Example of Metropolis-Hasrtings (MH) Algorithm: Beta Distribution $B(\alpha,\ \beta)$:

$$f(x) = \begin{cases} \dfrac{1}{B(\alpha,\ \beta)} x^{\alpha-1}(1-x)^{\beta-1}, & \text{for } 0 < x < 1, \\ 0, & \text{otherwise,} \end{cases}$$

for $\alpha > 0$ and $\beta > 0$.

$f_*(x)$ is taken as the uniform distribution between zero and one:

$$f_*(x) = \begin{cases} 1, & \text{for } 0 < x < 1, \\ 0, & \text{otherwise,} \end{cases}$$

$$q(x) = \frac{f(x)}{f_*(x)} = \frac{1}{B(\alpha,\ \beta)} x^{\alpha-1}(1-x)^{\beta-1}$$

$\omega(x_{i-1}, x^*)$ is given by:

$$\omega(x_{i-1}, x^*) = \frac{q(x^*)}{q(x_{i-1})} = \frac{x^{*\alpha-1}(1-x^*)^{\beta-1}}{x_{i-1}^{\alpha-1}(1-x_{i-1})^{\beta-1}}$$

From computational viewpoint, $\omega(x_{i-1}, x^*)$ is computed as:

$$\omega(x_{i-1}, x^*) = \exp\Big((\alpha - 1)\log(x^*) + (\beta - 1)\log(1 - x^*) - (\alpha - 1)\log(x_{i-1}) - (\beta - 1)\log(1 - x_{i-1})\Big)$$

is recommended.

B(a,b) Distribution

```
#include <math.h>
#include <stdio.h>

  int ix=1,iy=1;

void main(){

  float  a,b;
  long int i,j,n,m;
  double urnd(void);
  double x,x0,u,w;
```

```
double x1=0.0,x2=0.0;

for(i=1;i<=10000;i++) urnd();

scanf("%f%f%ld%ld",&a,&b,&m,&n);

x=0.5;
for(i=-m+1;i<=n;i++){
  x0=urnd();
  w=exp( (a-1.)*log(x0)+(b-1.)*log(1.-x0)
        -(a-1.)*log(x )-(b-1.)*log(1.-x ) );
  u=urnd();
  if( u<w ) x=x0;
  if( i >= 1 ){
    x1+=x/((double)n);
    x2+=x*x/((double)n);
  }
}

printf("# of Burn-in      = %10ld\n",m);
```

```
  printf("# of Random Draws = %10ld\n",n);
  printf("Parameters = (%7.1f,%7.1f)\n",a,b);
  printf("Mean     = %10.5lf, which should be close to %10.5f\n"
         ,x1,a/(a+b));
  printf("Variance = %10.5lf, which should be close to %10.5f\n"
         ,x2-x1*x1,a*b/((a+b)*(a+b)*(a+b+1.)));

}
/* ------------------------------------------------ */
double urnd(void)
{
   int    kx,ky;
   double rn;
/*
   Input:
     ix, iy:  Seeds
   Output:
     rn: Uniform Random Draw U(0,1)
*/
   kx=ix/53668;
```

```
52:     ix=40014*(ix-kx*53668)-kx*12211;
53:
54:     ky=iy/52774;
55:     iy=40692*(iy-ky*52774)-ky*3791;
56:
57:     rn=(float)(ix-iy)/2147483563.;
58:     rn-=(int)rn;
59:     if( rn<0.) rn++;
60:
61:     return rn;
62: }
```

In Lines 18, an appropriate value is given to `x` as an initial value, i.e., x_{-M+1}, where M is `m`.

In Lines 19 – 29, `i` moves from `-m+1` to `n`.

The first `m` random draws are discarded, and the second `n` ones are utilized for further analysis such as Lines 25 – 28.

`w` in Line 21 corresponds to the acceptance probability $\omega(x_{i-1}, x^*)$, where x_{i-1} and x^* are given by `x` and `x0` in Line 20, respectively

In Line 24, `x` is updated to `x0` with probability `w`.

Remarke: Theoretically, the acceptance probability `w` should be less than or equal to one, i.e., `w=min(w,1.)` have to be included between Lines 22 – 23.

However, as in Line 24, if `w` is greater than one, `x` is always updated to `x0`.

In this case, even if the maximum value of `w` is restricted to one, `x` is always updated to `x0`.

Therefore, the sentence `w=min(w,1.)` is redundant.

2.8.4 Sort Algorithm

In Bayesian analysis, sorting a sequence of n random draws x_1, x_2, $\cdots$, x_n, quantiles are obtained.

Therefore, a sort program is very important.

There are a lot of sort algorithms (for example,
see `https://shinoarchive.com/contents/1916/`).

Insertion Sort: The simplest algorithm, called the insertion sort (挿入ソート), is:

`Very Simple Sort Algorithm`

```
#include<stdio.h>
#include<math.h>
#include<time.h>

```

```
  int ix=1,iy=1;
  float y[100001];

void main()
{
  int i,n;
  int i025,i050,i500,i950,i975;
  float x[100001];
  float urnd(void);
  void  sort(float x[],int n);
  clock_t t0,t1;
  double  dt;

  for(i=1;i<=10000;i++) urnd();

  scanf("%d",&n);

  for(i=1;i<=n;i++){
    x[i]=urnd();
    y[i]=x[i];
```

```
  }

  t0=clock();
  sort(x,n);
  t1=clock();
  dt=(t1-t0)/((double)CLOCKS_PER_SEC);
/*
  for(i=1;i<=n;i++) printf("%10d %10.8f %10.8f\n",i,x[i],y[i]);
*/
  printf("Computational Time = %10.2lf\n",dt);
  i025=(int)( 0.025*(float)n );
  i050=(int)( 0.050*(float)n );
  i500=(int)( 0.500*(float)n );
  i950=(int)( 0.950*(float)n );
  i975=(int)( 0.975*(float)n );
  printf(" 2.5 percent point = %10.8f\n",(y[i025]+y[i025+1])/2.0 );
  printf(" 5   percent point = %10.8f\n",(y[i050]+y[i050+1])/2.0 );
  printf("50   percent point = %10.8f\n",(y[i500]+y[i500+1])/2.0 );
  printf("95   percent point = %10.8f\n",(y[i950]+y[i950+1])/2.0 );
  printf("97.5 percent point = %10.8f\n",(y[i975]+y[i975+1])/2.0 );
```

```
}
/* ================================================= */
float urnd(void)
{
  int    kx,ky;
  float  rn;
/*
  Input:
    ix, iy:  Seeds
  Output:
    rn: Uniform Random Draw U(0,1)
*/
  kx=ix/53668;
  ix=40014*(ix-kx*53668)-kx*12211;

  ky=iy/52774;
  iy=40692*(iy-ky*52774)-ky*3791;

  rn=(float)(ix-iy)/2147483563.;
  rn-=(int)rn;
```

```
  if( rn<0.) rn++;

  return rn;
}
/* ---------------------------------------- */
void sort(float x[],int n)
{
  int i,j,m;
  float xmin=1000000000.0;
  float w[100001];

  for(i=1;i<=n;i++) w[i]=x[i];
  for(i=1;i<=n;i++){
    y[i]=xmin;
    for(j=1;j<=n-i+1;j++){
      if( y[i] > w[j] ){
        y[i]=w[j];
        m=j;
      }
    }
```

```
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 *i*th minimum value.

in Line 85, remove the *i*th 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 $n/\log n$.