Aleatorietat en la calculadora CASIO fx-82MS

Autor: Lobatchewsky

"Lo que llamamos azar es nuestra ignorancia de la compleja maquinaria de la causalidad."

Jorge Luis Borges

ÍNDEX DE CONTINGUTS

0. INTRODUCCIÓ

Des de l'ESO, sovint en la resolució d'exercicis que comporten càlculs hem hagut de fer ús d'una calculadora científica. Aquesta calculadora, que en la majoria d'alumnes es tracta d'una CASIO model fx82, ens ha servit per a fer una àmplia varietat d'operacions: sumes, restes, multiplicacions, potències, càlcul de raons trigonomètriques... amb números amb els quals no sempre és agradable tractar.

Il·lustració 1: Casio fx-82MS.

Font: https://upload.wikimedia.org/wikipedia/commons/e/e8/Casio_fx-82MS.jpg

Però aquesta calculadora no només pot fer operacions amb números: podia generar números. Només se'ns va parlar d'aquesta possibilitat com una curiositat, sense entrar en com i per què la calculadora tenia aquesta funció ja que no era part del temari.

Generar nombres aleatòriament no és com calcular sumes o restes. Les operacions com les sumes i les restes, donats els nombres a operar, sempre donen el mateix resultat, i això és d'esperar d'una màquina. «Inventar» nombres no, en canvi, i tot i això una simple calculadora escolar ho podia fer. Això ho vaig trobar força interessant.

En realitat, una calculadora o un ordinador no pot inventar nombres, però pot generar-ne de manera que no s'observi cap patró a un primer cop d'ull, però hi ha d'haver un patró, una relació entre un nombre i el següent amb la qual s'ha programat la calculadora per què generi aquests nombres. És a dir, es pot desxifrar la propietat que té la sèrie de nombres a partir de la mateixa sèrie de nombres, tot i que això no sempre és fàcil.

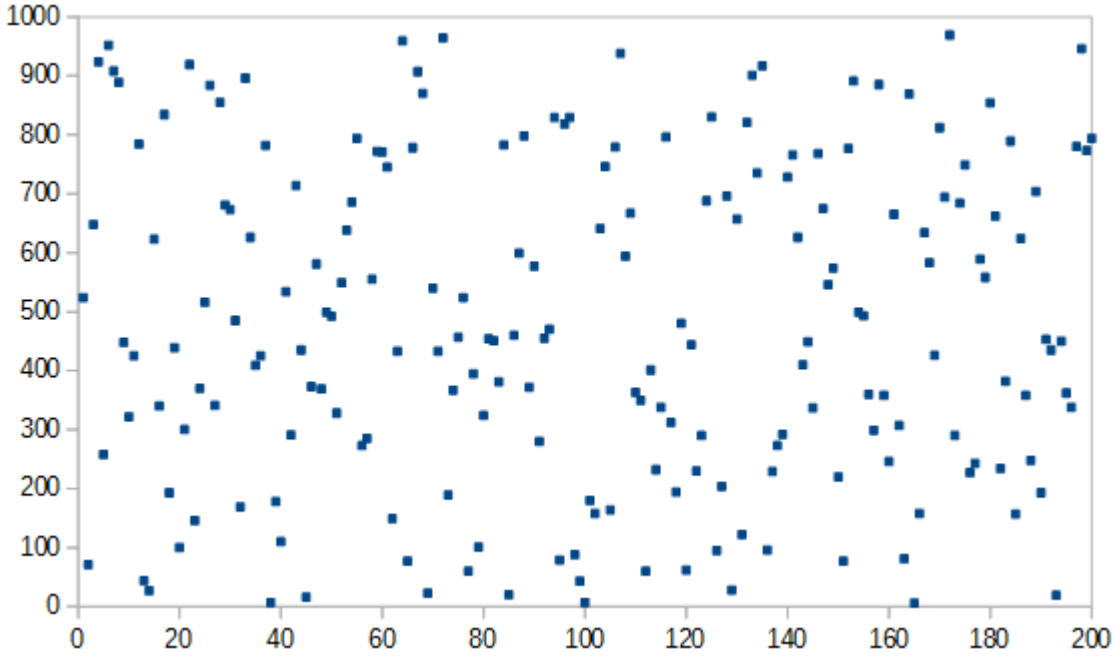
Aquesta és la motivació d'aquest projecte: reunir informació sobre la generació de nombres aleatoris per intentar desxifrar l'algoritme que genera la calculadora en prémer el botó *Ran#.*

1. ALEATORIETAT EN LA CALCULADORA CASIO FX-82MS

Com he dit, l'objectiu d'aquesta recerca és fer un anàlisi dels nombres obtinguts a partir de la calculadora. Per a fer això, òbviament, necessitem una sèrie de nombres generats per la calculadora. El procediment és simple: he pres la tecla *Ran#* 200 vegades i he anotat els valors, que són els següents:

```
[524, 71, 648, 924, 258, 952, 908, 889, 448, 322, 425, 784, 44, 27, 623,
340, 834, 193, 439, 100, 300, 919, 146, 370, 516, 884, 341, 855, 681, 673,
485, 169, 896, 626, 409, 425, 782, 7, 178, 110, 534, 291, 714, 435, 16,
373, 581, 369, 499, 492, 328, 549, 638, 686, 794, 273, 285, 555, 772, 771,
745, 149, 433, 959, 77, 778, 907, 870, 23, 540, 433, 964, 189, 367, 457,
524, 60, 395, 101, 324, 454, 451, 381, 783, 20, 460, 599, 798, 372, 577,
280, 455, 470, 829, 79, 818, 829, 88, 43, 7, 180, 158, 641, 746, 164, 779,
938, 594, 667, 363, 350, 60, 401, 232, 338, 796, 312, 194, 480, 62, 444,
230, 290, 688, 830, 95, 204, 696, 28, 657, 122, 821, 901, 735, 917, 96,
229, 273, 292, 728, 766, 626, 410, 449, 337, 768, 675, 546, 574, 220, 77,
777, 891, 499, 493, 360, 299, 885, 358, 246, 665, 307, 81, 869, 6, 158,
634, 583, 426, 812, 694, 969, 290, 684, 749, 227, 243, 589, 558, 854, 662,
234, 382, 789, 157, 624, 358, 248, 704, 193, 453, 435, 19, 450, 362, 338,
780, 946, 773, 794]
```

Aquesta llista de punts ens no ens dóna cap informació a primera vista, però podem provar si, en representant-les en un gràfic, obtenim alguna pista del generador.



Il·lustració 2: Representació de la sèrie de nombres obtinguts de la CASIO fx-82MS.

Font: elaboració pròpia

Novament no tenim cap resultat concloent: constatem que no tenim cap tipus de coneixement de l'algoritme o funció que els ha generat. Això suggereix una pregunta: és possible que els números siguin completament aleatoris? Per a respondre a aquesta pregunta, primer hem d'intentar deixar clara la definició d'aleatorietat.

1.1 QUÈ ÉS L'ALEATORIETAT?

Podem observar un fenomen impredictible, com el del llançament d'una moneda «justa». S'anomena així a una moneda on la probabilitat de que surti cara és igual a la probabilitat de que surti creu. Si assignem 1 a cara i 0 a creu tenim que:

$$p(0)=p(1)=\frac{1}{2}$$

Si a més a més tenim en compte que el resultat de llançar la moneda no està basat en l'anterior llançament, és obvi pensar que no podem predir el resultat del llançament següent. A més a més, observem que la probabilitat de totes les sèries obtingudes de *n* llançaments de moneda són equiprobables. Per exemple, les dues sèries de 10 nombres S_1 i S_2:

$$S_1=1001011110$$

$$S_2=1010100110$$

Són equiprobables, ja que com que la probabilitat de cada nombre *no* està condicionada pel nombre anterior, la probabilitat de cada sèrie ve donada pel producte de les probabilitats de cada nombre de la sèrie. Per a S_1 i S_2:

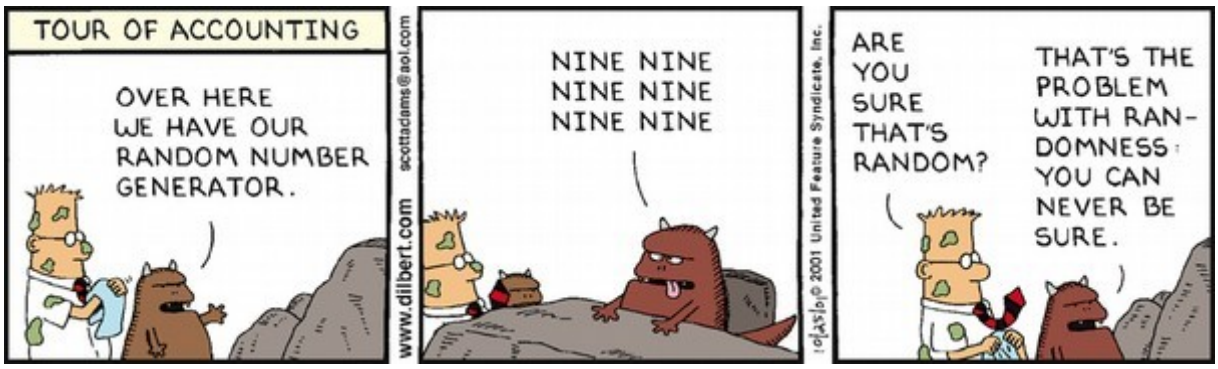
$$p(S_1)=p(1)\cdot p(0)\cdot p(0)\cdot 3\,p(1)...=\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}...=\frac{1}{2^{10}}\simeq 0{,}000977$$

$$p(S_2)=p(1)\cdot p(0)\cdot p(1)\cdot p(0)...=\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}...=\frac{1}{2^{10}}\simeq 0{,}000977$$

Com que les probabilitats són exactament iguals, no podem predir ni un resultat ni l'altre. Per tant, podríem definir aleatorietat de la següent manera:

Definició 1: Una seqüència de nombres és aleatòria quan no podem predir el nombre següent en la successió.

No obstant això, podem considerar la situació següent:



Il·lustració 3: Vinyeta que il·lustra la problemàtica que comporta la definició 1.

Font:
https://www.incibe.es/blogs/post/Seguridad/BlogSeguridad/Articulo_y_comentarios/comprobando_aleatoriedad

És a dir, podem considerar una nova seqüència de 10 nombres de l'estil:

$$S_3 = 0000000000$$

Que obtindrem amb una probabilitat:

$$p(S_3) = p(0)\cdot p(0)\cdot p(0)\cdot p(0)\ldots = \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}\ldots = \frac{1}{2^{10}} = p(S_1) = p(S_2)$$

I per tant tindrà el mateix grau d'impredictibilitat que S_1 i S_2, ja que en aquest cas tampoc podem predir el següent nombre. No obstant això, si obtinguéssim aquest resultat no només de llençar una moneda, sinó de qualsevol experiment possible, no podríem pensar que en realitat hi ha un patró? En realitat, moltes vegades aquests patrons són significatius i donen informació representativa del fenomen que s'està estudiant. Per això, ens és interessant el concepte de *complexitat de Kolmogorov* o *complexitat algorítmica.*

Definició 2: La complexitat d'una seqüència de nombres és la llargada del mínim algoritme que la descriu.

Aplicant aquesta definició a les sèries anteriors:

- S_1: un 1, dos 0s, un 1, un 0, quatre 1s i un 0.
- S_2: dues vegades un 1 i un 0, un 1, dos 0s i dos 1s i un 0.
- S_3: deu 0s.

Observem que aquesta definició s'assembla més al concepte intuïtiu d'aleatorietat que nosaltres tenim deixant S_3 com la sèrie més simple.

Tot i això es pot demostrar que la *complexitat de Kolmogorov* no és computable[1] i per tant de vegades és difícil dir si una sèrie és «més aleatòria» que una altra.

Existeix també la definició d'aleatorietat de Per Martin-Löf, inspirada per la definició de Kolmogorov:

Definició 3: Una seqüència és aleatòria si:

- Passa tots els possibles tests estadístics (prenent el llançament d'una moneda com a cas ideal).
- No pot ser produïda per un algoritme més curt que la pròpia sèrie.

Aquesta definició deixa, per exemple, el nombre π fora de qualsevol consideració d'aleatorietat. Per exemple, podríem definir la seqüència de les primeres 1000 xifres de π, i tot i que no observaríem cap patró en les xifres, en qualsevol moment podem esbrinar el nombre següent en la sèrie, i per tant no la podem considerar fortuïta.

Tornant al problema de la calculadora, a partir d'ara quan parlem sobre aleatorietat ho farem referint-nos a la definició de Martin-Löf, i conseqüentment intentarem trobar algun patró o regularitat mitjançant tests estadístics.

1 Veure demostració a https://en.wikipedia.org/wiki/Kolmogorov_complexity#Uncomputability_of_Kolmogorov_complexity

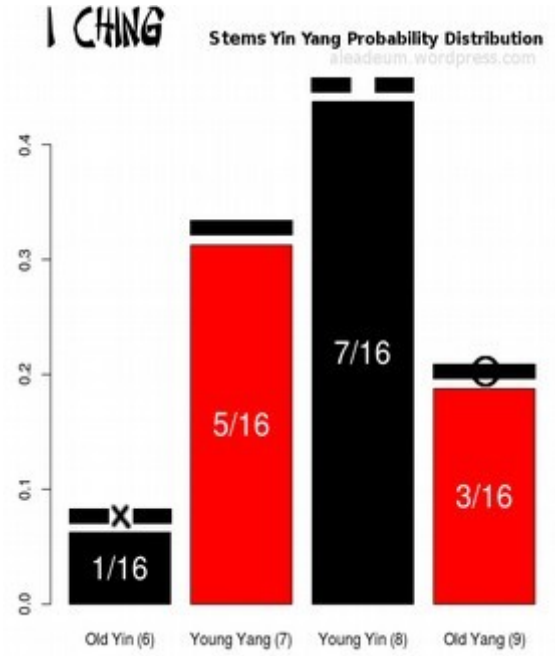
2. ALGORITMES DE MESURA DE L'ALEATORIETAT

Després d'haver-nos aproximat a una definició més rigorosa del concepte de l'aleatorietat, ara passarem a intentar-la mesurar.

