Alkalmazott Matematikai Lapok **28** (2011), 41-58.

ELJÁRÁS JÓ MINŐSÉGŰ VÉLETLEN EGÉSZ VEKTOROK GENERÁLÁSÁRA

TAKÁCS SZABOLCS

Alkalmazások során szükségünk lehet véletlen egész vektorok generálására, amelyek minőségét több szempont alapján is értékelhetjük.

Számunkra elsődlegesen az volt fontos, hogy a vektorok eloszlásának egyenletességét biztosítani tudjuk, továbbá a közöttük lévő nyilvánvalóan meglévő összefüggőség a lehető legkisebb mértékben legyen kimutatható. Természetesen az általánosan elfogadott informatikai feltételeknek is megpróbáltunk eleget tenni.

A felsorolt szempontokra több tesztet is alkalmaztunk. Összehasonlításként a MATLAB© véletlen egész vektorokat generáló rutinját használtuk, annak eredményeivel vetettük össze a saját generátorunk eredményeit.

A teszteket előre kiválasztottuk, nem az algoritmus eredményeinek függvényében döntöttünk használatuk mellett, azonban az eredmények ismeretében hajtottunk végre változtatásokat az eredeti algoritmuson – elérve így egy jobb eredményeket mutató verziót.

1. Mi a véletlen?

A véletlent definiálni - gyakorlati kritériumok alapján, mint az majd látható is lesz, nem könnyű feladat. Egzakt matematikai definíciók sokaságát lehet megtenni, melyek többé-kevésbé ekvivalensek egymással, azonban teljesen nyilvánvaló módon vannak eltérések. Vannak sorozatok, melyek az egyik definíció szerint véletlenszerűnek tekinthetőek, míg más definíciók szerint már nem „eléggé" véletlenek.

[3] egzaktul megfogalmaz néhány lehetséges definíciót. Milyen problémákat, illetve lehetőségeket biztosítanak számunkra ezek a definíciók, illetve milyen kapcsolat van a különböző definíciókat teljesítő sorozatok között? Van-e olyan sorozat, mely minden definíció szerint véletlen?

Tekintsünk két régebbi definíciót, melyek nélkülözik a teljes matematikai precizitást, azonban a probléma lényegére jól rávilágítanak.

1.1. Definíció. (Lehmer, 1951) A véletlen sorozat bizonytalan fogalmán olyan sorozatot értsünk, melynek későbbi elemeit az avatatlan személy nem tudja megjósolni; továbbá jól vizsgázik néhány szokásos statisztikai próbán; ezeknek a próbáknak a megválasztása függ attól is, hogy mire szeretnénk használni a sorozatot.

1.2. Definíció. (Franklin, 1962) Egy $U_1, \ldots, U_n$, $U_i \in [0,1)$ sorozat véletlen, ha rendelkezik minden olyan tulajdonsággal, amely az egyenletes eloszlású véletlen változókból álló független minták végtelen sorozatainak közös tulajdonsága.

Megállapítható, hogy Franklin definíciója lényegében Lehmer definíciójának pontosítása, azonban ez alapján a sorozatnak **minden** statisztikai próbát ki kell állnia. Definícióink nem elég pontosak – és [3] eszmefuttatása és tételei alapján, ez utóbbi definíciót szigorúan véve megállapíthatjuk, hogy véletlen sorozat nem létezik.

Ezért általában Lehmer kicsit liberálisabb definíciója alapján érdemes elindulni és dolgozni.

A fenti megállapítás, miszerint nincsen olyan véletlen sorozat, mely minden tesztet kielégítene, a következő egyszerű gondolatmenettel értelmezhető legkönnyebben.

Válasszunk egy olyan véletlen sorozatot, melyben csak 0, vagy 1 értékek szerepelhetnek. Egy 1000 hosszú sorozatban is, sok ismétlést kérve, kell lennie olyan esetnek, melyben egymás után 1000 darab 0 szerepel.

Ez a fajta lokális nem-véletlenszerűség a számítógépes alkalmazások számára katasztrofális helyzetet eredményezne – amit persze az alkalmazások stabilitása érdekében kénytelenek vagyunk kiszűrni. Holott egy igazi véletlen számsorozattól a lokális nem-véletlenszerű viselkedés elvárható.

Ebből következik [3] szerint az a tény, mely alapján el kell, hogy fogadjuk:

Megjegyzés. Nem létezik olyan véletlenszám sorozat, amely minden alkalmazáshoz tökéletesen megfelelő lenne.

2. Elvárások

Jóminőségű pszeudo-véletlen számokat, vagy vektorokat generálni nem könnyű, hiszen számos minőségbiztosítási szempontnak kell eleget tenni. [3] szerint nem létezik olyan pszeudo-véletlen számokat, vagy vektorokat generáló algoritmus, melyhez ne lehetne olyan tesztet előírni, amelyen fennakad. Egészen pontosan, ennek vizsgálatához olykor azt is nehézkes definiálni, hogy mit tekintsünk véletlen sorozatnak.

[3] több, egymással nem feltétlenül ekvivalens definíciót is felsorol. Vizsgálja annak fényében, hogy milyen matematikai tulajdonságaik – és milyen algoritmikus tulajdonságaik vannak az őket kielégítő sorozatoknak.

Vannak azonban általánosan elfogadott elvárások egy algoritmussal szemben ([6], [2]), melyeket illik teljesíteni. Ezek az alábbiakban foglalhatóak össze:

1. **Gyorsaság**

   Az alkalmazott algoritmusunknak nem szabad túl lassan generálnia a számokat, vektorokat, hiszen ezek általában nagyobb programok részeiként

alkalmazandók, így elvárható, hogy az alkalmazások sebessége ne egy véletlen szám meghatározásának sebességétől függjön.

