Műszaki Tudományos Közlemények vol. 18. (2023) 37–44.
DOI: Magyar: https://doi.org/10.33895/mtk-2023.18.07
Angol: https://doi.org/10.33894/mtk-2023.18.07

VÉLETLENSZÁM-GENERÁTOR

RANDOM NUMBER GENERATOR

Harangus Katalin,[1] Kakucs András[2]

[1] *Sapientia Erdélyi Magyar Tudományegyetem, Műszaki és Humántudományok Kar, Marosvásárhely, Románia, katalin@ms.sapientia.ro*

[2] *Sapientia Erdélyi Magyar Tudományegyetem, Műszaki és Humántudományok Kar, Marosvásárhely, Románia, kakucs2@ms.sapientia.ro*

Abstract

Illustration plays an important role during education: The Galton board is a suitable tool for illustrating random processes and explaining probability distributions. We have created this tool in a virtual version, which facilitates data collection for statistical processing of experimental data and also enables the study of non-symmetrical distributions. The random processes on the device are simulated, which requires a random number generator. Since there were some doubts about the software-generated pseudo-random numbers, we created a true random number generator based on the input noise of microcontrollers.

Keywords*: random number generator, Galton board.*

Összefoglalás

Az oktatás során fontos szerephez jut a szemléltetés: a véletlen folyamatok szemléltetésére, a valószínűségi eloszlások magyarázatára a Galton-deszka mutatkozik alkalmas eszköznek. Mi ezt az eszközt virtuális verziójában alkottuk meg, ami megkönnyíti a kísérleti eredmények statisztikai feldolgozása érdekében végzett adatgyűjtést, és lehetővé teszi a nem-szimmetrikus eloszlások tanulmányozását is. Az eszközön a véletlen folyamatokat szimuláljuk, amihez szükség volt egy véletlenszám-generátorra. Mivel a véletlen számok szoftveres előállítása körül bizonyos kételyek merültek fel, egy hardveres véletlenszám-generátort hoztunk létre, a mikrokontrollerek bemeneti zajára alapozva.

Kulcsszavak*: véletlenszám-generátor, Galton-deszka.*

1. Bevezetés

Az egyetemi hallgatók oktatása során alapvető tudást a valószínűségszámítás és a statisztika tárgyköréből is át szeretnénk adni. Legalább annyit, amennyi a véletlen jelenségek természetét tisztázza, és az azokat leíró, gyakorlati hasznú, számszerű összefüggéseket ismerteti. Mivel ez a tárgykör a matematikához tartozik, alapos megértése és az elmélyült tanulmányozása matematikusi gondolkozási módot igényel. Innen ered a feladat teljesítésében a legnagyobb kihívás: a mérnök gondolkozási módja nem azonos a matematikusával: míg az utóbbi a fogalmakat rendszerint nem köti a való világhoz, a mérnök úgyszólván csak annak él. A tapasztalat szerint a mérnök hallgatók elméleti matematikatudása még alapvető szinten is hiányos, és nem is igazán látják a szükségességét e hiány pótlásának. A nem mérnöki szakok hallgatói esetében még rosszabb a helyzet, e szakokon matematika oktatására nem is kerül sor. Így az absztrakt gondolkozási módot feltételező témakörök oktatásában igencsak fontos szerephez jut az elvont fogalmakat a valósághoz kapcsoló szemléltetőeszközök használata **[1, 2]**. Ilyen pl. a különféle algoritmusok működésének a szemléltetése is, szoftveres és hardveres eszközök segítségével. Ennek alapján született meg az ötlet, mi szerint a véletlen folyamatok természetének tisztázásában, amit a mennyiségi összefüggések megállapítása követ, a Galton-deszkára lehetne „támaszkodni”.

Sor került tehát egy virtuális Galton-deszka fejlesztésére. A Galton-deszka eredeti verziójában a normál eloszlás illusztrálását szolgáló eszköz. A virtuális változat a lejátszódó természeti jelenséget szimulálja, és ebbe a felhasználó be tud avatkozni, így a normáltól különböző eloszlást is szemléltetni lehet vele. A szimulálás számítástechnikai eszközök segítségével történik, miáltal a jelenség nemcsak megfigyelhetővé válik, hanem „menet közben" adatokat is gyűjthetünk róla, amelyeket valamilyen számítógépes programmal utólag feldolgozni, elemezni is tudunk.

A virtuális Galton-deszka véletlen jelenségeket szimulál, így szükség van egy véletlenszám-generátorra. Az eddigi tapasztalat azt mutatja, hogy a szoftveresen, algoritmusokkal előállított pszeudovéletlen számokkal való munka nem mindig vezet a várt eredményhez, így például pár száz egymást követő véletlen szám esetében sem igazán akar mindig összejönni az, hogy azok egyenletesen oszlanának el. Ennek kiküszöbölésére igazi véletlen számok hardveres előállítását céloztuk meg, ami a virtuális Galton-deszka megépítésében nem jelentett újabb költséget, ugyanis az azt vezérlő mikrokontrollerrel valósítottuk azt meg. A cikk ennek az ötletnek a megvalósítását írja le.

