Generating Random Numbers

The ability to generate high quality random numbers is crucial to molecular simulation, either for the generation of random velocities to start Molecular Dynamics simulations, or for the generation of the random number that control the trajectory of a Monte Carlo simulation. The generation of high quality random numbers has been the subject of much study (for an early review see jansson66[1]), though now the field almost always uses pseudorandom number generators, which produce a predictable stream of seemingly random numbers from an initial seed, via a mathematical function. In my opinion the best pseudo-random number generator for our work is currently the Mersenne Twister[2], which has a very large period ($2^{19937-1}$), and is very fast and consumes a small amount of memory. Implementations of the Mersenne Twister are available in a range of languages (including C, C++, Java and Fortran) under BSD or GPL licenses.

The majority of random number generators produce numbers that are uniformly distributed. Some applications require the generation of random numbers from different distributions, so methods have been developed that can convert uniformly generated random numbers generated between 0 and 1 (ξ) into random numbers (x) from a specified distribution ($F(x)$). These methods will now be enumerated;

(1) The Direct Approach. The most logical way to obtain a random number x from the distribution $F(x)$ is via the inversion of the the equation $F(x) = \xi$ to obtain;

$$\text{(a)} \quad x = F^{-1}(\xi)$$

If the inverse of F is known then this formula may be directly applied. Otherwise a numerical approximation of the inverse may be calculated and stored in a look-up table (though this would be a memory hungry and there would be discretisation errors).

(2) The Rejection Method. The rejection method of von Neumann provides a simple, though potentially inefficient method to generate random numbers from an arbitrary distribution ($F(x)$) in the range $a \le x \le b$. If c is the maximum value of $F(x)$ within this range (i.e. $F(x) \le c$ for $a \le x \le b$), then the method consists of the following steps;

- Generate a pair of random numbers, ξ_1 and ξ_2.
- Determine $x_1 = (a + \xi_1)(b - a)$
- Determine $x_2 = c \times \xi_2$
- If $x_2 - f(x_1) > 0$ the reject the pair of numbers and try again. Otherwise accept x_2 as the random number.

This method will only be efficient if the average value of $F(y)/c$ over the range $a \le x \le b$ is close to 1, otherwise the majority of trials will be rejected.

---

[1] Jansson, B, *Random Number Generators*, Victor Pettersons Bokindustri Aktiebolag, Stockholm, 1966
[2] http://www.math.keio.ac.jp/~matumoto/emt.html

Random Numbers from Specific Distributions

The Exponential Distribution

The distribution we wish to sample from is $f(x) = \exp(-x)$

The easiest method is the direct method, e.g. $x = -\ln(\xi)$.

The Normal Distribution

The normal (or gaussian) distribution has a special place in probability theory, and is given by;

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{ -\frac{(x-\mu)^2}{(2\sigma)^2} \right\}$$

where μ is the mean of the distribution, and σ is the standard deviation.

Random numbers can be generated on this distribution using the Box-Muller transformation.[3] This transforms two uniformly generated random numbers between -1 and 1 (ξ_1 and ξ_2) into two normally distributed random numbers (x_1 and x_2), via;

- Generate two uniform random numbers, ξ_1 and ξ_2, and evaluate $z = \xi_1^2 + \xi_2^2$.
- If $z \geq 1$ then reject these two numbers and return to the first step.
- Evaluate $w = \sqrt{\frac{-2\ln z}{z}}$
- $x_1 = w\xi_1$ and $x_2 = w\xi_2$ are now two independent random numbers on a normal distribution of mean 0.0 and standard deviation 1.0.
- These can be converted to random numbers with mean μ and standard deviation σ via $x' = x\sigma + \mu$.

The Bivariate Normal Distribution

The bivariate normal distribution[4] gives the joint probability of observing both x_1 and x_2 together, and is given by;

$$f(x_1, x_2) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left\{ -\frac{1}{2(1-\rho^2)} \left[\left(\frac{x_1-\mu_1}{\sigma_1} \right)^2 - 2\rho \frac{x_1-\mu_1}{\sigma_1} \frac{x_2-\mu_2}{\sigma_2} + \left(\frac{x_2-\mu_2}{\sigma_2} \right)^2 \right] \right\}$$

where x_1 and x_2 will have means μ_1 and μ_2, standard deviations σ_1 and σ_2, and correlation σ.

