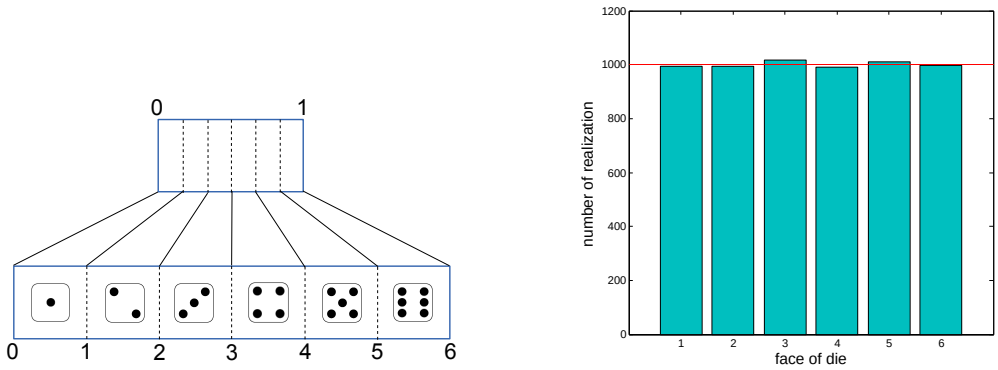


(a) Mapping from random numbers uniformly distributed between 0 and 1 to discrete random numbers.

(b) Statistics of the virtual die. Due to the finite number of realizations, the probability is close but not exactly uniform.

Figure 15.1: Virtual Die

generate the pseudo random numbers. A most popular method, known as linear congruential generator, generates a sequence of random numbers I_i by a recursive equation

$$I_{i+1} = a\, I_i + b \mod c \tag{15.4}$$

where a, b, and c are integer constants. Then, the random number $r_i = I_i/c$ is a pseudo random number in the range of $0 < r_i < 1$. The most of the random generators supplied by computer systems are based on this algorithm. In MATLAB, `rand()` generates random numbers between 0 and 1.

The quality of the random number depends on the choice of a, b, c. Note that total number of possible random numbers are limited to c at the best. This type of pseudo random numbers are periodic and after a certain iterations the same sequence comes back. The longest possible period is c. It has been shown that a set of parameters $a = 7^5 = 16807$, $b = 0$, and $c = 2^{31} - 1 = 2147483647$ provides excellent uniform random numbers with the maximum periodicity $2^{31} - 1$.

The recursive equation (15.4) must start with a certain initial number I_0 called "seed". If you do not specify a seed, a fixed default seed is used and the same sequence of random numbers is obtained every time. In order to avoid the use of the same random numbers, we must pick a different seed every time. For example, you can use the current date and time to reset the seed. In MATLAB, `rng('shuffle')` does it.

EXAMPLE 15.1 A Virtual Die

We make a virtual die based on the congruential generator. The uniform random numbers between 0 and 1 is mapped to integer random numbers between 1 and 6. First, we generate a basic uniform random number r and multiply 6 to it. If $N - 1 < 6r \le N\,(N = 1, \cdots, 6)$, then the die gives the value N. For example, if the random number is $r = 0.19$, then, $6r = 1.14$. This corresponds to 2 on the die. Figure 15.1a illustrates the mapping from the uniform random number to the die. Program 15.1, we roll the virtual die 6000 times. If the die is far, each face is realized 1000 times. The result is shown in Fig. 15.1b. It appears to be fair except for the small fluctuation. This fluctuation is not due to the low quality of the random number generator. Even if an ideal random numbers are used, there is still fluctuation. The fluctuation disappears only when the number of samples is infinitely large.

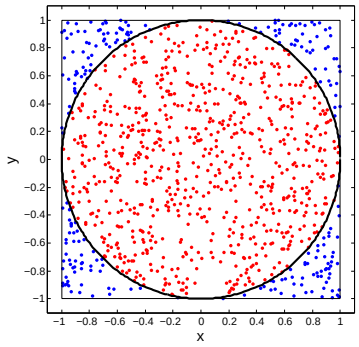


Figure 15.2: Monte Carlo method to evaluate the area inside a circle. The area inside the circle equals $4N_{\rm red}/(N_{\rm blue}+N_{\rm red})$ where the factor 4 is the area of the square.

EXAMPLE 15.2 Monte Carlo integration

Multidimensional integrals can be computationally demanding. There is a way to estimate them by random sampling. The result may not be very accurate but gives a rough estimate and particularly efficient for high dimensional integral. Here we evaluate the volume of a n-dimensional hypersphere of radius 1. The sphere is inscribed in a hypercube of edge length 2. The volume of the hypercube is 2^n. Now, we generates many random points uniformly distributed in the hypercube. Then, we count the number of points inside the hypersphere. The ratio of the population inside the sphere and the total population of the points is roughly speaking in proportion to the volume of the sphere to the volume of the cube. Since we know the volume of the cube, we can estimate the volume of the sphere from the ratio. As the number of the points increases to infinity, the result approaches to the exact volume $V_n = \pi^{n/2}/\Gamma(n/2+1)$ where $\Gamma(\cdot)$ is the gamma function.[1] Figure 15.2 illustrates the case for $n=2$. Generate a pair of standard uniform random numbers r_1 and r_2. Convert them to x and y coordinates by

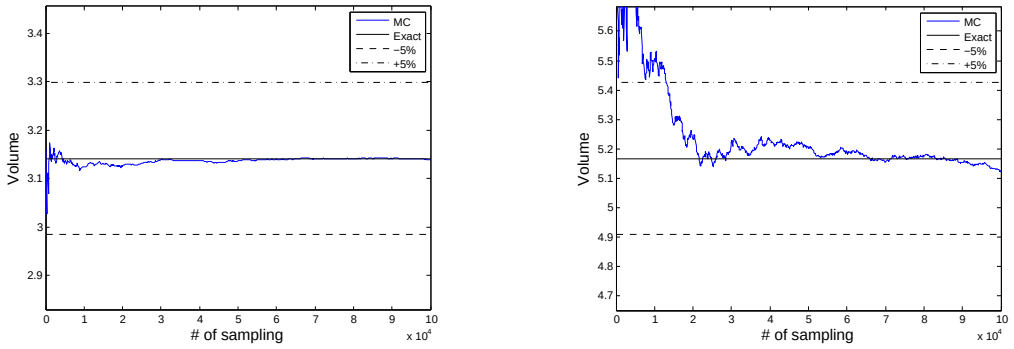
$$x = 2r_1 - 1, \qquad y = 2r_2 - 1 \tag{15.5}$$

which are uniform random numbers between -1 and 1. If $x^2+y^2<1$, then increment $N_{\rm IN}$ by one. After N random points, the volume of the sphere is estimated by $V_{\rm sphere} = \frac{N_{\rm IN}}{N} V_{\rm cube}$.

Program 15.2 evaluates the volume of the N-dimensional hypersphere using the Monte Carlo integration. Figure 15.3 shows that with 100000 sampling, we can evaluate the volume of two-dimensional circle rather accurately and the volume of the six-dimensional sphere within $\pm 5\%$ of error. If we use a standard integral method, the number of grid points increases as the power of N. On the other hand, the number of sampling points necessary to get a reasonable estimate by the Monte Carlo method increases slower than that. Therefore, the Monte Carlo method is advantageous for high dimensional integrals.

15.3 Non-uniform distributions

In the previous examples, we generated desired random numbers by stretching space or using two random numbers to cover two-dimensional space. In either cases, the resulting new random numbers are distributed uniformly. However, non-uniform distributions are common in physics. For example, the velocity of the gas particles in thermal equilibrium is Gaussian distributed (Maxwell's distribution). The chance you find



(a) 2D (b) 6D

Figure 15.3: Monte Carlo evaluation of the volume of hypersphere. As the number of sampling points increases, the result approaches to the exact value.

a slow particle is higher than a fast particle. Since computer systems usually supply only uniform random numbers, we need to transform it to a desired distribution.

Let $\hat{X}$ a stochastic variable and its realization is a random number x with a distribution $\chi(x)$. We introduce a new stochastic variable $\hat{Y} = f(\hat{X})$. The realization of $\hat{Y}$ is related to the realization of $\hat{X}$ through the same function $y = f(x)$. Then, the mathematical theorem tells that

$$\rho(y)|\mathrm{d}y| = \chi(x)|\mathrm{d}x| \tag{15.6}$$

and thus we have the distribution of y as

$$\rho(y) = \chi(x)\left|\frac{\mathrm{d}x}{\mathrm{d}y}\right| = \chi(x)\left|\frac{\mathrm{d}f^{-1}(y)}{\mathrm{d}y}\right|. \tag{15.7}$$

If x is uniformly distributed as

$$\chi(x) = \begin{cases} 1 & 0 < x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{15.8}$$

then we have

$$\rho(y) = \left|\frac{\mathrm{d}x}{\mathrm{d}y}\right| = \left|\frac{\mathrm{d}f^{-1}(y)}{\mathrm{d}y}\right| \tag{15.9}$$

By choosing an appropriate function $y = f(x)$, we can construct a random number generate with a desired distribution $\rho(y)$. For example, when $y = -\ln(x)$, $\rho(y) = \mathrm{e}^{-y}$. Figure 15.4 shows the mapping from uniform x between 0 and 1 to y exponentially distributed from 0 to ∞.

