Synthesis Lectures on
Mathematics & Statistics

SYNTHESIS
COLLECTION OF TECHNOLOGY

Shinil Cho

Monte Carlo Simulations Using Microsoft EXCEL®

Springer

Synthesis Lectures on Mathematics & Statistics

This series includes titles in applied mathematics and statistics for cross-disciplinary STEM professionals, educators, researchers, and students. The series focuses on new and traditional techniques to develop mathematical knowledge and skills, an understanding of core mathematical reasoning, and the ability to utilize data in specific applications.

Shinil Cho

Monte Carlo Simulations Using Microsoft EXCEL®

Springer

Shinil Cho
La Roche University
Pittsburgh, PA, USA

ISSN 1938-1743 ISSN 1938-1751 (electronic)
Synthesis Lectures on Mathematics & Statistics
ISBN 978-3-031-33885-4 ISBN 978-3-031-33886-1 (eBook)
https://doi.org/10.1007/978-3-031-33886-1

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

温故知新 (visiting old, learn new)

Another book on the Monte Carlo simulation? Yes, but this is a kind of book I wanted to have when I was a student. It guides you to explore the fantastic world of Monte Carlo simulations and acquire basic computational knowledge to create and run your own simulation programs. The intended readers are undergraduate to graduate students who are interested in engaging in simulation projects. There are several frequently cited simulation examples from probability distribution functions, computation of π-value, nuclear decay, and random walks, which are extended to classical diffusion problems, quantum diffusion Monte Carlo method, and Ising models along with descriptions of their procedures. A brief introduction of quantum annealing for optimization utilizing Ising models, and descriptions of chaos and fractals are also included with actual examples.

The programming language used in this book is the built-in Visual Basic for Application (VBA) of EXCEL®. The language is simple for numerical computation and can re-write legacy BASIC programs found in other renowned books. EXCEL's data analysis and chart capabilities are utilized for easier data analysis so that simulation codes are slim to show the essential part of the Monte Carlo method. The VBA codes in this book are not written in an elegant manner but the author attempts to present more descriptive codes for further applications by the readers.

Traditionally there always a discussion of how to generate random numbers before explaining simulation algorithms. While it is indeed critical to use "true" random numbers for random samplings, we conveniently use the EXCEL's "quasi-random" number generator, RND( ). There is no discussion of the random number generator, nor estimation of accuracy of the simulation results. As you learn more, these issues on simulations should be considered for better results for research, referring to more advanced books.

Computer simulations are like "real" experiments. You may not get what you predict at first. Still, it will be a scientist's satisfaction to watch outputs from your own simulation codes agree with theories. Have a wonderful simulation experience!

Pittsburgh, USA — Shinil Cho
Spring 2023

Contents

Probability Distribution Functions

1

Random phenomena are described with stochastic variables and associated probability distribution functions. Stochastic variables take sets of possible values, and their probability distribution functions are maps of the stochastic variables that show the distributions of what to be observed. The first chapter of this book overviews the frequently-appearing probability distributions, including the binomial, the Poisson, the Gaussian, and the uniform distributions. Several examples from physics are used to describe these distribution functions.

One of the important theorems of probability is the central limit theorem. A brief explanation of the theme is: the sum of many independent stochastic variables is also a stochastic variable, and tends to approach a Gaussian distribution with more and more terms. Any probability distribution function follows this theorem. We confirm the theorem with the probability distributions mentioned above.

1.1 Electron Spins in Magnetic Field—Binomial Distribution

1.1.1 Configuration of Spin Array

Imagine a horizontal array of N-free electron system [1]. If external magnetic field $\boldsymbol{B}$ is applied to the spin system in the vertical direction, then each of the electron spins array points either "up" or "down" in the z-direction with certain probabilities. Denote p as the probability that a spin is up and q as the probability that it is down. Because each spin is either up or down, the total probability must be one: $p + q = 1$. The spin state takes the equal probability, $p = q = 1/2$, when $\boldsymbol{B} = 0$.

S. Cho, *Monte Carlo Simulations Using Microsoft EXCEL®*,
Synthesis Lectures on Mathematics & Statistics,
https://doi.org/10.1007/978-3-031-33886-1_1

Among the N spins, suppose n spins are up and the remaining $n' = N - n$ spins are down. What is the probability $P(n)$ to have the configuration in the magnetic field $\boldsymbol{B}$? It is the product of the probability of each spin state from the first to the Nth spin:

$$pp\cdots pqq\cdots q = p^n q^{n'} = p^n q^{N-n} \tag{1.1}$$

If we concern only the number of up spins to be n out of the total number of spins, N, we need to find the number of possible ways to pick n spins out of N spins. That number is given by combinations, $C_N(n)$:

$$C_N(n) = \frac{N!}{n!(N-n)!}. \tag{1.2}$$

The probability $P(n)$ we are seeking is, thus, given by

$$P(n) = C_N(n)p^n q^{N-n} = \frac{N!}{n!(N-n)!}p^n q^{N-n} \tag{1.3}$$

This is the probability distribution function called the binomial distribution because the expression appears in the binomial expansion:

$$(p+q)^N = \sum_{n=0}^{N} \frac{N!}{n!(N-n)!}p^n q^{N-n} \tag{1.4}$$

Let's simulate the probability distribution to investigate how this distribution function looks like.

1.1.2 Simulation of Binominal Distribution

Consider an example of $N = 12$ free spins and $p = q = 1/2$. Why $N = 12$? We will explain why $N = 12$ is the preferred choice of the simulation in Sect. 1.5. For simulating the binominal distribution, we can flip a coin to set p (head) and q (tail). Alternatively, we can use a computer to simulate the probability distribution by generating a random number between 0 and 1, and making a up spin if the random numbers is less than 0.5, and a down spin if the number is equal and more than 0.5. Using random numbers is much more efficient to conduct the simulation many times. In this book, we use the EXCEL's built-in function, `RND()`.

Figure 1.1 lists a VBA code which outputs the total numbers of up spins and down spins out of the 12 spins in each trial. The total number of trials is 2,000 in this code. Then, we analyze the output data by using [`Histogram`] of the pull-down menu [`Data Analysis`] to make a tally of up spins. Next, we calculate the normalized probability to obtain n-up spins, `P(n) = [Frequency]/[Total number of trials]`, where $n = 0$ to 12 (no up spin to all spins up).

```
Sub Binominal()
Cells(1, 1)="Simulation of the binominal distribution function"
    Cells(2, 3) = "Binominal"
    Cells(3, 2) = "# of up-spins"
    Dim PU(2000)                    '2000 trials.
    N = 12                          'An array of 12 spins.
Randomize                           'Initialize RND()-function.
For i = 1 To 2000                   'Coin flipping 2000 times.
       UP = 0                       '# of up spins.
       DN = 0                       '# of down spins.
          For k = 1 To N            'Determine spin up/down of each of 12 spins.
              If Rnd() < 0.5 Then
                 UP = UP + 1
          Else
                 DN = DN + 1
          End If
              Next k
    PU(i) = UP
  Cells(3 + i, 2) = PU(i)   'Write # of up-spins for each trial in cell(row, column).
Next i
End Sub
```

Fig. 1.1 VBA code for acquiring number of up-spins form 2,000 trials

NOTE: Refer to Appendix A1.1 for enabling EXCEL's VBA macro. The default setting of [`Data Analysis`] is disabled. Refer to Appendix A1.2 for enabling the capability.

In Fig. 1.2, the smooth curve onset to the simulated probability distribution is the exact binomial functions with $N = 12$ and $p = q = 0.5$. Although the binominal function is discrete, we used a smooth curve for better graphical presentation. The random number-based simulation represents the binominal probability distribution very well.

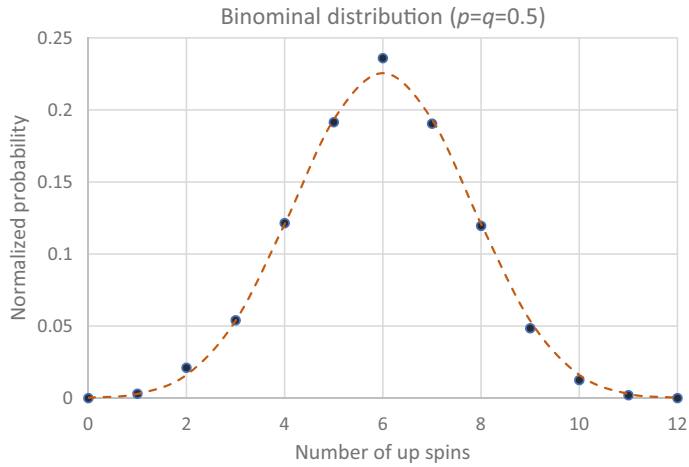


Fig. 1.2 Binominal distribution of 12 spins of equal probabilities of up/down orientations

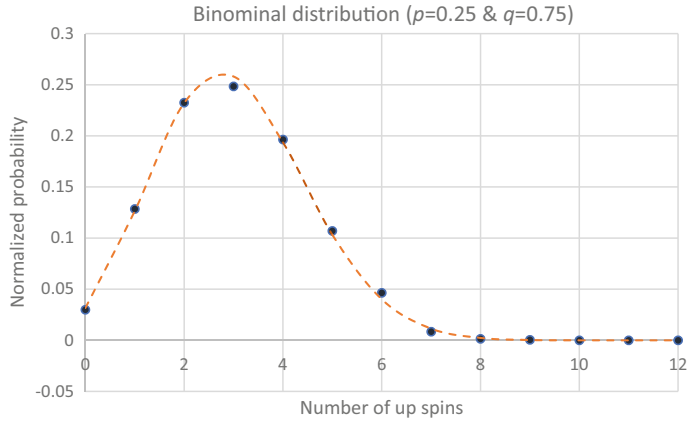


Fig. 1.3 Binominal distribution of 12 spins of unequal probabilities of up/down orientations

A simulation of nonequal probabilities of p and q can be also performed by changing the accepted range of random numbers for up spins. For example, if $p = 0.25$ and $q = 0.75$, when random numbers less than 0.25 as p, otherwise q:

```
If Rnd() < 0.25 Then
UP = UP + 1
Else
DN = DN + 1
End If
```

Figure 1.3 shows the simulated binomial of $p = 0.25$ and $q = 0.75$ with the exact distribution.

1.2 Radioactive Decay—Poisson Distribution

1.2.1 Decay Equation

The radioactive decays can be described by the following differential equation

$$\frac{dN}{dt} = -\lambda N \tag{1.5}$$

and its solution is an exponential function, $N(t) = N_0 e^{-\lambda t}$, where $N(t)$ is the number of active (i.e., not yet decayed) radioactive nuclei at time t, N_0 is $N(t = 0)$, and λ is the decay constant [2].

Suppose we count of radioactive decays for, say, one-minute, and repeat the same measurements several times, the measured counts fluctuate at each time because radioactive nuclei decay randomly, and we do not know which and when the decay occurs. It is said that the probability of observing n decays per unit time (at time t) follows the Poisson distribution function

$$P_n(t) = \frac{v^n}{n!}e^{-v} \tag{1.6}$$

where n is the number of decays observed per unit time, and v is the average number of decays per unit time [3]. The ratio of the number of the active nuclei at time t, $N(t)$, to the total number of active nuclei at $t = 0$, N_0, $N(t)/N_0 = e^{-\lambda t}$, is called the decay rate, but the decay rate is not the number of decayed radioactive nuclei decay per unit time at time t. How can we obtain the Poisson probability from the exponential decay equation?

The Poisson distribution of the radioactive decay can be derived in the following way. The mean life time of a radioactive nucleus is given by $T_{mean} = \int_0^\infty t\,w(t)dt = 1/\lambda$ where $w(t)\mathrm{d}t$ is the normalized probability of occurring a single decay in a time interval $[t, t + dt]$ or the normalized decay rate, which should be given by $A \cdot N(t)/N_0 = Ae^{-\lambda t}$ where A is the normalization constant. Because the total probability must be 1, the normalization constant can be calculated:

$$\int_0^\infty w(t)dt = A\int_0^\infty e^{-\lambda t}dt = 1, \text{ and thus } A = \lambda.$$

Therefore, the normalized probability of a single decay in $[t, t + \mathrm{d}t]$ is given by $w(t)dt = \lambda e^{-\lambda t}dt$.

Using $w(t)$, the probability that no decay occurs in the time t is given by the following probability distribution function:

$$P_0(t) = \left[1 - \int_0^t w(t')dt'\right] = e^{-\lambda t}. \tag{1.7}$$

Next, suppose only a single decay occurs in time t. If the single decay occurs at time t_1 where $t_1 \le t$, then no decay must be observed after t_1 until t. Therefore, we obtain

$$P_1(t) = \int_0^t dt_1 w(t_1)P_0(t - t_1) = \int_0^t dt_1\left(\lambda e^{-\lambda t_1}\right)\left(e^{-\lambda(t-t_1)}\right) = \lambda t e^{-\lambda t}. \tag{1.8}$$

Suppose the probability that two decays occur in time t. Because the second decay is not affected by the first decay, that is if the decay process is a Markov process, we can apply Eq. (1.8) for each decay. Namely, if the first single decay occurs at time t_1, and the second single decay occurs at time t_2 where $t_1 \le t_2 \le t$, the probability of the second

decays at t_2 in the time interval $t - t_1$ is given by Eq. (1.8) with t replaced with $t - t_1$. In this way, we obtain the probability of two decays given by

$$P_2(t) = \int_0^t dt_1 w(t_1) P_1(t - t_1) dt'$$
$$= \int_0^t dt_1 \left(\lambda e^{-\lambda t_1}\right)\left(\lambda(t - t_1) e^{-\lambda(\xi - t_1)}\right) = \frac{(\lambda t)^2}{2} e^{-\lambda t} dt. \tag{1.9}$$

Now, we apply the mathematical induction. Guessing from Eq. (1.9), we may assume that the probability of occurring n-decays in time t, the first decay occurs at time t_1, and $(n - 1)$ decays occur after t_1. That is, the probability of n-decays, $P_n(t)$, would be

$$P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t}. \tag{1.10}$$

With Eq. (1.10), the probability of $(n + 1)$-decays in the time t where the first decay occurs at time t_1, and n-decays occur after t_1 is given by

$$P_{n+1}(t) = \int_0^t dt_1 w(t_1) P_n(t - t_1) = \int_0^t dt_1 \left(\lambda e^{-\lambda t_1}\right)\left(\frac{\lambda^n (t - t_1)^n}{n!} e^{-\lambda(t - t_1)}\right) = \frac{(\lambda t)^{n+1}}{(n + 1)!} e^{-\lambda t}. \tag{1.11}$$

Therefore, the probability of radioactive decay of n nuclei in time t is given by the probability distribution

$$P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t}. \tag{1.12}$$

We are getting the Poisson function! Now, the average number of decays, ν, can be calculated by

$$\nu = \sum_{n=0} n P_n(t) = \sum_{n=0} \frac{n(\lambda t)^n}{n!} e^{-\lambda t} = (\lambda t) e^{-\lambda t} \sum_{n=1} \frac{(\lambda t)^{n-1}}{(n - 1)!} = (\lambda t) e^{-\lambda t} e^{+\lambda t} = \lambda t. \tag{1.13}$$

Equation (1.13) means that, if we recall that the constant λ is the decay rate, i.e., the number of decays in time t should be given by λt. Because n/N is very small for radioactive decays, n is usually close to the average of n. Thus, it may be replaced with the average number of decays, ν.

We finally we obtain the Poisson distribution function (1.6) as the probability of observing n-decays in a given time interval.

1.2.2 Binominal Distribution to Poisson Distribution

The Poisson distribution of observed number of radioactive nuclear decays can be also derived from the binominal distribution we discussed in Sect. 1.1. Suppose the probability of a single decay until time t is p, and then the probability of un-decay is $q = 1 - p$. Of the total N nuclei, the probability of decaying n nuclei (and thus not decaying the rest of $N - n$ nuclei) is given by $p^n q^{N-n}$. Recall that there are $C_N(n)$-ways to select the n nuclei from N nuclei, which is given by

$$C_N(n) = \frac{N!}{n!(N-n)!}. \tag{1.14}$$

Therefore, the probability of radioactive decay of n nuclei in time t is given by the binominal distribution

$$P(n) = \frac{N!}{(N-n)!n!} p^n q^{N-n} \tag{1.15}$$

Now, because $p = n/N$ is very small in terms of radioactive decays, n is usually close to the average of n, i.e., ν, we use $p \approx \nu/N << 1$. Now, Eq. (1.15) can be rewritten as

$$P(n) = \frac{N!}{(N-n)!n!} \left(\frac{\nu}{N}\right)^n \left(1 - \frac{\nu}{N}\right)^{N-n}. \tag{1.16}$$

By using the Sterling's approximation for an integer m, $m! \approx \sqrt{2\pi m} \cdot e^{-m} m^m$, and an approximation for exponential function, $(1 - \frac{\nu}{N})^N \approx \lim_{N\to\infty} (1 - \frac{\nu}{N})^\nu = e^{-\nu}$, the binomial distribution approaches to the Poisson distribution, $P(n) = \frac{\nu^n}{n!} e^{-\nu}$, when the probability p of the binomial distribution is small in such a way that the product np remains finite.

Proof From Eq. (1.16) and the Stirling's approximation, we obtain

$$\begin{aligned} P(n) &\approx \frac{\sqrt{N} \cdot e^{-N} \cdot N^N}{\sqrt{N-n} \cdot e^{-(N-n)} \cdot (N-n)^{(N-n)}} \frac{1}{n!} \left(\frac{\nu}{N}\right)^n \left(1 - \frac{\nu}{N}\right)^{N-n} \\ &\approx \sqrt{\frac{N}{N-n}} \frac{1}{e^n} \frac{1}{\left(1 - \frac{n}{N}\right)^{N-n}} \frac{\nu^n}{n!} e^{-\nu} \left(1 - \frac{\nu}{N}\right)^{-n} \\ &\approx \frac{\left(1 - \frac{\nu}{N}\right)^{-n}}{\left(1 - \frac{n}{N}\right)^{\frac{1}{2}-n}} \frac{\nu^n}{n!} e^{-\nu} \approx \frac{\nu^n}{n!} e^{-\nu}. \end{aligned}$$

■

1.3 Gaussian Distribution

1.3.1 Poisson to Gaussian

When n is large, the Poisson distribution (1.6) approaches to the Gaussian distribution:

$$P(n) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(n-\nu)^2}{2\sigma^2}} \tag{1.17}$$

where ν is the mean value and σ is the standard deviation. We will find $\sigma = \sqrt{\nu}$ in this case.

Proof If $n >> 1$, using the Sterling formula, $n! \approx \sqrt{2\pi n} \cdot e^{-n} n^n n$, and $\sqrt{2\pi n} \approx \sqrt{2\pi\nu}$, the Poisson distribution (1.16) becomes

$$P(n) \approx \frac{1}{\sqrt{2\pi\nu}} \left(\frac{\nu}{n}\right)^{n+\frac{1}{2}} e^{n-\nu} = \frac{1}{\sqrt{2\pi\nu}} \frac{e^{\Delta}}{\left(1+\frac{\Delta}{\nu}\right)^{\nu+\Delta+(1/2)}}, \text{ where } \Delta = n - \nu. \tag{1.18}$$

Because

$$\ln\left(1+\frac{\Delta}{\nu}\right)^{\nu+\Delta+(1/2)} = \left(\nu+\Delta+\frac{1}{2}\right)\ln\left(1+\frac{\Delta}{\nu}\right) = \left[\left(\frac{\Delta}{\nu}\right) - \frac{1}{2}\left(\frac{\Delta}{\nu}\right)^2\right]$$
$$= \Delta + \frac{\Delta^2}{2\nu} + O(\Delta^3),$$

we obtain $\left(1+\frac{\Delta}{\nu}\right)^{\nu+\Delta+(1/2)} \approx e^{\Delta+(\Delta^2/2\nu)}$, and therefore,

$$P(n) = \frac{1}{\sqrt{2\pi\nu}} \frac{e^{\Delta}}{\left(1+\frac{\Delta}{\nu}\right)^{\nu+\Delta+(1/2)}} = \frac{1}{\sqrt{2\pi\nu}} e^{-\Delta^2/2\nu} = \frac{1}{\sqrt{2\pi\nu}} e^{-(n-\nu)^2/2\nu}. \tag{1.19}$$

■

This is the normalized Gaussian distribution with the mean ν, and the standard deviation $\sigma = \sqrt{\nu}$.

Notice that the numerical difference between the Poisson and the Gaussian distributions of the same n is actually subtle when the average value is $\nu \geq 10$ or so. Figure 1.4 shows these probability distributions when $\nu = 10$ where the curve is the Gaussian distribution whereas the orange curve is the Poisson distribution.

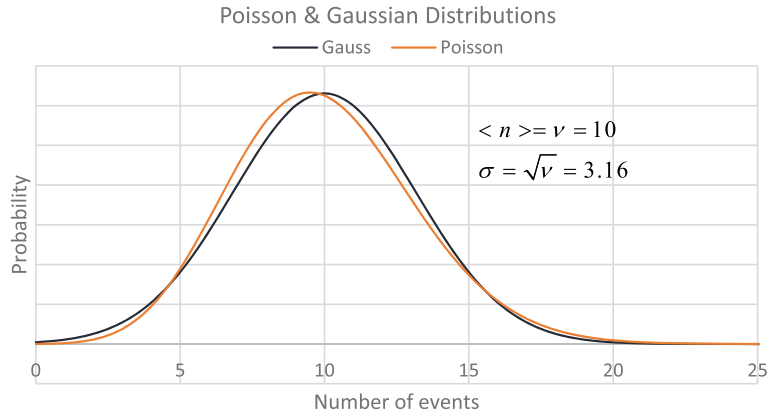


Fig. 1.4 Poisson and Gaussian distributions of mean value = 10

1.3.2 Binominal to Gaussian

Since the Poisson distribution function can be derived from the binominal function, and the Poisson distribution can be approximated by the Gaussian distribution, the binomial distribution (1.4) should also approach to be Gaussian as $n \to \infty$ (and $pn \to \infty$). Using the Sterling formula again, we obtain

$$\begin{aligned} P(n) &= \frac{N!}{n!(N-n)!} p^n q^{N-n} \\ &= \frac{1}{\sqrt{2\pi N}} \left(\frac{n}{N}\right)^{-n-(1/2)} \left(\frac{N-n}{N}\right)^{-M+n-(1/2)} p^n (1-p)^{N-m} \\ &= \frac{1}{\sqrt{2\pi N}} \exp\left[-\left(n+\frac{1}{2}\right)\ln\left(\frac{n}{N}\right) - \left(N-m+\frac{1}{2}\right)\ln\left(\frac{N-m}{m}\right)\right. \\ &\left. +n\ln n + (N-n)\ln(1-p)\right]. \end{aligned}$$

By setting $n = Np + \xi$ where $\xi << Np$, and keeping the dominant terms, we obtain

$$P(n) = \frac{1}{\sqrt{2\pi N}} \frac{1}{\sqrt{p(1-p)}} \exp\left[-\frac{1}{2}\frac{\xi^2}{Np(1-p)}\right] = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(n-<n>)^2}{2\sigma^2}} \tag{1.20}$$

where $\sigma = \sqrt{Np(1-p)}$ and $\xi = n - Np = n - < n >$.

Equations (1.19) and (1.20) are consequences of the central limit theorem, which states *any* probability distribution approaches to the Gaussian distribution. According to the central limit theorem, even a uniform probability distribution function will approach to the Gaussian! We can generate a set of uniform random numbers using the

RND() function, and investigate if the uniform distribution function approaches to the Gaussian distribution.

1.4 White Noise—Uniform Distribution to Gaussian Distribution

Figure 1.5 lists a small VBA code to generate random numbers that follow the Gaussian distribution, called inverse Gaussian random numbers. Twelve uniform random numbers generated between −0.5 and +0.5 using the RND() function are added to make one random number. The subtraction of 0.5 from each random number is to make the desired range of random numbers. The distribution of many of such random numbers will form the Gaussian distribution with the mean 0 and the standard deviation 1.

Alternatively, EXCEL's built-in function, NORMINV(RND(),0,1), which generates random numbers having the mean 0 and the standard deviation 1, can be called in a VBA program. Figure 1.6 shows a VBA code that generates 2,000 inverse Gaussian random numbers.

Figurer 1.7 shows the output from each of the inverse Gaussian random number. Both can generate Gaussian random numbers to the same degree of precision. In both charts, the broken orange lines show the normalized Gaussian distribution of the mean and the standard deviation calculated from the data of each method.

Remark: The NORMINV() function was introduced in EXCEL 2010. The NORMINV function was available in earlier excel version. It is still available in excel 2016, and VBA of EXCEL 2019 (and EXCEL 365) can only call the NORMINV() function. The calling the

```
Sub InvGauss()
Cells(1,1)="Inverse Gaussian random numbers from uniform distribution using RND()"
For i=1 to 2000
   X=0
      For j = 1 To 12
          X = X + (Rnd()-0.5)
      Next j
    Cells(2+i, 1)=i: Cells(2+i, 3)=X
Next i
End Sub
```

Fig. 1.5 VBA Code for generating Gaussian random numbers from RND()-function

```
Sub GaussDist()
Cells(1,1)="Gaussian random numbers from EXCEL's NORMINV()-function"
    For k = 1 To 2000
        Cells(k, 1) = k
        X = Application.WorksheetFunction.NormInv(Rnd(), 0, 1)
        Cells(k, 2) = X
    Next k
End Sub
```

Fig. 1.6 VBA Code for generating inverse Gaussian random numbers from NORMINV() function

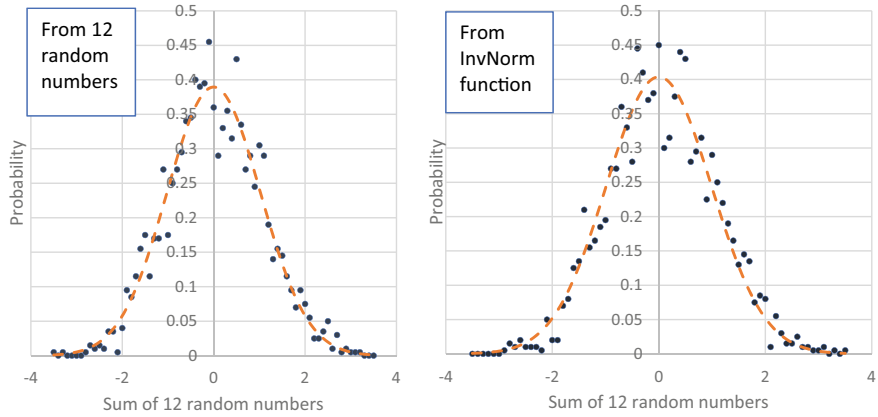


Fig. 1.7 Inverse Gaussian random numbers generated by two different methods

`NORMINV()` function from a VBA code takes a much longer computation time, and a run time error may occur when it is called from a numerically intensive program.

1.5 Central Limit Theorem

Why can the sum of 12 uniform random numbers represent the Gaussian distribution with the mean 0 and the standard deviation 1? What will happened if we add 24 uniform random numbers? We can find the answers by studying the central limit theorem.

For proving the central limit theorem, first, we define the moment and the characteristic function of a probability distribution, $p(x)$. We also use the Fourier transform [4]. The mean value of the mth-power of a random variable, x, of a probability distribution $p(x)$ is called the *mth*-moment:

$$\langle x^m \rangle = \int x^m p(x) dx \tag{1.21}$$

The first moment is the mean (average), $\langle x \rangle$, and the second moment appears in the standard deviation $\sigma = \sqrt{\langle x^2 \rangle - \langle x \rangle^2}$. The characteristic function, $\phi(k)$, which is the Fourier transform of the probability distribution function, can generate the moments.

$$\begin{aligned} \phi(k) &= \int_{-\infty}^{+\infty} dk e^{ikx} p(x) = \int_{-\infty}^{+\infty} dk p(x) \left[1 + ikx - \frac{1}{2!} k^2 x^2 - \frac{i}{3!} k^3 x^3 + \cdots \right] \\ &= 1 + ik\langle x \rangle - \frac{k^2}{2!} \langle x^2 \rangle + \cdots \end{aligned} \tag{1.22}$$

Notice that the characteristic function for a Gaussian distribution has only the first and the second moments:

$$\phi_{Gauss}(k) = \int_{-\infty}^{+\infty} dx e^{ikx} \left\{ \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x - \langle x \rangle)^2}{2\sigma^2} \right] \right\} = e^{ik\langle x \rangle} \exp\left(-\frac{1}{2} k^2 \sigma^2 \right). \quad (1.23)$$

Now, define the arithmetic average the independent random variables of a probability distribution function, $p(x)$:

$$\xi = \frac{1}{n} \sum_{i=1}^{n} x_i. \quad (1.24)$$

The new variable ξ is also a random variable. The central limit theorem states that the probability distribution function, $P(\xi)$, will be Gaussian with a sufficiently large n.

Proof Instead of $P(\xi)$, we can use another probability distribution function, $Q(\xi - \langle x \rangle)$, where the random variable is shifted by its mean value, and thus $Q(\xi - \langle x \rangle) = P(\xi)$. Because the random variables $\{x_1, x_2, \ldots, x_n\}$ are independent, the probability $P(\xi)$ should be given by the products of the probability of each random variable, $p(x_i)$, i.e., $P(\xi) = p(x_1)p(x_2)\cdots p(x_n)$. Similar to Eq. (1.22), the characteristic function of $Q(\xi - \langle x \rangle)$ is given by

$$\begin{aligned}
\Phi(k) &= \int_{-\infty}^{+\infty} d\xi e^{ik(\xi - \overline{x})} Q(\xi - \langle x \rangle) \\
&= \int_{-\infty}^{+\infty} \exp\left[\frac{ik}{n} \{ (x_1 - \langle x \rangle) + (x_2 - \langle x \rangle) + \cdots + (x_n - \langle x \rangle) \} \right] p(x_1) dx_1 p(x_2) dx_2 \cdots p(x_n) dx_n \\
&= \int_{-\infty}^{+\infty} \exp\left[\frac{ik}{n} (x_1 - \langle x \rangle) \right] p(x_1) dx_1 \int_{-\infty}^{+\infty} \exp\left[\frac{ik}{n} (x_2 - \langle x \rangle) \right] p(x_2) dx_2 \cdots \int_{-\infty}^{+\infty} \exp\left[\frac{ik}{n} (x_n - \langle x \rangle) \right] p(x_n) dx_n \\
&= \left[\phi\left(\frac{k}{n} \right) \right]^n = \left[1 - \frac{1}{2} k^2 \sigma^2 + \cdots \right] \to e^{-\frac{1}{2} k^2 \sigma^2} \; as \; n \to \infty.
\end{aligned} \quad (1.25)$$

Taking the inverse Fourier transform of the characteristic function, $\Phi(k)$, we obtain $Q(\xi - \overline{x})$:

$$Q(\xi - \overline{x}) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \Xi(k) e^{-ikc} dk = \frac{1}{\sqrt{2\pi}(\sigma/\sqrt{n})} \exp\left[-\frac{1}{2} \frac{(\xi - \langle x \rangle)^2}{(\sigma/\sqrt{n})^2} \right] \quad (1.26)$$

Equation (1.26) is a Gaussian distribution of the mean value $< x >$ and the standard deviation $\sigma' = \sigma/\sqrt{n}$. Therefore, any probability distribution function will approach to Gaussian with a large number of the random variables of the probability distribution function. ■

Table. 1.1 Gaussian distribution with different σ from uniform random numbers

n	$\sigma = \sqrt{\frac{n}{3}(0.25)}$	σ from EXCEL's STDEV.S-function
12	1.00	1.0240
24	1.41	1.4071
36	1.73	1.7174
48	2.00	2.0168

Example In Sect. 1.4, we create a Gaussian distribution of the mean 0 and the standard deviation 1 by summing 12 uniform random numbers. From the central limit theorem, we now understand why we get the specific distribution function.

A random number of uniform distribution between -0.5 and $+0.5$ has

$$\langle x \rangle = 0 \text{ and } \langle x^2 \rangle = \int_{-0.5}^{+0.5} x^2 dx = \frac{1}{3}(0.25).$$

According to the central limit theorem, the sum of the n-terms of the uniform random numbers $\{x_1, x_2, \ldots, x_n\}$, $\xi = (x_1 + x_2 + \cdots + x_n)/n$, has

$$\langle \xi \rangle = 0 \text{ and } \sigma = \sqrt{\langle x^2 \rangle - \langle x \rangle^2} = \sqrt{\frac{n}{3}(0.25)}. \tag{1.27}$$

Table 1.1 shows the σ-values from Eq. (1.27) for several different n's and the computed σ-values from the average of 2,000 ξ's. Computing the sum of 12 terms generates a useful Gaussian distribution of the mean 0 and the standard deviation 1. This is the reason why $N = 12$ is the magic number in the binomial and uniform distributions.

References

1. Reif F (2009) Fundamentals of statistical and thermal physics. Waveland Press, Long Grove, IL
2. Ling SI, Sanny J, Moebs W (2012) University physics, vol 3. Open Stax: Rice University. https://openstax.org/details/books/university-physics-volume-3)
3. Buczy BM (2009) Poisson distribution of radioactive decay. https://www.semanticscholar.org/paper/Poisson-Distribution-of-Radioactive-Decay-Buczyk/a60e5ede525d744017b6a295915b40c111aa9e10
4. Cho S (2018) Fourier transform and its applications using microsoft EXCEL®. A Primer IOP Concise Physics. Morgan & Claypool, San Rafael, CA

Idea of Monte Carlo Simulations

2

Science theories must be verified with experiments. However, there are many situations where actual experiments are not feasible. Sometimes, even if we can establish an equation to describe a theoretical model, solving the equation would be impossible or extremely challenging. To circumvent this, we may apply an approximation and/or conduct numerical calculations of the model, which will be compared with experimental data. We will also attempt to perform simulations of a theoretical model to predict its outcomes. One of the efficient computational methods is called the Monte Carlo simulation [1]. It applies "random samplings" to obtain statistically significant results rather than attempting to perform whole computations for computational speed and efficiency. In this charter, we describe several well-known mathematical examples of the Monte Carlo method to study its basic idea.

2.1 Calculation of π

This is an example that would be the most-cited example of the Montero Carlo simulation in articles and books [2]. Figure 2.1 depicts a square of unit length and a quarter of a circle of the radius of the unit length. The area ratio of the quarter circle to the square is π/4. In the Monte Carlo method calculates the ratio by randomly generating a point (x, y) in the unit square ($0 \le x \le 1$ and $0 \le y \le 1$)), and then determines if the point is within the quarter circle. After repeating this step many times, and the calculate compute the ratio of the number of points within the quarter circle, N_c, to the total number of points generated, N, should be close to $\pi/4 \approx 0.7854$:

$$\frac{N_c}{N} \to \frac{\pi}{4} \text{ as } N \to \infty \tag{2.1}$$

S. Cho, *Monte Carlo Simulations Using Microsoft EXCEL®*,
Synthesis Lectures on Mathematics & Statistics,
https://doi.org/10.1007/978-3-031-33886-1_2

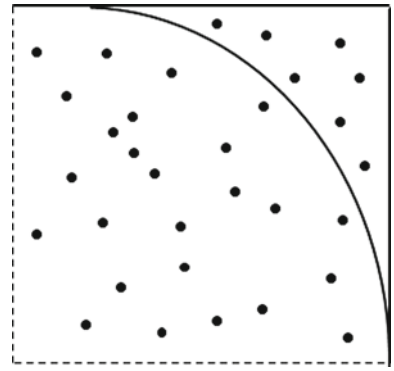


Fig. 2.1 Area-ratio for computing π

In an illustrative Fig. 2.1, there are numbers of randomly distributed points in a square of unit length where $N = 32$ and $N_c = 25$ and $N_c/N = 0.78125$.

The computational procedure, which is called the hit-or-miss method, takes the following steps:

(1). Generate of a point (x, y) of two random numbers of $0 \le x \le 1$ and $0 \le y \le 1$;
(2). Check if the point within the circles: if $x^2 + y^2 \le 1$, then add it to N_c, and calculate the ratio N_c/N; and
(3). Repeat the above steps N-times.

With a sufficiently large N, the computed ratio should approach to $\pi/4$. Figure 2.2 lists the VBA code we created. Here there are 50 sets of 1,000 random samplings in this code to observe the transition of the ratio as the number of random samplings increase. The code computes the result of the hit-and-error of 1,000 random samplings to accumulate the number of points inside the quarter circle and calculates the ratio. This cycle is repeated 50 times.

Figure 2.3 is a chart of the computed result, using EXCEL's option of [`Scatter with Smooth Lines and Markers`] of the [`Charts`] menu. The dots of the chart show the computed accumulated ratio at 1,000, 2,000, …, 50,000 samplings. The ratio approaches to the target value although the convergence is slow.

