Generating data from a probability distribution function

M. Gintner

In this note I explain a method for generating data distributed by a probability distribution function $f(x)$. Then the method is applied to the half-normal distribution function.

Let us define a cummulative distribution function (CDF)

$$F(\xi) = \int_{-\infty}^{\xi} f(x)\, dx. \tag{1}$$

It gives probability that variable x will be found in the interval $(-\infty, \xi)$. The situation is depicted in Fig. 1. Since the total probability is 1 the distribution function $f(x)$ is normalized which results in $F(\infty) = 1$.

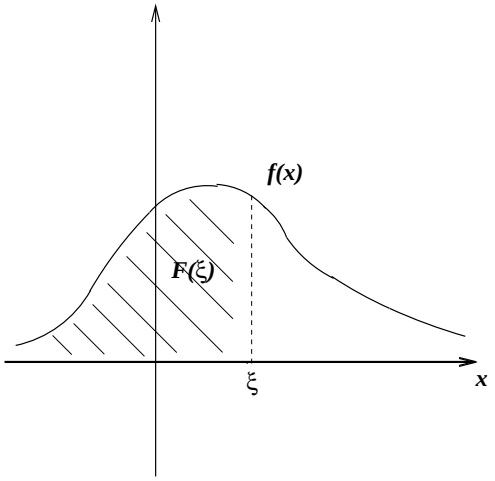

Figure 1: The probability distribution function and its CDF.

The probability that $\xi_1 < x < \xi_2$ is given as

$$\Delta q \equiv F(\xi_2) - F(\xi_1) = \int_{\xi_1}^{\xi_2} f(x)\, dx, \tag{2}$$

which coincides with the area under $f(x)$ bounded within the interval (ξ_1, ξ_2). Let us divide the x-axis into intervals with equal areas Δq under the probability curve $f(x)$ as is shown in Fig. 2. The values of x randomly generated according to $f(x)$ must be evenly distributed among these intervals. Since this must be true for the equiareal intervals of any size (smallness) it implies that all areas closed between neighbouring data points should be equal to each other. In other words, ξ's have to be distributed in such a way that $F(\xi)$'s are distributed uniformly. So if we generate uniformly distributed values $\{q_k\}_{k=1}^n \subset (0, 1)$ then values $\{\xi(q_k)\}_{k=1}^n$, where $\xi(q)$ is the inverse function of (1), are distributed according to $f(x)$. This is shown in Fig. 3.

We use the method described above for generating values distributed according to the Half-Normal Distribution (HND). The Normal Distribution (ND) is given by the function

$$N(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right], \tag{3}$$

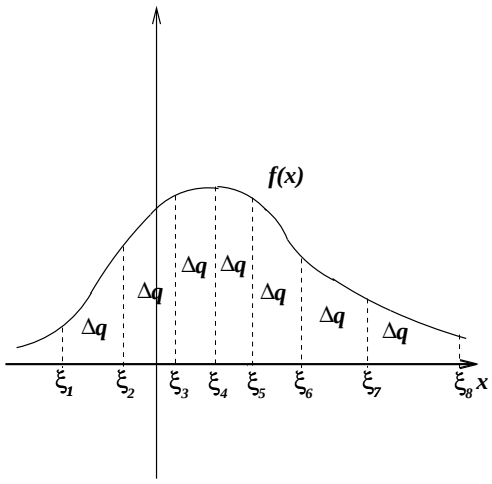

Figure 2: The division of the x-axis into intervals with equal areas Δq under the distribution function $f(x)$.

where $\mu = \langle x \rangle$ is the mean value of x and σ is its variance, $\sigma^2 = \langle x^2 \rangle - \langle x \rangle^2$. HND is obtained when we consider one half of ND only and normalize it

$$N_{1/2}(x;\mu,\sigma) = \begin{cases} 2N(x;\mu,\sigma), & x \geq \mu \\ 0, & x < \mu \end{cases} . \tag{4}$$

Of course, μ and σ denote neither the mean value nor the variance of x for $N_{1/2}$. In this case the mean value is

$$\langle x \rangle = \sqrt{\frac{2}{\pi}}\sigma. \tag{5}$$

To get an explicit analytic formula for CDF of $N_{1/2}$ we use the following trick. Let us define a 2-dim distribution

$$P(x,y) = \begin{cases} N_{1/2}(x;\mu,\sigma)N_{1/2}(y;\mu,\sigma), & x,y \geq \mu \\ 0, & x \vee y < \mu \end{cases} . \tag{6}$$