For multi-dimensional cases like Example 15.2, we consider a transformation $y_i = f_i(x_1, x_2, \cdots)$. The distribution for $\{y_i\}$ is given by

$$\rho(y_1, y_2, \cdots)\mathrm{d}y_1\mathrm{d}y_2\cdots = \chi(x_1, x_2, \cdots)\left|\frac{\partial(x_1, x_2, \cdots)}{\partial(y_1, y_2, \cdots)}\right| \tag{15.10}$$

where $|\partial()/\partial()|$ is the Jacobian determinant. For two-dimensional case,

$$\rho(y_1, y_2) = \chi(x_1, x_2)\begin{vmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} \\ \frac{\partial x_2}{\partial y_1} & \frac{\partial x_2}{\partial y_2} \end{vmatrix} \tag{15.11}$$

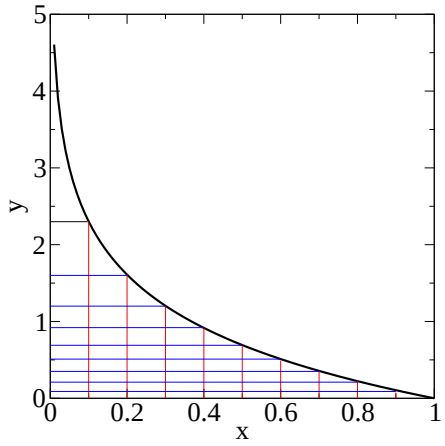


Figure 15.4: Mapping from random number x uniformly distributed between 0 and 1 to y exponentially distributed from 0 to ∞. The transformation function is $y = -\ln(x)$.

15.4 Gaussian random number

Stochastic variables with a Gaussian distribution is defined by

$$\rho(x) = \frac{\sigma}{\sqrt{2\pi}} \mathrm{e}^{-x^2/2\sigma^2} \tag{15.12}$$

and it has mean $\langle x \rangle = 0$ and variance $\langle x^2 \rangle - \langle x \rangle^2 = \sigma^2$. When $\sigma^2 = 1$, it is called normal distribution. The Gaussian distributed stochastic variables are ubiquitous particularly in statistical physics. For example, many fluid particles collide with a Brownian particle during a short period of time. The force exerted on the Brownian particle by the fluid particle is stochastic and Gaussian distributed.

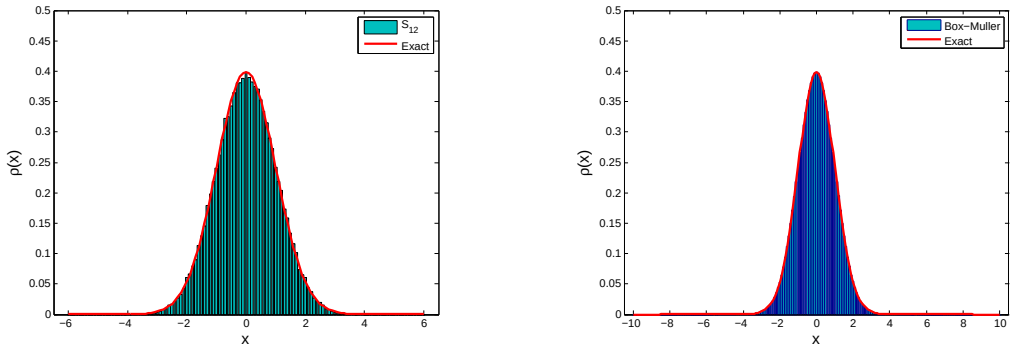
There is a mathematical reason why the normal distribution is ubiquitous. Consider N independent stochastic variables X_i, $i = 1, \cdots, N$ with an identical distribution. The mean and variance are given by $\langle X_i \rangle = \mu$ and $\langle (X_i - \mu)^2 \rangle = \sigma^2$. Then, the distribution of a stochastic variable defined by

$$S_N = \frac{X_1 + X_2 + \cdots + X_N - N\mu}{\sqrt{N\sigma^2}} = \frac{1}{\sqrt{N\sigma^2}} \sum_{i=1}^{N} (X_i - \mu) \tag{15.13}$$

approaches a normal distribution as $N \to \infty$. This is known as central limit theorem.[2] Note that the distribution of X_i does not have to be Gaussian. As long as the mean and variance are well defined, the sum of such non-Gaussian stochastic variables is Gaussian distributed.

Now, we want to generate random numbers drawn from the normal distribution. The normally distributed random generator returns values close to 0 more often than larger values. One way is to utilize the central limit theorem. Consider 12 random numbers X_i, $i = 1, \cdots, 12$ uniformly distributed between 0 and 1. As discussed in the previous section, X_i has mean $\mu = 1/2$ and variance $\sigma^2 = 1/12$. While N is not close to ∞, the sum of the 12 random numbers,

$$S_{12} = \frac{X_1 + X_2 + \cdots + X_{12} - 12\mu}{\sqrt{12\sigma^2}} = \sum_{i=1}^{12} X_i - 6 \tag{15.14}$$



(a) The sum of 12 uniform random numbers (b) Box-Muller mnethod

Figure 15.5: Two generators of normally distributed random generator. The distribution is constructed from 100,000 realizations. The distribution of S_{12} is strictly zero for $|x| > 6$ and thus rare events are not included. In principle, the Box-Muller method can generate rare random numbers. However, it is so rare that $|x| > 6$ is not realized with this 100,000 sampling.

is, roughly speaking, normally distributed. A main problem of this method is that the largest value it can generate is only 6. The true normal distribution allows very large number up to infinity although the probability to obtain such a large value is very small. The probability density at $x = 6$ is about 10^{-8}. Only one out of 100 million realizations hits that high value. Hence, it is not a significant deficiency for most applications. If such rare events are not important, the sum of 12 uniform random numbers is a simple way to generate the normal distribution. Figure 15.5a shows the distribution of S_{12} which matches well to the true normal distribution.

If rare events need to be taken into account, use a mathematically rigorous transformation known as the Box-Muller method. It turns out that the transformation of two uniformly distributed random numbers x_1 and x_2 to the two Gaussian distributed random numbers y_1 and y_2 is more convenient. Consider the transformations

$$y_1 = \sqrt{-2\ln x_1}\cos(2\pi x_2) \tag{15.15a}$$
$$y_2 = \sqrt{-2\ln x_1}\sin(2\pi x_2) \tag{15.15b}$$

and their inverse

$$x_1 = \exp^{-(y_1^2+y_2^2)/2} \tag{15.16a}$$
$$x_2 = \frac{1}{2\pi}\arctan\frac{y_2}{y_1} \tag{15.16b}$$

The corresponding distribution function for y_1 and y_2 is

$$\rho(y_1,y_2) = \begin{vmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} \\ \frac{\partial x_2}{\partial y_1} & \frac{\partial x_2}{\partial y_2} \end{vmatrix} = -\left(\frac{1}{\sqrt{2\pi}}e^{-y_1^2/2}\right)\left(\frac{1}{\sqrt{2\pi}}e^{-y_2^2/2}\right) \tag{15.17}$$

This distribution indicates that y_1 and y_2 are independent and normally distributed random numbers. Using two uniformly distributed random numbers x_1 and x_2, we generate two normally distributed random numbers. Unlike, the sum of 12 random numbers, y_1 and y_2 are truly Gaussian distributed from $-\infty$ to ∞.

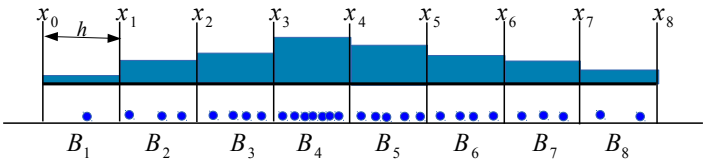


Figure 15.6: Generating histogram from continuous random numbers. Circles indicate the random numbers. The number of the circles in a bin corresponds to the height of the bar above it.

EXAMPLE 15.3

Generate random numbers that are normally distributed. Program 15.3 implements the Box-Mullar method. We check the distribution of random numbers by plotting a histogram. To construct a histogram, first we create bins by dividing the entire range into many small intervals. For eaxmple, one bin corresponds to a segment between x and $x+h$ where h is the width of the bin. See Fig. 15.6. Now, we generate a random number and check which bin the random number belongs to. After generating sufficient number of random numbers, count the number of random numbers in each bin. That is the height of the bar in the histogram. Figure 15.5b shows the distribution of random numbers generated by the Box-Muller method. The width of the bins is very small in this plot and the plot looks like continuous line, However. it is a histogram.

