IF sentense

```
1: #include<stdio.h>
2: #include<math.h>
3:
4: void main()
5: {
6:
7: /*
8:    ==    !=    >    <    >=    <=
9:    &&    ||
10: */
11:
12:   int i;
13:
14:   scanf("%d",&i);
15:
16:   if( i == 1 ){
17:     printf("i is equal to 1.\n");
18:   }
```

```
19:   else if( (i > 1) && (i <= 5) ){
20:     printf("i is greater than 1 and less than or equal to 5.\n");
21:   }
22:   else{
23:     printf("i is less than 1 or greater than 5.\n");
24:   }
25:
26: }
```

In Line 22, `else{` is erquivalent to `else if( (i < 1) || (i > 5) ){`.

`==` indicates “equal to”, while `!=` indicates “not equal to”.

`&&` indicates “and”, while `||` indicates “or”.

2.8 Sampling Method II: Three Sampling Methods

We want to generate random draws from $f(x)$, called the **target density (目的密度)**, but we consider the case where it is hard to sample from $f(x)$.

Now, suppose that it is easy to generate a random draw from another density $f_*(x)$, called the **sampling density (サンプリング密度)** or **proposal density (提案密度)**.

In this case, random draws of X from $f(x)$ are generated by utilizing the random draws sampled from $f_*(x)$.

Let x be the the random draw of X generated from $f(x)$.

Suppose that $q(x)$ is equal to the ratio of the target density and the sampling density, i.e.,

$$q(x) = \frac{f(x)}{f_*(x)}. \tag{1}$$

Then, the target density is rewritten as:

$$f(x) = q(x) f_*(x).$$

Based on $q(x)$, the acceptance probability is obtained.

Depending on the structure of the acceptance probability, we have three kinds of sampling techniques, i.e., **rejection sampling (棄却法)**, **importance resampling (重点的リサンプリング法)** and the **Metropolis-Hastings algorithm (メトロポリス－ハスティング・アルゴリズム)**.

See Liu (1996) for a comparison of the three sampling methods.

Thus, to generate random draws of x from $f(x)$, the functional form of $q(x)$ should be known and random draws have to be easily generated from $f_*(x)$.

2.8.1 Rejection Sampling (棄却法)

In order for rejection sampling to work well, the following condition has to be satisfied:

$$q(x) = \frac{f(x)}{f_*(x)} < c,$$

where c is a fixed value.

That is, $q(x)$ has an upper limit.

As discussed below, $1/c$ is equivalent to the acceptance probability.

If the acceptance probability is large, rejection sampling computationally takes a lot of time.

Under the condition $q(x) < c$ for all x, we may minimize c.

That is, since we have $q(x) < \sup_x q(x) \leq c$, we may take the supremum of $q(x)$ for c.

Thus, in order for rejection sampling to work efficiently, c should be the supremum of $q(x)$ with respect to x, i.e., $c = \sup_x q(x)$.

Let x^* be the random draw generated from $f_*(x)$, which is a candidate of the random draw generated from $f(x)$.

Define $\omega(x)$ as:

$$\omega(x) = \frac{q(x)}{\sup_z q(z)} = \frac{q(x)}{c},$$

which is called the **acceptance probability (採択確率)**.

Note that we have $0 \leq \omega(x) \leq 1$ when $\sup_z q(z) = c < \infty$.

The supremum $\sup_z q(z) = c$ has to be finite.

This condition is sometimes too restrictive, which is a crucial problem in rejection sampling.

A random draw of X is generated from $f(x)$ in the following way:

(i) Generate x^* from $f_*(x)$ and compute $\omega(x^*)$.

(ii) Set $x = x^*$ with probability $\omega(x^*)$ and go back to (i) otherwise.

In other words, generating u from a uniform distribution between zero and one, take $x = x^*$ if $u \leq \omega(x^*)$ and go back to (i) otherwise.

The above random number generation procedure can be justified as follows.

Let U be the uniform random variable between zero and one, X be the random variable

generated from the target density $f(x)$,

X^* be the random variable generated from the sampling density $f_*(x)$, and x^* be the realization (i.e., the random draw) generated from the sampling density $f_*(x)$.

Consider the probability $P(X \leq x | U \leq \omega(x^*))$, which should be the cumulative distribution of X, $F(x)$, from Step (ii).

The probability $P(X \leq x | U \leq \omega(x^*))$ is rewritten as follows:

$$P(X \leq x | U \leq \omega(x^*)) = \frac{P(X \leq x, U \leq \omega(x^*))}{P(U \leq \omega(x^*))},$$

where the numerator is represented as:

$$\begin{aligned} P(X \leq x, U \leq \omega(x^*)) &= \int_{-\infty}^{x} \int_{0}^{\omega(t)} f_{u,*}(u,t)\, \mathrm{d}u\, \mathrm{d}t = \int_{-\infty}^{x} \int_{0}^{\omega(t)} f_u(u) f_*(t)\, \mathrm{d}u\, \mathrm{d}t \\ &= \int_{-\infty}^{x} \Big(\int_{0}^{\omega(t)} f_u(u)\, \mathrm{d}u \Big) f_*(t)\, \mathrm{d}t = \int_{-\infty}^{x} \Big(\int_{0}^{\omega(t)} \mathrm{d}u \Big) f_*(t)\, \mathrm{d}t \\ &= \int_{-\infty}^{x} \Big[u \Big]_{0}^{\omega(t)} f_*(t)\, \mathrm{d}t = \int_{-\infty}^{x} \omega(t) f_*(t)\, \mathrm{d}t = \int_{-\infty}^{x} \frac{q(t)}{c} f_*(t)\, \mathrm{d}t = \frac{F(x)}{c}, \end{aligned}$$

and the denominator is given by:

$$P(U \leq \omega(x^*)) = P(X \leq \infty, U \leq \omega(x^*)) = \frac{F(\infty)}{c} = \frac{1}{c}.$$

In the numerator, $f_{u,*}(u, x)$ denotes the joint density of random variables U and X^*.