2.2 Calculation of Definite Integrals

Definite integral, $\int_a^b f(x)dx$, can be computed in a similar fashion described in Sect. 2.1 because the integral can be interpreted as the area under the curve $f(x)$ in the interval of $a \le x \le b$. For simplicity, we assume that the function $f(x) \ge 0$ and monotonous. By taking randomly sampled points (x, y) in a rectangular area of $(b - a) \cdot \text{Max } f(x)$ where we estimate the area under the curve $f(x)$.

```
Sub MonteCarloPai()
Cells(1, 2) = "Monte Carlo Simulation of Pai-calculation"  'Title of this code.
   Cells(2, 1) = "N"
   Cells(2, 2) = "Pi"
       N = 0              'Initialize the total # of data points
       Hit = 0            'Initialize # of hits within the circle of radius 1.
       Ratio = 0          'Ratio=Hit/N.
       Pi = 0             'Initialize Pi.
       Samplings=1000     '# of random samplings.
       Nset =50           '# of sets of 1000 random samplings.
'Display the result every at 1000 samplings.
'Repeat a set of 1000 random samplings 50 times, totally 1000x50=50000 times.
   For i = 1 To Nset
          For j = 1 To samplings
              fx = Rnd()
              fy = Rnd()
              Z = fx ^ 2 + fy ^ 2
                 If Z <= 1 Then
                    Hit = Hit + 1
                 Else
                    GoTo Skip
                 End If
Skip:       Next j
      N = N + samplings  'Accumulate total number of randomly generated points.
      Ratio = Hit / N    'Ratio of quarter circle to square.
      Pai = 4 * Ratio
            Cells(i + 2, 2) = Pi
            Cells(i + 2, 1) = N
   Next i
End Sub
```

Fig. 2.2 VBA code for Monte Carlo (hit-ot-miss) calculation of π

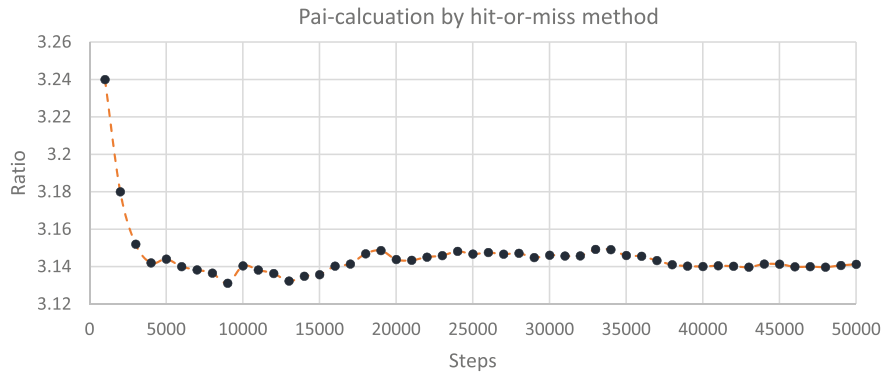


Fig. 2.3 Transition of area ratio while performing hit-or-miss method

Let's calculate

$$I = \int_0^1 \frac{dx}{1+x^2} = \frac{\pi}{4} = 0.785398\ldots \tag{2.2}$$

The integrant is monotonically decreasing in $0 \leq x \leq 1$. Figures 2.4 and 2.5 are the VBA code for this calculation and the transition of the computed value with more samplings.

```
Sub Integral()
Cells(1,1)="Integral of 1/(1+x^2) for [0,1] by counting random points"
N = 0
Hit = 0
     For i = 1 To 500
         For j = 1 To 100
             fx = Rnd()
             fy = Rnd()
                 F = 1 / (1 + fx * fx)
                 If fy <= F Then
                     Hit = Hit + 1
                 Else
                     GoTo Skip
                 End If
Skip:     Next j
       N = N + 100          'Accumulated total number of randomly generated points.
       Ratio = Hit / N      'Ratio of points under the curve to the total points.
             Cells(i + 2, 1) = N
             Cells(i + 2, 2) = Ratio
       Next i
End Sub
```

Fig. 2.4 VBA code for computing integral by hit-or-miss method

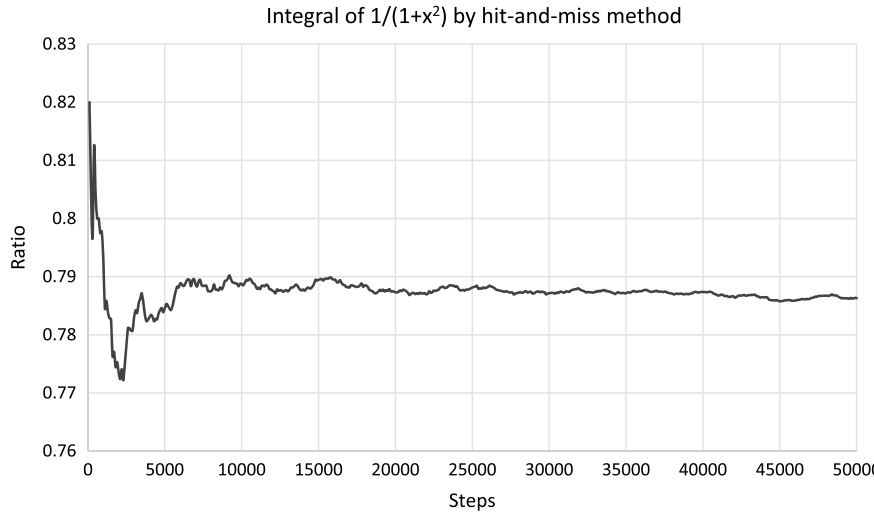


Fig. 2.5 Transition of integral value while performing hit-or-miss method

The program checks if a point (x, y) randomly generated within a unit square is below the curve of $1/(1 + x^2)$, and then compute the ratio of the points below the curve to the total points. The code computes the result of the hit-and-error at every 100 samplings, repeating 500 times. The asymptotic value approaches to the theoretical value.

The Monte Carlo simulation provides another method for computing the integral. Applying the mean value theorem for integrals,

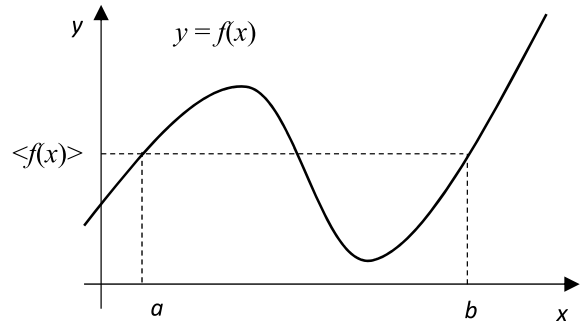


Fig. 2.6 Mean value theorem

$$\int_a^b f(x)dx = (b-a) < f(x) >, \tag{2.3}$$

we can also compute the definite integral. If we can compute $< f(x) >$, the integral (2.5) can be evaluated. How can we do that efficiently? The key idea is to perform the random sampling to find the mean value, $< f(x) >$. That is, as illustrated in Fig. 2.4, applying the "random sampling" of N points, $< f(x) >$ can be evaluated by considering its values at N abscissae, $\{x_i\}$, chosen at random with equal probability anywhere within the interval $[a, b]$. Then $< f(x) >$ will be computed by the following equation:

$$< f(x) >= \frac{1}{N}\sum_{i=1}^{N} f(x_i) \text{ where } a \le x_i \le b. \tag{2.4}$$

Figures 2.5 list the VBA code that outputs the result shown in Fig. 2.6. The output approaches to 0.786. The code computes the result of the hit-and-error at every 100 samplings, repeating 500 times, and calculates the average at 100, 200, …, 50,000 samplings.

There are several techniques to reduce the variance and improving the efficiency and the integral. However, they are beyond the scope of this book. If a reader is interested in the nature of computational improvement, refer to books on computational physics [2].

```
Sub Intergral2()
Cells(1,1)="Integral of 1/(1+x^2) for [0,1] by calculating <f(x)>"
N = 0
SumF = 0
    For i = 1 To 500
        For j = 1 To 100
        fx = Rnd()
            F = 1 / (1 + fx * fx)      'Value of integrant at a random point.
            SumF = SumF + F            'Sum of the values.
        Next j
             N = N + 100
             AveF = SumF / N           'Arithmetic average of the sum.
                  Cells(i + 2, 1) = N
                  Cells(i + 2, 2) = AveF
     Next i
End Sub
```

Fig. 2.7 VBA code for computing integral based on random sampling

2.3 Radioactive Decay

As we described in Chap. 1, the radioactive decay is given by $dN/dt = -\lambda N$, where λ is the decay constant. We should be able to simulate the exponential decay $N(t)$ using a predetermined decay constant. The Monte Carlo simulation determines an event of decayed/undecayed by generating a random number:

(1) if the random number is smaller than or equal to the pre-determined λ-value, then it is decayed, and reduce the current number of undecayed nuclei N by 1;
(2) if the random number is larger than the λ-value, then it is undecayed, and keep the same N;
(3) repeat steps 1 and 2 to output $N(t)$ and calculate $ln[N(t)]]$ until the desired number of time-segments; and
(4) draw a graph of $\ln(N(t)) = \ln(N(0)) - \lambda t$, and find its equation. The slope value of the equation should be the pre-determined λ-value.

Figures 2.7 and 2.8 are the simulation program we created, and its outputs. The λ-value computed by the simulation is 0.0099, which is essentially the same as the pre-set value 0.01.

2.4 Random Walk

Random walk is a random process that traces a path of random steps known as a walker. The walker's displacement consists of successive random movements taking the pre-determined step width. Because random walk can be applied to various physics models we describe throughout this book, it is crucial to learn it. The Monte Carlo simulation can compute random walk processes easily and efficiently.

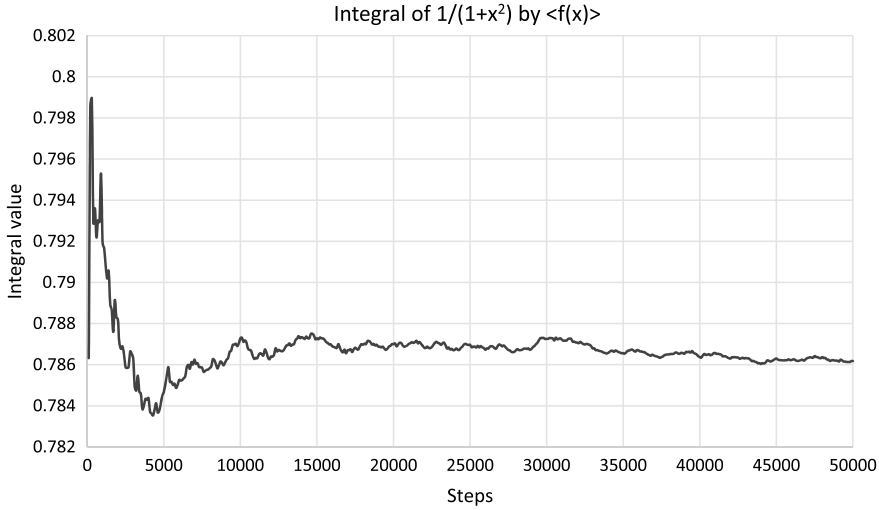


Fig. 2.8 Transition of integral values while performing random sampling

2.4.1 One-Dimensional Random Walk

Suppose a walker is initially at the origin, $x(0) = 0$, and moves along the straight line where each step randomly takes $\pm L$. Assume the walker has a probability p of a step to the right and a probability of $q = 1 - p$ of a step to the left. This is equivalent to the binomial distribution. After N-steps, the distance from the initial position is given by

$$x(N) = \sum_{i=1}^{N} s_i \text{ where } s_i = \pm L, \text{ and thus, } x^2(N) = \left(\sum_{i=1}^{N} s_i\right)^2 = \sum_{i=1}^{N} s_i^2 + \sum_{i \neq j}^{N} s_i s_j. \tag{2.5}$$

If $p = q = 1/2$, the average $x(N)$, $< x(N) >$, will be zero: $< x(N) >= \frac{1}{N} \sum_{i=1}^{N} s_i = 0$, and the product, $s_i s_j$ equals $+ L^2$ and $- L^2$ when $i \neq j$ with equal probability, and thus

$$\sum_{i \neq j}^{N} \langle s_i s_j \rangle = 0, \text{ and } < x^2(N) >= \sum_{i=1}^{N} < s_i^2 >= NL^2. \tag{2.6}$$

Defining the time interval for each step to be Δt, the time after N steps is given by $t = N\Delta t$, and

$$< x^2(N) >= NL^2 = \left(\frac{L^2}{\Delta t}\right)(N\Delta t) = Dt \tag{2.7}$$

```
Sub NucDecay()
Cells(1,1)="Time dependence of radioactive nuclear decay"
  Dim StatusN(10000) As Single   'StatusN=0 is undecayed and StatusN=1 is decayed.
    NT = 10000                   'Number of nuclei at t=0: N(0).
    Lamda = 0.01                 'Probability of nuclear decay.
'Initial condition (0 means not decayed):
     For i = 1 To NT
        StatusN(i) = 0            'All are undecayed.
     Next i
'# of undecayed nuclei:
    Undecayed = NT
    Cells(2, 2) = "time" : Cells(3, 2) = 0
    Cells(2, 3) = "N/N0" : Cells(3, 3) = 1
    Cells(2, 4) = "Ln(N)": Cells(3, 4) = Log(1)
       Timeseg = 100
          For k = 1 To Timeseg
              For i = 1 To NT
                   If StatusN(i) = 1 Then GoTo Skip
                   If Rnd() <= Lamda Then StatusN(i) = 1
                   If StatusN(i) = 1 Then Undecayed = Undecayed - 1
Skip:         Next i
                   Ratio = Undecayed / NT
                   Cells(k + 3, 2) = k
                   Cells(k + 3, 3) = Ratio
                   Cells(k + 3, 4) = Log(Ratio)
          Next k
End Sub
```

Fig. 2.9 VBA code for simulating radioactive decay

where $t = N\Delta$t, and $D = L^2/\Delta$t. Equation (2.7) is called Einstein's relation that appears in his theory of Brownian movement [3]. Equation (2.7) states that the mean square distance, $< x^2(N) >$, from the starting point ($x = 0$) at N steps is proportional to time to take N steps.

Figures 2.9 shows a VBA code that computes the positions of 100 independent walkers at each Monte Carlo step. In this program, $D = 1$ because we set $L = 1$ and $\Delta t = 1$. The step width is 1, and each walker moves ± 1, depending on the random number generated by the `RND()` function. The program also computes the arithmetic average of the positions of the 100 walkers at each step.

Figure 2.10 shows the positions of four walkers selected from the 100 walkers at each step. Out of the four walkers, three of them moves similarly while one walker takes quite different steps. This can be possible in a stochastic process. Figure 2.13 shows the arithmetic average of x^2, $< x^2 >$, at each step, and the linear dependence on the number of steps is clearly observed. The computed slope value is $D = 1.0096$. The computed D-value is each simulation runs but consistent with $D = L^2/\Delta$t $= 1$ (Figs. 2.11, 2.12).

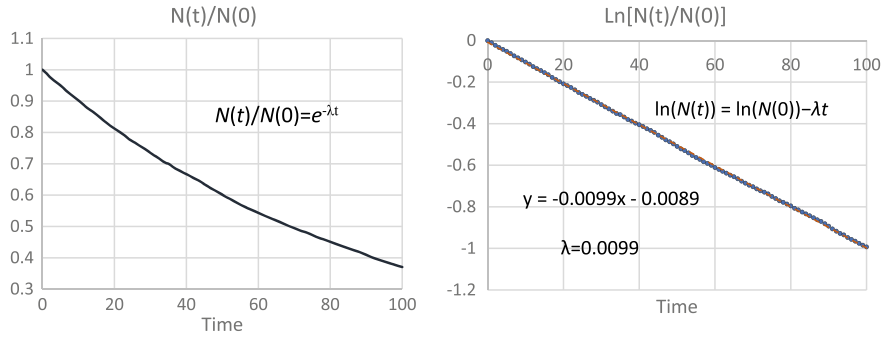


Fig. 2.10 Time dependence of number of remaining radioactive nuclei

2.4.2 Two-Dimensional Random Walk

Two-dimensional random walk on a square lattice is essentially the same as two one-dimensional random walks in two different directions orthogonal each other. If we consider the random walk, using the one-dimensional case given by Eq. (2.5) for the x- and the y- direction independently with the same step width, the distance to the position $(x(N), y(N))$ at the N steps from the origin is given by

$$< r^2(N) >=< x^2(N) + y^2(N) >= 2NL^2 = 2Dt. \tag{2.8}$$

The random walk on a square lattice is independent random walks in the two orthogonal directions with the same step size (± 1) taken in the one-dimensional random walk. The VBA code is listed in Appendix A2.1, and Fig. 2.14 shows the output from the code. The slope value indicates $< r^2 > = dDt$ with $d = 2$, and $D = 1$.

If we consider a random walk in arbitrary two-dimensional direction with a randomly changing angle θ to the x-axis, the position $(x(n), y(n))$ at n-steps is given by the set of two equations:

$$\begin{cases} x(n) = x(n-1) + L\cos\theta_n = L\sum_{i=1}^{n}\cos\theta_i \\ y(n) = y(n-1) + L\sin\theta_n = L\sum_{i=1}^{n}\sin\theta_i \end{cases} \tag{2.9}$$

where the angle θ_i ($i = 1, 2, \ldots, n$), randomly varies between 0 and 2π. The distance after N-steps is

$$< r^2(N) >=< x(N)^2 + y(N)^2 >= L^2\left[\langle\left(\sum_{i=1}^{N}\cos\theta_i\right)^2\rangle + \langle\left(\sum_{i=1}^{N}\sin\theta_i\right)^2\rangle\right] = \left(\frac{L^2}{\Delta t}\right)t \tag{2.10}$$

```
Sub RandomWalk()
Cells(1,1)="1-dimensional random walk with step width = +/-1"
'NP = # of random walkers.
'Nsteps= # of steps taken by each walker.
  Dim X(100) As Double       'Up to 100 walkers.
  Dim X2ave(1000) As Double  'Up to Nsteps=1000 steps.
    NP = 100                 '# of walkers.
    Nsteps = 1000            '1000 steps each of which takes 10 random walks.
          Cells(2, 1) = "Step"
          Cells(2, 2) = "<x^2>"
          Cells(1, 10) = "Walkers"
'Create a table: Column= # of walkers & Row = positions are every 10 random walks.
   For k = 1 To NP
     Cells(2, 3 + k) = k
   Next k
'Stating at the origin:
   For k = 1 To NP
      X(k) = 0
      Cells(3, 3 + k) = X(k)
   Next k
      Cells(3, 1) = 0               'Initial time.
      Cells(3, 2) = X2ave(1)        'Initial <x^2>.
'Each walker takes 10 random walks at each of 1000 steps.
'Write the positions at every 10 random walks.
'The total number of random walks is thus 10*Nsteps.
For j = 1 To Nsteps
   X2ave(j) = 0
     For k = 1 To NP
       For m = 1 To 10             'Random walk 10 times.
         If Rnd() < 0.5 Then
            X(k) = X(k) + 1
         Else
            X(k) = X(k) - 1
         End If
       Next m
       Cells(j + 3, 3 + k) = X(k)      'Position of the k-th walker at its 10th step.
       X2ave(j) = X2ave(j) + X(k) ^ 2 'Sum of x^2 of the k-th walker.
     Next k
Next j
'Write step numbers = 10N steps:
   For j = 1 To Nsteps
      Cells(j + 3, 1) = j * 10
   Next j
'Calculate <x^2>:
   For j = 1 To Nsteps
      X2ave(j) = X2ave(j) / NP
      Cells(j + 3, 2) = X2ave(j)
   Next j
End Sub
```

Fig. 2.11 VBA code of one-dimensional random walk

where $t = N\Delta t$. Figures 2.15 and 2.16 show a trace of the random walk of two-dimensional space and the mean square distance from the origin obtained by the VBA code listed in Fig. 2.17. In this simulation, $L = 1$ and $\Delta t = 1$ and ths $D = L^2/\Delta t = 1$. The simulation outputs $D = 0.9987$. The "random-angle" walk has only a single random variable, θ with no cross-term across the x and y coordinates. Thus, this, it is essentially one-dimensional random walk. We describe the Brownian motion more in Chap. 3.

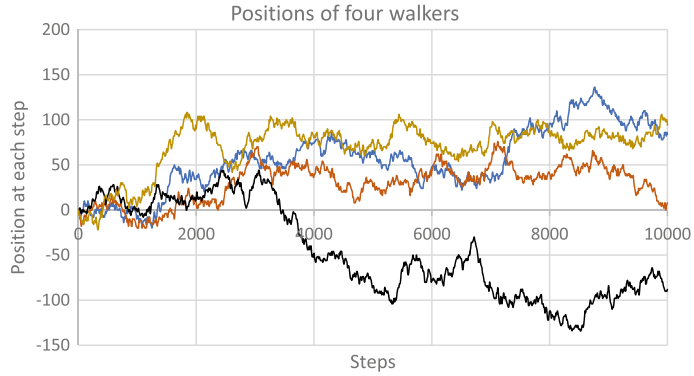


Fig. 2.12 Positions of four independent walkers at each step

Fig. 2.13 Time dependence of mean square distance $< x^2 >$

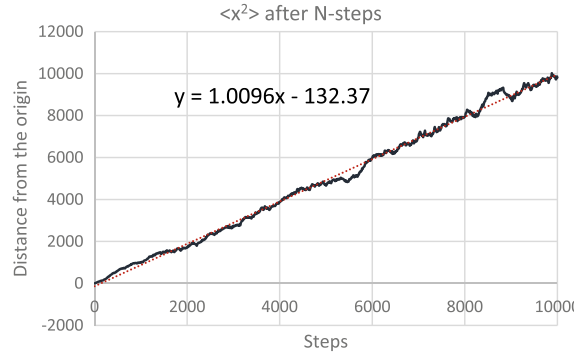
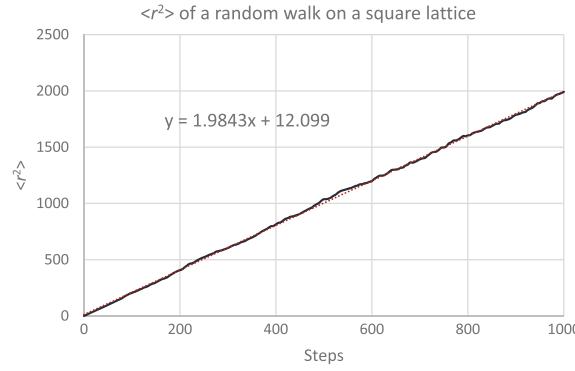


Fig. 2.14 Two-dimensional random walk on square lattice



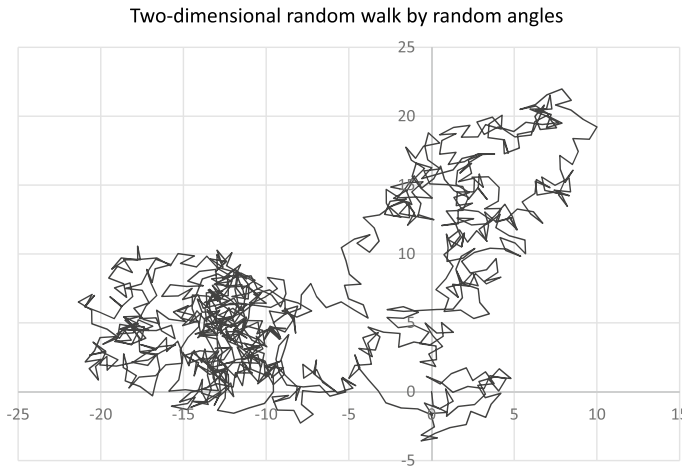


Fig. 2.15 Two-dimensional random walk pattern taking random angles

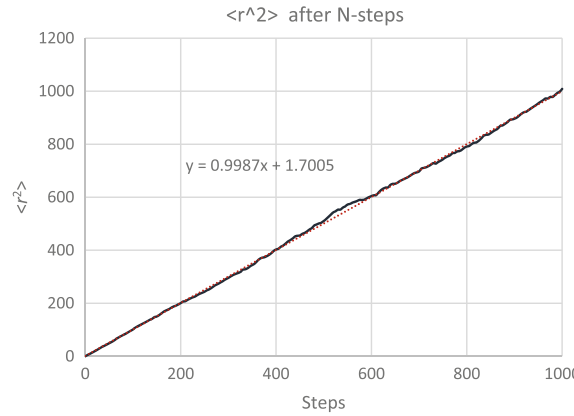


Fig. 2.16 Mean square distance $< r^2 >$ from the origin after N-steps

2.5 Percolation

Suppose a large porous rock is submerged under water for a long time, will the water ever reach the center of the stone? The original percolation problem was proposed by Broadbent and Hammersley [4]. A similar problem is that at what concentration of the conductive particles in a medium placed between the conducting plates conducts electricity. Figure 2.18 illustrates schematic diagrams that model the electrical conductivity between two plates and randomly placed metal particles between the plates. In these diagrams, the metal particles are replaced with cells formed in a square lattice. This model is called a bond percolation. In this model, if two black cells are adjacent horizontally

```
Sub RndAngle()
Cells(1,1)="Probability distribution function of two-dimensional random walk"
  Dim x(1000) As Double       'x-coordinate at each step per walker.
  Dim y(1000) As Double       'y-coordinate at each step per walker.
  Dim r2(1000) As Double      'r^2 are each step per walker.
  Dim r2ave(1000) As Double   '<r^2>
  Dim theta As Double
    step = 1                  'Sep width.
    Nsteps = 1000             'Total # of random steps.
    NP = 1000                 'Number of walkers.
       Cells(3, 1) = "Step width" : Cells(, 2) = step
       Cells(4, 1) = "Nsteps" : Cells(4, 2) =  Nsteps
       Cells(5, 1) = "# Walkers" : Cells(5, 2) = NP    '# of walkers.
  For i = 1 To NP
     x(i) = 0
     y(i) = 0
  Next i
     For j = 1 To Nsteps
        Cells(9 + j, 4) = j
     Next j
Randomize
'Writing (x,y) of a single walker:
    Cells(8, 4) = "Steps"
    Cells(8, 5) = "x(1)"
    Cells(8, 6) = "y(1)"
    Cells(8, 8) = "Steps"
    Cells(8, 9) = "<r^2>"
 'Start at the origin:
    Cells(9, 4) = 0
    Cells(9, 5) = 0
    Cells(9, 6) = 0
    Cells(9, 8) = 0
    Cells(9, 9) = 0
  For j = 1 To Nsteps
     r2ave(j) = 0
       For i = 1 To NP
          r2(i) = 0
          theta = 2 * 3.141592654 * Rnd()   'Random angle between 0 and 2*pi
          x(i) = x(i) + step * Cos(theta)
          y(i) = y(i) + step * Sin(theta)
          r2(i) = x(i) ^ 2 + y(i) ^ 2       'Distance from the origin.
          r2ave(j) = r2ave(j) + r2(i)       'Sum of each walker's r^2 at each step.
                Cells(9 + j, 5) = x(1)
                Cells(9 + j, 6) = y(1)
        Next i
                r2ave(j) = r2ave(j) / NP     'Calculate average.
                   If (j Mod 5) = 0 Then     'Output average value at every 5 steps.
                      Cells(9 + j / 5, 8) = j
                      Cells(9 + j / 5, 9) = r2ave(j)
                   End If
  Next j
End Sub
```

Fig. 2.17 VBA code for two-dimensional random walk using random angles

and/or vertically, they are connected whereas if they are adjacent diagonally, they are not connected [5].

The medium has a certain threshold concentration (occupation rate) to establish a conductive condition. In other words, once the concentration exceeds the threshold, the medium becomes conductive. This change occurs suddenly and called a geometrical phase transition. Our percolation model is a bond percolation on the square lattice of 400×400 cells of an EXCEL's spread sheet, making some of the cells "occupied." We change the concentration of occupied cells, p, and observe how the occupied cells are connected from

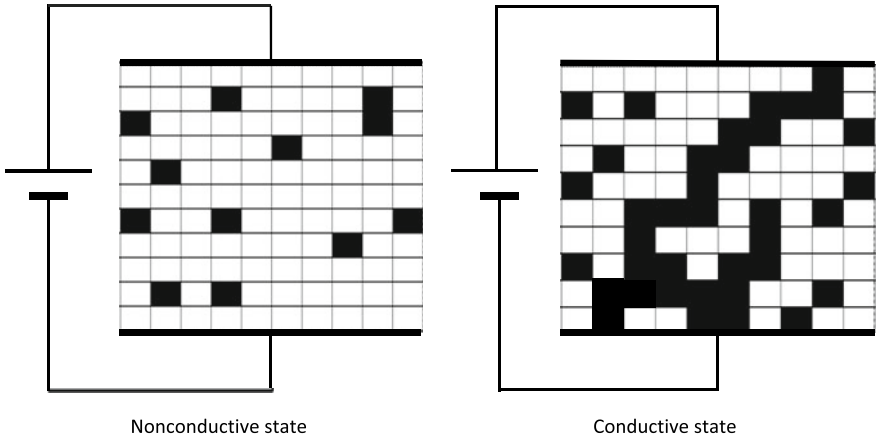


Fig. 2.18 Schematic diagram of percolation problem

the top (`Row 1`) to the bottom (`Row 100`) of the spread sheet. Figures 2.19 shows the VBA code created for this geometry.

We used a sub procedure (or subroutine) for repeatedly used codes. In the VBA code of Fig. 2.19, there is a sub procedure, [`RenumberClusters:`].

Figure 2.20 shows the probability of connective state computed from the VBA code. The probability abruptly increases once the concentration of occupied cells exceeds about 0.59. This is a geometrical phase transition. The theoretical critical concentration is known to be $p_c = 0.5928$, and we estimate $p_c = 0.593$. Notice that the 400×400 square lattice and the 40×40 square lattice appear to indicate the same p_c although their transition curves depend on the lattice size. More detailed computation indicates the probability of connection has the concentration dependence of $(p - p_c)^{\beta}$ where $\beta = 5/36$ near the critical concentration [6, 7].

Figure 2.21 shows two screen shots of the computed bond patterns on 200×200 lattice when $p = 0.50$ and 0.70. When $p = 0.50$, there are many clusters of variety of sizes, but they are not large ("long") enough to attain the end-to-end connection. When $p = 0.70$, there is at least one large cluster that attains the end-to-end connection.

Practical note: We limited the lattice size 200×200 for optimal display on screen and paper. Screen displays of 200×200 cells were reduced to 10%, and then took the screen shots. Furthermore, we reduced the horizontal scales of the screen shots so that the cells become square.

```
Sub Percolation()
Cells(1,1)="Percolation problem using a square lattice of [Nsize]x[Nsize] cells."
  Dim Site(401, 401) As Integer        '400x400 cells.
  Dim cluster(160801) As Long          'Total number of [Nsize+1]x[Nsize+1]cells.
     Nsize = 400 : Nsize1 = Nsize + 1
     Nstart = 30 : Nend = 80           'Concentration range is from Nstart to Nend.
     Total = 1000                      '# of trial runs.
For REP = Nstart To Nend
    Concentration = REP / 100          'Occupation concentration.
       Cells(Nsize + 5, 2) = "p"
       Cells(Nsize + 5, 3) = "Occp.Freq"
       Cells(Nsize + 5, 5) = "No. of black sites connected"
   Freq = 0                            '# of connected events.
                                       '[Freq/Total] will be probability of connections.
For Im = 1 To Total
     Black = 0
     num = 1                           'Label clusters with sequential number (num).
'Initialize all cells to be white (vacant):
   For i = 1 To Nsize1
     For j = 1 To Nsize1
        Site(i, j) = 0
        Cells(i, j).Interior.Color = RGB(255, 255, 255)  'White (vacant)
     Next j
   Next i
'Random placement of black sites as the occupied cells:
   For i = 2 To Nsize1
      For j = 2 To Nsize1
         If Rnd() < Concentration Then
             Site(i, j) = 1
             Cells(i, j).Interior.Color = RGB(0, 0, 0)    'Black (occupied)
             Black = Black + 1
          Else
             Goto Skip
          End If
Skip:  Next j
  Next i
         Cells(Nsize + 5, 8) = Black
'Label clusters by number:
   For i = 1 To Nsize1 ^ 2
      cluster(i) = i
   Next i
      For i = 2 To Nsize1
         For j = 2 To Nsize1
         'If the site(i,j) is white (vacant), skip labeling:
             If Site(i, j) = 0 Then GoTo Skip
         'If the site (i,j) is black (occupied), label cluster number:
            If Site(i, j - 1) = 0 And Site(i - 1, j) = 0 Then
                Site(i, j) = num  'Label cluster number to the isolated cell.
                num = num + 1     'Increase the cluster number for the next cluster.
            End If
            If Site(i, j - 1) > 0  And Site(i - 1, j) = 0 Then
                Site(i, j) = Site(i, j - 1)              'Vertically connected.
            End If
            If Site(i, j - 1) = 0 And Site(i - 1, j) > 0 Then
                 Site(i, j) = Site(i - 1, j)             'Horizontally connected.
            End If
                If Site(i, j - 1) > 0 And Site(i - 1, j) > 0 Then
         GoSub RenumberClusters  'Join two clusters with different numbers.
                End If
Skip: Next j
    Next i
'Check if the upper side and the lower side are connected.
    For i = 2 To Nsize1
        For j = 2 To Nsize1
            If cluster(Site(2, i)) = cluster(Site(Nsize1, j)) And Site(2, i) <> 0
And Site(Nsize1, j) <> 0 Then
               Freq = Freq + 1
               GoTo StopChecking
             End If
        Next j
   Next i
StopChecking: Next Im
      Cells(Nsize + 6 + REP - Nstart, 2) = Probability
      Cells(Nsize + 6 + REP - Nstart, 3) = Freq / Total
Next REP
Exit Sub
'-------------------------------------------------------------------------
RenumberClusters:
'Renumbering when two clusters with different numbers are joined.
      If cluster(Site(i, j - 1)) > cluster(Site(i - 1, j)) Then
          NA = cluster(Site(i, j - 1))
          NB = cluster(Site(i - 1, j))  'Take the smaller cluster number NB.
      Else
          NA = cluster(Site(i - 1, j))
          NB = cluster(Site(i, j - 1))
      End If
          Site(i, j) = NB
              For k = 1 To num - 1      'Replace all cluster numbers NA with NB.
                  If cluster(k) = NA Then cluster(k) = NB
              Next k
 Return
'-------------------------------------------------------------------------------
End Sub
```

Fig. 2.19 VBA code of bond percolation on square lattice

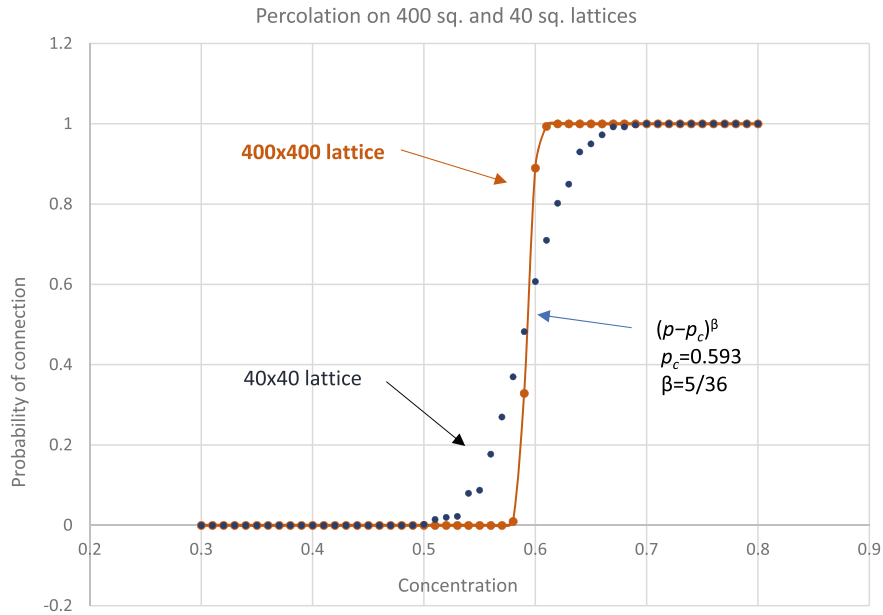


Fig. 2.20 Lattice size dependence on the critical concentration p_c

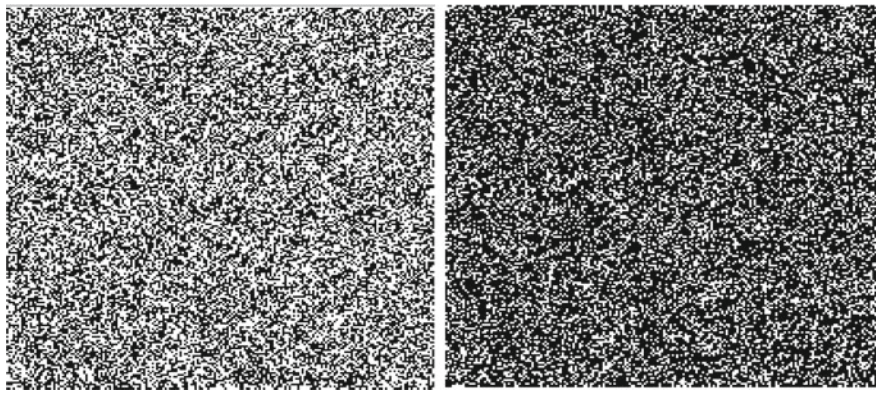


Fig. 2.21 Clusters at $p = 0.50$ (left) and 0.70 (right) on 200×200 square lattice

References

1. Metropolis N (1987) The beginning of the Monte Carlo method. Los Alamos Science (Special Issue dedicated to Stanislaw Ulam). https://library.lanl.gov/cgi-bin/getfile?00326866.pdf
2. Koonin SE (1987) Computational physics. Benjamin/Cummings, Menlo Park, CA
3. Einstein A (1998) Investigations on the theory of the brownian movement. Dover, New York, NY
4. Broadbent S, Hammersley J (1957) Percolation processes I. Crystals and mazes. Math Proc Cambridge Philos Soc 53(3):629–641. https://www.cambridge.org/core/journals/mathematical-proceedings-of-the-cambridge-philosophical-society/article/abs/percolation-processes/C00CC4943F48228F8AC8031092FE84EC)
5. Site Percolation—Wolfram MathWorld. https://mathworld.wolfram.com/SitePercolation.html
6. Sykes MF, Glen M, Gaunt DS (1974) The percolation probability for the site problem on the triangular lattice. J Phys A: Math Gen 7(9):L105–L108. https://iopscience.iop.org/article/10.1088/0305-4470/7/9/002)
7. Smirnov S, Werner W (2001) Critical exponents for two-dimensional percolation. Math Res Lett 8:729–744 (arxiv, Cornell University). https://arxiv.org/abs/math/0109120v2

Brownian Motion and Diffusion Equation 3

Brownian motion is random movement of particles such as pollen suspended in water [1]. The random movement of particles is caused by collisions with surrounding small particles. One of Einstein's monumental works is a study of Brownian motion [2].