15.4.1 Exponential distributions

An exponential distribution with a rate parameter $\lambda > 0$ defined by

$$\rho(x) = \begin{cases} \lambda \mathrm{e}^{-\lambda x} & x > 0 \\ 0 & x < 0 \end{cases} \tag{15.18}$$

and its mean and variance are $\mu = 1/\lambda$ and $\sigma^2 = 1/\lambda^2$.

The exponential distribution is also common in physics. For example, the energy of a system in a thermal equilibrium is a stochastic variable and distributed exponentially as

$$\rho(E) = \frac{1}{Z}\mathrm{e}^{-E/k_\mathrm{B}T} \tag{15.19}$$

where Z is a normalization constant. This distribution is known as the Boltzmann distribution.

The exponentially distributed random numbers can be obtained by a transformation $y = -\ln(x)/\lambda$ where x is a uniformly distributed between 0 and 1.

15.4.2 Evaluation of Mean

To analyze stochastic systems, we often evaluate a mean of a physical quantity $f(\hat{X})$ which is a function of stochastic variable $\hat{X}$. The analytic expression is defined by

$$\langle f \rangle = \sum_i f(x_i) P_i \tag{15.20}$$

for a discrete system and

$$\langle f \rangle = \int f(x)\rho(x)\mathrm{d}x \tag{15.21}$$

for a continuous system. We evaluate this summation or integral using the Monte Carlo method. The procedure is very simple. Generate N random numbers r_i, $i = 1, \cdots, N$ out of a desired distribution P_i or $\rho(x)$. The mean is simply

$$\langle f \rangle = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} f(r_i) \tag{15.22}$$

Of course, in practice, we use a finite number of sampling N. We need to make it sure that N is large enough to get an accurate value.

EXAMPLE 15.4

Consider a stochastic variable $\hat{X}$ of normal distribution. We evaluate the 2nd and 4th moments, $\langle x^2 \rangle$ and $\langle x^4 \rangle$, using the Monte Carlo method. It is known that $\langle x^4 \rangle = 3\langle x^2 \rangle^2$. We will check if the Monte Carlo simulation can get the same answer. Program 15.4 evaluates the moments using the Box-Muller method. With $N = 1000$, the agreement is not bad but much better agreement is obtained with $N = 1000000$.

```
N=   1000000,  V4/(V2)^2 = 3.0009e+00
N=      1000,  V4/(V2)^2 = 2.9033e+00
```

15.5 Applications in Physics

15.5.1 Thermal Speed

Particles in a three-dimensional gas at temperature T has random velocity v and its probability distribution of the velocity is given by the Maxwell's distribution

$$\rho(\mathbf{v}) = \sqrt{\frac{m}{2\pi k_{\mathrm{B}} T}} \mathrm{e}^{-mv^2/2k_{\mathrm{B}}T} \tag{15.23}$$

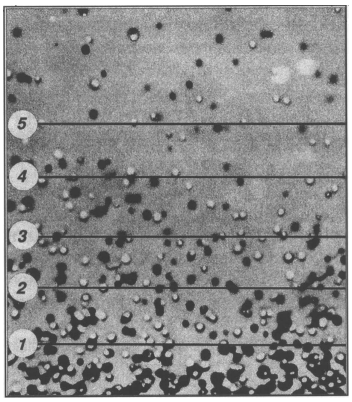
where m is the mass of the particle and T and k_{B} are temperature of the gas and the Boltzmann constant, respectively. The mean velocity is clearly $\langle v \rangle = 0$ since v and $-v$ have the equal probability. On the other hand, the mean *speed* does not vanish. The exact answer can obtained as

$$\langle |v| \rangle = \iiint |v| \rho(\mathbf{v})\, \mathrm{d}v_x\, \mathrm{d}v_y\, \mathrm{d}v_z = \sqrt{\frac{m}{2\pi k_{\mathrm{B}} T}} \int_0^\infty \int_0^\pi \int_0^{2\pi} v^3 \mathrm{e}^{-mv^2/2k_{\mathrm{B}}T}\, \mathrm{d}\phi \sin\theta\, \mathrm{d}\theta\, \mathrm{d}v = \sqrt{\frac{8k_{\mathrm{B}}T}{\pi m}} \tag{15.24}$$

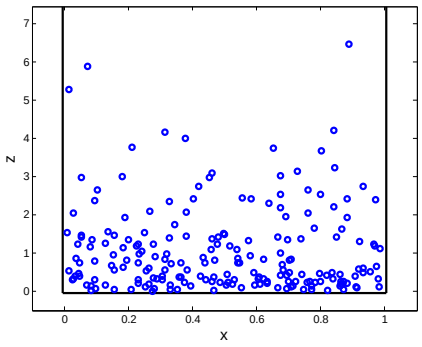
Now, we try to evaluate the mean *speed* of hydrogen molecules in the air using the random numbers. Since the Maxwell's distribution is Gaussian, we can use the normally distributed random numbers with mean value $\mu = 0$ and variance $\sigma = \sqrt{\frac{k_{\mathrm{B}}T}{m}}$. Here, as an exercise, we try to confirm Eq. (15.24) by the direct Monte Carlo simulation.

The statistical analysis tells that the relative error of the finite sampling is about $1/\sqrt{N}$. Therefore, with $N = 100000$ sampling, we expect $\frac{\Delta v}{\langle |v| \rangle} \sim 10^{-4}$. Program 15.5 generates normally distributed random numbers $\mathbf{v}_i$ and calculates the mean speed $\langle |v| \rangle = \frac{1}{N} \sum_i^N |v_i|$. The result shows the error less than 10^{-4} as expected.

```
mean speed = 1.77558e+03 (exact=1.77566e+03)
relative error = 4.6397e-05
```



(a) Experiment



(b) Exponential Distribution

Figure 15.7: Snapshot of the sedimentation diffusion equilibrium

15.5.2 Sedimentation-diffusion equilibrium

A colloidal particle of mass m under the uniform gravity g does not fall down completely to the bottom of the container. Random collisions with the fluid particles push the colloidal particle upward against the gravity. The height of the particle from the bottom is stochastic and distributed as

$$\rho(z) = \frac{mg}{k_{\mathrm{B}}T} \mathrm{e}^{-mgz/k_{\mathrm{B}}T} \tag{15.25}$$

which is known as barometric formula.[3] The barometric formula is one of the example of the Boltzmann distribution. Two hundred colloidal particles are placed in a fluid. Generate an image showing the location of the colloidal particles.

Measure the height using the gravitational length $\ell_g = k_{\mathrm{B}}T/mg$ as unit, the distribution is simply

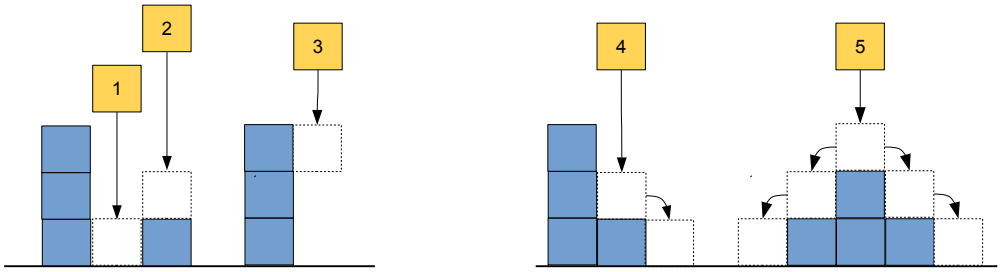
$$\rho(z) = \mathrm{e}^{-z} \tag{15.26}$$

The horizontal positions of the particles are uniformly distributed whereas the vertical positions are exponential. Program 15.6 generates a snapshot image of the sedimentation-diffusion equilibrium. The result is shown in Fig. 15.7. The distribution of particles obtained by the Monte Carlo simulation resembles to the experimental observation.

15.5.3 Surface Growth: Random Deposition Models

The current technology demands high quality of materials. Crystals grows by themselves but we want to control the growth of the materials. The vapor deposition method and molecular beam epitaxy (MBE) allow us to deposit atoms on top of the substrate in a controlled manner. We want to simulate such a surface growth process in computer. A simplest model is the random deposition model.[4] (See Fig. 15.8a.) The lateral position of the particles are randomly chosen. Particles either falls down to the substrate (particle 1 in Fig 15.8a) or stick to on top of the other particle (particle 2). It is interesting that even we choose the lateral position of the particles by uniform random numbers, the resulting surface is not smooth at all. This model generates a densely packed crystal but with large surface roughness.

In order to quantify the surface roughness, we first define the height of the surface as a stochastic variable. We consider a one-dimensional substrate of the size L and deposit N particles on it. As particles are



(a) Ballistic deposition model with or without overhang.

(b) Ballistic deposition model with surface relaxation.

