```
85:       for(j=m;j<=n-i;j++) w[j]=w[j+1];
86:     }
87: }
```

In Line 73, a large value is given to `xmin`.

In Lines 78 – 84, given `i`, we search the ith minimum value.

in Line 85, remove the ith minimum data and reconstruct the new vector.

Quick Sort: In the previous sort algorithm, the computational number of times is given by $n + (n-1) + (n-2) + \cdots + 1 = n(n-1)/2$, which is $O(n^2)$. That is, computational time is proportional to n^2.

Computational time of the following sort algorithm, called the quick sort (クイック・ソート), is proprtional to $n \log n$.

The following sort algorithm is extremely faster than the previous one in order of $n/\log n$.

Quick Sort Algorithm

```
1: #include<stdio.h>
2: #include<math.h>
3: #include<time.h>
4:
5:   int ix=1,iy=1;
6:
7: void main()
8: {
9:   int i,n;
10:  int i025,i050,i500,i950,i975;
11:  float x[100001];
12:  float y[100001];
13:  float urnd(void);
14:  void  qsort(float x[],int l,int n);
15:  clock_t t0,t1;
16:  double  dt;
17:
18:  for(i=1;i<=10000;i++) urnd();
```

```
19:
20:   scanf("%d",&n);
21:
22:   for(i=1;i<=n;i++){
23:     x[i]=urnd();
24:     y[i]=x[i];
25:   }
26:
27:   t0=clock();
28:   qsort(y,1,n);
29:   t1=clock();
30:   dt=(t1-t0)/((double)CLOCKS_PER_SEC);
31: /*
32:   for(i=1;i<=n;i++) printf("%10d %10.8f %10.8f\n",i,x[i],y[i]);
33: */
34:   printf("Computational Time = %10.2lf\n",dt);
35:   i025=(int)( 0.025*(float)n );
36:   i050=(int)( 0.050*(float)n );
37:   i500=(int)( 0.500*(float)n );
38:   i950=(int)( 0.950*(float)n );
```

```
39:   i975=(int)( 0.975*(float)n );
40:   printf(" 2.5 percent point = %10.8f\n",(y[i025]+y[i025+1])/2.0 );
41:   printf(" 5   percent point = %10.8f\n",(y[i050]+y[i050+1])/2.0 );
42:   printf("50   percent point = %10.8f\n",(y[i500]+y[i500+1])/2.0 );
43:   printf("95   percent point = %10.8f\n",(y[i950]+y[i950+1])/2.0 );
44:   printf("97.5 percent point = %10.8f\n",(y[i975]+y[i975+1])/2.0 );
45: }
46: /* ================================================ */
47: float urnd(void)
48: {
49:   int    kx,ky;
50:   float  rn;
51: /*
52:   Input:
53:     ix, iy:  Seeds
54:   Output:
55:     rn: Uniform Random Draw U(0,1)
56: */
57:   kx=ix/53668;
58:   ix=40014*(ix-kx*53668)-kx*12211;
```

```
59:
60:   ky=iy/52774;
61:   iy=40692*(iy-ky*52774)-ky*3791;
62:
63:   rn=(float)(ix-iy)/2147483563.;
64:   rn-=(int)rn;
65:   if( rn<0.) rn++;
66:
67:   return rn;
68: }
69: /* ================================================== */
70: #define THRESHOLD 10
71: #define STACKSIZE 32
72:
73: void qsort(float a[],int first,int last)
74: /****************  Quick Sort  ****************/
75: /************ a[first],...,a[last] ************/
76: {
77:   int i,j,left,right,p;
78:   int leftstack[STACKSIZE],rightstack[STACKSIZE];
```

```
79:   float x,t;
80:   void inssort(float a[],int first,int last);
81:
82:   left=first; right=last; p=first;
83:   for(;;) {
84:     if( right-left<=THRESHOLD ) {
85:       if( p==first ) break;
86:       p--;
87:       left = leftstack[p];
88:       right=rightstack[p];
89:     }
90:     x=a[ (left+right)/2 ];
91:     i=left; j=right;
92:     for(;;) {
93:       while( a[i]<x ) i++;
94:       while( x<a[j] ) j--;
95:       if( i>=j ) break;
96:       t=a[i]; a[i]=a[j]; a[j]=t;
97:       i++; j--;
98:     }
```

```
99:     if( i-left>right-j ) {
100:      if( i-left>THRESHOLD ) {
101:        leftstack[p]=left;
102:        rightstack[p]=i-1;
103:        p++;
104:      }
105:      left=j+1;
106:    } else {
107:      if( right-j>THRESHOLD ) {
108:        leftstack[p]=j+1;
109:        rightstack[p]=right;
110:        p++;
111:      }
112:      right=i-1;
113:    }
114:  }
115:  inssort(a,first,last);
116: }
117: /* ------------------------------------------------------- */
118: void inssort(float a[],int first,int last)
```

```
119: {
120:   int i,j;
121:   float x;
122:
123:   for( i=first+1;i<=last;i++ ) {
124:     x=a[i];
125:     for( j=i-1;j>=first && a[j]>x;j-- ) a[j+1]=a[j];
126:       a[j+1]=x;
127:   }
128: }
```

2.9 Sampling Method III: Gibbs Sampling

The sampling methods introduced in Sections 2.8.1 – 2.8.3 can be applied to the cases of both univariate and multivariate distributions.