A didaktikai eszközt magát a 2022. szeptember utolsó napjaiban Bécsben tartott, „25th International Conference on Interactive Collaborative Learning" rendezvényen a „Didactic Tool for Intuitive Random Process Visualization" előadásunkban már bemutattuk és publikáltuk. E cikk második része az ott elhangzottak rövid összefoglalása.

2. Véletlen számok előállítása

A véletlen számok előállítása a legtöbb programozói környezetben egyszerű, mert rendszerint a rendelkezésünkre áll egy e célból megírt függvény **[3]**, például:

– C-ben (így az Arduino programozásakor) és C++-ban: *rand()*, amely egy 0 és *RAND_MAX* közötti egész számot visszaadó függvény, ahol a felső határérték a programozói környezettől függ, de garantáltan az legalább 32767;
– Visual Basicben: *Rnd()*, ami egy, a [0, 1) intervallumból származó véletlen számot képvisel;
– Excelben *RAND()*, ami a szintén a [0, 1) intervallumból származó véletlen számot adó függvény, vagy *RANDBETWEEN(bottom, top)*, amely a [*bottom*, *top*] intervallumon értelmezett egész véletlen számot visszatérítő függvény.

Ezek a függvények valamilyen algoritmus alapján állapítják meg a soron következő, névlegesen egyenletes eloszlású véletlen szám értékét, ami miatt azok kiszámított, tehát nem ténylegesen véletlen számok. Ezeket pszeudovéletlennek (álvéletlennek) nevezik. Gyakorlatilag az egymást követő számok véletlennek tűnnek, azonban mindig ugyanolyan sorrendben következnek. Ahhoz, hogy az ismétlődés esélyét csökkentsük, rendszerint megvan a beállítási lehetősége annak, hogy honnan kezdjük „olvasni" a sort. Erre van C-ben a *randomSeed(x)* függvény, Visual Basicben pedig egy *Randomize(x)* utasítás, mindkettő a paraméterenként megadott számú, *x*. tagot állítja be a sor kezdetének. Nyilván, ha mindig ugyanazzal a paraméterrel dolgoznánk, akkor mindig ugyanott kezdődne a sor, így a számsort pl. a számítógép órája alapján kijelölt helyen való kezdéssel lehet véletlenszerűbbé tenni (a paraméter pl. az éjfél óta eltelt másodpercek száma).

Amennyiben ténylegesen véletlen számra van szükségünk, akkor azt valamilyen fizikai eszközzel tudjuk előállítani **[3]**. Ez lehet pl. akár egy dobókocka vagy pedig egy hardveres véletlenszám-generátor. Működésük alapján nem egy algoritmus, hanem valami véletlen lefolyású fizikai jelenség áll. A kereskedelemben kapható RPG100 integrált áramkör a 16 bites véletlen számot a félvezetők zajára alapozva állítja elő, azonban ezt az áramkört a számítógéphez kell illeszteni.

Egyszerűbb és olcsóbb megoldást keresvén, a mikrokontrollerek analóg bemenetének zajára alapozva építettük meg a saját verziónkat, amely ugyan lassabb a példaként említett áramkörnél, de elvileg tetszőleges felbontású véletlen számokat tudunk előállítani vele.

A megvalósítás a bárki számára elérhető, olcsó, könnyen programozható és számítógéphez egyszerűen csatlakoztatható mikrokontrolleres Arduino és ESP-fejlesztőlapra alapoz: mi az Arduino Mega **[4]** és az ESP32-es **[5]** verziókat próbáltuk ki. A kettő közötti lényeges különbség a sebességben rejlik: az ESP32-é nagyságrenddel nagyobb.

A működési elv a mikrokontroller „lebegő", be nem kötött bemeneteinek véletlenszerűen alakuló feszültségének mintavételezésén alapul. A bemenet állapotának egyértelműsítésére általában azt egy nagyobb értékű ellenállással a testre, digitális bemenetek esetén pedig a pozitív tápfeszültségre kötik. Ezek az ellenállások („pull down", „pull up") néhány esetben a mikrokontroller beépített elemei, amelyeket szoftveresen lehet engedélyezni, egyébként nekünk kell azt az áramkörhöz csatlakoztatni. Ahhoz, hogy a bemenet állapota véletlenszerű legyen, arra nem kell

rákötni semmit sem, és a belső pull down/pull up ellenállást (ha van ilyen) is ki kell iktatnunk.

Ha egy lebegő digitális bemenetet mintavételezünk, akkor annak állapota véletlenszerűen hol LOW, hol pedig HIGH lesz. Ha pedig egy analóg bemenetet, akkor a mintavételezett feszültség értéke alakul véletlenül a legkisebb és a legnagyobb lehetséges értékek közötti tartományon.

