Laboratori de Telemàtica 3

Memòria de la

Pràctica 2

David Guillen Fandos
Toni Jaume Mir

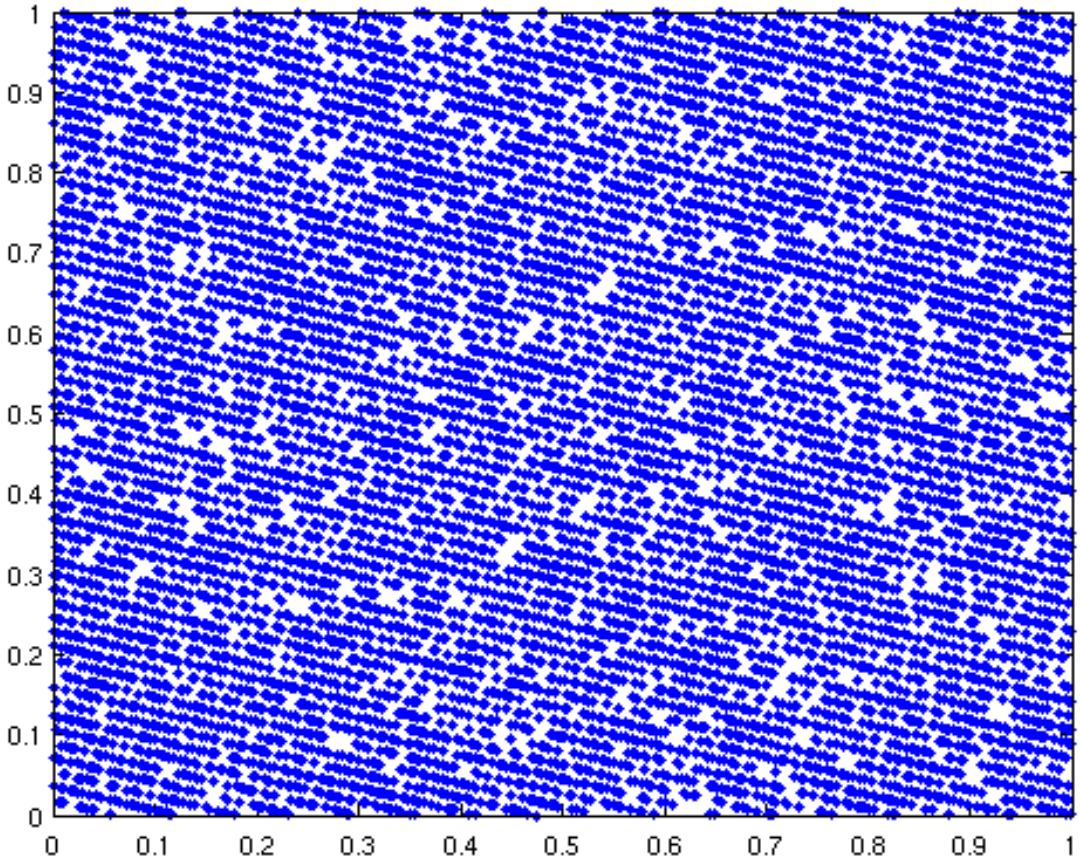
· Estudi previ

```
function estadistica( v )
      fprintf('Mitjana: %d\n',mean(v))
      fprintf('Mediana: %d\n',median(v))
      fprintf('Variança: %d\n',var(v));
      fprintf('Període: %d\n',seqperiod(v));
      figure('Name','Correlació');
      plot(xcorr(v))
      figure('Name','Histograma');
      hist(v,100)
      figure('Name','Dispersió 2D');
      plot(v(1:length(v)-1),v(2:length(v)),'.')
      figure('Name','Dispersió 3D');
      plot3(v(1:length(v)-2),v(2:length(v)-1),v(3:length(v)),'.')
end
```

· Exercici 1

Mitjana: 0,4997151
Mediana: 0,4994234
Variància: 0,0844338
Període: 9972

Diagrama de dispersió 2D:

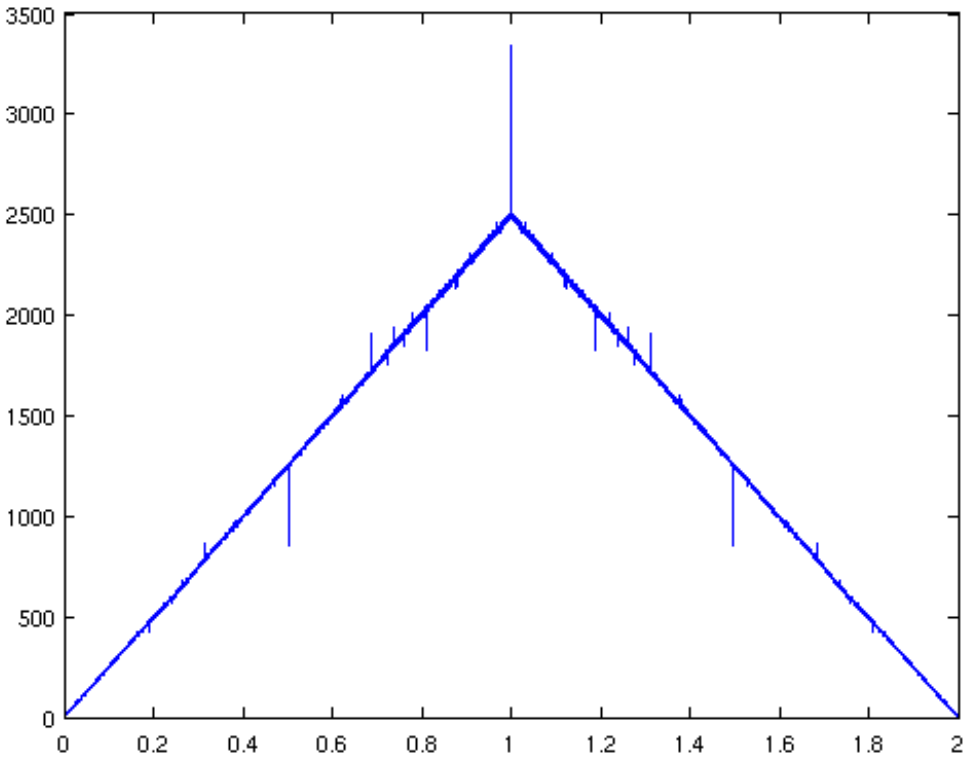


Podem veure que el període és 9972, cosa que provoca una repetició de nombres molt gran. Per aquest motiu el digrama de dispersió de 10000 mostres és el mateix que el diagrama obtingut amb un nombre superior de mostres.

