Metropolis Hasting

Overview:
In statistics and statistical physics, the Metropolis Hastings algorithm is a Markov chain Monte Carlo (MCMC) method for obtaining a sequence of random samples from a probability distribution from which direct sampling is difficult.

By M.Aek Progs AD

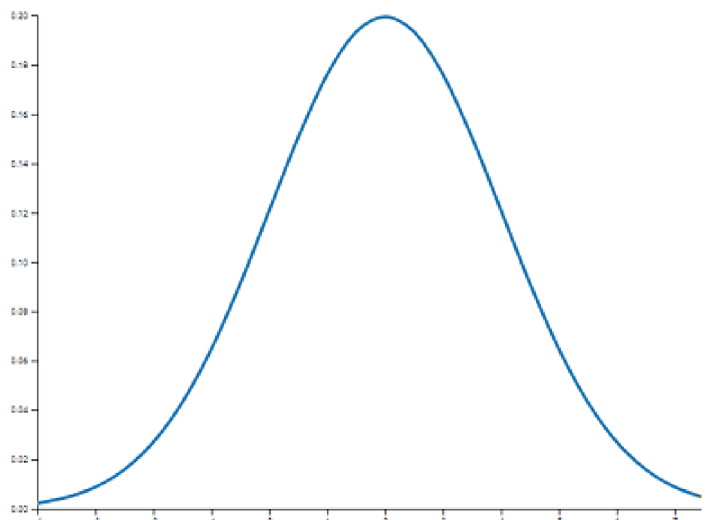
https://www.facebook.com/M.Aek.Progs.Angedevil.AD/

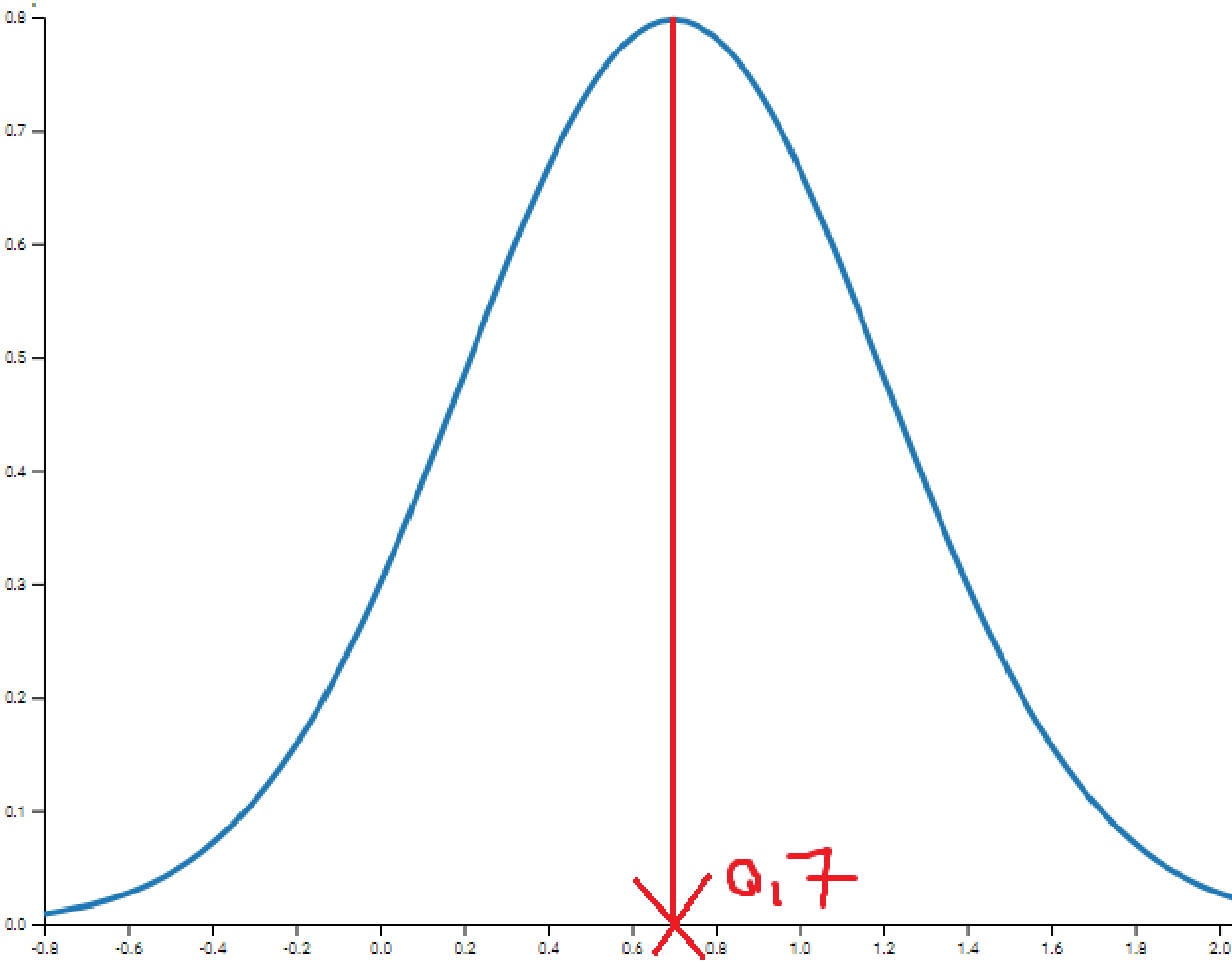
https://www.youtube.com/channel/UC6AhJIORlsp56XDwqfNSTsg