The Gibbs sampler in this section is the random number generation method in the multivariate cases.

The Gibbs sampler shows how to generate random draws from the unconditional densities under the situation that we can generate random draws from two conditional densities.

Geman and Geman (1984), Tanner and Wong (1987), Gelfand, Hills, Racine-Poon and Smith (1990), Gelfand and Smith (1990), Carlin and Polson (1991), Zeger and Karim (1991), Casella and George (1992), Gamerman (1997) and so on developed the Gibbs sampling theory.

Carlin, Polson and Stoffer (1992), Carter and Kohn (1994, 1996) and Geweke and Tanizaki (1999, 2001) applied the Gibbs sampler to the nonlinear and/or non-Gaussian state-space

models.

There are numerous other applications of the Gibbs sampler.

The Gibbs sampling theory is concisely described as follows.

We can deal with more than two random variables, but we consider two random variables X and Y in order to make things easier.

Two conditional density functions, $f_{x|y}(x|y)$ and $f_{y|x}(y|x)$, are assumed to be known, which denote the conditional distribution function of X given Y and that of Y given X, respectively.

Suppose that we can easily generate random draws of X from $f_{x|y}(x|y)$ and those of Y from $f_{y|x}(y|x)$.

However, consider the case where it is not easy to generate random draws from the joint density of X and Y, denoted by $f_{xy}(x, y)$.

In order to have the random draws of (X, Y) from the joint density $f_{xy}(x, y)$, we take the following procedure:

(i) Take the initial value of X as x_{-M}.

(ii) Given x_{i-1}, generate a random draw of Y, i.e., y_i, from $f(y|x_{i-1})$.

(iii) Given y_i, generate a random draw of X, i.e., x_i, from $f(x|y_i)$.

(iv) Repeat the procedure for $i = -M+1, -M+2, \cdots, 1$.

From the convergence theory of the Gibbs sampler, as M goes to infinity, we can regard x_1 and y_1 as random draws from $f_{xy}(x, y)$, which is a joint density function of X and Y.
M denotes the **burn-in period**, and the first M random draws, (x_i, y_i) for $i = -M+1, -M+2, \cdots, 0$, are excluded from further consideration.
When we want N random draws from $f_{xy}(x, y)$, Step (iv) should be replaced by Step (iv)', which is as follows.

(iv)' Repeat the procedure for $i = -M+1, -M+2, \cdots, N$.

As in the Metropolis-Hastings algorithm, the algorithm shown in Steps (i) – (iii) and (iv)'

is formulated as follows:

$$f_i(u) = \int f^*(u|v) f_{i-1}(v)\, \mathrm{d}v.$$

For convergence of the Gibbs sampler, we need to have the invariant distribution $f(u)$ which satisfies $f_i(u) = f_{i-1}(u) = f(u)$. If we have the reversibility condition, i.e.,

$$f^*(v|u) f(u) = f^*(u|v) f(v),$$

the random draws based on the Gibbs sampler converge to those from the invariant distribution, which implies that there exists the invariant distribution $f(u)$.

Therefore, in the Gibbs sampling algorithm, we have to find the transition distribution, i.e., $f^*(u|v)$.

Here, we consider that both u and v are bivariate vectors.

That is, $f^*(u|v)$ and $f_i(u)$ denote the bivariate distributions. x_i and y_i are generated from $f_i(u)$ through $f^*(u|v)$, given $f_{i-1}(v)$.

Note that $u = (u_1, u_2) = (x_i, y_i)$ is taken while $v = (v_1, v_2) = (x_{i-1}, y_{i-1})$ is set.

The transition distribution in the Gibbs sampler is taken as:

$$f^*(u|v) = f_{y|x}(u_2|u_1) f_{x|y}(u_1|v_2)$$

Thus, we can choose $f^*(u|v)$ as shown above.

Then, as i goes to infinity, (x_i, y_i) tends in distribution to a random vector whose joint density is $f_{xy}(x, y)$.

See, for example, Geman and Geman (1984) and Smith and Roberts (1993).