Mindkét esetben a kapott jel az illető bemenet zajfeszültségétől függ. Hogy mi lenne e zaj forrása, arra több magyarázat is létezik, valószínűleg azok kombinált hatásáról van szó. Az egyik forrás a félvezetők (meg a vezetők és a dielektrikumok) termikus zaja, amelyre a fent említett RPG100 integrált áramkör is alapoz. Egy másik fontos összetevő az „atmoszferikus zaj”, amelyet egy állomásra nem hangolt rádió sistergéseként hallhatóvá is tehetünk. Ez utóbbinak több összetevője is van, természeti jelenségekből és emberi tevékenységből fakadóan. Az emberi tevékenységből származó összetevők egy része szabályos ismétlődést mutathat, mint pl. az elektromos hálózat 50 Hz-es zaja, így az nem alakul teljes mértékben véletlenszerűen.

A mikrokontrollerek bemeneti zajának a harmadik forrása a tapasztalatunk szerint maga a mikrokontroller és a körülötte levő áramkör. Így az általunk tesztelt Wemos Lolin32 klón (ami egy ESP32-re épített fejlesztőlap) 32-es bemenete kifejezetten zajos, jóval zajosabb a többi bemenetnél, ami vélhetően valamiféle tervezési hibából származik.

Az Arduino 10 bites, az ESP32 pedig 12 bites felbontással mintavételezi az analóg bemenet feszültségét, amelyet egy egész szám formájában olvashatunk le az analogRead() függvény segítségével.

Ha egy analóg bemenetre nem kötünk rá semmit, akkor a környezeti, atmoszferikus zajt antennaként felfogó vezetékek hossza minimális lesz. Ekkor a mintavételezett feszültség kaotikusan változik, ami a fejlesztőlap elemeinek saját zajától függ, tehát kevés lesz benne az esetlegesen szabályszerűséget mutató összetevő. A tapasztalat szerint a zajfeszültség túl lassan változik, tehát két, egymást követő jel nagysága között nincs nagy különbség. Ha ezt a mintavételezett feszültséget használnánk a véletlen szám előállításához, akkor az egymást követő számok között sem lenne nagy a különbség.

Ahhoz, hogy a véletlenszám-generátorunk ténylegesen használható véletlen számokat hozzon létre, a mintavételező áramkörnek (az analóg-digitális konverternek) azt a hiányosságát használjuk fel, mi szerint a mintavételezéssel kapott, a jel nagyságát adó szám utolsó egy-két bitje még egy igazán stabil feszültség mérésekor is ingadozó, amivel a zaj esetlegesen szabályosan változó összetevőit is ki lehet iktatni.

A működési elv tehát a következőképpen algoritmizálható:

– mintavételezzük a lebegő analóg bemenetet;
– megállapítjuk a kapott digitális jel utolsó egy-két bitjét;
– ezeket hozzáadjuk egy bitekből álló sorhoz;
– ha ez a sor már elég hosszú, akkor azt véletlen számként értelmezzük (pl. 16 bittel egy 0 és 65535 közötti véletlen számot kapunk).

A C-ben megírt, Arduinóra és ESP32-re alkalmazott programunkban, csak az utolsó bit megtartásával, ez a következőképpen néz ki **[6]**:

```
B = byte(analogRead(Pin0) &
0b00000001) |
(byte(analogRead(Pin1) &
0b00000001) << 1) |
...
(byte(analogRead(Pin7) &
0b00000001) << 7);
```

ahol *analogRead(PinX)* az analóg bemenetről beolvasott érték, *0b00000001* egy maszk (amelynek csak az utolsó bitje 1). Az & művelettel ezt a maszkot alkalmazzuk (bitenkénti „és”), minek következtében a beolvasott értéknek csak az utolsó bitjét tartjuk meg (a többit nullázzuk). Ezt a bitet a << *n* művelettel eltoljuk balra (ahol n adja meg azt, hogy a megtartott bit hova kerüljön), majd az eredményt egy-byte-os számmá alakítjuk (*byte(...)* – erre azért van szükség, mert az *analogRead()* függvény által visszatérített érték nem egybájtos adat) –, majd a | művelettel (bitenkénti „vagy”) beillesztjük a már létrehozott sorba. Az eredmény egy byte lesz. Ha két-byte-os számra van szükség, akkor azt két byte-ból a $B1 + 256 \times B2$ művelettel kapjuk meg.

A véletlen számok létrehozásának felgyorsítása végett nyolc különböző analóg bemenetet mintavételezünk egyszerre, a nyolc véletlen bit létrehozásához.

Ha csak egy véletlen „igen/nem” adatra van szükségünk (mint pl. a virtuális Galton-deszka megépítésénél), akkor még egyszerűbb a dolgunk: az analóg bemenetről beolvasott egész érték páros-páratlan értékét kell figyelnünk: a generált véletlen szám a kettővel való osztás maradékaként 0 vagy 1.