La definició que ens és més útil per a la nostra tasca és la definició de Martin-Löf, ja que podem idear tests estadístics per intentar trobar patrons o algun tipus d'informació d'alguna sèrie de nombres, per exemple, l'obtinguda a partir de la de la calculadora. Però que ha portat a la investigació matemàtica a idear aquestes proves? És a dir, per què ens pot interessar trobar patrons en sèries de nombres?

2.1 HISTÒRIA DE L'ALEATORIETAT

El concepte d'aleatorietat ha estat motiu de reflexió del pensament humà ja des de l'antiguitat. A l'antiga Grècia, s'assumia que tots els esdeveniments succeeixen segons alguna o algunes lleis determinades, sense cabuda per a l'arbitrarietat en cap de les seves manifestacions (determinisme). En canvi, anteriorment, a la Xina de fa 3000 anys, ja es feien estudis sobre l'aleatorietat escrits en el I Ching, un llibre xinès que data del 1150 abans de Crist, tot i que la filosofia xinesa es va centrar en els aspectes no matemàtics de l'aleatorietat en aquella època.



Il·lustració 4: Distribució del ying i el yang segons el I ching. La flosofia xinesa es va centrar en els aspectes no matemàtics de l'aleatorietat.

Font: https://aleadeum.com/2013/07/12/the-i-ching-random-numbers-and-why-you-are-doing-it-wrong/

Arreu del món es creia que estava relacionada amb el destí i que per tant l'atzar podia ajudar a predir el futur: els xinesos analitzaven esquerdes en closques de tortugues, a occident es llençaven els daus. Com a anècdota, Juli Cèsar va llençar els daus abans de creuar el riu Rubicone, a Itàlia, en el seu camí a Roma. En fer-ho va trencar la llei causant un conflicte armat. Va ser llavors quan va pronunciar la famosa frase «ālea iacta est», que significa «el dau ha estat llençat».

Però aquesta creença popular no va trobar el seu lloc en la ciència fins molts segles més tard, quan al voltant de l'any 1700 Isaac Newton va formular un model físic teòric determinista, que suggereix que petites discrepàncies entre el model i valors obtinguts de la realitat provenen d'errors en la mesura.

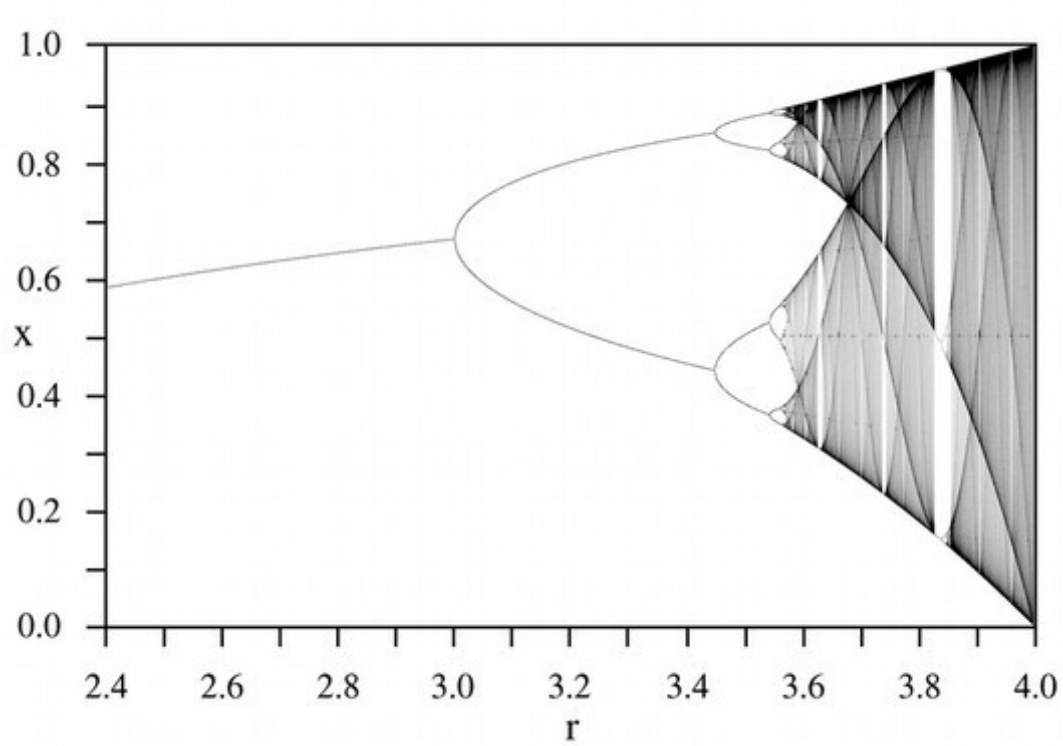
Il·lustració 5: Retrat d'Isaac Newton, creador d'un model físic determinista.

Font: https://upload.wikimedia.org/wikipedia/commons/5/50/Sir_Isaac_Newton_by_Sir_Godfrey_Kneller,_Bt.jpg

Tot i això, a la natura ja s'havien observat fenòmens que semblaven estar definits més aviat arbitràriament, sobretot en matemàtiques: les xifres del nombre π, arrels, logaritmes... No va ser fins a principis del segle XX que es va observar que

l'aleatorietat regeix el món de la física quàntica, fins al punt que aquest només pot ser estudiar mitjançant el càlcul de probabilitats.

Una altra mostra d'arbitrarietat en el comportament de la natura, és el cas dels «sistemes caòtics». Es tracta de sistemes físics deterministes que depenen molt sensiblement de les variables inicials, com en el problema dels *n* cossos, que intenta determinar les trajectòries d'un nombre *n* de cossos en ser moguts només per les forces de gravetats creats per les seves pròpies masses. Va ser estudiat ja al voltant de l'any 1880, per Henry Poincaré. Un altre exemple és el del mapa logístic, on amb una equació bastant simple, es generen gràfics com el següent[2]:



Il·lustració 6: Mapa logístic, exemple de caos determinista.

Font: https://en.wikipedia.org/wiki/Bifurcation_diagram#/media/File:LogisticMap_BifurcationDiagram.png

Més enllà del joc i de la presa de decisions «justes», l'aleatorietat no ha estat aprofitable durant els últims segles. Però a partir del segle XX, amb l'aparició dels

2 Per a més informació veure https://en.wikipedia.org/wiki/Logistic_map

primers ordinadors i amb l'incentiu de la Segona Guerra Mundial i la posterior Guerra Freda, es van començar a utilitzar mètodes computacionals, alguns basats en nombres generats a l'atzar.

Entre les aplicacions més importants en la actualitat, trobem la modelització. Es tracta de trobar un model més o menys simple d'un fenomen que pot ser bastant complex. A partir d'aquest model, podem fer una simulació: estudiar el fenomen més a fons a partir de modificar les variables que defineixen el model. La modelització i la simulació són dos dels principals usos de la estadística actualment, i són àmpliament utilitzades en física i economia. Una altra aplicació és el càlcul aproximat d'equacions diferencials i en la integració de funcions, on entrarem més en detall més endavant.

2.2 TEST DE RUNS

La primera prova que farem a la nostra sèrie de nombres és una de les proves del NIST (National Institute of Standards and Technology).

Una de les aplicacions més importants que té l'aleatorietat en l'actualitat és en criptografia. El seu ús adequat és clau en la seguretat d'aplicacions informàtiques i sistemes operatius, i per tant l'ús d'una seqüència que no proporcioni el grau d'impredictibilitat necessari porta a una vulnerabilitat en la seguretat del sistema.

Això és el que va passar el 2006 amb la seguretat del sistema operatiu Debian i derivats, quan es va utilitzar una seqüència suposadament aleatòria sense haver provat la seva qualitat. Aquest va ser l'incentiu que va fer que el NIST dissenyés un seguit de 15 proves per tal de determinar la qualitat d'aquesta seqüència. Si una seqüència passava tots aquests tests podia considerar-se aleatòria (observem que es va utilitzar el criteri de Martin-Löf).

D'aquestes 15 proves, la primera s'anomena «test de runs», i és la que aplicarem a la nostra sèrie.

L'algoritme consisteix en separar la mostra en dues parts iguals: per una banda els que estan per sota de la mitjana aritmètica (als quals assignem un 0) i els que estan per sobre (als quals assignem un 1). Un cop feta l'assignació, es compta el nombre de vegades que es canvia de 0 a 1 i es compara amb el que caldria esperar d'una sèrie sense cap patró.

Per exemple, si comptéssim el número de runs de S_1 i de S_2:

$$S_1 = 1001011110\,;\, runs(S_1) = 6$$

$$S_2 = 1010100110\,;\, runs(S_2) = 8$$

Aquesta és la sèrie binària equivalent a la obtinguda de la calculadora[3]:

```
[1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0,
0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1,
0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0,
0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1,
0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1,
0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,
1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,
1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
```

Que té un total de 97 runs. Però que significa que tingui 97 runs? És bo o dolent per a algú que vulgui trobar patrons en la sèrie de la calculadora?

Es pot demostrar que si la seqüència és aleatòria el número de runs es distribueix segons una variable normal de mitjana μ i variància σ^2 tal que

$$\mu = \frac{2n_1 n_2}{n_1 + n_2} + 1$$

$$\sigma^2 = \frac{2n_1 n_2 (2n_1 n_2 - n_1 - n_2)}{(n_1 + n_2)^2 (n_1 + n_2 - 1)}$$

On n_1 i n_2 són el número d'1s i de 0s, indistintament. Aquesta distribució es denota per $N(\mu,\sigma^2)$.

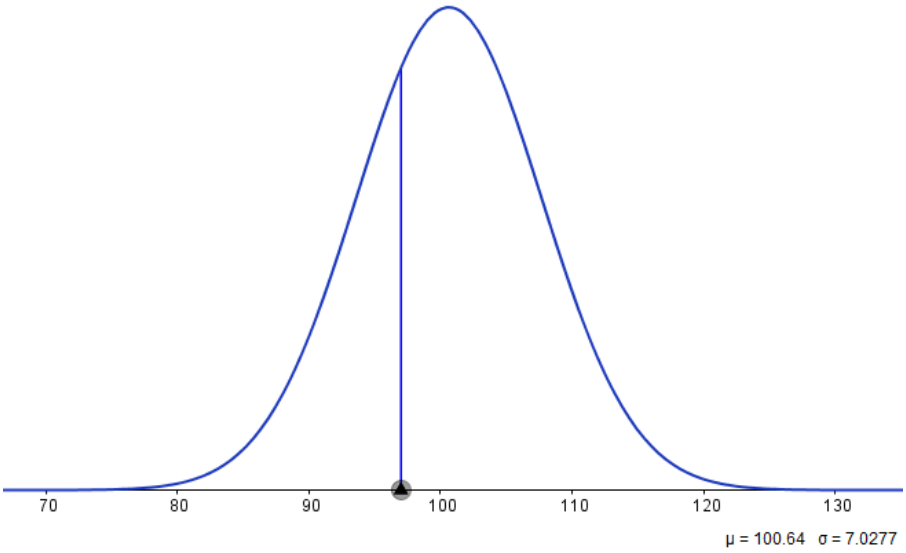
Per a la nostra sèrie, els valors obtinguts són

$$\mu = 100.64$$

$$\sigma^2 = 7.02775888708$$

3 Veure *Programa 1* a l'annex.

Que unes dades es distribueixen de manera normal o d'una altra forma, vol dir que en ser representades dibuixen gràfic característic. En el cas de la corba normal, el gràfic és el següent:



Il·lustració 7: Corba de distribució normal.

Font: elaboració pròpia

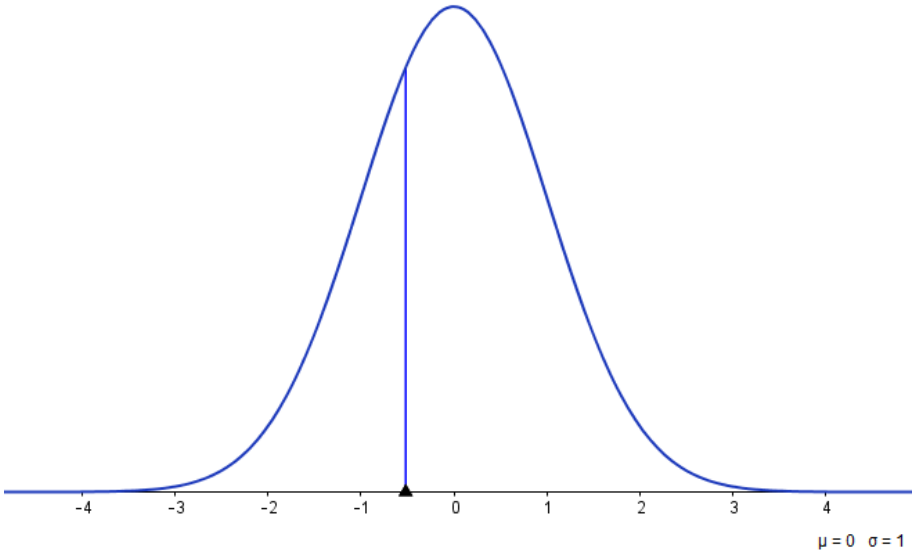
El gràfic representa la probabilitat de cada número de runs. Observem que el valor 97 és bastant probable, és a dir, no indica cap peculiaritat de les dades que hem estudiat.