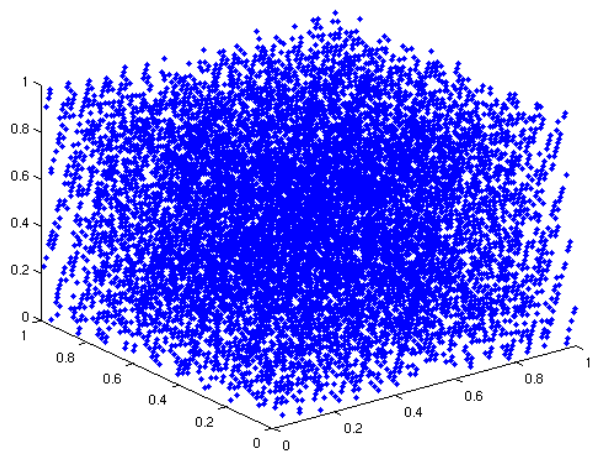
El fet que el diagrama presenti un patró de línies ens indica amb tota claredat que alguna cosa falla. Idealment les mostres haurien d'estar disperses d'una forma uniforme i no pas formant figures.

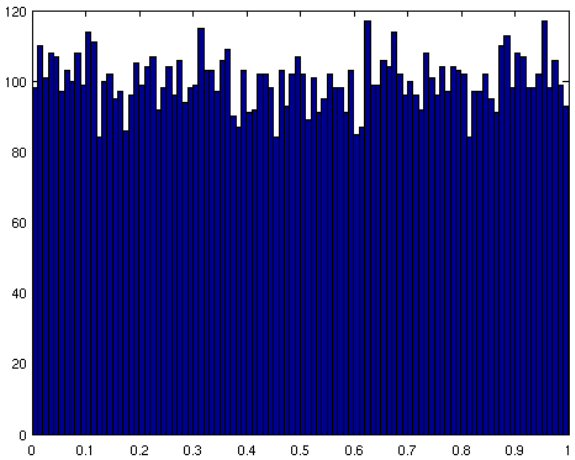
Comprovem que la correlació és l'esperada. Conté una delta (un gran pic) a l'origen i als extrems decreix. La forma triangular es deu a

que la seqüència aleatòria no és de longitud infinita, pel que queda enfinestrada per una senyal rectangular. La convolució de les dues senyals rectangulars dóna lloc a una finestra triangular.



També hem representat l'histograma i la dispersió 3D.





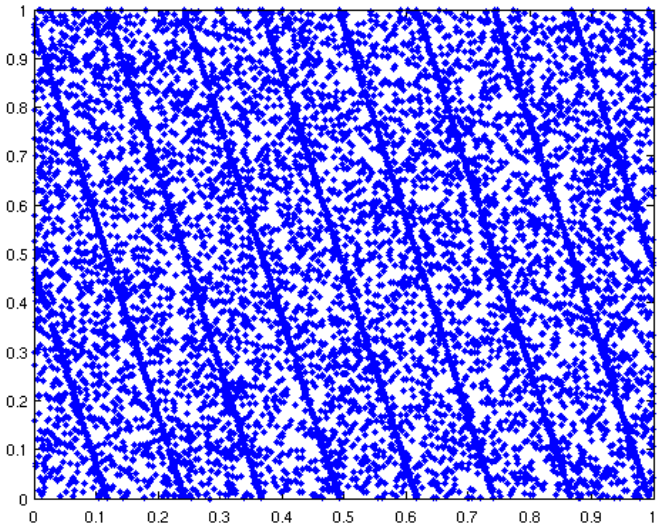
El diagrama de dispersió en 3D no ens aporta massa informació pel fet que és difícil de veure en una projecció 2D, en canvi l'histograma sí que ens és útil. Idealment l'histograma hauria de ser pla completament, ja que aquest és una aproximació discreta de la funció

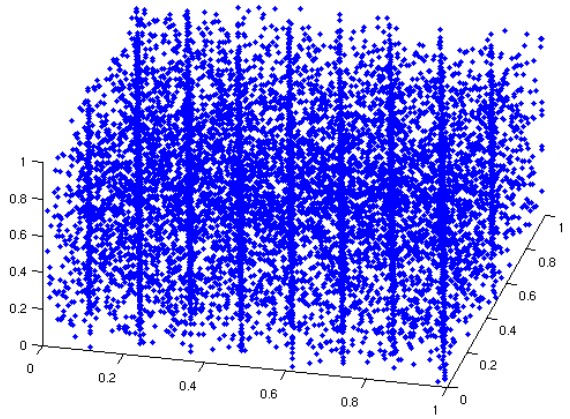
de densitat espectral. En el cas de mostres independents la correlació és una delta i la densitat espectral és un nombre real (funció constant).

- **Exercici 2**

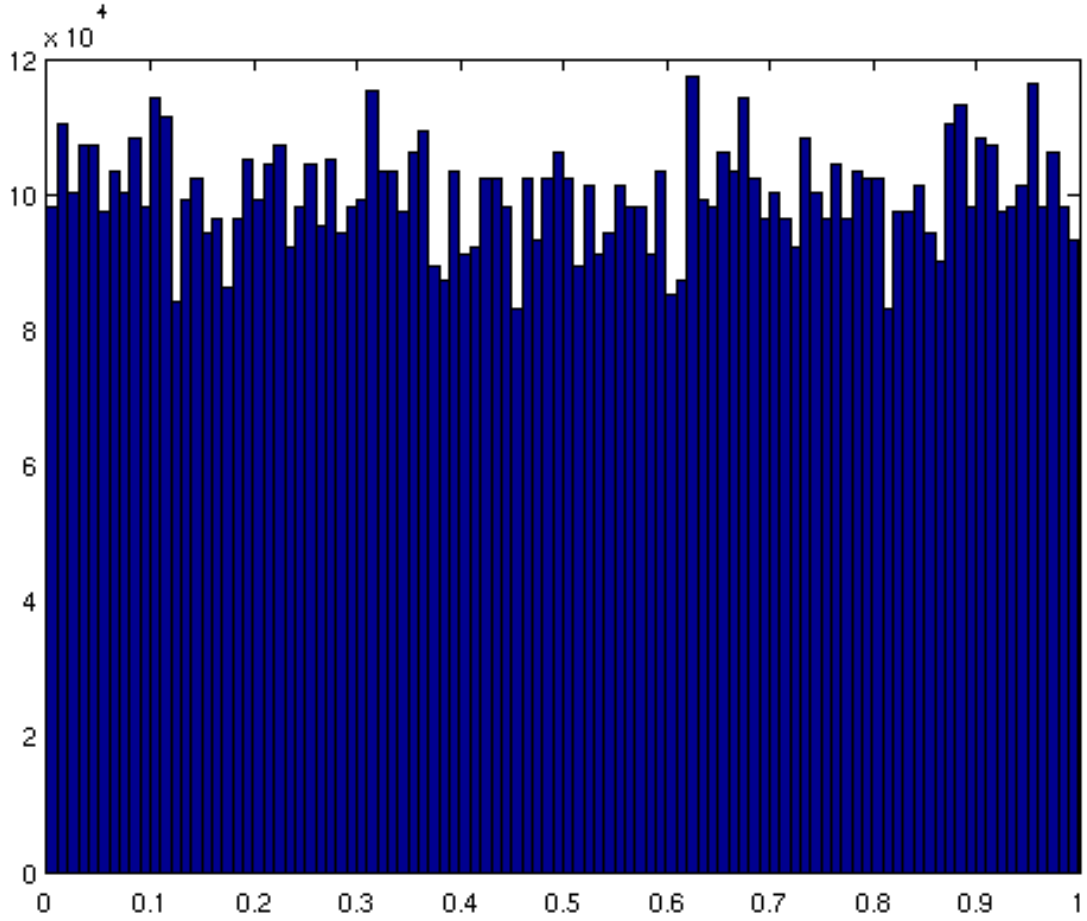
Mitjana:	0,4995498
Mediana:	0,4992480
Variància:	0,08438781
Període:	10000000