2. **Egyszerűség**

   Minél könnyebben, egyszerűbben implementálható egy algoritmus, annál könnyebb lesz a felhasználhatósága. Világos, hogy egy túlbonyolított algoritmus – bár viselkedését tekintve tűnhet kaotikusabbnak – alapvetően, alkalmazási szempontból lehet mégis rosszabb, mint egy egyszerű, szintén „eléggé kaotikus" verzió.

3. **Szabadság**

   Az algoritmusunk akkor lesz megfelelő, ha az számítógépes rendszertől függetlenül alkalmazható, azaz pl. az operációs rendszer vagy a használt gép nem akadályoz minket a felhasználhatóságában.

4. **Megismételhetőség**

   Nagyon fontos szempont, hogy bár alapvetően véletlennek tűnő számokat, vektorokat szeretnénk előállítani a megadott paraméterek alapján (hiszen algoritmizálunk), azonos paraméterbeállítások mellett (azaz azonos feltételek biztosításával) a program determinisztikussága megmaradjon.

5. **Ciklushossz**

   A pszeudo-véletlen algoritmusok egyik nagy hátránya szokott lenni, hogy a ciklushossz (azaz, hány előállított szám után ismétlődik már a számok sorozata) túl rövid.

6. **Statisztikai próbák**

   A fenti, alapvetően strukturális, informatikai feltételeket (első 4 pont), illetve az 5., matematikai feltételt egy hatodik feltétel teljesítése teszi teljesebbé: nevezetesen, hogy előre meghatározott, az algoritmus viselkedését vizsgáló statisztikai teszteken is meg kell felelnie a programnak.

   Ezeket az olykor nehezen ellenőrizhető kritériumokat statisztikai próbák segítségével fogjuk vizsgálni. Megnézzük, hogy az alkalmazott algoritmusunk mennyire ad hasonló eredményeket ahhoz, mintha egy valóban véletlen (pl. rulett-kerékről) származó számsorozattal dolgoznánk.

Teszteléseink során ezeket a fenti szempontokat vettük alapul. Az első 4 szempont ellenőrzése alapvetően egyszerű volt – ezekre fogunk először kitérni.

Az 5. szempont vizsgálata némiképpen eltérő lesz, erről majd a későbbiekben még lesz szó.

A 6., statisztikai próbák szempontjához előre meghatároztuk, hogy mely teszteket fogjuk alkalmazni – elkerülve így azt a csábítást, hogy az algoritmus vizsgálta során, annak megismerése után olyan teszteket válogassunk ki, melyeknek megfelel az algoritmus. A módosítások az eredeti algoritmuson mind azt a célt szolgálták,

hogy az előre meghatározott teszteken, kontrolált körülmények között tudjunk jól teljesíteni.

Az egyenletesség tesztelésére a χ^2-próba különböző, strukturált változatait és a Kolmogorov–Szmirnov-tesztet alkalmaztuk.

Az összefüggések feltárására szintén χ^2-próbát és monotonitási együtthatókat (Kendall-gamma) alkalmaztunk, illetve a Knuth által javasolt sorozattesztet, valamint annak egy átrendezés-tesztnek nevezett módosítását.

Az eredeti algoritmus forrása [6], és ugyancsak ismerteti [5], a rá vonatkozó minőséghez kapcsolódó bizonyításokkal egyetemben. Az eredeti algoritmust azonban több ponton is módosítottuk, illetve specifikáltuk – ezen módosítások, specifikációk hatását teszteltük különböző minőségbiztosítási eljárások segítségével.

A teszteléshez használt eljárásokat részint [3], részint [4] javaslatai alapján készítettük el.

3. Az eredeti algoritmus leírása

Az algoritmus egész vektorokat vesz egy véges, rögzített halmazból, majd e véletlenszerűen kiválasztott vektorokat adja össze, és az eredményt visszatranszformálja egy adott élhosszúságú kockába (általános esetben tetszőleges tartományba). Lépéseit a következőképpen fogalmazhatjuk meg:

1. **Inicializálás**

$$A \in \mathbb{Z}_+^{n\times m}, \quad n << m, \quad n, m \in \mathbb{Z}_+$$
$$\xi_j \sim U(m) \in \mathbb{Z}_+, \quad j \in [1, \ldots, K] \quad D \in \mathbb{Z}_+^n, \quad K \in \mathbb{Z}_+,$$

ahol:

- A mátrix összes $n \times n$-es részmátrixának determinánsaira teljesül, hogy a legnagyobb közös osztójuk 1.
- ξ_j véletlen, egyenletes eloszlású valószínűségi változók.
- D a pozitív ortáns adott d élhosszúságú kockája (természetesen itt bármilyen tartomány választható).
- K számú véletlen egész vektort szeretnénk generálni.

Azaz egy D tartomány – jelen leírásban a d élhosszúságú, pozitív ortánsban elhelyezkedő n dimenziós kocka egész rácspontjaira szeretnénk véletlen vektorokat generálni.

2. **Ciklus és levágás**

$$\begin{aligned} \mu_0 &= \mathbf{0}, \\ \nu &:= \xi, \\ \mu_i &= \mu_{i-1} + a_\nu, \\ i &= 1 \dots K. \end{aligned}$$

Ha $\mu_{i,j} > d$ (vagy $\mu_{i,j} < 1$) akkor $\mu_{i,j} := \mu^*_{i,j}$, ahol

$$\begin{aligned} \mu_{i,j} \equiv \mu^*_{i,j} \mod(d), \qquad & 1 \le \mu^*_{i,j} \le d, \\ j = 1, \dots, n. & \end{aligned}$$

Azaz a 0 vektorból indulva az A mátrix oszlopainak véletlen sorrendben – egyenletes eloszlású valószínűségi változók realizációinak segítségével – történő egységnyi együtthatójú lineáris kombinációját vesszük, ezek adják a véletlen vektorokat.