3. A hardveres véletlenszám-generátor tesztelése

A tesztelés során a program egy ciklusában 2 egy-byte-os adatot hozunk létre, amelyekből egy 0 és 65 535 közötti egész véletlen számot számítunk ki. A tapasztalat azt mutatja, hogy a mintavételezés után egy kis időnek el kell telnie ahhoz, hogy az áramkörök visszaálljanak a nyugalmi állapotukba, ellenben a kapott mennyiségek nem alakulnak véletlenszerűen. Ha a véletlenszám-generálást egyetlen bemenet mintavételezésével oldanánk meg, akkor a két egymást követő bit meghatározása közötti várakozás miatt a folyamat nagyon lassú lenne. Így inkább nyolc különböző bemenetet mintavételezünk sorban, várakozás nélkül, remélvén, hogy ezek nincsenek hatással egymásra. A byte-ok (8-8 bit) meghatározása között egy rövid időre megszakítjuk a program futását. Az így kapott számokból 16 384 tagú sorozatokat, mintákat állítottunk elő. E minták növekvő sorrendbe rendezésével az **1. ábrá**n látható grafikonokhoz jutottunk.

Azt tapasztaltuk, hogy az egymást követő mintavételezések közötti várakozás hossza befolyásolja a nyert adatsor minőségét. Várakozás nélkül az adatsorok nagyszámú zérus értéket tartalmaznak. 2 ms várakozás már javít a helyzeten, de mind az ESP32, mind az Arduino Mega, erősen nemlineáris viselkedést mutat.

A 10 ms-os várakozás már lényegesen javít a helyzeten, az Arduino esetében már elegendőnek is tűnik. Az Arduino órajel-frekvenciája nagyságrenddel kisebb az ESP-jénél, így a 10 ms-hoz hozzáadódik a nagyobb ciklusidő miatti késlekedés is.

A 25 ms-os várakozás az ESP32 esetében is jó közelítéssel lineáris mintához vezet.

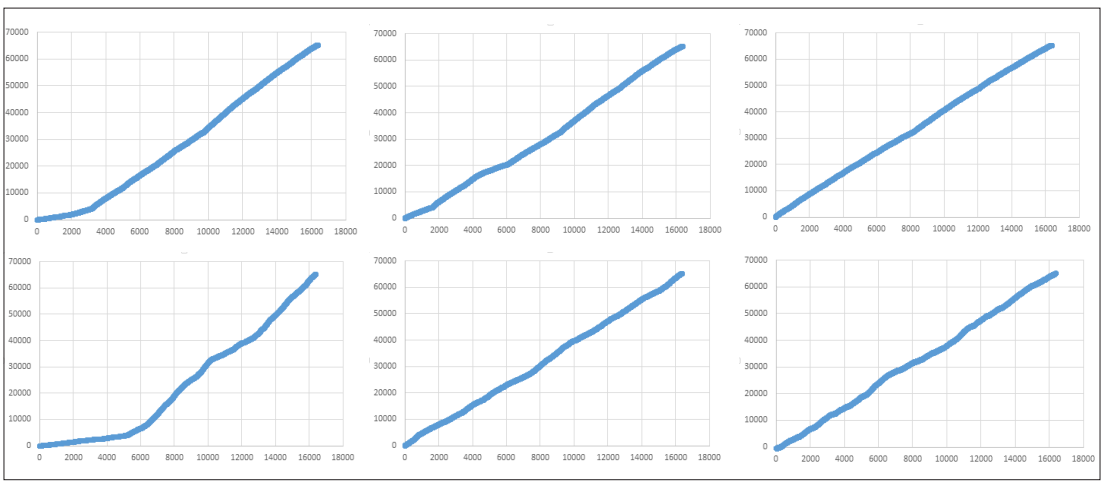
Az Arduino Mega esetében az **1. ábrá**n bemutatott, 25 ms-os várakozással mintavételezett számsornál jobb minőségűt is kaptunk (a példaként bemutatott 10 ms-os minta is jobb annál); a következőkben ezt elemeztük tovább, két ok miatt:

– mivel a kapott minták közül a gyengébb minőségűek közé tartozik;
– mivel a szemléltetőeszközünket ezzel az áramkörrel volt egyszerűbb megépíteni.

Az elemzés során kiszámoltuk a minta empirikus átlagát és szórását: $m = 32518{,}35$, $\sigma = 19234{,}06$. Mivel a kapott számok a kívánalom szerint 0 és 65 535 között egyenletes eloszlásúak kellene, hogy legyenek, az elméleti átlag 32 767,50, a szórás pedig 18 918,32 lenne. Azt tapasztaljuk, hogy ezek egymáshoz elég közel álló számok.

Kiszámoltuk a minta empirikus eloszlásfüggvényét, valamint meghatároztuk az elméleti eloszlásfüggvénynek a minta tagjaira számolt értékét. E két számsor alapján Excelben a χ^2 próbával (*CHISQ.TEST*) elvégeztük az illeszkedésvizsgálatot, a visszatérített érték pedig 1 volt, ami a tökéletes illeszkedésre utal (pedig láthatjuk, hogy az nem éppen tökéletes), miszerint igen nagy valószínűséggel állíthatjuk azt, hogy a kapott véletlen számok egyenletes eloszlásúak.