Noteu que hem realitzat aquest experiment amb moltes més mostres que les requerides per l'enunciat. L'objectiu era veure si el període era realment més gran. Donat que no hem estat capaços de trobar una repetició en les mostres fent ús de nombres molt grans podem concloure que té un període molt gran (quasi infinit a efectes pràctics).





Tornem a veure una clara dependència entre les mostres. Es veuen unes rectes molt marcades entre l'espai de punts més o menys uniformement distribuïts. Tot i que és capaç de generar molts nombres sense repeticions (període gran) hi ha un conjunt de nombres molt més probables que d'altres (els de les rectes). L'histograma és revelador en aquest cas:



Es veu que no és pla i que abunden més unes mostres que altres (amb una diferència prou significativa). La correlació és molt semblant a l'anterior cas.

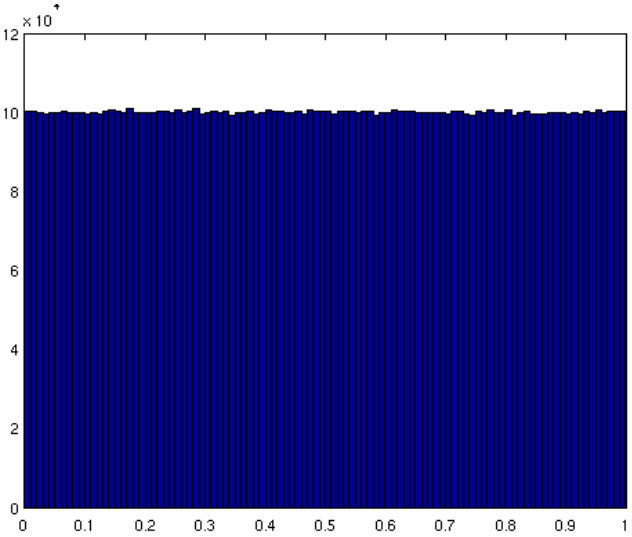
- **Exercici 3**

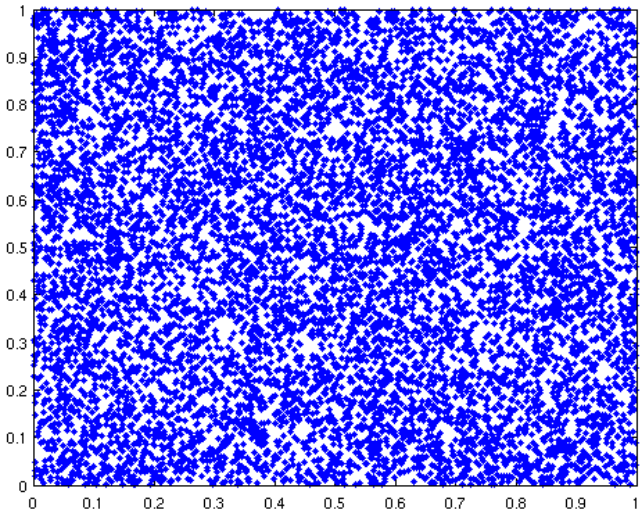
Mitjana: 0,4998944
Mediana: 0,4997585
Variància: 0,0833257
Període: 10000000

Si hem de triar entre dos generadors LCG està clar que triarem el segon. El motiu principal és que no té un període curt, pel que és poc “previsible”, cosa important si volem fer simulacions llargues. Seria desastrós fer ús de nombres aleatoris que es repetissin amb freqüència ja que això ens portaria a resultats erronis en les simulacions.

Caldria tenir en compte també el cost computacional de cada generador. Potser per a certes aplicacions és important que sigui ràpid més que no pas de qualitat. A més es pot considerar una combinació de dos algorismes (tot i que caldria comprovar que realment funciona) per tal de minimitzar el cost computacional i mantenir la qualitat.

Ara veiem els resultats de l'estudi previ aplicats al Twister-Mersenne. Com es pot apreciar l'histograma és completament pla i el diagrama de dispersió 2D mostra les mostres molt disperses i sense cap “forma” o distribució en especial. També té un període molt llarg (més de 10M de mostres).



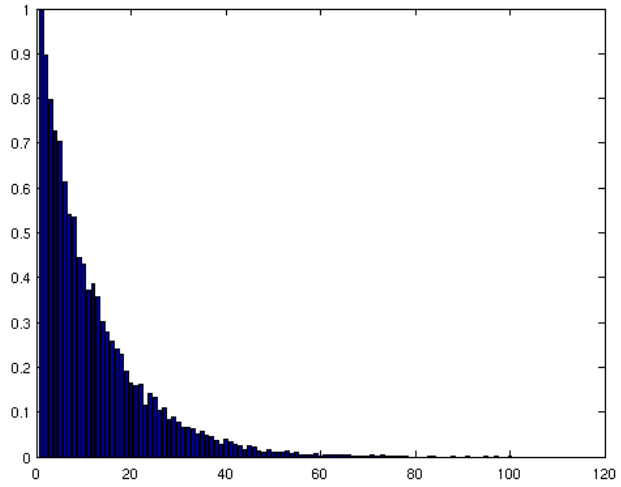


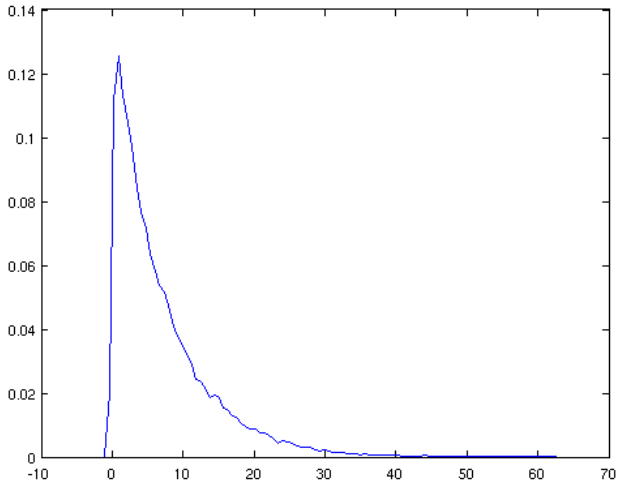
· Exercici 4

El codi per a generar l'exponencial és el següent:

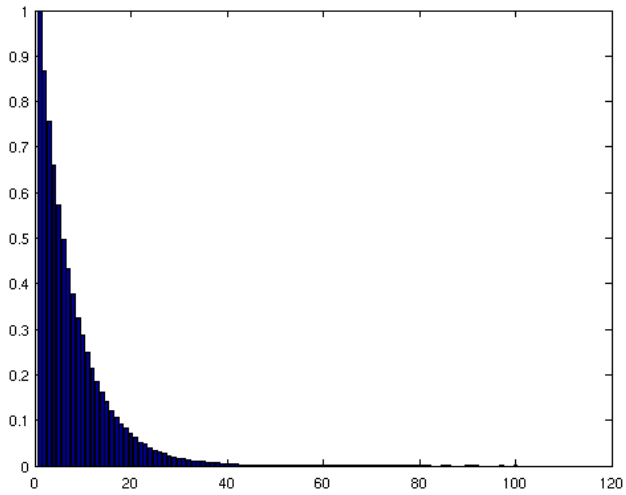
```
double lt3_exponencial(double media) {
	return -media*log(1-uniform(0,1));
}
```

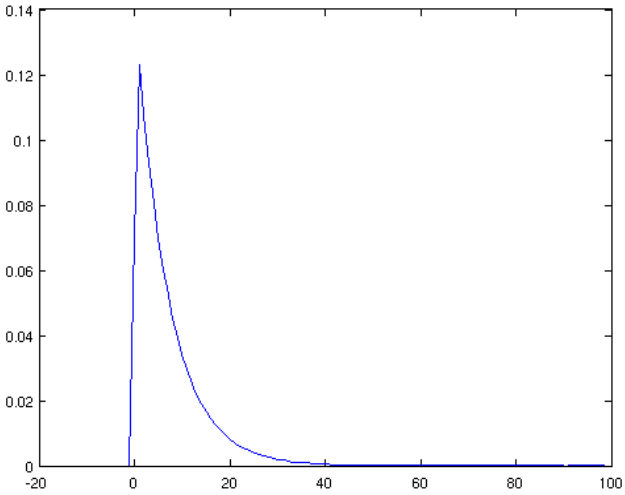
La primera prova que fem és fent ús del primer generador, el LCG.





Com podem veure apareix una exponencial ben definida tot i que amb un cert “soroll” a diferents freqüències. En canvi com veiem a continuació el generador de Twister-Mersenne és bastant més net.



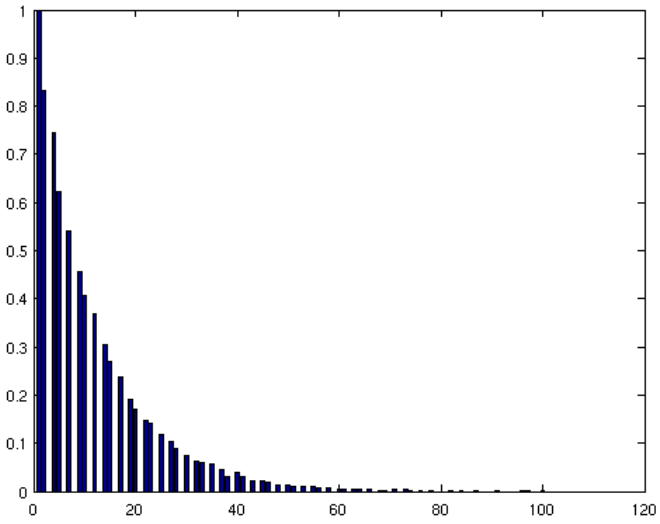


En aquest cas l'exponencial és més marcada (decreix molt ràpidament) i apareix amb una línia molt més fina i sense irregularitats (hem pres 1M de mostres).

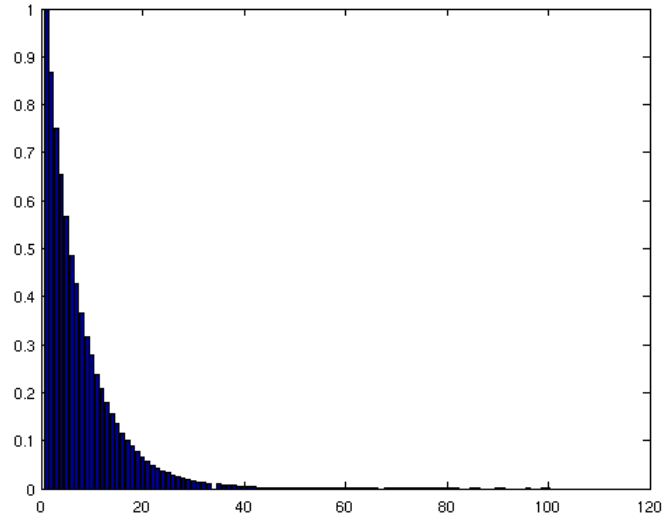
· Exercici 5

Hem realitzat 3 simulacions amb 1M de mostres. La primera correspon al LCG, la segona a l'algorisme Twister-Mersenne i la tercera correspon al generador intern de matlab (que en realitat és un Twister-Mersenne segons indica l'ajuda de matlab).

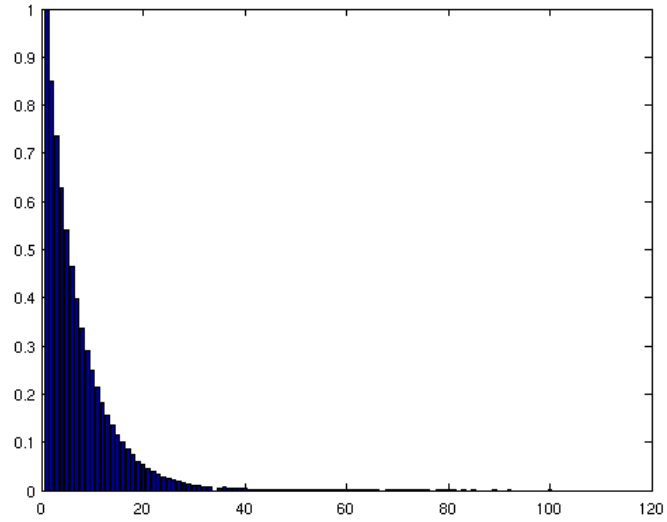
Hem obtingut l'histograma de les 3 simulacions:



Com es pot veure en el cas del LCG apareixen “forats” a l'histograma deguts al reduït període del generador (els nombres es repeteixen i per això hi ha barres que valen 0.) Hem pres 100 barres per l'histograma.



En canvi en el cas del Twister el resultat és una exponencial molt ben definida sense “forats”.



El generador intern de matlab és en principi el mateix i és per això que les gràfiques són molt similars.