Aquests càlculs es poden estandarditzar prenent $\mu=0$ i $\sigma^2=1$. Si anomenem $X\sim N(\mu,\sigma^2)$ a les dades de la nostra sèrie i Z a les dades de la sèrie equivalent normalitzada ($Z\sim N(0,1)$), podem transformar X en Z segons:

$$Z=\frac{X-\mu}{\sigma^2}$$

Podem fer aquest calcul amb el nombre de runs, obtenint $Z=-0.517946056273$

Llavors, la corba de distribució corresponent és la següent:



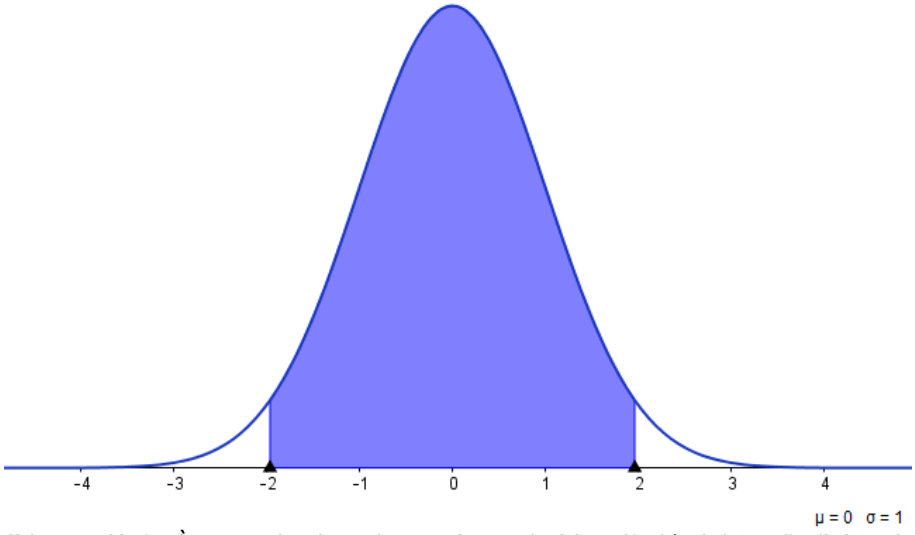
Il·lustració 8: Corba de distribució normal estandaritzada.

Font: elaboració pròpia.

Qualsevol dels dos gràfics ens serveix per a veure que les dades de la calculadora passen el test de runs, ja que se situen bastant prop del que cal esperar d'una sèrie de dades sense cap patró o peculiaritat que la defineixi. Per estimar com de bona és aquesta conclusió podem plantejar la següent pregunta: on hauria d'estar un número de runs que no passés el test de normalitat?

Això és el que es coneix com a nivell de confiança, i per a una distribució normal aquests valors estan estandarditzats. En el nostre cas, el nivell de confiança és la probabilitat que correspon a cada número de runs d'acord amb la distribució normal. Aquesta probabilitat correspon a l'àrea sobre la corba de distribució centrada en la mitjana.

En la majoria d'anàlisis estadístics s'accepta com a bo que el resultat estigui entre el 95% dels resultats més probables. En una distribució normal *N(0,1)* aquest percentatge correspon a l'àrea corresponent a l'interval *[-1,96,1,96].*



Il·lustració 9: Àrea amb els valors més probables (95% del total) d'obtenir d'una distribució normal

Font: elaboració pròpia

Observem que el nostre valor de *Z* està lluny dels valors límit de l'interval, i per tant les nostres dades encaixen molt bé en aquesta distribució. Per tant, confirmem que la sèrie que hem analitzat no presenta cap singularitat pel que fa a la relació entre dades més grans i més petites que la mitjana de la sèrie, tal com caldria esperar d'una sèrie aleatòria.

2.3 El mètode Montecarlo

La relació entre dades per sobre i per sota de la mitjana passa el test de normalitat i per tant podem considerar la sèrie aleatòria en aquest sentit. Tot i això potser el mateix gràfic de les dades pot donar informació sobre aquestes.

Per exemple, el valor de runs que hem obtingut s'hauria pogut adquirir de diferents maneres. Sense pegar un cop d'ull a la representació de les dades, aquest número de runs s'hauria pogut obtenir a partir d'una sèrie que canvia molt ràpid entre 0 i 1 durant els primers números de la sèrie i més lentament durant els últims, la qual cosa ens dóna informació sobre la sèrie però no es veu reflectida en el test de runs. Per això és important una prova que ens permeti saber si aquests nombres es distribueixen de manera homogènia o si es concentren en una regió del gràfic, la qual cosa ens indicaria que la sèrie no és aleatòria.

Tot i que serveix com a test estadístic, el mètode Montecarlo va ser ideat per fer aproximacions numèriques d'expressions matemàtiques molt complexes (aproximació de càlcul integral, d'equacions diferencials...). Per a explicar el mètode, ho farem mitjançant el càlcul d'una aproximació numèrica del nombre π.

2.3.1 CÀLCUL APROXIMAT DE π

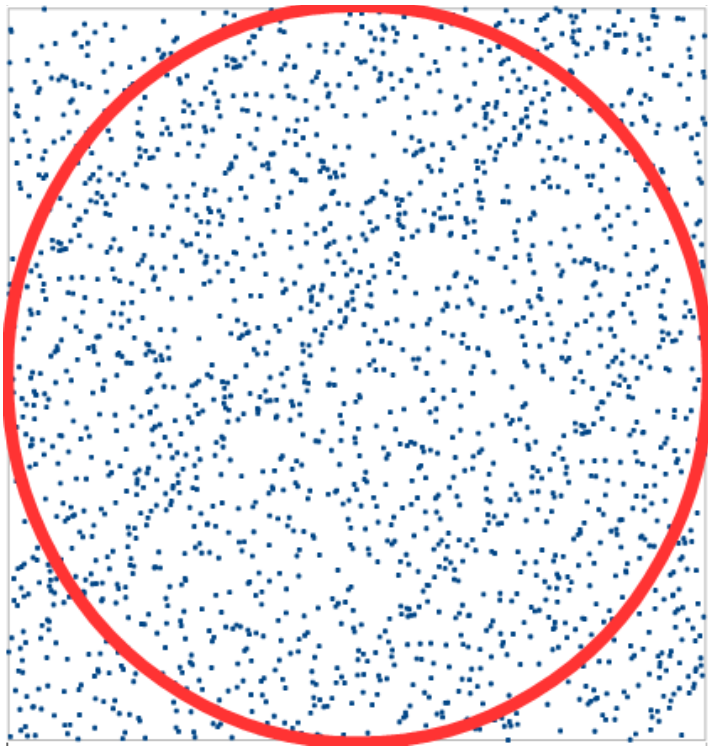
El mètode Monte Carlo es basa en la generació de nombres aleatòriament, però, com podem relacionar això amb el nombre π?

Bé, el nombre π és per definició el quocient entre el perímetre d'una circumferència i el seu diàmetre. Trobem també el nombre π en l'expressió de l'àrea d'una circumferència en funció del seu radi, i això és interessant per a dur a terme el Monte Carlo. Si tenim en compte que la distribució dels punts és homogènia, és a dir, en una mateixa àrea hi ha (aproximadament) el mateix número de punts, podem establir la següent relació:

$$\frac{n_c}{n_q} = \frac{S_c}{S_q}$$

On n_c i n_q són el número de punts en el cercle i en el quadrat i S_c i S_q les superfícies del cercle i del quadrat.

Tenint en compte això, podem definir un quadrat que limiti la posició dels punts i un cercle inscrit d'on obtenir el valor de π de la següent manera:



Il·lustració 10: Gràfic per visualitzar la proporció entre punts dins i fora del cercle.

Font: elaboració pròpia.

Així, si comptem el nombre de punts a l'interior del cercle n_c i el dividim entre el nombre de punts dins del quadrat n_q i multipliquem tot això per 4, obtindrem una aproximació de π.

$$\frac{n_c}{n_q}=\frac{S_c}{S_q}=\frac{\pi r^2}{(2r)^2}=\frac{\pi r^2}{4r^2}=\frac{\pi}{4}$$

Per fer el càlcul utilitzarem un generador de nombres pseudoaleatoris, el qual explicarem més en detall posteriorment. La sèrie de nombres depèn dels valors inicials dels paràmetres *a, c, m* i X_0 (anomenats *seed* o llavor). Per als valors

$$a=1561 \qquad c=1153 \qquad m=2^{15} \qquad X_0=62$$

Hem trobat que hi ha 25752 punts dins del cercle i 32768 en total. Llavors, segons la relació que hem deduït abans:

$$\pi\approx\frac{n_c}{n_p}=\frac{25752}{32768}=3{,}1435546875$$

Podem dir que aquest resultat és bo ja que té un error relatiu inferior al 0,07%. Per tant, podem concloure que la distribució de les dades que hem utilitzat és homogènia.

2.3.2 Aplicació del mètode Montecarlo com a prova estadística

En l'exemple anterior, hem utilitzat uns punts de distribució homogènia per tal d'obtenir una bona aproximació de pi, i així ha sigut. Ara ho farem a l'inrevés, a partir d'una sèrie que en principi no sabem com és, esbrinarem com és aquesta sèrie a partir de calcular π.

Per fer-ho, com que es tracta d'una sèrie de 200 números que van del 0 al 1000 i per tant estarien emmarcats en un rectangle, hem de dividir cada nombre de la sèrie entre 5 per tal de que els punts confirmen un quadrat. Un cop feta aquesta transformació i prenent 200 com a longitud dels costats del quadrat i per tant 100 com a radi de la circumferència, operem com en el cas anterior.

El resultat que obtenim és

$$\pi=\frac{n_c}{n_q}=\frac{165}{200}=3.3$$

Tot i que en principi no és tan bona com la sèrie pseudoaleatòria podem considerar que un error inferior al 5% és bo, ja que també influeix el nombre de punts utilitzats en el càlcul, que és significativament inferior en la sèrie de la calculadora.

Per tant, podem considerar que la sèrie de la calculadora passa també aquest test.

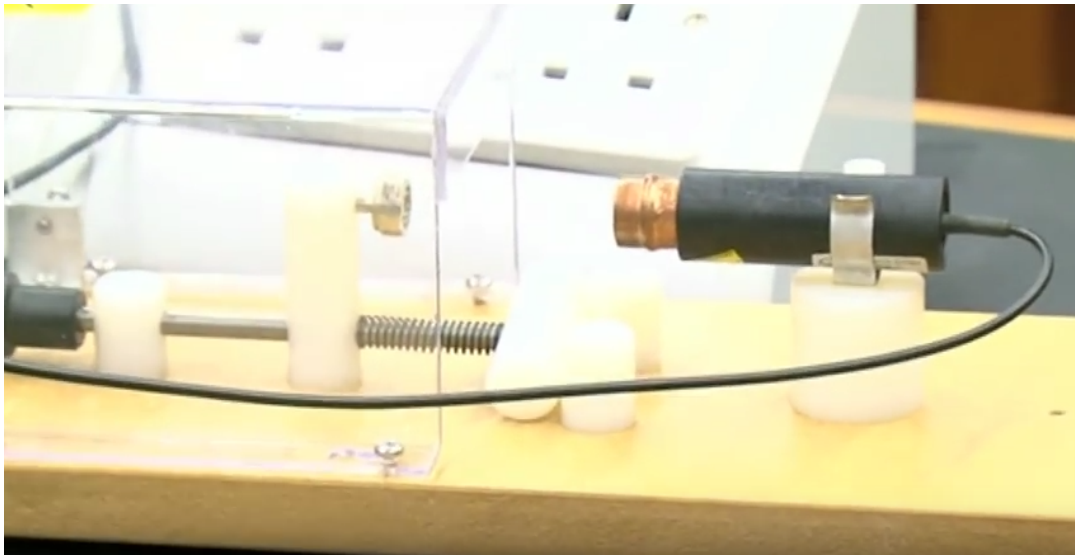
3. LA CALCULADORA GENERA NÚMEROS PSEUDOALEATORIS

Fins ara hem observat que la calculadora passa el test de runs i el test de Montecarlo, i per tant podem estar segurs de que sigui quina sigui el mètode que utilitza per generar-los, aquest és prou eficaç. Com que no hem tingut èxit analitzant les dades, ara buscarem quina pot ser la font dels nombres.

3.1 ALEATORIETAT I PSEUDOALEATORIETAT

Com deia en la introducció, pot realment una màquina «inventar-se» números a l'atzar? En principi no, ja que quan li demanem a un ordinador, màquina, calculadora... que generi nombres li hem de dir com fer-ho i per tant no pot actuar amb arbitrarietat.

Però els nombres no tenen perquè provindre de la calculadora: la naturalesa està plena fonts d'aleatorietat (anomenats TRNG, True Random Number Generation). L'exemple més comú és el llançament d'un dau, o d'una moneda, ja que són esdeveniments prou simples però sense absolutament cap manera de predir-ne el resultat, però tots dos tenen el mateix problema: els nombres s'obtenen massa lentament. Hi ha altres maneres que són igual de bones i ens donen un major nombre de resultats, com per exemple l'anàlisi del soroll de fons, o el nombre d'electrons emesos en un procés de desintegració radioactiva, que es considera un dels millors mètodes per obtenir nombres purament aleatoris.



Il·lustració 11: Muntatge per a obtenir nombres aleatòris a partir de la descomposició d'un isòtop radioactiu.