A gyanúsan jó egyezés miatt egy másik, nem-parametrikus próbát is elvégeztünk, a Kolmogorov–



1. ábra. *Növekvő sorrendbe rendezett minták.*
Fent ESP32, lent Arduino Mega.
Balról jobbra: 2 ms, 10 ms és 25 ms várakozás két mintavétel között.
A vízszintes tengelyen a létrehozott véletlen szám sorszáma, a függőlegesen annak nagysága látható

Szmirnov-tesztet. Ez az empirikus és az elméleti eloszlásfüggvény közötti legnagyobb eltérés alapján dönti el azt, hogy a kettő azonos eloszláshoz tartozónak tekinthető-e, vagy sem. Az eltérés D_i abszolút értékének változását a **2. ábra** mutatja.

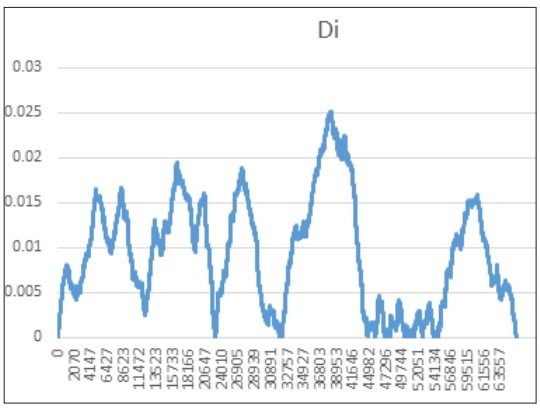
D_i legnagyobb értékét egy D_{kr}, a megbízhatósági szinttől és a minták számától függő kritikus értékkel kell összehasonlítani: ha az meghaladja a kritikus értéket, akkor az illető megbízhatósági szint mellett nem lehet elfogadni azt, hogy a minta az adott elméleti eloszlással rendelkezne.

A nem túlságosan jó minőségűnek tartott, 16 384 számból álló mintánk esetében a szokásos megbízhatósági szintek mellett a K-S teszt a χ^2 próbával ellentmondó eredményre vezetett, vagyis nem sikerült bizonyítani az eloszlás egyenletes mivoltát. Azonban ha a K-S tesztet nem a minta elemeivel meghatározott eloszlásfüggvénnyel, hanem az intervallumokba csoportosítással kapott hisztogrammal végezzük el, akkor az is pozitív eredményhez vezet.

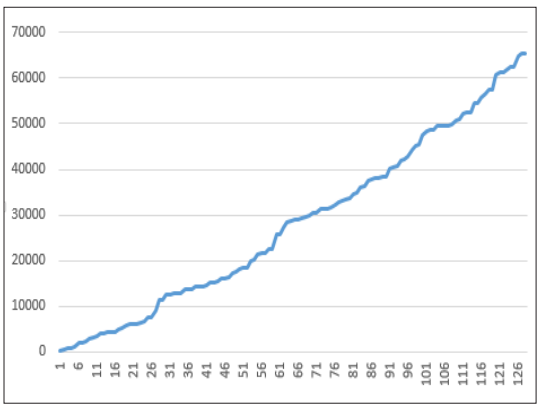
Ellenőrzésképpen ugyanezeket a számításokat Excelben generált pszeudovéletlen számsorokkal is elvégeztük, az eredmények hasonlóképpen jók vagy még jobbak is voltak.

A szoftveresen generált számsorokkal kapcsolatban azonban az a kifogás merült fel, miszerint ha azok nem elég hosszúak, akkor az egyenletestől távol álló eloszlást mutathatnak. Emiatt a hardveresen létrehozott számsorainkat, a létrehozásuk sorrendjében, kisebb részmintákra, rövidebb számsorokra osztottuk, és megvizsgáltuk azt, hogy azok empirikus átlaga és szórása miként alakul. A **3. ábrá**n pl. a 256 darab 128 elemű részmintára felosztott számsorunk esetét láthatjuk, ahol az empirikus átlag 28 307,41 és 36 125,93 között változik.

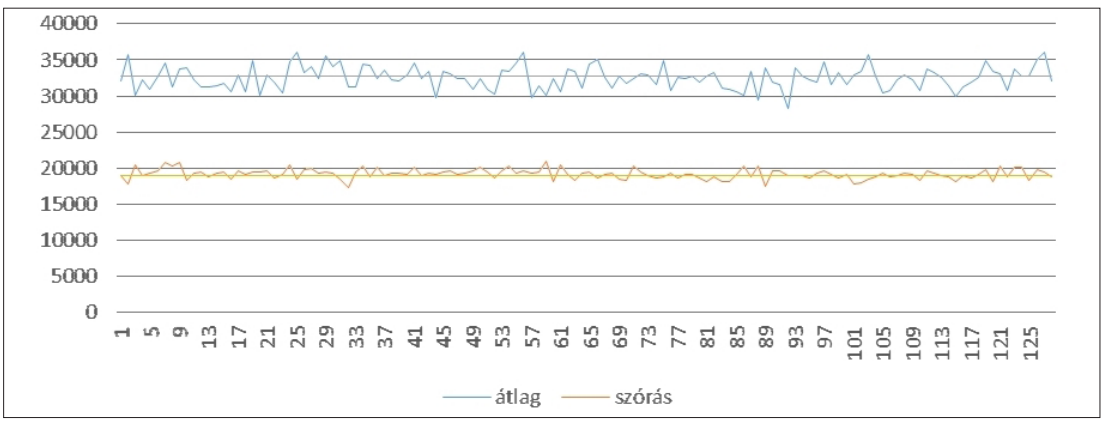
A teljes minta átlagától a legnagyobb eltérést a 28 307,41-os érték mutatja (a 92. részminta). Az ebben a mintában levő, sorrendbe állított számok grafikonja a **4. ábrá**n látható. Az erre a mintára