https://github.com/Angedevil-AD

- Choose an arbitrary point ($\mathcal{X}t$) to be the first observation in the sample. -

choose an arbitrary probability density g(x | y) ....sometimes written Q(x | y) that suggests a candidate for the next sample value x, given the previous sample value y. A usual choice is to let g (x | y) be a Gaussian distribution centered at y, so that points closer to y are more likely to be visited next.



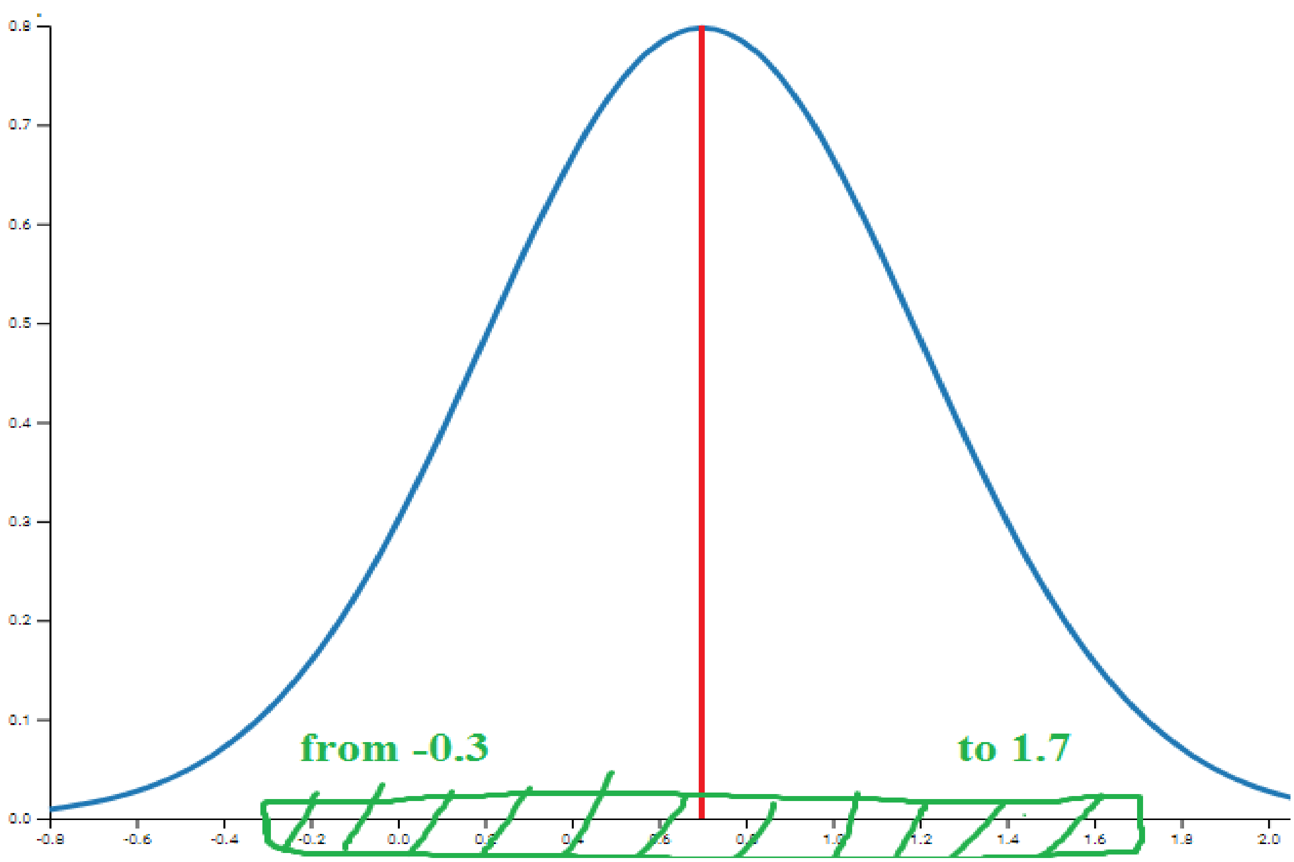


lets Xt = 0.7   Sigma = 0.5
get Points from (center -SIGMA) to (center + 2SIGMA)
0.7 - 2SIGMA   <---> 0.7 + 2SIGMA
*0.7 - 2*0.5   <-------> 0.7 + 2*0.5*
-0.3   <---->   1.7
there are thousens of propability between -0.3 & 1.7

lets take 200 point from -0.3 to 1.7
incresement = 0.01

propabilities:

-0.3000 -0.2900 -0.2800 -0.2700 -0.2600 -0.2500 -0.2400 -0.2300 -0.2200
-0.2000 -0.1900 -0.1800 -0.1700 -0.1600 -0.1500 -0.1400 -0.1300 -0.1200
-0.1000 -0.0900 -0.0800 -0.0700 -0.0600 -0.0500 -0.0400 -0.0300 -0.0200
0.0000 0.0100 0.0200 0.0300 0.0400 0.0500 0.0600 0.0700 0.0800 0.0900
0.1000 0.1100 0.1200 0.1300 0.1400 0.1500 0.1600 0.1700 0.1800 0.1900