Az algoritmus megtalálható [5] alatt, ahol a levágást – ha túllépnénk a kockán (vagy az adott tartományon) az algoritmus egy maradékos osztással oldja meg, mely az adott koordinátán visszaterel minket a tartomány belsejébe.

3.1. Tétel. *Amennyiben az* $\{a_1, \dots, a_m\} \subset \mathbb{Z}^n$ *vektorrendszer tartalmazza az* $\mathbb{R}^n$ *egy lineáris bázisát, a determinánsokra vonatkozó feltétel fennáll és* $\exists t \in [1, m]$: $a_t = 0$, *úgy:*

$$\forall u \in D: \quad \lim_{k \to \infty} P\left(\mu_k = u\right) = \frac{1}{\prod_{j=1}^{n} d_j} = d^{-n},$$

azaz minden u rácspont egyenlő valószínűséggel vétetik fel, amennyiben a generált vektorok száma a végtelenhez tart [6].

Megjegyzés. A tételben szereplő d_j értékek most egyenlőek, hiszen a d élhosszúságú kocka rácspontjaira generálunk. Általában egy $D = (d_1, \dots, d_n)$, pozitív egész élhosszak által meghatározott téglába állítjuk elő a vektorokat.

Bizonyítás. A tétel bizonyítása [6] cikkben megtalálható. □

3.1. Következmény. *A tételből következik az is, hogy ha a* D *tartomány akkora, melyben a rácspontok száma meghaladja az alkalmazott véletlenszám generátor ciklushosszának mértékét, úgy az így megalkotott algoritmusunk ciklushossza meghaladja az eredeti véletlenszámgenerátor ciklushosszát, hiszen pozitív (sőt, határértékben egyenlő) valószínűséggel minden rácspontot előállít az algoritmusunk.*

4. Az algoritmus fejlesztése, javítása

Megjegyzés. Az általunk alkalmazott algoritmusban seed beállítás is szerepelt, ami azt jelenti, hogy amennyiben K vektorra volt szükségünk, úgy a generált μ vektorokat csak egy bizonyos érték után kezdtük el egy mátrixba feltölteni (a seed érték eléréséig szabadon generált a program, nem történ kiíratás). Ez random-seedet eredményezett számunkra, hiszen így az adott D tartomány egy pszeudo-véletlen pontjából indítottuk el az algoritmust.

Megjegyzés. Az eredeti algoritmus leírásában látszik, hogy egyetlen pont van, ahol döntési lehetőségünk van: az A mátrixunk megválasztása. Az algoritmushoz olyan A mátrixra van szükségünk, melyre igaz, hogy az $n \times n$-es aldetereminánsok legnagyobb közös osztója 1.

Amennyiben ez nem teljesül, úgy az algoritmus során sávokat generálnánk, nem tudnánk kitölteni az egész, kitöltendő D tartományt.

4.1. VÁLTOZAT. *Az A mátrixot első körben az alábbi technikával generáltuk: egy $n \times n$-es egységmátrixon végrehajtottunk adott számú (n^2) Gauss-eliminációs eljárást fordított irányban (ezzel elértük, hogy a kapott mátrix determinánsa továbbra is 1 maradjon). A fordított irány azt jelentette, hogy most az egységmátrix sorainak véletlen számszorosát hozzáadtuk / kivontuk egymásból.*

Ezt az eljárást alkalmaztuk egymást után többször. Így garantáltuk, hogy minden $n \times n$-es aldeterminánsa legnagyobb közös osztója 1 maradt, hiszen tartalmazott több olyan részmátrixot is, melynek determinánsa 1.

Fontos azt is megemlíteni, hogy az algoritmus mindenképpen használ egy külső, véletlen szám generátort, mellyel egyenletes eloszlásból származó pszeudo-véletlen számokat nyerünk a vektorok összegeinek előállításához, illetve ezt a generátort használjuk az A mátrix előállításához is.

Az algoritmus teszteléséhez különböző statisztikai eljárásokat alkalmaztunk. Egyik oldalról fontos szempont volt az eloszlás egyenletességének tesztelése. Erre részint χ^2-próbát, részint Kolmogorov–Szmirnov-tesztet választottuk.

Az egyenletességet vizsgáló teszteket a 100 oldalhosszúságú, pozitív ortánsban elhelyezkedő, 2, 3 és 5 dimenziós kockába történő, 2000 vektor generálására végeztük el.

A vektoraink függetlenségének tesztelésére a későbbiekben bemutatásra kerülő teszteket használtuk. Szintén a 100 oldalhosszúságú, pozitív ortánsban elhelyezkedő kockába generált véletlen egész vektorokkal dolgoztunk, azonban a megbízhatóság érdekében 5000 vektort generáltunk (a [3] szakirodalomban a minimális 4000-es érték szerepel).

4.1. Módosítás az algoritmuson

Kétfajta módosítást hajtottunk végre az algoritmuson. Az egyik az A mátrix generálása, a másik pedig egy ritkítás beállítása.

Generálás

Az algoritmus lelkét képező A mátrix megválasztása fontos lépés. A módosításban az alábbi lépéseket tettük.

1. Először a már ismertetett módon végrehajtottuk az A mátrix generálását, majd annak segítségével generáltunk egy megfelelően hosszú (2000 darabos) véletlen vektorrendszert, amit egy M mátrixba rendeztünk.
2. Az A mátrix elejéről megtartottuk a Gauss-elimináció segítségével generált $n \times n$-es részt, így már garantálni tudtuk, hogy a determinánsok relatív prímek legyenek. Ezután az M mátrixból választottunk egy megfelelő nagyságú szeletet, azaz: az A mátrixba már előzetesen legenerált, ebből az eljárásból származó pszeudó-véletlen vektorok kerültek. (Ezzel azt szerettük volna elérni, hogy az algoritmus lényegében a környezettől független legyen – ne teljes egészében a MATLAB© generátorával dolgozzunk.)