In this chapter, we describe several approaches to model Brownian motion:

1. a particle colliding with the surrounding particles (1-dimension and 2-dimensions),
2. stochastic equations that lead to diffusion equations,
3. random walks to model diffusion processes, and
4. analytical solutions for diffusion equations.

We also compute numerical solutions of equations of particle diffusion and heat flow with given initial and boundary conditions. This approach is also applicable to Schrödinger equations, which will be discussed in the next chapter.

3.1 Motion of a Particle Driven by Collisions with Surrounding Particles

3.1.1 One-Dimensional Collision

The following description is inspired by the article by Ronlund [3]. Let's describe a one-dimensional elastic head-on collision between a target particle of mass M at velocity V and an incident particle of mass m at average velocity v. Using the momentum and the energy conservation laws, the velocities after a single collision are given by

S. Cho, *Monte Carlo Simulations Using Microsoft EXCEL®*,
Synthesis Lectures on Mathematics & Statistics,
https://doi.org/10.1007/978-3-031-33886-1_3

$$\begin{cases} v' = \frac{2MV+(m-M)v}{m+M} = \frac{2V+(\zeta-1)v}{1+\zeta} \\ V' = \frac{2mv+(M-m)V}{m+M} = \frac{2\zeta+(1-\zeta)V}{1+\zeta} \end{cases} \tag{3.1}$$

where $\zeta = m/M$.

Because the incident particles randomly collide with the target from either side, we can define $v = \pm v_0$. After collisions of N times, the position and velocity of the target particle are given by the following iteration formula:

$$x_N = x_{N-1} + V_N \Delta t \quad \text{where} \quad V_N = \frac{2\zeta + (1-\zeta)V_{N-1}}{1+\zeta} \tag{3.2}$$

where Δt is the collision time, i.e., the time interval between two successive collisions. In this collision process, we assume Δt is constant, and no simultaneous multiple collisions occur. Since we assume the velocities of the incident particles are $\pm v_0$, there are no secondary collisions of the surrounding particles.

Figures 3.1 and 3.2 show a VBA code, and the result of $< x^2 >$ as a function of the number of steps, N, where the number of steps, $N = 1$ to 5,000. The number of target particles is 1,000 to compute $< x^2 >$. Except the motion at the beginning, $< x^2 >$ is proportional to the number of collisions N or the time $t = N\Delta t$. The initial transit motion is very short in a realistic situation, and we will not observe it. Although selection of $\zeta = m/M$, Δt, and v_0 determines the linear coefficient, the result always shows the linear relationship as the consequence of one-dimensional random walk based on collisions. Hence, the time dependence of $< x^2 >$ is equivalent to that of the random walker shown in Fig. 2.13, and reveals the Einstein's relation in the one-dimensional random walk that we discussed in Sect. 2.4.1. In Fig. 3.4, the collision parameters are: $\zeta = 0.01$, $\Delta t = 0.01$, $v_0 = 10$, and $N_{\max}\Delta t = 50$.

3.1.2 Two-Dimensional Collision

A collision-based model on a plane should reflect more realistically to the motion of a pollen particle floating on water. When we apply two-dimensional elastic collision to the Brownian motion, the collisions are not necessarily head-on, and more detailed analysis of collisions is required.

Suppose, in the laboratory frame, the incident particles have mass m and changes the velocity $\vec{v}$ to $\vec{v}'$ due to a single collision whereas the target particle has mass M and changes the velocity $\vec{V}$ to $\vec{V}'$ due to the collision. The momentum and the energy conservations are given by:

$$\begin{cases} m\vec{v} + M\vec{V} = m\vec{v}' + M\vec{V}', \\ (1/2)mv^2 + (1/2)MV^2 = (1/2)mv'^2 + (1/2)MV'^2 \end{cases} \tag{3.3}$$

```
Sub OneDr2()
Cells(1,1)="One-dimensional collision based random walk"
Dim X(5000, 5000)  'X(i,j) = i-th Brownian particle system and j-th collision.
Dim BV(5000, 5000) 'Velocity of i-th Brownian system after j-th collision.
Dim R2(5000)
dt = 0.01          'Time interval between two successive collisions.
v0 = 10            'Average velocity of media particles.
M = 0.01           'Mass ratio of [Media particle]/[Brownian particle].
Nsteps = 5000      '# of collisions per Brownian particle.
NP = 1000          '# of Brownian particle systems.
'Initialization:
     For i = 1 To NP
         X(i, 0) = 0
         BV(i, 0) = 0
     Next i
'Random selection of the velocity of the incident particles at each collison:
   For j = 1 To Nsteps - 1
       R2(j) = 0
           For i = 1 To NP
               If Rnd() < 0.5 Then
                   v = v0
               Else
                   v = -v0
               End If
                   BV(i, j) = (2 * M * v + (1 - M) * BV(i, j - 1)) / (1 + M)
                   X(i, j) = X(i, j - 1) + BV(i, j) * dt
                   R2(j) = R2(j) + X(i, j) ^ 2
           Next i
   Next j
        Cells(8, 2) = "Time"
        Cells(8, 3) = "<R^2>"
        R2(0) = 0
    For j = 0 To Nsteps - 1
        Cells(9 + j, 2) = j * dt
        Cells(9 + j, 3) = R2(j) / NP
    Next j
End Sub
```

Fig. 3.1 VBA code for one dimensional kinetic motion of Brownian particle

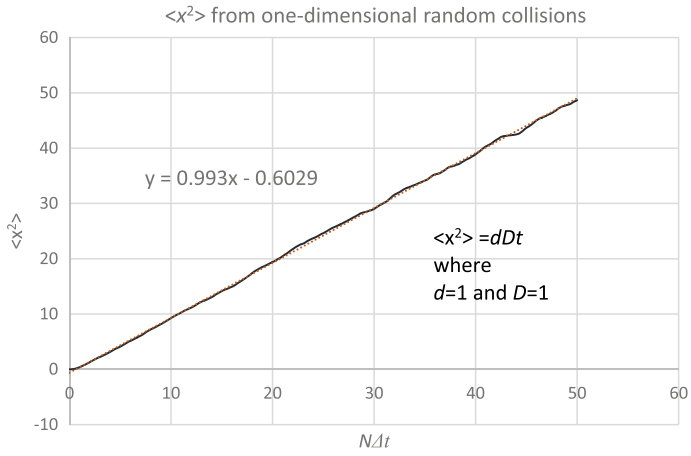


Fig. 3.2 Time dependence of $< x^2 >$ in one-dimensional random collisions

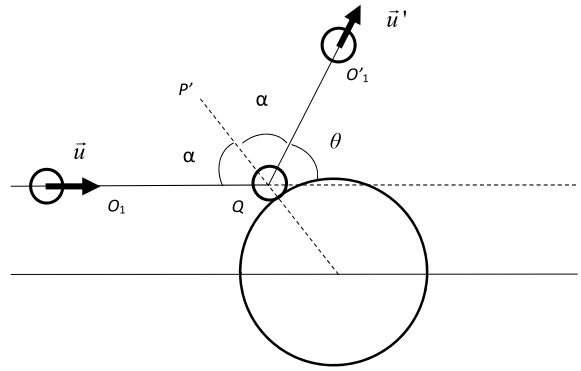


Fig. 3.3 An incident particle colliding with the target particle in the target particle frame

Figure 3.3 illustrates a collision of an incident particle in the frame of a target particle where the velocity of the incident particle before and after a collision are given by the relative velocities, $\vec{u} = \vec{v} - \vec{V}$ and $\vec{u}' = \vec{v}' - \vec{V}'$. No sizes of the incident particles are considered as the incident particles are much smaller than the target. In this figure, $\angle O_1 Q P' = \angle P' Q O'_1 = \alpha$, and $\theta = \pi - 2\alpha$, and the angle α can be considered as a random variable between $-\pi$ and $+\pi$, which represents non-head on collisions.

Now consider the collision in the center of mass frame. The center of mass before and after a collision are given by

$$\vec{r}_{CM} = \frac{m\vec{r} + M\vec{R}}{m+M} \text{ and } \vec{r}'_{CM} = \frac{m\vec{r}' + M\vec{R}'}{m+M} \tag{3.4}$$

where $\vec{r}$, $\vec{r}'$, $\vec{R}$, and $\vec{R}'$ are positions of the incident particle and the target particle before and after a collision in the laboratory frame. From the set of Eqs. (3.4), the velocities of the center of mass before and after a collision are given by

$$\vec{v}_{CM} = \frac{m\vec{v} + M\vec{V}}{m+M} \text{ and } \vec{v}'_{CM} = \frac{m\vec{v}' + M\vec{V}'}{m+M} \tag{3.5}$$

where the velocities appeared in the above equations are

$$\vec{v}_{CM} = \frac{d\vec{r}_{CM}}{dt},\ \vec{v}'_{CM} = \frac{d\vec{r}'_{CM}}{dt},\ \vec{V} = \frac{d\vec{R}}{dt}, \text{ and } \vec{V}' = \frac{d\vec{R}'}{dt}.$$

Because of the Eq. (3.3) of momentum conservation, Eqs. (3.5) lead that the velocity of the center of mass before and after a collision remains the same: $\vec{v}_{CM} = \vec{v}'_{CM}$.

We now take several steps to obtain the iterative equation of the target particle in the laboratory frame.

(i) Using the relative velocity of the incident particle to the target, $\vec{u} = \vec{v} - \vec{V}$, and Eq. (3.5), $\vec{v}$ and $\vec{V}$ in the laboratory frame can be modified to the following equations:

$$\begin{cases} \vec{v} = \vec{v}_{CM} + \dfrac{M}{m+M}\vec{u} = \vec{v}_{CM} + \zeta\vec{u} \end{cases} \tag{3.6}$$

$$\begin{cases} \vec{V} = \vec{v}_{CM} - \dfrac{M}{m+M}\vec{u} = \vec{v}_{CM} - \zeta\vec{u} \end{cases} \tag{3.7}$$

where

$$\zeta = \frac{M}{m+M}. \tag{3.8}$$

By the same token, the velocities of the incident and the target particles after a collision are given by

$$\vec{v}' = \vec{v}'_{CM} + \zeta\vec{u}' \text{ and } \vec{V}' = \vec{v}'_{CM} - \zeta\vec{u}'. \tag{3.9}$$

(ii) Using Eqs. (3.6) to (3.9), Eq. (3.3) of the conservation of energy becomes

$$\frac{1}{2}(\vec{v}_{CM} + \zeta\vec{u})^2 + \frac{1}{2}(\vec{v}_{CM} - \zeta\vec{u})^2 = \frac{1}{2}\left(\vec{v}'_{CM} + \zeta\vec{u}'\right)^2 + \frac{1}{2}\left(\vec{v}'_{CM} - \zeta\vec{u}'\right)^2. \tag{3.10}$$

From Eq. (3.10), we obtain $u^2 = u'^2$, and thus $u = u'$.

(iii) The velocity of the target particle before a collision is given by Eq. (3.7), and thus, $\vec{v}_{CM} = \vec{V} + \zeta\vec{u}$. Using $\vec{v}'_{CM} = \vec{v}_{CM}$, we obtain the velocity of the target particle after a collision:

$$\vec{V}' = \vec{v}_{CM} - \zeta\vec{u}' = \vec{V} - \zeta(\vec{u}' - \vec{u}). \tag{3.11}$$

If we can express the relative velocity $\vec{u}'$ after a collision with variables before the collision, Eq. (3.11) can be the iterative equation to compute the velocity of the target particle successively. Define unit vectors, $\vec{e}_{\parallel}$ and $\vec{e}_{\perp}$, which are parallel and vertical to the relative velocity, $\vec{u}$, respectively. Referring to Fig. (3.3) and using $u = u'$, the relative velocity after the collision, $\vec{u}'$, can be written as

$$\vec{u}' = (u'\vec{e}_{\parallel})\cos\theta + (u'\vec{e}_{\perp})\sin\theta = (u\vec{e}_{\parallel})\cos\theta + (u\vec{e}_{\perp})\sin\theta. \tag{3.12}$$

In Eq. (3.12), the unit vector $\vec{e}_{\parallel}$ is along the relative velocity $\vec{u}$, i.e., $(u\vec{e}_{\parallel})\cos\theta = \vec{u}\cos\theta$. Because the direction of the unit vector $\vec{e}_{\perp}$ is $\pi/2$ from the direction of $\vec{u}$, using the component expression of the relative velocity, $\vec{u} = (u_x, u_y)$ and $\vec{e}_{\perp} = (e_{\perp x}, e_{\perp y})$ in the laboratory frame, the components of $u'\vec{e}_{\perp} = u\,\vec{e}_{\perp}$ in the laboratory frame can be calculated by the rotational matrix:

$$u'\vec{e}_{\perp} = u\vec{e}_{\perp} = u\begin{pmatrix} e_{\perp,x} \\ e_{\perp,y} \end{pmatrix} = u\begin{pmatrix} \cos(\pi/2) & -\sin(\pi/2) \\ \sin(\pi/2) & \cos(\pi/2) \end{pmatrix}\begin{pmatrix} e_{//x} \\ e_{//y} \end{pmatrix} = \begin{pmatrix} -ue_{//y} \\ ue_{//x} \end{pmatrix} = \begin{pmatrix} -u_y \\ u_x \end{pmatrix} \tag{3.13}$$

(iv) Therefore, in the laboratory coordinates, using Eqs. (3.12) and (3.13), Eq. (3.11) becomes

$$\vec{V}' = \vec{V} - \zeta(\vec{u}' - \vec{u}) = \vec{V} - \zeta(u\vec{e}_{\parallel}\cos\theta + u\vec{e}_{\perp}\sin\theta - \vec{u}), \tag{3.14}$$

or in the matrix form, we obtain

$$\begin{pmatrix} V_x' \\ V_y' \end{pmatrix} = \begin{pmatrix} V_x \\ V_y \end{pmatrix} - \zeta\begin{pmatrix} u_x\cos\theta - u_y\sin\theta - u_x \\ u_y\cos\theta + u_x\sin\theta - u_x \end{pmatrix} = \begin{pmatrix} V_x \\ V_y \end{pmatrix} + \zeta\begin{pmatrix} u_x(1-\cos\theta) + u_y\sin\theta \\ u_y(1-\cos\theta) - u_x\sin\theta \end{pmatrix}. \tag{3.15}$$

Equation (3.15) will be our iteration equation for the two-dimensional collision-based Brownian motion, and the position of the target particle will be given by $\vec{x}' = \vec{x} + \vec{V}'\Delta t$ after a single collision.

Figures 3.4 is a VBA code that uses the above iterative formula (3.15). For calculation of the average distance, we used 4,000 iterations of the target particles. Notice that selection of $\zeta = m/M$, Δt, and v_0 determines the linearity. In Fig. 3.4, the collision parameters are $\zeta = 0.01$, $\Delta t = 0.01$, $v_0 = 10$, and the total number of collisions is 4,000.

Figure 3.5 is the trajectory of the two-dimensional trajectory of a target particle from the above VBA code. The chart is drawn using [`Scatter with Straight Lines`].

Figure 3.6 shows the computed the mean square distance, $< R^2 >$, from the VBA code. Similar to the one-dimensional case, it reveals that $R^2 = {R_x}^2 + {R_y}^2$ becomes a linear function of the number of steps or time. In Fig. 3.6, $< R^2 >$ is proportional to $d \bullet t$ and $d = 2$. This result is considered to be equivalent to Fig. 2.14 of the two-dimensional random walk, indicating that the collision process can be modeled as the random walks.

3.2 Langevin Equation

The motion of a Brownian particle with mass M at velocity $\vec{v}$ can be described by the Langevin equation [4]. It is a Newton's equation of motion, $M\frac{d\vec{v}}{dt} = \vec{K}(t)$, where the external force, $\vec{K}(t)$, can be divided into two parts:

(1) the usual viscous drag on the particle, $-\beta M\vec{v}$, where, for a sphere of radius R, Stokes's law gives $\beta = 6\pi R\eta/M$ and η is the coefficient of viscosity; and

```
Cells(1, 1) = "Two-dimensional collision based random walk"
  Dim X(4000, 4000)
  Dim Y(4000, 4000)
  Dim Vx(4000, 4000)
  Dim Vy(4000, 4000)
  Dim ux(4000, 4000)
  Dim uy(4000, 4000)
  Dim R2(4000, 4000)
  Dim R2ave(4000)
     M = 0.01              'Mass ratio = m/(M+m).
     Nsteps = 4000         '# of collisions.
     NP = 4000             '# of systems of a Brownian particle.
     v0 = 10               'Speed of media particles.
     dt = 0.01             'Time interval between two successive collisions.
     Pi = 3.1415932654
   Cells(8, 5) = "# collisions"
   Cells(8, 6) = "x"
   Cells(8, 7) = "y"
   Cells(8, 9) = "Time"
   Cells(8, 10) = "<R^2>"
'Initial conditions starting at the origin:
   Cells(9, 5) = 0
   Cells(9, 6) = 0
   Cells(9, 7) = 0
   Cells(9, 9) = 0
   Cells(9, 10) = 0
'Initialization:
Randomize
   For i = 1 To NP
      X(i, 0) = 0
      Y(i, 0) = 0
      Vx(i, 0) = 0
      Vy(i, 0) = 0
      R2(i, 0) = 0
   Next i
      R2ave(0) = 0               'Initial <R^2>.
  For i = 1 To NP                 'For each Brownian particle after one collision.
    If Rnd() < 0.5 Then
        v = v0
    Else
        v = -v0
    End If
      alpha = Pi * (2 * Rnd() - 1)  'Generate random angle.
      theta = Pi - 2 * alpha
    'Relative velocity:
      ux(i, 1) = v * Cos(alpha) - Vx(i, 0)
      uy(i, 1) = v * Sin(alpha) - Vy(i, 0)
    'Position after one collision:
       X(i, 1) = X(i, 0) + (Vx(i, 0) + M * ((1 - Cos(theta)) * ux(i, 1) + Sin(theta)
* uy(i, 1))) * dt
       Y(i, 1) = Y(i, 0) + (Vy(i, 0) + M * ((1 - Cos(theta)) * uy(i, 1) - Sin(theta)
* ux(i, 1))) * dt
       R2(i, 1) = X(i, 1) ^ 2 + Y(i, 1) ^ 2
  Next i
     For i = 1 To NP
        R2ave(1) = R2ave(1) + R2(i, 1)
     Next i
  R2ave(1) = R2ave(1) / NP             '<R^2> after one collision.
     Cells(10, 5) = 1 * dt
     Cells(10, 6) = X(1, 1)
     Cells(10, 7) = Y(1, 1)
     Cells(10, 9) = R2ave(1)
 For j = 2 To Nsteps - 1                'Collision steps.
    R2ave(j) = 0
        For i = 1 To NP                 'Brownian particles.
            If Rnd() < 0.5 Then
                v = v0
            Else
                v = -v0
            End If
       alpha = Pi * (2 * Rnd() - 1)     'Generate random angle.
       theta = Pi - 2 * alpha
            ux(i, j - 1) = v * Cos(alpha) - Vx(i, j - 1)
            uy(i, j - 1) = v * Sin(alpha) - Vy(i, j - 1)
  Vx(i, j) = Vx(i, j - 1) + M * (ux(i, j - 1) * (1 - Cos(theta)) + uy(i, j - 1) *
Sin(theta))
  Vy(i, j) = Vy(i, j - 1) + M * (uy(i, j - 1) * (1 - Cos(theta)) - ux(i, j - 1) *
Sin(theta))
        X(i, j) = X(i, j - 1) + Vx(i, j) * dt
        Y(i, j) = Y(i, j - 1) + Vy(i, j) * dt
'Calculate distance from the origin after 1 collision:
        R2(i, j) = X(i, j) ^ 2 + Y(i, j) ^ 2
           If i <> 1 Then GoTo Skip
               Cells(9 + j, 5) = j * dt
               Cells(9 + j, 6) = X(1, j)
               Cells(9 + j, 7) = Y(1, j)
Skip:          R2ave(j) = R2ave(j) + R2(i, j)
           Next i
              R2ave(j) = R2ave(j) / NP        'Compute average R^2.
         If (j Mod 10) = 0 Then
             Cells(10 + j / 10, 9) = j * dt
             Cells(10 + j / 10, 10) = R2ave(j)
         End If
  Next j
End Sub
```

Fig. 3.4 VBA code of 2-dimensional kinetic motion of Brownian particle

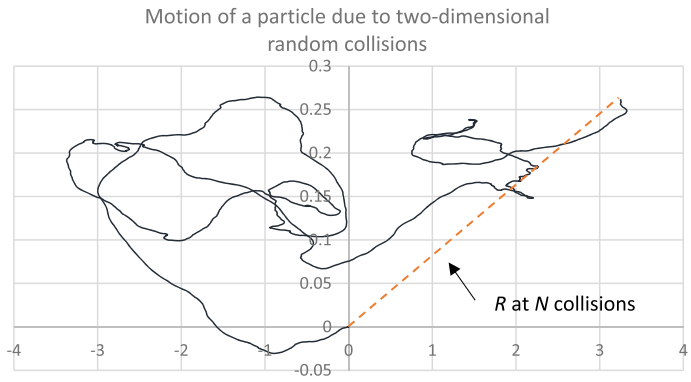


Fig. 3.5 Two-dimensional trajectory of a particle due to collisions with surround particles

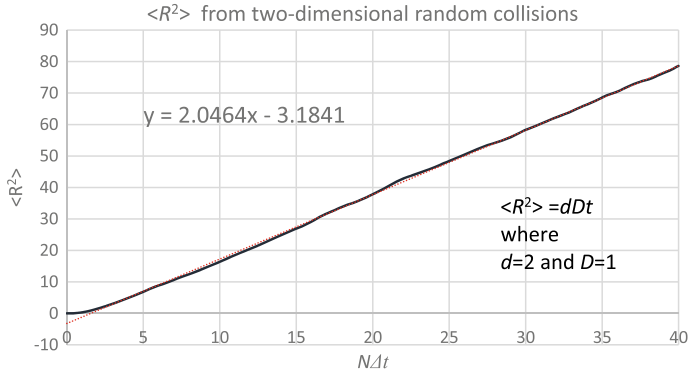


Fig. 3.6 Time dependence of $< R^2 >$

(2) the other is a rapidly fluctuating stochastic force, $M\vec{F}(t)$, representing the effect of moving impacts on the particle.

We show that, starting with the Langevin equation, we can obtain Einstein's relation for Brownian motion. Using the explicit external force, $\vec{K}(t) = M\left(-\beta\vec{v} + \vec{F}(t)\right)$, the equation of motion becomes

$$\frac{d\vec{v}}{dt} = \frac{d^2\vec{x}}{dt^2} = -\beta\vec{v} + \vec{F}(t). \tag{3.16}$$

From Eq. (3.16), consider the following equation:

$$\vec{x} \cdot \frac{d\vec{v}}{dt} = \vec{x} \cdot \frac{d^2\vec{x}}{dt^2} = -\beta\vec{x} \cdot \vec{v} + \vec{x} \cdot \vec{F}(t). \tag{3.17}$$

Because

$$\frac{d^2(\vec{x})^2}{dt^2} = \frac{d}{dt}\left(2\vec{v}\cdot\frac{d\vec{x}}{dt}\right) = 2\left(\frac{d\vec{x}}{dt}\right)^2 + 2\vec{x}\cdot\frac{d^2\vec{x}}{dt^2},$$

we obtain

$$\vec{x}\cdot\frac{d^2\vec{x}}{dt^2} = \frac{1}{2}\frac{d^2(\vec{x})^2}{dt^2} - \left(\frac{d\vec{x}}{dt}\right)^2,$$

and the Eq. (3.17) becomes

$$\vec{x}\cdot\frac{d^2\vec{x}}{dt^2} = \frac{1}{2}\frac{d^2(\vec{x})^2}{dt^2} - \left(\frac{d\vec{x}}{dt}\right)^2 = -\beta\vec{x}\cdot\vec{v} + \vec{x}\cdot\vec{F}(t). \tag{3.18}$$

Therefore,

$$\frac{1}{2}\frac{d^2(\vec{x})^2}{dt^2} - (\vec{v})^2 = -\frac{1}{2}\beta\frac{d(\vec{x})^2}{dt} + \vec{x}\cdot\vec{F}(t). \tag{3.19}$$

Define

$$\alpha = \left\langle\frac{d(\vec{x})^2}{dt}\right\rangle,$$

and we obtain

$$\frac{d\alpha}{dt} - 3\frac{k_B T}{M} = -\beta\alpha \tag{3.20}$$

where we used the uncorrelation relationship between $\vec{x}$ and $\vec{F}(t)$, i.e., $\left\langle\vec{x}\cdot\vec{F}(t)\right\rangle = 0$. With the equipartition theorem of thermodynamics, $(1/2)M < v^2 > = (1/2)d{\cdot}k_{\mathrm{B}}T$ where $d = 1$, 2, or 3 dimensions of the system, the solution of the differential Eq. (3.20) for α is given by

$$\alpha = \frac{2dk_B T}{\beta M} + ce^{-\beta t}$$

where c is a constant. After a sufficiently long time, the exponential term should be negligible, and the resultant solution will be

$$\alpha = \left\langle\frac{d(\vec{x})^2}{dt}\right\rangle = \frac{d}{dt}\left\langle(\vec{x})^2\right\rangle = d\frac{2k_B T}{\beta M}, \tag{3.21}$$

and thus

$$\left\langle(\vec{x})^2\right\rangle = d\frac{2k_B T}{\beta M}t = dDt \text{ where } D = \frac{2k_B T}{\beta M}. \tag{3.22}$$

Equation (3.22) is Einstein's relationship we discussed in Sects. 2.4 and 3.1. Therefore, the Langevin equation to model Brownian motion is an alternate expression of the collision-based kinematics of the target particle and the random walk process.

3.3 Smoluchowski Equation to Diffusion Equation

3.3.1 Smoluchowski Equation to Fokker-Plank Equation

In this section, we do not concern what causes the random motion but want to describe the positions of a Brownian particle as a function of time as a stochastic process. Let x be the position of a randomly moving particle in one-dimension, and $P(x_1 \mid x_2, t)dx_2$ be the probability that the particle at $x = x_1$ at $t = 0$ is forwarded to $[x_2, x_2 + dx_2]$ at time t. Define an intermediate time t_0 in the time interval $[0, t]$, and the position in $[y, y + \mathrm{d}y]$ at t_0, and suppose a Brownian particle from x_1 at $t = 0$ to x_2 at time t via the intermediate position y at $t = t_0$, the following probabilities can be written:

$$P(x_1|y, t_0) \text{ for the step } x_1 \text{to } y, \text{ and } P(y|x_2, t - t_0) \text{ for the step } y \text{ to } x_2. \tag{3.23}$$

Thus, $P(x_1|x_2, t) = P(x_1|y, t_0)P(y|x_2, t - t_0)$.

We assume the random motion is a Markov process: any event after a given time, t, is not affected by any event before t. Then, taking all possibilities of the intermediate positions, y, we obtain the Smoluchowski equation [5].

$$P(x_1|x_2, t) = \int P(x_1|y, t_0)P(y|x_2, t - t_0)dy. \tag{3.24}$$

Form the Smoluchowski equation, we can derive a differential equation, called the Fokker-Plank equation:

$$\frac{\partial P(x_1|x, t)}{\partial t} = -\frac{\partial}{\partial x}[A(x)P(x_1|x, t] + \frac{1}{2}\frac{\partial^2}{\partial x^2}[B(x)P(x_1|x, t)] \tag{3.25}$$

where $A(x) = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \int d\xi (\xi - x) P(x|\xi, \Delta t)$ and $B(x) = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \int d\xi (\xi - x)^2 P(x|\xi, \Delta t)$ are called the drift term, and $B(y)$ is the diffusion term of the distribution, respectively [6].

Proof Let's start with the Smoluchowski Eq. (3.24) with modification:

$$P(x_1|y, t_0 + \Delta t) = \int P(x_1|y - \xi, t_0)P(y - \xi|y, \Delta t)d\xi \tag{3.26}$$

where Δt will be a small time-increment and $t = t_0 + \Delta t$. The Taylor expansion of the left side gives the following expression to the first order:

$$P(x_1|y, t_0 + \tau) = P(x_1|y, t_0) + \frac{\partial P(x_1|y, t_0)}{\partial t}\Delta t,$$

where

$$\frac{\partial P(x_1|y, t_0)}{\partial t}\Delta t = \int P(x_1|y - \xi, t_0)P(y - \xi|y, \Delta t)d\xi - P(x_1|y, t_0) - P(x_1|y, t_0). \tag{3.27}$$

Setting z=y−ξ , the integrand of the righthand side becomes

$$P(x_1|y - \xi, t_0)P(y - \xi|y, \Delta t) = P(x_1|z, t_0)P(z|z + \xi, \Delta t),$$

which can be expanded to

$$\begin{aligned} &P(x_1|z, t_0)P(z|z + \xi, \Delta t) = P(x_1|z, t_0)P(z|z, \Delta t) + \sum_{n=1} \frac{(-1)^n}{n!}\xi^n \frac{\partial^n}{\partial z^n}[P(x_1|z, t_0)P(z|z + \xi, \Delta t)] \\ &= P(x_1|z, t_0) - \xi \frac{\partial}{\partial z}[P(x_1|z, t_0)P(z|z + \xi, \Delta t)] + \frac{1}{2}\xi^2 \frac{\partial^2}{\partial z^2}[P(x_1|z, t_0)P(z|z + \xi, \Delta t)] - +... \end{aligned} \tag{3.28}$$

Therefore, Eq. (3.27) becomes

$$\begin{aligned} &\frac{\partial P(x_1|y, t_0)}{\partial t}\Delta t \\ &= \int d\xi \left\{ -\xi \frac{\partial}{\partial z}[P(x_1|z, t_0)P(z|z + \xi, \Delta t)] + \frac{1}{2}\xi^2 \frac{\partial^2}{\partial z^2}[P(x_1|z, t_0)P(z|z + \xi, \Delta t)] \right\} \\ &= -\frac{\partial}{\partial y}\left[P(x_1|y, t_0) \int \xi P(y - \xi|y, \Delta t)d\xi \right] + \frac{1}{2}\frac{\partial^2}{\partial y^2}\left[P(x_1|y, t_0) \int \xi^2 P(y - \xi|y, \Delta t)d\xi \right] \end{aligned} \tag{3.29}$$

Taking the limit, $\Delta t \to 0$, we obtain the Fokker-Plank equation:

$$\begin{aligned} &\frac{\partial P(x_1|y, t_0)}{\partial t} \\ &= -\frac{\partial}{\partial y}\left[P(x_1|y, t_0) \lim_{\Delta t \to 0} \frac{1}{\Delta t} \int d\xi(\xi) P(y - \xi|y, \Delta t) \right] \\ &+ \frac{1}{2}\frac{\partial^2}{\partial y^2}\left[P(x_1|z, t_0) \lim_{\Delta t \to 0} \frac{1}{\Delta t} \int d\xi(\xi)^2 P(y - \xi|y, \Delta t) \right] \\ &= -\frac{\partial}{\partial y}[A(y)P(x_1|y, t_0)] + \frac{1}{2}\frac{\partial^2}{\partial y^2}[B(y)P(x_1|y, t_0)], \end{aligned} \tag{3.30}$$

where $A(x)$ and $B(x)$ are defined by

$$A(y) = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \int d\xi(\xi) P(y - \xi | y, \Delta t) \text{ and } B(y) = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \int d\xi(\xi)^2 P(y - \xi | y, \Delta t). \tag{3.31}$$

■

3.3.2 Fokker Plank Equation to Diffusion Equation

In a Brownian motion, $P(x_1|x, t)$ is a symmetric function of $x - x_1$, corresponding to the equal probability of movement to the right or to the left. This is called detailed balancing condition. With this condition, the drift term, which is equal to the first moment should be zero: $A(x) = 0$. If the diffusion term, $B(x)$, is further independent of the position, the Fokker-Plank Eq. (3.30) reduces to a diffusion equation:

$$\frac{\partial P(x_1|x, t)}{\partial t} = \frac{B}{2} \frac{\partial^2 P(x_1|x, t)}{\partial x^2}. \tag{3.32}$$

This is a diffusion equation from which. we can obtain Einstein's relation as follows.

With the probability distribution function, $P(x|x, t) \equiv P(x, t)$, the average position is given by $< x(t) >= \int x P(x, t) dx$. Consider the patrial time derivative of $< x >$:

$$\frac{\partial < x(t) >}{\partial t} = \int x \frac{\partial P(x, t)}{\partial t} dx. \tag{3.33}$$

Using the right-side of diffusion Eq. (3.32) and the property,

$$P(x, t) \to 0 \text{ and } x \frac{\partial P(x, t)}{\partial t} \to 0 \text{ as } x \to \pm\infty,$$

we obtain

$$\frac{B}{2} \int x \frac{\partial^2 P(x, t)}{\partial x^2} dx = \frac{B}{2} x \frac{\partial P(x, t)}{\partial x} \Big|_{-\infty}^{+\infty} - \frac{B}{2} \int \frac{\partial P(x, t)}{\partial x} dx = 0. \tag{3.34}$$

Thus,

$$\frac{\partial < x(t) >}{\partial t} = \int x \frac{\partial P(x, t)}{\partial t} dx = \frac{B}{2} \int x \frac{\partial^2 P(x, t)}{\partial x^2} = 0, \tag{3.35}$$

and hence $< x >$ = constant. From the initial condition, $x = 0$ at $t = 0$, we have $< x > = 0$.

Next, consider the partial time derivative of $< x^2 >$:

$$\frac{\partial < x^2(t) >}{\partial t} = \int x^2 \frac{\partial P(x,t)}{\partial t} dx = \frac{B}{2} \int x^2 \frac{\partial^2 P(x,t)}{\partial x^2}$$
$$= \frac{B}{2} x^2 \frac{\partial P(x,t)}{\partial x}\Big|_{-\infty}^{+\infty} - 2\frac{B}{2} \int x \frac{\partial P(x,t)}{\partial x} dx$$
$$= -B\, P(x,t)|_{-\infty}^{+\infty} + B = B \tag{3.36}$$

Therefore, we obtain Einstein's relation, $< x^2(\mathrm{t}) > = Bt$, where $B = 2D$, from Eq. (3.36) which originates from the Fokker-Plank equation.

Form the above arguments, we are now aware that the diffusion equation can derive Einstein's relation, and hence can be applied to describe Brownian motion. From the argument of Sect. 2.4, the random walk can also derive Einstein's relation. This means we can also apply the random walk to diffusion processes to describe Brownian motion. Therefore, the Monte Carlo simulation can be applied to Brownian motion and diffusion processes by using random walk models.

3.4 Diffusion Process by Random Walk

In this section, we compute the dynamical distribution of diffusing particles. We compute multiple random walks to each of which the Monte Carlo simulation is applied, and then visualize the distribution of the random walkers.

3.4.1 One-Dimensional Diffusion

The VBA code for a single random walker (particles) listed in Fig. 2.11 can be modified to multiple walkers to simulate the one-dimensional diffusion process. In the code of Fig. 3.7, there are 5,000 random walkers to show their distributions at different times.

Figures 3.8 shows the normalized probability distribution of one-dimensional 5,000 walkers at 500 and 5,000 steps. The initial positions of the walkers are all at the origin. The broken lines are Gaussian distribution functions fitted to the walkers' distributions. In Sect. 3.5 below, we show that the distribution function is Gaussian at a fixed time. As we depict the analytical solution of one-dimensional diffusion equation in the next section, we can verify these observations.