0.2000 0.2100 0.2200 0.2300 0.2400 0.2500 0.2600 0.2700 0.2800 0.2900
0.3000 0.3100 0.3200 0.3300 0.3400 0.3500 0.3600 0.3700 0.3800 0.3900
0.4000 0.4100 0.4200 0.4300 0.4400 0.4500 0.4600 0.4700 0.4800 0.4900
0.5000 0.5100 0.5200 0.5300 0.5400 0.5500 0.5600 0.5700 0.5800 0.5900
0.6000 0.6100 0.6200 0.6300 0.6400 0.6500 0.6600 0.6700 0.6800 0.6900
0.7000 0.7100 0.7200 0.7300 0.7400 0.7500 0.7600 0.7700 0.7800 0.7900
0.8000 0.8100 0.8200 0.8300 0.8400 0.8500 0.8600 0.8700 0.8800 0.8900
0.9000 0.9100 0.9200 0.9300 0.9400 0.9500 0.9600 0.9700 0.9800 0.9900
1.0000 1.0100 1.0200 1.0300 1.0400 1.0500 1.0600 1.0700 1.0800 1.0900
1.1000 1.1100 1.1200 1.1300 1.1400 1.1500 1.1600 1.1700 1.1800 1.1900
1.2000 1.2100 1.2200 1.2300 1.2400 1.2500 1.2600 1.2700 1.2800 1.2900
1.3000 1.3100 1.3200 1.3300 1.3400 1.3500 1.3600 1.3700 1.3800 1.3900
1.4000 1.4100 1.4200 1.4300 1.4400 1.4500 1.4600 1.4700 1.4800 1.4900
1.5000 1.5100 1.5200 1.5300 1.5400 1.5500 1.5600 1.5700 1.5800 1.5900
1.6000 1.6100 1.6200 1.6300 1.6400 1.6500 1.6600 1.6700 1.6800 1.6900
1.7000



lets get a random point from propabilities.