Next we switch to the polar coordinates (r,φ)

$$x = r\cos\varphi + \mu, \quad y = r\sin\varphi + \mu. \tag{7}$$

$$r = \sqrt{(x-\mu)^2 + (y-\mu)^2}, \quad \tan\varphi = \frac{y-\mu}{x-\mu}. \tag{8}$$

Since $dxdy = rdrd\varphi$ we get

$$P(x,y)\,dxdy = R(r)\Phi(\varphi)\,drd\varphi,$$

where

$$R(r) = \frac{r}{\sigma^2}\exp\left(-\frac{r^2}{2\sigma^2}\right), \quad r \in \langle 0,\infty); \qquad \Phi(\varphi) = \begin{cases} 2/\pi, & \varphi \in \langle 0,\pi/2\rangle \\ 0, & \varphi \in (\pi/2, 2\pi) \end{cases} . \tag{9}$$

The variable φ is distributed uniformly over the interval $\langle 0,\pi/2\rangle$. When we make the substitution $z = r^2/2$ we obtain

$$R(r)\,dr = f(z)\,dz, \quad z \in \langle 0,\infty),$$

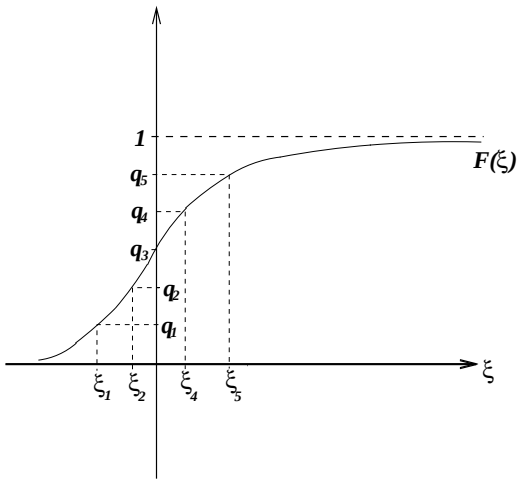

Figure 3: The $f(x)$-distributed values of ξ are obtained from uniformly distributed q's, $q \in (0,1)$, through the inverse mapping F^{-1}.

where $dz = rdr$, and

$$f(z) = \frac{1}{\sigma^2} \exp\left(-\frac{z}{2\sigma^2}\right). \tag{10}$$

The CDF of $f(z)$ is

$$F(\xi) = \int_0^{\xi} f(z)\, dz = 1 - e^{-\xi/\sigma^2} \equiv q. \tag{11}$$

The inverse relation provides a value ξ of the variable z which corresponds to a given $q \in \langle 0, 1)$

$$\xi = \sigma^2 \ln \frac{1}{1-q}.$$

From the relation between z and r we get $r = r(q)$

$$r = \sqrt{2z} = \sigma \sqrt{2 \ln \frac{1}{1-q}}. \tag{12}$$

So when we randomly generate q uniformly distributed over the interval $q \in \langle 0, 1)$ we obtain r distributed according to $R(r)$ in (9). In addition, we will randomly generate φ uniformly distributed over the interval $\langle 0, \pi/2\rangle$. From there we can obtain a random x distributed by HND (4) when we plug $r(q)$ and φ in one of the transformation relation between the polar and cartesian coordinates (7)

$$x(q, \varphi) = r(q) \cos\varphi + \mu = \sigma[-2\ln(1-q)]^{1/2} \cos\varphi + \mu, \quad q \in \langle 0, 1), \quad \varphi \in \langle 0, \pi/2\rangle \tag{13}$$

or

$$y(q, \varphi) = r(q) \sin\varphi + \mu = \sigma[-2\ln(1-q)]^{1/2} \sin\varphi + \mu, \quad q \in \langle 0, 1), \quad \varphi \in \langle 0, \pi/2\rangle. \tag{14}$$