3.4.2 Two-Dimensional Diffusion

The VBA code shown in Fig. 3.9 is an extension of the single random walker code on a two-dimensional plane without directional restriction as shown in Fig. 2.17. The

```
Sub RandomDiff()
Cells(1,1)="One-dimensional diffusion by random walk"
'Place 5000 particles at the origin initially.
'Record the particle positions at every 10 walks.
'Repeat the above set 100 times to have totally 1000 steps.
Dim x(5000)                          '# of particles.
    Cells(1, 3) = "time interval"
    Cells(2, 2) = "x"
    N0 = 5000
       For i = 1 To N0
           x(i) = 0
           Cells(3 + i, 2) = x(i)
       Next i
Randomize
For i = 1 To N0
    For N = 1 To 500                  'The first 500 iterations.
        For k = 1 To 10               'Display positions at every 10 intervals.
            If Rnd() < 0.5 Then
                x(i) = x(i) + 1
            Else
                x(i) = x(i) - 1
            End If
        Next k
            Cells(2, 2 + N) = N
            Cells(3 + i, 2 + N) = x(i)
    Next N
Next i
End Sub
```

Fig. 3.7 VBA code for one-dimensional diffusion by 5,000 random walkers

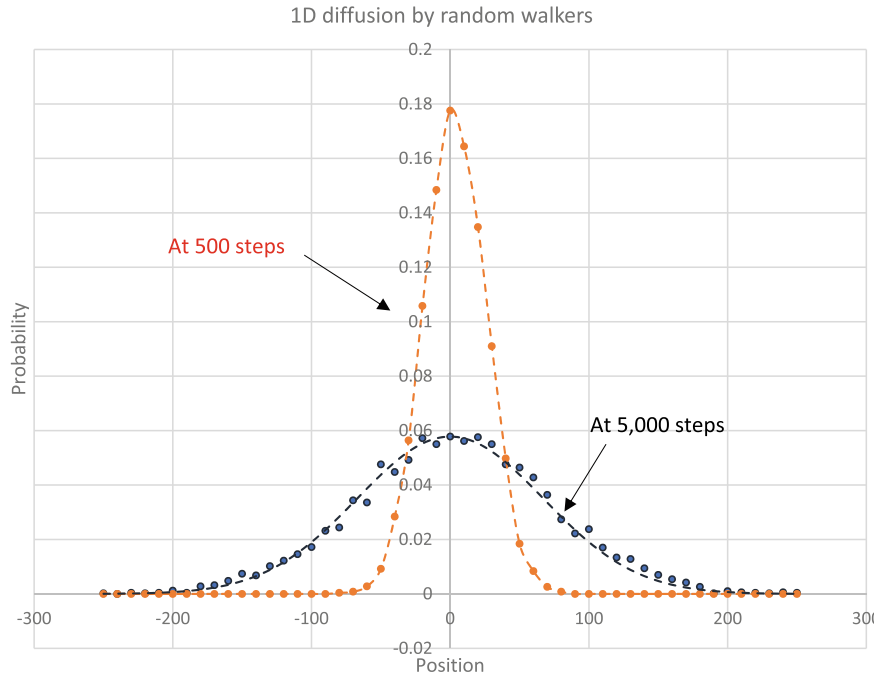


Fig. 3.8 Distribution of 1-dimensional random walkers at different steps

number of particles is 5,000, all of which are at the origin initially, and we calculated the distributions of the walkers at 500, 2,500, and 5,000 steps.

```
Sub Twodiff()
Cells(1, 1) = "Diffusion by two-dimensional random walks of NP=5,000 particles"
Dim x(5000) As Double
Dim y(5000) As Double
Dim r(5000) As Double
Dim theta As Double
  Ntrial = 500: Cells(4, 2) = trial_no      '# of random walks.
  step = 1: Cells(6, 2) = step              'Step width per walk.
  NP = 5000                                 '# of particles.
  pi=3.141592654
'All particles are at the origin initially:
For k = 0 To NP
    x(k) = 0
    y(k) = 0
Next k
    Cells(9, 2) = "x"
    Cells(9, 3) = "y"
    Cells(9, 4) = "r"
    Cells(9, 4) = "r at t"
Randomize
'Positions after 1 trial (500 walks):
For k = 1 To NP
    For I = 1 To Ntrial
        theta = 2 * pi * Rnd()              'Generate a random angle.
        x(k) = x(k) + step * Cos(theta)
        y(k) = y(k) + step * Sin(theta)
        r(k) = Sqr(x(k) ^ 2 + y(k) ^ 2)
    Next i
        Cells(9 + k, 2) = x(k)
        Cells(9 + k, 3) = y(k)
        Cells(9 + k, 4) = r(k)
Next k
'Positions after 5 trials (5x500=2500 walks):
For k = 0 To NP
    x(k) = 0
    y(k) = 0
Next k
   For k = 1 To NP
       For i = 1 To 5 * Ntrial
           theta = 2 * pi * Rnd()
           x(k) = x(k) + step * Cos(theta)
           y(k) = y(k) + step * Sin(theta)
           r(k) = Sqr(x(k) ^ 2 + y(k) ^ 2)
       Next i
        Cells(9, 5) = "r at 5t"
        Cells(9 + k, 5) = r(k)
   Next k
'Positions after 10 trials (10x500=5000 walks):
For k = 0 To NP
    x(k) = 0
    y(k) = 0
Next k
    For k = 1 To NP
        For i = 1 To 10 * Ntrial
            theta = 2 * pi * Rnd()
            x(k) = x(k) + step * Cos(theta)
            y(k) = y(k) + step * Sin(theta)
            r(k) = Sqr(x(k) ^ 2 + y(k) ^ 2)
        Next i
            Cells(9, 6) = "r at 10t"
            Cells(9 + k, 6) = r(k)
     Next k
End Sub
```

Fig. 3.9 VBA code of two-dimensional random walks of 5,000 walkers

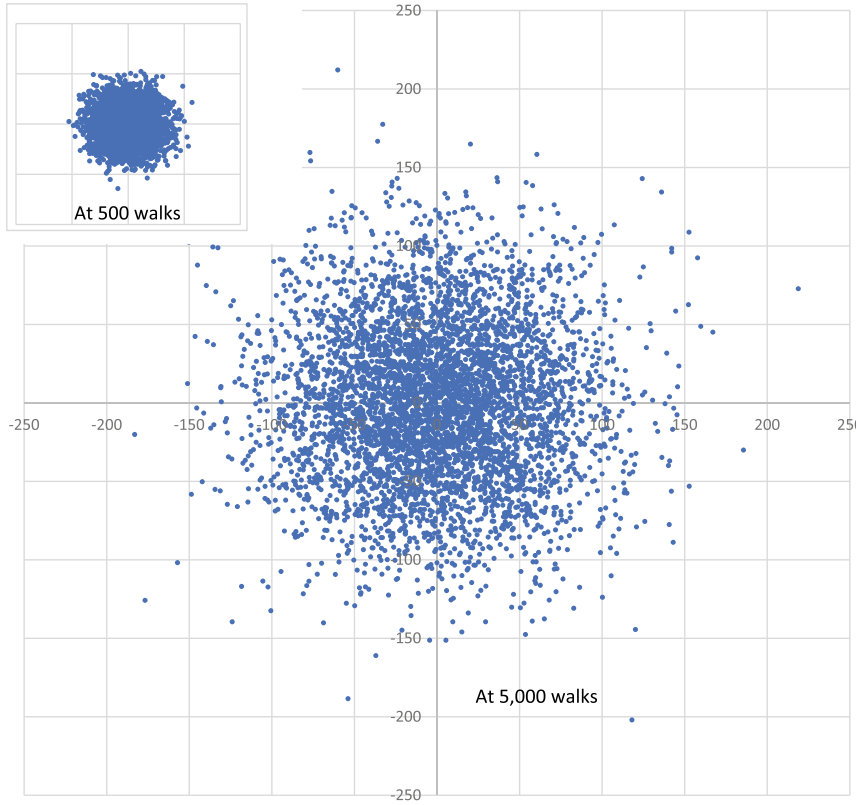


Fig. 3.10 Distribution of 5000 walkers at 500 and 5,000 steps

Figure 3.10 shows the two-dimensional distribution of the 5,000 particles at 5,000 steps where the distribution at 500 walks is onset with the same area scale.

Figure 3.11 shows the computed radial probability distributions of two-dimensional random walkers at 500 and 5,000 walks. Broken lines are theoretical two-dimensional radial probability functions, $A \cdot r \cdot \exp(-\alpha r^2)$ from Gaussian distribution functions $\exp(-\alpha r^2)$ where constants A and α are optimized to each distribution. From this figure, we observe that the walkers' distribution is Gaussian at a given time. This means the two-dimensional diffusion equation has a Gaussian solution, which leads to Einstein's Relation. This was also confirmed in Sect. 2.4.2.

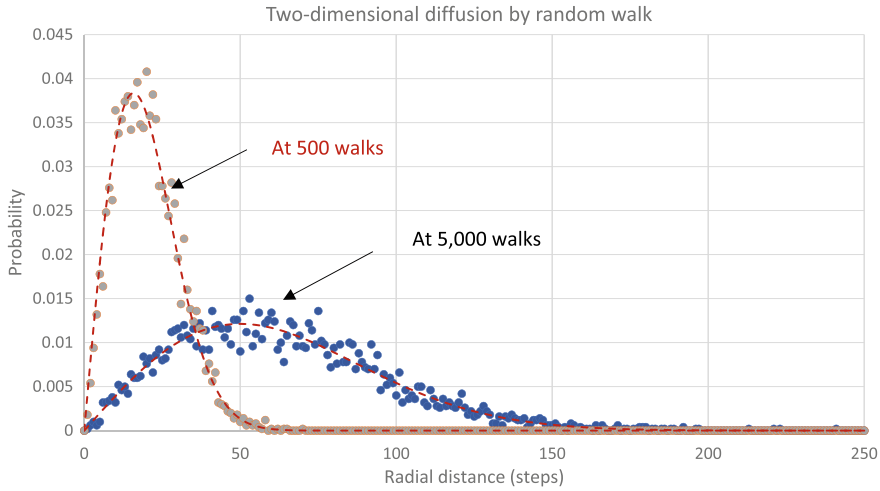


Fig. 3.11 Radial probability distribution of random walkers at 500 and 5,000 steps

3.5 Analytical Solution of One-Dimensional Diffusion Equation

The analytical solution brings important insight to the simulations. Here, we have two analytical solutions: the trial function method that is seen for finding the ground state of the quantum mechanical harmonic oscillator [7], and the spectral method of applying the Fourier transform [8].

3.5.1 Trial Function Method

Let's solve the diffusion equation

$$\frac{\partial P(x_1|x,t)}{\partial t} = D\frac{\partial^2 P(x_1|x,t)}{\partial x^2} \tag{3.37}$$

using a trial function, $P(x,t) = t^{-p}F(\mu)$ where $\mu = x^2/4Dt$.

From the trial function, we obtain:

$$\begin{aligned}
\frac{\partial P}{\partial t} &= -pt^{-p-1}F(\mu) - \mu t^{-p-1}\frac{dF}{d\mu}, \\
\frac{\partial P}{\partial x} &= t^{-p}\frac{dF}{d\mu}\frac{\partial \mu}{\partial x} = \frac{xt^{-p-1}}{2D}\frac{dF}{d\mu}, \\
\frac{\partial^2 P}{\partial x^2} &= \frac{t^{-p-1}}{2D}\frac{dF}{d\mu} + \frac{\mu t^{-p-1}}{D}\frac{d^2F}{d\mu^2},
\end{aligned} \tag{3.38}$$

and thus, the diffusion Eq. (3.32) becomes

$$\mu \frac{d}{d\mu}\left(\frac{dF}{d\mu} + F\right) + \frac{1}{2}\left(\frac{dF}{d\mu} + 2pF\right) = 0. \tag{3.39}$$

Taking $p = 1/2$, the above equation becomes

$$\mu \frac{d}{d\mu}\left(\frac{dF}{d\mu} + F\right) + \frac{1}{2}\left(\frac{dF}{d\mu} + F\right) = 0,$$

And, to satisfy the equation, we may set

$$\frac{dF}{d\mu} + F = 0. \tag{3.40}$$

A solution of the above equation is $F(\mu) = ce^{-\mu}$, and we obtain

$$P(x,t) = t^{-p}F(\mu) = ct^{-1/2}\exp\left(-\frac{x^2}{4Dt}\right). \tag{3.41}$$

The normalization condition, $\int_{-\infty}^{+\infty} P(x,t)dx = 1$, determines the constant c to be $c = 1/\sqrt{4\pi D}$,

and the normalized probability distribution is given by

$$P(x,t) = \frac{1}{\sqrt{4\pi Dt}}\exp\left(-\frac{x^2}{4Dt}\right). \tag{3.42}$$

For a fixed time, the distribution is Gaussian of the mean vale, $< x > \; = 0$, and the standard deviation,$\sigma = \sqrt{2Dt}$. Therefore, from $\sigma^2 = \langle x^2\rangle - \langle x\rangle^2$, we obtain Einstein's relation, $< x^2 > \; = 2Dt$.

3.5.2 Spectral Method

In the spectral method, the Fourier transform is applied to the diffusion equation. It was indeed Joseph Fourier who applied the Fourier transform to heat transfer for the first time. We define the Fourier transform of $P(x, t)$ and the inverse Fourier transform as

$$\begin{cases} FT[P(x,t)] = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty} P(x,t)e^{-ikx}dx \equiv \rho(k,t) \\ FT^{-1}[\rho(k,t)] = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty} \rho(x,t)e^{+ikx}dk = P(x,t) \end{cases} \tag{3.43}$$

By replacing $P(x, t)$ with $FT^{-1}[\rho(k, t)]$, the diffusion Eq. (3.37) becomes

$$\frac{\partial}{\partial t}\left[\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty}\rho(k,t)e^{+ikx}dk\right]=D\frac{\partial^2}{\partial x^2}\left[\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty}\rho(k,t)e^{+ikx}dk\right].$$

Thus,

$$\begin{aligned}\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty}\frac{\partial\rho(k,t)}{\partial t}e^{+ikx}dk&=D\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty}\frac{\partial^2}{\partial x^2}\Big(\rho(k,t)e^{+ikx}\Big)dk\\&=D\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty}\rho(k,t)[-k^2]e^{+ikx}dk.\end{aligned}\tag{3.44}$$

From the Fourier transform of Eq. (3.44), we obtain

$$\begin{aligned}&\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty}dxe^{-ik'x}\left[\int_{-\infty}^{+\infty}dk\frac{\partial\rho(k,t)}{\partial t}e^{+ikx}\right]\\&=D\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty}dxe^{-ik'x}\int_{-\infty}^{+\infty}dk\rho(k,t)[-k^2]e^{+ikx}dk.\end{aligned}\tag{3.45}$$

Since $\int_{-\infty}^{+\infty}e^{i(k-k')x}dx=2\pi\delta(k-k')$ where $\delta(k-k')$ is the Dirac delta function, Eq. (3.45) becomes

$$\frac{\partial\rho(k,t)}{\partial t}=-k^2D\rho(k,t).\tag{3.46}$$

This is the "decay equation" we analyzed in Sect. 2.3 in the k-space. Thus, its solution is given by

$$\rho(k,t)=\rho_0e^{-k^2Dt}.\tag{3.47}$$

The constant, ρ_0, can be determined as follows. Taking the inverse Fourier transform of the solution (3.47), we obtain

$$P(x,t)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty}dk\rho(k,t)e^{+ikx}dk=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty}\rho_0e^{-k^2Dt}e^{+ikx}dk\tag{3.48}$$

From Eq. (3.48) at $t=0$, we obtain $P(x,0)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{+\infty}\rho_0e^{+ikx}dk$. Thus,

$$FT[P(x,\ 0)] = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} P(x,0) e^{-ikx} dk = \rho_0 \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{-ikx} dk = \rho_0. \tag{3.49}$$

For the initial condition $P(x,0) = \delta(x)$, we obtain

$$\rho_0 = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} \delta(x) e^{-ikx} dk = \frac{1}{\sqrt{2\pi}},$$

and the solution, $P(x, t)$, with the initial condition $P(x,0) = \delta(x)$ is given by Eq. (3.48):

$$\begin{aligned} P(x,t) &= \frac{1}{2\pi} \int_{-\infty}^{+\infty} dk e^{-k^2 Dt} e^{+ikx} = \frac{1}{2\pi} e^{-(x^2/4Dt)} \int_{-\infty}^{+\infty} dk e^{-Dt[k-(ix/2Dt)^2]} \\ &= \frac{1}{2\pi} e^{-(x^2/4Dt)} \sqrt{\frac{\pi}{Dt}} = \frac{1}{\sqrt{4\pi\, Dt}} e^{-(x^2/4Dt)}. \end{aligned} \tag{3.50}$$

This is the same as the solution (3.42) from the trial function method.

3.6 Numerical Analysis of One-Dimensional Diffusion Equation

In this section, we use a simplified notation of the diffusion Eq. (3.37):

$$\frac{\partial u(x,t)}{\partial t} = D \frac{\partial^2 u(x,t)}{\partial x^2}. \tag{3.51}$$

Differential Eq. (3.51) can describe various physical phenomena including particle diffusion, and thermal conduction. Even the Schrödinger equation can be altered to the diffusion equation as we will describe in Sect. 4.1. Although the particle diffusion can be modeled by the Monte Carlo random walk as we have found in this chapter, Eq. (3.51) can be also solved by carrying out the iterative computation with both the initial and the boundary conditions that are specific to a model to be analyzed. In this section, we solve the diffusion equation with two different boundary conditions for one-dimensional particle diffusion and thermal conduction.

For the numerical computation, we need to obtain a difference equation from the differential equation. From the Taylor expansion of a function $u(x, t)$:

$$\begin{aligned} u(x, t+\Delta t) &= u(x,t) + (\Delta t) \frac{\partial u(x,t)}{\partial t} + \frac{1}{2} (\Delta t)^2 \frac{\partial^2 u(x,t)}{\partial t^2} + \cdots, \\ u(x+\Delta x, t) &= u(x,t) + (\Delta x) \frac{\partial u(x,t)}{\partial x} + \frac{1}{2} (\Delta x)^2 \frac{\partial^2 u(x,t)}{\partial x^2} + \cdots, \end{aligned}$$

$$u(x-\Delta x,t)=u(x,t)-(\Delta x)\frac{\partial u(x,t)}{\partial x}+\frac{1}{2}(\Delta x)^2\frac{\partial^2 u(x,t)}{\partial x^2}+\cdots, \tag{3.52}$$

Thus, we obtain the first partial derivatives with respect to t and x:

$$\begin{aligned}\left(\frac{\partial u(x,t)}{\partial t}\right)&\approx\frac{u(x,t+\Delta t)-u(x,t)}{\Delta t},\\ \left(\frac{\partial u(x,t)}{\partial x}\right)&\approx\frac{u(x+h,t)-u(x,t)}{\Delta x},\end{aligned} \tag{3.53}$$

And the second partial derivative with respect to x is given by

$$\left(\frac{\partial^2 u(x,t)}{\partial x^2}\right)_{x=x}\approx\frac{1}{(\Delta x)^2}[u(x+h,t)+u(x-h,t)-2u(x,t)]. \tag{3.54}$$

Therefore, the diffusion Eq. (3.51) is now converted to

$$\frac{u(x,t+\Delta t)-u(x,t)}{\Delta t}=D\frac{1}{(\Delta x)^2}[u(x+h,t)+u(x-h,t)-2u(x,t)], \tag{3.55}$$

and we can use the following iterative expression for numerical analysis:

$$u(x,t+\Delta t)=u(x,t)+D\frac{\Delta t}{(\Delta x)^2}[u(x+h,t)+u(x-h,t)-2u(x,t)]. \tag{3.56}$$

An important consideration for the numerical calculation is the condition on the position increment, Δx, and the time increment, Δt. In order for computational stability, the two increment parameters must satisfy

$$\Delta x\geq\sqrt{2D\Delta t}. \tag{3.57}$$

The above condition relates to the spectral method discussed in Sect. 3.5.2. From the spectral method, the probability distribution that follows the diffusion equation is Gaussian with the mean zero and the standard deviation $\sigma=\sqrt{2Dt}$. From the spread, the distribution can be expected to have a spread about $\sqrt{2D\Delta t}$ during each time step, Δt, of a numerical computation. The special step, say h, must be larger than the possible spread to allow the distribution to spread. We can also infer that the condition (3.49) is due to the intrinsic property of the "uncertainty" of the Fourier transform [9].

3.6.1 Particle Diffusion

Imagine a number of particles, $\rho(x, t)$, diffuse along a one-dimensional tube. Suppose we have 1,000 freely moving independent particles confined in a tube of length L, and

initially they are all at one end of the tube ($x = 0$), i.e., $\rho(0, 0) = 1{,}000$. For the particle diffusion with the initial condition, we have:

(1) Diffusion equation:

$$\frac{\partial \rho(x,t)}{\partial t} = D\frac{\partial^2 \rho(x,t)}{\partial x^2} \tag{3.58}$$

(2) Initial condition:

$$\rho(0,0) = 1000, \text{ otherwise } \rho(x,0) = 0. \tag{3.59}$$

What is the boundary condition for this problem? It is somewhat tricky. The particles are confined in the one-dimensional box, and there is no particle in and out of the tube. It is quite different from the "fixed number of particles" at the boundary. Although the particles at $x = 0$ initially, they should be able to move freely inside the tube and the number of particles remains the same in the tube. In this case, the boundary condition should be $\frac{\partial \rho(x,t)}{\partial x} = 0$ at both ends ($x = 0$ and L). It specifies that there is no particle exchange across the boundaries. It is analogous to the "free-end" of a wave on a string.

Now, we need to write down the boundary condition, $\frac{\partial \rho(x,t)}{\partial x} = 0$, in equations at $x = 0$ and L. By adding two equations of equation set (3.52) and replacing $u(x, t)$ with $\rho(x, t)$, we obtain

$$\frac{\partial \rho(x,t)}{\partial x} = \frac{\rho(x+h,t) - \rho(x-h,t)}{2h}. \tag{3.60}$$

Thus, the proper boundary conditions for no particle passing through at both ends are given by the following equations:

$$\begin{cases} \left.\dfrac{\partial \rho(x,t)}{\partial x}\right|_{x=0} = \dfrac{\rho(h,t) - \rho(-h,t)}{2h} = 0. \\ \left.\dfrac{\partial \rho(x,t)}{\partial x}\right|_{x=L} = \dfrac{\rho(L+h,t) - \rho(L-h,t)}{2h} = 0. \end{cases} \tag{3.61}$$

Using the length and the time increments, $L = N{\cdot}h$, $x = j{\cdot}h$, $t = k{\cdot}\Delta t$, $j = 0,1,2, \ldots$, and $k = 0,1, 2, \ldots$, the boundary conditions (3.61) become

$$\begin{cases} \rho(1,k) - \rho(-1,k) = 0 & \text{at } x = 0. \\ \rho(N+1,k) - \rho(N-1,k) = 0 & \text{at } x = L. \end{cases} \tag{3.62}$$

Because we only define $0 \le x \le L$, we must eliminate $\rho(-1, k)$ and $\rho(N + 1, k)$ from the boundary conditions (3.62). Here is the tricky part. Similar to Eq. (3.56), the difference equation for $\rho(x, t)$ is given by

$$\rho(j, k+1) = \rho(j, k) + R[\rho(j+1, k) + \rho(j-1, k) - 2\rho(j, k)] \tag{3.63}$$

where $R = D\Delta t/h$.

From Eqs. (3.63), we obtain

$$\begin{cases} \rho(0, k+1) = \rho(0, k) + R[\rho(1, k) + \rho(-1, k) - 2\rho(0, k)] \\ \rho(N, k+1) = \rho(N, k) + R[\rho(N+1, k) + \rho(N-1, k) - 2\rho(0, k)] \end{cases} \tag{3.64}$$

Combining the equations sets (3.62) and (3.64), we can eliminate $\rho(-1, k) = \rho(1, k)$ and $\rho(N + 1, k) = \rho(N - 1, k)$, and the boundary conditions becomes

$$\begin{cases} \rho(0, k+1) = \rho(0, k) + 2R[\rho(1, k) - \rho(0, k)]. \\ \rho(N, k+1) = \rho(N, k) + 2R[\rho(N-1, k) - \rho(N, k)]. \end{cases} \tag{3.65}$$

Figure 3.12 lists the VBA code for the particle diffusion analysis where we assume $D = 0.5$, $L = 25$, $h = 0.5$, $\Delta t = 0.1$. The initial condition is given by Eq. (3.59), and the boundary condition is given by Eq. (3.65).

Figure 3.13 shows the number of particles as a function of position in the cylinder at $t = 5$, 15, and 50. The data dots are outputs from the VBA code. The broken lines are computed distributions using Eq. (3.42) and the obtained number of particles at $x = 0$. These distributions are Gaussian distributions (3.42) at fixed times. The tube length is 25 but Fig. 3.13 shows only part of the tube: $0 \leq x \leq 20$.

3.6.2 Heat Conduction

Imagine a metal rod of length, L, has uniform temperature at 100°C initially. The rod is then in thermal contact with a heat reservoir of temperature 0 °C at each end. What is the temperature distribution, $T(x, t)$, inside the rod at later time? The following heat equation describes the problem:

(1) Heat equation:

$$\frac{\partial T(x, t)}{\partial t} = D\frac{\partial^2 T(x, t)}{\partial x^2} \tag{3.66}$$

where D is the thermal diffusivity.

(2) Initial condition: $T(x, 0) = 100$ for $0 < x < L$, and T(0,0) = T(L, 0) = 0.

```
Sub ParticleDiff()
Cells(1, 1) = "One-dimensional particle diffusion in a 1-dim box"
'Place 1000 particles at the origin initially.
'Neuman Boundary condition for particle confinement.
Dim N(51, 301)     '# of particles P(x,t) at x and t. P(1,1) is at x=0 and t=0.
    Cells(2, 3) = "Time -->"
    Cells(3, 2) = "Position"
'At t=0, all particles are at the origin:
    N0 = 1000       '# of particles.
    MX = 51         '0<=x<=(MX-1)*h.
    MT = 301        '0<=t<=dt*(MT-1).
    h = 0.5         'Position increment. x=(i-1)*h, and i=1 to MX.
    dt = 0.1        'Time increment. Time=(j-1)*dt and j=1 to MT.
    D = 0.5         'Diffusion constant.
    R = D * dt / h ^ 2      'h>=SQRT(2*D*dt) = 0.316, and it is acceptable.
'Position label:
   For i = 1 To MX
      Cells(3 + i, 2) = (i - 1) * h
   Next i
'Time label:
   For j = 1 To MT
      Cells(2, 3 + j) = (j - 1) * dt
   Next j
'Initialize all P(i, j):
    For i = 1 To MX
        For j = 1 To MT
            N(i, j) = 0
        Next j
    Next i
'Place N particles at x=0 and t=0:
        N(1, 1) = N0
 For k = 2 To MT
   N(1, k) = N(1, k - 1) + 2 * R * (N(2, k- 1) - N(1, k - 1))        'B.C. at x=0
     For j = 2 To MX - 1
   N(j, k) = N(j, k - 1) + R * (N(j + 1, k - 1) + N(j - 1, k - 1) - 2 * N(j, k - 1))
     Next j
   N(MX, k) = N(MX, k - 1) + 2 * R * (N(MX - 1, k- 1) - N(MX, k - 1))  'B.C. at x=MX
 Next k
'Outputs:
    For j = 1 To MX
        For k = 1 To MT
            Cells(3 + j, 3 + k) = N(j, k)
        Next k
    Next j
End Sub
```

Fig. 3.12 VBA code for one-dimensional particle diffusion ($N = 1000$)

(3) Boundary condition: $T(0, t) = \mathrm{T}(L, t) = 0$.

Figure 3.14 lists the VBA code for the thermal analysis where we assume $D = 1$, $L = \pi$, $\Delta x = 0.1$, and $\Delta t = 0.01$.

Figure 3.15 shows the temperature distribution $T(x, t)$ computed at $t = 0$, 0.05, 0.15, 0.30, and 0.60. The temperature distribution approaches to a sine-curve as heat transfer progresses.

We also computed the numerical values of the analytical solution discussed below by taking the first 4 terms ($n = 4$) at times selected in Fig. 3.15. The computed numerical values (shown in dots) are superposed onto each curve in Fig. 3.15.

Figures 3.16 and 3.17 show the time dependence of temperature $T(t)$ and $\ln(T(t))$ at the midpoint $x = 1.57$. Referring to the analytical solution given by Eq. (3.67) shown below, the temperature stays at the initial value 100°C for a short while, then the temperature

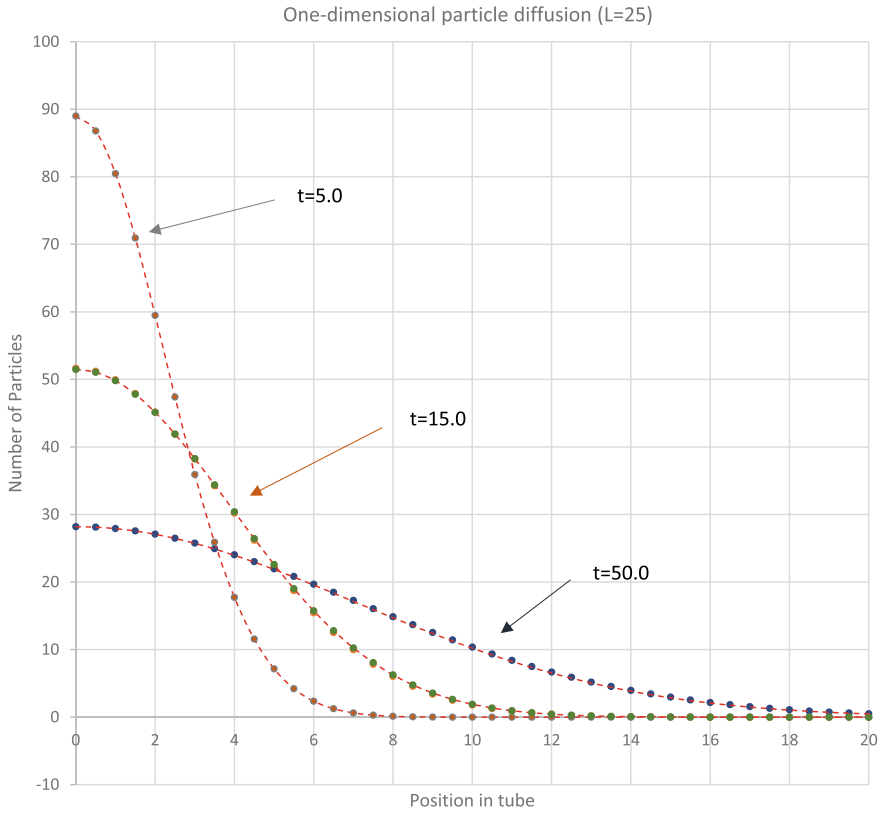


Fig. 3.13 Particle diffusion in a cylinder

starts decreasing "multi-exponentially" with the dominant term of $n = 1$. The higher terms decrease rapidly.

3.6.3 Analytical Solution of Heat Equation

The analytical solution of the heat equation and the given initial and the boundary conditions is given by

$$T(x,t) = \frac{200}{\pi} \sum_{n=1}^{\infty} \left[\frac{1-(-1)^n}{n} \right] e^{-n^2 t} \sin(\pi x) \tag{3.67}$$

where $D = 1$ and $L = \pi$. [10]

Let us derive the solution briefly. By setting $T(x, t) = G(t)X(x)$ to separate the variables x and t, we obtain.

```
Sub HeatEq2()
Cells(1, 1) = "One-dimensional heat equation of T(x, t)"
'1D heat conduction in a rod between two heat reservoirs of temperature 0-degree.
'Initial condition: T(x, 0)=100 for 0<x<L;  T(0, 0)=0, and T(L, 0)=0. L=rod length.
'Boundary condition: T(0, t)=T(L, t)=0
Dim T(21, 301)                    'Temperature at (x, t), 0<=x<=L and 0<=t<=30.
Dim Theory(21)                    'Evaluation of analytical solution.
    Cells(2, 2) = "Time -->"
    Cells(3, 2) = "Position"
    L = 3.1415192654              'Rod length=pi.
    h = L / 20                    'Space increment=0.15708.
    D = 1                         'Diffusion constant.
    dt = 0.01                     'Time increment.
    R = D * dt / h ^ 2            'Remark h>SQRT(2*D*tau) = 0.14142: acceptable.
'Position label:
    For i = 0 To 20
        Cells(4 + i, 2) = i * h
    Next i
'Time label:
    For j = 0 To 300
        Cells(2, 3 + j) = j * dt
    Next j
'Initialization including boundary condition at x=0 and x=20*h:
    For j = 1 To 300
        For i = 0 To 20
            T(i, j) = 0
            Cells(4 + i, 3 + j) = 0
        Next i
    Next j
       j = 0
           For i = 1 To 19
               T(i, j) = 100
               Cells(4 + i, 3) = 100
           Next i
'Boundary conditions at x=0 and x=L:
    For j = 0 To 300
        T(0, j) = 0
        T(20, j) = 0
        Cells(4, 3 + j) = 0
    Next j
'Run iterative step:
    For j = 1 To 300
        For i = 1 To 19                          'Keep the edge (i=20) at Temp=0.
   T(i, j) = U(i, j - 1) + R * (T(i + 1, j - 1) + T(i - 1, j - 1) - 2 * T(i, j - 1))
   Cells(4 + i, 3 + j) = T(i, j)
        Next i
    Next j
'Compute numerical value of analytical solution at t=0.6:
 For i = 0 To 20
    Theory(i) = 0
      For k = 1 To 9 Step 2
Theory(i) = Theory(i) + (200 / 3.141593) * (2 / k) * Sin(k * i * h) * Exp(-k^2 *
0.6)
      Next k
         Cells(4 + i, 1) = Theory(i)               'Display in Column A.
 Next i
End Sub
```

Fig. 3.14 VBA Code for heat conduction through a rod between two heat reservoirs

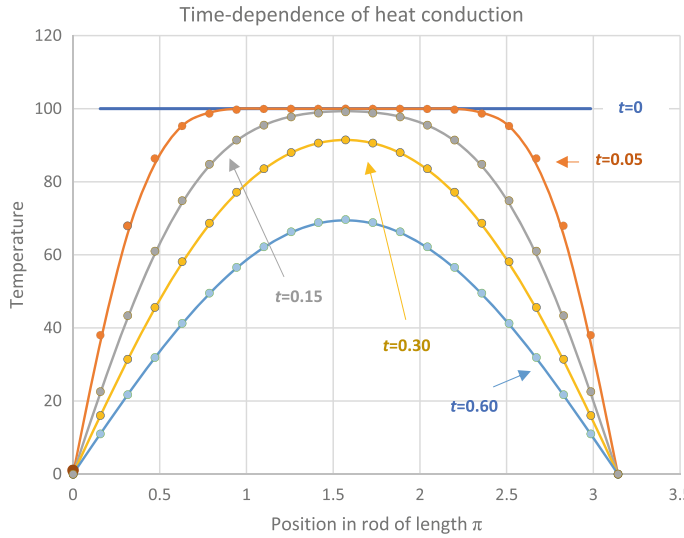


Fig. 3.15 Heat conduction between through a conductive rod with fixed temperatures at both ends

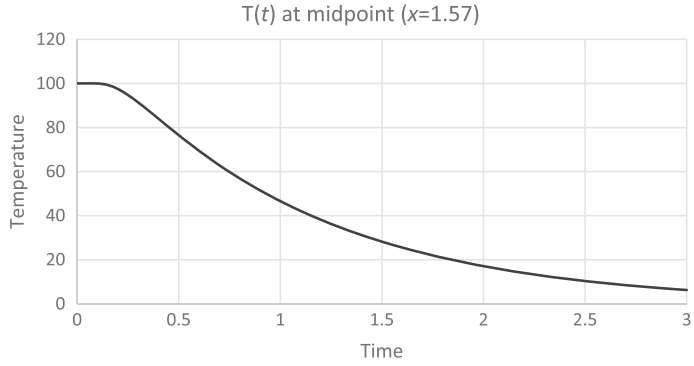


Fig. 3.16 Time dependence of temperature at the midpoint of the rod

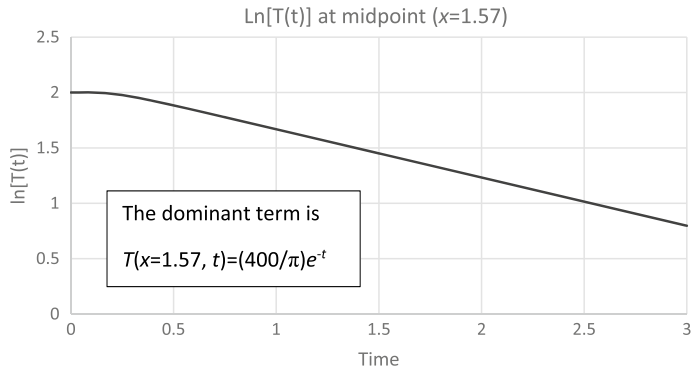


Fig. 3.17 Logarithmic time dependence of temperature at the midpoint of the rod

$X''(x) + \lambda X(x) = 0$ and $G'(t) + \lambda DG(t) = 0$ where λ is the separation constant.

(1) $X''(x) + \lambda X(x) = 0$: using the boundary condition, $X(0) = X(L) = 0$, we obtain

$X_n(x) = c_1 \sin\left(\frac{n\pi}{L}x\right)$ where c_1 is a constant, $\lambda = (n\pi/L)^2$, and $n = 1, 2, 3, \ldots$

(2) $G'(t) + \lambda DG(t) = 0$: using the obtained $\lambda = (n\pi/L)^2$,

$$G_n(t) = c_2 \exp\left[-D\left(\frac{n\pi}{L}\right)^2 t\right].$$

Thus, the temperature $T(x, t)$ is given by

$$T(x,t) = \sum_n G_n(t) X_n(x) = \sum_n A_n e^{-D\left(\frac{n\pi}{L}\right)^2 t} \sin\left(\frac{n\pi}{L}x\right)$$

where $A_n = c_1 c_2$.