Furthermore, under the condition that there exists the invariant distribution, the basic result of the Gibbs sampler is as follows:

$$\frac{1}{N} \sum_{i=1}^{N} g(x_i, y_i) \longrightarrow \mathrm{E}(g(x, y)) = \iint g(x, y) f_{xy}(x, y)\, \mathrm{d}x\, \mathrm{d}y, \quad \text{as } N \longrightarrow \infty,$$

where $g(\cdot, \cdot)$ is a function.

The Gibbs sampler is a powerful tool in a Bayesian framework.

Based on the conditional densities, we can generate random draws from the joint density.

Remark 1: We have considered the bivariate case, but it is easily extended to the multivariate cases.

That is, it is possible to take multi-dimensional vectors for x and y.

Taking an example, as for the tri-variate random vector (X, Y, Z), if we generate the ith random draws from $f_{x|yz}(x|y_{i-1}, z_{i-1})$, $f_{y|xz}(y|x_i, z_{i-1})$ and $f_{z|xy}(z|x_i, y_i)$, sequentially, we can obtain the random draws from $f_{xyz}(x, y, z)$.

Remark 2: Let X, Y and Z be the random variables.

Take an example of the case where X is highly correlated with Y.

If we generate random draws from $f_{x|yz}(x|y, z)$, $f_{y|xz}(y|x, z)$ and $f_{z|xy}(z|x, y)$, it is known that convergence of the Gibbs sampler is slow.

In this case, without separating X and Y, random number generation from $f(x, y|z)$ and $f(z|x, y)$ yields better random draws from the joint density $f(x, y, z)$.

Example: Bivariate Normal Distribution: Consider two random variables, X and Y. From two conditional distributions $f(x|y)$ and $f(y|x)$, we make sure whether we can generare random draws of the joint ditribution $f(x, y)$.

Suppose that the joint distribution of X and Y is the following bivariate normal distribution:

$$f(x,y) = \Big(\frac{1}{\sqrt{2\pi}}\Big)^2 \begin{vmatrix} 1 & \rho \\ \rho & 1 \end{vmatrix}^{-1/2} \exp\Big(-\frac{1}{2}\begin{pmatrix} x & y \end{pmatrix}\begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}^{-1}\begin{pmatrix} x \\ y \end{pmatrix}\Big)$$
$$= \frac{1}{2\pi\sqrt{1-\rho^2}} \exp\Big(-\frac{1}{2(1-\rho^2)}(x^2 - 2\rho xy + y^2)\Big)$$

That is, we have $\mathrm{E}(X) = \mathrm{E}(Y) = 0$, $\mathrm{V}(X) = \mathrm{V}(Y) = 1$, and $\mathrm{Cov}(X, Y) = \rho$.

The marginal distribution of Y is given by:

$$f(y) = \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}y^2),$$

because of $\mathrm{E}(Y) = 0$ and $\mathrm{V}(Y) = 1$.

Thereore, the conditional distribution of X given Y is:

$$\begin{aligned} f(x|y) = \frac{f(x,y)}{f(y)} &= \frac{1}{\sqrt{2\pi(1-\rho^2)}} \exp\Big(-\frac{1}{2(1-\rho^2)}(x^2 - 2\rho xy + y^2) + \frac{1}{2}y^2\Big) \\ &= \frac{1}{\sqrt{2\pi(1-\rho^2)}} \exp\Big(-\frac{1}{2(1-\rho^2)}(x^2 - 2\rho xy + \rho^2 y^2)\Big) \\ &= \frac{1}{\sqrt{2\pi(1-\rho^2)}} \exp\Big(-\frac{1}{2(1-\rho^2)}(x - \rho y)^2\Big) \end{aligned}$$

which is the normal distribution with mean ρY and variance $1 - \rho^2$, i.e., $N(\rho Y,\ 1 - \rho^2)$.

Smilarly, $f(y|x)$ is given by $N(\rho X,\ 1 - \rho^2)$.

Thus, we obtain the following two conditional distributions:

$$X|Y \ \sim \ N(\rho Y,\ 1 - \rho^2), \qquad Y|X \ \sim \ N(\rho X,\ 1 - \rho^2).$$

From the two conditional distributions, we check $\begin{pmatrix} X \\ Y \end{pmatrix} \ \sim \ N\Big(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}\Big)$