Below is the C++ code which generates random values distributed according to the Half-Normal Distribution (4) using the relation (13).

```
// ************halfnormalgen.cpp***************
// ********************************************
// This program generates random numbers
// from "mu" to "+infinity" distributed
// by the half-normal distribution based
// on the normal distribution N(mu,sigma).
// RND generator from NumRecC is used.
// It generates 'M' sets of 'N' values.
//
// "mu" ...mean value of N(mu,sigma)
// "sigma" ...variance of N(mu,sigma)
// ********************************************
// author: M.Gintner
// date: Feb 26, 2007
// ********************************************

#include <cstdlib>
#include <iostream>
#include <fstream>
#include <math.h>

#define PI 3.1415927

using namespace std;

// ********************************************
// here is the random generator from Numerical Recipes in C (ran2)
// first the parameters for the generator

//#define KAS_RND_IM1 2147483564 // Borisova verzia
#define KAS_RND_IM1 2147483563 // NumRecipC verzia
#define KAS_RND_IM2 2147483399
#define KAS_RND_AM (1.0/KAS_RND_IM1)
#define KAS_RND_IMM1 (KAS_RND_IM1-1)
#define KAS_RND_IA1 40014
#define KAS_RND_IA2 40692
#define KAS_RND_IQ1 53668
#define KAS_RND_IQ2 52774
#define KAS_RND_IR1 12211
#define KAS_RND_IR2 3791
#define KAS_RND_NTAB 32
#define KAS_RND_NDIV (1+KAS_RND_IMM1/KAS_RND_NTAB)
#define KAS_RND_EPS 1.2e-7
#define KAS_RND_RNMX (1.0-KAS_RND_EPS)

/* the random generator routine */
```

```
double KAS_rndm(long *idum)
{
   int j;
   long k;
   static long idum2=123456789;
   static long iy=0;
   static long iv[KAS_RND_NTAB];
   double temp;

//   cout << "*idum = " << *idum << "\n";
   if (*idum <= 0){
     if (-(*idum) < 1) *idum = 1;
     else *idum = -(*idum);
     idum2 = (*idum);
     for (j=KAS_RND_NTAB+7;j>=0;j--){
       k=(*idum)/KAS_RND_IQ1;
       *idum = KAS_RND_IA1*(*idum-k*KAS_RND_IQ1)-k*KAS_RND_IR1;
       if (*idum < 0) *idum += KAS_RND_IM1;
       if (j < KAS_RND_NTAB) iv[j] = *idum;
     }
     iy = iv[0];
   }
   k = (*idum)/KAS_RND_IQ1;
   *idum = KAS_RND_IA1*(*idum-k*KAS_RND_IQ1)-k*KAS_RND_IR1;
   if (*idum < 0) *idum += KAS_RND_IM1;
   k = idum2/KAS_RND_IQ2;
   idum2 = KAS_RND_IA2*(idum2-k*KAS_RND_IQ2)-k*KAS_RND_IR2;
   if (idum2 < 0) idum2 += KAS_RND_IM2;
   j = iy/KAS_RND_NDIV;
   iy = iv[j] - idum2;
   iv[j] = *idum;
   if (iy < 1) iy += KAS_RND_IMM1;
   if ((temp=KAS_RND_AM*iy) > KAS_RND_RNMX) return KAS_RND_RNMX;
   else return temp;
}
// ********************************************
// Function to generate random numbers from <0,1>:
double ran(long *idum)
       {
        return KAS_rndm(idum);
       }
// ********************************************
// Function to get random value of angle "phi"
// (phiMin <= phi < phiMax):
double RNDphi( double phiMin, double phiMax, long *idum)
       {
       return phiMin+(phiMax-phiMin)*ran(idum);
       }
// ********************************************
// Function to get random value of "r"
```

```
// (0 < r < +infinity):
double RNDr (double sigma, long *idum)
       {
       double q;
       do{ q = ran(idum); } while( q == 1.0 );
       return sigma*sqrt( -2.0*log(1-q) );
       }
// ****************************************

int main()
{
    int i,N;
    long *idum;
    double rnd,r,phi,mu,sigma;
    int j=1,M;

    *idum=-7;

// input of initial parameters through the standard device:
    cout << "****************************************\n";
    cout << "This program generates datasets of\n";
    cout << "half-normal distributed random values.\n";
    cout << "The half-normal distribution considered here\n";
    cout << "is the normalized right-hand half of\n";
    cout << "the normal distribution N(mu,sigma), where\n";
    cout << "‘mu’ is the mean value and ‘sigma’ is the variance.\n";
    cout << "****************************************\n";
    cout << "the initial value of the generator seed ‘*idum’ = " << *idum << "\n";
    cout << "INPUT:\n";
    cout << "the parameter ‘mu’ = ";
    cin >> mu;
    cout << "the parameter ‘sigma’ = ";
    cin >> sigma;
    cout << "the number of datasets = ";
    cin >> M;
    cout << "the number of random generated values in each dataset = ";
    cin >> N;

// opening output file for rnd numbers:
    ofstream output;  // define the write-only file object "output"
    output.open ("data.tmp"); // associate physical file "data.tmp"
                              // with the object "output"

    cout << "-------------------------\n";
    cout << "generating the values...\n";

// generating half-normal distributed random values in datasets:
    do{
      output << "{";
      for(i=1;i<=N;++i)
```

```
         {
          phi = RNDphi(0,PI/2,idum);
          r = RNDr(sigma,idum);
          rnd = r*cos(phi)+mu;
//          cout << "rnd(" << i << ") = " << rnd << "\n"; // output to screen
          output << rnd << (i==N ? "}\n" : ",");
         };
       j++;
//       cout << "-------------------------\n"; // output to screen
       }
    while(j<=M);

// closing output file for rnd numbers:
    output.close(); // disassociate physical file "data.tmp"
                    // with the object "output"

    cout << "...has finished\n\n";
    cout << "The generated values has been written into the file 'data.tmp'.\n";
    cout << "The next run of the program will overwrite the contents of the file.\n\n";
    cout << "In the next run the random number generator\n";
    cout << "will generate the same set of random numbers\n";
    cout << "unless the value of '*idum' in the program is changed.\n\n";

    system("PAUSE"); // Press any key to continue... (comment out in Linux)
    return 0;
}
// ******************************************
```

