariate* očigledno generiše sekvence koje nisu slučajne.

Funkcija	Uspješnih prolazaka	Neuspješnih prolazaka	Procenat uspješnosti
uniform	836	164	83,6%
randint	932	68	93,2%
betavariate	35	965	0,35%
rand_22	726	274	72,6%

Tabela 2 – *Rezultati Poker testa za različite ulazne sekvence*

4.4. *Runs test*

Runs test je također pogodan za utvrđivanje stepena slučajnosti sekvence. Test, osim sekvence, ne uključuje dodatne ulazne parametre, a kao rezultat daje Z-vrijednost, odnosno vrijednost koja pokazuje koliko neka vrijednost odstupa od srednje vrijednosti kod normalne raspodjele. Nakon toga se pronalazi odgovarajuća kritična vrijednost Z u tabeli [13] za određeni nivo povjerenja i ta vrijednost se poredi sa dobijenom Z-vrijednosti. Ako je Z-vrijednost veća od kritične vrijednosti, onda se hipoteza H_0 odbacuje za izabrani nivo povjerenja. U suprotnom, hipoteza se prihvata.

Funkcija	Z-vrijednost	H_0 (α=0.05)
uniform	1.36	Prihvaćena
randint	0.87	Prihvaćena
betavariate	2.33	Odbačena
rand_22	0.75	Prihvaćena

Tabela 3 – *Rezultati Runs testa*

U tabeli 3 prikazani su rezultati *Runs* testa provedenog nad izabranim PRNG algoritmima. Iskorištena je implementacija *Runs* testa data u [18]. U ovom testu je izabrano $\alpha = 0.05$, pa kritična vrijednost Z iznosi 1.65 [13]. Iz tabele 3 se vidi da svi algoritmi, osim *betavariate*, prolaze *Runs* test, odnosno, za njih se može tvrditi da generišu slučajne sekvence, sa nivoom povjerenja od 95%.

5. ZAKLJUČAK

U radu je opisan koncept slučajnosti i objašnjena je osnovna razlika između generatora slučajnih i pseudo-slučajnih sekvenci. S obzirom na to da su računarski algoritmi koji se koriste za generisanje slučajnih brojeva deterministički, sekvence koje oni generišu su zapravo pseudo-slučajne. S tim u vezi, ovaj rad se bavi analizom kvaliteta pojedinih PRNG algoritama, pomoću različitih statističkih testova koji se koriste u ove svrhe, a imaju za cilj da pokažu koliko su uniformne i slučajne sekvence koje generišu pojedini generatori.

U radu su analizirane i testirane postojeće implementacije algoritama za koje se u zvaničnoj dokumentaciji navodi da generišu uniformne sekvence (*uniform*, *randint*), kao i implementacija algoritma koji generiše sekvence koje ne prate uniformnu raspodjelu (*betavariate*). Cilj je bio da se za pomenute implementacije PRNG algoritama pokaže da li zaista generišu sekvence onako kako je to opisano u njihovoj zvaničnoj dokumentaciji. Rezultati statističkih testova su to potvrdili, sa visokim nivoom povjerenja.

Dodatno, u okviru rada je implementiran algoritam *rand_22* za koji je pokazano da generiše sekvence koje prate uniformnu raspodjelu, te se može ravnopravno koristiti u primjenama koje zahtijevaju velike sekvence pseudo-slučajnih brojeva sa uniformnom raspodjelom.

U okviru rada su provedeni i dokumentovani sljedeći testovi: *Chi-square* test, Grafički test, Poker test i *Runs* test. Grafički test je na vizuelan način pokazao da algoritmi *uniform*, *randint* i *rand_22* generišu uniformo raspoređene slučajne brojeve, dok se to ne može reći za algoritam *betavariate*, što je i očekivano. Sa druge strane, ostali testovi su kroz numeričke pokazatelje potvrdili ovu činjenicu.

Kao dio budućeg istraživanja, nameće se ideja da se uključe i ostali statistički testovi u testni scenario, kao i da se poveća skup PRNG algoritama i njihovih implementacija, kako bi se imao bolji uvid u stepen uniformnosti i slučajnosti generisanih sekvenci. Osim toga, pošto ovi algoritmi generišu veoma velike sekvence u realnom vremenu, interesantno bi bilo analizirati performanse pojedinih algoritama, kao jedan od važnih segmenata kada je u pitanju njihova evaluacija.