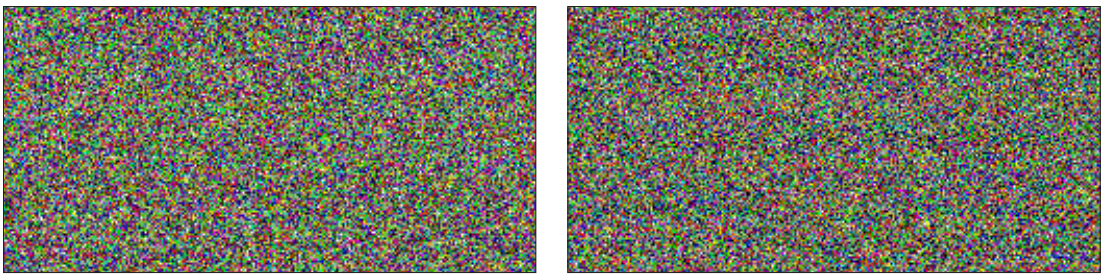
2. ábra. *Az empirikus és az elméleti sűrűségfüggvény közötti eltérés abszolút nagysága*



4. ábra. *Az átlagtól legnagyobb eltérést mutató részminta elemeinek grafikonja*



3. ábra. *Az empirikus átlag és szórás változása az elméleti értékek körül,a 128 elemű részekre bontott mintára*



5. ábra. *A véletlen számok bitmapként való megjelenítése (balra Arduino Mega, jobbra ESP32)*

elvégzett χ^2 próba és a K-S teszt (amelyet a 128 számmal végeztünk el, nem alkalmaztuk a hisztogramos közelítést) szerint az egyenletes eloszlás hipotézise még mindig igen nagy valószínűséggel fogadható el.

Az esetleges ismétlődések és a sorra következő (tehát nem a rendezett számsorok) eloszlásának egyenletességének vizsgálatára további eszközöket is be szoktak vetni.

Elterjedten használt megoldás a számsorok bitenkénti vizuális megjelenítése, ahol a 0-s bitek fekete, az 1-es bitek pedig egy fehér képpontot kódolnak egy fekete-fehér bitmap-ben. Ha az eloszlás tényleg egyenletes, akkor ez az ábra egyenletesen szürke kell, hogy legyen, és semmiféle ismétlődést, mintázatot nem kell mutatnia. Mi ezt az elvet egy kissé másképp alkalmaztuk, egy képponthoz nem egy bitet, hanem egy byte-ot rendeltünk, így a kapott ábra pixelei 256 különböző színnel rendelkezhetnek. Ez az ábra is ismétlődésektől, mintázatoktól mentes és egyöntetűen szürke kell, hogy legyen, tehát nem lehetnek rajta pl. pirosas vagy kékes árnyalatú foltok sem. Az ismertetett mintára az **5. ábra** bal oldalán látható, 256×128 pixel méretű, bitmap-formátumú képet kaptuk. Ugyanazon az ábrán a jobb oldali téglalap az ESP32-vel, 25 ms várakozási idővel generált számsorral kapott eredményt mutatja. Az ábrák vizsgálatával meggyőződhetünk arról, hogy a nyert számsorok valóban egyenletes szórásúak.

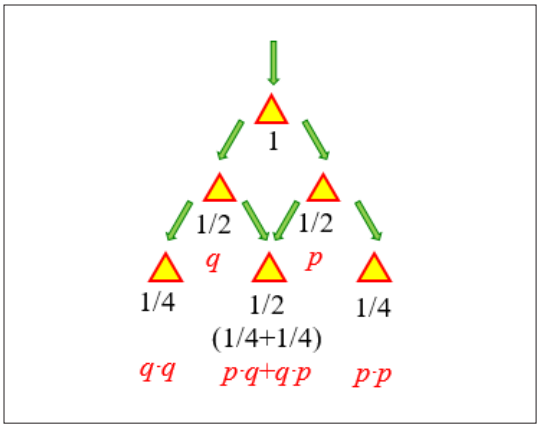
4. A Galton-deszka mint didaktikai eszköz

A véletlen folyamatok intuitív szemléltetésére Francis Galton angol polihisztor 1889-ben egy szerkezetet épített, amelyet Galton-deszkaként ismerünk. Ez tulajdonképpen egy ferdén elhelyezett sík lap, amelybe sakkmintaszerűen szegeket vert be, egymástól pontos távolságra. A lap alján tartórekeszeket alakított ki, amelyekben a felülről, ugyanabból a pontból elindított korongok vagy golyók gyűlnek össze.