Ritkítás

Már említettük, hogy beállítható ritkítás is az algoritmusban. Most a már említett seed mellett még ezt is alkalmaztunk. Így egy jóval nagyobb vektorrendszert alkalmaztunk – nevezetesen 2, 4, 8, 16-szoros ritkítással dolgoztunk és teszteltünk.

Megjegyzés. Nem alkalmaztunk ennél nagyobb ritkítást, mert bár teszteltük, de nem hozott érdemi javulást, vagy romlást a 16-szoroshoz képest, viszont nagyban megnövelte a futásidőt.

A ritkítással mérhetően nem javultak az eredményeink az egyenletesség tekintetében, azonban az összefüggőség esetén már mást tapaszaltunk.

Az első teszt, amit alkalmaztunk a nagy elemszámra való tekintettel a Kolmogorov–Szmirnov-próba volt. Ennek során először a peremeloszlásokat teszteltük, majd minden perem esetén azokat 2, 4 és 5 részre bontottuk, és ezen vágások mentén kialakult téglákban vizsgáltuk a többi koordináta egyenletességét.

A második teszt – ugyanezen téglák kombinációján – annak tesztelése volt, hogy minden téglában egyenletesen vannak-e jelen a generált rácspontok. Ez persze 2^n, illetve 4^n, 5^n téglát jelentett, melyekre χ^2-próbát alkalmaztunk.

Megjegyzés. Ezen a ponton nyer jelentőséget az a tény, hogy a szigorú pozitív kvadránsba generáltunk vektorokat, ugyanis így valóban értelmes osztópontokat hozhattunk létre – nem $50 - 51$ pont került az egyes vágásokba, hanem $50 - 50$, vagy $25 - 25 - 25 - 25$ stb.

A Kolmogorov–Szmirnov-tesztek közül az egyváltozós esetek előfeltétele, hogy folytonos F-eloszlást vizsgálunk. Ismert, hogy a kétváltozós esetnek is az a feltétele, hogy mindkét valószínűségi változó egy-egy folytonos F-, és G-eloszlásból származzon, majd ezen F-, és G-eloszlásokat vizsgáljuk egyenlőség szempontjából.

A Kolmogorov–Szmirnov-tesztek kétváltozós esetében a véletlen egész vektorokat úgy kezeljük, mint ha folytonos eloszlásból származnának. (Például IQ-tesztek esetén is úgy feltételezzük, hogy folytonos eloszlás húzódik meg a háttérben, csak a mérőeszközünk nem tudja ezt mérni).

Így az alábbi eljárást fogjuk elvégezni.

1. Generálunk egy V $n \times m$-es vektorrendszert, azaz m darab n dimenziós vektort. Azt vizsgáljuk, hogy az m darab vektor minden egyes peremeloszlása egyenletes-e.

   Válasszuk ki például V első oszlopát (ami most így egy m hosszú vektor lett) tesztelésre, jelölje ezt V_1.

   Legyen Max és Min V_1 maximális és minimális eleme.

2. Veszünk egy w vektort, mely Max és Min között minden egész értéket felvesz, ráadásul mindet egyenlő mértékben. Ez azt jelenti, hogy minden értékből

$$\left\lfloor \frac{n}{Max - Min + 1} \right\rfloor$$

   darabot veszünk.

   Ezzel elérjük, hogy a V_1 vektorral majdnem megegyező hosszúságú, annak legnagyobb és legkisebb eleme között egyenletes eloszlásból származó w vektorunk legyen.

3. Most w és V_1 eloszlását a fent nevezett illeszkedés-vizsgálati eszközökkel összehasonlítjuk. Így azt feltételezhetjük, hogy mindkettő ugyanazon mérőeszközzel készült, egyenletes eloszlásból származó minta – továbbá, a tesztnek az elemszámok nagyon eltérő volta miatti gyengülését mindenképpen kizárjuk, hiszen majdnem megegyező méretű vektorokat hasonlítunk össze.

4.2. Összefüggőség tesztelése

4.2.1. A Kendall-gamma monotonitási együttható

Tegyük fel, hogy adott $A = (X_1, Y_1)$ és $B = (X_2, Y_2)$ pontpár.

Amennyiben $X_1 > X_2$ és $Y_1 > Y_2$, úgy azt mondjuk, hogy A és B konkordáns viszonyban vannak egymással (pozitív irányú az összefüggés, monoton növő kapcsolat van közöttük).

Amennyiben $X_1 > X_2$, de $Y_1 < Y_2$, úgy azt mondjuk, hogy A és B diszkordáns viszonyban vannak egymással (negatív irányú az összefüggés közöttük, monoton csökkenő kapcsolatot mérünk).