Figure 15.8: A random deposition model with surface relaxation. The lateral position is randomly selected and a particle is placed on the surface particle from the above. Then, it steps down to the local minimum.

deposited, the surface grows. However, the growth is not uniform. The height h_i at the lateral position x_n, $n = 1, \cdots, L$ is random number drawn out of a certain probability distribution $P(h; N)$, which we want to find by computer simulation. We calculate the mean height and the second moment by

$$\langle h \rangle = \frac{1}{L} \sum_{i=1}^{L} h_i, \qquad \langle h^2 \rangle = \frac{1}{L} \sum_{i=1}^{L} h_i^2, \tag{15.27}$$

Rigorously speaking L should be infinitely large but in computer we use a large finite number. In order to get a good statistics, we need a large surface area. Alternatively, we can simulate many copies of a smaller surface and add them up for statistical calculation.

Now, we define the surface roughness as variance of the height

$$w = \sqrt{\langle h^2 \rangle - \langle h \rangle^2}. \tag{15.28}$$

For the simple random deposition model, we can calculate it analytically. The probability that the particle hits a lateral position is $p = 1/L$. The probability distribution of the height h after N particles are deposited is given by a binomial distribution

$$P(h; N) = \frac{N!}{h!(N-h)!} p^h (1-p)^{N-h} \quad \xrightarrow[N \to \infty]{} \quad \frac{1}{\sqrt{2\pi w^2}} \mathrm{e}^{-(h - \langle h \rangle)^2 / 2w^2}. \tag{15.29}$$

The mean height is

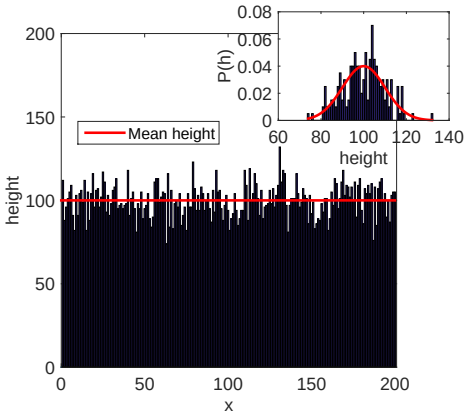
$$\langle h \rangle = \sum_{h=1}^{N} h P(h; N) = Np = \frac{N}{L} \tag{15.30}$$

which grows linearly with N as expected. The surface roughness is

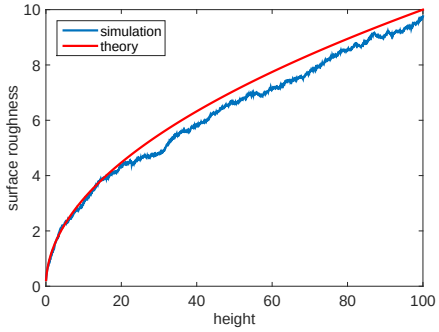
$$w = \sqrt{Np(1-p)} = \sqrt{\frac{N}{L}(1 - 1/L)} \approx \sqrt{\frac{N}{L}} = \sqrt{\langle h \rangle}. \tag{15.31}$$

As the height of the surface grows, the roughness also grows but as the sqrt of the height.

Now we turn to the simulation. Here is the algorithm.



(a) Surface roughening. (Inset: Height distribution)



(b) Growth of surface roughness.

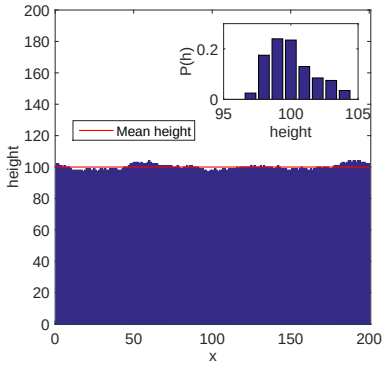
Figure 15.9: Surface growth with the ballistic deposition model without surface relaxation.

Algorithm 15.1 Ballistic deposition model without surface relaxation

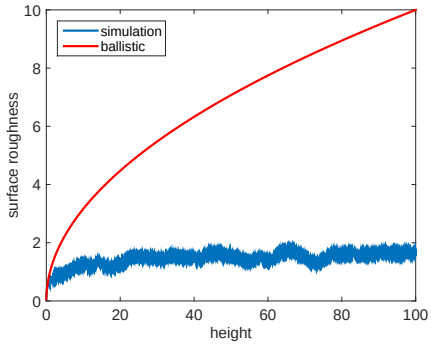
1. Set the height of the surface $y(x)$ to zero at all position $x =, \cdots, L$.
2. Generate a random position x between 1 and L.
3. Deposit the particle at x. (Increment $y(x)$ by one.
4. Repeat the deposition for N times.
5. Evaluate the mean height and the roughness.

Program 15.7 impliments this algorithm. Figures 15.9 show the results. The suraface size $L = 200$ and the number of particles $N = 100,000$ are used. The profile of the surface (Fig. 15.9a) shows that the surface is not smooth at all. Some part is much higher and other part much lower than the average height. The inset plots the distribution of height. The different between the higest and the lowest height is as big as 50, a half of the mean height! Figure 15.9b plots the growth of the surface roughness as a function of the mean height. The result of the simulation agrees with the theory (15.31). The simulation data is noisy because the surface area $L = 200$ is not big enough to get a good statistics.

The above result is a bit unrealistic since the surface roughness is not so big in the real matrials. A problem of the simple random deposition model is the surface roughness increases indefinitely, which never happens in the real world. A reasonable modification to the simple random deposition is to include the effect of the surface diffusion.[5] Once a particle is absorbed on to the surface, it can diffuse on the surface until it find a more stable position. It is called surface relaxation. It can be model by a biased random walk. The particle moves to a neighbor site lower than the present site. (See particle 4 in Fig. 15.8b.) If there are multiple sites lower than the present site, one of them are picked at random. (See particle 5 in Fig. 15.8b.) When the surface relaxation is taken into account, the roughness grows as $w = \langle h \rangle^{\beta}$ up to a certain value where β is called the growth exponent. When the mean height reaches a crossover height h_c, the roughness does not grow any loner and stay the same.[5]



(a) Surface roughening.



(b) Growth of surface roughness.

Figure 15.10: Surface growth with the ballistic deposition model with surface relaxation.

The following algorithm adds the surface relaxation to the ballistic deposition model.

Algorithm 15.2 Ballistic deposition model with surface relaxation

1. Set the height of the surface $y(x)$ to zero at all position $x =, \cdots, L$.
2. Generate a random position x between 1 and L.
3. If $y(x-1) \geq y(x)$ and $y(x+1) \geq y(x)$, then deposit it at x. Go to Step 7.
4. If $y(x-1) < y(x)$ and $y(x+1) > y(x)$, then
   - If $r < 0.5$, then $x = x - 1$ (jump to the left).
   - Otherwise, $x = x + 1$ (jump to the right).
   - Go back to Step 3.
5. $if y(x-1) > y(x)$, then $x = x + 1$ (jump to the right). Go back to Step 3.
6. The last possibility is $y(x+1) > y(x)$. Thus, $x = x - 1$ (jump to the left). Go back to Step 3.
7. Repeat the deposition for N times.
8. Evaluate the mean height and the roughness.

Program 15.8 impliments the algorithm and simulate the surface growth with relaxation. The results are plotted in Figures 15.10. The surface profile plotted in 15.10a indicates the the surface is much smoother now. The height distribution in the inset shows that the difference between the highest and lowest is less than 10. The growth of the surface roughness shown in Fig. **??** suggests that the roughness deos not grow after initial growth. Threfor,e this model is more realistic than the simple ballistic deposition model.

Finally, we includes a possibility to form the overhang (particle 3 in Fig. 15.8a) in Program 15.9. No surface relaxation is considered. Theresulting materials is highly porous. The profile of the surface plotted in

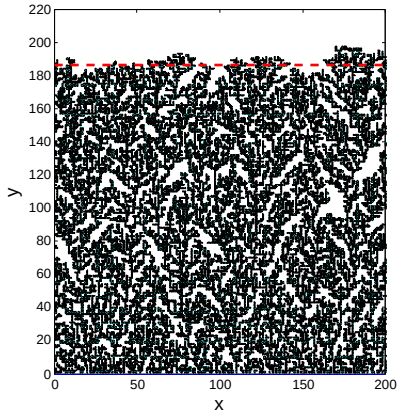


Figure 15.11: Growth of a surface based on a ballistic deposition model with possibility of overhang structures.

Fig. 15.11 shows many hollow regions. After $N = 200,000$ particles is deposited, the surface height reached nearly 200 which is twice as high as the previous growth models, indicating that almost a half of the space is not filled. Note also that the size of the empty space varies widely. See Ref. [4] for the detailed discussion of this growth pattern,

15.6 Problems

15.1 Using the random numbers, calculate the mean μ and variance σ^2 of random numbers uniformly distributed between 0 and 1. Compare results with the theoretical values.

15.2 Find $\langle |x| \rangle_{\text{normal}}$ by stochastic calculation and compare the result with the analytic solution.