Font: captura de pantalla de Random Numbers – Numberphile
https://www.youtube.com/watch?v=SxP30euw3-0

Aquests mètodes són útils per a certs usos, com pot ser la modelització, però a vegades el ritme de generació d'aquests sistemes no és suficient, i per tant les dades obtingudes de la natura no són viables. En aquests casos es recorre als PRNG (PseudoRandom Number Generators), on s'utilitzen nombres generats mitjançant una fórmula matemàtica coneguda. Però que hagin estat generats mitjançant un mètode conegut no significa que no els puguem considerar en certa manera aleatoris: com hem comprovat en el càlcul de π, utilitzant un conjunt de punts d'un PRNG hem aconseguit una bona aproximació.

Com que la calculadora és de gama baixa podem assumir que no incorpora cap sensor i per tant considerarem que el generador es tracta d'un PRNG. A continuació veurem quins són els més utilitzats.

3.2 GENERADORS CONGRUENCIALS

Els generadors congruencials es basen en expressions de la forma:

$$X_{i+1}=f(X_i)\,mod\,m$$

on *m* indica el període de la sèrie i l'expressió «mod m» significa que X_{i+1} i $f(X_i)$ tenen el mateix residu en ser dividits per m.

L'estudi d'aquesta relació s'anomena aritmètica modular i va ser introduïda per primer cop per Carl Friedrich Gauss el 1801 en el seu llibre *Disquisitiones Arithmeticae.* Per entendre millor el funcionament d'aquest operador, podem provar amb un exemple numèric:

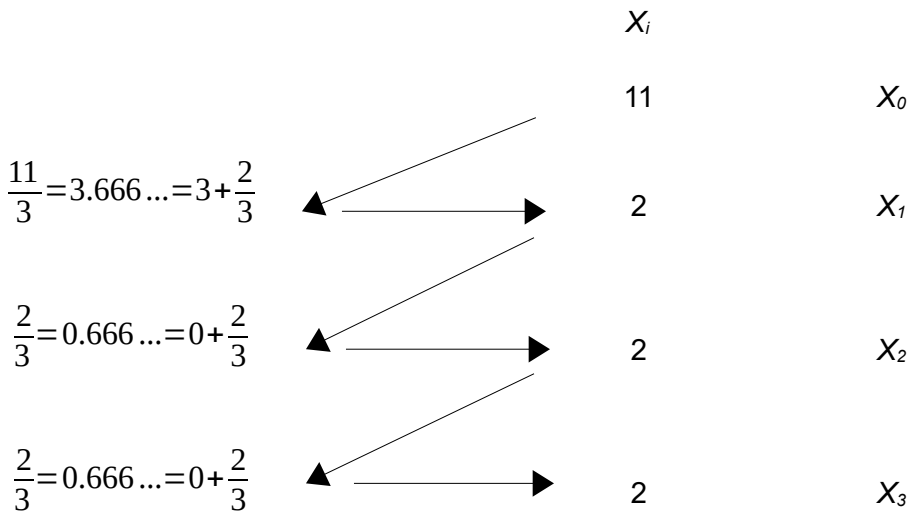
$$11\equiv 8\,mod\,3$$

Hem dit que això vol dir que 2 i 8 donen el mateix residu en ser dividits per 3. Ho comprovem:

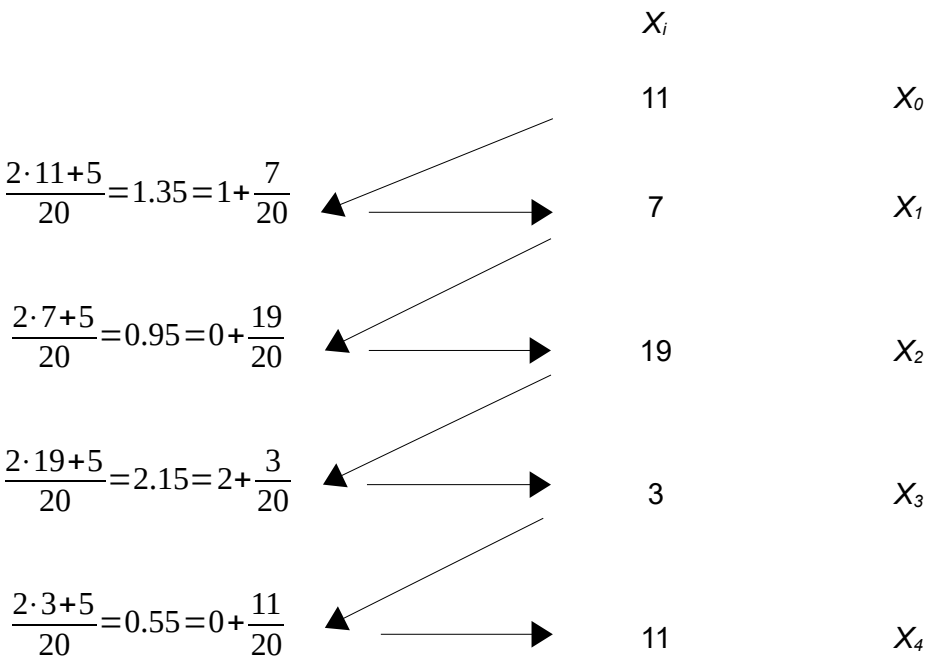
$$\frac{11}{3}=3.666...=3+\frac{2}{3} \qquad \frac{8}{3}=2.666...=2+\frac{2}{3}$$

Observem que efectivament, 11 i 8 tenen el mateix residu (2) en ser dividits per 3.

El residu d'una divisió sempre és un nombre natural, i per tant, podem dividir-lo novament. Obtindrem un altre residu que també podrem dividir, i així successivament fins a obtenir una seqüència de nombres:



Però aquest mètode sempre retorna el mateix nombre! Per això no s'utilitza X_i en el generador, sinó una un nombre que ve donat per una funció $f(X_i)$. Per exemple, si $X_0=11$, $f(X_i)=2X_i+5$ i $m=20$:



Obtenim aquests valors, que no semblen tenir en un primer cop d'ull cap relació. Observem que hem obtingut el valor inicial en operar unes quantes vegades, i si continuéssim operant, obtindríem la mateixa sèrie ja que cada nombre està basat en l'anterior.

Per aquesta relació que existeix entre un nombre de la sèrie i el següent, no podem dir que es generen nombres completament aleatoris. En altres paraules, donat qualssevol valor X que introduïm en l'expressió podrem trobar el valor X_{i+1} següent. Això sí, si tenim en compte que si utilitzem una funció simple de l'estil $f(X_i)=aX_i+c$ ja obtenim resultats amb un cert grau d'"aleatorietat", aquesta correlació entre un nombre i l'anterior no suposa un problema en moltes aplicacions.

Un altre possible problema que podem observar és que es genera una sèrie periòdica. Però si tenim en compte que un generador congruencial és un mètode especialment ràpid amb les eines computacionals actuals i que podem generar períodes de 2^{20} números sense gaire dificultat, això no suposarà un problema en la majoria d'ocasions.

3.2.1 GENERADOR CONGRUENCIAL LINEAL

Els Generadors Congruencials Lineals (sovint abreviats com GCL) són els que tenen la forma següent:

$$X_{i+1}=\left(a\,X_i+c\right)\,mod\,m$$

On

a és el multiplicador, $0\leq a<m$

c és l'increment, $0\leq c<m$

m és el mòdul, $m>0$

X_0 és la llavor, $0\leq X_0<m$

A més a més, hem de considerar els casos $a=0$ i $a=1$, ja que

- Si $a=0$

$$X_{i+1}=\left(0\,X_i+c\right) mod\, m=c\, mod\, m$$

és a dir, *X* només prendrà un valor.

- Si $a=1$

$$X_1=X_0+c$$

$$X_2=X_1+c=X_0+2c$$

$$X_i=X_0+ic$$

fent dubtosa l'aleatorietat d'aquesta sèrie.

Però per tal de maximitzar la "aleatorietat" de la sèrie haurem de tenir en compte altres factors sobre aquestes variables:

- *m* ha de ser com més gran possible ja que el període serà igual o menor que el mòdul.
- És convenient que *m* sigui de la forma $m=2^g$ per a optimitzar l'eficiència de l'algoritme.
- *c* i *m* han de ser primers relatius.
- *a* ha de ser divisible entre tots els factors primers de *m.*
- *a* ha de ser divisible entre 4 si m és múltiple de 4.

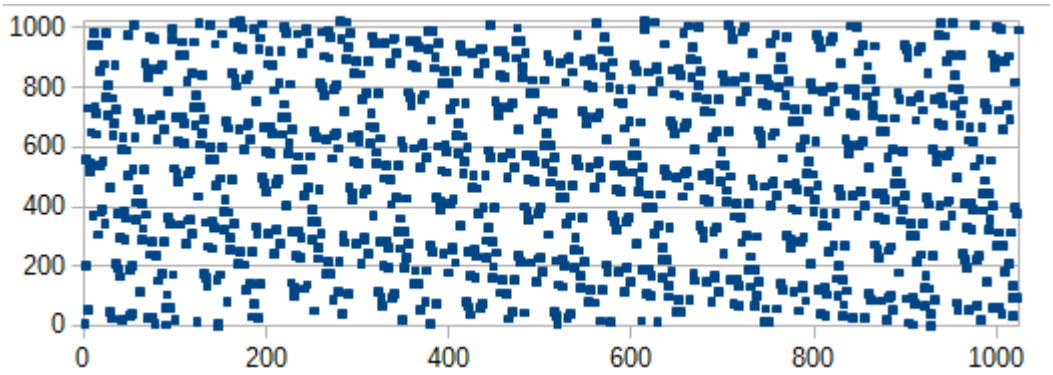
- A vegades s'utilitza el valor $c = 0$, ja que en suprimir una operació en l'algoritme, s'augmenta la velocitat d'aquest sense disminuir significativament la qualitat de la seqüència obtinguda. Aquest algoritme s'anomena Generador Congruencial Multiplicatiu (GCM).

Per visualitzar les propietats de la sèrie obtinguda observarem la distribució dels resultats en un gràfic. Per a generar les dades necessàries per a fer els gràfics, podem programar un algoritme basat en l'expressió del GCL.

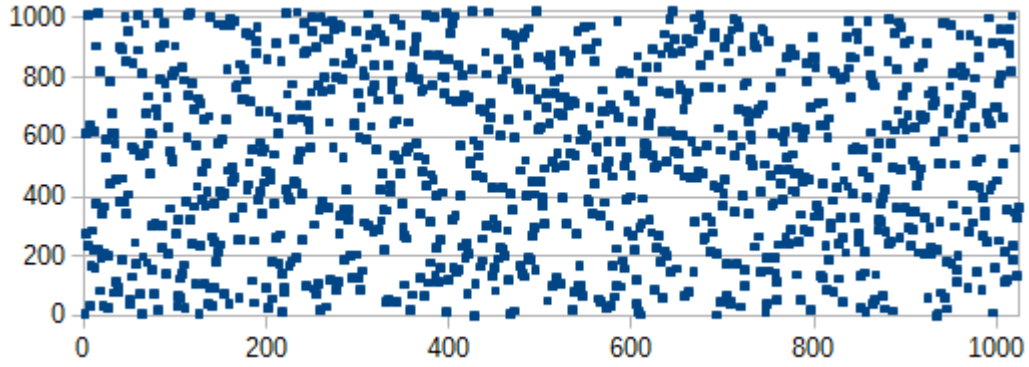
Anàlisi del paràmetre a

Per veure com afecta el paràmetre *a* a la distribució de les dades, donarem les altres variables uns valors fixos. Utilitzarem $m=2^g$ ja que es és l'expressió que s'utilitza més per la velocitat de càlcul que dóna. La seqüència obtinguda la podem importar a un full de càlcul, per a ser representada.

