Lecture 4: Random Number Generators

Random number generators are commonly used in particle-code simulations and in Monte Carlo simulations (e.g., Snell, 1975).

4.1. Uniform Random Number Generator

Uniform random number generator is a machine-dependent function. The following function is modified from the subroutine RANDU in IBM/SSP (Scientific Subroutine Package). In general, one should be able to find a machine-dependent intrinsic function of "uniform random number generator" from a given FORTRAN complier. However, for convenience, one can always use the following function to generate a uniform random number in a 32-bits machine.

```
C  This function obtains a machine-dependent uniform random number.
C  This function is good for 32 bits computer.
C  This function is modified from IBM/SSP subroutine RANDU
C  Constants used in this functions include
C  2147483648=2**31
C  0.4656613E-9=2.**(-31)
C  65539=65536+3=(2**16)+3
C
      function ran(ix)
C If ix*65539 > 2147483647 then iy = MOD(ix*65539, 2147483648)
      iy=ix*65539
      if(iy)5,6,6
    5  iy=iy+2147483647+1
    6  yfl=iy
      yfl=yfl*.4656613E-9
      ran=yfl
      ix=iy
      return
      end
```

```
//float ran(int &);
//C  This function obtains a machine-dependent uniform random number.
//C  This function is good for 32 bits computer.
//C  This function is modified from IBM/SSP subroutine RANDU
//C  Constants used in this functions include
//C  2147483648=2**31
//C  0.4656613E-9=2.**(-31)
//C  65539=65536+3=(2**16)+3
//C
//      function ran(ix)
float ran(int &ix){
    int iy;
    float yfl;
    //C If ix*65539 > 2147483647 then iy = MOD(ix*65539, 2147483648)
    iy=ix*65539;               //      iy=ix*65539
    if(iy<0){                  //      if(iy)5,6,6
        iy=iy+2147483647+1;    //    5  iy=iy+2147483647+1
```

```
    }
    yfl=iy;                    //    6  yfl=iy
    yfl=yfl*.4656613E-9;       //      yfl=yfl*.4656613E-9
    ix=iy;                     //      ix=iy
    return yfl;                //      ran=yfl
                               //      return
}                              //      end
```

Exercise 4.1.

Write a program to generate more than 1000 uniform random numbers. Plot distribution of these random numbers and compare it with the profile of uniform distribution function.