MATLAB Source Codes

Program 15.1

```
%*************************************************************************
%*     Example 15.1                                                      *
%*     filename: ch15pr01.m                                              *
%*     program listing number: 15.1                                      *
%*                                                                       *
%*     This program simulate a dice using a psueo random number          *
%*     generator.    (random() in MATLAB is not used.)                   *
%*                                                                       *
%*     Programed by Ryoichi Kawai for Computational Physics Course.      *
%*     Last modification:  02/15/2015.                                   *
%*************************************************************************
clear all

% parapmeters for random number generators
a = int64(16807);
b = int64(0);
c = int64(2147483647);

% get a seed
x=int64(input('Seed='));

% generate uniform random numbers
N=6000;
for i=1:N
    x = mod(a*x,c);
    r(i)=double(x)/double(c);
end

% statistics of virtual die
P(1:6)=0;
for i=1:N
    n=ceil(6*r(i));
    P(n)=P(n)+1;
end

p=bar(P);
set(p,'facecolor',[0,0.75,0.75])
hold on
q=plot([0,7],[1000,1000],'--');
set(q,'color','red')
xlabel('face␣of␣die','fontsize',14)
ylabel('number␣of␣realization','fontsize',14)
hold off
```

Program 15.2

```
%*************************************************************************
%*     Example 15.2                                                      *
%*     filename: ch15pr02.m                                              *
%*     program listing number: 15.2                                      *
%*                                                                       *
%*     This program  evaluate the value of pi using the Monte Carlo      *
%*     integeration of a circle.                                         *
%*     Uses: rand() in MATLAB                                            *
%*                                                                       *
%*     Programed by Ryoichi Kawai for Computational Physics Course.      *
%*     Last modification:  02/25/2017.                                   *
```

```
%*************************************************************************
clear all
clc

% random points on a square
N=100000;
x=random('unif',-1.0,1.0,[1,N]);
y=random('unif',-1.0,1.0,[1,N]);

% points inside the circle
hit=0;
i=0;
for n=1:N
    if x(n)^2+y(n)^2 < 1
        hit=hit+1;
    end
    if mod(n,100)==0 % evaluate at every 100
        i=i+1;
        PI(i)=hit/n*4; % estimate of pi
    end
end

p=plot([1:i]*100,PI);
hold on
q=plot([0, N], [pi,pi]);
set(q,'color','black')
xlabel('# of sampling','fontsize',14)
ylabel('V_2','fontsize',14)
axis([0 N pi*0.9 pi*1.1])
hold off
```

Program 15.3

```
%*************************************************************************
%*     Example 15.3                                                      *
%*     filename: ch15pr03.m                                              *
%*     program listing number: 15.3                                      *
%*                                                                       *
%*     This program generates Gaussian distributed random numbers        *
%*     using the Box-Muller method.                                      *
%*                                                                       *
%*     Programed by Ryoichi Kawai for Computational Physics Course.      *
%*     Last modification:  02/25/2017.                                   *
%*************************************************************************
clear all

N=10000000;
x = rand(N,2);
y(:,1) = sqrt(-2*log(x(:,1))).*cos(2*pi*x(:,2));
y(:,2) = sqrt(-2*log(x(:,1))).*sin(2*pi*x(:,2));
ymin=min(y(:));
ymax=max(y(:));
fprintf('the largest deviation=%f, %f\n',ymin,ymax)

h=histogram(y,201,'Normalization','pdf');
hold on
% true normal distribution
K=201;
xmin=floor(ymin);
xmax=ceil(ymax);
X=linspace(xmin,xmax,K);
```

```
F=exp(-X.^2/2)/sqrt(2*pi);
p=plot(X,F);
set(p,'color','red','linewidth',2)
xlabel('x','fontsize',14)
ylabel('\rho(x)','fontsize',14)
legend('Box-Muller','Exact')
axis([xmin xmax 0.0 0.5])
hold off
```

Program 15.4

```
%*************************************************************************
%*     Example 15.4                                                      *
%*     filename: ch15pr04.m                                              *
%*     program listing number: 15.4                                      *
%*                                                                       *
%*     This program evaluates 1st through 4th moments of normal          *
%*     distribution.                                                     *
%*                                                                       *
%*     Programed by Ryoichi Kawai for Computational Physics Course.      *
%*     Last modification:  02/25/2017.                                   *
%*************************************************************************
clear all;

N=100000;

v=normrnd(0.0,1.0,[N,1]);

fprintf('order␣␣␣␣moment\n')
for i=1:4
    m(i)=sum(v.^i)/N;
    fprintf('%3d␣␣␣␣␣%␣10.4e\n',i,m(i));
end
r=m(4)/m(2)^2;
fprintf('\nm4/(m2)^2␣=␣%8.4d␣(exact=3)\n',r)
```

Program 15.5

```
%*************************************************************************
%*     Section 15.5.1                                                    *
%*     filename: ch15pr05.m                                              *
%*     program listing number: 15.5                                      *
%*                                                                       *
%*     This program evaluate the mean speed of the gas particles         *
%*     in a thermal equilibrium.                                         *
%*                                                                       *
%*     Programed by Ryoichi Kawai for Computational Physics Course.      *
%*     Last modification:  02/25/2017.                                   *
%*************************************************************************
clear all;

% parameters
T=300.;  % Temperature in K
k=1.380658e-23; % Boltzman constant in J/K
m=2*1.672623e-27; % H2 mass in kg

% Maxwell distribution
N=1000000;
s=sqrt(k*T/m);
v=normrnd(0.0,s,[N,3]); % 3 components (vx, vy, vz)
```

```
% speed
speed=sqrt(v(:,1).^2+v(:,2).^2+v(:,3).^2);
% mean
mean=sum(speed)/N;
% theory
exact=2*s*sqrt(2/pi);
%error
error=abs(mean-exact)/exact;

fprintf('mean␣speed␣=␣%10.5e␣(exact=%10.5e)\n',mean,exact)
fprintf('relative␣error␣=␣%10.5e\n',error)
```

Program 15.6

```
%*************************************************************************
%*     Section 15.5.2                                                    *
%*     filename: ch15pr06.m                                              *
%*     program listing number: 15.6                                      *
%*                                                                       *
%*     This program generates distribution of particles under gravity    *
%*     thermal diffusion.                                                *
%*                                                                       *
%*     Programed by Ryoichi Kawai for Computational Physics Course.      *
%*     Last modification:  02/26/2017.                                   *
%*************************************************************************
clear all

N=1000; % number of particles

x=rand(N,1); %horizontal position = uniform
y=rand(N,1); % vertical position = exponential
z=-log(y);
zmax=max(z(:))+1;

subplot(1,2,1)
p=plot([-0.005 1.005],[-0.05,-0.05], [-0.005,-0.005],...
       [-0.05,zmax],[1.005,1.005],[-0.05,zmax]);
set(p,'color','black','linewidth',2)
hold on
p=plot(x,z,'.');
set(p,'linewidth',2)
axis([-0.1 +1.1 -0.5 zmax])
xlabel('x','fontsize',14)
ylabel('z','fontsize',14)
hold off

subplot(1,2,2)
h=histogram(z,int32(2*zmax),'Normalization','pdf');
hold on
Z=linspace(0.0,zmax,201);
P=exp(-Z);
q=plot(Z,P,'--');
set(q,'linewidth',2,'color','red')
xlabel('z','fontsize',14)
ylabel('probability␣density','fontsize',14)
hold off
```

Program 15.7

```
%*************************************************************************
%*     Section 15.5.3                                                    *
%*     filename: ch15pr07.m                                              *
%*     program listing number: 15.7                                      *
%*                                                                       *
%*     This program simulatesa the surface growth using ballistic        *
%*     deposit model.                                                    *
%*                                                                       *
%*     Programed by Ryoichi Kawai for Computational Physics Course.      *
%*     Last modification:  02/26/2017.                                   *
%*************************************************************************
clear all  % clear all variables
close all  % close alll figures

N=20000; % number of particles to be deposited
L=200; % surface area

y=zeros(L,1); % reset the height of the surface
x=ceil(rand(N,1)*L);  % horizontal position of the particles

k=0;
for i=1:N
    y(x(i))=y(x(i))+1;  % ballistic growth

    % record the evolution of the growth after every 10 particles is
    % deposited
    if mod(i,10)==0
        k=k+1;
        z(k)=sum(y,1)/L;  % mean height
        w(k)=sqrt(sum((y-z(k)).^2)/L);  % roughness
    end
end

% calculate the height distribution
n1=min(y);  % lowest
n2=max(y);  % heighest
h=zeros(n2-n1+1,1);
for i=1:L
    n=y(i)-n1+1;
    h(n)=h(n)+1;
end
h=h/sum(h);

% theoretical height distribution (Gaussian formula)
mu=real(N)/L;
sg=real(N)*(L-1)/L^2;
g=zeros(n2-n1+1,1);
for n=n1:n2
    g(n-n1+1) = exp(-(n-mu)^2/(2*sg))/sqrt(2*pi*sg);
end

% Figure 1: profile of the surface
b=bar([1:L],y);
hold on
plot([0,L],[0,0]);  % draw the base line
axis equal % fix the aspect ratio (needed for movie)
axis([0 L+1 0 N/L*2])  % fix the axis range
hold on
p=plot([0,L],[z(k),z(k)]);
set(p,'color','red','linewidth',1)
legend(p,'Mean height')
```

```
xlabel('x','fontsize',14)
ylabel('height','fontsize',14)
hold off

% Figure 2: Evolution of the surface roughness
figure
p=plot(z,w);
hold on
set(p,'linewidth',2)
q=plot(z,sqrt(z));
set(q,'color','red','linewidth',2)
xlabel('height','fontsize',14)
ylabel('surface roughness','fontsize',14)
legend('simulation','theory')
legend('location','northwest')
hold off

% Figure 3: Heifht distribution
figure
bar([n1:n2],h);
hold on
p=plot([n1:n2],g);
set(p,'color','red','linewidth',2);
xlabel('height','fontsize',14)
ylabel('P(h)','fontsize',14)
legend('simulation','theory')
legend('location','northeast')
```