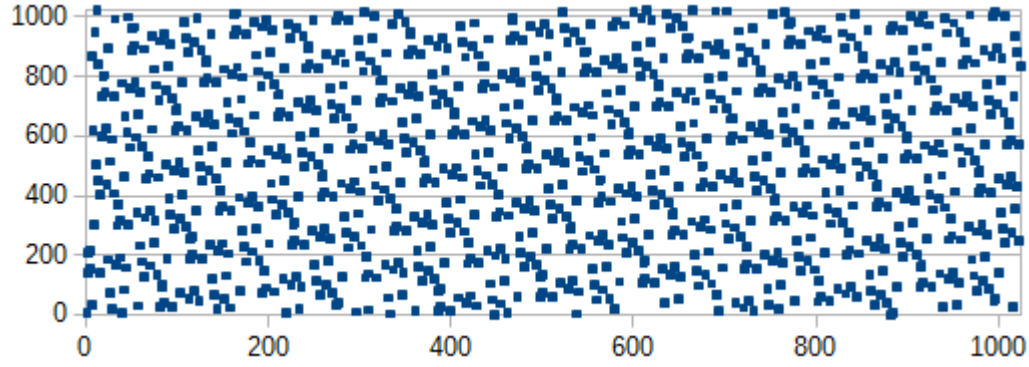
a =33



a = 101



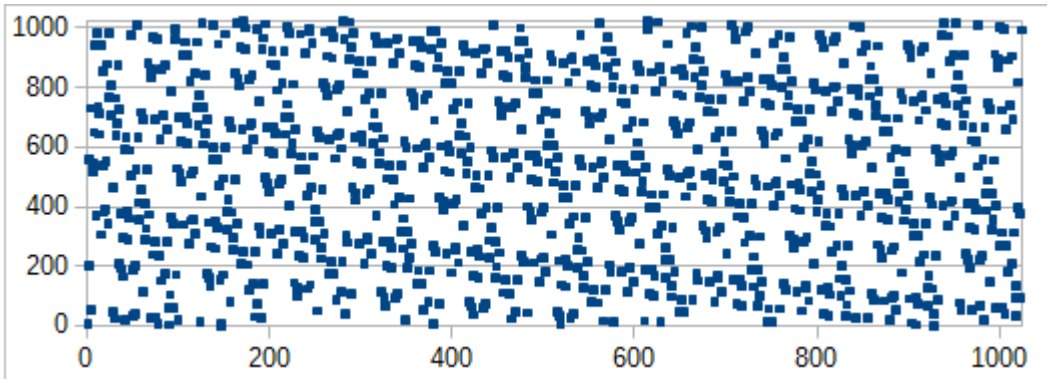
a = 193



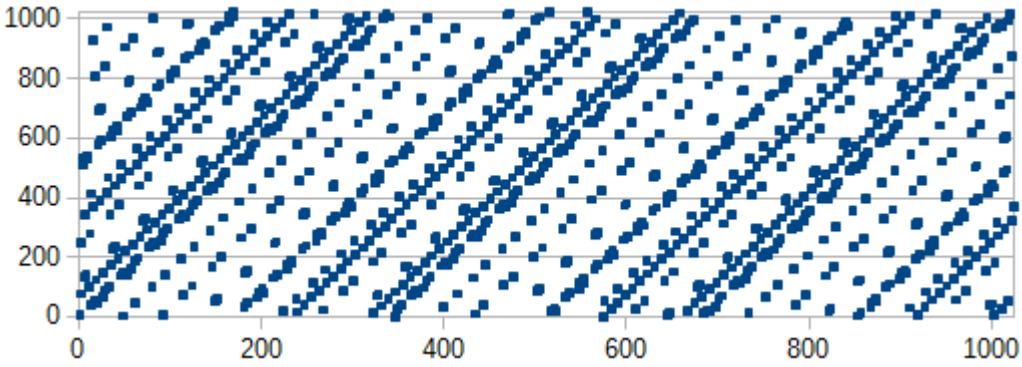
Anàlisi del paràmetre c

$a = 33$ $m = 1024$ $X_0 = 6$

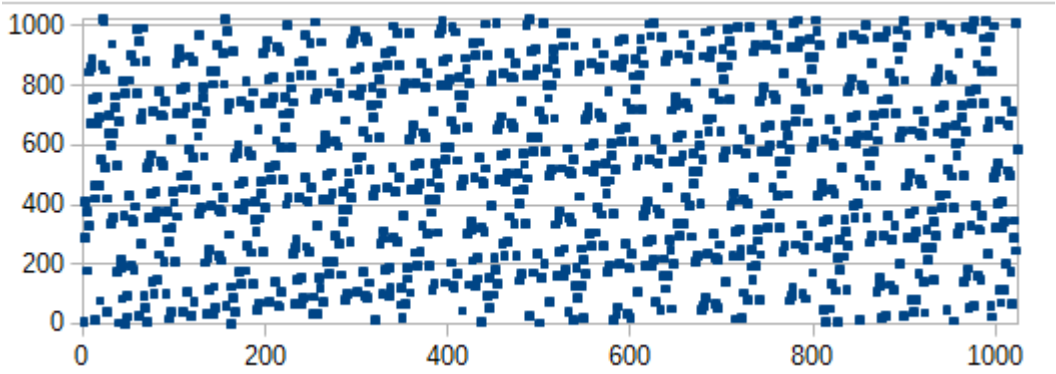
$c = 5$



$c = 51$



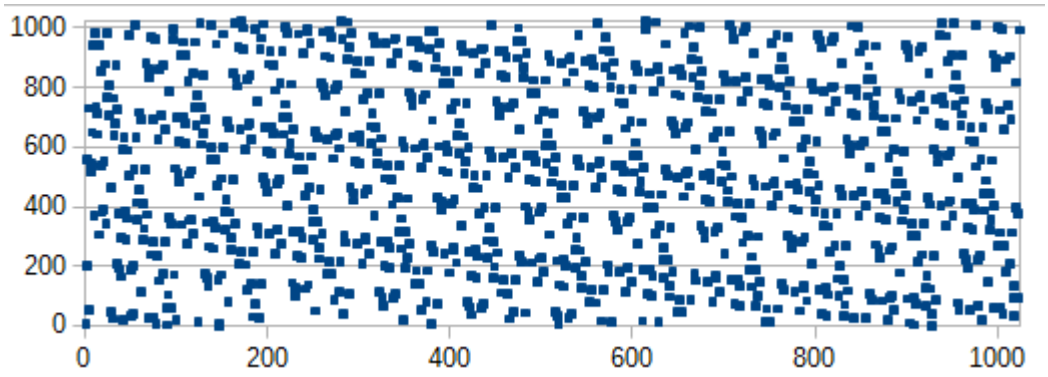
$c = 91$



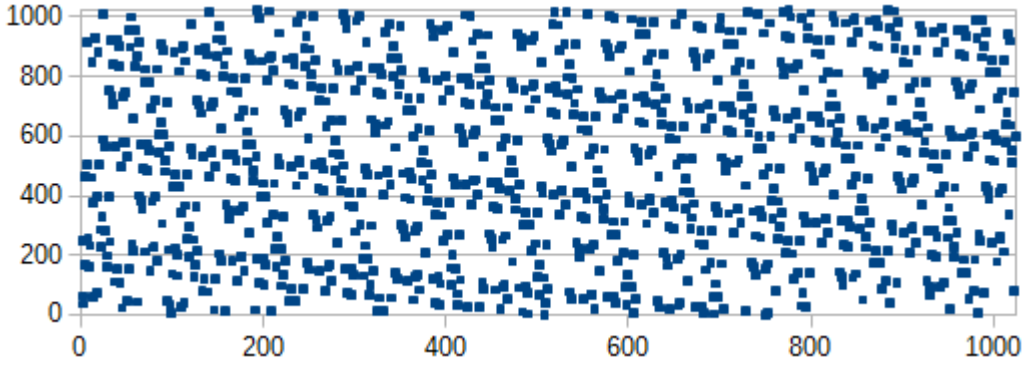
Anàlisi del paràmetre X_0

$a = 33$ $c = 5$ $m = 1024$

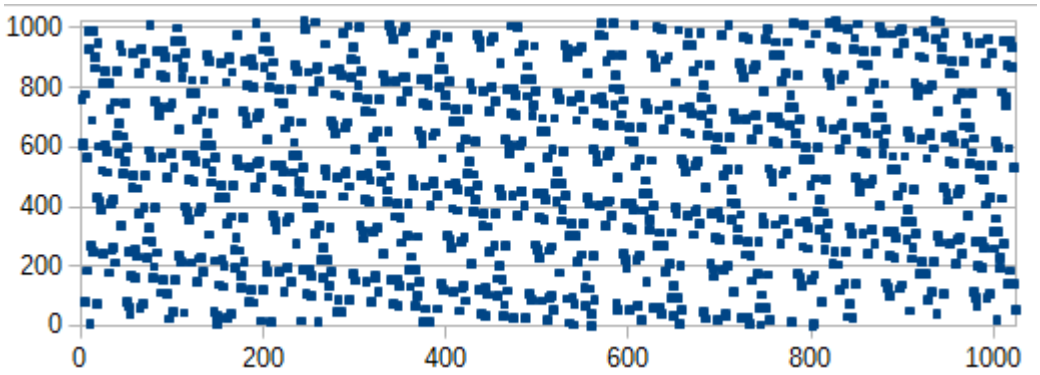
$X_0 = 6$



$X_0 = 250$



$X_0 = 763$

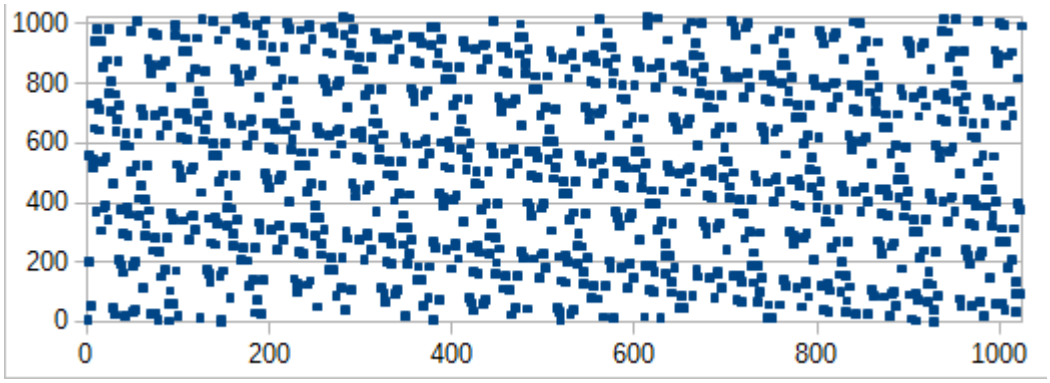


Anàlisi del paràmetre m

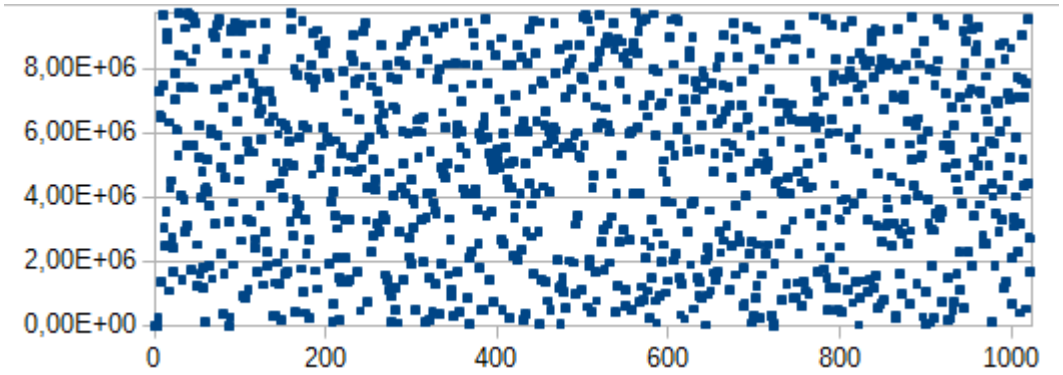
En aquest cas, no dibuixarem els períodes sencers, sinó que utilitzarem els primers 1024 (2^{10}) nombres, per a facilitar el calcul de la sèrie.

$a = 33$ $c = 5$ $X_0 = 6$

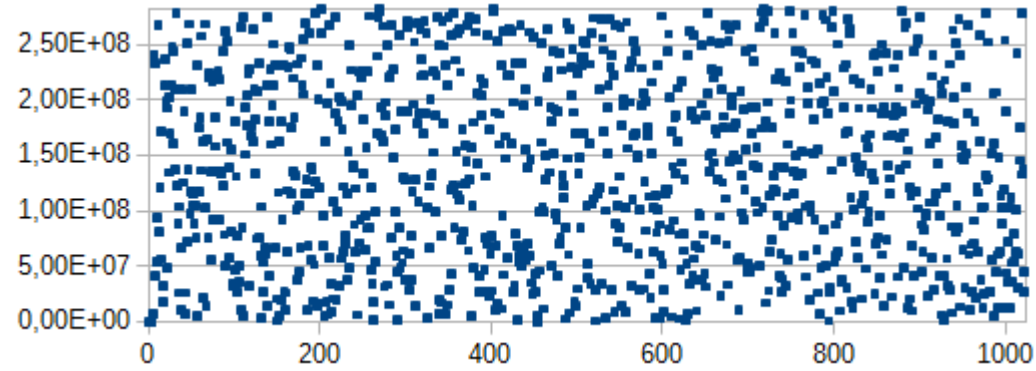
$m = 2^{10}$



$m = 5^{10}$



$m = 7^{10}$



Conclusions

Observem que tot i que variem qualsevol dels paràmetres inicials, la concentració de les dades no varia, es manté homogènia. Tot i ser uniforme, podem apreciar que està definida per rectes d'un mateix pendent definit pels paràmetres *a* i *c,* mentre que X_0 en determina les ordenades a l'origen (quan variem X_0 el pendent d'aquestes rectes no canvia). Les característiques d'aquestes rectes són les que determinen la distribució de les dades.

Però si prenem valors molt grans d'*m*, com en els gràfics $m = 5^{10}$ i $m = 7^{10}$, aquestes rectes no són apreciables ja que només estem tractant amb una part molt petita del període total de la sèrie. En general, podem concloure que com més petita sigui la fracció del període total, més "aleatòria" serà la sèrie resultant.

Tenint en compte això he modificat l'algoritme inicial perquè no retorni un període màxim, sinó un número *n* de números arbitrari, introduït per l'usuari.

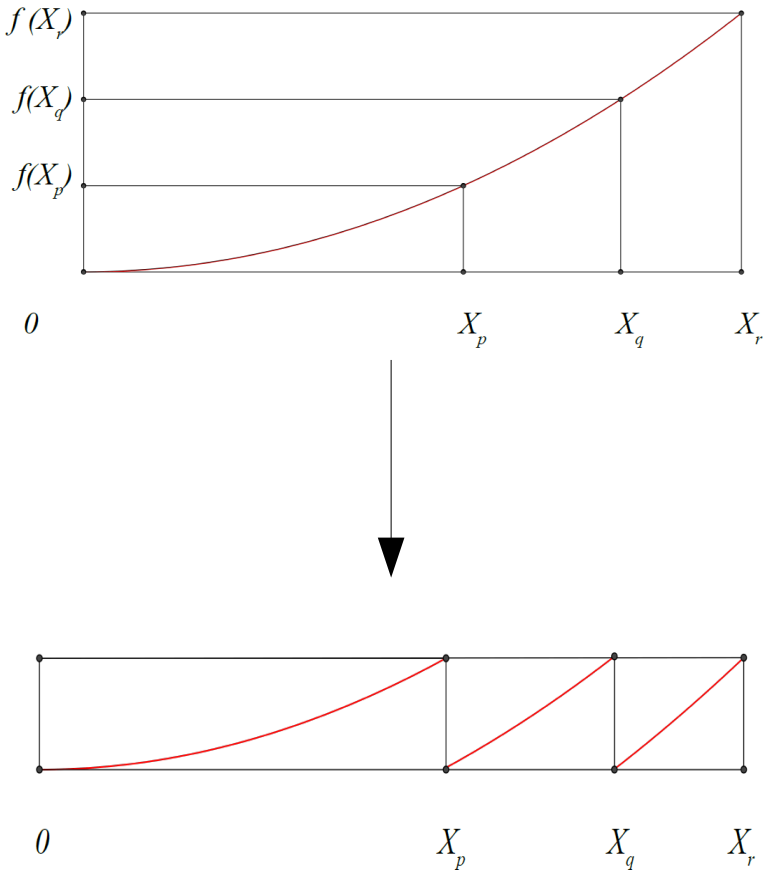
3.2.2 GENERADOR CONGRUENCIAL QUADRÀTIC

Ara estudiarem la distribució de les dades creades per un generador de la forma

$$X_{i+1}=(aX_i^2+bX_i+c)\,mod\,m$$

per veure com afecta $f(Xi)$ a la distribució de les dades, i per tant si valen la pena les operacions extra que es fan, ja que aquestes fan l'algoritme més lent.