Világos, hogy ha egy ponthalmazban a konkordáns párok p_+ száma a nagyobb, úgy azt mondhatjuk, hogy a ponthalmaz összességében monoton növő, a pontok egymáshoz képest monoton növekednek. Amennyiben a diszkordáns párok

p_- száma a nagyobb, úgy ennek ellenkezőjét állíthatjuk – azaz monoton csökkenő viszonyt vélelmezhetünk a ponthalmazunk esetén. Tehát

$$p_+ = \sharp\left\{\left(A\left(X_1, Y_1\right), B\left(X_2, Y_2\right)\right) \mid \left(X_1 < X_2\right) \wedge \left(Y_1 < Y_2\right)\right\},$$
$$p_- = \sharp\left\{\left(A\left(X_1, Y_1\right), B\left(X_2, Y_2\right)\right) \mid \left(X_1 < X_2\right) \wedge \left(Y_1 > Y_2\right)\right\}.$$

Az is világos, hogy nem minden pontpárt tudunk összehasonlítani – hiszen lehetnek egyező X vagy Y koordináták. Erre az esetre olyan korrekciót hajtunk végre, hogy p_+ és p_- számok különbségét nem az összes létező pároshoz viszonyítjuk, hanem az összeshasonlítható párokra. Az így képezett monotonitási együttható:

$$\gamma = \frac{p_+ - p_-}{p_+ + p_-}.$$

Megjegyzés. Technikai kérdés, hogy a pontpárok összehasonlítását miként végezzük. Ha sorban haladva a már összehasonlítottakat kihagyjuk, vagy rosszabb szervezés miatt minden pontot minden ponttal akár többször is összehasonlítunk – az arányokon, illetve azok hányadosán ez a részlet nem fog változtatni.

Jackknife módszer segítségével mérjük az így elkészített monotonitási együtthatók 95%-os konfidencia-intervallumát. Ezt úgy tesszük, hogy a generált vektorok közül mindig kihagyva a megfelelőt, pszeudo-statisztikákat nyerünk a fenti monotonitási mértékre, majd utána trimmelés segítségével (adott számú legnagyobb és legkisebb elem elhagyásával) meghatározzuk a kívánt szélességű intervallumot.

Megjegyzés. A jackknife eljárásról bővebben [1] alatt olvashatunk. A trimmelés során nem teszünk mást, mint a nagyság szerint sorbarendezett elemek első és utolsó darabjaitól (egy előre meghatározott arányban, szimmetrikusan) megszabadulunk.

Amennyiben ez az intervallum a ponthalmazunk esetén tartalmazza a 0 értéket, úgy azt mondhatjuk, hogy a ponthalmazunk lényegében azonos arányban tartalmaz monoton növekedő és csökkenő pontpárosokat.

4.2.2. Átrendezés- és sorozatteszt

[3] alapján a sorozattesztet, illetve annak átdolgozását alkalmaztuk.

1. **Átrendezésteszt**

   Az eljárásunkat abból a szempontból vizsgáljuk, hogy a monoton növő és csökkenő részsorozatok tulajdonságai átrendezés hatására megváltoznak-e. Ugyanis, ha az átrendezés változtat az eredményeken az azt jelenti, hogy menet közben valamit kihasználtunk, vagy elvesztettünk, és ennek hatása van a sorozatunk viselkedésére.

Az átrendezést az alábbi módon értjük: tekintsük az adott, generált V vektorrendszerünket. A V vektorrendszert egy permutáció segítségével rendezzük át, a permutáció által adott sorrendben felsorolva újra a vektorokat, legyen ez V^*. Amennyiben a monoton növő és csökkenő részsorozatok aránya, száma V és V^* esetén szignifikánsan eltérő, úgy azt mondhatjuk, hogy az eredeti sorozatunkban jelentősége volt a vektorok sorrendjének, az nem egészen volt véletlenszerű.

2. **Sorozatteszt**

   A sorozatteszt [3] alatt megtalálható, melynek lényege, hogy a véletlen számok generálásakor vizsgálható, hogy hány monoton növő és csökkenő részsorozatot generál az algoritmus. Ezek természetesen nem függetlenek egymástól (így pl. az esetszám tesztelésére egyszerű χ^2-próba nem alkalmazható). A függetlenség azért sérül, mert teljesen nyilvánvaló módon egy növekedő sorozatot mindenképpen egy csökkenő kell, hogy kövessen.

   Lényegében annyi történik, hogy összeszámoljuk: a generált vektorrendszerünkön lexikografikus rendezés után hány darab, milyen hosszú monoton növő részsorozat keletkezett. Ezeket utána a fenti [3] hivatkozáson megadott módon statisztikai vizsgálatnak vetjük alá.

5. Informatikai és algoritmus-szervezési szempontok

Az alkalmazott statisztikai eljárásokat majd azután ismertetjük, hogy az alapvető informatikai elvárások teljesüléséről meggyőződtünk.

Az elején felsorolt szempontok tehát, melyeket ellenőriznünk kell a következők:

1. **Gyorsaság**

   Az algoritmus a MATLAB© véletlen generátorával lényegében egyező idő alatt futott, amennyiben nem kértünk túlzott mérvű ritkítást. Világos, hogy egy 20-30-szoros ritkítás 20-30-szor annyi vektor generálását jelenti, amelyek közül csak minden 20. vagy 30. kerül felhasználásra. Így a ritkítás pl. 5000 vektor generálása során már mérhető időveszteségeket okoz.

2. **Egyszerűség**

   Az algoritmus leírásán látszik, hogy annak struktúrája nem túl bonyolult – bár használ egy szubrutint, nevezetesen: mindenképpen alkalmaznunk kell egy külső véletlenszám generátort az algoritmus beindításához, illetve köztes működtetéséhez (minden olyan esetben, amikor szükségünk van egy véletlen számra). Ez feltétlenül bonyolultabbá teszi az algoritmust pl. annál az algoritmusnál, melyet szubrutinként meghívunk.

3. **Szabadság**

   A számítógépes rendszertől annyiban független csak, hogy olyan számítógépes felület kell az algoritmus számára, mely már rendelkezik az előző pontban is említett szubrutinnal. Azonban ettől a ponttól valóban bármely rendszeren alkalmazható.

4. **Megismételhetőség**

   Amennyiben az alkalmazott szubrutin paramétereit ismerjük, úgy a többi paraméter fixálása után nyilván az eljárás ugyanannyira lesz reprodukálható, mint amennyire a szubrutinként alkalmazott véletlenszám generátorunk az volt.