Program 15.8

```
%*************************************************************************
%*     Section 15.5.3                                                    *
%*     filename: ch15pr08.m                                              *
%*     program listing number: 15.8                                      *
%*                                                                       *
%*     This program simulatesa the surface growth using ballistic        *
%*     deposit model with surface relaxation.                            *
%*                                                                       *
%*     Programed by Ryoichi Kawai for Computational Physics Course.      *
%*     Last modification:  02/26/2017.                                   *
%*************************************************************************
clear all
close all;

% To show the real time growth, set true in th following line
movie = true;

N=20000;
L=200;

y=zeros(L,1);
x=floor(rand(N,1)*L);  % random position

k=0;
for i=1:N

    j0=mod(x(i),L)+1;  % random deposition

    % lateral diffusion
    found = false;
    while not(found)
```

```
        j1=mod(j0-2,L)+1;  % left neighbor
        j2=mod(j0,L)+1;    % right neighbor

        if y(j0)<=y(j1) && y(j0)<=y(j2)  % both sides are higher
            y(j0)=y(j0)+1;               % no diffusion
            found = true;
        elseif  y(j0)> y(j1) && y(j0)>y(j2)  % both sides are lower
            if rand() > 0.5
                j0=j1;                       % diffuse to the left
            else
                j0=j2;                       % diffuse to the right
            end
        elseif y(j0)<=y(j1)                  % left side is higher
            j0=j2;                           % diffuse to the right
        else                                 % right side is higher
            j0=j1;                           % diffuse to the eft
        end
    end

    % record the evolution of the growth after every 10 particles is
    % deposited    if mod(i,10)==0
        k=k+1;
        z(k)=sum(y,1)/L;
        w(k)=sqrt(sum((y-z(k)).^2)/L);
end


% calculate the height distribution
n1=min(y);  % lowest
n2=max(y);  % heighest
h=zeros(n2-n1+1,1);
for i=1:L
    n=y(i)-n1+1;
    h(n)=h(n)+1;
end
h=h/sum(h);

% Figure 1: profile of the surface
b=bar([1:L],y);
hold on
plot([0,L],[0,0]);  % draw the base line
axis equal % fix the aspect ratio (needed for movie)
axis([0 L+1 0 N/L*2])  % fix the axis range
hold on
p=plot([0,L],[z(k),z(k)]);
set(p,'color','red','linewidth',1)
legend(p,'Mean height')
xlabel('x','fontsize',14)
ylabel('height','fontsize',14)
hold off

% Figure 2: Evolution of the surface roughness
figure
p=plot(z,w);
hold on
set(p,'linewidth',2)
q=plot(z,sqrt(z));
set(q,'color','red','linewidth',2)
xlabel('height','fontsize',14)
ylabel('surface roughness','fontsize',14)
legend('simulation','ballistic')
```

```
legend('location','northwest')
hold off

% Figure 3: Heifht distribution
figure
bar([n1:n2],h);
hold on
xlabel('height','fontsize',14)
ylabel('P(h)','fontsize',14)
```

Program 15.9

```
%*************************************************************************
%*     Section 15.5.3                                                    *
%*     filename: ch15pr09.m                                              *
%*     program listing number: 15.9                                      *
%*                                                                       *
%*     This program simulatesa the surface growth using ballistic        *
%*     deposit model with overhangs.                                     *
%*                                                                       *
%*     Programed by Ryoichi Kawai for Computational Physics Course.      *
%*     Last modification:  02/15/2015.                                   *
%*************************************************************************
clear all
close all

N=20000;
L=200;

% preparation for plotting
plot([0,L],[0,0]);
axis equal
axis([0 L+1 0 N/L*1.5])
hold on

y=zeros(L,1);
x=floor(rand(N,1)*L);    % random position

k=0;
for i=1:N
    j0=mod(x(i),L)+1;
    j1=mod(x(i)-1,L)+1;
    j2=mod(x(i)+1,L)+1;
    if y(j0)<y(j1) || y(j0)<y(j2)
        y(j0)=max(y(j1),y(j2));   % stick to the next site
    else
        y(j0)=y(j0)+1;            % regular deposition
    end

    % draw the particle
    pos = [j0-0.5 y(j0)-0.5 1. 1.];
    rectangle('Position',pos,'Curvature',[1 1],'FaceColor','Blue')
    drawnow

    % record the evolution of the growth after every 10 particles is
    % deposited.
    if mod(i,10)==0
        k=k+1;
        z(k)=sum(y,1)/L;
        w(k)=sqrt(sum((y-z(k)).^2)/L);
    end
```

```
end

p=plot([0,L],[z(k),z(k)]);
set(p,'color','red','linewidth',2)
legend(p,'Mean height')
hold off
```

Python Source Codes

Program 15.1

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
%*************************************************************************
%*     Example 15.1                                                      *
%*     filename: ch15pr01.py                                             *
%*     program listing number: 16.1                                      *
%*                                                                       *
%*     This program simulate a dice using a psueo random number          *
%*     generator.                                                        *
%*                                                                       *
%*     Programed by Ryoichi Kawai for Computational Physics Course.      *
%*     Last modification:  02/25/2017.                                   *
%*************************************************************************
"""
import numpy as np
import matplotlib.pyplot as plt

# parapmeters for random number generators
a=np.int64(16807)
b=np.int64(0)
c=np.int64(2147483647)

# get a seed
x=np.int64(input('Seed='))

# generate uniform random numbers
N=6000
r=np.zeros(N)
for i in range(0,N):
    x=np.mod(a*x,c)
    r[i]=np.float(x)/np.float(c)

# statistics of virtual die
P=np.array([0,0,0,0,0,0])
D=np.array([1,2,3,4,5,6])
for i in range(0,N):
    n=np.int(np.ceil(6.0*r[i]))-1
    P[n]=P[n]+1;

plt.figure(figsize=(6,5))
plt.bar(D,P,0.9)
plt.plot([0.5,6.5],[N/6,N/6],'--r')
plt.xlabel('face of die',fontsize=14)
plt.ylabel('number of realization',fontsize=14)
plt.show()
```

Program 15.2

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
%*************************************************************************
%*     Example 15.2                                                      *
%*     filename: ch15pr02.m                                              *
%*     program listing number: 15.2                                      *
%*                                                                       *
%*     This program  evaluate the value of pi using the Monte Carlo      *
%*     integeration of a circle.                                         *
%*     Uses: numpy random package                                        *
%*                                                                       *
%*     Programed by Ryoichi Kawai for Computational Physics Course.      *
%*     Last modification:  02/25/2017.                                   *
%*************************************************************************
"""
import numpy as np
import matplotlib.pyplot as plt

# random points on a square
N=100000
x=np.random.uniform(-1.0,1.0,N)
y=np.random.uniform(-1.0,1.0,N)

# points inside the circle
hit=0.0
M=np.int(N/100)
PI=np.zeros(M)
i=0
for n in range(0,N):
    if x[n]**2+y[n]**2 < 1.0:
        hit+=1.0

    if n>0 and np.mod(n,100)==0: # evaluate at every 100
        PI[i]=hit/n*4.0 # estimate of pi
        i+=1

plt.figure(figsize=(6,5))
T=np.linspace(100,N,M)
plt.plot(T,PI)
plt.plot([0, N], [np.pi,np.pi],'--r')
plt.xlabel('# of sampling',fontsize=14)
plt.ylabel(r'$\pi$ by Monte Carlo integration',fontsize=14)
plt.axis([0, N, np.pi*0.9, np.pi*1.1])
plt.show()
```

Program 15.3

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
%*************************************************************************
%*     Example 15.2                                                      *
%*     filename: ch15pr02.m                                              *
%*     program listing number: 15.2                                      *
%*                                                                       *
%*     This program  evaluate the value of pi using the Monte Carlo      *
%*     integeration of a circle.                                         *
%*     Uses: numpy random package                                        *
```

```
%*                                                                      *
%*     Programed by Ryoichi Kawai for Computational Physics Course.    *
%*     Last modification:  02/25/2017.                                  *
%************************************************************************
"""
import numpy as np
import matplotlib.pyplot as plt