---

[3] http://en.wikipedia.org/wiki/Box-Muller_transform
[4] http://www.statisticalengineering.com/bivariate_normal.htm

This distribution can be written using a vector notation,[5] using $X = (x_1, x_2)$, $B = (\mu_1, \mu_2)$ and $A = \begin{pmatrix} \sigma_1 & \rho\sigma_1\sigma_2 \\ \rho\sigma_2\sigma_1 & \sigma_2 \end{pmatrix}$ (note that this is a positive definite symmetric matrix[6]);

$$f(X) = \frac{1}{2\pi\sqrt{\det(A)}} \left\{ \exp\left[-\frac{(X-B)^T A^{-1} (X-B)}{2} \right] \right\}$$

To generate random numbers on this distribution you could;[7]

- First generate two, uncorrelated, random numbers from a normal distribution of mean 0.0, standard deviation 1.0 (e.g. via the Box-Muller method). These two random numbers will be called x_1 and x_2.
- Compute the correlated random numbers (X_1 and X_2) from the bivariate normal distribution via the equations;

$$X_1 = \mu_1 + \sigma_1 x_1$$
$$X_2 = \mu_2 + \sigma_2 \left[x_1\rho + x_2\sqrt{1-\rho^2} \right]$$

  Note that both X_1 and X_2 will be from the standard normal distribution if ρ is equal to 0.

The Multivariate Normal Distribution

The multivariate normal, or multinormal distribution gives the joint probability of picking x_1 to x_n together, and is given by;

$$f(X) = \frac{1}{\sqrt{(2\pi)^n \det(A)}} \exp\left\{ -\frac{(X-B)^T A^{-1} (X-B)}{2} \right\}$$

i.e. this is exactly the same equation as the vector form of the bivariate normal distribution, with the exception that X is a rank n vector rather than a rank 2 vector, and A is an n by n symmetric positive definite matrix (which is the covariance matrix).

To generate random numbers in this distribution you could;[8]

- First generate a vector of n independent random numbers from the normal distribution of mean 0 and standard deviation 1. This vector is called Z.
- This vector can be transformed into the multivariate normal distribution via;

$$X = B + VZ \quad \text{where } VV^T = A^{-1}$$

  Note that all of the elements of X will be from the standard normal distribution if all of the off-diagonal terms of A are equal to zero (e.g. there is no correlation between the values).

---

[5] http://rkb.home.cern.ch/rkb/AN16pp/node20.html
[6] The square n*n matrix A is positive definite if $x^T Ax > 0$ for all non-zero x. This requires that all of the eigenvalues of A are positive.
[7] http://www.statisticalengineering.com/bivariate_normal.htm
[8] http://www.ds.unifi.it/VL/VL_EN/special/special8.html

Perl script to generate and histogram normally distributed random numbers.

```
#!/usr/bin/perl

sub rangau
{
      my $ran1 = (2.0*rand()) - 1.0;
      my $ran2 = (2.0*rand()) - 1.0;

      my $z = $ran1**2 + $ran2**2;

      while ($z >= 1.0)
      {
            $ran1 = (2.0*rand()) - 1.0;
            $ran2 = (2.0*rand()) - 1.0;
            $z = $ran1**2 + $ran2**2;
      }

      my $w = sqrt( (-2.0*log($z)) / $z);

      return ($w*$ran1,$w*$ran2);
}

my @hist;
my $minx = -10.0;
my $maxx = 10.0;
my $nbins = 500;
my $binwdth = ($maxx-$minx)/$nbins;

sub hist
{
      my ($val) = @_;

      my $bin = int(($val - $minx) / $binwdth);

      if (!defined($hist[$bin])){ $hist[$bin] = 1;}
      else { $hist[$bin] ++;}

}

sub scalenormal
{
      my ($val,$sig,$mean) = @_;

      return ($sig*$val) + $mean;
}

sub printhist
{
      for (my $i=0; $i<=$nbins; $i++)
      {
            if (defined($hist[$i]))
            {
                  my $x = $i*$binwdth + $minx;
                  print "$x  $hist[$i]\n";
            }
      }
}

$mean = 5.0; $stddev = 3.0;
for ($i=0; $i<50000; $i++)
{
      ($ran1,$ran2) = rangau();
      $ran1 = scalenormal($ran1,$stddev,$mean);
      $ran2 = scalenormal($ran2,$stddev,$mean);
      hist($ran1); hist($ran2);
}

printhist();
```