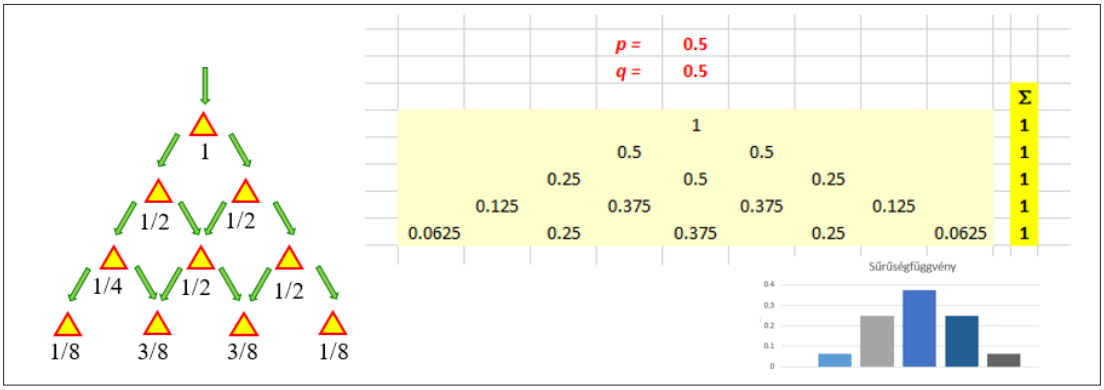
A korong lefelé csúszása közben beleütközik az első szegbe, ami megakadályozza a további szabad lecsúszását. Itt tehát az pattan egyet, és véletlenszerűen, p valószínűséggel a szeg jobb vagy q valószínűséggel annak a bal oldalán fog tovább csúszni lefelé, a következő sor szegig. A „klasszikus” verzióban $p = q = 0{,}5$. Ekképpen, ha elég sok korongot csúsztatunk lefelé, akkor azoknak kb. a fele fog a második sor első szegére jutni, és fele fog a második szegre esni.

A második sorban a jobbra vagy balra történő kitérés megismétlődik. A jobbra-balra kitérés valószínűsége ugyanaz, így valamely lehetséges útvonalon való továbbhaladás valószínűsége feleződik. A második sor középső szegét azonban a lefelé csúszó korong két különböző úton is elérheti, így annak a valószínűsége, hogy oda jusson, a két megfelelő valószínűség összege lesz (**6. ábra**).

A harmadik és a további sorokban kialakult helyzet bonyolultabbá válik. A magyarázathoz szükséges számításokat pl. a Pascal-háromszögnek Excelben való felépítésével tehetjük áttekinthetőbbé (**7. ábra**).



6. ábra. *A második szeggel való ütközés*



7. ábra. *További szegekkel való ütközés*

5. A virtuális Galton-deszka

A szemléltetőeszközünk egy virtuális Galton-deszka (**8. ábra**). Valójában ez egy kézzel fogható eszköz, amelyen a valódi deszkán végbemenő jelenséget szimuláljuk. A készülék „lelke” egy Arduino Mega 2560-as fejlesztőpanel, amelyhez egy 32×32-es RGB LED-mátrixot csatlakoztattunk. A „korong” egy pixel (a panelen egy színes LED), amely felülről lefelé állandó (a felhasználó által módosítható) sebességgel „esik”. Ha az esés közben egy „szeghez” ér (amely szintén egy pixel), akkor egy véletlenszám-generátor által előállított szám segítségével eldöntjük, hogy az merre térjen ki (0 – balra, vagy 1 – jobbra).

A korongok balra-jobbra kitérésének a valószínűsége az eredeti Galton-deszkának megfelelően lehet 0,5, azonban a továbbfejlesztett verziónkban azt a stand dőléséhez is igazíthatjuk, amivel az általános binomiális eloszlást is tanulmányozhatjuk, nem csak a szimmetrikus, $p = 0{,}5$-nek megfelelő esetet. Például ha megemeljük az eszköz bal oldalát, akkor a korongok inkább jobb oldalra fognak esni, az eloszlásuk pedig ferde lesz.

A standnak a függőleges helyzettől való eltérésének α dőlésszögét egy analóg, MMA7261QT-típusú gyorsulásmérő érzékelővel mérjük meg.