This is a MATHEMATICA code which calculates the Half-Normal cummulative distribution function and compares it to half-normal generated data obtained by the C++ code above. The comparison is done in the graphical form. The data are plotted in a percentile plot. Figure 4 shows an example of the graph.

```
(* Finds the current directory: *)
Directory[]

(* Sets the required directory: *)
SetDirectory["c:/MinGW/Dev-Cpp/miki/half-normal"]

(* Defines Half-normal distribution function:   *)
ff[x_,mu_,sigma_]:=2/(sigma*Sqrt[2*Pi])*Exp[-(x-mu)^2/(2*sigma^2)];

(* Defines cumulative Half-normal DF: *)
mu=0;
sigma=1;
FF[eta_] := Integrate[ff[x, mu, sigma], {x, mu, eta}];

(* Graph of Half-normal CDF: *)
p1 = Plot[FF[x], {x, mu, 4*sigma}];

(* Reads in the generated dataset and sorts it in ascending manner: *)
dataset = Read["data.txt"];
dataset = Sort[dataset, Less];

(* Percentile graph of generated data: *)
n=Length[dataset];
perc = Table[{data[[i]], i/(n + 1)}, {i, n}];
p2=ListPlot[perc,PlotJoined->True];

(* Show plots p1 and p2 in one graph: *)
Show[p1,p2]

(* Export the graph in a GIF format: *)
Display["halfnormal200.gif", Show[p1, p2],"GIF"];

(* Export the graph in an EPS format: *)
Display["halfnormal200.eps",Show[p1,p2],"EPS"];
```

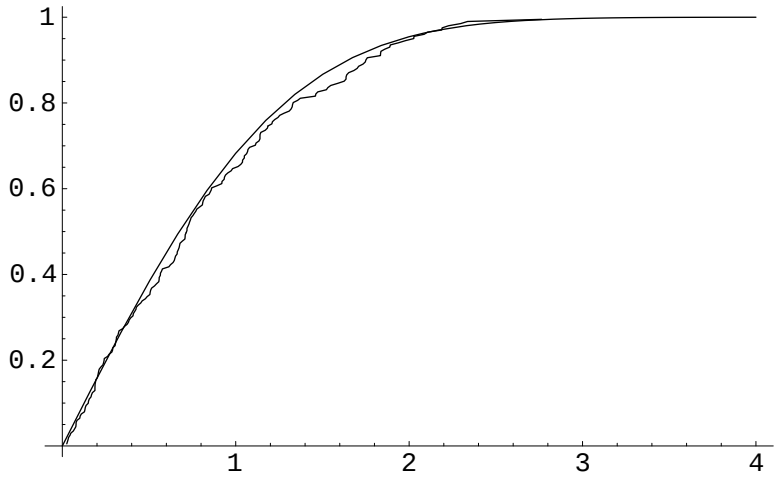

Figure 4: The comparison of data generated by the C++ code to the half-normal CDF with $\mu = 0$, $\sigma = 1$. There were 200 values generated and plotted in the percentile plot. The smooth plot represents CDF, the zigzagged one represents the data.