Gibbs Sampler

```
1: #include <math.h>
2: #include <stdio.h>
3:
4:   int ix=1,iy=1;
5:
6: void main(){
7:
8:   float rho;
9:   long int i,m,n;
10:  double nrnd(void);
11:  double x,y;
12:  double x1=0.0,x2=0.0;
13:  double y1=0.0,y2=0.0;
14:  double xy1=0.0;
15:
16:  for(i=1;i<=10000;i++) nrnd();
17:
18:  scanf("%f%ld%ld",&rho,&m,&n);
```

```
19:
20:   x=0.0;
21:   y=0.0;
22:   for(i=-m+1;i<=n;i++){
23:     x=rho*y+sqrt( 1.-rho*rho )*nrnd();
24:     y=rho*x+sqrt( 1.-rho*rho )*nrnd();
25:     if( i >= 1 ){
26:       x1+=x/((double)n);
27:       x2+=x*x/((double)n);
28:       y1+=y/((double)n);
29:       y2+=y*y/((double)n);
30:       xy1+=x*y/((double)n);
31:     }
32:   }
33:
34:   printf("# of Burn-in      = %10ld\n",m);
35:   printf("# of Random Draws = %10ld\n",n);
36:   printf("Rho               = %7.2f\n",rho);
37:   printf("Mean       = ( %10.5lf, %10.5lf )\n",x1,y1);
38:   printf("Variance   = ( %10.5lf, %10.5lf )\n",x2,y2);
```

```
39:   printf("Covariance =   %10.5lf\n",xy1);
40:
41: }
42: /* ---------------------------------------------- */
43: double urnd(void)
44: {
45:   int    kx,ky;
46:   double rn;
47: /*
48:   Input:
49:     ix, iy:  Seeds
50:   Output:
51:     rn: Uniform Random Draw U(0,1)
52: */
53:   kx=ix/53668;
54:   ix=40014*(ix-kx*53668)-kx*12211;
55:
56:   ky=iy/52774;
57:   iy=40692*(iy-ky*52774)-ky*3791;
58:
```

```
59:   rn=(float)(ix-iy)/2147483563.;
60:   rn-=(int)rn;
61:   if( rn<0.) rn++;
62:
63:   return rn;
64: }
65: /* ------------------------------------------------ */
66: double nrnd(void)
67: {
68:   double rn,r1,r2;
69:   double pi=3.1415926535897932385;
70:   double urnd();
71:
72:   r1=urnd(); r2=urnd();
73:
74:   rn=sqrt( -2.*log(r1) )*sin( 2.*pi*r2 );
75:
76:   return rn;
77: }
```

References

Carlin, B.P. and Polson, N.G., 1991, "Inference for Nonconjugate Bayesian Models Using the Gibbs Sampler," *Canadian Journal of Statistics*, Vol.19, pp.399 – 405.

Carlin, B.P., Polson, N.G. and Stoffer, D.S., 1992, "A Monte Carlo Approach to Nonnormal and Nonlinear State Space Modeling," *Journal of the American Statistical Association*, Vol.87, No.418, pp.493 – 500.

Carter, C.K. and Kohn, R., 1994, "On Gibbs Sampling for State Space Models," *Biometrika*, Vol.81, No.3, pp.541 – 553.

Carter, C.K. and Kohn, R., 1996, "Markov Chain Monte Carlo in Conditionally Gaussian State Space Models," *Biometrika*, Vol.83, No.3, pp.589 – 601.

Casella, G. and George, E.I., 1992, "Explaining the Gibbs Sampler," *The American Statistician*, Vol.46, pp.167 – 174.

Chib, S. and Greenberg, E., 1995, "Understanding the Metropolis-Hastings Algorithm," *The American Statistician*, Vol.49, No.4, pp.327 – 335.

Gamerman, D., 1997, *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*, Chapman & Hall.

Gelfand, A.E. and Smith, A.F.M., 1990, "Sampling-Based Approaches to Calculating Marginal Densities," *Journal of the American Statistical Association*, Vol.85, No.410, pp.398 – 409.

Liu, J.S., 1996, "Metropolized Independent Sampling with Comparisons to Rejection Sampling and Importance Sampling," *Statistics and Computing*, Vol.6, pp.113 – 119.

Mengersen, K.L., Robert, C.P. and Guihenneuc-Jouyaux, C., 1999, "MCMC Convergence Diagnostics: A Reviewww," in *Bayesian Statistics, Vol.6*, edited by Bernardo, J.M., Berger, J.O., Dawid, A.P. and Smith, A.F.M., pp.514 – 440 (with discussion), Oxford University Press.

Smith, A.F.M. and Gelfand, A.E., 1992, "Bayesian Statistics without Tears: A Sampling-Resampling Perspective," *The American Statistician*, Vol.46, No.2, pp.84 – 88.

Smith, A.F.M. and Roberts, G.O., 1993, "Bayesian Computation via Gibbs Sampler and Related Markov Chain Monte Carlo Methods," *Journal of the Royal Statistical Society*, Ser.B, Vol.55, No.1, pp.3 – 23.

Tanner, M.A. and Wong, W.H., 1987, "The Calculation of Posterior Distributions by Data Augmentation," *Journal of the American Statistical Association*, Vol.82, No.398, pp.528 – 550 (with discussion).

Tierney, L., 1994, "Markov Chains for Exploring Posterior Distributions," *The Annals of Statistics*, Vol.22, No.4, pp.1701 – 1762.