4.2. Random Number Generator of a General Non-uniform Probability Function

From the uniform random number generator, one can obtain random number of any given probability function. The following subroutine is an example of random number generator of any given probability function FUNC, in which random number between XL and XR are obtained. FMAX is the maximum of the probability function FUNC(x). A main program need provide an external probability function in order to use the following subroutine to generate a random number.

```
C  This subroutine obtains a random number of a given function FUNC(x).
C  The resulting random number is in the range of (XL, XR)
C  Fmax>0 is the maximum of the function FUNC(x) for XL<x<XR
C  ix is the seed of uniform random number.
C  program stop if fmax .le.0, or XR.LE.XL, or FUNC(x)<0 for XL<x<XR,
C
      subroutine ranfunc(x,FUNC,XL,XR,Fmax,ix)
      external FUNC
C--------------
      if(Fmax.le.0.) then
      print *,'program stop because Fmax<0, Fmax=', Fmax
      stop
      endif
      if(XR.le.XL) then
      print *,'program stop because XR.LE.XL, XR,XL=', XR,XL
      stop
      endif
C--------------
    1  continue
      x=ran(ix)*(XR-XL)+XL
      y=ran(ix)*Fmax
      y0= FUNC(x)
C--------------
      if(y0.lt.0.) then
      print *,'program stop because FUNC(x)<0, FUNC(x)=', y0
      stop
      endif
```

```
C--------------
      if(y.gt.y0) go to 1
      return
      end
```

```
//void ranfunc(float &,float (*FUNC), float , float , float , int &);
//C  This subroutine obtains a random number of a given function FUNC(x).
//C  The resulting random number is in the range of (XL, XR)
//C  Fmax>0 is the maximum of the function FUNC(x) for XL<x<XR
//C  ix is the seed of uniform random number.
//C  program stop if fmax .le.0, or XR.LE.XL, or FUNC(x)<0 for XL<x<XR,
//C
//      subroutine ranfunc(x,FUNC,XL,XR,Fmax,ix)
//      external FUNC
void ranfunc(float &x,float (*FUNC)(float), float XL, float XR, float Fmax, int &ix){
    float y, y0;
                                  //C--------------
    if(Fmax<=0.){                 //      if(Fmax.le.0.) then
        printf("program stop because Fmax<0, Fmax=, %f \n", Fmax);
                                  //      print *,'program stop because Fmax<0, Fmax=', Fmax
        exit (EXIT_FAILURE);      //      stop
    }                             //      endif
    if(XR<=XL){                   //      if(XR.le.XL) then
        printf("program stop because XR<=XL, XR,XL=, %f %f \n", XR,XL);
                                  //  print *,'program stop because XR.LE.XL, XR,XL=', XR,XL
        exit (EXIT_FAILURE);      //      stop
    }                             //      endif
                                  //C--------------
continue1:                        //    1  continue
    x=ran(ix)*(XR-XL)+XL;         //      x=ran(ix)*(XR-XL)+XL
    y=ran(ix)*Fmax;               //      y=ran(ix)*Fmax
    y0= FUNC(x);                  //      y0= FUNC(x)
                                  //C--------------
    if(y0<0.){                    //      if(y0.lt.0.) then
        printf("program stop because FUNC(x)<0, FUNC(x)=, %f \n", y0);
                                  //  print *,'program stop because FUNC(x)<0, FUNC(x)=', y0
        exit (EXIT_FAILURE);      //      stop
    }                             //      endif
                                  //C--------------
    if(y>y0) {goto continue1;}    //      if(y.gt.y0) go to 1
                                  //      return
}                                 //      end
```

Exercise 4.2.

Write a program to generate more than 1000 random numbers of a given function. Plot distribution of these random numbers and compare it with the profile of the given distribution function.

4.3. Random Number Generator of a Normal Distribution Function

According to *Law of Large Numbers*, and *Central Limit Theorem* (e.g., Snell, 1975), random number of normal distribution function can be obtained from uniform random number generator. According to subroutine GAUSS in IBM/SSP (Scientific Subroutine Package), random number of normal distribution function, with mean equal to zero and standard deviation equal to one, can be obtained from

$$Y = \frac{(\sum_{i=1}^{K} x_i) - \frac{K}{2}}{\sqrt{K/12}} \tag{4.1}$$

where all x_i are obtained from a uniform random number generator. Y is a random number of normal distribution with mean equal to zero and standard deviation equal to one.

Exercise 4.3.

Verify Eq. (4.1) based on *Law of Large Numbers*, and *Central Limit Theorem*.

Hint: According to Law of Large Numbers, if the mean of x_i is μ, and variance of x_i is σ^2 then the mean and variance of $S = x_1 + x_2 + ... + x_K$ will be $K\mu$ and $K\sigma^2$, respectively. For uniform random number generator, the corresponding probability function is $f(x) = x$ where $0 \le x \le 1$. It can be easily shown that the mean of this uniform probability function is 1/2 and the variance of this uniform probability function is 1/12. Thus, if $S = x_1 + x_2 + ... + x_K$ and x_i are obtained from uniform random number generator, the mean of S should be $K/2$ and variance of S should be $K/12$.

Exercise 4.4.

Based on Eq. (4.1), write a program to generate more than 1000 random numbers of a normal distribution function with $K = 12$ and $K = 6$. Plot distributions of these two sets of random numbers and compare them with the profile of normal distribution function. Compare CPU time used in generating random numbers in Exercise 4.2 and in this Exercise.

References

Snell, J. L., *Introduction to Probability Theory With Computing,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

System/360 Scientific Subroutine Package Version III, Programmer's Manual, 5th edition, IBM, New York, 1970.