If we assume the initial condition $T(x, 0) = f(x)$,

$$T(x,0) = f(x) = \sum_n A_n \sin\left(\frac{n\pi}{L}x\right).$$

This is a sine-Fourier series, and the coefficient A_n is given by

$$A_n = \frac{2}{L}\int_0^L f(x) \sin\left(\frac{n\pi}{L}\right) dx.$$

Therefore,

$$T(x,t) = \frac{2}{L}\sum_n \left[\int_0^L f(x)\sin\left(\frac{n\pi}{L}\right)dx\right] e^{-D(\frac{n\pi}{L})^2 t}\sin\left(\frac{n\pi}{L}x\right).$$

As the special case, $f(x) = 100$ at $x = 0$, $L = \pi$, $D = 1$, we obtain

$$A_n = \frac{2}{L}\int_0^L f(x)\sin\left(\frac{n\pi}{L}\right)dx = \frac{200}{\pi}\left[\frac{1-(-1)^2}{n}\right],$$

and the solution of the heat Eq. (3.66) is given by

$$\begin{aligned} T(x,t) &= \frac{200}{\pi}\sum_n \left[\frac{1-(-1)^n}{n}\right] e^{-n^2 t}\sin(nx) \\ &= \frac{400}{\pi}\left[e^{-t}\sin(x) + \frac{1}{3}e^{-9t}\sin(9x) + \frac{1}{5}e^{-25t}\sin(5x) + \frac{1}{7}e^{-49t}\sin(7x) + \cdots\right]. \end{aligned} \tag{3.68}$$

References

1. Philipse AP (2011) Notes on brownian motion. https://www.semanticscholar.org/paper/Notes-on-Brownian-Motion-Philipse/0f300a40087bfa70882d823ab25b61ef2c47a6d4
2. Hellerman S (2005) Brownian motion and the atomic theory. https://www.physics.princeton.edu/www/legacy/gransasso2006/lectures/einstein.pdf
3. Ronlund M-E (2011) Modeling brownian motion with elastic collisions. https://physics-archive.wooster.edu/JrIS/Files/Ronlund_Web_Article.PDF
4. Pathria RK, Beale PD (2021) Statistical mechanics. Elsevier Science & Technology
5. Islam MA (2004) Einstein–Smoluchowski diffusion equation: a discussion. Phys Scr 70:120. https://iopscience.iop.org/article/10.1088/0031-8949/70/2-3/008/pdf
6. Cross M (2006) Physics 123b statistical physics Fokker–Plank equation. http://www.pmaweb.caltech.edu/~mcc/Ph127/b/index.html
7. Medved A, Davis R, Vasquea P (2020) Understanding fluid dynamics from Langevin and Fokker–Plank equations. www.mdpi.com/2311-5521/5/1/40/html
8. Berntsson F (1999) A spectral method for solving the sideways heat equation. Inverse Problems 15:891. https://iopscience.iop.org/article/10.1088/0266-5611/15/4/305/meta
9. Cho S (2018) Fourier transform and its applications using Microsoft EXCEL®. A Primer IOP Concise Physics. Morgan & Claypool, San Rafael, CA
10. Zill DG (2011) Advanced engineering mathematics, 4th edn. Jones and Bartlett Publishers, Sudbury, MA

Quantum Diffusion Monte Carlo Method 4

There are a limited number of cases where we can find exact solutions through Schrödinger equations [1]. By introducing the imaginary "fake" time, Schrödinger equations become "diffusion equations," to which we can apply Monte Carlo simulations to find their solutions.

In this chapter, first, we solve the first potential problem of quantum mechanics we learn, a particle in one-dimensional infinite potential well, to describe how the Schrödinger equation evolves with imaginary time to approach to a true wave function from an assumed initial wave function. Second, we apply a radioactive decay equation to introduce a way to compute the ground state of a particle in the on-dimensional box. This approach is similar to what we disused in Sect. 2.3. Third, we describe sophisticated random walkers' Monte Carlo algorithms that can be applied to atoms and molecules to find their ground states.

Note: In this chapter, we use an arbitrary unit system or the arbitrary unit (*a.u.*) system where $m = 1$, $\hbar = 1$, and $e = 1$ as well as the unit system of eV and $\mathring{A}$ used in spectroscopy.

4.1 One-Dimensional Infinite Potential Well

4.1.1 Imaginary Time Schrödinger Equation

We focus on the one-dimensional potential well problem to introduce Schrödinger equation with the imaginary time. The time-independent Schrödinger equation is the given by

$$i\hbar\frac{\partial\psi(x,t)}{\partial t} = \hat{H}\psi(x,t) \quad \text{where} \quad \hat{H} = -\frac{1}{2}\nabla^2 + V(x) \tag{4.1}$$

S. Cho, *Monte Carlo Simulations Using Microsoft EXCEL®*,
Synthesis Lectures on Mathematics & Statistics,
https://doi.org/10.1007/978-3-031-33886-1_4

For a real-value potential $V(x)$, the wave function $\psi(x)$ is a real function, and we write the time-independent Schrödinger equation as

$$\left[-\frac{1}{2}\frac{d^2}{dx^2}+V(x)\right]\psi(x)=E\psi(x) \quad \text{where} \quad \int \psi^2 dx=1. \tag{4.2}$$

Now, back to the time-dependent Schrödinger Eq. (4.1), if we replace the time variable t with the imaginary time, $\tau = i\hbar t = it$ in *a.u.*, the Schrödinger equation becomes

$$\frac{\partial\psi(x,\tau)}{\partial\tau}=-\hat{H}\psi(x,\tau). \tag{4.3}$$

Equation (4.3) can be regarded as a diffusion equation with the imaginary time τ. For solving the one-dimensional diffusion equation by iteration, we can utilize the iteration method described in Sect. 3.6. Taking a discrete short imaginary time increment, $\Delta\tau$, the time evolution of the wave function $\psi(\tau)$ can be given by

$$\psi(\tau_{j+1},x)=\psi(\tau_j,x)+\left(\frac{\partial\psi}{\partial\tau}\right)\Delta\tau. \tag{4.4}$$

Combing Eqs. (4.3) and (4.4), we obtain

$$\psi(\tau_j,x)=\psi(\tau_{j-1},x)-\hat{H}\psi(\tau_{j-1},x)\Delta\tau. \tag{4.5}$$

Therefore, starting with a reasonably assumed initial wave function, $\psi(\tau_0, x)$, and applying the iterative method using the iteration expression (4.5), the initial wave function should evolve to the true wavefunction $\psi(\tau, x)$ as $j\to\infty$.

Energy E can be calculated from Eq. (4.2):

$$\begin{aligned} E&=\langle\psi|\hat{H}\psi\rangle=\int dx\psi(x)\left[\left(-\frac{1}{2}\frac{d^2}{dx^2}+V(x)\right)\psi(x)\right]\\ &=\int dx\left[\frac{1}{2}\left(\frac{d\psi}{dx}\right)^2+V(x)\psi^2(x)\right]\to\sum_k(\Delta x)\left[\frac{1}{2}\frac{\psi_k-\psi_{k-1}}{(\Delta x)^2}+V_k\psi_k^2\right]. \end{aligned} \tag{4.6}$$

4.1.2 A Particle in One Dimensional Potential Box

Consider a particle in a one-dimensional infinite potential well:

$$V(x)=0 \text{ if } 0\le x\le 1; \text{ and } V(x)=\infty \text{ if } x\le 0 \text{ and } x\ge 1. \tag{4.7}$$

The time-independent Schrödinger equation in *a.u.* is

$$-\frac{1}{2}\frac{d^2\psi(x)}{dx^2} = E\psi(x) \quad \text{for } 0 \le x \le 1; \text{ and otherwise } \psi(x,t) = 0. \tag{4.8}$$

The exact solution is well known, and given by $\psi_n(x) = \sqrt{2}\sin(n\pi x)$, $n = 1,2, \ldots$, and the ground state wavefunction and the energy ($n = 1$) are given by

$$\psi_1(x) = \sqrt{2}\sin(\pi x) \text{ and } E_1 = \pi/2 = 4.9348\ldots \tag{4.9}$$

Let's take the fake time approach. The Schrödinger equation with the imaginary time, $\tau = it$ in *a.u.*, is given by

$$\frac{\partial\psi(x,\tau)}{\partial\tau} = \frac{1}{2}\frac{\partial^2\psi(x,\tau)}{\partial x^2} \text{ for } 0 \le x \le 1; \text{ and otherwise } \psi(x,t) = 0. \tag{4.10}$$

As the initial wave function, we use a triangular wave function, $\psi(\tau_0, x) = x(1-x)$, which satisfies the boundary condition,

$$\psi(0,\tau) = \psi(1,\tau) = 0$$

and should be relatively close to the exact wave function. We expect this initial triangular wavefunction should evolve the exact same way or as a very similar wave function.

Following the derivation of the iterative expression (3.56), the second derivative with respect to the coordinate can be given by

$$\begin{aligned}\frac{1}{2}\frac{\partial^2\psi(x,\tau)}{\partial x^2} &= \frac{1}{2}\frac{\psi'(x+\Delta x,\tau) - \psi'(x-\Delta x,\tau)}{\Delta x} \\ &= \frac{1}{2}\frac{\psi(x+\Delta x,\tau) + \psi(x-\Delta x,\tau) - 2\psi(x,\tau)}{\Delta x^2}\end{aligned} \tag{4.11}$$

where we used the first derivatives are given by

$$\begin{cases}\psi'(x+\Delta x,\tau) \approx \dfrac{\psi(x+\Delta x,\tau,) - \psi(x,\tau)}{\Delta x}, \text{ and} \\ \psi'(x-\Delta x,\tau) \approx \dfrac{\psi(x,\tau) - \psi(x-\Delta x,\tau)}{\Delta x}.\end{cases} \tag{4.12}$$

Therefore, we obtain

$$\psi(\tau_j, x) = \psi(\tau_{j-1}, x) + \frac{\Delta\tau}{(\Delta x)^2}\big[\psi(\tau_{j-1}, x+\Delta x) + \psi(\tau_{j-1}, x-\Delta x) - 2\psi(\tau_{j-1}, x)\big]. \tag{4.13}$$

Once the normalized wave function is computed at each iteration, the energy E is calculated by using Eq. (4.6). In this case, $V(x) = 0$ and the energy of the particle is

$$E = \frac{1}{2}\sum_{k} \frac{\left[\psi(x_k, \tau_j) - \psi(x_{k-1}, \tau_j)\right]^2}{(\Delta x)}. \tag{4.14}$$

Figure 4.1 lists the VBA code that calculates and the ground state wave function and energy. There are 20 points in the potential well to compute the wave function. The imaginary time increment is taken to $\Delta\tau = 0.0005$ for the computational stability condition (3.57) we discussed in Sect. 3.6.

Figure 4.2 shows the computed wave function at the selected points and the exact solution. The broken line is the exact solution and the orange dots are the computed wave function. The computed ground state energy with this code is $E_1 = 4.926$ after

```
Sub Boxsub()
Cells(1,1) = "1D potential well (0<x<1) using imaginary time diffusion iteration"
    Dim phi(20)                'Wavefunction.
        N = 20                 'Number of data points (x) in the potential well.
        h = 1 / N
        dtau = 0.0005          'Imaginary time interval.
        R = dtau / h ^ 2       'R=0.0005/0.0025 < 1: good for computation stability.
        NP = 100               'Number of iterations.
           Cells(4, 2) = "x"
           Cells(4, 3) = "phi(x)"
           Cells(4, 4) = "Exact Solution"
'Set the initial wave function. Boundary condition is phi(0)=phi(1)=0:
    For j = 0 To N
        x = j * h
        phi(j) = x * (1 - x)
    Next j
        GoSub Normalization     'Normalization of the initial wave function.
'Iteration to evolve the normalized initial wave function:
   For it = 1 To NP
         For j = 1 To N - 1
             PS = phi(j) + R * (phi(j - 1) + phi(j + 1) - 2 * phi(j))
             phi(j) = PS
         Next j
      GoSub Normalization       'Normalization of the wave function at j-iteration:
         E = 0                  'Initialize energy.
         For k = 1 To N
             E = E + ((phi(k) - phi(k - 1)) ^ 2) / (2 * h)
         Next k
   Next it
         For j = 0 To N
             dx = j / N
             Cells(5 + j, 2) = dx
             Cells(5 + j, 3) = phi(j)
             Cells(5 + j, 4) = Sqr(2) * Sin(3.14159 * dx)     'Exact solution.
         Next j
      Cells(3, 2) = "Energy =": Cells(3, 3) = E
Exit Sub
'-----------------------------------------------------------------
Normalization:
    Norm = 0
        For j = 0 To N
            Norm = Norm + phi(j) ^ 2
        Next j
            Norm = 1 / Sqr(h * Norm)
                For j = 1 To N
                    phi(j) = phi(j) * Norm
                Next j
Return
'-----------------------------------------------------------------
End Sub
```

Fig. 4.1 VBA code of a quantum particle in one-dimensional potential box

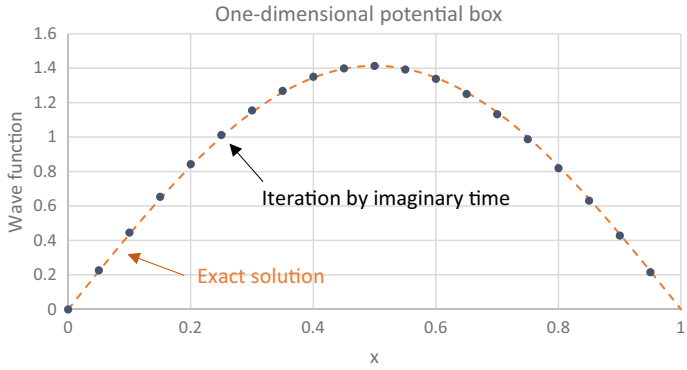


Fig. 4.2 Exact solution of the ground state wave function and computed values

100 iterations. The computed energy and the wave function are in good agreement with the exact solution ($E_0 = 4.9348$). While a shorter imaginary time increment and more iterations will produce even more accurate results, the computation time will be longer as a result.

4.1.2.1 Alternative Approach Using Decay Equation

Comparing the imaginary time Schrödinger Eq. (4.3) with the radioactive decay Eq. (1.5), we could use the idea of finding the decay constant by the "decay or not-decay" method discussed in Sect. 2.3 to the one-dimensional potential well problem. As shown in Fig. 2.10, the "decay constant" should correspond to the energy value of Eq. (4.3). Let's try to determine the ground state energy with the following steps.

(1) Select the potential well of the interval, $|x| < 1/2$, for easier computation.
(2) Create a uniform random distribution of walkers in $|x| < 1/2$ $|x| < 1/2$ as the initial wave function.
(3) The walkers are moved in a step `ds` according to random numbers generated:

```
If Rnd() < 0.5 Then
   x(j) = x(j) + ds
Else
   x(j) = x(j) - ds
End If
```

(4) When a walker steps out the potential well, we simply eliminate it or move it outside the well. Otherwise, keep the walker inside the well.

(5) Repeat steps (1) to (4) to observe the number of walkers inside the well. The number of walkers in the potential well follows the decay Eq. (4.2), and hence we should be able to obtain the ground state energy as the "decay constant."

Figures 4.3 and 4.4 show the VBA code and the result, respectively. In Fig. 4.4, from the 5th step, as the walkers are spreading out of the box, $N(t)$ starts decreasing. We obtained $E_1 = 4.9498$ for the equation of logarithmic function of $[N(t)/N(0)]$ which is a linear function of time.

```
Sub SqWellDiff()
'Potential Box: V (i)= 0 for -w <x < w where w=0.5.
'The ground state energy = 4.9348 with m=1 and Dirac h =1.
'Fake time to simulate the quantum diffusion by random walk.
'Once walkers are |x|>w, they will be eliminated.
'Then, obtain the time-dependence of number of walkers in the box.
  Dim x(10000) As Double  'Positions of walkers.
    N0 = 5000             'Initial number of walkers.
    ds = 0.1              'Step width.
    dtau = ds * ds        'Imaginary time interval.
    MCs = 100             '# of Monte Carlo steps per walker.
    N = N0                '# of walkers after each random walk.
    w = 0.5               'Potential width.
    NoCount = 1000000     'Place walkers out of the potential box.
'Display initial values
  Cells(1, 2) = "Initial # of walkers": Cells(1, 5) = N0
  Cells(2, 2) = "Step length": Cells(2, 5) = ds
  Cells(3, 3) = "Time interval": Cells(3, 5) = dt
  Cells(3, 2) = "Number of Monte Carlo steps per walker": Cells(3, 5) = MCs
  Cells(6, 2) = "Time interval"
  Cells(6, 3) = "N/N0"
  Cells(6, 4) = "Log((N/N0)"
      For i = 1 To N0           'Define initial positions of walkers.
          x(i) = 0.             'All walkers are at the origin.
      Next i
'# of walkers at tau=0:
  Cells(7, 2) = 0
  Cells(7, 3) = N / N0
  Cells(7, 4) = Log(N / N0)
Randomize
For imcs = 1 To MCs
'Run random walks and eliminate walkers if they are outside the well (|x|>0.5).
  For j = 1 To N0
    If x(j) = NoCount Then
       GoTo Stopcount
    Else
       If Rnd() < 0.5 Then
          x(j) = x(j) + ds
       Else
          x(j) = x(j) - ds
       End If
    End If
  If Abs(x(j)) < w Then
     GoTo Stopcount     'Leave the walker if it is withing |w|.
  Else
     x(j) = NoCount     'If the walker is outside the box, move it to NoCount.
     N = N - 1          'Reduce # of walkers by 1.
  End If
     Cells(7 + j, 2) = dtau * j
Stopcount: Next j
     Cells(7 + imcs, 3) = N / N0
     Cells(7 + imcs, 4) = Log(N / N0)
Next imcs
End Sub
```

Fig. 4.3 Computed energy based on diffusion of random walkers

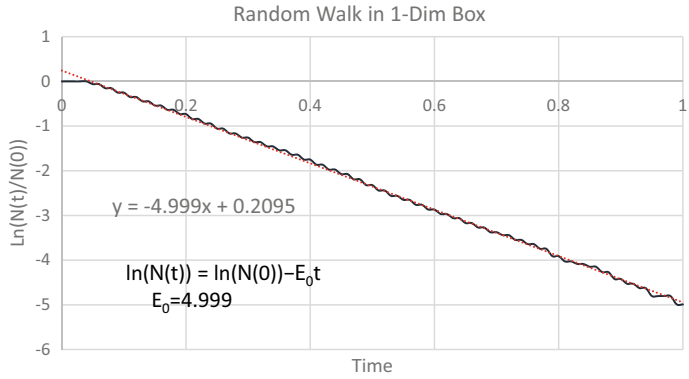


Fig. 4.4 Time dependence of number of walkers in one-dim potential box

The distribution of walkers $N(x)$ remaining in the potential box should represent the wavefunction at a sufficiently long time when random walks are well sampled. However, in this approach, the number of walkers continue decreasing, and we cannot wait for a long time. In order to retain a certain number of walkers, a new set of parameters, e.g., $N(0) = 8{,}000$, $ds = 0.02$, and number of simulation runs MCs $= 500$, is used in the VBA code to obtain an adequately sized $N(x)$. Figure 4.5 is the normalized probability of $n(x)$ where the theoretical wave function given by Eq. (4.7) is onset to the walkers' distribution. The number of remaining walkers is 2051 in this run. Some other methods that retain $n(x)$ need to be developed that require running along a Monte Carlo simulation.

For other potential energy systems, a more sophisticated approach of quantum diffusion method is proposed. We describe the method in Sect. 4.2 that attempts to maintain the

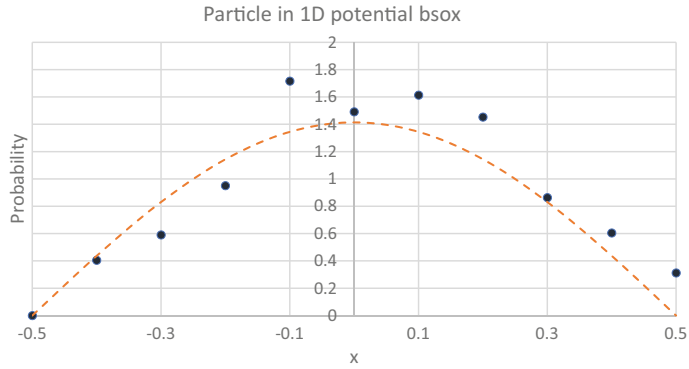


Fig. 4.5 Simulated and exact ground state wave functions

number of walkers and more careful consideration of the asymptotic behavior of the ground state.

Similarity with the heat equation
If we compare the Schrodinger equation of a quantum particle in the one-dimensional potential well with the heat Eq. (3.66) with a specific boundary and initial conditions, the time dependence of the temperature distribution shown in Fig. 3.15 is mathematically equivalent to the wave function of a particle in the one-dimensional potential well. The dominant term of Eq. (3.68) will be the remaining term, after a sufficiently long time, and the space part $X(t)$ of the temperature has the same sine-function dependence of the ground state wave function. In other words, the distribution of the random walkers shows the ground state energy and wave function after many random walks.

4.2 Quantum Diffusion Monte Carlo Method

4.2.1 Basic Idea of Quantum Diffusion Monte Carlo Method

The imaginary time Schrödinger equation is a diffusion equation, and as we discussed in Chap. 3, the random walk can be applied to analyze the diffusion equation. This random walk based approach of a Monte Carlo simulation, called the quantum diffusion Monte Carlo (QDMC) method, can be applied to the Schrödinger equation. The method is also called "quantum random walk method." However, there is a new concept, "quantum walk", which describes the stochastic transition of quantum states. In order to avoid confusion, we use the term, "quantum diffusion Monte Carlo" in this book.

The description below is based on Gould and Tobocjeck [3] which is a simplified version of Kosztin, Faber, and Schulten [4]. The basic idea is to move random walkers with the imaginary time Schrödinger equation while retaining the number of walkers while running the simulation so that the ground state energy as well as its wave function may be estimated accurately.

Let's start from scratch. The one-dimensional time-dependent Schrodinger equation in *a.u.* ($\hbar = 1$, $m = 1$, and $e = 1$) is given by

$$i\hbar\frac{\partial\psi}{\partial t} = \hat{H}\psi \quad \text{where} \quad \hat{H}(x,t) = -\frac{1}{2}\frac{\partial^2}{\partial x^2} + V(x). \tag{4.15}$$

If we know the eigen values and corresponding eigne functions to be $\{E_n$ and $\varphi_n(x)$; $n = 1, 2, \ldots.\}$, the time-dependent wave function is given by

$$\psi(x,t) = \sum_n c_n\varphi_n(x)e^{-iE_nt/\hbar} \text{ where } E_0 < E_1 < E_2 < \ldots \tag{4.16}$$

The quantum diffusion Monte Carlo methods finds the ground state energy and its wavefunction. The discussion below assumes that the potential $V(x)$, and thus the eigen functions are real functions. Using the imaginary time, $\tau = it$, Eqs. (4.15) and (4.16) become

$$\frac{\partial \psi}{\partial \tau} = \left[\frac{1}{2}\frac{\partial^2}{\partial x^2} - V(x)\right]\psi \text{ and } \psi(x,t) = \sum_n c_n \varphi_n(x) e^{-E_n \tau}. \tag{4.17}$$

Therefore, we should obtain $\psi(x,\tau) \rightarrow c_0\varphi_0(x)e^{-E_0\tau}$ as $\tau \rightarrow \infty$ because only the ground state will be dominant. However, depending on the value of E_0, you may not obtain the ground state energy with this method:

$$\begin{aligned} &(1) \text{ If } E_0 > 0, \text{ then } \psi(x, \tau \rightarrow \infty) = 0; \\ &(2) \text{ If } E_0 < 0, \text{ then } \psi(x, \tau \rightarrow \infty) = \infty; \text{ and} \\ &(3) \text{ If } E_0 = 0, \text{ then } \psi(x, \tau \rightarrow \infty) = c_0\varphi_0. \end{aligned} \tag{4.18}$$

In order to establish the steady state of case (3), we add an arbitrary constant reference energy, V_{ref}, which is adjusted so that a computed steady state is close to the ground state; if $V_{\text{ref}} = E_0$. Then $\psi(x, \tau \rightarrow \infty) = c_0\varphi_0$. Notice that the new potential energy, $V(x) - V_{ref}$, must be always positive. Although we can guess V_{ref} to establish the steady state, this procedure would not yield precise value of E_0. Instead, we can estimate E_θ by using the following equation:

$$E_0 = < V > = \frac{\sum_i n_i V(x_i)}{\sum_i n_i} \tag{4.19}$$

where n_{i} is the number of walkers at potion x_{i} at time τ.

Proof With the reference potential V_{ref}, the Schrödinger equation and the asymptotic wave function are

$$\frac{\partial \psi}{\partial \tau} = \frac{1}{2}\frac{\partial^2 \psi}{\partial x^2} - [V(x) - V_{ref}]\psi \tag{4.20}$$

where

$$\psi(x,t) \sim c_0\varphi_0(x)e^{-(E_0 - V_{ref})\tau}. \tag{4.21}$$

Because $\frac{\partial \psi}{\partial x} \rightarrow 0$ as $\tau \rightarrow \infty$, by integrating Eq. (4.20) with respect to x, we obtain

$$\int \frac{\partial \psi}{\partial \tau} dx = -\int [V(x) - V_{ref}]\psi dx. \tag{4.22}$$

From the asymptotic form (4.21), the wave function satisfies $\frac{\partial \psi}{\partial \tau} = -(E_0 - V_{ref})\psi$, and thus, its integral is

$$\int \frac{\partial \psi}{\partial \tau} dx = -\int (E_0 - V_{ref})\psi dx,$$

which should be the same as Eq. (4.22). Therefore, we obtain $E_0 \int \psi dx = \int V(x)\psi dx$, and thus

$$E_0 = \frac{\int V(x)\psi dx}{\int \psi dx}. \tag{4.23}$$

Using the distribution of number of walkers, n_i at $x = xi$, Eq. (4.23) can be also computed from

$$E_0 = \frac{\sum n_i V(x_i)}{\sum n_i} =< V > . \tag{4.24}$$

■

The procedure of the QDMC simulation is now summarized as an extension of the approach described in Sect. 4.1.2.

(1) Place the initial number of walkers, N_0, at the initial positions $\{x_i; i = 1, 2, \ldots, N_0\}$.
(2) Set $V_{ref} = \sum V(x_i)/N_0$.
(3) Pick a walker and move the walker by a step width Δs: $+ \Delta s$ or $- \Delta s$.

Notes:

(i) The time step must be $(\Delta s)^2 \geq 2D\Delta\tau$ for computational stabilization as we discussed in Sect. 3.6. We take $(\Delta s)^2 = \Delta\tau$ because D=1/2 if we take $\hbar$=1 and m=1.
(ii) A step width Δs with the Gaussian modulation with the Gaussian inverse cumulative distribution function of mean 0 may be also used in this step. The corresponding EXCEL's bult-in function is `NORMINV(RND(),0,1)` which we discussed in Sect. 1.4. As we describe in Sect. 4.2.1, there is no significant difference between these two steps.

(4) Compute $\Delta V = V(x) - V_{\text{ref}}$.
(5) Generate a random number r using the `RND()` function, and carry out the Monte Carlo simulation to reach the smaller energy state:
   (i) If $\Delta V > 0$ *and* $r < \Delta V \cdot \Delta\tau$, then remove the walker,
   (ii) If $\Delta V < 0$ *and* $r < -\Delta V \cdot \Delta\tau$, then add another walker at the position, and
   (iii) Otherwise, leave the walker.

   NOTE: The above method is essentially the method of Metropolis & Hastings which we will discuss in Chap. 5.

(6) Repeat steps (3) to (5) for each of the N_0 walkers, and calculate the mean potential energy and the current number of the walkers using

$$V_{ref} = \langle V \rangle = -\frac{a}{N_0 \Delta\tau}(N - N_0). \tag{4.25}$$

NOTE: The parameter a is adjusted so that the number of the walkers will be about the same. It is found that a is about $\hbar$=1.

(7) Repeat steps (3) to (6) until $< V >$ has reached a steady value.

(8) The distribution of walkers represents the wave function, and the normalized wave function is given by

$$\phi(x_i) = \frac{N_i}{\sqrt{\sum_i N_i^2}} \tag{4.26}$$

where N_i is the number of walkers at x_i. The number distribution can be calculated by EXCEL's [`Histogram`] (the frequency count at x_i) of the [`Data Analysis`] menu.

The above discussion is for one-dimensional systems but it can be augmented to three-dimensional and multi-electron systems [4].

4.2.2 Harmonic Oscillator

The best potential system for this simulation is the one-dimensional harmonic oscillator with $V(x) = (1/2)x^2$ in *a.u.* The exact ground state energy is known: $E_0 = 1/2$. Figures 4.6 shows the VBA code we created where we used $N_0 = 5{,}000$, $ds = 0.1$, $dtau = 0.01$, and run Monte Carlo steps 100,000 times. The initial distribution of walkers is a random distribution in $-w \leq x \leq +w$. The asymptotic curve of the energy approaches closely to 0.5 when $w = 2$ as shown in Fig. 4.6. Several different w values are selected for observing its dependence on the result, and we observed no appreciable w-dependence.

Instead of simple "binary" random walks, we may also use a random walk with a Gaussian probability distribution. Figure 4.7 shows the asymptotic behaviors of the ground state energy E_0 of the harmonic oscillator using the `NormInv` function (left) and the $\pm ds$ (right) for the Monte Carlo criterion:

```
Sub Quantum1()
Cells(1, 1) = "One-dimensional harmonic oscillator by QDMC method."
'V (i)= 0.5*x(i)^2           'Potential energy.
    Dim x(40000) As Double
        N0 = 5000            'Desired number of walkers.
        ds = 0.1             'Step width.
        dtau = ds * ds       'Imaginary time step.
        MCs = 10000          '# of Monte Carlo steps per walker.
        N = N0               'Initial number of walkers equal to the desired number.
        w = 2                'Initial width of region for walkers.
        Vref = 0             'Reference potential for convergence.
'Display initial values:
    Cells(2, 2) = "Number of walkers at t=0": Cells(2, 5) = N0
    Cells(3, 2) = "Step length": Cells(3, 5) = ds
    Cells(4, 2) = "Number of Monte Carlo steps per walker": Cells(4, 5) = MCs
  For i = 1 To N                 'Define initial positions of walkers.
     x(i) = (2 * Rnd() - 1) * w  'Walkers are randomly distributed within -1<x<+1.
     Vref = Vref + 0.5 * x(i) ^ 2  'Initial reference potential.
  Next i
        Vref = Vref / N            'Averaged reference potential.
         Esum = 0
            Cells(5, 2) = "MC steps"
            Cells(9, 1) = "MC steps"
            Cells(5, 3) = "N"
            Cells(5, 4) = "Energy"
            Cells(9, 2) = "Energy"
            Cells(5, 5) = "Vref"
    List = 0            'Index to output the energy value after every 1000 MC steps.
Randomize
For imcs = 1 To MCs
'Calculate V-Vref and generate random number using RND().
'Add/remove walkers.
'Walk sub procedure.
    GoSub QDiff1:
        Vave = Vsum / N                              'Mean potential.
        Vref = Vave - (N - N0) / (N0 * dtau)         'New reference energy.
'Data accumulation:
    Esum = Esum + Vave
        If (imcs Mod 100) = 0 Then
            Energy = Esum / imcs
                Cells(6, 2) = imcs
                Cells(6, 3) = N
                Cells(6, 4) = Energy
                Cells(6, 5) = Vref
                Cells(10 + List, 1) = imcs
                Cells(10 + List, 2) = Energy
            List = List + 1
        End If
Next imcs
'Output for Ni-distribution (Histogram):
        Cells(9, 4) = "Bin"
        Cells(9, 5) = "Label of walkers"
        Cells(9, 6) = "x(i)
    For ibin = -50 To 50
        Cells(60 + ibin, 4) = ibin / 10
    Next ibin
        For kk = 1 To N
            Cells(9 + kk, 5) = kk
            Cells(9 + kk, 6) = x(kk)
        Next kk
Exit Sub
'-------------------------------------------------------------
QDiff1:
  Ninit = N                              '# of walkers at beginning of trial.
  Vsum = 0
    For j = Ninit To 1 Step -1
      If Rnd() < 0.5 Then
         x(j) = x(j) + ds
       Else
         x(j) = x(j) - ds
       End If
 Potential = 0.5 * x(j) ^ 2              'Potential function.
 dV = Potential - Vref
       If dV < 0 Then
          If Rnd() < -dV * dtau Then     'Check to add walker.
             N = N + 1                   'Add another walker.
             x(N) = x(j)                 'New walker.
             Vsum = Vsum + 2 * Potential 'Factor of 2 since two walkers at x(i).
           Else
             Vsum = Vsum + Potential     'Leave the walker and add its potential.
           End If
```

Fig. 4.6 VBA code for one-dimensional harmonic oscillator

```
        Else                              'If dV >= 0,
            If Rnd() < dV * dtau Then     'then the walker needs to be removed.
               x(j) = x(N)
               N = N - 1                  'Remove a walker and subtract its potential.
               Vsum = Vsum - Potential
            Else                          'Otherwise, no change.
               Vsum = Vsum + Potential
            End If
        End If
  Next j
 Return
'-------------------------------------------------------------
End Sub
Function FND2CHI(R, A, Alpha, Beta)                          'Laplacian = (2nd derivative)^2.
FND2CHI = FNCHID2(R, A, Alpha, Beta) + 2 * FNCHID(R, A, Alpha, Beta) / R
End Function
'-------------------------------------------------------------------
Function FNF(R, A, Alpha, Beta)                              'f(R)=exp(R/(Alpha*(1+Beta*R))).
FNF = Exp(R / (Alpha * (1 + Beta * R)))
End Function
'-------------------------------------------------------------------
Function FNFD(R, A, Alpha, Beta)                             '1st derivative of FNF.
FNFD = FNF(R, A, Alpha, Beta) / (Alpha * (1 + Beta * R) ^ 2)
End Function
'-------------------------------------------------------------------
Function FNFD2(R, A, Alpha, Beta)                            '2nd derivative of FNF.
FNFD2 = FNFD(R, A, Alpha, Beta) ^ 2 / FNF(R, A, Alpha, Beta) - 2 * Beta * FNF(R, A, Alpha,
Beta) / Alpha / (1 + Beta * R) ^ 3
End Function
'-------------------------------------------------------------------
Function FND2F(R, A, Alpha, Beta)                            'Laplacian = (2nd derivative)^2.
FND2F = FNFD2(R, A, Alpha, Beta) + 2 * FNFD(R, A, Alpha, Beta) / R
End Function
'-------------------------------------------------------------------
Function FNDIST(x, y, z)                                     'Length of a position vector.
FNDIST = Sqr(x ^ 2 + y ^ 2 + z ^ 2)
End Function
'-------------------------------------------------------------------
```

Fig. 4.6 (continued)

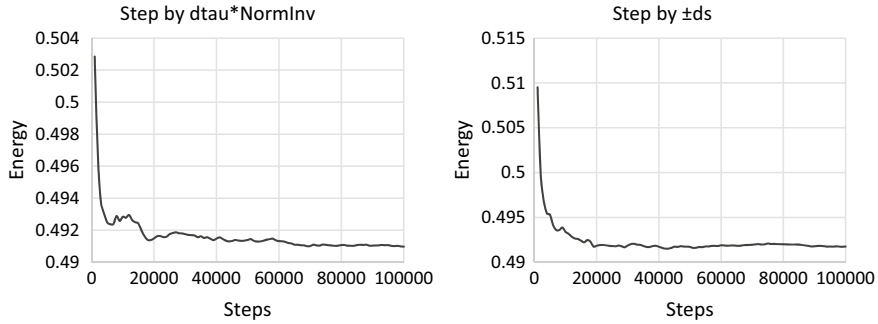


Fig. 4.7 Asymptotic behavior of simulated ground state energy of harmonic oscillator

NormInv:

```
x(j) = x(j) + ds*Application.WorksheetFunction.NormInv(Rnd(),0,1)
```

Discrete step *ds*:

```
If Rnd() < 0.5 Then
      x(j) = x(j) + ds
    Else
```

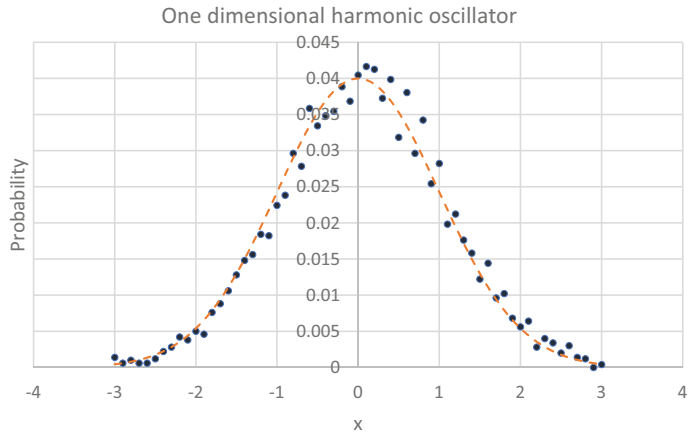


Fig. 4.8 Distribution of random walkers in one-dimensional harmonic oscillator

```
    x(j) = x(j) - ds
End If
```

Both methods converge the energy to about 0.497. Since the exact value is 0.500, the estimated ground state energy is quite accurate although the computation time of the `NormInv` is much longer.

Figure 4.8 shows the normalized distributions of the walkers using [`Histogram`] of the [`Data Analysis`] menu. The exact wave function of the ground state is Gaussian. The broken line is the Gaussian distribution function with the mean value and the standard deviation calculated from the final walker distribution: $< x > = 0$, and $\sigma = 1.003$. Overall, the simulation can extract the wave function of the harmonic oscillator very well.