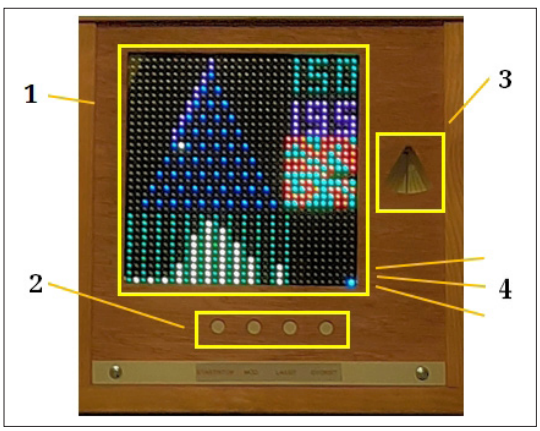
Ha a Galton-deszka nincs elbillentve, akkor a korongok egyforma valószínűséggel pattannak jobbra vagy balra. Ha a deszkát megbillentjük, akkor ez a valószínűség megváltozik:

$$p = 0{,}5 + \sin \alpha.$$

E képletnek nyilván csak akkor van értelme, ha α a [–30°, +30°] intervallumon van; ha $\alpha < -30°$, akkor $p = 0$, ha pedig $\alpha > +30°$, akkor $p = 1$.

A tartóban csak korlátozott számú korong férne el, így ha az feltelik, akkor az oszlopok magasságát skálázással csökkentenünk kell. A korongok számát tehát egy idő után már nem a világító pixelek száma jelenti.

Minden századik leguruló korong után az Arduino a soros porton keresztül kiküldi a tartóban összegyűjtött korongok számát, amelyet például az Arduino IDE program „Serial monitor” ablakából kimásolhatunk további elemzések elvégzése céljából: megrajzolhatjuk a gyakoriságok hisztogramjait, kiszámíthatjuk az eloszlás empirikus átlagát és szórását, statisztikai próbákat végezhetünk el.



8. ábra. *A virtuális Galton-deszka*
1- 32×32-ES led-panel
2- Kezelőgombok, sorban: start/stop, üzemmód-beállítás, animáció lassítása, animáció gyorsítása
3- Inklinométer, amely a billenés szögének szinuszával kiszámított ***p*** *valószínűségét mutatja*
4- Üzemmód-indikátor:
- felső led: „igazi” véletlenszám
- középső led: pszeudo-véletlenszám
- alsó led: a szög által befolyásolt pszeudo-véletlenszám

6. Következtetések

Az egyszerűen megépíthető hardveres véletlenszám-generátorokról szóló értekezések gyakran utalnak a mikrokontrollerek lebegő (be nem kötött) bemeneteinek zajára, amelynek a mintavételezésével véletlenszerűen alakuló adatokat kapunk. Amennyiben egy mikrokontrolleres rendszer programozásában van szükségünk a véletlen számokra, akkor az ötlet megvalósításához nincs is szükségünk külön eszközökre, csupán a véletlen számot előállító kódot kell megírnunk. A tapasztalatunk szerint azonban egy bemenet állapota, legyen az analóg vagy digitális, habár véletlenül, de túl lassan változik, így az egymást követő véletlen adatok szórása nem lesz kielégítő. Emiatt az alapötletet úgy fejlesztettük tovább, egy analóg bemenet mintavételezésével kapott véletlen értékek utolsó bitével dolgozunk, és ha egy adott intervallumból vett véletlen számokra van szükségünk, nem pedig egyszerű „igen/nem" kimenetekre, akkor az így nyert bitek sorozatából állítjuk össze azokat.

A tapasztalat azt mutatta, hogy ha az analóg bemenetek mintavételezése túl nagy frekvenciával történik, akkor a kapott eredmény nem a kívánt lesz: valószínűleg a mintavételező áramkörnek nem lesz ideje az alaphelyzetbe való visszatéréshez. Ha véletlen számok hosszú sorozatát szeretnénk előállítani, a minták vétele közötti szünet beiktatása miatt a véletlen bitek egyetlen analóg bemenetről való beolvasása egy hosszadalmasabb folyamat lenne. Emiatt a véletlen szám egy byte-jának megállapítására nem egy, hanem nyolc bemenet állapotát olvassuk be egyszerre (tulajdonképpen egymás után, de beiktatott megszakítások nélkül).

Az így kapott számok egyenletes eloszlásúnak mutatkoztak. Ezekből kiindulva normál eloszlású számokat a Box–Müller-eljárással tudunk előállítani.

Szakirodalmi hivatkozások

[1] Mcloughlin C., Krakowski K.: T*echnological Tools for Visual Thinking: What Does the Research Tell Us?* Paper presented at the Apple. University Consortium Academic and Developers Conference, James Cook, 13.1–13.12 (2001)

[2] Khasanovna U. N.: *The Importance of Didactic Tools in the Teaching of Educational Science*. Academicia Globe: Inderscience Research, 2. (2021) 76–79.

[3] Johnston D.: *Random Number Generators—Principles and Practices: A Guide for Engineers and Programmers*. Walter de Gruyter GmbH & Co KG, 2018

[4] *** Arduino Mega Rev. 3. Documentation, https://docs.arduino.cc/hardware/mega-2560

[5] *** ESP32 Reference, https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/index.html

[6] *** Arduino Language Reference, https://www.arduino.cc/reference/en/