Per a generar la variable aleatòria geomètrica simplement prenem la part entera d'una variable aleatòria exponencial de mitja igual a la raó de la geomètrica. Tot i això també es pot fer de forma equivalent una distribució geomètrica amb el següent codi:

```
log(1-uniform(0,1)*(1.0f-1.0f/media))/log(1.0f-1.0f/media);
```

Aquesta expressió és fruït del mètode de la inversió. Ara bé, si volem que ens retorni un valor entre un valor mínim i un valor màxim cal modificar el codi una mica.

En primer lloc es pot fer d'una manera molt simple que consisteix en generar valors i descartar aquells que no es trobin en l'interval, però això és ineficient especialment si el valor mínim és alt (comparat amb la mitjana).

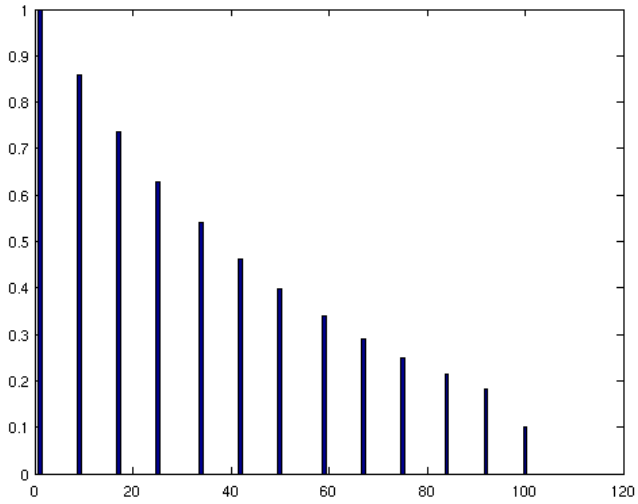
La solució que aporta l'enunciat consisteix en generar valors entre 0 i vmax-vmin i desplaçar-los vmin, cosa correcta ja que l'exponencial és una funció sense memòria. El problema és que cal “corregir” la mitja per tal que això funcioni correctament. La necessitat d'aquesta correció es pot veure en un cas simple: suposem vmin=100, vmax=200 i una mitjana de 3. Està clar que si generem una exponencial de mitjana 3 limitada a un interval de 100 no és el mateix, ja que la mitjana ha de variar per a canviar el decaïment de la corba.

Així el codi queda de la següent manera:

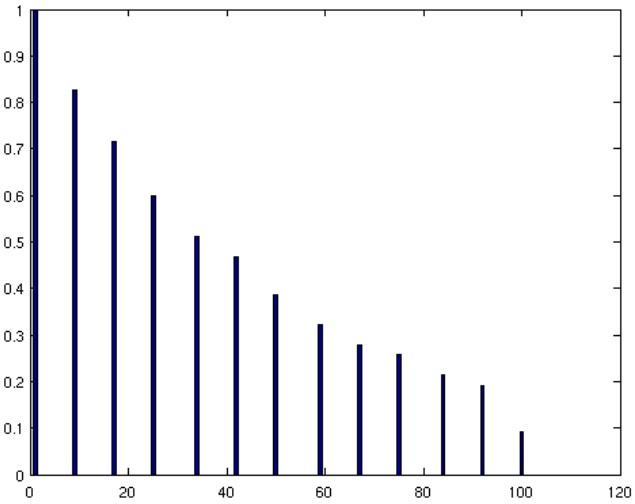
```
double lt3_geometrica(double vmin, double vmax, double media) {
  int valor =
       log(1-uniform(0,1)*pow(1.0f-1.0f/media,vmin))
            /log(1.0f-1.0f/media);
  while (valor > (vmax-vmin)) {
       valor =
            log(1-uniform(0,1)*pow(1.0f-1.0f/media,vmin))
                 /log(1.0f-1.0f/media);
  }
  return valor + vmin;
```

Generem el valor al que li sumem el mínim i corregim la mitja elevant-la al mínim. Només acceptem el valor si es troba en l'interval.

Afegim les gràfiques per al cas que s'esmenta a l'enunciat. No hi ha massa diferència entre els dos generadors aleatoris (ja que prenem un interval petit).



Twister-Mersenne



LCG

· Exercici 6

Tot seguit es presenten els algorismes emprats:

```
double lt3_normal(double media, double varianza, double metodo) {
   const double e = 2.718281828f;
   const int n = 12;

   switch ((int)metodo) {
   case 1: {
         double sign = 1.0f;
         if (uniform(-1,1) < 0) sign = -1.0f;
         while (true) {
               double xi = lt3_exponencial(1);
               double ui = uniform(0,1);
               double c = 1.315487709f;  // sqrt(2*e/PI)
               double tx = pow(e,-xi)*c;
               double fx =
(1.0f/sqrt(2*M_PI*varianza))*pow(e,-0.5f*((xi-media)*(xi-media)/varianza));

               if (fx/tx >= ui)
                     return xi*sign;
         }
   }break;
   case 2: {
         double phi = uniform(0,2*M_PI);
         double r = sqrtf(-2*log(uniform(0,1)));

         double x = r*cos(phi) + media;
         return x;
   }break;
   case 3: {
         double suma = 0;
         for (int i = 0; i < n; i++) {
```

```
                suma += uniform(0,1);
            }
            suma -= n/2;
            return suma*sqrtf(varianza)+media;
    }break;
    }
}
```

El codi és bastant auto-explicatiu. En el primer cas fem ús del que es diu a l'enunciat sobre el mètode de hit/miss. Durant un nombre indeterminat d'iteracions busquem nombres aleatoris que accepetem o descartem en funció del criteri esmentat. En el segon codi simplement fem ús de la fórmula que ens ve donada. En l'últim algorisme només hem de sumar diverses V.A. uniformes.

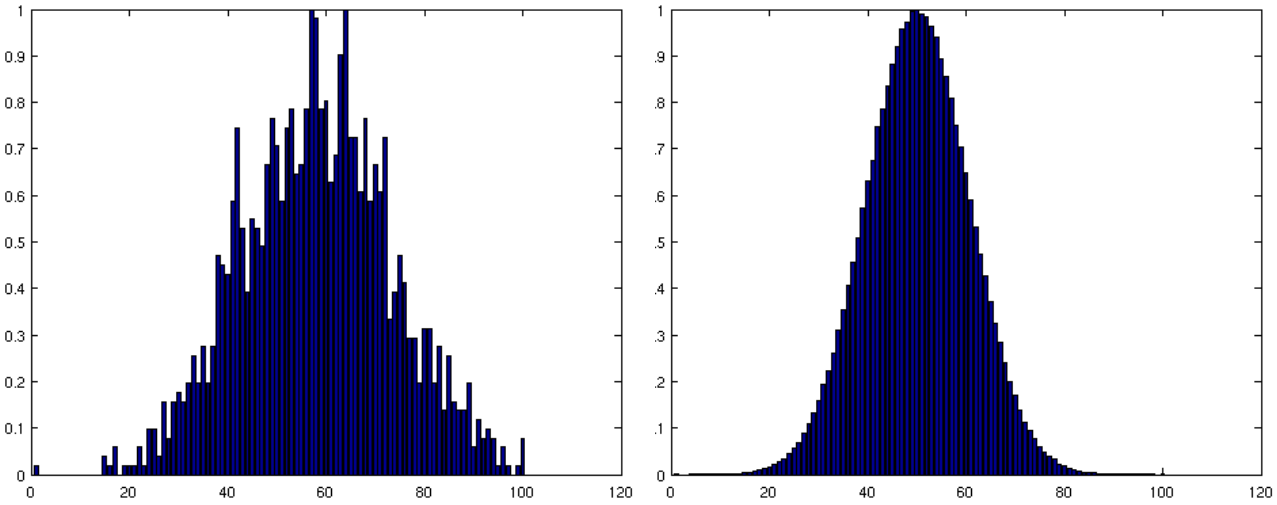
El resultat de les simulacions per a cada mètode es troba seguidament:

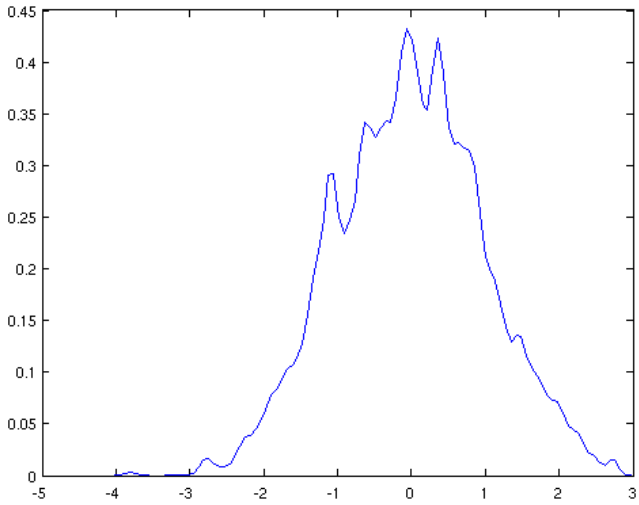
Hit-miss

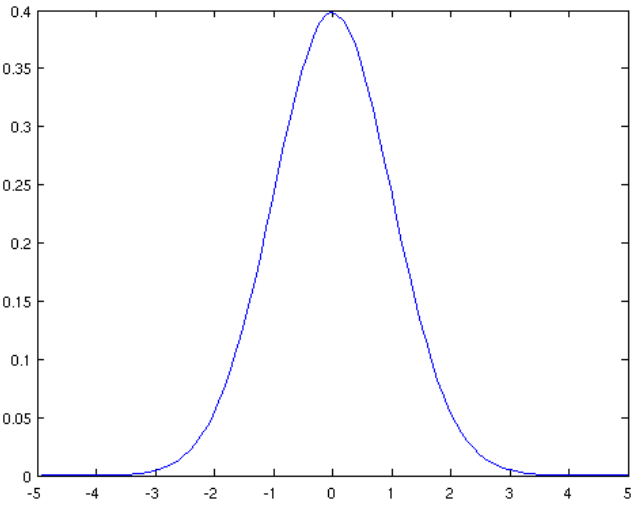
Twister-Mersenne:
- Mitja: -0,00039
- Variància: 0,999

LCGa:
- Mitja: -0,0227
- Variància: 1.007

Tot i que sembla que ambdós generadors són prou bons només cal fer un cop d'ull a l'histograma per veure que el LCG aplicat a hit-miss proporciona una sortida molt dolenta (quasi bé no és una gaussiana, tot i que els moments són els esperats).





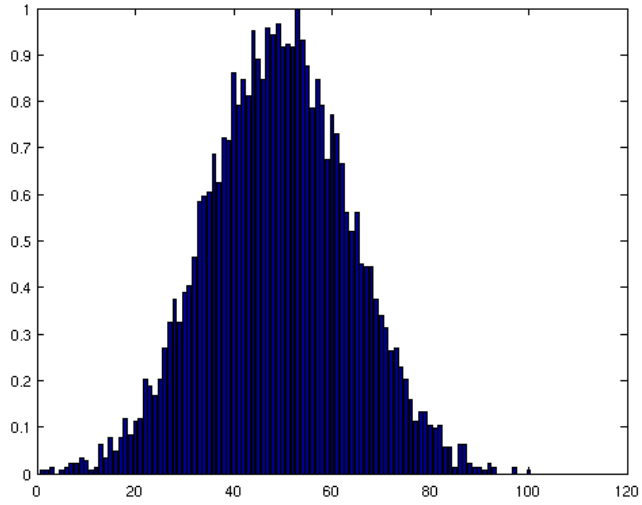


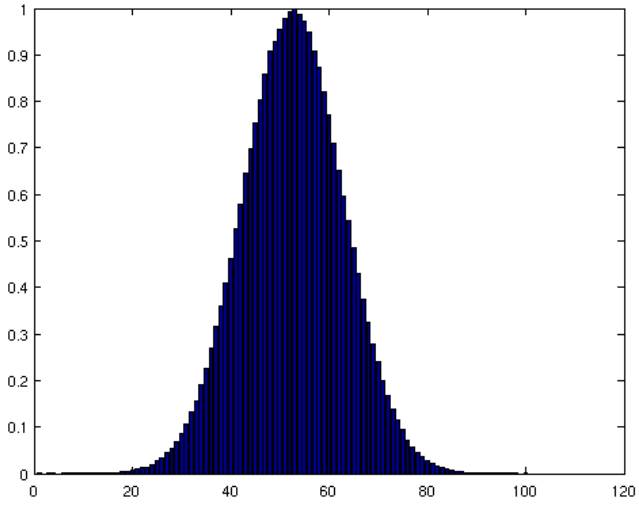
Box Muller

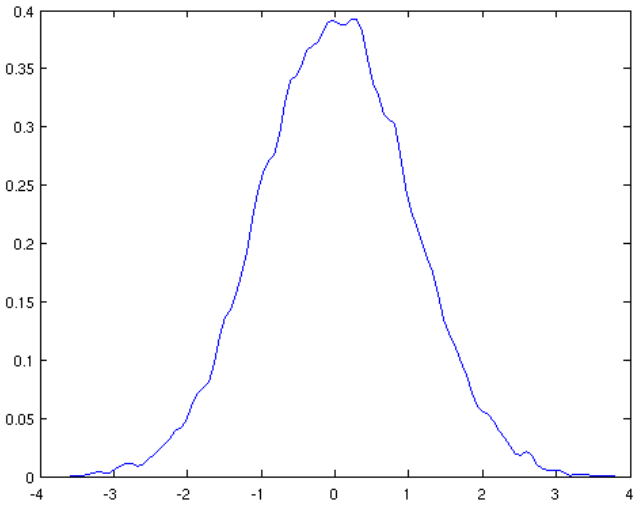
Twister-Mersenne:
- Mitja: -0,00022
- Variància: 1,0014