Aquest és un cas més interessant, ja que no tots els trams de paràbola tenen la mateixa aparença: podrem veure si és el mateix tram el que es repeteix en la successió o si, en canvi, són intervals regulars d'una paràbola $y=mx^2+nx+o$ del tipus $[0,X_p],[X_{p+1},X_q],[X_{q+1},X_r]\ldots$, de manera que $d[0,f(X_p)]=d[f(X_{p+1}),f(X_q)]=d[f(X_{q+1}),f(X_r)]\ldots$ És a dir:



Tot seguit comprovarem aquestes hipòtesis procedint com en el cas anterior.

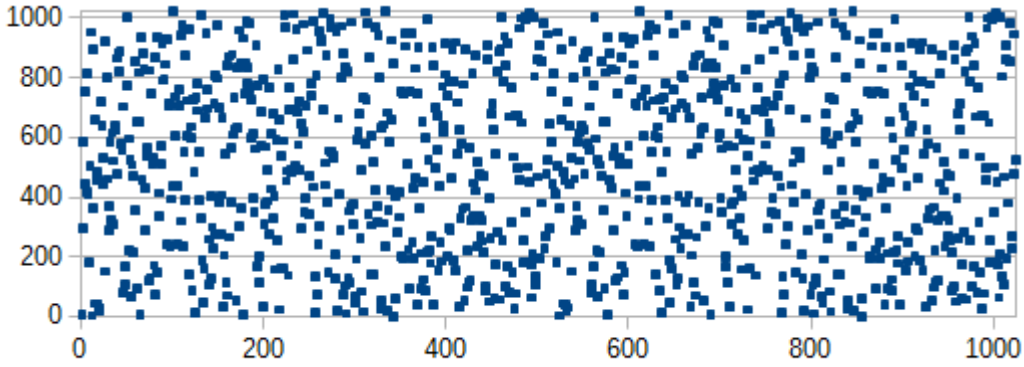
Il·lustració 12: Hipòtesi de comportament dels generadors congruencials.

Font: elaboració pròpia.