Because the random draws of U and X^* are independently generated in Steps (i) and (ii) we have $f_{u,*}(u, x) = f_u(u)f_*(x)$, where $f_u(u)$ and $f_*(x)$ denote the marginal density of U and that of X^*.

The density function of U is given by $f_u(u) = 1$, because the distribution of U is assumed to be uniform between zero and one.

Thus, the first four equalities are derived.

Furthermore, in the seventh equality of the numerator, since we have:

$$\omega(x) = \frac{q(x)}{c} = \frac{f(x)}{cf_*(x)},$$

$\omega(x)f_*(x) = f(x)/c$ is obtained.

Finally, substituting the numerator and denominator shown above, we have the following equality:

$$P(X \leq x | U \leq \omega(x^*)) = F(x).$$

Thus, the rejection sampling method given by Steps (i) and (ii) is justified.

The rejection sampling method is the most efficient sampling method in the sense of precision of the random draws, because using rejection sampling we can generate mutually independently distributed random draws.

However, for rejection sampling we need to obtain the c which is greater than or equal to the supremum of $q(x)$.

If the supremum is infinite, i.e., if c is infinite, $\omega(x)$ is zero and accordingly the candidate x^* is never accepted in Steps (i) and (ii).

See, for example, Boswell, Gore, Patil and Taillie (1993), O'Hagan (1994) and Geweke (1996) for rejection sampling.

Example of Rejection Sampling: 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:

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

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

Consider the case of $\alpha > 1$ and $\beta > 1$.

$$q'(x) = 0 \qquad \Longrightarrow \qquad x = \frac{\alpha - 1}{\alpha + \beta - 2}$$

Note that there exists the supremum of $q(x)$ when $\alpha > 1$ and $\beta > 1$.

$$\sup_x q(x) = \frac{1}{B(\alpha,\ \beta)}\Big(\frac{\alpha-1}{\alpha+\beta-2}\Big)^{\alpha-1}\Big(\frac{\beta-1}{\alpha+\beta-2}\Big)^{\beta-1}$$

$$\omega(x) = \frac{q(x)}{\sup_x q(x)} = \frac{x^{\alpha-1}(1-x)^{\beta-1}(\alpha+\beta-2)^{\alpha+\beta-2}}{(\alpha-1)^{\alpha-1}(\beta-1)^{\beta-1}}$$

From computational viewpoint,

$$\omega(x) = \exp\Big((\alpha-1)\log(x) + (\beta-1)\log(1-x) + (\alpha+\beta-2)\log(\alpha+\beta-2) \\ - (\alpha-1)\log(\alpha-1) - (\beta-1)\log(\beta-1)\Big)$$

is recommended.

B(a,b) Distribution

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

  int ix=1,iy=1;

void main(){

  float  a,b;
  int    i,j,n;
  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%d",&a,&b,&n);

  for(i=1;i<=n;i++){
```

```
19:     LABEL:
20:     x0=urnd();
21:     w=exp( (a-1.)*log(x0)+(b-1.)*log(1.-x0)
22:      +(a+b-2.)*log(a+b-2.)-(a-1.)*log(a-1.)-(b-1.)*log(b-1.) );
23:     u=urnd();
24:     if( u<w ) x=x0;
25:     else goto LABEL;
26:     x1+=x/((double)n);
27:     x2+=x*x/((double)n);
28:   }
29:
30:   printf("# of Random Draws = %5d\n",n);
31:   printf("Parameters = (%7.1f,%7.1f)\n",a,b);
32:   printf("Mean     = %10.5lf, which should be close to %10.5f\n"
33:          ,x1,a/(a+b));
34:   printf("Variance = %10.5lf, which should be close to %10.5f\n"
35:          ,x2-x1*x1,a*b/((a+b)*(a+b)*(a+b+1.)));
36:
37: }
38: /* ---------------------------------------------- */
```

```
39: double urnd(void)
40: {
41:    int    kx,ky;
42:    double rn;
43: /*
44:    Input:
45:      ix, iy:  Seeds
46:    Output:
47:      rn: Uniform Random Draw U(0,1)
48: */
49:    kx=ix/53668;
50:    ix=40014*(ix-kx*53668)-kx*12211;
51:
52:    ky=iy/52774;
53:    iy=40692*(iy-ky*52774)-ky*3791;
54:
55:    rn=(float)(ix-iy)/2147483563.;
56:    rn-=(int)rn;
57:    if( rn<0.) rn++;
58:
```

```
59:     return rn;
60: }
```

In Lines 19 – 25, one random draw from $f(x)$ is generated as `x`.

In Line 20, a candidate from $f_*(x)$ is generated as `x0`.

In Lines 26 and 27, $\mathrm{E}(X)$ and $\mathrm{E}(X^2)$ are estimated, using the `n` random draws generated from $f(x)$.

In Lines 32 – 35, theoretical values and their estimates of $\mathrm{E}(X)$ and $\mathrm{E}(X^2)$ are compared to make sure whether the source code works well.