5. **Hosszú ciklusidő**

   A MATLAB© ciklusideje kellően hosszú, és az algoritmus leírásából látható, hogy szubrutinként ezt az eljárást használó algoritmusunk ciklusideje részben ehhez kötött – bár ennél akár hosszabb is lehet, amennyiben a generálásban meghatározott tartomány rácspontjainak számosságát növeljük.

6. **Statisztikai próbák**

   Ennek ismertetése a következő fejezetben történik.

Megjegyzés. Az informatikai kritériumokat lényegében teljesítettük – bár az algoritmus egyszerűsége garancia volt arra nézve, hogy az első négy ponttal nem lehet problémánk.

Az 5-ös pont teljesítése részint a szubrutinként alkalmazott, külső generátoron múlik, tehát amennyiben ez megfelelően lett megválasztva, úgy a mi algoritmusunk is jól fog viselkedni. Másik oldalról [6] alapján a D tartomány rácspontjainak számától is erőteljesen függ a ciklushossz.

A statisztikai próbák teljesítése alapvetően nem a szubrutinon múlik, hanem az A mátrix megválasztásán.

6. Az algoritmussal elért eredményeink a különböző statisztikai teszteken

A vizsgált algoritmust a MATLAB© véletlen egészeket generáló rutinjával hasonlítottuk össze.

6.1. Kolmogorov–Szmirnov-statisztika alkalmazása

A tesztek eredményeinek ismertetése előtt emlékeztetünk rá, hogy 2000 vektort generáltunk a szigorúan a pozitív ortánsban elhelyezkedő, 100-as oldalhosszúságú, adott dimenziós kockába.

Megjegyzés. Bár 100 futás történt, a táblázatokban tört értékek szerepelhetnek. Ugyanis 100 futás esetén, adott dimenzió és adott törés (kocka éleinek felosztása) miatt 100-nál természetesen több statisztikai próba készült.

Például 3 dimenzióban, 4 törés esetén (minden élen 4 részre bontunk) $4 * 3 * 2$ darab kocka keletkezik, tehát ennyi darab kisebb tartományban tesztelük a generált vektorok egyenletességét. (Általánosságban – ha van törés – $\dim * (\dim - 1) * \text{törés}$.) Azaz, e fenti példa esetén ez $24 * 100 = 2400$ tesztet jelent.

Az alkalmazott statisztika feltételei az alábbi módon írhatóak le.

Adott X_i koordinátát szeletekre bontjuk, nevezetesen az alábbi átkódolásokat, töréseket vezetjük be.

$$T_{i,2} = \begin{cases} 1, & X_i \in [1, 50]; \\ 2, & X_i \in [51, 100]. \end{cases}$$

$$T_{i,4} = \begin{cases} 1, & X_i \in [1, 25]; \\ 2, & X_i \in [26, 50]; \\ 3, & X_i \in [51, 75]; \\ 4, & X_i \in [76, 100]. \end{cases}$$

$$T_{i,5} = \begin{cases} 1, & X_i \in [1, 20]; \\ 2, & X_i \in [21, 40]; \\ 3, & X_i \in [41, 60]; \\ 4, & X_i \in [61, 80]; \\ 5, & X_i \in [81, 100]. \end{cases}$$

A Kolmogorov–Szmirnov-statisztikát az egyenletesség tesztelésére alkalmazzuk $\forall j \neq i$, X_j koordinátára, minden lehetséges $T_{i,k}$ mentén, ahol $k \in \{2, 4, 5\}$. Azaz, $KS\left(T_{i,k}, X_j\right)$, ahol $1 \leq i, j \leq dim$, $i \neq j$ és $k \in \{2, 4, 5\}$. A Kolmogorov–Szmirnov-statisztika első koordinátáján a bontásban résztvevő koordináta szerepel, míg a második koordináta mutatja, hogy mely koordináta egyenletességét szeretnénk tesztelni. A vektorrendszerünk dimenzióját most *dim* jelöli.

Az alábbi táblázatokban 100 futás eredményeinek százalékos megoszlását láthatjuk (3 tizedesre kerekítve), a fenti bontások esetén.

MATLAB©					
		Perem	törés = 2	törés = 4	törés = 5
2D	Nincs ritkítás	0	0	0	0
2D	Ritkítás = 2	0	0	0	0
2D	Ritkítás = 4	0	0	0	0
2D	Ritkítás = 8	0	0	0.125	0
2D	Ritkítás = 16	0	0.25	0.125	0.3
3D	Nincs ritkítás	0	0	0	0.033
3D	Ritkítás = 2	0	0	0	0
3D	Ritkítás = 4	0	0	0	0.067
3D	Ritkítás = 8	0	0	0	0.033
3D	Ritkítás = 16	0	0	0	0
5D	Nincs ritkítás	0	0	0	0
5D	Ritkítás = 2	0	0	0	0.03
5D	Ritkítás = 4	0	0.025	0	0.01
5D	Ritkítás = 8	0	0.05	0	0
5D	Ritkítás = 16	0	0	0	0.01

SAJÁT					
		Perem	törés = 2	törés = 4	törés = 5
2D	Nincs ritkítás	0	0.25	0	0
2D	Ritkítás = 2	0	0	0	0.2
2D	Ritkítás = 4	0	0.5	0.125	0.2
2D	Ritkítás = 8	0	0	0.125	0.1
2D	Ritkítás = 16	0	0.25	0	0
3D	Nincs ritkítás	0	0.583	0.292	0.367
3D	Ritkítás = 2	0	0.167	0.125	0.1
3D	Ritkítás = 4	0	0	0	0.1
3D	Ritkítás = 8	0	0.167	0	0
3D	Ritkítás = 16	0	0	0.125	0.033
5D	Nincs ritkítás	0.6	1.425	1.113	1.01
5D	Ritkítás = 2	1	0.4	0.125	0.06
5D	Ritkítás = 4	0.2	0.025	0.013	0.02
5D	Ritkítás = 8	0	0	0.013	0
5D	Ritkítás = 16	0	0	0.025	0.01