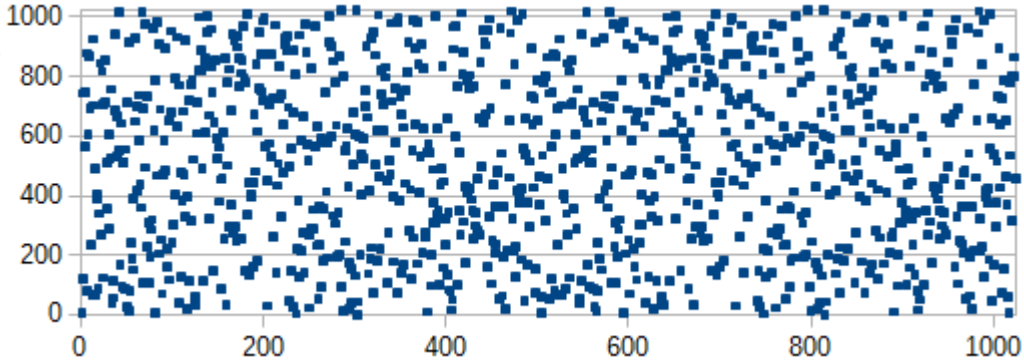
Anàlisi del paràmetre a

b= 21 c = 5 m = 1024 X_0 = 6

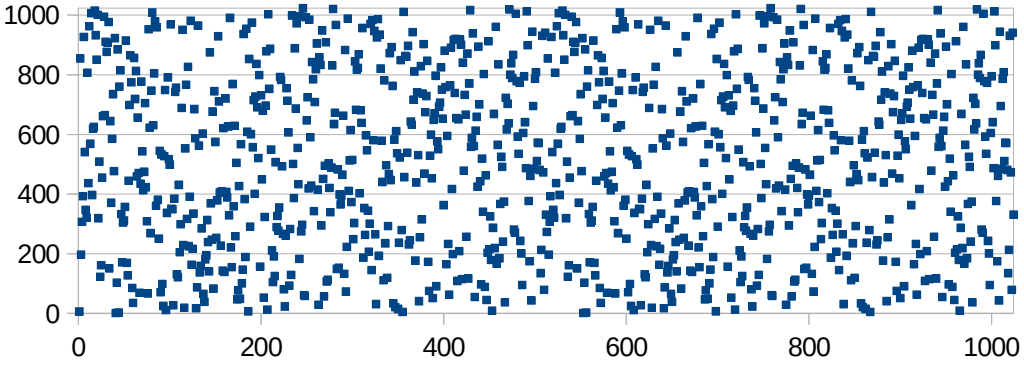
a = 17



a = 33



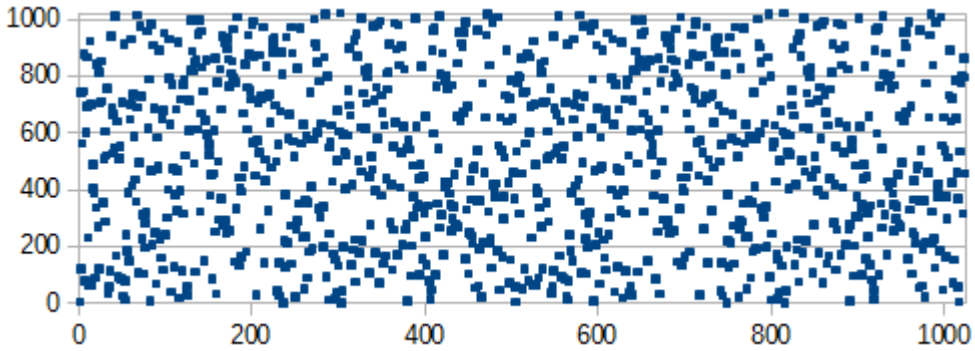
a = 77



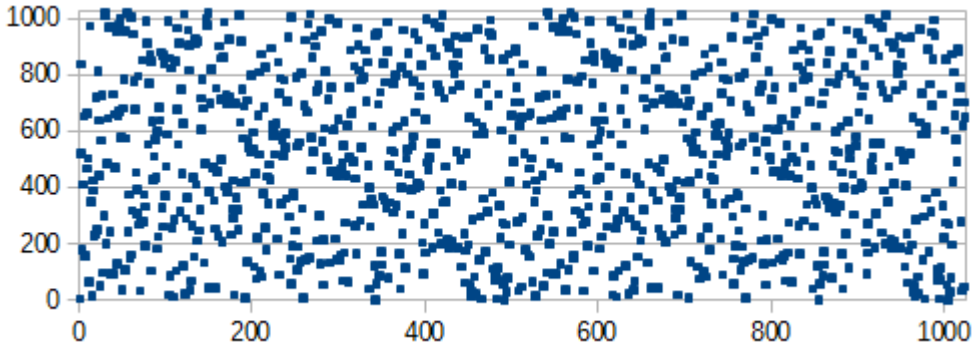
Anàlisi del paràmetre b

a = 17 c = 5 m = 1024 X_0 = 6

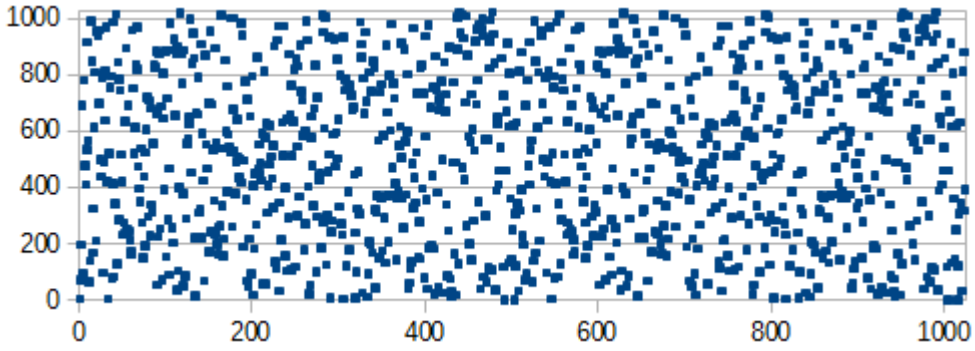
b = 21



b = 37



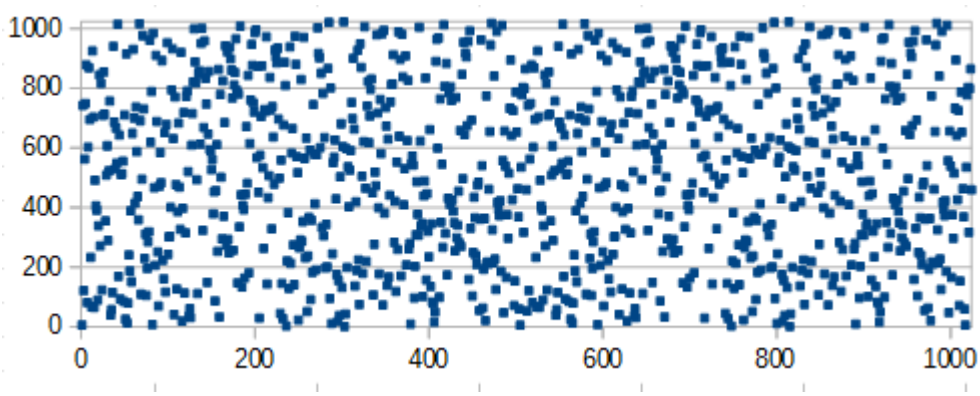
b = 101



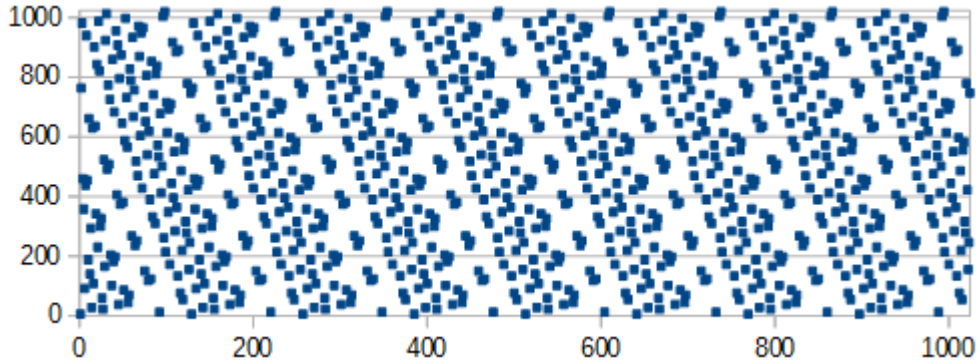
Anàlisi del paràmetre c

a = 17 b = 21 m = 1024 X_0 = 6

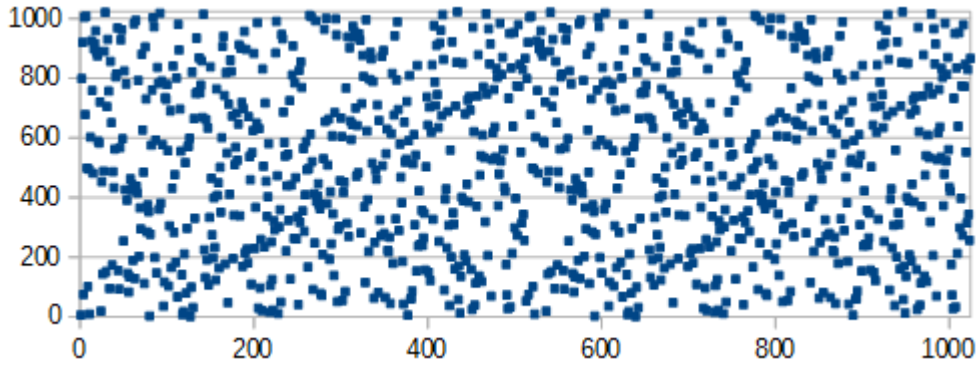
c = 5



c = 26



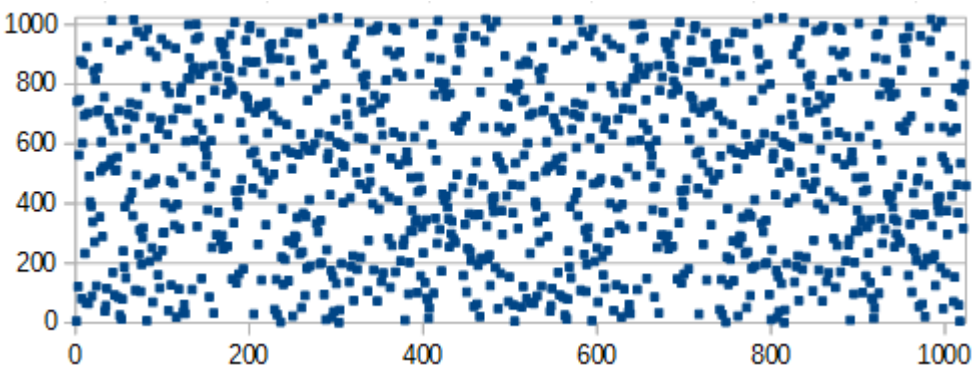
c = 61



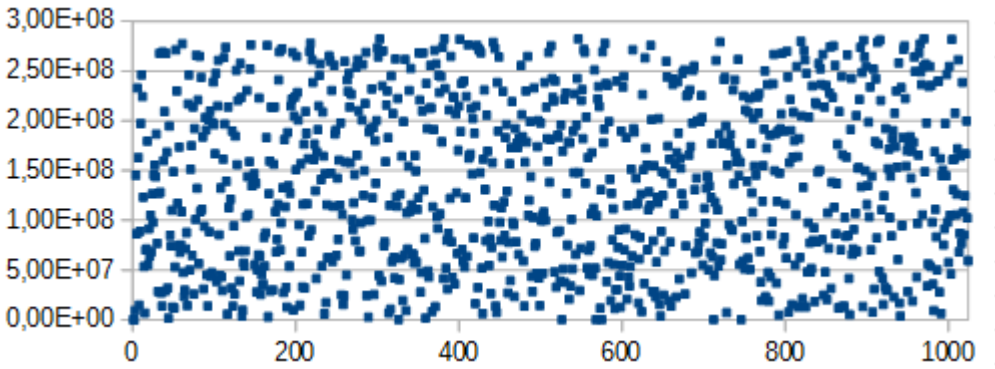
Anàlisi del paràmetre m

a = 17 b = 21 c = 5 X_0 = 6

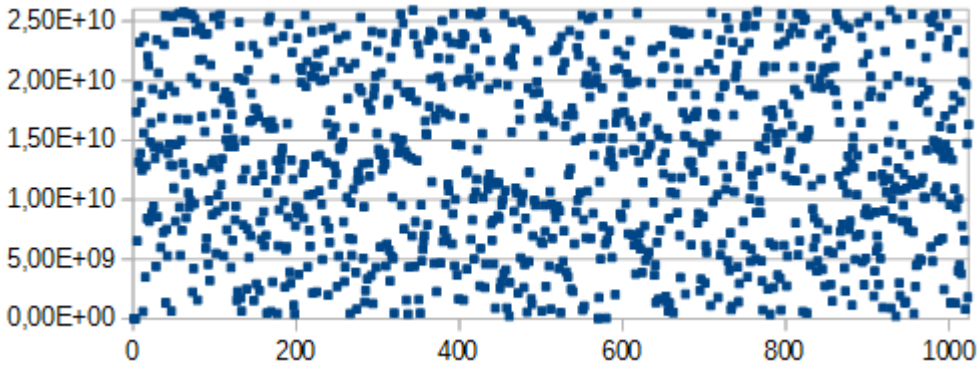
m = 2^{10}



m = 7^{10}



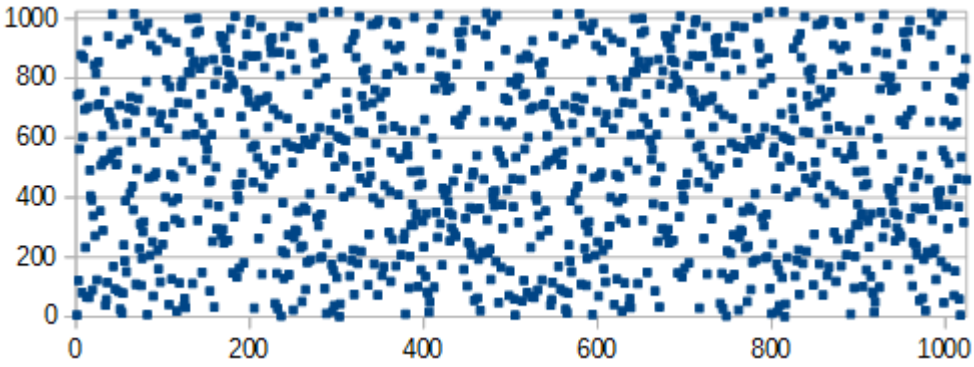
m = 11^{10}



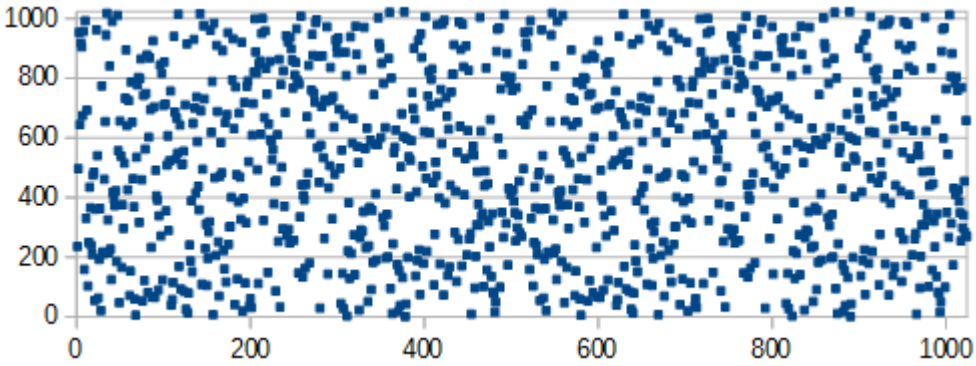
Anàlisi del paràmetre X_0

a = 17 b = 21 c = 5 m = 2^{10}

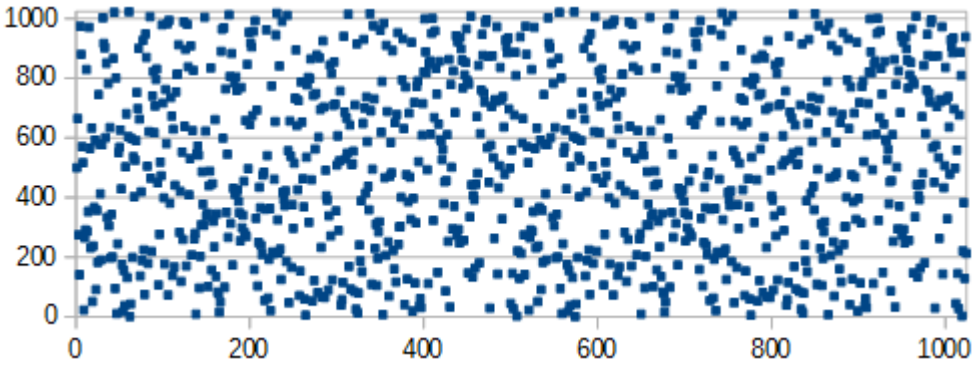
X_0 = 6



X_0 = 236



X_0 = 500



Conclusions

Aquesta vegada observem que les línies que defineixen els gràfics són menys marcades (amb la excepció del cas $c = 26$) i que n'hi ha en direccions perpendiculars (els eixos no estan en relació 1:1). La distribució és també homogènia, i en general les series de dades molt semblants.

És per això que s'utilitza el lineal, ja que obtenint un similar equivalent, és més eficient en quant a temps, ja que l'ordinador ha de realitzar menys operacions.

En realitat, el GCL és dels generadors més utilitzats per això: per la seva rapidesa de càlcul i els seus bons resultats.

4. TESTS BASATS EN QUE EL GENERADOR DE LA CALCULADORA ÉS UN GCL

4.1 TEST DE DISTÀNCIA MÍNIMA

Hem vist que un Generador Congruencial Lineal és el més utilitzat per a fins on s'utilitzen grans quantitats de nombres, però això no és necessari en el cas de la calculadora. El que sí que comparteix amb el generador de la calculadora és que dóna tots els nombres enters en un interval donat: [0,1000] en el cas de la calculadora i [0,*m*] en el cas del GCL. Llavors, és possible que la sèrie de la calculadora sigui d'un pròpia d'un GCL.

Per a comprovar-ho podem calcular la «distància» que hi ha entre les dues sèries. De la mateixa manera que la distància entre dos punts *P* i *Q* qualssevol del pla es calcula com:

$$P=(P_x,P_y)$$

$$Q=(Q_x,Q_y)$$

$$d(P,Q)=\sqrt{(P_x-Q_x)^2+(P_y-Q_y)^2}$$

Podem imaginar les sèries com a punts de 200 coordenades. Si *R* és la sèrie obtinguda del GCL i *C* la de la calculadora:

$$R=(R_1,R_2,R_3,\ldots)\qquad C=(C_1,C_2,C_3,\ldots)$$

$$d(R,C)=\sqrt{(R_1-C_1)^2+(R_2-C_2)^2+(R_3-C_3)^2+\ldots}$$

El programa fa això per a un número de vegades en funció del rang de valors de *a, c, m* i X_0. S'ha d'anar amb compte amb quins valors es trien, ja que el temps necessari per realitzar l'algoritme és proporcional al número de cicles (vegades que genera la sèrie *R* i calcula la «distància» entre aquesta i *C*), i el número de cicles ve donat per

$$n=(m_n-m_0+1)(a_n-a_0+1)(c_n-c_0+1)$$

Per exemple, per a $m_n=1100, m_0=1000, a_n=100, a_0=1, c_n=100, c_0=1$ el nombre de cicles és

$$n=(1100-1000+1)(100-1+1)(100-1+1)=1010000$$

Llavors, podem suposar que el temps necessari per dur a terme l'algorisme serà

$$t = 1010000\,k$$

On *k* és un nombre que depèn de la potència de l'ordinador que efectua l'algoritme.

En definitiva, el temps que trigarà un ordinador en efectuar l'algoritme és molt sensible als rangs de valors que li hem introduït.

El problema d'aquest mètode, però, és que no té en compte la distribució de les dades. Per a aquest mateix rang de valors d'*a, c* i *m* obtenim el següent resultat:

```
a0 = 1
an = 100
c0 = 1
cn = 100
m0 = 1000
mn = 1100
Smin = 3736.72851569
amin = 14
cmin = 8
mmin = 1024

[524, 176, 424, 824, 280, 856, 728, 984, 472, 472, 472, 472, 472, 472, 472, 472,
 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472,
 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472,
 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472,
 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472,
 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472,
 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472,
 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472,
 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472,
 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472,
 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472,
 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472, 472,
 472, 472, 472, 472, 472, 472, 472, 472]
```

Il·lustració 13: Resultats del programa.

Font: elaboració pròpia.

Que no s'assembla en res a la nostra sèrie, però és la que té la distància mínima d'entre totes les que hem provat. Tot i que no podem assegurar que no sigui un GCL, ja que amb aquesta prova només podem provar un nombre finit de combinacions de paràmetres, i la que suposadament porta la calculadora pot no estar entre aquestes. Aquest resultat suggereix buscar altres maneres d'enfocar el problema.

4.2 TEST DEL PARÀMETRE A

Una altra manera d'atacar el problema és utilitzant la propietat que, suposant que els nombres estan generats per un GCL, cada número està relacionat amb l'anterior segons

$$X_{i+1} = a\,X_i + c \bmod m$$

per tant també és cert que

$$X_i = a\,X_{i-1} + c \bmod m$$

Llavors, si restem les dues equacions obtenim que

$$X_{i+1} - X_i = a\left(X_i - X_{i-1}\right) \bmod m$$

$$\Delta X_i = a\,\Delta X_{i-1} \bmod m$$

És a dir, podem relacionar un increment en dos nombres de la sèrie amb l'increment entre els dos nombres anteriors. Per tant, si trobem un valor a que sigui vàlid per a tots els increments de la sèrie i després trobem el valor de *b*, haurem confirmat que el generador és un GCL.

Els resultats, però, no son exitosos. Utilitzant els valors

$$a_0 = 1000;\ a = 2000;\ m_0 = 1000;\ m = 2000$$

i amb un criteri de validesa de només 5 coincidències necessàries per considerar *a* i *m* bons, només hem obtingut com a solucions:

a	1307	1521	1913
m	1917	1043	1000

Tenint en compte que si la sèrie fos un GCL hauríem de tenir 200 coincidències, no podem considerar aquests resultats satisfactoris.

5. CONCLUSIONS

L'objectiu d'aquest treball era el d'investigar els usos i aplicacions dels nombres generats aleatòriament i intentar deduir l'algoritme que els genera en la calculadora CASIO fx-82MS.

A partir de mètodes d'anàlisi estadístic (el test de runs i el mètode Montecarlo) he intentat trobar algun patró o alguna pista de què podria estar fent l'algoritme, però tots els tests han portat a que la sèrie és homogènia i per tant tot indica que aquesta és aleatòria.

Però una màquina no pot generar nombres aleatòriament per ella mateixa. Per tatnt, he pegat una ullada a un dels principals generadors de nombres aleatoris: el Generador Congruencial Lineal, i l'hem programat en el llenguatge *Python*. Hem observat que, efectivament, no es tracta d'un mitjà per obtenir nombres completament aleatòriament, ja que en representar les dades es veien patrons ja a simple vista, però que tot i això pot donar sèries de dades prou bones, ja que en calcular una aproximació de π basant-me en que els nombres de la sèrie estan distribuïts homogèniament he obtingut un resultat prou bo, cosa que confirma la hipòtesi.

Finalment, he suposat que l'algoritme de la calculadora és un GCL i he intentat trobar una sèrie que s'assemblés a una obtinguda de la calculadora, de dues maneres. En primer lloc considerant la serie com un punt de 200 coordenades i una altra sèrie obtinguda d'un GCL com a un altre punt. Llavors, esperava que la sèrie que estigués a «menys distància» seria la més semblant a la nostra, però no ha estat així. També he provat de trobar un GCL semblant a la serie a partir de la propietat que defineix els nombres d'aquest generador, però novament no he tingut èxit. Tot i que si mantenim el supòsit que la calculadora utilitza un GCL aquest podria estar funcionant amb paràmetres amb valors diferents als que hem provat, tot apunta a que es tracta d'una altra font de nombres.