# random points on a square
N=100000
x=np.random.uniform(-1.0,1.0,N)
y=np.random.uniform(-1.0,1.0,N)

# points inside the circle
hit=0.0
M=np.int(N/100)
PI=np.zeros(M)
i=0
for n in range(0,N):
    if x[n]**2+y[n]**2 < 1.0:
        hit+=1.0

    if n>0 and np.mod(n,100)==0: # evaluate at every 100
        PI[i]=hit/n*4.0 # estimate of pi
        i+=1

plt.figure(figsize=(6,5))
T=np.linspace(100,N,M)
plt.plot(T,PI)
plt.plot([0, N], [np.pi,np.pi],'--r')
plt.xlabel('# of sampling',fontsize=14)
plt.ylabel(r'$\pi$ by Monte Carlo integration',fontsize=14)
plt.axis([0, N, np.pi*0.9, np.pi*1.1])
plt.show()
```

Program 15.4

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
%************************************************************************
%*     Example 15.4                                                     *
%*     filename: ch15pr04.m                                             *
%*     program listing number: 15.4                                     *
%*                                                                      *
%*     This program evaluates 1st through 4th moments of normal         *
%*     distribution.                                                    *
%*                                                                      *
%*     Programed by Ryoichi Kawai for Computational Physics Course.     *
%*     Last modification:  02/25/2017.                                  *
%************************************************************************
"""
import numpy as np

N=100000

v=np.random.normal(0.0,1.0,N)
m=np.zeros(4)
print('order ','  moment  ')
for i in [1,2,3,4]:
    m[i-1]=sum(v**i)/N
```

```
    print('{0:3d}␣␣␣{1:␣10.4e}'.format(i,m[i-1]))

r=m[3]/m[1]**2
print('\n␣m4/m2**2={0:8.4f}␣(exact=3)'.format(r))
```

Program 15.5

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
%*************************************************************************
%*     Section 15.5.1                                                    *
%*     filename: ch15pr05.m                                              *
%*     program listing number: 15.5                                      *
%*                                                                       *
%*     This program evaluate the mean speed of the gas particles         *
%*     in a thermal equilibrium.                                         *
%*                                                                       *
%*     Programed by Ryoichi Kawai for Computational Physics Course.      *
%*     Last modification:  02/25/2017.                                   *
%*************************************************************************
"""
import numpy as np

# parameters
T=300.              # Temperature in K
k=1.380658e-23     # Boltzman constant in J/K
m=2*1.672623e-27   # H2 mass in kg

# velocity at Maxwell distribution
N=100000000
s=np.sqrt(k*T/m)
v=np.random.normal(0.0,s,[N,3]) # 3 components (vx, vy, vz)

# speed
speed=np.sqrt(v[:,0]**2+v[:,1]**2+v[:,2]**2)
# mean
mean=sum(speed)/N
#theory
exact=2*s*np.sqrt(2/np.pi)
# error
error=np.abs(mean-exact)/exact

print('mean␣speed␣=␣{0:10.5e}␣(exact={1:10.5e})'.format(mean,exact))
print('relative␣error␣=␣{0:10.5}\n'.format(error))
```

Program 15.6

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
%*************************************************************************
%*     Section 15.5.2                                                    *
%*     filename: ch15pr06.py                                             *
%*     program listing number: 16.6                                      *
%*                                                                       *
%*     This program generates distribution of particles under gravity    *
%*     thermal diffusion.                                                *
%*                                                                       *
%*     Programed by Ryoichi Kawai for Computational Physics Course.      *
```

```
%*     Last modification:  02/26/2017.                                  *
%*********************************************************************
"""
import numpy as np
import matplotlib.pyplot as plt

N=1000  # number of particles

 # horizontal position = uniform
x=np.random.rand(N)
# vertical position = exponential
y=np.random.rand(N) # vertical position = exponential
z=-np.log(y)
zmax=np.max(z)+1.


plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot([-0.05,-0.05],[-0.05, zmax],'-k',linewidth=2)
plt.plot([-0.05, 1.05],[-0.05,-0.05],'-k',linewidth=2)
plt.plot([ 1.05, 1.05],[-0.05, zmax],'-k',linewidth=2)
plt.plot(x,z,'.');
plt.axis([-0.1, +1.1, -0.5, zmax])
plt.xlabel('x',fontsize=14)
plt.ylabel('z',fontsize=14)
plt.subplot(1,2,2)
plt.hist(z,2*np.int(zmax),normed=1)
Z=np.linspace(0.0,zmax,201)
P=np.exp(-Z)
plt.plot(Z,P,'--r',linewidth=2)
plt.xlabel('z',fontsize=14)
plt.ylabel('probability␣density',fontsize=14)
plt.show()
```

Program 15.7

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
%*********************************************************************
%*     Section 15.5.3                                                *
%*     filename: ch15pr07.m                                          *
%*     program listing number: 15.7                                  *
%*                                                                   *
%*     This program simulatesa the surface growth using ballistic    *
%*     deposit model.                                                *
%*                                                                   *
%*     Programed by Ryoichi Kawai for Computational Physics Course.  *
%*     Last modification:  02/26/2017.                               *
%*********************************************************************
"""
import numpy as np
import matplotlib.pyplot as plt

N=20000  # number of particles to be deposited
L=200    #surface area

y=np.zeros(L,dtype=np.int)  # reset the height of the surface
x=np.random.randint(0,L,size=N)  # horizontal position of the particles

z=np.zeros(N)
```

```
w=np.zeros(N)
k=0
for i in range(0,N):
    y[x[i]]+=1  # ballistic growth

    # record the evolution of the growth after every 10 particles is deposited
    if np.mod(i,10)==0:
        z[k]=sum(y.astype(float))/L  # mean height
        w[k]=np.sqrt(sum((y.astype(float)-z[k])**2)/L)  # roughness
        k+=1

# calculate the height distribution
n1=min(y)  # lowest
n2=max(y)  # heighest
Ny=n2-n1+1
n=np.linspace(n1,n2,Ny)
h=np.zeros(Ny,dtype=np.int)

for i in range(0,L):
    j=y[i]-n1
    h[j]+=1
# normalization
h=h.astype(float)/sum(h)

# theoretical height distribution (Gaussian formula)
m=np.float(N)/L
s=np.float(N*(L-1))/L**2
g=np.exp(-(n-m)**2/(2.*s))/np.sqrt(2*np.pi*s)

# Figure 1: profile of the surface
plt.figure(figsize=(12,5))
X=np.linspace(1.0,L,L)
plt.bar(X,y,1.05,color='k')
plt.plot([0,L],[0,0],'-k',linewidth=4)  # draw the base line
plt.plot([0,L],[z[k-1],z[k-1]],'--r',label='Mean height')
plt.xlabel('x',fontsize=14)
plt.ylabel('height',fontsize=14)
plt.show()

# Figure 2: Evolution of the surface roughness
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(z[0:k],w[0:k],'-k',label='simulation')
plt.plot(z[0:k],np.sqrt(z[0:k]),'--r',label='theory')
plt.xlabel('height',fontsize=14)
plt.ylabel('surface roughness',fontsize=14)
plt.legend(loc=3)

# Figure 3: Heifht distribution
plt.subplot(1,2,2)
plt.bar(n,h,1.05,color='k')
plt.plot(n,g,'--r')

plt.xlabel('height',fontsize=14)
plt.ylabel('P(h)',fontsize=14)
plt.legend(loc=1)
plt.show()
```

Program 15.8

```
#!/usr/bin/env python3
```

```
# -*- coding: utf-8 -*-
"""
%*************************************************************************
%*     Section 15.5.3                                                    *
%*     filename: ch15pr08.m                                              *
%*     program listing number: 15.8                                      *
%*                                                                       *
%*     This program simulatesa the surface growth using ballistic        *
%*     deposit model with surface relaxation.                            *
%*                                                                       *
%*     Programed by Ryoichi Kawai for Computational Physics Course.      *
%*     Last modification:  02/26/2017.                                   *
%*************************************************************************
"""
import numpy as np
import matplotlib.pyplot as plt

N=20000
L=200

y=np.zeros(L,dtype=np.int)  # reset the height of the surface
x=np.random.randint(0,L,size=N)  # horizontal position of the particles

z=np.zeros(N)
w=np.zeros(N)
k=0
for i in range(0,N):

    # lateral diffusion
    j0=x[i]
    found = False
    while not(found):
        j1=np.mod(j0-1,L) # left neighbor
        j2=np.mod(j0+1,L) # right neighbor

        if y[j0]<=y[j1] and y[j0]<=y[j2]:  # both sides are higher
            y[j0]+=1                       # no diffusion
            found = True
        elif y[j0]>y[j1] and y[j0]>y[j2]:   # both sides are lower
            if np.random.rand() > 0.5:
                j0=j1                       # diffuse to the left
            else:
                j0=j2                       # diffuse to the right

        elif y[j0]<=y[j1]:                  # left side is higher
            j0=j2                           # diffuse to the right
        else:                               # right side is higher
            j0=j1                           # diffuse to the eft