Megjegyzés. A tesztek – mint az látható – elfogadható eredményre vezettek, ugyanis a célkitűzést, miszerint nagyságrendileg olyan jó eredményeket szeretnénk elérni, mint a MATLAB© által használt véletlen egész vektorokat generáló algoritmus, lényegében teljesítettük. (Természetesen nem értük el azt a fajta, lényegében 0 valószínűségű határt, amit a MATLAB© produkált, viszont általában az 1%-os küszöb alatt tudtunk maradni).

Még az 5 dimenziós eset ritkítás mentes értékei is a 95%-os határ alatt maradtak (alig lépték át az 1%-ot), bár ezen a ritkítás segített, és 4-es ritkítás után már a MATLAB©-hoz hasonló eredményeket tudtunk kimutatni.

6.2. A χ^2-teszt eredményei

A χ^2-próba eredményeit nem foglaljuk táblázatokba, ugyanis nem akadt fenn sem a MATLAB©, sem az általunk alkalmazott algoritmus.

6.3. A Kendall-féle teszt eredményei

Annyi módosítást alkalmaztunk még, hogy nem minden sorsolásban minden koordinátát minden más koordinátával vetettünk össze, hanem bármely bontás esetén véletlenszerűen választottunk ki 1-1 koordináta párt, akiket összehasonlítottunk egymással.

Például, 5 dimenzió esetén, ha a második koordináta alapján 4 részre bontottuk a generáláshoz alkalmazott kockát, majd azon belül a 3. szeletet teszteltük, akkor hol az első és negyedik, hol a harmadik és ötödik koordináta monotonitását hasonlítottuk össze, és így tovább.

Minden esetben a fent már említett 95%-os szignifikancia-szint melletti konfidencia-intervallumot fogjuk a táblázatokban megmutatni. A konfidencia-intervallumot jackknife eljárás [1] segítségével becsültük meg 100 futás eredményei alapján. A ritkítási beállítások itt is a „nincs ritkítás", 2, 4, 8 és 16 voltak.

Megjegyzés. Mindegyik konfidencia-intervallum tartalmazza a 0 értéket, azaz a vektorokat nézve, koordinátánként sztochasztikusan domináns pár nem található, illetve szignifikánsan nem mutatható ki, hogy valamely koordinátapár mentén monoton növekedést, vagy csökkentést generálna szisztematikusan az algoritmus.

Továbbá megállapítható az is, hogy a konfidencia-intervallumok hossza nem túl variábilis, azaz egyik dimenzió egyik ritkításának esetén sem tapasztaltunk érdemileg hosszabb, vagy rövidebb konfidencia-intervallumot.

6.4. Az átrendezésteszt és a sorozatteszt eredményei

Az előzetes teszteket 32-szeres ritkítás mellett is elvégeztük, azonban e két tesztet futásideje és számításigénye okán csak 8-szoros ritkításig teszteltük magasabb esetszámon. Látható lesz, hogy a vizsgált dimenziók esetén már a 8-szoros ritkítás is elfogadhatóan jó eredményeket mutat.

Nincs Ritkítás	MATLAB©	SAJÁT
2D	[-0.0417; 0.0423]	[-0.0445; 0.0503]
3D	[-0.0680; 0.0417]	[-0.0516; 0.0458]
5D	[-0.0413; 0.0372]	[-0.0516; 0.0505]
Ritkítás = 2		
2D	[-0.0407; 0.0463]	[-0.0569; 0.0404]
3D	[-0.0414; 0.0425]	[-0.0528; 0.0557]
5D	[-0.0387; 0.0360]	[-0.0845; 0.0793]
Ritkítás = 4		
2D	[-0.0490; 0.0480]	[-0.0510; 0.0379]
3D	[-0.0431; 0.0381]	[-0.0516; 0.0554]
5D	[-0.0472; 0.0483]	[-0.0491; 0.0586]
Ritkítás = 8		
2D	[-0.0458; 0.0342]	[-0.0476; 0.0470]
3D	[-0.0405; 0.0351]	[-0.0533; 0.0419]
5D	[-0.0664; 0.0417]	[-0.0603; 0.0471]
Ritkítás = 16		
2D	[-0.0418; 0.0384]	[-0.0504; 0.0514]
3D	[-0.0368; 0.0432]	[-0.0510; 0.0500]
5D	[-0.0451; 0.0417]	[-0.0453; 0.0394]

6.4.1. Az átrendezésteszt eredményei

A χ^2-próba teszteredményihez hasonlóan itt sem foglaljuk táblázatba az adatokat, mert minden generált vektorrendszer átment ezen a próbán.

Az átrendezés hatására nem változott semmi. Azt tapasztaltuk, hogy az eljárások – 100 futásból – egyszer sem fogtak el egy generált rendszert sem.

6.4.2. Sorozatteszt eredményei

Ez a teszt (lásd [3]) a ritkítási paraméter beállításának tesztelését volt hivatott elvégezni.

Az eljárás során egy alap mátrixot generálva ritkításonként, több tesztet is végrehajtottunk.

Továbbra is 2, 3 és 5 dimenziós vektorokkal dolgoztunk. 5000 darab véletlen vektort generáltunk 1-1 sorozatban, a ritkítások mértékét változtatva.

Azt tapasztaltuk, hogy a teszten fennakadt futások aránya nem változik (sem a MATLAB©, sem a mi generátorunk esetén) a ritkítás paraméter növelésével.