4.2.3 Three-Dimensional Harmonic Oscillator

The VBA code can be modified for the three-dimensional harmonic oscillator by adding y and z-coordinates. The potential energy for the three-dimensional harmonic oscillator is given by

$$V(x, y, z) = -\frac{1}{2}\left(x^2 + y^2 + z^2\right) \tag{4.27}$$

The Monte Carlo criterion to add/remove walkers is applied to each coordinate independently. The VBA code is listed in Appendix A2.2 where the walkers are uniformly distributed in a given ranges at $t = 0$. In the code, the sub procedure, `QDiff3`, uses the three-dimensional potential energy $V(x, y, z)$, and the random walks are independently

taken in the three directions. The estimated ground state energy from the simulation is 1.495, and the exact energy is 1.50. The three coordinates of the three-dimensional harmonic oscillator are separable and the ground state wavefunction and the energy can be computed by the one-dimensional simulation three times: $E = E_x + E_y + E_z$ and $\psi(x,y,z) = \psi_1(x)\psi_2(y)\psi_3(z)$.

4.2.4 Hydrogen Atom

The potential energy of a hydrogen atom is

$$V(x, y, z) = -\frac{1}{\sqrt{x^2 + y^2 + z^2}}. \tag{4.28}$$

The initial distribution of walkers is set to be a random placement of 9,000 walkers in the intervals of $-w \leq x, y, z \leq +w$ where $w = 2a_0$ and $a_0 = 1.398$ is the Bohr radius in *a.u.* (with $e = 1$, $m = 1$, $\hbar = 1$). The random walks are carried out in the three directions independently with a fixed step of $ds = 0.1$ and $d\tau = 0.01$. The number of the Monte Carlo steps, MCs, are repeated 4,000,000 times for each walker. It took almost 4 days to complete the simulation with a notebook PC! Fig. 4.9 shows the asymptotic behaviors of the ground state energy E_0 of the hydrogen atom, which fluctuates at around -0.4966. This is a fairly accurate correlation with -0.5 of the exact solution. From Fig. 4.9, the number of Mont Carlo steps would be sufficient if MCs is about 20,000 to find the ground state energy.

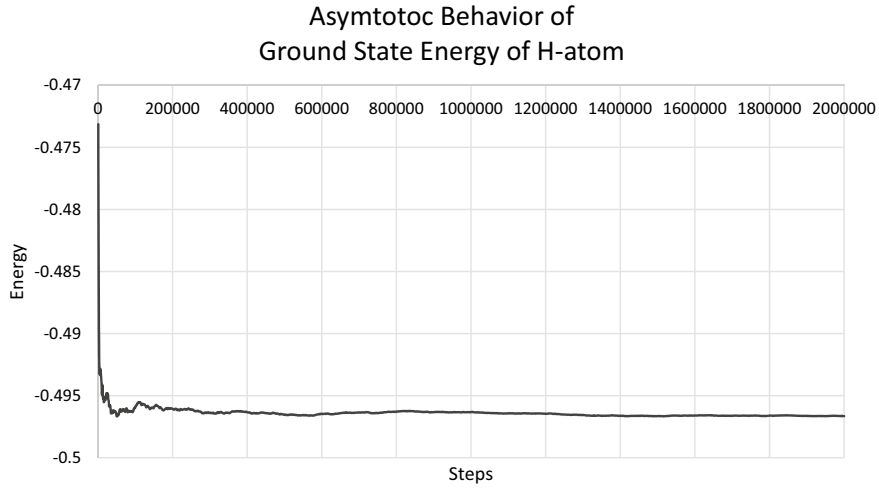


Fig. 4.9 Asymptotic behavior of simulated ground state energy of hydrogen atom

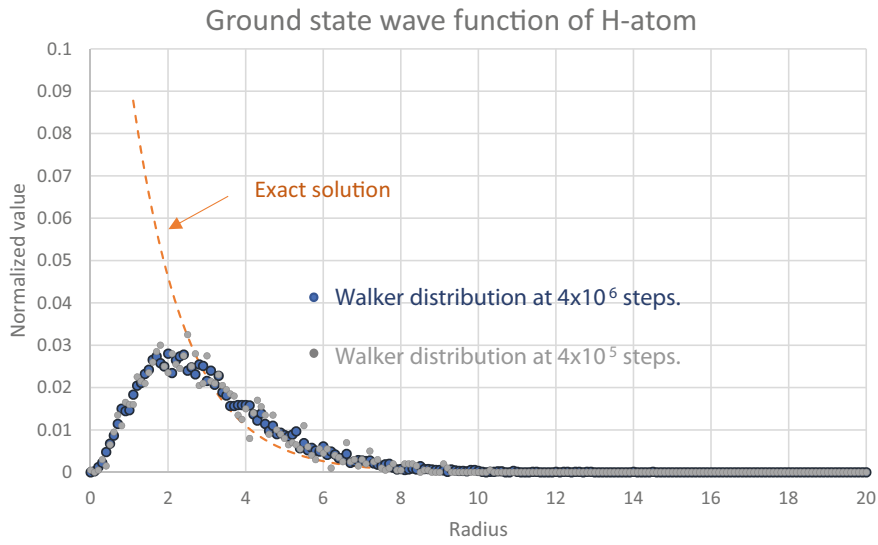


Fig. 4.10 Radial wave function of hydrogen atom

While the ground state energy converges quickly as expected, the wavefunction does not approach to the exact solution, $(1/\pi \cdot {a_0}^{3/2})\exp[-r/a_0]$, in the region of the smaller radius. If the number of walkers and/or the simulation steps are used more, the variation of distribution of the walkers becomes less while the overall distribution pattern has no noticeable difference. Figure 4.10 shows the radial wave functions from simulations of two different steps and the exact solution. The random walk step width defined with `InvNorm` function also outputs very much similar results.

As Kosztin et al. point out that the error in the vicinity of the origin may be reduced by optimizing N, ds, and the number of simulation steps. Because the coordinate variables in the potential energy (4.24) are not independent, independent random changes of the coordinates would take much longer time for the ground state wave function, which is neither sparable, to reach more accurate function form [4].

4.2.5 Helium Atom

The potential energy of a helium atom is

$$V(x_1, y_1, z_1, x_2, y_2, z_2) = -\frac{2}{\sqrt{x_1^2 + y_1^2 + z_1^2}} - \frac{2}{\sqrt{x_2^2 + y_2^2 + z_2^2}} + \frac{1}{\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 (z_1 - z_2)^2}} \tag{4.29}$$

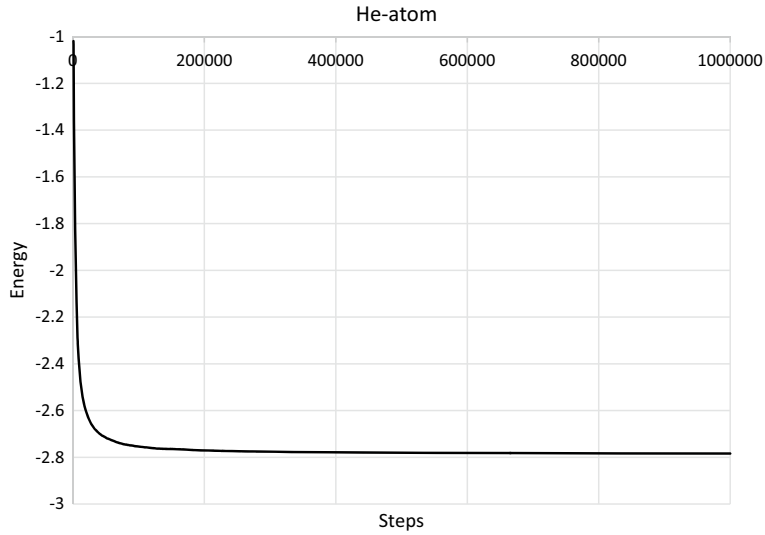


Fig. 4.11 Asymptotic behavior of simulated ground state energy of helium atom

We focus on the ground state energy for the two-electron system, and obtain $E = -2.80$ in *a.u.* or -76.2 eV. The known value is -78.72 eV, and the simulation reasonably estimates the ground state energy.

NOTE: Because EXCEL often had a run time error, "`Unable to get the NormInv property of the WorksheetFunction class`", for the potential energy, we use the fixed step $ds = 0.1$ for random walks which are applied to each walker at each direction of x, y, and z coordinates independently. Figure 4.11 shows the asymptotic behavior of computed energy values where the initial random configuration of walkers has the range of $20a_0$ where a_0 is the Bohr radius. Figure 4.11 is the result of 8,000 walkers and 1,000,000 steps. The number of Mont Carlo steps would be sufficient if the MCs are about 20,000 to find the ground state energy.

4.2.6 Hydrogen Molecule

Figure 4.12 shows the configuration of the hydrogen molecule, and the distances defined are

$$r_{1L,R} = |\vec{r}_1 \pm \frac{1}{2} S\hat{z}| \quad \text{and} \quad r_{12} = |\vec{r}_1 - \vec{r}_2|.$$

The potential energy of a hydrogen molecule is

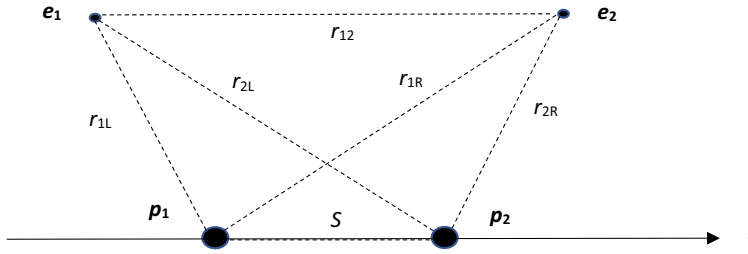


Fig. 4.12 Hydrogen molecule

$$V(\vec{r}) = -\frac{1}{r_{1L}} - \frac{1}{r_{1R}} - \frac{1}{r_{2L}} - \frac{1}{r_{2R}} + \frac{1}{r_{12}} + ER \tag{4.30}$$

where $\vec{r}$ is a collectively expressed notation for the six coordinates of two electrons, and ER is the inter-proton potential energy. With the Born–Oppenheimer approximation, the inter-proton distance can be regarded as the constant compared with the electrons' motions. The known average inter-proton distance is $S = 1.9783$ in *a.u.* or 0.79Å, and thus $ER = 1/S$.

We also do not include small interactions due to the electron spins. The ground state of the two-electrons in the molecule in an antisymmetric spin-singlet state. Then, the wave function must be symmetric by the Pauli's exclusion principle, and it can be chosen to be positive everywhere.

The number of walkers is 8,000 each of which takes 100,000 steps. The simulated ground state energy measures to be approximately $E = -1.144$ in *a.u.* or -31.5 eV. Compared with the known values, -31.866 eV, is a fairly acceptable result. The number of Monte Carlo steps required much more to find the ground state energy, reflecting the complexity of the structure.

4.3 Variational Monte Carlo and Path Integral Monte Carlo Methods

Variational Monte Carlo (VMC) and Path Integral Monte Carlo (PIMC) methods can be applied to compute the ground state energy of many-electron atoms and molecules. The VMC method assumes a trial wave function, and computes the energy from the trial function. The choice of the trial function is the key to obtain a better energy estimation. On the other hand, the PIMC method uses the idea of the imaginary time Schrödinger equation to refine the trial function for more accurate evaluation of the energy. As we

described the simplest example in Sect. 4.1, the imaginary time Schrödinger equation is a "diffusion equation", and the PIMC method creates a diffusion equation with a "drift" term. Both methods can use several common sub procedures to make a trial function. We evaluate the ground state energy of hydrogen molecule from both methods, following the description of these methods found in the excellent book written by Koonin [2] and other publications [5, 6].

4.3.1 Variational Monte Carlo (VMC) Method

The VMC method uses a trial function of the ground state of a hydrogen molecule $\phi(\vec{r})$. Suppose $\psi(\vec{r})$ is the exact ground state wave function. The variational energy is given by

$$E_{\text{var}} = \frac{\langle\phi|\hat{H}|\phi\rangle}{\langle\phi|\phi\rangle} = \frac{\int d\vec{r}\phi^2(\vec{r})\left[\frac{1}{\phi}\hat{H}\phi\right]}{\int d\vec{r}\phi^2(\vec{r})} = \frac{\int d\vec{r}w(\vec{r})\varepsilon(\vec{r})}{\int d\vec{r}\phi^2(\vec{r})} \tag{4.31}$$

where $\vec{r}$ is a collectively expressed notation for the six coordinates of two electrons. From the last expression of the above equation, the variational energy can be interpreted as the average of an energy defined as

$$\varepsilon(\vec{r}) = \frac{1}{\phi}\hat{H}\phi \text{ with a weighting function } w(\vec{r}) = \phi^2. \tag{4.32}$$

where $\hat{H} = -\frac{\hbar^2}{2m}\sum_{i=1,2}\nabla_i^2 + V(\vec{r})$ is the Hamiltonian of a hydrogen molecule and the potential energy has the form of Eq. (4.30).

The VMC method generates a configuration of electrons according to the weighting function ϕ^2, and then computes the average ε over the configuration. The trial function should be a good approximation of the exact wave function $\psi(\vec{r})$. As we discussed in Sect. 4.2.2, the true ground state wave function is symmetric and positive everywhere. For a hydrogen molecule, the trial wave function can be constructed using: (1) wave functions originated from the ground state wave function of the hydrogen atom, which is given by $\varphi(\vec{r}) = \exp(-r/a_0)$ where a_0 is the Bohr radius, and (2) an interaction factor between two electrons. The following trial function is constructed:

$$\phi(\vec{r}_1, \vec{r}_2) = \varphi(\vec{r}_1)\varphi(\vec{r}_2)f(r_{12}) \tag{4.33}$$

where $\varphi(\vec{r}_i) = \exp(-r_{iL}/A) + \exp(-r_{iR}/A)$ represents an electron of $i = 1$ and 2 in each molecular orbit around two protons. The suffixes, R and L, in the trial wave functions are for the two protons as shown in Fig. 4.13. The parameter A is a variational parameter to be determined as described below. The interaction third factor $f(r_{12})$, is given by

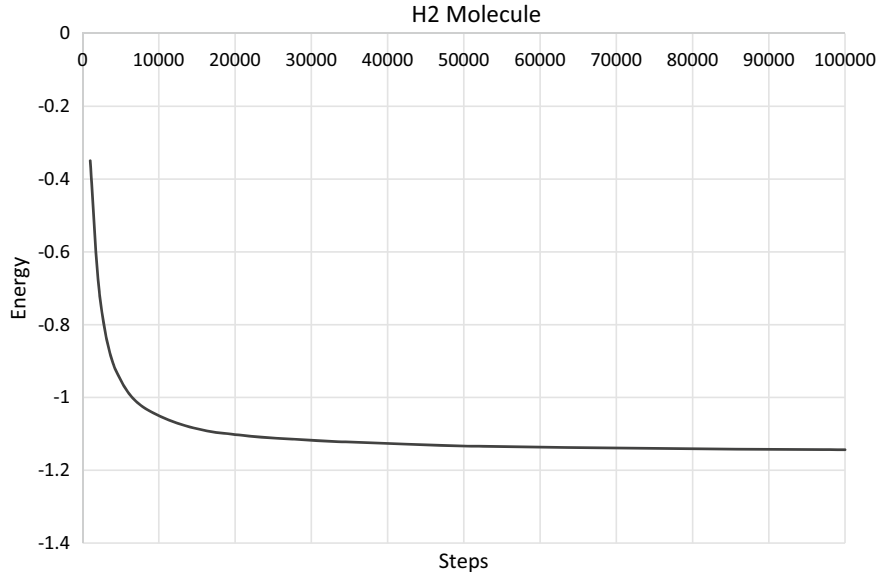


Fig. 4.13 Asymptotic behavior of simulated ground state energy of hydrogen molecule

$$f(r_{12}) = \exp\left[\frac{r_{12}}{\alpha(1+\beta r_{12})}\right] \tag{4.34}$$

with two additional variational parameters α and β [7].

The parameters A and α can be determined with the condition imposed on the trail wave function when each of the electrons is at close proximity of either the proton or electrons become very close to each other ($r_{\mathrm{iL,R}} = 0$, $r_{2\mathrm{L,R}} = 0$, or $r_{12} = 0$). While executing the VMC method, the transition of the energy given by Eq. (4.32), ε, must be consistently smooth with no sudden large variations. This means that the trial function should be smooth and not diverge as $r_{\mathrm{iL,R}} = 0$, $r_{2\mathrm{L,R}} = 0$, or $r_{12} = 0$ where the Coulomb potential energy (4.30) becomes negative infinite. These constraining conditions indicate that the parameter A in Eq. (4.34) should satisfy the following transcendental equation:

$$A = \frac{a_0}{1+\exp(-S/A)} \tag{4.35}$$

where S is the inter-proton distance. The parameter A approaches to $a_0/2$ when S > > 1. The constraints also indicate $\alpha = 2a_0$ [2]. The remaining parameter β needs to be determined independently. The correlation term, $f(r_{12})$, approaches to 1 when two electrons are very close ($r_{12} \sim 0$), where the trail function does not diverge.

Figure 4.14 lists the VBA code using the trial function (4.33). In this program, we use the angstrom and electron-volt based unit. There are three control parameters for computing averages to observe how the energy value changes: the number of ensembles (`NEnsemble`), the number of samples in each ensemble (`Size`), and the number of random samplings for each ensemble (`SamplingFreq`). The code varies the inter-proton distance (`S`) and computes the energy to find the distance at which the energy becomes

```
Sub Variational()
Cells(1, 1) = "Variational Monte Carlo (VMC) methods for the ground state of a H2 molecule"
Dim SET(200, 4000)        'SET of 200 samplings of 200 ensembles, each of which has 20 samples.
Dim Weight(4000)          'Weight of each SET member; 200 ensembles x 20 samples.
Dim Config(200)           'Individual configuration.
Dim CSave(200)            'Save the individual configuration.
'Define physical constants:
  Cells(2, 1) = "Physical constants"
  HMratio = 7.6359         '(h-bar)^2/ mass.
  E2 = 14.409              '(Electron charge)^2.
  Bohr = 0.5299            'Bohr radius in angstrom.
'Define parameters in the wave function:
  Alpha = 2 * Bohr         'Parameter alpha in f(R)= EXP(R/(Alpha*(1+Beta*R))).
  Beta = 20                'Parameter beta in f(R)= EXP(R/(Alpha*(1+Beta*R))).
  '*** Run the code with different Beta-values to find the minimum energy.***
  Cells(3, 1) = "Parameters in wave function"
  Cells(5, 2) = "Parameter Alpha in f(R)= EXP(R/(Alpha*(1+Beta*R)))"
  Cells(5, 11) = Alpha
  Cells(6, 2) = "Parameter Beta": Cells(6, 11) = Beta
'Write control parameters:
  Cells(8, 1) = "Control parameters for simulation"
  Cells(10, 2) = "Metropolis step size (Delta)"
  Cells(10, 7) = 0.4: Delta = Cells(10, 7)
  Cells(11, 2) = "Time step for the VMC calculation (DT)"
  Cells(11, 7) = 0.01: DT = Cells(11, 7)
  Cells(12, 2) = "Number of thermalization steps (NTherm)"
  Cells(12, 7) = 150: NTherm = Cells(12, 7)
  Cells(14, 2) = "Number of ensembles to be run (NEnsemble)"
  Cells(14, 7) = 200: NEnsemble = Cells(14, 7)
  Cells(15, 2) = "Number of samples per ensemble (Size)"
  Cells(15, 7) = 20: Size = Cells(15, 7)
  Cells(16, 2) = "Sampling Frequency (SamplingFreq)"
  Cells(16, 7) = 200: SamplingFreq = Cells(16, 7)
For Ntrials = 1 To 20
  For SS = 50 To 120          'Change the interproton distance from 0.5 to 1.0 angstroms.
    S = SS / 100: S2 = S / 2   'S = interproton separation.
    A = Bohr                   'Set parameter A-value at start.
    Aold = 0 'Parameter A in the wave functions exp(-r1R/A)+exp(-r1L/A) etc. satisfies the
equation below.
      Do Until Abs(A - Aold) < 0.000001
         Aold = A
         A = Bohr / (1 + Exp(-S / Aold))
      Loop
  Cells(4, 2) = "Parameter A in the wave function exp(-r1R/A)+exp(-r1L/A) etc."
  Cells(4, 11) = A
'-------------------------------------------------------------------------------
Randomize
   GoSub StartingConfig                                 'Generate starting configuration.
      For Istep = 1 To NTherm
          GoSub Metropolis
      Next Istep
          SumE = 0                                      'Energy.
   For IEnsemble = 1 To NEnsemble                       'Loop over ensembles.
       EnsembleE = 0                                    'Energy of an ensemble.
         For Istep = 1 To SamplingFreq * Size
             GoSub Metropolis                           'Take a Metropolis step.
         If Istep Mod SamplingFreq <> 0 Then GoTo Skip1 'Measure energy.
            GoSub LocalEnergy                           'Calculate epsilon.
            EnsembleE = EnsembleE + Epsilon
            'Cells(20, 2) = "Epsilon=": Cells(20, 6) = Epsilon
Skip1:   Next Istep
            SumE = SumE + EnsembleE                     'Refresh total sums.
            EnsembleE = EnsembleE / Size                'Ensemble average.
            'Cells(21, 2) = "Eigen value=": Cells(21, 6) = EnsembleE
```

Fig. 4.14 VBA code for variational Monte Carlo method (H_2-molecule)

```
            AvgE = SumE / (IEnsemble * Size)               'Average E.
            U = AvgE + E2 / S                              'Net molecular potential energy.
             'Cells(22, 2) = "Grand average E =": Cells(22, 6) = AvgE
             'Cells(23, 2) = "Net molecular potential =": Cells(23, 6) = U
    Next IEnsemble
       Cells(25, 2) = "S": Cells(25, 3) = "U"
       Cells(26 + SS - 49, 2) = S: Cells(26 + SS - 49, 2 + Ntrials) = U
  Next SS
 Next Ntrials
Exit Sub
' ---------------------------------------------------------------
Metropolis:
  For i = 1 To SamplingFreq                          'SamplingFreq = 200.
     CSave(i) = Config(i)                            'Save current configuration.
     Config(i) = Config(i) + Delta * (Rnd() - 0.5)   'Generate trial configuration.
  Next i
    GoSub PHIandDISTANCE
        Wtrial = Phi * Phi
         If Wtrial / WT < Rnd() Then                 'Reject if Wtrial is too small.
             For i = 1 To SamplingFreq               'Restore the old configuration.
                 Config(i) = CSave(i)
             Next i
          Else:
              WT = Wtrial                            'Refresh the weight if Wtrial is accepted.
          End If
Return
'---------------------------------------------------------------
LocalEnergy:               'Calculate epsilon (i.e., local energy) for a given configuration.
    GoSub PHIandDISTANCE                             'Calculate phi and distance.
      R1DOTR12 = x1 * (x1 - x2) + y1 * (y1 - y2) + z1 * (z1 - z2)
                                                     'Dot products to be used several times below.
      SR12Z = S * (z1 - z2)
      R1LDOTR12 = R1DOTR12 + SR12Z / 2
      R1RDOTR12 = R1DOTR12 - SR12Z / 2
      R2LDOTR12 = R1LDOTR12 - R12 ^ 2
      R2RDOTR12 = R1RDOTR12 - R12 ^ 2
'Kinetic energy:
TPOP = 2 * FND2F(x, y, z, R12, A, Alpha, Beta) / FNF(x, y, z, R12, A, Alpha, Beta)
TEMP = FND2CHI(x, y, z, R1R, A, Alpha, Beta) + FND2CHI(x, y, z, R1L, A, Alpha, Beta)
TPOP = TPOP + TEMP / (CHI1R + CHI1L)
TEMP = FND2CHI(x, y, z, R2R, A, Alpha, Beta) + FND2CHI(x, y, z, R2L, A, Alpha, Beta)
TPOP = TPOP + TEMP / (CHI2R + CHI2L)
TEMP = FNCHID(x, y, z, R1L, A, Alpha, Beta) * R1LDOTR12 / R1L
CROSS = (TEMP + FNCHID(x, y, z, R1R, A, Alpha, Beta) * R1RDOTR12 / R1R) / (CHI1L + CHI1R)
TEMP = -FNCHID(x, y, z, R2R, A, Alpha, Beta) * R2RDOTR12 / R2R
TEMP = TEMP - FNCHID(x, y, z, R2L, A, Alpha, Beta) * R2LDOTR12 / R2L
CROSS = CROSS + TEMP / (CHI2L + CHI2R)
TPOP = TPOP + 2 * FNFD(x, y, z, R12, A, Alpha, Beta) / FNF(x, y, z, R12, A, Alpha, Beta) *
CROSS / R12
TPOP = -0.5 * HMratio * TPOP                         'Plank constant/mass coefficient.
'Potential energy and total energy:
    VPOP = -E2 * (1 / R1L + 1 / R1R + 1 / R2L + 1 / R2R - 1 / R12)
    Epsilon = TPOP + VPOP
Return
'---------------------------------------------------------------
PHIandDISTANCE:            'Calculate wave functions and distances for a given configuration.
 'NOTE: the total wave function is given by phi=(CH1R+CH1L)*(CHI2R+CHI2L)*F.
 x1 = Config(1): y1 = Config(2): z1 = Config(3)          'Positions of electrons.
 x2 = Config(4): y2 = Config(5): z2 = Config(6)
 R1L = FNDIST(x1, y1, z1 + S2, R, A, Alpha, Beta)        'Electron-proton distance.
 R1R = FNDIST(x1, y1, z1 - S2, R, A, Alpha, Beta)
 R2L = FNDIST(x2, y2, z2 + S2, R, A, Alpha, Beta)
 R2R = FNDIST(x2, y2, z2 - S2, R, A, Alpha, Beta)
 R12 = FNDIST(x1 - x2, y1 - y2, z1 - z2, R, A, Alpha, Beta) 'Inter-electron distance.
 F = FNF(x, y, z, R12, A, Alpha, Beta)                   'Electron-electron function.
 CHI1R = FNCHI(x, y, z, R1R, A, Alpha, Beta)             'Electron-proton function.
 CHI1L = FNCHI(x, y, z, R1L, A, Alpha, Beta)
 CHI2R = FNCHI(x, y, z, R2R, A, Alpha, Beta)
 CHI2L = FNCHI(x, y, z, R2L, A, Alpha, Beta)
 Phi = (CHI1R + CHI1L) * (CHI2R + CHI2L) * F             'Total wavefunction.
Return
'---------------------------------------------------------------
StartingConfig:                                      'Generate a starting configuration.
  For i = 1 To SamplingFreq
     Config(i) = (Rnd() - 0.5) * A                   'Initial positions of electrons.
  Next i
     Config(3) = Config(3) + S2                      'Center electron 1 at right.
     Config(6) = Config(6) - S2                      'Center electron 2 at left.
       GoSub PHIandDISTANCE                          'Calculation of wave function.
```

Fig. 4.14 (continued)

```
       WT = Phi^2                                          'Weight.
Return
'-----------------------------------------------------------------
End Sub
'-----------------------------------------------------------------
'Define several functions below:
'-----------------------------------------------------------------
Function FNCHI(x, y, z, R, A, Alpha, Beta)                 'Electron-proton wave function.
  FNCHI = Exp(-R / A)
End Function
'-----------------------------------------------------------------
Function FNCHID(x, y, z, R, A, Alpha, Beta)                '1st derivative of FNCHI.
  FNCHID = -FNCHI(x, y, z, R, A, Alpha, Beta) / A
End Function
'-----------------------------------------------------------------
Function FNCHID2(x, y, z, R, A, Alpha, Beta)                '2nd derivative of FNCHI.
  FNCHID2 = FNCHI(x, y, z, R, A, Alpha, Beta) / A ^ 2
End Function
'-----------------------------------------------------------------
Function FND2CHI(x, y, z, R, A, Alpha, Beta)                'Laplacian = (2nd derivative)^2.
  FND2CHI = FNCHID2(x, y, z, R, A, Alpha, Beta) + 2 * FNCHID(x, y, z, R, A, Alpha, Beta) / R
End Function
'-----------------------------------------------------------------
Function FNF(x, y, z, R, A, Alpha, Beta)                    'f(R)=exp(R/(Alpha*(1+Beta*R))).
  FNF = Exp(R / (Alpha * (1 + Beta * R)))
End Function
'-----------------------------------------------------------------
Function FNFD(x, y, z, R, A, Alpha, Beta)                    '1st derivative of FNF.
  FNFD = FNF(x, y, z, R, A, Alpha, Beta) / (Alpha * (1 + Beta * R) ^ 2)
End Function
'-----------------------------------------------------------------
Function FNFD2(x, y, z, R, A, Alpha, Beta)                    '2nd derivative of FNF.
  FNFD2 = FNFD(x, y, z, R, A, Alpha, Beta) ^ 2 / FNF(x, y, z, R, A, Alpha, Beta) - 2 * Beta *
  FNF(x, y, z, R, A, Alpha, Beta) / Alpha / (1 + Beta * R) ^ 3
End Function
'-----------------------------------------------------------------
Function FND2F(x, y, z, R, A, Alpha, Beta)                    'Laplacian = (2nd derivative)^2.
  FND2F = FNFD2(x, y, z, R, A, Alpha, Beta) + 2 * FNFD(x, y, z, R, A, Alpha, Beta) / R
End Function
'-----------------------------------------------------------------
Function FNDIST(x, y, z, R, A, Alpha, Beta)                    'Length of a position vector.
  FNDIST = Sqr(x ^ 2 + y ^ 2 + z ^ 2)
End Function
'-----------------------------------------------------------------
```

Fig. 4.14 (continued)

minimal. We also run this code for various β-value to find the lowest energy value and the inter-proton distance for the minimum energy. In this VBA code, the titles of sub procedures are intentionally written in bold to emphasize the common sub procedures to the VBA code of the PIMC method. Functions defined in both codes are also common.

Figure 4.15 shows the ground state energy of hydrogen as a function of the inter-proton distance S using the VMC method. The curve fitted to data points is drawn for illustration purposes only. The ground state energy depends on the parameter β. It increases until $\beta = 20$ or so, and then shows much less dependence for $\beta > 20$. The minimum energy we obtained is -30.7 eV at around S $= 0.73$Å with the parameters $\alpha = 1.0598$ and $\beta = 20$. This is fairly close to the known value of -31.688 eV at 0.75 Å.

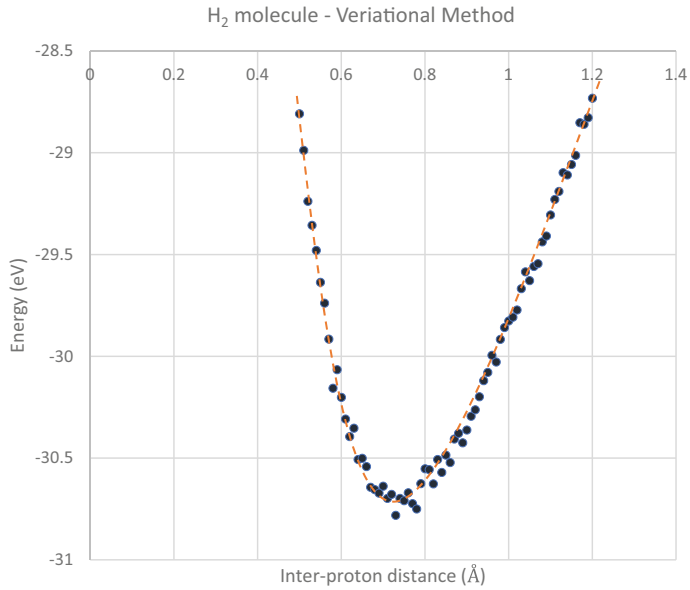


Fig. 4.15 The ground state energy as a function of the inter-proton distance with VMC method

4.3.2 Path Integral Monte Carlo (PIMC) Method

Starting with the same trail function as used in the VMC method, we refine it to the exact state:

$$\psi(\vec{r},t) = -\exp\left[\int_0^t E_n(t')dt'/\hbar\right] e^{-Ht/\hbar}\phi(\vec{r}) \tag{4.36}$$

In the above equation, $E_n(t')$ is an as-yet-undetermined c-number function. As long as the exact state ψ_0. As long as the trial function ϕ is not orthogonal, i.e., $\langle\psi_0|\phi\rangle \neq 0$, $\psi(\vec{r},t) \to \psi_0(\vec{r})$ as $t \to \infty$.

Now, consider a modification of the variational energy

$$E(t) = \frac{\langle\phi|H|\psi(t)\rangle}{\langle\phi|\psi(t)\rangle} = \frac{\int d\vec{r}\phi(\vec{r})\psi(\vec{r},t)\varepsilon(\vec{r})}{\int d\vec{r}\phi(\vec{r})\psi(\vec{r},t)} \tag{4.37}$$

where $\varepsilon(\vec{r})$ is defined as Eq. (4.32). Notice that $E(0) = E_{var}$ is defined by Eq. (4.31), and $E(t)$ should approach to the exact ground state energy E_0 as $t \to \infty$.

Define $G(\vec{r},t) = \phi(\vec{r})\psi(\vec{r},t)$, then Eq. (4.37) becomes $E(t) = \frac{\int d\vec{r}G(\vec{r},t)\varepsilon(\vec{r})}{\int d\vec{r}G(\vec{r},t)}$.

The exact energy $E(t \to \infty)$ can be computed from the average of the energy $\varepsilon(\vec{r})$ over the distribution function $G(\vec{r}, t)$. Therefore, a Monte Carlo evaluation of $E(t)$ requires a "SET" of N configurations $\{\vec{r}_1, \ldots, \vec{r}_N\}$ distributed according to $G(\vec{r}, t)$ through which $E(t)$ can be estimated by the average across the N configurations.

$$E(t) \sim \frac{1}{N} \sum_{i=1}^{N} \varepsilon(\vec{r}_i). \tag{4.38}$$

How do we generate the SET of N configuration? An allocation using the uniform random numbers can be applied. Starting with $G(\vec{r}, 0) = \phi(\vec{r})\psi(\vec{r}, 0) = \left[\phi(\vec{r})\right]^2$. (Note: We assumed the real wave function for the real potential energy), the time evolution can be obtained from $\hbar \frac{\partial \psi}{\partial t} = \left(E_n - \hat{H}\right)\psi$, and $G(\vec{r}, t) = \phi(\vec{r})\psi(\vec{r}, t)$ satisfies the following time-evolution equation:

$$\begin{aligned} &\frac{\partial G(\vec{r}, t)}{\partial t} = \phi(\vec{r}) \frac{\partial \psi(\vec{r}, t)}{\partial t} = \frac{1}{\hbar}\left[E_n(t) - \phi(\vec{r})\hat{H}\frac{1}{\phi(\vec{r})}\right] G(\vec{r}, t) \\ &= \frac{\hbar}{2m}\frac{\partial^2 G(\vec{r}, t)}{\partial \vec{r}^2} - \frac{\partial}{\partial \vec{r}}\left[D(\vec{r})G(\vec{r}, t)\right] - \frac{1}{\hbar}\left[\varepsilon(\vec{r}) - E_n(t)\right] G(\vec{r}, t) \end{aligned} \tag{4.39}$$

where

$$D(\vec{r}) = \frac{\hbar}{m}\frac{1}{\phi(\vec{r})}\frac{\partial \phi(\vec{r})}{\partial \vec{r}} = \frac{\hbar}{m}\frac{\partial \ln \phi(\vec{r})}{\partial \vec{r}} \tag{4.40}$$

is the drift term, which tends to keep $G(\vec{r}, t)$ confined to regions where $\phi(\vec{r})$ is large. The time evolution of G over a short time from t to $t + \Delta t$ can be represented by the path integral formula:

$$G(\vec{r}, t + \Delta t) = \int d\vec{r}' P(\vec{r}, \vec{r}'; \Delta t) G(\vec{r}', t) \tag{4.41}$$

where the integral kernel is given by

$$\begin{aligned} P(\vec{r}, \vec{r}'; \Delta t) &= \exp\left\{-\left[\varepsilon(\vec{r}) - E_n(t)\right]\Delta t/\hbar\right\} \\ &\times \left(\frac{m}{2\pi\hbar\Delta t}\right)^3 \exp\left\{\frac{-\left[\vec{r} - \vec{r}' - D(\vec{r}')\Delta t\right]^2}{2\hbar\Delta t/m}\right\}. \end{aligned} \tag{4.42}$$

In the integral kernel, the first exponential term acts to keep the system in regions of space where ε is most negative, and the second exponential term acts to diffuse the system about $\vec{r}'$ through a normalized Gaussian probability with variance, $\hbar\Delta t/m$, and mean, $D\Delta t$.

A configuration at time t at point generates a contribution to $\vec{r}'$ generates to $G(\vec{r}, t+\Delta t)$ equal to $P(\vec{r}, \vec{r}'; \Delta t)$. This is generalized by placing in the new SET a configuration $\vec{r}$ chosen according to the distribution $\exp\left\{\frac{-[\vec{r}-\vec{r}'-D(\vec{r}')\Delta t]^2}{2\hbar\Delta t/m}\right\}$, and then weighting the importance of this configuration by $\exp\left\{-\left[\varepsilon(\vec{r}) - E_n(t)\right]\Delta t/\hbar\right\}$.

Figure 4.16 lists the VBA code that runs PIMC to find the ground stare energy of hydrogen molecule. In the VBA code, several sub procedure codes to define a trial wave function are common to the VMC code of Fig. 4.14. They are `[Metropolis:]`, `[LocalEnergy:]`, `[PHIandDISTANCE:]`, and `[StartingConfig:]`. Similar to the code of VMS, three parameters for computing averages are used to observe how the energy value changes: the number of ensembles (`NEnsemble`), the number of samples in each ensemble (`Size`), and the number of random samplings (`SamplingFreq`). The PIMC code changes the inter-proton distance (`S`) to obtain the distance that provides the minimum energy. We also run this code for various β-value to find the lowest energy value and the inter-proton distance for the minimum energy.