random index .... ex ..99
look for index 99 in propabilities arr

point[99] = 0.6900 <----- the candidate

the algorithm say:
for each iteration t:
-Generate a candidate y for the next sample by picking from the distribution g(y , xt) ...
we, have xt = 0.7   and the candidate y = 0.6900

-Calculate the acceptance ratio a

f(x) is gaussian function

$$f(x) = \frac{1}{\sigma\sqrt{2\,\pi}}\,\mathrm{e}^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

lets gx = f(xt) and gy = f(y)

gx = gaussian_function(xt)   and   gy=gasian_function(y)

gx= 0.7979

gy = 0.7977

ratio a = gy / gx = 0.7977 / 0.7979 = 0.9998

-accept or reject condidate x':

-generate random number u , u belong [0 , 1]
-if u <= a , accept the condidate... by setting xt+1 = y
-if u >   a,   reject the candidate ... by setting x t+1 = xt

accept or reject ???
generate random number (0,1) ..
u =0.1400
a = 0.9998
u < a -----> Accept
xt = y----> xt = 0.6900

now move to iteration 2:

xt = 0.6900
generate candidate y:
y = 0.5900
calcul gauss:
gx = 0.7977
gy = 0.7788
Calculate ratio:
a = 0.7977/ 0.7788
a = 0.9763
accept or reject ???
generate random number (0,1)
u = 0.6200
u < a -----> Accept
x t = y = 0.5900

Iteration 3:
xt = 0.5900
generate candidate y:
y = 0.3000
calcul gauss:
gx = 0.7788
gy = 0.5794
Calculate ratio:
a = 0.7439
generate random number (0,1)
u = 0.4550
u < a -----> Accept
x t = y = 0.3000

iteration 4:

xt = 3000
generate candidate y:
y = 1.0800
calcul gauss:
gx = 0.5794
gy =0.5977
Calculate ratio:
a = 1.0317
generate random nuber (0,1)
u = 0.1650
u < a -----> Accept
x t = y = 1.0800
.
.
.
until Iteration N

c++ exemple:

```
for (itera = 0; itera<50;itera++)
{
printf("Iteration %d:\n\n",itera);
Q(x,y);
printf("Current X  : %.4f  \n",x);
printf("Candidate Y: %.4f  \n\n",y);

double gx = Gaussfunc(x);
double gy = Gaussfunc(y);

printf("gauss function current   x: %.4f  \n",gx);
printf("gauss function candidate y: %.4f  \n\n",gy);

a = gy / gx;
printf("calc ratio gy/gx : %.4f  \n\n",a);

u = (double)ran0_1() / 1000;
printf("randomiz 0..1 : %.4f  \n\n",u);

if (u <= a)
{
printf("Accept the candidate -> y  : %.4f  \n\n",y);
x = y;
y = 0.0;
printf("SWAPING  x = y = : %.4f  \n\n",x);
}

else
{
printf("Reject the candidate -> y  : %.4f  \n\n",y  );
x = x;
y = 0.0;
}
```

```
double Q(double X,double &Y)
{

double mean = X;
double stdev = 0.5;
double pts =0;

double startp = mean - 2*stdev;
double endp   = mean + 2*stdev;

unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
mt19937 rand_num(seed);
uniform_int_distribution<long long> dist(0, 200);
int index = dist(rand_num);
cout << " rand index: " << index<< endl;

pts = startp;
Y = pts+(0.01*index);

}
```

```
double Gaussfunc(double x)
{
double stdev = 0.5;
double u = 0.7;
return  (1 / (sqrt(2*PI)*stdev))  *   exp(   - ( ( pow(x-u,2) ) /  (2 * pow(stdev,2)  )     )   );
}
```

```
int ran0_1()
{
unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
mt19937 rand_num(seed)
uniform_int_distribution<long long> dist(0, 1000);

int idx = dist(rand_num) ;
return idx ;
}
```