Sorozatteszt eredményei

Az alábbi két táblázatban ismertetjük, hogy az adott dimenzióban hány sorozatot generáltunk, illetve az adott sorozatszám mellett milyen arányban fogta meg a sorozatteszt a tesztelt algoritmust.

Futások száma

Ritkítás	2D	2D	3D	3D	5D	5D
mértéke	MATLAB©	Saját	MATLAB©	Saját	MATLAB©	Saját
Nincs ritkítás	10000	10000	10000	10000	10000	10000
Minden 2.	10000	10000	10000	10000	5000	5000
Minden 4.	5000	5000	1000	1000	1000	1000
Minden 8.	1000	1000	1000	1000	500	500

Megfogott futások aránya

Ritkítás	2D	2D	3D	3D	5D	5D
mértéke	MATLAB©	Saját	MATLAB©	Saját	MATLAB©	Saját
Nincs ritkítás	5,4%	5,3%	5,4%	5,4%	5,4%	5,2%
Minden 2.	5,4%	5,4%	5,6%	5,3%	5,3%	5,8%
Minden 4.	5,6%	5,2%	6,8%	4,8%	5%	5,1%
Minden 8.	5,7%	4,7%	4,1%	4,8%	4,6%	4,8%

A táblázatokból kiderül, hogy 1-2 tizedes eltérés van a MATLAB© generátora és a saját generátorunk között – ráadásul mindegyik az 5%-os hibahatár környékén mozog (a 6%-ot egy esetben lépi át a MATLAB© generátora).

Ez azt jelenti, hogy az általunk alkalmazott generátor ez alapján a teszt alapján sem mutat összességében rosszabb képet, mint a MATLAB© által használt általánosan elfogadott véletlen egész vektorokat generáló algoritmus.

7. Összegzés

Összefoglalásképpen elmondható, hogy a [6] cikkben ismertetett algoritmus átdolgozásával egy elfogadható eredményeket mutató algoritmust sikerült alkotni.

A vizsgálatra előre kiválasztott teszteken a referenciaként használt MATLAB© véletlen vektorokat generáló rendszere és az általunk javított algoritmus hasonlóan jó eredményeket mutattak.

Megállapítható, hogy a ritkítási paraméter használatával az algoritmusunk a magasabb dimenzióknál javuló tendenciát mutat, illetve nem különbözik lényegesen egymástól ebben a tekintetben sem a MATLAB© beépített és az általunk ismertetett véletlen egész vektorokat generáló algoritmus. Érdemes észrevenni, hogy a túl nagy ritkítás sem feltétlenül jó – részint nem javít már az eredményeken érdemben, a futásidőt viszont nagyban megnöveli.

Megfigyelhető, hogy az eloszlás illeszkedésének szempontjából a ritkításnak érdemi jelentősége nincsen (ahogy az várható is volt). Az egymás után generált vektorok közötti összefüggőség csökkentésére magasabb dimenzióban volt igazán hatása. Tapasztalataink szerint minden 16., vagy még későbbi vektor figyelembe vétele már nem javított érdemben az eredményeken, tehát a ritkítási paramétert elegendő pl. 8-as értékre beállítani.

Fontos kiemelni azt a tényt, hogy az ezen technikával megalkotott generátor ciklusa nem a belsejében alkalmazott szubrutintól függ elsősorban, hanem a generálás során kitöltendő tartományban található rácspontok számától, melynek segítségével tetszőlegesen hosszú ciklushosszal rendelkező algoritmus konstruálható.

Hivatkozások

[1] Efron, B. and Gong, G.: *"A Leisurely Look at the Bootstrap, the Jackknife, and Cross-Validation"*, The American Statistican **37**, (1983) 36–48.

[2] Janke, W.: *Pseudo Random Numbers: Generation and Quality Checks*, Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms, Lecture Notes, J. Grotendorst, D. Marx, A. Muramatsu (Eds.), John von Neumann Institute for Computing, Jülich, NIC Series, Vol. **10**, ISBN 3-00-009057-6, (2002) 447–458.

[3] Knuth, D.: *A programozás művészete*, Vol. **2**, Műszaki Könyvkiadó, Budapest, (1987) 51–87; 153–177.

[4] Marsaglia, G.: *Diehard Battery of Tests of Randomness*, (1995). http://stat.fsu.edu/pub/diehard/

[5] Ramirez Alfonsin, J. L.: *The Diophatine Frobenius Problem*, Oxford Lecture Series in Mathematics and its Applications, Vol. **30**, Oxford University Press, (2005).

[6] Vizvári, B.: *Generation of Uniformly Distributed Random Vectors of Good Quality*, Rutcor Research Report, (1994).

(Beérkezett: 2011. január 10.)

TAKÁCS SZABOLCS
Károli Gáspár Református Egyetem
Bölcsésztudományi Kar, Pszichológiai Intézet, Általános lélektani és módszertani tanszék
1037, Budapest, Bécsi út 324, 5. épület, fszt.
e-mail: tretarkhon@gmail.com

A PROCEDURE FOR GENERATING HIGH QUALITY PSEUDO-RANDOM INTEGER VECTORS

Szabolcs Takács

We present a method for generating high quality pseudo-random numbers or vectors, focusing on the applicability in applications.

The following measure of quality is used: the distribution of the generated vectors must be uniform and the cross-dependency between the generated vectors must be low. Other, traditional aspects from computer science are also considered.

We apply various tests to verify the quality of the generated sequences. As a reference, we used MATLAB©'s pseudo-random algorithm to compare our results against. We have fixed the set of test to be applied before generating any results. However, after the set of test methods have been fixed, we did tune our algorithm to improve it's performance on the tests.