    # record the evolution of the growth after every 10 particles is
    # deposited
    if np.mod(i,10)==0:
        z[k]=sum(y.astype(float))/L  # mean height
        w[k]=np.sqrt(sum((y.astype(float)-z[k])**2)/L)  # roughness
        k+=1

# calculate the height distribution
n1=min(y)  # lowest
n2=max(y)  # heighest
Ny=n2-n1+1
```

```
n=np.linspace(n1,n2,Ny)
h=np.zeros(Ny,dtype=np.int)

for i in range(0,L):
    j=y[i]-n1
    h[j]+=1
# normalization
h=h.astype(float)/sum(h)

# Figure 1: profile of the surface
plt.figure(figsize=(12,5))
X=np.linspace(1.0,L,L)
plt.bar(X,y,1.05,color='k')
plt.plot([0,L],[0,0],'-k',linewidth=4)  # draw the base line
plt.plot([0,L],[z[k-1],z[k-1]],'--r',label='Mean␣height')
plt.xlabel('x',fontsize=14)
plt.ylabel('height',fontsize=14)
plt.show()

# Figure 2: Evolution of the surface roughness
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(z[0:k],w[0:k],'-k',label='simulation')
plt.plot(z[0:k],np.sqrt(z[0:k]),'--r',label='theory')
plt.xlabel('height',fontsize=14)
plt.ylabel('surface␣roughness',fontsize=14)
plt.legend(loc=3)

# Figure 3: Heifht distribution
plt.subplot(1,2,2)
plt.bar(n,h,1.05,color='k')
plt.xlabel('height',fontsize=14)
plt.ylabel('P(h)',fontsize=14)
plt.show()
```

Program 15.9

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
%*************************************************************************
%*     Section 15.5.3                                                    *
%*     filename: ch15pr09.m                                              *
%*     program listing number: 15.9                                      *
%*                                                                       *
%*     This program simulatesa the surface growth using ballistic        *
%*     deposit model with overhangs.                                     *
%*                                                                       *
%*     Programed by Ryoichi Kawai for Computational Physics Course.      *
%*     Last modification:  02/26/2017.                                   *
%*************************************************************************
"""
import numpy as np
import matplotlib.pyplot as plt

movie=False  # Set this to True to show real time growth (very slow)

N=20000
L=200
fig, ax=plt.subplots()
```

```
ax.set_xlim([0,L])
ax.set_ylim([0,N/L*3])


y=np.zeros(L,dtype=np.int)  # reset the height of the surface
x=np.random.randint(0,L,size=N)  # horizontal position of the particles

z=np.zeros(N)
w=np.zeros(N)
k=0
for i in range(0,N):

    # lateral diffusion
    j0=x[i]
    j1=np.mod(j0-1,L) # left neighbor
    j2=np.mod(j0+1,L) # right neighbor
    if y[j0]<y[j1] or y[j0]<y[j2]:
        y[j0]=np.max( (y[j1],y[j2]) )  # stick to the next site
    else:
        y[j0]+=1                # regular deposition

    # draw the particle
    c=plt.Circle((j0,y[j0]), 0.5, color='b')
    ax.add_artist(c)
    if movie:
        plt.pause(0.0001)

    # record the evolution of the growth after every 10 particles is
    # deposited
    if np.mod(i,10)==0:
        z[k]=sum(y.astype(float))/L  # mean height
        w[k]=np.sqrt(sum((y.astype(float)-z[k])**2)/L)  # roughness
        k+=1

ax.plot([0,L],[z[k-1],z[k-1]],'--r')
plt.show()
```

Bibliography

[1] Daniel Zwillinger. *CRC Stanbdard Mathematical Tables and Formula.* CRC Press, 35th edition, 2012. Section 6.14.

[2] N. G. Van Kampen. *Stochastic Processes in Physics and Chemistry.* North Holland, 3rd edition, 2007. Section I.7.

[3] Mário N. Berberan-Santos, EvenyN. Bodunov, and Lionello Pogliani. On the barometric formula. *American Journal of Physics*, 65:404–412, 1997.

[4] Albert-Laszlo Barabasi and Harry Eugene Stanley. *Fractal Concepts in Surface Growth.* Cambridge University Press, 1995. Chapter 2.

[5] Albert-Laszlo Barabasi and Harry Eugene Stanley. *Fractal Concepts in Surface Growth.* Cambridge University Press, 1995. Chapter 5.

CHAPTER 16

RANDOM WALKS

A drunkard leaves his favorite bar and walks to his home. After N steps, how far is he from the bar? This is a basic question of random walk problems. An interesting mathematics such as Wiener process evolved from this simple question and many important theories have been developed in many fields of science based on the random walk model. In this chapter, we focus on discrete random walks where step size is finite and fixed. Continuous random walk is discussed in Chapter 18.

16.1 One-dimensional Random Walk

A particle in a one-dimensional space jumps from one site to an adjacent site at random with a probability $p_L = p$ to the right and $p_R = 1 - p$ to the left. See Fig. 16.1. The position of the particle is specified by integer index assigned to the grid point. We assume $p = \frac{1}{2}$ for now. Then, we have unbiased random walk ($p_L = p_R$). Initially a particle is placed at x_0. Where is the particle after N steps? There is no definite answer to this question. The trajectory of the particle is not uniquely determined by the initial condition since the direction of jump is probabilistic. Therefore, the position of the particle at time t is stochastic variable $\hat{X}_t$ defined with sample space $x \in \mathbb{Z}$ and probability distribution $P_t(x)$. Here time, $t = 0, 1, \cdots, N$ is just the number of jumps the particle made and thus discrete. The stochastic variable $\hat{X}_t$ as a function time t is a sequence of random variables $\{\hat{X}_0, \hat{X}_1, \hat{X}_2, \cdots\}$, which is called stochastic process. For example, if the particle was initially at $x = 0$, the possible outcome is $\{0\}$the probability is $P_0(x) = \delta_{x\,0}$ where $\delta_{m\,n}$ is a Kronecker's delta. At $t = 1$, the particle is either at $x = 1$ or $x = -1$. Thus, the possible outcome is $\{\pm 1\}$

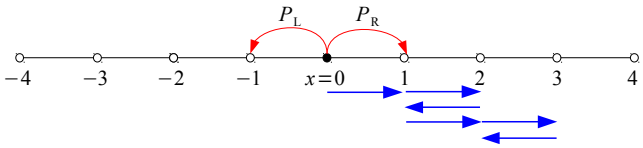


Figure 16.1: One-dimensional discrete random walk. The blue arrows indicate a realization of 6-steps trajectory, $RRLRRL$

and the associate probability is

$$P_1(x) = \begin{cases} \frac{1}{2} & \text{for } x = \pm 1 \\ 0 & \text{otherwise} \end{cases}. \tag{16.1}$$

After the second jumps, the possible outcomes of $\hat{X}_2$ are now $\{-2, 0, 2\}$ with the probability

$$P_2(x) = \begin{cases} \frac{1}{2} & \text{for } x = 0 \\ \frac{1}{4} & \text{for } x = \pm 2 \\ 0 & \text{otherwise} \end{cases}. \tag{16.2}$$

This problem can be solved analytically for $t = N$. Suppose that the particle jumps to the right N_R times and to the left $N_L = N - N_R$. (Note that $N = N_R + N_L$.) For example, the blue arrows in Fig. 16.1 represents an trajectory of $N = 6$ steps of which $N_R = 4$ steps to the right and $N_L = 2$ to the left. The final position $x(N_R, N) = (N_R - N_L) = 2$. The probability to have this particular trajectory $RRLRRL$ is $\left(\frac{1}{2}\right)^6$. However, several other trajectories have the same N_R and N_L, e.g., $RRRRLL$. Simple combinatorial calculation tells that there are

$$W(N_R, N) = \frac{N!}{N_R!\,(N - N_R)!} \tag{16.3}$$

different ways to reach the same point. Noting that the final point is $x = (N_R - N_L) = (2N_R - N)$, $N_R = \frac{1}{2}(N + x)$ and $N_L = \frac{1}{2}(N - x)$. Hence, the probability to find the particle at x after N steps is

$$P_N(x) = \frac{N!}{\left(\frac{N+x}{2}\right)!\left(\frac{N-x}{2}\right)!}\left(\frac{1}{2}\right)^N. \tag{16.4}$$

When $N \pm x$ is not an even integer, this result fails. This is because when N is even, the particle cannot stop at any odd site and similarly when N is odd, no even site is reachable by the particle.

When $N \gg 1$, the probability (16.4) becomes Gaussian*

$$P_N(x) \approx \frac{1}{\sqrt{2\pi N}} e^{-x^2/2N}. \tag{16.5}$$

The mean position is $\langle X_N \rangle = 0$ for any N. The variance increases as $\langle X_N^2 \rangle = N$. This means that on average the drunkard is still at the bar after a long walk!

*A special care is needed to make x continuous since x/a is exclusively even or odd.