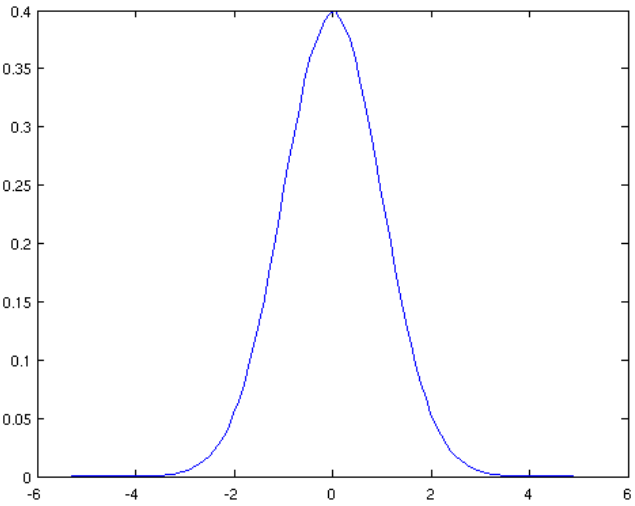
LCGa:
- Mitja: 0,0128
- Variància: 1.011

En aquest cas els moments obtinguts són iguals o lleugerament més dolents que en el cas anterior però podem veure com ara l'histograma és molt semblant a l'esperat en el cas del LCG. Aquest algorisme és capaç de generar una bona gaussiana amb valors aleatoris de poca qualitat i molt repetitius. És millor que el hit/miss en general i especialment en cost computacional (és determinista).







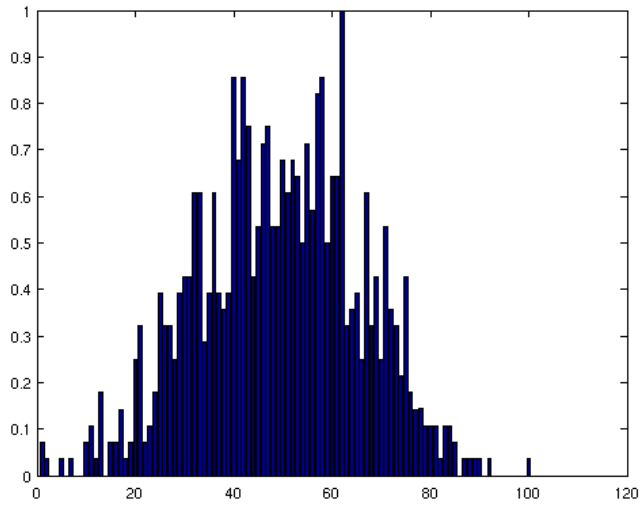


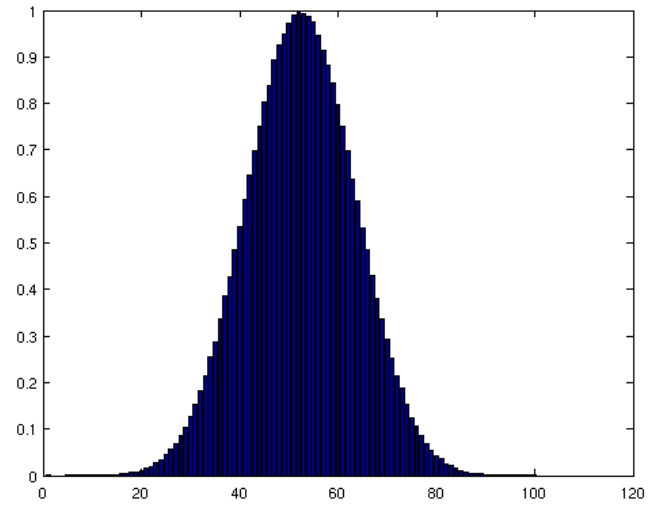
Convolució (n=12)

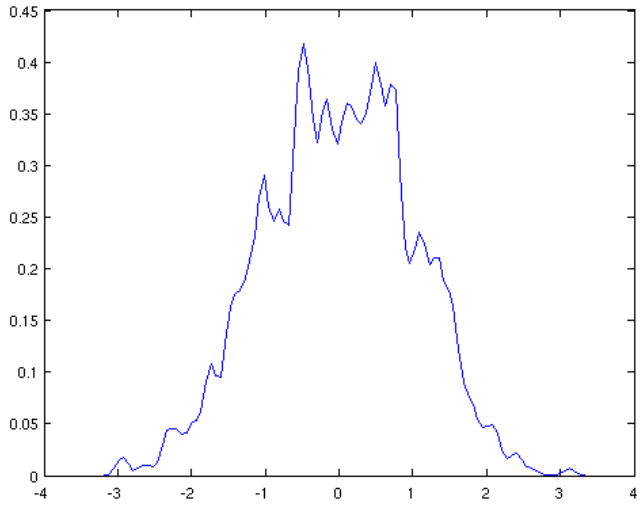
Twister-Mersenne:
- Mitja: -0.00039
- Variància: 0,999

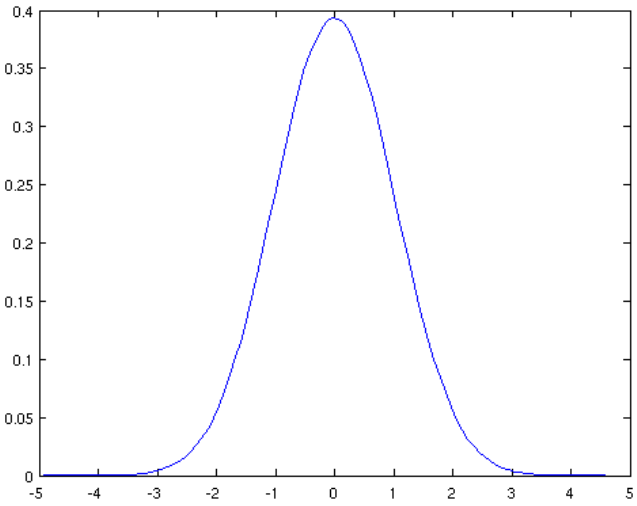
LCGa:
- Mitja: -0.0224
- Variància: 1.006

En aquest cas succeeix el mateix que al primer apartat. Tot i obtenir una bona estadística es veu a l'histograma que la funció no és una bona gaussiana, hi ha massa soroll (molts pics). El Twister es comporta correctament, pel que l'algorisme és bo, simplement el LCG no és capaç de produir valors suficientment bons i poc repetitius (els pics són justament aquests valors que es repeteixen tant al LCG).