LITERATURA

[1] Heath, Michael T. 2002. Scientific computing: an introductory survey. Boston: McGraw-Hill.
[2] Gannon, L. J. and Schmittroth, L. A. "COMPUTER GENERATION AND TESTING OF RANDOM NUMBERS", United States: N. p., 1963. Web. doi:10.2172/4143109.
[3] Marsaglia, G., "Random numbers fall mainly in the plains", Proc. Nat. Acad. Sci., 61:25-28,1968.
[4] Haramoto, H. and Matsumoto, M. "Again, random numbers fall mainly in the planes: xorshift128+ generators." ArXiv abs/1908.10020 (2019): n. pag.
[5] Katzgraber, H. "Random Numbers in Scientific Computing: An Introduction." ArXiv abs/1005.4117 (2010): n. pag.
[6] Marsland S. "Machine Learning, An Algorithmic Perspective", (CRC Press), 2011.
[7] Matsumoto, M. and Nishimura, T. "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator", ACM Trans. Model. Comput. Simul. 8, 1 (Jan. 1998), 3–30. DOI:https://doi.org/10.1145/272991.272995

[8] Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. "Numerical Recipes", in C. Cambridge University Press, Cambridge, 1995.
[9] Panneton, F., L'ecuyer, P. and Matsumoto, M. "Improved long-period generators based on linear recurrences modulo 2", ACM Trans. Math. Softw. 2006, 32, 1–16.
[10] Knuth, D. E. "The art of computer programming", 3rd edition, Addison Wesley, 1997.
[11] NIST/SEMATECH e-Handbook of Statistical Methods (2012), [Online]. Preuzeto sa: https://www.itl.nist.gov/div898/handbook/eda/section3/eda3674.htm.
[12] Biebighauser, D. "Testing Random Number Generators", University of Minnesota - Twin Cities, REU Summer, 2000.
[13] Glen S., "Statistics How To" (2021), [Online]. Preuzeto sa: https://www.statisticshowto.com/probability-and-statistics/find-critical-values/.
[14] Brown G. R., "Dieharder: A Random Number Test Suite" (2021), [Online]. Preuzeto sa: https://webhome.phy.duke.edu/~rgb/General/dieharder.php.
[15] The National Institute of Standards and Technology (NIST), "Random Bit Generation" (2021), [Online]. Preuzeto sa: https://csrc.nist.gov/projects/random-bit-generation.
[16] L'ecuyer, P., Simard, R., "TestU01: A C Library for Empirical Testing of Random Number Generators", ACM Transactions on Mathematical Software, Vol. 33, 4, article 22, 2007.
[17] Towards Data Science, "Poker Test: pattern-based random number detection in Python" (2020), [Online]. Preuzeto sa: https://towardsdatascience.com/poker-test-pattern-based-random-number-detection-in-python-ec2dba4955bc.
[18] GeeksForGeeks, "Runs Test of Randomness in Python" (2020), [Online]. Preuzeto sa: https://www.geeksforgeeks.org/runs-test-of-randomness-in-python/.
[19] Ryabko, B. Y., & Monarev, V. A. (2005). Using information theory approach to randomness testing. *Journal of Statistical Planning and Inference*, *133*(1), 95-110.
[20] Wichmann, B. A., & Hill, I. D. (2006). Generating good pseudo-random numbers. *Computational Statistics & Data Analysis*, *51*(3), 1614-1622.
[21] Feng, Y., & Hao, L. (2020). Testing Randomness Using Artificial Neural Network. *IEEE Access*, *8*, 163685-163693.
[22] Lambić, D., & Nikolić, M. (2017). Pseudo-random number generator based on discrete-space chaotic map. *Nonlinear Dynamics*, *90*(1), 223-232.

Aleksandar Keleč, ma, Elektrotehnički fakultet, Univerzitet u Banjoj Luci
Kontakt: aleksandar.kelec@etf.unibl.org
Oblast interesovanja: Razvoj web i mobilnih aplikacija, Sigurnost računarskih sistema, Kriptografija i računarska zaštita

Nikola Obradović, ma, Elektrotehnički fakultet, Univerzitet u Banjoj Luci
Kontakt: nikola.obradovic@etf.unibl.org
Oblast interesovanja: Internet programiranje, Objektno-orijentisano programiranje i modelovanje, Razvoj mobilnih aplikacija, Machine learning

Igor Dujlović, ma, Elektrotehnički fakultet, Univerzitet u Banjoj Luci
Kontakt: igor.dujlovic@etf.unibl.org
Oblast interesovanja: Internet programiranje, Objektno-orijentisano programiranje i modelovanje, Razvoj mobilnih aplikacija, Informacioni sistemi

prof. dr Zoran Mitrović, Elektrotehnički fakultet, Univerzitet u Banjoj Luci
Kontakt: zoran.mitrovic@etf.unibl.org
Oblast interesovanja: Nelinearna analiza, Teorija fiksne tačke, Najbolje aproksimacije