En conclusió, aquest ha sigut un treball molt ambiciós, ja que tracta amb temes matemàtics d'actualitat com la criptografia, i per tant ha de ser forçosament complicat. Tot i que la meta inicial ha quedat lluny d'estar assolida, he pogut estudiar per damunt el concepte de l'aleatorietat, el qual és bastant interessant, i he pogut veure la seva cabuda en el món actual en àmbits tan diversos com ho són els mètodes numèrics o la seguretat informàtica.

Ha estat una experiència molt interessant, i voldria agrair al meu tutor Claudio Cosin i també a Marià Cano la imprescindible ajuda que m'han proporcionat en la realització d'aquest treball.

6. BIBLIOGRAFIA

Algorithmically random sequence

https://en.wikipedia.org/wiki/Algorithmically_random_sequence

Data de consulta: 12 d'octubre de 2015

Statistical randomness

https://en.wikipedia.org/wiki/Statistical_randomness

Data de consulta: 12 d'octubre de 2015

History of randomness

https://en.wikipedia.org/wiki/History_of_randomness

Data de consulta: 9 de gener del 2016

Historical Notes History [of randomness] per Stephen Wolfram

https://www.wolframscience.com/reference/notes/967c

Data de consulta: 9 de gener del 2016

Estadística Compuracional per Antonio Salmerón Cerdán i María Morales Giraldo

http://www.ual.es/~asalmero/papers/libro.pdf

Data de consulta: 10 d'octubre de 2015

Monte Carlo method

https://en.wikipedia.org/wiki/Monte_Carlo_method

Data de consulta: 10 de desembre de 2015

Comprovando la aleatoriedad per Miguel Herrero

https://www.incibe.es/blogs/post/Seguridad/BlogSeguridad/Articulo_y_comentarios/comprobando_aleatoriedad

Data de consulta: 16 de gener de 2016

Métodos para probar números per Héctor Martínez RubinCelis

https://www.incibe.es/blogs/post/Seguridad/BlogSeguridad/Articulo_y_comentarios/comprobando_aleatoriedad

Data de consulta: 16 de gener de 2016

A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications per Andrew Rukhin et al.

http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf

Data de consulta: 16 de gener de 2016

The Python Tutorial

https://docs.python.org/2/tutorial/

Data de consulta: 10 d'octubre de 2015

Distribución normal

https://es.wikipedia.org/wiki/Distribuci%C3%B3n_normal

Data de consulta: 10 d'octubre de 2015

Annex

I. PROGRAMES

Aquí es troben tots els programes que he escrit i han donat escrits per a la realització del treball. Tots han estat escrits amb el llenguatge de programació *Python 3.*

Test de runs

```
print("RunTest per a una serie de nombres")
print("----------------------------------------" + '\n')

print("Serie a analitzar:")

serie =  open ("serie.txt","r")
nums = serie.readlines()
seriecalc = []

for number in nums:
    numberl = list(number)
    numberl.pop(-1)
    a = ''
    for i in numberl:
        a = a + str(i)
    a = int(a)
    seriecalc.append(a)

serie.close()

print(str(seriecalc) + '\n')
del(nums)

M = 0
float(M)

for x in seriecalc:
    M += x
mean = M/len(seriecalc)
print ("Mitjana de la serie: " + str(mean) + '\n')

sbinari = []

for x in seriecalc:
    if x < mean:
        sbinari.append(0)
    else:
        sbinari.append(1)
```

```
print("Serie en binari: " + '\n' + str(sbinari) + '\n')

n1 = 0 #numero de 0s
n2 = 0 #numero de 1s

for x in sbinari:
    if x == 1 :
        n1 += 1
    elif x == 0:
        n2 += 1
print("Numero de 0s: " + str(n1))
print("Numero d'1s: " + str(n2)+ '\n')


if seriecalc[0] < mean:
    b = 0
else:
    b = 1

runs = 1

for a in sbinari:
    if a != b:
        runs += 1
        b = a

print("Numero de runs: " + str(runs) + '\n')

mu = (2 * n1 * n2)/(n1 + n2) + 1
print("mu = " + str(mu))

sigma_2 = (2 * n1 * n2 * (2* n1 * n2 - n1 - n2))/((n1 + n2)**2
* (n1 + n2 -1))
sigma_2 = sigma_2**0.5
print("sigma^2 = " + str(sigma_2))
N = (runs-mu)/sigma_2
print("N = " + str(N) + '\n')

input()
```

Aproximació de π amb el mètode Montecarlo utilitzant un GCL

```
print("APROXIMACIO DE PI UNTILITZANT EL METODE MONTECARLO I UN
GLC")
print("-------------------------------------------------------
----")


a = int(input("a = "))
c = int(input("c = "))
X0 = int(input("X0 = "))
g = int(input("m = 2^g, input g = "))
m = 2**g
print("m = " + str(m) + '\n')

P = []

P. append(X0)
X = X0

while True:
    Xi = (a * X + c) % m
    if Xi == X0:
        break
    else:
        P.append(Xi)
        X=Xi


PI = []

float (m)
r = m/2
goals = 0

for i in range(0,m):
    p = i-1
    x = P[p]
    float (i)
    float (x)
    d = ((r-i)**2+(r-x)**2)**0.5
```

```
    if d <= r:
        goals+=1
        PI.append(x)


print("Nombre de punts a l'interior del cercle: " +
str(len(PI)))
print("Nombre total de punts: " + str(m)+ '\n')

pi = 4*(float(goals)/float(len(P)))
print("Aproximacio de pi amb "+str(m)+" punts:")
print(pi)


input()
```

Mètode Montecarlo com a test d'homogeneïtat de la sèrie de la calculadora

```
print("METODE MONTECARLO COM A TEST ESTADISTIC")
print("----------------------------------------")
serie =  open ("serie.txt","r")
nums = serie.readlines()

seriecalc = []

for number in nums:
    numberl = list(number)
    numberl.pop(-1)
    a = ''
    for i in numberl:
        a = a + str(i)
    a = int(a)
    seriecalc.append(a)

serie.close()

print("Serie de la calculadora: " + '\n' + str(seriecalc) +
'\n')

serieq = []

for i in seriecalc:
    float(i)
    x = i/5
    serieq.append(x)
print("Serie de la calculadora dividida entre 5: " + '\n' +
str(serieq) + '\n')

PI = []

r = 100
goals = 0

for i in range(0,200):
    p = i-1
```

```
    x = serieq[p]
    float (i)
    float (x)
    d = ((r-i)**2+(r-x)**2)**0.5

    if d <= r:
        goals+=1
        PI.append(x)


print("Nombre de punts a l'interior del cercle: " +
str(len(PI)))
print("Nombre total de punts: " + str(200)+ '\n')

pi = 4*(float(goals)/float(len(serieq)))

print("Aproximacio de pi amb "+str(200)+" punts:")
print(pi)


input()
```

Generador congruencial lineal (1 període)

```
print("GENERADOR CONGRUENCIAL LINEAL")
print("----------------------------")
print("m=2^g" + '\n' + "Introduiu: " + '\n')

a=int(input("a ="))
c=int(input("c ="))
g=int(input("g ="))
X0=int(input("X_0= "))

cla = open("serie gcl.txt","w")

cla.write("Serie obtinguda amb un generador congruencial
lineal amb els parametres:"+'\n')
cla.write("a = " + str(a)+'\n')
cla.write("c = " + str(c)+'\n')
cla.write("g = " + str(g)+'\n')
cla.write("X0 = " + str(X0)+'\n')
cla.write('\n')

m=2**g

cla.write(str(X0)+"\n")
X=X0

while True:
     Xi=(a*X+c)%m
     if Xi==X0:
           cla.close()
           break
     else:
           cla.write(str(Xi)+"\n")
           X=Xi
```

Generador congruencial lineal (n nombres)

```
print("LINEAL CONGRUENTIAL GENERATOR"+"\n")
print("Introduiu:"+"\n")

a=int(input("a ="))
c=int(input("c ="))
m=int(input("m ="))
X0=int(input("X_0 ="))
print("")
n=int(input("Total de nombres a la serie: "))

cla = open("serie gcl.txt","w")

cla.write("Serie obtinguda amb un generador congruencial
lineal amb els parametres:"+'\n')
cla.write("a =" + str(a)+'\n')
cla.write("c = " + str(c)+'\n')
cla.write("m = " + str(m)+'\n')
cla.write("X0 = " + str(X0)+'\n')
cla.write("\n" + "n = " + str(n) + "\n")


cla.write(str(X0)+"\n")
X=X0
k=1
while True:
        Xi=(a*X+c)%m
        k = k+1
        if k>n:
                cla.close()
                break
        else:
                cla.write(str(Xi)+"\n")
                X=Xi
```

Generador congruencial quadràtic (n nombres)

```
print("GENERADOR CONGRUENCIAL LINEAL")
print("----------------------------")
print("m=2^g" + '\n' + "Introduiu: " + '\n')

a=int(input("a ="))
c=int(input("c ="))
g=int(input("g ="))
X0=int(input("X_0= "))

cla = open("serie gcl.txt","w")

cla.write("Serie obtinguda amb un generador congruencial
lineal amb els parametres:"+'\n')
cla.write("a = " + str(a)+'\n')
cla.write("c = " + str(c)+'\n')
cla.write("g = " + str(g)+'\n')
cla.write("X0 = " + str(X0)+'\n')
cla.write('\n')

m=2**g


cla.write(str(X0)+"\n")
X=X0

while True:
     Xi=(a*X+c)%m
     if Xi==X0:
           cla.close()
           break
     else:
           cla.write(str(Xi)+"\n")
           X=Xi
```

Test de distància mínima

```
print("TEST DE DISTÀNCIA MÍNIMA")
print("------------------------")

serie =  open ("serie.txt","r")
nums = serie.readlines()

seriecalc = []

for number in nums:
    numberl = list(number)
    numberl.pop(-1)
    a = ''
    for i in numberl:
        a = a + str(i)
    a = int(a)
    seriecalc.append(a)


print(str(seriecalc) + '\n')

Smin = 10000
amin = 0
cmin = 0
mmin = 0
X0 = 524

print("Introduiu els seguents parametres:"+'\n')
a0 = int(input("a0 = "))
an = int(input("an = "))
c0 = int(input("c0 = "))
cn = int(input("cn = "))
m0 = int(input("m0 = "))
mn = int(input("mn = ") + '\n')
random_min = []

for m in range(m0,mn):
    for a in range(a0,an):
        for c in range(c0,cn):
```

```
            random = []
            random.append(X0)
            X=X0
            k=1
            while True:
                Xi=(a*X+c)%m
                k = k+1
                if k>200:
                    break
                else:
                    X=Xi
                    random.append(X)
            S = 0
            for i in range(1,200):
                x = seriecalc.pop(i)
                seriecalc.insert(i,x)
                y = random.pop(i)
                random.insert(i,y)
                float(x)
                float(y)
                S = S + (x-y)**2
            S = S**0.5
            if S < Smin:
                Smin = S
                amin = a
                cmin = c
                mmin = m
                random_min = random

print("Smin = " + str(Smin))
print("amin = " + str(amin))
print("cmin = " + str(cmin))
print("mmin = " + str(mmin) + '\n')
print(random_min)

input()
```

Test del paràmetre a

```
print("TEST DEL PARAMETRE A")
print("--------------------")

serie =  open ("serie.txt","r")
nums = serie.readlines()

seriecalc = []

for number in nums:
    numberl = list(number)
    numberl.pop(-1)
    a = ''
    for i in numberl:
        a = a + str(i)
    a = int(a)
    seriecalc.append(a)

serie.close()

print("Serie de la calculadora: " + '\n' + str(seriecalc) +
'\n')

diferencies = [453]

for i in range(2,200):
    a = seriecalc[i]
    n = i - 1
    b = seriecalc[n]
    D = abs(a-b)
    diferencies.append(D)

print("Increments :" + str(diferencies) + '\n')

a0 = int(input("a0 = "))
a = int(input("a = "))
m0 = int(input("m0 = "))
m = int(input("m = "))
```

```
p = int(input("Introduiu numero de coincidencies per
considerar que el valor d'a es bo:"))

solucions_a = []
solucions_m = []

for i in range(a0,a):
    print(i)
    for j in range(m0,m):
        contador = 0
        for k in range(0, (len(diferencies)-1)):
            if diferencies[k]%j == (diferencies[k+1]*i)%j:
                contador += 1
        if contador >= p:
            print("Solucio:")
            print("a = " + str(i))
            print("m = " +str(j))
            solucions_a.append(i)
            solucions_m.append(j)

print("Nombre de possibles solucions: " +
str(len(solucions_a)))
print("Valors valids d'a:" + str(solucions_a))
print("Valors valids d'm:" + str(solucions_m))

input()
```

II. CD

He inclòs en un CD fitxer el software que he utilitzat o creat durant la realització del treball :

- Instal·lador de *IDLE for Python 3.*
- La serie de la calculadora.
- El test de runs aplicat a la sèrie de la calculadora.
- Càlcul aproximat de π amb el mètode Montecarlo a partir de nombres generats amb un Generador Congruencial Lineal.
- Càlcul de π amb la sèrie de la calculadora utilitzant el mètode Montecarlo per tal d'estimar la homogeneïtat de la seqüència.
- Generador Congruencial Lineal que genera un període de la sèrie un cop introduïts els paràmatres.
- Generador Congruencial Lineal que genera *n* nombres un cop introduïts els paràmetres inicials.
- Test de distància mínima.
- Test del paràmetre *a.*