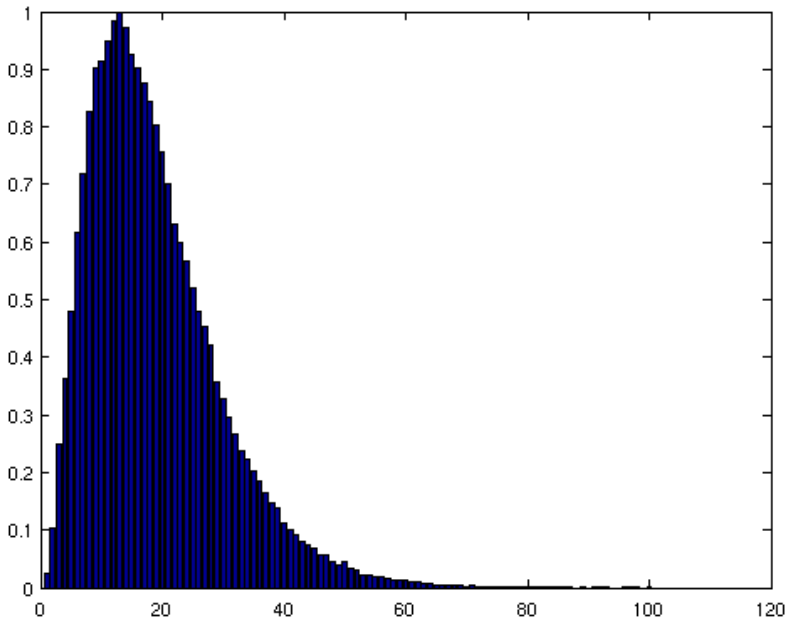


· Exercici 7

En aquest experiment fem ús del següent codi per a simular la suma de diverses exponencials (o la convolució d'aquestes).

```
double lt3_convolucion_exponencial(double media, double numExp) {
   int nume = numExp;
   double resultat = 0;
   while (nume--) {
         resultat += lt3_exponencial(media);
   }
   return resultat;
}
```

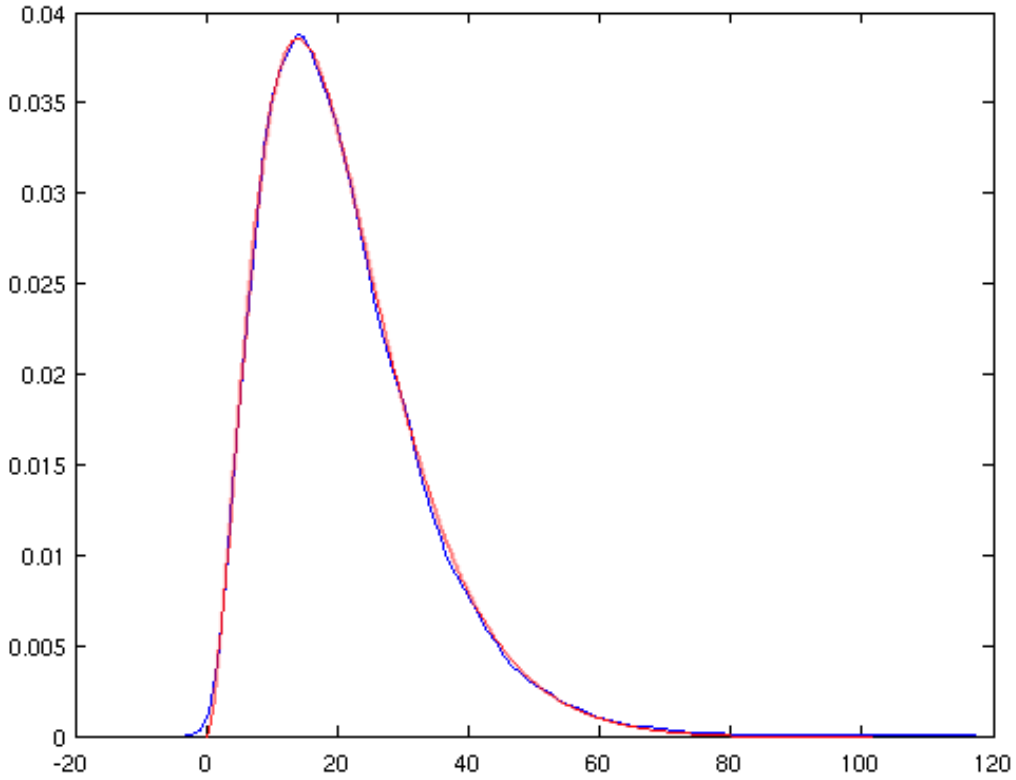
Realitzem l'experiment i veiem el seu histograma:



Com es pot veure l'histograma és la funció gamma d'ordre 3. Per corroborar això fem ús de la funció de matlab anomenada “gamfit” que ens dóna els paràmetres d'ajust de la funció gamma. El resultat és:

α: 2,9895 ~ 3
β: 7,0055 ~ 7

Com es pot veure l'ajust és molt bo, ja que quasi bé no cal arrodonir els valors per a obtenir els correctes. Ho podem veure també a l'ajustament gràfic entre les dues corbes, l'esperada (vermell) i l'obtinguda (blau):



· Exercici 8

El codi que hem escrit per a l'exercici és el següent:

```
double lt3_chi2(double media, double varianza, double k) {
     int k2 = k;
     double resultat = 0;
```

```
        while (k2--) {
             double chi = lt3_normal(media,varianza,1);
             resultat += chi*chi;
        }
        return resultat;
   }
```

Per al cas particular de k=2 la distribució es converteix en una exponencial. Només cal substituir k=2 a l'expressió general per a veure-ho. Tot seguit comprovem els valors per als percentils i els comparem:

Resultat experimental:

	5,00%	10,00%	95,00%
4	9,47	7,78	0,72
6	12,61	10,65	1,64
13	22,35	19,82	5,86
600	658,38	644,74	544,01

Resultat teòric:

	5,00%	10,00%	95,00%
4	9,4877	7,7794	0
6	12,5916	0	1,6354
13	22,3620	19,8119	5,8919
600	658,0936	644,8004	544,1801

Com es pot veure comparant les dues taules els resultats són quasi idèntics. Donem per bona la funció.

· Exercici 9

Per a realitzar el test de Pearson hem preparat una petita funció de matlab que realitza els tests i ens mostra els resultats. Després, manualment, comprovem a la taula si està per sota o per sobre del valor.

```
function pearson_test_uniform(v,n)
    % v mostres, n categories
    histo = hist(v,n);
    % V.A. uniforme, valor esperat = 1/num cat
    valor_esp = length(v)./n
    % Calcul
    resta = histo-valor_esp
    % Suma al cuadrat
    suma = 0;
    for i=1:n
        suma = suma + resta(i)*resta(i);
    end
    suma./valor_esp
end
```

La taula mostra el valor de k per als diferents valors de “r” i per als dos algorismes triats.

	Twister	LCG
r=2	0,44	0,169
r=6	2,60	27,25
r=11	11,72	51,00
r=24	27,16	108,26
r=51	51,19	232,80
r=101	88,33	510,93

Comprovem que el Twister supera amb un bon marge el test de Pearson per a diverses categories. En canvi el LCG no el supera, pel que podem corroborar la nostra hipòtesi que les mostres que ens dóna no són estadísticament independents.