Figure 4.17 show the ground state energy as a function of the inter-proton distance when $\alpha = 1.0598$ and $\beta = 20$. Compared with the VMC method, the PIMC method yields a better result of approximant -31.7 eV at 0.75 Å, consistent with the known values. The curve fitted to data points is drawn for illustration purposes only. The ground state energy shows some dependence on the parameter β. Similar to the case of the VMC method, it increases until $\beta = 20$ or so, and then shows much less dependence for $\beta > 20$.

```
Sub PIMC()
Cells(1, 1) = "Path Integral Monte Carlo (PIMC) methods for the ground state of a H2 molecule"
Dim SET(6, 50)          'SET of 6 simulations of 10 ensembles, each of which has 5 samples.
Dim Weight(50)          'Weight of each SET member; 10 ensembles x 5 samples = 50.
Dim Config(6)           'Individual configuration.
Dim CSave(6)            'Save the individual configuration.
Dim Drift(6)            'Drift vector of individual configuration.
'Define physical constants:
  Cells(2, 1) = "Physical constants are in the natural units"
  HMratio = 7.6359:     '(h-bar)^2/ mass.
  E2 = 14.409           '(electron charge)^2.
  Bohr = 0.5299         'Bohr radius.
'Define parameters in the wave function:
  Cells(3, 1) = "Parameters in wave function"
  Cells(4, 2) = "Parameter A in the wave function exp(-r1R/A)+exp(-r1L/A) etc ":
  Alpha = 2 * Bohr       'Parameter in f(R)=EXP(R/(Alpha*(1+Beta*R))).
  Beta = 20              'Parameter f(R)=EXP(R/(Alpha*(1+Beta*R))).
  '*** Run the code with different Beta-values to find the minimum energy.***
  Cells(5, 2) = "Parameter Alpha in f(R)=EXP(R/(Alpha*(1+Beta*R)))": Cells(5, 15) = Alpha
  Cells(6, 2) = "Parameter Beta": Cells(6, 15) = Beta
'Write control parameters:
  Cells(8, 1) = "Control parameters for simulation"
  Cells(10, 2) = "Metropolis step size (Delta)"
  Cells(10, 7) = 0.2: Delta = Cells(10, 7)
  Cells(11, 2) = "Time step for the Path Integral Monte Carlo calculation (DT)"
  Cells(11, 7) = 0.02: DT = Cells(11, 7)
  Cells(12, 2) = "Number of thermalization steps (NTherm)"
  Cells(12, 7) = 140: NTherm = Cells(12, 7)
  Cells(13, 2) = "Number of SETs (NSET)"
  Cells(13, 7) = 30: NSET = Cells(13, 7)
  Cells(14, 2) = "Number of ensembles to be run (NEnsemble)"
  Cells(14, 7) = 10: NEnsemble = Cells(14, 7)
  Cells(15, 2) = "Number of samples per ensemble (Size)"
  Cells(15, 7) = 5: Size = Cells(15, 7)
  Cells(16, 2) = "Sampling Frequency (SamplingFreq)"
  Cells(16, 7) = 6: SamplingFreq = Cells(16, 7)
  Cells(25, 2) = "S"
  Cells(25, 3) = "U"
'-------------------------------------------------------------------------------
For Col = 1 To 2
  For SS = 50 To 120
     S = SS / 100: S2 = S / 2      'S = interproton separation.
     A = Bohr                      'Set parameter A-value at start.
     Aold = 0 'Parameter A in the wave function, exp(-r1R/A)+exp(-r1L/A) etc., satisfies the
equation below.
  Do Until Abs(A - Aold) < 0.000001
  Aold = A
  A = Bohr / (1 + Exp(-S / Aold))
  Loop
  Cells(4, 15) = A
'-------------------------------------------------------------------------------
Randomize
'PIMC calculation. Thermalization and data-taking.
  Cells(18, 2) = "PIMC calculation with time step= "
  Cells(18, 6) = DT
      GoSub InitialSET
'Calculate multiplication factors to be used several times (DT=0.01 is the time step):
  HBMDT = HMratio * DT
  SQDT = Sqr(HBMDT)
   For Istep = 1 To NTherm
       GoSub StepSET
   Next Istep
     SumE = 0                                        'Energy.
  For IEnsemble = 1 To NEnsemble                     'Loop over Ensembles.
   EnsembleE = 0
    For Istep = 1 To SamplingFreq * Size
      GoSub StepSET                                  'Take a time step.
      If Istep Mod SamplingFreq <> 0 Then GoTo Skip1  'Same times measure energy
         EnsembleE = EnsembleE + Epsilon
         'Cells(20, 2) = "Epsilon=": Cells(20, 3) = Epsilon
Skip1: Next Istep
   SumE = SumE + EnsembleE                            'Update total sums.
   EnsembleE = EnsembleE / Size                       'Ensemble average.
'Cells(21, 2) = "Eigen value=": Cells(21, 3) = EnsembleE
   AvgE = SumE / (IEnsemble * Size)                   'Average E.
   U = AvgE + E2 / S                                  'Net molecular potential energy.
'Cells(22, 2) = "Grand average E =": ells(22, 4) = AvgE
'Cells(23, 10) = SumA / (IEnsemble * Size * SamplingFreq)
   Next IEnsemble
```

Fig. 4.16 VBA code for path integral Monte Carlo method (H_2-molecule)

```
      Cells(SS - 23, 2) = S: Cells(SS - 23, 20 + Col) = U
  Next SS
Next Col
Exit Sub
' ----------------------------------------------------------------
Metropolis:
For i = 1 To SamplingFreq                              'SamplingFreq = 6.
CSave(i) = Config(i)                                   'Save current configuration.
Config(i) = Config(i) + Delta * (Rnd() - 0.5)          'Generate trial configuration.
Next i
    GoSub PHIandDISTANCE
    Wtrial = Phi ^ 2
        If Wtrial < WT * Rnd() Then                     'Reject if trial phi is too small.
           For i = 1 To SamplingFreq                    'Restore the old configuration.
               Config(i) = CSave(i)
           Next i
       Else:
           WT = Wtrial                                  'Update the weight.
       End If
Return
'---------------------------------------------------------------
LocalEnergy:                'Calculate epsilon (i.e., local energy) for a given configuration.
    GoSub PHIandDISTANCE    'Calculate phi and distance.
R1DOTR12 = x1 * (x1 - x2) + y1 * (y1 - y2) + z1 * (z1 - z2) 'Dot products to be used several
times below.
SR12Z = S * (z1 - z2)
R1LDOTR12 = R1DOTR12 + SR12Z / 2
R1RDOTR12 = R1DOTR12 - SR12Z / 2
R2LDOTR12 = R1LDOTR12 - R12 ^ 2
R2RDOTR12 = R1RDOTR12 - R12 ^ 2
'Kinetic energy
TPOP = 2 * FND2F(R12, A, Alpha, Beta) / FNF(R12, A, Alpha, Beta)
TEMP = FND2CHI(R1R, A, Alpha, Beta) + FND2CHI(R1L, A, Alpha, Beta)
TPOP = TPOP + TEMP / (CHI1R + CHI1L)
TEMP = FND2CHI(R2R, A, Alpha, Beta) + FND2CHI(R2L, A, Alpha, Beta)
TPOP = TPOP + TEMP / (CHI2R + CHI2L)
TEMP = FNCHID(R1L, A, Alpha, Beta) * R1LDOTR12 / R1L
CROSS = (TEMP + FNCHID(R1R, A, Alpha, Beta) * R1RDOTR12 / R1R) / (CHI1L + CHI1R)
TEMP = -FNCHID(R2R, A, Alpha, Beta) * R2RDOTR12 / R2R
TEMP = TEMP - FNCHID(R2L, A, Alpha, Beta) * R2LDOTR12 / R2L
CROSS = CROSS + TEMP / (CHI2L + CHI2R)
TPOP = TPOP + 2 * FNFD(R12, A, Alpha, Beta) / FNF(R12, A, Alpha, Beta) * CROSS / R12
TPOP = -0.5 * HMratio * TPOP                           'Plank constant/mass coefficient.
'Potential energy and total energy
VPOP = -E2 * (1 / R1L + 1 / R1R + 1 / R2L + 1 / R2R - 1 / R12)
Epsilon = TPOP + VPOP
Return
'---------------------------------------------------------------
PHIandDISTANCE:             'Calculate wave functions and distances for a given configuration.
'NOTE: the total wave function is given by phi=(CH1R+CH1L)*(CHI2R+CHI2L)*F.
x1 = Config(1): y1 = Config(2): z1 = Config(3)          'Positions of electrons.
x2 = Config(4): y2 = Config(5): z2 = Config(6)
R1L = FNDIST(x1, y1, z1 + S2)                           'Electron-proton distance.
R1R = FNDIST(x1, y1, z1 - S2)
R2L = FNDIST(x2, y2, z2 + S2)
R2R = FNDIST(x2, y2, z2 - S2)
R12 = FNDIST(x1 - x2, y1 - y2, z1 - z2)                 'Inter-electron distance.
F = FNF(R12, A, Alpha, Beta)                            'Electron-electron function.
CHI1R = FNCHI(R1R, A, Alpha, Beta)                      'Electron-nucleus function.
CHI1L = FNCHI(R1L, A, Alpha, Beta)
CHI2R = FNCHI(R2R, A, Alpha, Beta)
CHI2L = FNCHI(R2L, A, Alpha, Beta)
Phi = (CHI1R + CHI1L) * (CHI2R + CHI2L) * F             'Total wavefunction.
Return
'---------------------------------------------------------------
StepSET:
Ebar = 0: Wbar = 0                                      'Zero E and weight sum.
   For ISET = 1 To NSET                                 'Loop over SET.
       For k = 1 To SamplingFreq                        'Acquire a configuration.
           Config(k) = SET(k, ISET)
       Next k
GoSub DriftVector
   For k = 1 To SamplingFreq
       v = Rnd()
       ETA = Application.WorksheetFunction.NormInv(v, 0, 1) 'Generate a gaussian random number.
       Config(k) = Config(k) + Drift(k) + ETA * SQDT
    Next k
GoSub LocalEnergy                                       'Calculate new local energy.
'Weight(ISET) = Weight(ISET) * Exp(-Epsilon * DT)
```

Fig. 4.16 (continued)

```
Weight(ISET) = Exp(-Epsilon * DT)
Ebar = Ebar + Weight(ISET) * Epsilon
Wbar = Wbar + Weight(ISET)
       For k = 1 To SamplingFreq
           SET(k, ISET) = Config(k)
       Next k
    Next ISET
Epsilon = Ebar / Wbar                                    'Weighted average energy.
Norm = NSET / Wbar                                       'Renormalize weights.
   For ISET = 1 To NSET
      Weight(ISET) = Norm * Weight(ISET)
    Next ISET
Return
'------------------------------------------------------------------------
DriftVector:
    GoSub PHIandDISTANCE                                  'Calculate the trial function.
'Calculate factors to be used several times for Electron 1.
FACTA = HBMDT * (FNCHID(R1L, A, Alpha, Beta) / R1L + FNCHID(R1R, A, Alpha, Beta) / R1R) /
(CHI1L + CHI1R)
FACTB = HBMDT * (FNCHID(R1L, A, Alpha, Beta) / R1L - FNCHID(R1R, A, Alpha, Beta) / R1R) /
(CHI1L + CHI1R)
FACTE = HBMDT * FNFD(R12, R, Alpha, Beta) / FNF(R12, A, Alpha, Beta) / R12
Drift(1) = FACTA * x1 + FACTE * (x1 - x2)                 'Drift for electron 1
Drift(2) = FACTA * y1 + FACTE * (y1 - y2)
Drift(3) = FACTA * z1 + FACTB * S2 + FACTE * (z1 - z2)
'Calculate factors to be used several times for electron 2
FACTA = HBMDT * (FNCHID(R2L, A, Alpha, Beta) / R2L + FNCHID(R2R, A, Alpha, Beta) / R2R) /
(CHI2L + CHI2R)
FACTB = HBMDT * (FNCHID(R2L, A, Alpha, Beta) / R2L - FNCHID(R2R, A, Alpha, Beta) / R2R) /
(CHI2L + CHI2R)
Drift(4) = FACTA * x2 - FACTE * (x1 - x2)                 'Drift for electron 2.
Drift(5) = FACTA * y2 - FACTE * (y1 - y2)
Drift(6) = FACTA * z2 + FACTB * S2 - FACTE * (z1 - z2)
Return
'---------------------------------------------------------------
InitialSET:
     GoSub StartingConfig                                 'Generate configuration at t=0.
For Istep = 1 To NTherm                                   'Thermalization.
     GoSub Metropolis
Next Istep
   For Istep = 1 To 10 * NSET                             'Generate the SET.
     GoSub Metropolis
         If Istep Mod 10 <> 0 Then GoTo Skip2
             ISET = Istep / 10
                  For k = 1 To SamplingFreq
                       SET(k, ISET) = Config(k)
                  Next k
Skip2:   Next Istep
         For ISET = 1 To NSET                              'Set all weight 1.
             Weight(ISET) = 1
         Next ISET
Return
'---------------------------------------------------------------
StartingConfig:                                          'Generate a starting configuration.
For i = 1 To SamplingFreq
Config(i) = (Rnd() - 0.5) * A                            'Initial positions of electrons
Next i
Config(3) = Config(3) + S2                               'Center electron 1 at right
Config(6) = Config(6) - S2                               'Center electron 2 at left
    GoSub PHIandDISTANCE                                 'Calculation of wave function
    WT = Phi^2                                           'Weight
Return
'---------------------------------------------------------------
End Sub
'---------------------------------------------------------------
'Define several functions below.
'---------------------------------------------------------------
Function FNCHI(R, A, Alpha, Beta)                         'Electron-proton wave function.
FNCHI = Exp(-R / A)
End Function
'---------------------------------------------------------------
Function FNCHID(R, A, Alpha, Beta)                        '1st derivative of FNCHI.
FNCHID = -FNCHI(R, A, Alpha, Beta) / A
End Function
'---------------------------------------------------------------
Function FNCHID2(R, A, Alpha, Beta)                       '2nd derivative of FNCHI.
FNCHID2 = FNCHI(R, A, Alpha, Beta) / A ^ 2
End Function
'---------------------------------------------------------------
```

Fig. 4.16 (continued)

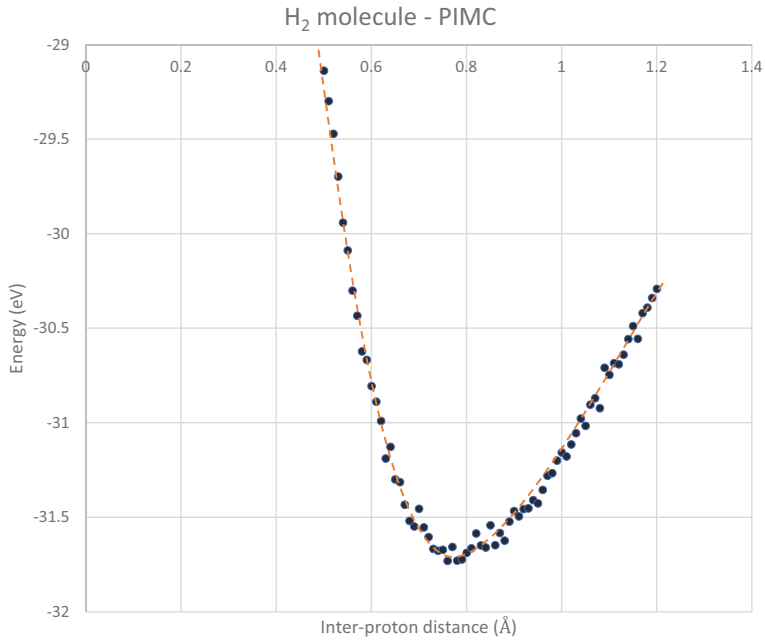


Fig. 4.17 The ground state energy as a function of the inter-proton distance with PIMC method

References

1. Liboff RL (2003) Introductory quantum mechanics. Addison-Wesley, Boston, MA
2. Koonin SE (1987) Computational physics. Benjamin/Cummings, Menlo Park, CA
3. Gould H, Tobochnik J (1996) An introduction to computer simulation methods. Addison-Wesley, Edmonton, Canada
4. Kosztin I, Faber B, Schulten K (1996) Introduction to the diffusion Monte Carlo method. Am J Phys 64:633. https://arxiv.org/abs/physics/9702023
5. Tao P (2016) An introduction to quantum monte carlo methods. A Primer IOP Concise Physics. Morgan & Claypool, San Rafael, CA
6. Traynor CA, Anderson JB (1990) A quantum Monte Carlo calculation of the ground state energy of the hydrogen molecule. J Chem Phys 94:3657. https://aip.scitation.org/doi/abs/10.1063/1.459737
7. Drummond ND, Towler MD, Needs RJ (2004) Jastrow correlation factor for atoms, molecules, and solids. Phys Rev B70:235119. https://arxiv.org/abs/0801.0378

5 Metropolis–Hastings Algorithm for Ising Model

Models in statistical physics have vast degrees of freedom and finding exact solutions are very challenging. One such system is the Ising model, which is simple yet provides essential physics of magnetic systems [1, 2]. Although we can find an exact solution for the one-dimensional Ising model, it requires sophisticated mathematics to obtain exact solutions for two and three-dimensional cases. In the past, many scientists attempted to solve the Ising models unsuccessfully, and we expressed such efforts in a term, "Ising disease." Instead of seeking exact solutions, various approximations and other computational methods such as series expansion of thermodynamic functions and Padé approximation have been proposed to analyze the critical phenomena of the Ising model [3]. Yet another approach to the Ising model is using a Monte Carlo simulation to observe the temperature or the magnetic field dependence of the spin dynamics. One of the well-known Monte Carlo algorithms proposed by Metropolis and Hastings [4] is very useful for the Ising model. Although it was not so easy to perform the simulations when the processing speed of computers were slow, we can now apply simulations by using personal computers at home. Nowadays, even an animated spin dynamics can be observed on the Internet [5].

In this chapter, we introduce the Metropolis–Hastings algorithm. Once we do, we apply the Metropolis–Hastings algorithm to the one-dimensional Ising model and compare the simulation outcomes with the exact solutions to validate the simulation program. In addition, we simulate the two-dimensional Ising model of the square lattice to demonstrate the critical behavior of the Ising model by expanding the simulation program of the one-dimensional case.

S. Cho, *Monte Carlo Simulations Using Microsoft EXCEL®*,
Synthesis Lectures on Mathematics & Statistics,
https://doi.org/10.1007/978-3-031-33886-1_5

Related to the Ising model, we also introduce quantum annealing. Recent development of quantum computation is remarkable. There are two types of quantum computers: quantum gate base and quantum annealing base computations [6]. The latter uses the Ising model to model and solve optimization problems in various fields. We describe the basic idea of quantum annealing and how optimization problems are converted to the Ising model.

5.1 Algorithm of Metropolis and Hastings

The microscopic state of a thermodynamic system is always changing even when the thermodynamic system is in thermal equilibrium with a heat reservoir of temperature $\beta = 1/k_B T$. Suppose a possible microscopic state at temperature β have an internal energy, E_j where $j = 1, 2, 3, \ldots..$, which, in principle, can be calculated from the Hamiltonian of the system. The Gibbs–Boltzmann (GB) distribution function expresses the probability that the internal energy of the thermodynamic system is E_j, and is proportional to $\exp(-\beta E_\mathrm{j})$ [7]. With the GB distribution, the mean value of a thermodynamic variable, Q, is given by

$$\langle Q \rangle = \frac{\sum_j Q_j e^{-\beta E_j}}{\sum_j e^{-\beta E_j}} = \sum_j P_{GB}(\{E_j\}) Q_j \text{ where } = \{Q_{ij};\, j = 1, 2, \ldots\}, \tag{5.1}$$

where

$$P_{GB}(\{E_j\}) = \frac{e^{-\beta E_j}}{Z(\beta)} \tag{5.2}$$

is the Gibbs–Boltzmann (GB) distribution function, and its denominator, $Z(\beta) = \Sigma\exp(-\beta E_\mathrm{j})$ is called the partition function of the system at temperature β.

If we can calculate the partition function, we will be able to obtain the thermodynamics functions. However, because of the complexity of the thermodynamic system, we often cannot calculate the GB distribution function or the thermodynamic functions without approximations. Other than applying approximations, we may find the equilibrium state of a modeled thermodynamic system by simulation.

Because the thermodynamic state at a given temperature takes the lowest energy for the given temperature, we can start the simulation with an arbitrary (randomly chosen) initial state, and then randomly generate a next trial state with a certain transition probability to observe if the energy change makes the system energy lower. Repeat this process until we find a lowest energy state of the system, expecting that a series of the trial-and-error (or hit-or-miss) steps will eventually come to the true equilibrium state. One of the simulation algorithms proposed by Metropolis and Hastings is simple and effective, and especially appropriate to the Ising model. Below is the brief description of their algorithm.

The transition of thermodynamic state is considered to be a Markov process. That is, the present state, collectively denoted as $\{\vec{X}_n\}$, where n indicates the number of present transition step, is determined only by the state of the immediately previous step, $\{\vec{X}_{n-1}\}$. Suppose the transition $\{\vec{X}_{n-1}\} \rightarrow \{\vec{X}_n\}$ occurs with the transition probability denoted as $M_n(\vec{X}_n|\vec{X}_{n-1})$. Similar to the argument of deriving the Poisson distribution in Sect. 1.2.1, the probability to have the configuration of the state, $P_n(\vec{X}_n)$, is given by

$$P_n(\vec{X}_n) = \sum_{\{\vec{X}_{n-1}\}} M_n(\vec{X}_n|\vec{X}_{n-1}) P_{n-1}(\vec{X}_{n-1}). \tag{5.3}$$

Because $\sum_{\{X_n\}} P_n(\vec{X}_n) = 1$, Eq. (5.3) must satisfy the normalization condition

$$\sum_{\{\vec{X}_n\},\{\vec{X}_{n-1}\}} M_n\left(\vec{X}_n|\vec{X}_{n-1}\right) P_{n-1}(\vec{X}_{n-1}) = 1. \tag{5.4}$$

We are aiming to find a method where the system configuration eventually will reach the thermodynamically equilibrium state with given external parameters including temperature and magnetic field. If we find the probability of the stable equilibrium state, $P_{equilibroum}(\vec{X})$, , it should be the GB distribution function (5.2). From Eq. (5.3), once the step reaches the equilibrium state, the probability function, $P_{equilibrium}(\vec{X})$, should be steady with no more change in Eq. (5.3). Therefore, the equilibrium state should satisfy

$$P_{equilibrium}(\vec{X}_n) = \sum_{\{\vec{X}_{n-1}\}} M_n(\vec{X}_n|\vec{X}_{n-1}) P_{equilibrium}(\vec{X}_{n-1}). \tag{5.5}$$

Because the GB distribution function (5.2) is for the equilibrium state, it also satisfies

$$P_{PG}(\vec{X}_n) = \sum_{\{\vec{X}_{n-1}\}} M_n(\vec{X}_n|\vec{X}_{n-1}) P_{PG}(\vec{X}_{n-1}). \tag{5.6}$$

We want to determine $P_{GB}(\vec{X})$ from the above equation. However, with only Eq. (5.6), we cannot determine $P_{GB}(\vec{X})$. We need an additional condition, which is provided by what we call the condition of detailed balancing. Having detail balancing means that the probability of the transition, $M_n(\vec{X}_n|\vec{X}_{n-1})$, and the probability of the backward transition, $M_n(\vec{X}_{n-1}|\vec{X}_n)$, is the same in the equilibrium state:

$$M_n(\vec{X}_{n-1}|\vec{X}_n) P_{PG}(\vec{X}_n) = M_n(\vec{X}_n|\vec{X}_{n-1}) P_{PG}(\vec{X}_{n-1}). \tag{5.7}$$

The transition probability $M_n(\vec{X}_{n-1}|\vec{X}_n)$ proposed by Metropolis and Hastings is

$$M_n(\vec{X}_n|\vec{X}_{n-1}) = \min\left(1,\ e^{-\beta\left(E(\vec{X}_n)-E(\vec{X}_{n-1})\right)}\right) \tag{5.8}$$

where min (1, q) means to take the smaller value between 1 and q. $E(\vec{X}_n)$ is the internal energy of the system of the present state $\{\vec{X}_n\}$. The transition probability (5.8) indeed satisfies the detailed balancing condition.

Proof

(i) If $E(\vec{X}_n) > E(\vec{X}_{n-1})$, then $M_n(\vec{X}_n|\vec{X}_{n-1}) = e^{-\beta\left(E(\vec{X}_n)-EW(\vec{X}_{n-1})\right)}$.

With this condition of the Hamiltonian, Eq. (5.8) becomes $M_n(\vec{X}_{n-1}|\vec{X}_n) = 1$. Thus, we obtain the detailed balancing condition,

$$\frac{M_n(\vec{X}_n|\vec{X}_{n-1})}{M_n(\vec{X}_{n-1}|\vec{X}_n)} = e^{-\beta\left(E(\vec{X}_n)-E(\vec{X}_{n-1})\right)} = \frac{P_{GB}(\vec{X}_n)}{P_{GB}(\vec{X}_{n-1})} \tag{5.9}$$

(ii) If $E(\vec{X}_n) \leq E(\vec{X}_{n-1})$, in a similar manner, we also get the detailed balancing condition.

Because $\exp(-\beta E)$ is the probability that the internal energy of the system is E,

$$e^{-\beta\left(E(\vec{X}_n)-E(\vec{X}_{n-1})\right)} = e^{-\beta E(\vec{X}_n)}/e^{-\beta E(\vec{X}_{n-1})}$$

is the ratio of probabilities of the present state, $\{\vec{X}_n\}$, to the previous state, $\{\vec{X}_{n-1}\}$. Thus, Eq. (5.8) means that we determine that transition probability to be:

1. 1 if $e^{-\beta E(\{\vec{X}_n\})} < e^{-\beta E(\{\vec{X}_{n-1}\})}$; and
2. $e^{-\beta\left(E(\vec{X}_n)-E(\vec{X}_{n-1})\right)}$ if $e^{-\beta H(\vec{X}_n)} \geq e^{-\beta H(\vec{X}_{n-1})}$. ■

If $E(\vec{X}_n) \leq E(\vec{X}_{n-1})$, then $e^{-\beta E(\vec{X}_n)} \leq e^{-\beta E(\vec{X}_{n-1})}$ and the internal energy decreases or remains the same by the transition $\{\vec{X}_{n-1}\} \rightarrow \{\vec{X}_n\}$. This is the preferred transition, and we accept it. On the other hand, if $E(\vec{X}_n) > E(\vec{X}_{n-1})$, the internal energy increases, then $p = e^{-\beta\left(E(\vec{X}_n)-E(\vec{X}_{n-1})\right)}$ is less than 1, and the transition step is accepted with the probability p. This can be done by comparing p with a uniform random number r in the interval [0, 1], and accepting the step if $r < p$.

Computational steps of the Metropolis algorithm are now summarized:

1. Initial microscopic configuration of the system. It is often created by a random configuration.
2. Calculate the transition probability with the following steps (i) to (vi):
   (i). Make a random trial configuration change;

(ii). Compute the change in the energy of the system, $\Delta E = E_{\text{new}} - E_{\text{previous}}$, from the Hamiltonian H;
(iii). $\Delta E \leq 0$, then accept the trial configuration change and go to step 3;
(iv). If $\Delta E > 0$, then compute $p = \exp(-\beta \Delta E)$;
(v). Generate a random number, r, where $0 \leq r \leq 1$; and
(vi). If $r \leq p$, then accept the new configuration; otherwise, keep the previous configuration.

3. Determine value of the thermodynamic variables.
4. Repeat steps 2 and 3 to acquire a sufficient number of configurations (thermalization cycles).
5. Periodically calculate averages over system configurations to observe how the internal energy changes.

In the next section, we describe the Ising model and apply the Metropolis and Hastings algorithm.

5.2 Application to Ising Model

The Ising model is a model of the structure of a magnetic substance where spontaneous magnetic polarization in the same direction giving rise to a macroscopic magnetic field [2, 7]. The model is an array of N-lattice sites, where $n = 1, 2, 3, \ldots, N$ with the periodic boundary condition. Each lattice site has a spin half variable s_i ($i = 1, 2, \ldots., N$): $s_i = +1$ or -1. The energy of the system in a given configuration of N spins, $\{s_i\}$, is given by

$$E_I\{s_i\} \equiv E_s = -\sum_{i \neq j} J_{ij} s_i s_j - B \sum_i s_i \tag{5.10}$$

where J_{ij} is a magnetic coupling constant, J, only for the nearest neighbor spin pairs, otherwise zero. Notice if $J > 0$, the Ising model represents ferromagnetism whereas if $J < 0$, it is anti-ferromagnetism.

The partition function is calculated as

$$Z(B, T) = \sum_{s_1} \sum_{s_2} \cdots \sum_{s_N} \exp[-\beta E_I\{s_i\}] \text{ where } \beta = 1/k_B T. \tag{5.11}$$

The Helmholtz free energy, the system energy, the heat capacity, the magnetization, and the magnetic susceptibility can be calculated by the partition function:

$$F(B, \beta) = -\frac{1}{\beta} \ln Z(B, T) \tag{5.12}$$

$$U(B, \beta) = \langle E_I(B, \beta) \rangle = \frac{1}{Z} \sum_s E_s e^{-\beta E_s} = -\frac{\partial}{\partial \beta} \ln Z \tag{5.13}$$

$$C(B,\beta) = \frac{\partial U(B,\beta)}{\partial T} = -k_B\beta^2 \frac{\partial U(B,\beta)}{\partial \beta}$$
$$= -k_B\beta^2\left[-\frac{1}{Z^2}\frac{\partial Z}{\partial \beta}\sum_S E_s e^{-\beta E_s} - \frac{1}{Z}\sum_S E_s^2 e^{-\beta E_s}\right] \tag{5.14}$$
$$= -k_B\beta^2\left[\langle E_s^2\rangle - \langle E_s\rangle^2\right] \equiv -k_B\beta^2(\Delta E)^2$$

$$\langle M(B,\beta)\rangle = \frac{1}{Z}\sum_s M_s e^{-\beta E_s} = \frac{1}{\beta}\frac{\partial}{\partial B}\ln Z \tag{5.15}$$

$$\chi(B,\beta) = \frac{\partial\langle M\rangle}{\partial B} = -\frac{1}{Z^2}\frac{\partial Z}{\partial B}\sum_S M_s e^{-\beta E_s} - \frac{1}{Z}\sum_S \beta M_s^2 e^{-\beta E_s} \tag{5.16}$$
$$= \beta\left(\langle M^2\rangle - \langle M\rangle^2\right) \equiv \beta(\Delta M)^2$$

$$\text{where } \frac{\partial Z}{\partial B} = \sum_s \beta M_s e^{-\beta E_s}. \tag{5.17}$$

In the computer simulation, we compute the arithmetic average of $< M >$, $< M^2 >$, $< E >$, and $< E^2 >$, from which we compute $(\Delta E)^2$ and $(\Delta M)^2$ to calculate the heat capacity and the susceptibility using Eqs. (5.16) and (5.17).

We now apply the Metropolis algorithm to the Ising model of N spins. The algorithm seeks the lowest energy state by flipping the randomly sampled spin state and to see if the system energy decreases. This approach is particularly suitable because the spin interaction is limited to the nearest neighbor in the Ising model, and the change in the energy can be calculated locally within the nearest neighbor interactions. For the Ising model, the Metropolis algorithm takes the following steps:

1. Select an initial configuration of N spins, $\{\vec{X}_0\} = \{s_{i0}; i = 1, 2, \ldots, N\}$. The simplest initial configuration is the "random" where each spin is choses to be $+1$ or -1 randomly. This can be done easily by generating random numbers. The periodic boundary condition is better to reduce the effect of the size of the system to be simulated.
2. Choose a single spin at random. Flip the spin, and calculate the change in energy of the system, ΔE, and apply the transition probability $p = \exp(-\beta\Delta E)$ referring to Eq. (5.29) for the one-dimensional and Eq. (5.30) for the two-dimensional models.
   (i) If $\Delta E < 0$, then accept the change, and go to step 3.
   (ii) If $\Delta E > 0$, then accept the change with the probability p with the following rule:
      (a) Generate a random number r $(0 \le r \le 1)$ using `RND()`, and if $r \le p$, accept the new spin configuration; and
      (b) Otherwise, retain the previous configuration.
3. Repeat Step 2 (many times).

4. Calculate the averages of the magnetization and the energy, the magnetic susceptibility, and the heat capacity.
5. Output the result.

5.3 One-Dimensional Ising Model

For a one-dimensional Ising model, we can obtain the exact solution of the partition function from which analytical forms of the thermodynamic functions can be obtained without advanced mathematics. For validating the simulation program, we compute the numerical values of the thermodynamic variables from the Metropolis and Hastings algorithm, and compare them with the exact solution.

5.3.1 Exact Solution

We describe the matrix method proposed by Kramers and Wannier in 1941 [8]. Define a 2×2 matrix, P, whose matrix elements are given by

$$\langle s|P|s'\rangle = \exp\left[\beta\left(Jss' + \frac{1}{2}B(s+s')\right)\right].$$

The explicit form of the matrix is

$$P = \begin{bmatrix} \langle +1|P|+1\rangle & \langle +1|P|-1\rangle \\ \langle -1|P|+1\rangle & \langle -1|P|-1\rangle \end{bmatrix} = \begin{bmatrix} e^{\beta(J+B)} & e^{-\beta J} \\ e^{-\beta J} & e^{\beta(J-B)} \end{bmatrix}. \tag{5.18}$$

With the matrix P, the partition function can be written as

$$\begin{aligned} Z(B,T) &= \sum_{s_1}\sum_{s_2}\cdots\sum_{s_N}\exp\left[\beta\sum_{k=1}^{N}\left(Js_ks_{k+1} + \frac{1}{2}B(s_k+s_{k+1})\right)\right] \\ &= \sum_{\{s\}}\langle s_1|P|s_2\rangle\langle s_2|P|s_3\rangle\cdots\langle s_N|P|s_1\rangle = \sum_{s_1}\left\langle s_1|P^N|s_1\right\rangle = Tr\left[P^N\right]. \end{aligned} \tag{5.19}$$

If we diagonalize the matrix P to find its eigenvalues, λ_+ and λ_-, where $\lambda_+ > \lambda_-$, then

$$Tr\left[P^N\right] = \lambda_+^N + \lambda_-^N. \tag{5.20}$$

The two eigen values are solution of $\lambda^2 - 2\lambda e^{\beta J}\cosh(\beta B) + 2\sinh(2\beta B) = 0$, and they are

$$\lambda_{\pm} = e^{\beta J}\left[\cosh(\beta B) \pm \sqrt{\sinh^2(\beta B) + e^{-4\beta J}}\right]. \tag{5.21}$$

Notice that the partition function $Z(B,\beta) = \lambda_+^N + \lambda_-^N = \lambda_+^N\left(1 + (\lambda_-/\lambda_+)^N\right) \to \lambda_+^N$ as $N \to \infty$, and the Helmholtz free energy per spin is given by

$$f(B,\beta) \equiv -\frac{1}{\beta}\ln\lambda_+ = -J - \frac{1}{\beta}\ln\left[\cosh(\beta B) + \sqrt{\sinh^2(\beta B) + e^{-4\beta J}}\right]. \tag{5.22}$$

From the Helmholtz free energy, the magnetization per spin is given by

$$M(B,\beta) = \frac{\partial f}{\partial B} = \frac{\sinh(\beta B)}{\sqrt{\sinh^2(\beta B) + e^{-4\beta J}}}. \tag{5.23}$$

The susceptibility is given by $\chi(B,\beta) = \dfrac{\partial M}{\partial B} = \dfrac{\beta\cosh(\beta B)e^{-4\beta J}}{\left[\sinh^2(\beta B) + e^{-4\beta J}\right]^{3/2}}$. (5.24)

Because the temperature and the magnetic field always appear as βJ and βB in thermodynamic functions, for computational purpose, we define the susceptibility as

$$\tilde{\chi}(B,\beta) \equiv \frac{1}{\beta}\chi(B,\beta) = \frac{\cosh(\beta B)e^{-4\beta J}}{\left[\sinh^2(\beta B) + e^{-4\beta J}\right]^{3/2}}. \tag{5.25}$$

The internal energy, $\tilde{E}(B,\beta) \equiv \beta U(B,\beta)$ is given by

$$\begin{aligned}\tilde{E}(B,\beta) &= -\left[\beta J + \frac{\beta B\sinh(\beta B)\left(\cosh(\beta B) + \sqrt{\sinh^2(\beta B) + e^{-4\beta J}}\right) - 2\beta J e^{-4\beta J}}{\sqrt{\sinh^2(\beta B) + e^{-4\beta J}}\left(\cosh(\beta B) + \sqrt{\sinh^2(\beta B) + e^{-4\beta J}}\right)}\right] \\ &= E_1 + E_2 + E_3\end{aligned} \tag{5.26}$$

where

$$\begin{aligned}E_1 &= -\beta J \\ E_2 &= -\frac{\beta B\sinh(\beta B)}{\sqrt{\sinh^2(\beta B) + e^{-4\beta J}}} \\ E_3 &= \frac{2\beta J e^{-4\beta J}}{\sqrt{\sinh^2(\beta B) + e^{-4\beta J}}\left(\cosh(\beta B) + \sqrt{\sinh^2(\beta B) + e^{-4\beta J}}\right)}\end{aligned}$$

The heat capacity is

$$\tilde{C}(B,\beta) \equiv \frac{1}{k_B} C(B,\beta) = -\beta^2 \frac{\partial U}{\partial \beta} = -\beta^2 \frac{\partial}{\partial \beta}\left(\frac{\tilde{E}}{\beta}\right) = \tilde{E} - \beta \frac{\partial \tilde{E}}{\partial \beta}$$
$$= \left[E_1 - \beta \frac{\partial E_1}{\partial \beta}\right] + \left[E_2 - \beta \frac{\partial E_2}{\partial \beta}\right] + \left[E_3 - \beta \frac{\partial E_3}{\partial \beta}\right]. \tag{5.27}$$

Calculations of these temperature derivatives are straight forward but lengthy in the expressions.

$$\beta \frac{\partial E_1}{\partial \beta} = \beta J \quad E_1 - \beta \frac{\partial E_1}{\partial \beta} = 0,$$

$$\beta \frac{\partial E_2}{\partial \beta} = -\frac{\beta B \sinh(\beta B) + (\beta B)^2 \cosh(\beta B)}{\sqrt{\sinh^2(\beta B) + e^{-4\beta J}}} + \frac{\beta B \sinh(\beta B)\left\{\beta B \sinh(\beta B) - 2\beta J e^{-4J\beta}\right\}}{\left[\sqrt{\sinh^2(\beta B) + e^{-4\beta J}}\right]^3},$$

$$\beta \frac{\partial E_3}{\partial \beta} = \frac{2\beta J(1-4\beta J)e^{-4\beta J}}{\left[\sqrt{\sinh^2(\beta B) + e^{-4\beta J}}\right]^2 + \cosh(\beta B)\sqrt{\sinh^2(\beta B) + e^{-4\beta J}}}$$
$$- \frac{2(\beta J)e^{-4\beta J}}{\left[\left[\sqrt{\sinh^2(\beta B) + e^{-4\beta J}}\right]^2 + \cosh(\beta B)\sqrt{\sinh^2(\beta B) + e^{-4\beta J}}\right]^2}[A],$$

where

$$[A] = 2\beta B \sinh(\beta B)\cosh(\beta B) - 4\beta J e^{-4\beta J}$$
$$+\beta B \sinh(\beta B)\sqrt{\sinh^2(\beta B) + e^{-4\beta J}}$$
$$+\frac{\cosh(\beta B)\left\{\beta B \sinh(\beta B)\cosh(\beta B) - 2\beta J e^{-4\beta J}\right\}}{\sqrt{\sinh^2(\beta B) + e^{-4\beta J}}}. \tag{5.28}$$

5.3.2 Monte Carlo Simulation

For validating our simulation program, we check the magnetic field dependence of thermodynamic functions at a fixed temperature. We also investigate the temperature dependence at nearly zero-field. The exact values of the thermodynamic functions are computed separately from the VBA codes from Eqs. (5.23), (5.25), (5.26), and (5.28) using `Autofill` of EXCEL.

The probability used in the Metropolis algorithm, $p = \exp(-\beta \Delta E)$, for the one-dimensional chain is given by

$$p = e^{-\beta E_I(\{s_{trial}\}) + \beta E_I(\{s_{old}\})} = e^{-2\beta s_i (J \cdot f + B)} \quad \text{where} \quad f = s_{i-1} + s_{i+1}. \tag{5.29}$$

Table 5.1 Possible nearest neighbor spin configurations of one-dimensional Ising model

s_{i-1}- s_i–s_{i+1} ● = ↑: up spin ○ = ↓: down spin	$f = s_{i-1} + s_{i+1}$	$-2\beta s_i(Jf + B)$	$p = e^{-2\beta s_i(J \cdot f + B)}$
$s_i = +1$ (●)			
●-●-●	-2	$-4\beta J - 2\beta B$	$e^{-4\beta J - 2\beta B}$
●-●-○	0	$-2\beta B$	$e^{-2\beta B}$
○-●-○	$+2$	$4\beta J - 2\beta B$	$e^{4\beta J - 2\beta)}$
$s_i = -1$ (○)			
●-○-●	-2	$4\beta J + 2\beta B = -(4\beta J - 2\beta B)$	$e^{4\beta J + 2\beta B} = 1/e^{-(4\beta J + 2\beta B)}$
●-○-○	0	$2\beta B$	$e^{2\beta B} = 1/e^{-2\beta B}$
○-○-○	$+2$	$-4\beta J + 2\beta B = -(4\beta J - 2\beta B)$	$e^{-(4\beta J - 2\beta B)} = 1/e^{4\beta J - 2\beta B}$

There are several possible local spin configurations among three spins, s_{i-1}, s_i, and s_{i+1}. Table 5.1 lists all possible local configurations and the corresponding f-values. In this table, the upward arrow and the downward arrow represent up spins (-1) and down spins ($+1$).

Figure 5.1 shows the VBA code of the Ising chain where the temperature is fixed at $\beta J = 0.1$, which is a constant `JJ` in the code, while varying the magnetic field, βB, which is a variable `B` in the code. The periodic boundary condition, $s_1 = s_N$, is used for reducing the size effect. The VBA code can be also modified to fix the field and vary the temperature, and the temperature dependence with the near zero-field can be obtained easily. The VBA code computes average values of thermodynamic functions after thermalizations of "`SamplingFreq`" times per ensemble of `Nx`-spin chain, and then computes the "ensemble" averages. The number of ensembles is given by "`Size`." The code repeats the simulation "`SIM`" times to compute the average thermodynamic functions of `SIM` runs.

Figures 5.2, 5.3, 5.4, 5.5 show the exact solutions (broken lines) and simulation results (dots) of thermodynamic functions of a system of 3,000 spins (`Totspins`) with the periodic boundary condition. We compute averages of repeated thermalizations (`SamplingFreq`) 30,000 times per ensemble. The number of ensembles of spin systems (`Size`) is 400. The range of magnetic field is $-3.0 \leq \beta B \leq +30.0$ and the temperature is fixed at $\beta J = 0.1$. Simulations using spin chains with these simulation parameters less than listed here imply that higher-order of thermodynamic functions require a greater number of spins, thermalizations and a larger number of ensembles of spin systems. Furthermore, with the same numbers of spins and thermalizations, the averages using more ensembles show less variations. In other words, while the outputs of magnetization and

energy were very much replicated as shown in these figures, susceptibility and heat were not.

Figures 5.6, 5.7, 5.8 show the temperature dependence of the energy, the susceptibility, and the heat capacity with near zero-field ($\beta B = 0.001$). The near-zero field needs to be used because there is no spontaneous magnetization in the one-dimensional Ising model. The system parameters are the same: 3,000 spins with the periodic boundary condition, averages of 400 samples each of which repeated thermalizations 30,000 times. The magnetization is not shown here because of the absence of the spontaneous magnetization.

From Eqs. (5.23)–(5.28), the exact solutions at $B = 0$ are given by

$$\begin{cases} M(\beta, 0) = 0, \\ \tilde{\chi}(\beta, 0) = \dfrac{1}{\beta}\chi(\beta, 0) = e^{2\beta J}, \\ \tilde{E}(\beta, 0) = \beta U(\beta, 0) = -\beta J \tanh(\beta J), \\ \tilde{C}(\beta, 0) = \dfrac{C(\beta, 0)}{k_B} = (\beta J)^2 \mathrm{sech}^2(\beta J). \end{cases}$$

The exact solutions and the simulation outputs are shown in broken lines and dots, respectively. Although the simulations are conducted with a small B-field, their outputs follow over all tendencies of the thermodynamic functions at zero-field.

```
Sub Ising1DS()
Cells(1, 1) = "Metropolis Simulation of the 1D Ising Model."
'S(i)is the spin state +/-1 where i = 1, 2, ..., Nx.
'R(2+F/2, (S+3)/2) is the flip probability where the spin is S, and
'the sum of the n.n. spins is F given by F=0, +/-2 for 1-dim (3 ways):
'2+F/2=3 if F=2, 2+F/2=2 if F=0, and 2+F/2=1 if F=-2.
'Note: (S+3)/2=2 if S=1 and (S+3)/2=1 if S=-1.
'Calculate magnetization (M), susceptibility (Chi), enemy (U), and
'Heat capacity (Cb) vs B-field.
Dim S(4000) As Integer   'Spin lattice S(x).
Dim R(3, 2) 'Flip probab. where the spin is S and the sum of neighboring spins is F.
   Nx = 3000
   Totspins = Nx:      'A spin lattice has Nx spins.
   SIM = 5             '# of running simulation program.
   SamplingFreq = 30000  'Sampling freq. of thermal sweeps at each simulation run.
   Size = 400          '# of ensembles of spin chains to compute the ensemble average.
          Cells(10, 2) = "B"
          Cells(10, 3) = "M per spin"
          Cells(10, 4) = "Theory"
          Cells(10, 5) = "Energy"
          Cells(10, 6) = "Chi"
          Cells(10, 7) = "Cb"
          Count = 0           'For writing the thermodynamic variables.
  JJ = 0.1                    'Spin coupling constant: JJ=J/kT.
  For BB = -30 To 30
      B = BB / 10             'External magnetic field from -3.0 to +3.0 by step 0.1.
      Cells(2, 1) = "Nx=": Cells(2, 2) = Nx: '# of spins in the x-direction: cell B2.
'Display parameters:
      Cells(2, 4) = "JJ=": Cells(2, 5) = JJ
      Cells(3, 4) = "B=": Cells(3, 5) = B
'# of ensembles, # of samples, the sample size, and the current sampling frequency:
```

Fig. 5.1 VBA Code of magnetic field dependence of one-dimensional Ising model

```
      Cells(2, 7) = "SIM": Cells(2, 8) = SIM
      Cells(2, 10) = "# of Ensembles": Cells(2, 11) = Size
      Cells(3, 10) = "Freq": Cells(3, 11) = SamplingFreq
'Names of thermodynamic variables:
      Cells(2, 13) = "TotE="
      Cells(3, 13) = "Mag="
      Cells(2, 17) = "Chi="
      Cells(2, 21) = "Cb="
'Calculate the flip probability for given JJ and B:
      For i = 1 To 3
          R(i, 2) = Exp(-2 * (JJ * (2 * i - 4) + B))     'F=2*i-4.
          R(i, 1) = 1 / R(i, 2)
      Next i
'Cleaning up the spins on screen:
          Cells(5, 1) = ""
          Cells(5, 1).Interior.Color = RGB(255, 255, 255)         'White.
              For i = 1 To Nx
                  Cells(6, i).Interior.Color = RGB(255, 255, 255) 'White.
              Next i
'Run the simulation by "SIM" times for given JJ and B.
'Each run has "SamplingFreq" times of thermalization to calculate thermal variables.
'Repeat this step by "Size" times, and then calculate the averages.
      'Initialize thermodynamic variables:
          SumM = 0        'M
          SumM2 = 0       'M^2
          SumE = 0        'Energy
          SumE2 = 0       'E^2
          SumChi = 0      'Susceptibility
          SumCb = 0       'Specific heat
Randomize
For Irun = 1 To SIM                              'Run this program SIM times.
     Cells(5, 1).Interior.Color = RGB(255, 0, 0)  'Cell in RED.
        'This cell will be displayed "FINAL" once the simulation will be completed.
'Randomize the spin lattice S(i) where i=1 to Nx, and display the result on screen.
'Use the periodic boundary condition.
  For i = 1 To Nx                                          'Horizontal loop only.
     If i > 1 Then
       Im = i - 1
     Else
       Im = Nx
     End If
         If Rnd() < 0.5 Then
           S(Im) = 1
           Cells(6, 2 + i).Interior.Color = RGB(0, 0, 0)        'Black is spin up.
         Else
           S(Im) = -1
           Cells(6, 2 + i).Interior.Color = RGB(255, 255, 255)  'White is spin down.
         End If
   Next i
'Initialize M and E before start ensemble-thermalization at each simulation run:
 EnsembleM = 0       'Magnetization (M) per ensemble.
 EnsembleM2 = 0      'M^2 per ensemble.
 EnsembleE = 0       'Energy (E) per ensemble.
 EnsembleE2 = 0      'E^2 per ensemble.
      For isweep1 = 1 To Size
           For isweep2 = 1 To SamplingFreq 'Loop over sweeps in a ensemble.
               GoSub Thermalization        'Do a thermalization sweep of the lattice.
           Next isweep2
  'Calculate E and M for this lattice after each thermalization:
           Mag = 0           'Zero magnetization for this lattice.
           SumSS = 0         'Zero sum of interaction with right and left neighbors.
              For i = 1 To Nx                           'Horizontal loop.
                 If i > 1 Then
```

Fig. 5.1 (continued)

```
                  Im = i - 1
                Else
                  Im = Nx
                End If
                   Mag = Mag + S(i)                        'Sum of Magnetization.
                   SumSS = SumSS + S(i) * S(Im)            'Sum of interactions.
             Next i
                E = (-JJ * SumSS - B * Mag):               'Energy.
       'Update ensemble sums:
             EnsembleM = EnsembleM + Mag
             EnsembleM2 = EnsembleM2 + Mag ^ 2
             EnsembleE = EnsembleE + E
             EnsembleE2 = EnsembleE2 + E ^ 2
      Next isweep1
'Compute & display ensemble/total E, M, Chi, Cb:
     'Ensemble average:
             EnsembleM = EnsembleM / Size
             EnsembleM2 = EnsembleM2 / Size
             EnsembleE = EnsembleE / Size
             EnsembleE2 = EnsembleE2 / Size
             Chi = Abs(EnsembleM2 - EnsembleM ^ 2)
             Cb = Abs(EnsembleE2 - EnsembleE ^ 2)
      'Update total sums:
             SumM = SumM + EnsembleM
             SumM2 = SumM2 + EnsembleM ^ 2
             SumE = SumE + EnsembleE
             SumE2 = SumE2 + EnsembleE ^ 2
             SumChi = SumChi + Chi
             SumCb = SumCb + Cb
Next Irun
'Display the spin lattice after running program by SIM-times for given B and JJ:
  GoSub Lattice
         'Total average values:
             TotE = SumE / SIM
             M = SumM / SIM
             Chi = SumChi / SIM
             Cb = SumCb / SIM
         'Display total values per spin:
             Cells(2, 14) = TotE / Totspins
             Cells(3, 14) = M / Totspins
             Cells(2, 18) = Chi / Totspins
             Cells(2, 22) = Cb / Totspins
Cells(5, 1).Interior.Color = RGB(255, 255, 255) 'White after completing simulations.
Cells(5, 1) = "FINAL"                           'Simulation completed.
     GoSub Lattice                              'Mapping latest spin map.
        Count = Count + 1                       'Index to write outputs in sequence.
        Cells(10 + Count, 2) = B
        Cells(10 + Count, 3) = Cells(3, 14)
        Cells(10 + Count, 4) = Cells(2, 14)
        Cells(10 + Count, 5) = Cells(2, 18)
        Cells(10 + Count, 6) = Cells(2, 22)
  Next BB
Exit Sub
'-------------------------------------------------------------------------
Thermalization:
'Subroutine to run a Metropolis thermalization sweep of the lattice.
     For i = 1 To Nx
          If i < Nx Then
             Ip = i + 1
          Else
             Ip = 1                                 'i index of "+x" neighbor.
          End If
              If i > 1 Then
```

Fig. 5.1 (continued)

```
                Im = i - 1
              Else
                 Im = Nx                                    'i index of "-x" neighbor.
              End If
           SPIN = S(i)                                      'Spin at site (i).
           F = S(Ip) + S(Im)                                'Sum of 2 neighbor spins.
             If Rnd() > R(2 + F / 2, (3 + SPIN) / 2) Then
                GoTo NoFlip                                 'No spin flip.
             Else
                 S(i) = -SPIN                               'Flip the spin.
             End If
NoFlip: Next i
Return
'-------------------------------------------------------------------------------
Lattice:
  'Display the spin patterns on screen.
     For i = 1 To Nx
        If S(i) = 1 Then
           Cells(6, 2 + i).Interior.Color = RGB(0, 0, 0)       'Black is spin up.
        Else:
           Cells(6, 2 + i).Interior.Color = RGB(255, 255, 255) 'White is spin down.
        End If
     Next i
Return
'-------------------------------------------------------------------------------
End Sub
```

Fig. 5.1 (continued)

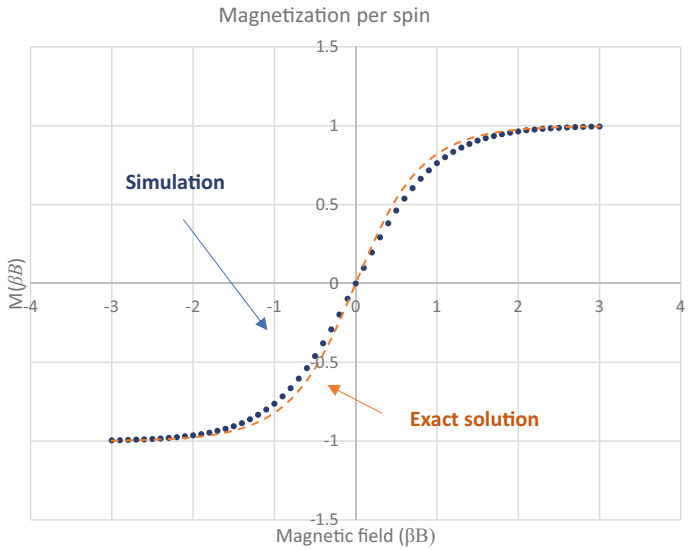


Fig. 5.2 B-field dependence of magnetization of Ising chain ($N = 3{,}000$) at a fix temperature

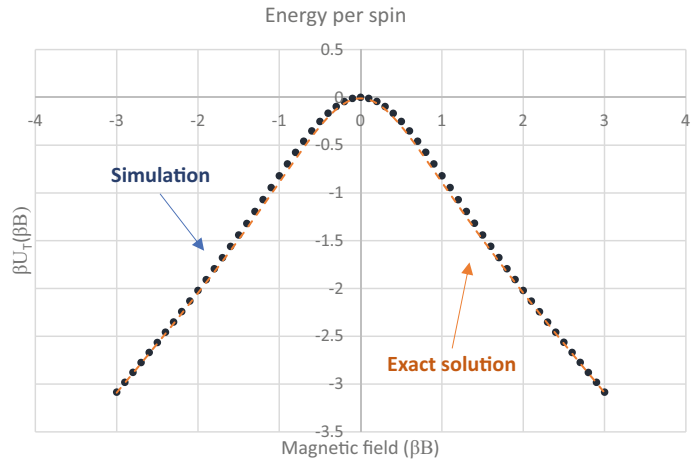


Fig. 5.3 B-field dependence of energy of Ising chain ($N = 3{,}000$) at a fix temperature

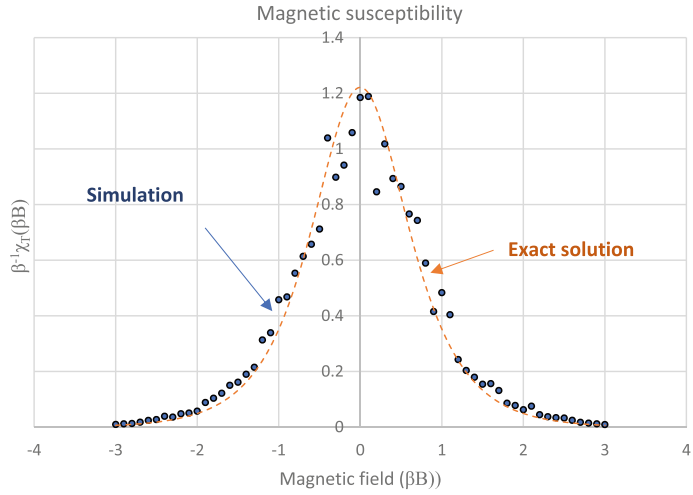


Fig. 5.4 B-field dependence of magnetic susceptibility of Ising chain ($N = 3{,}000$) at a fix temperature

5.4 Two-Dimensional Ising Model

The one-dimensional VBA code can be augmented to the two-dimensional model on a square lattice. We see if the simulation of the two-dimensional Ising model exhibits spontaneous magnetization and the critical phenomena with the zero-field. The critical temperature is known to be $\beta_c J = k_B T_c/J = 2.269$ for the two-dimensional Ising model

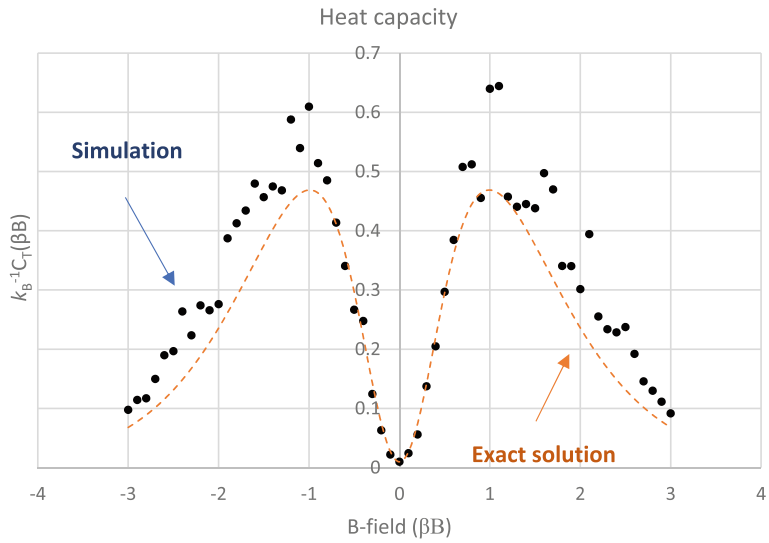


Fig. 5.5 B-field dependence of heat capacity of Ising chain ($N = 3{,}000$) at a fix temperature

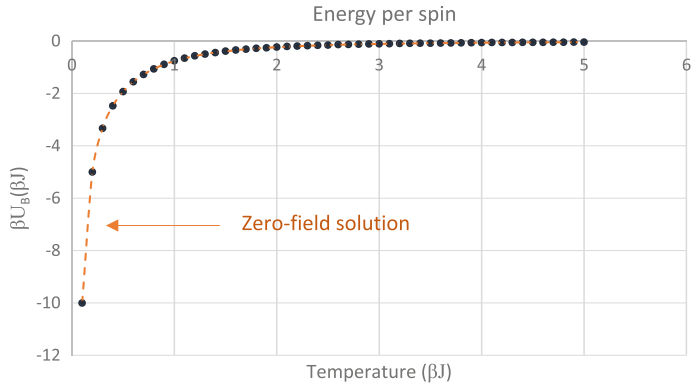


Fig. 5.6 Temperature dependence of energy of Ising chain ($N = 3{,}000$) at near zero-field

on a square lattice. The essential part of the two-dimensional case is the ratio of the probabilities before and after flipping one spin. Similar to the one-dimensional model (Table 5.2),

$$H(\{s_t\}) - H(\{s\}) = -2Js_{i,j}(s_{i,j+1} + s_{i,j-1} + s_{i+1,j} + s_{i-1,j}) - 2Bs_i \tag{5.30}$$

$$p = \frac{w(\{s_t\})}{w(\{s\})} = e^{-H(\{s_t\})+H(\{s\})} = e^{-2s_i(J\cdot f+B)}$$

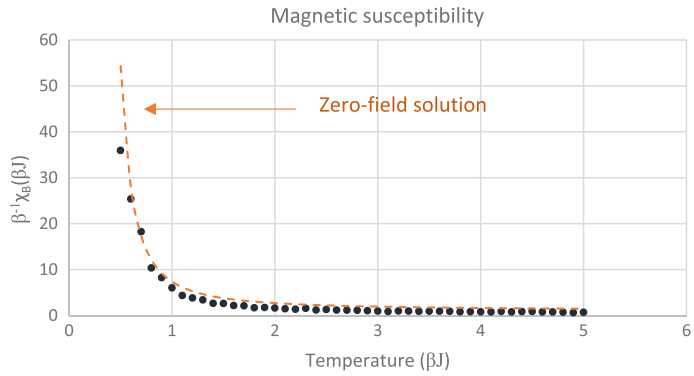


Fig. 5.7 Temperature dependence of susceptibility of Ising chain ($N = 3{,}000$) at near zero-field

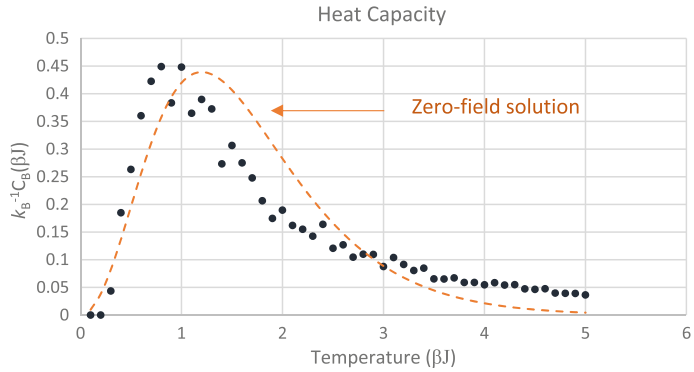


Fig. 5.8 Temperature dependence of heat capacity of Ising chain ($N = 3{,}000$) at near zero-field

$$\text{where } f = s_{i,j+1} + s_{i,j-1} + s_{i+1,j} + s_{i-1,j} \tag{5.31}$$

Figure 5.9 shows the VBA code for the two-dimensional Ising on 96×96 square lattice. The VBA code computes the temperature dependence of thermodynamic functions (M, U, χ_T, and C_T) with $B = 0$ field. The temperature range is $1.0 \leq \beta J \leq 3.0$. Sampling frequency (thermalization cycle) is 20,000 per computational ensemble, and the ensemble size = 200 of spin lattices. The VBA code computes average values of thermodynamic functions after thermalizations of "`SamplingFreq`" times per ensemble, and then computes the ensemble averages. The number of ensembles is given by "`Size`." The code repeats the simulation "`SIM`" times to compute the average thermodynamic functions of `SIM` runs. In this code, we set `SIM = 1`.

In Figs. 5.9, 5.10, 5.11, 5.12, 5.13, the Ising system is a 96×96 square lattice. There are 20,000 thermalizations for each of 300 ensembles of spin lattices with "near zero"

Table 5.2 Possible nearest neighbor spin configurations of Ising square lattice

$s_{i,j-1}$ $s_{i-1,j}$ -$s_{i,j}$-$s_{i+1,j}$ $s_{i,j+1}$ Legend ● = ↑: up spin ○ = ↓: down spin	$f = s_{i,j+1}$ $+s_{i,j-1}$ $+s_{i+1,j}$ $+s_{i-1,j}$	$f = 2I - 6$ $-2\beta s_i(J \cdot f + B)$	$R = e^{-2\beta s_i(J \cdot f + B)}$
If $s_{i,j} = 1$, then $R(\uparrow) = e^{-2[\beta Jf + \beta B]}$			
○ ○ ● ○ ○	-4	$I = 1; f = -4$ $-2[\beta J(-4) + \beta B]$	$e^{-2[\beta J(-4)+\beta B]}$
● ○ ○ ● ○ & ○ ● ○ ○ ●	-2	$I = 2; f = -2$ $-2[\beta J(-2) + \beta B]$	$e^{-2[\beta J(-2)+\beta B]}$
○ ○ ● ● ○ & ○ ● ● ○ ○	-2	$I = 2; f = -2$ $-2[\beta J(-2) + \beta B]$	$e^{-2[\beta J(-2)+\beta B]}$
● ○ ○ ● ○ & ● ● ● ● ○	0	$I = 3; f = 0$ $-2[\beta J(0) + \beta B]$	$e^{-2[\beta J(-2)+\beta B]}$
● ○ ● ● ○ & ● ● ○ ○ ●	0	$I = 3; f = 0$ $-2[\beta J(0) + \beta B]$	$e^{-2[\beta J(0)+\beta B]}$
● ○ ○ ● ● & ○ ● ● ○ ●	0	$I = 3; f = 0$ $-2[\beta J(0) + \beta B]$	$e^{-2[\beta J(0)+\beta B]}$
● ○ ● ● ● & ● ● ● ○ ●	$+2$	$I = 4; f = +2$ $-2[\beta J(2) + \beta B]$	$e^{-2[\beta J(2)+\beta B]}$
● ● ○ ● ● & ● ● ○ ● ●	$+2$	$I = 4; f = +2$ $-2[\beta J(2) + \beta B]$	$e^{-2[\beta J(2)+\beta B]}$
● ● ● ● ●	$+4$	$I = 5; f = +4$ $-2[\beta J(4) + \beta B]$	$e^{-2[\beta J(4)+\beta B]}$
If $s_{i,j} = -1$, then $R(\downarrow) = e^{-2\beta s_i(J \cdot f + B)} = e^{2\beta(J \cdot f + B)} = 1/e^{-2\beta(J \cdot f + B)} = 1/R(\downarrow)$			

field. The sudden changes of the magnetization, the susceptibility and the heat capacity are caused by the phase transition of the Ising model. The onset graphs show detail temperature dependence near the critical temperature $\beta_c J$ with the same simulation condition. Similar to the phase transition of percolation we discussed in Sect. 2.5, the lattice size does not affect the transition temperature but the change in steepness of the translation

```
Sub Isibg2DCRT()
Cells(1, 1) = "Metropolis Simulation of the 2D Ising Model"
'Ising model using the Monte Carlo (Metropolis & Hastings) method.
'S(i,j)is the spin state +/-1 where i = 1, 2, ..., Nx; and j=1, 2, ..., Ny.
'R(3+F/2, (S+3)/2) is flip probab. when the spin is S and the sum of the n.n. spins
is F.
'F=0, +/-2, +/-4.
'The x-direction is vertical and the y-direction is horizontal on the spread sheet.
  Dim S(1000, 1000) As Integer       'Spin lattice S(x,y).
  Dim R(5, 2) 'Flip probab when the spin is S and the sum of neighboring spins is F.
    Nx = 96 : Ny = 96
    Totspins = Nx * Ny
    Cells(2, 1) = "Nx=": Cells(2, 2) = Nx:  '# of spins in the x-direction: cell B2.
    Cells(3, 1) = "Ny=": Cells(3, 2) = Ny:  '# of spins in the y-direction: Cell B3.
      Cells(Nx + 10, 2) = "JJ"
      Cells(Nx + 10, 3) = "kTJ"
      Cells(Nx + 10, 4) = "M"
      Cells(Nx + 10, 5) = "E"
      Cells(Nx + 10, 6) = "Chi"
      Cells(Nx + 10, 7) = "Heat Cap"
   k = 0          'Index to specify the cells to write calculated magnetization etc.
   B = 0          'External B-field.
   SIM = 1        '# of simulation runs. This progun runs the simulation once.
SamplingFreq = 20000 'Sampling freq. of thermal sweeps at each simulation run.
Size = 200         '#s of ensembles & samples, sample size, and current sampling
freq.
   Cells(4, 7) = "SIM=": Cells(4, 8) = SIM
   Cells(3, 10) = "SmplFreq": Cells(3, 11) = SamplingFreq
   Cells(4, 10) = "Ensemble=": Cells(4, 11) = Size
'Label outputs:
      Cells(2, 13) = "TotE="
      Cells(3, 13) = "Mag="
      Cells(2, 17) = "Chi="
      Cells(2, 21) = "Cb="
      Cells(5, 1).Interior.Color = RGB(255, 255, 255) 'White
        'This cell will be displayed "FINAL" once the simulation will be completed.
 For j = 1 To Ny                                  'Vertical loop
         If j > 1 Then
             Jm = j - 1
         Else
             Jm = Ny
         End If
    For i = 1 To Nx                               'Horizontal loop.
         If i > 1 Then
             Im = i - 1
         Else
             Im = Nx
         End If
             If Rnd() < 0.5 Then
                 S(Im, Jm) = 1
             Else
                 S(Im, Jm) = -1
             End If
     Next i
   Next j
For TempJJ = 10 To 30
    KTJ = TempJJ/10                    'Spin coupling constant (KTJ=kT/J)
    JJ = 1 / KTJ                       'JJ=J/kT = 1.0 to 3.0
'Display parameters:
    Cells(2, 4) = "JJ=": Cells(2, 5) = JJ
    Cells(3, 4) = "B=": Cells(3, 5) = B
'Clean up spins on screen:
    Cells(5, 1) = ""
       For j = 1 To Ny
          For i = 1 To Nx
              Cells(i + 6, j).Interior.Color = RGB(255, 255, 255)     'All white.
          Next i
```

Fig. 5.9 VBA code for two-dimensional Ising model on 96×96 square lattice at zero field

```
        Next j
'Calculate the flip probability for given JJ, and B:
         For i = 1 To 5
            R(i, 2) = Exp(-2 * (JJ * (2 * i - 6) + B))      'S=1.
            R(i, 1) = 1 / R(i, 2)                           'S=-1.
         Next i
For Irun = 1 To SIM                                         'Run simulation SIM-times.
'Cells(2, 7) = "# runs": Cells(2, 8) = Irun
 Cells(5, 1).Interior.Color = RGB(255, 0, 0)                'Cell in RED.
        'This cell will be displayed "FINAL" once the simulation will be completed.
'Initialize thermodynamic variables:
     SumM = 0                                               'Magnetization.
     SumM2 = 0                                              'M^2.
     SumE = 0                                               'Energy.
     SumE2 = 0                                              'E^2.
     SumChi = 0                                             'Susceptibility.
     SumCb = 0                                              'Specific heat.
Randomize (1)
'Randomize the spin lattice S(x,y)=S(i,j) and display on screen:
    For j = 1 To Ny                                         'Vertical loop.
         If j > 1 Then
            Jm = j - 1
         Else
            Jm = Ny
         End If
       For i = 1 To Nx                                       'Horizontal loop.
           If i > 1 Then
              Im = i - 1
           Else
              Im = Nx
           End If
               If Rnd() < 0.5 Then
                    S(Im, Jm) = 1
               Else
                    S(Im, Jm) = -1
               End If
         Next i
       Next j
'Initialize M and E before thermalization at each simulation run:
        EnsembleM = 0
        EnsembleM2 = 0
        EnsembleE = 0
        EnsembleE2 = 0
For Iensemble = 1 To Size
   'Calculate E & M for this lattice:
          Mag = 0          'Zero magnetization for this lattice.
          SumSS = 0        'Zero sum of interaction with upper and left neighbors.
  For iSweep2 = 1 To SamplingFreq  'Loop over sweeps in a ensemble.
       GoSub Thermalization2       'Do a sweep of the lattice SamplingFreq-times.
  Next iSweep2
    For j = 1 To Ny                                         'Vertical loop.
         If j > 1 Then
            Jm = j - 1
         Else
            Jm = Ny
         End If
       For i = 1 To Nx                                       'Horizontal loop.
           If i > 1 Then
              Im = i - 1
           Else
              Im = Nx
           End If
             Mag = Mag + S(i, j)                            'Sum of Magnetization.
```

Fig. 5.9 (continued)

```
            SumSS = SumSS + S(i, j) * (S(Im, j) + S(i, Jm)) 'Sum of Interactions.
        Next i
     Next j
          E = -JJ * SumSS - B * Mag                          'Internal energy.
   'Update ensemble sums:
       EnsembleM = EnsembleM + Abs(Mag)
       EnsembleM2 = EnsembleM2 + Mag ^ 2
       EnsembleE = EnsembleE + E
       EnsembleE2 = EnsembleE2 + E ^ 2
Next Iensemble
   'Compute and display ensemble and total E, M, Ch, Cb:
      'Ensemble average
        EnsembleM = EnsembleM / Size
        EnsembleM2 = EnsembleM2 / Size
        EnsembleE = EnsembleE / Size
        EnsembleE2 = EnsembleE2 / Size
        Chi = Abs(EnsembleM2 - EnsembleM ^ 2)
        Cb = Abs(EnsembleE2 - EnsembleE ^ 2)
   'Update total sums:
        SumM = SumM + EnsembleM
        SumE = SumE + EnsembleE
        SumChi = SumChi + Chi
        SumCb = SumCb + Cb
Next Irun
   'Calculate Total Average values:
       M = SumM / SIM
       TotE = SumE / SIM
       Chi = SumChi / SIM
       Cb = SumCb / SIM
   'Display total values per spin:
       Cells(3, 14) = M / Totspins
       Cells(2, 14) = TotE / Totspins
       Cells(2, 18) = Chi / Totspins
       Cells(2, 22) = Cb / Totspins
       Cells(5, 1).Interior.Color = RGB(255, 255, 255)
                                          'White after completing the simulations.
       Cells(5, 1) = "FINAL"              'Simulation completed.
        GoSub Lattice2
    k = k + 1
       Cells(Nx + 10 + k, 2) = JJ
       Cells(Nx + 10 + k, 3) = kTJ
       Cells(Nx + 10 + k, 4) = Cells(3, 14)
       Cells(Nx + 10 + k, 5) = Cells(2, 14)
       Cells(Nx + 10 + k, 6) = Cells(2, 18)
       Cells(Nx + 10 + k, 7) = Cells(2, 22)
  Next TempJJ
Exit Sub
'-------------------------------------------------------------------------
Thermalization2:               'Subroutine to do a Metropolis sweep of the lattice.
   For j = 1 To Ny
      If j < Ny Then
         Jp = j + 1
      Else
         Jp = 1                        'j index of upper neighbor.
      End If
      If j > 1 Then
         Jm = j - 1
      Else
         Jm = Ny                       'j index of lower neighbor.
      End If
    For i = 1 To Nx
       If i < Nx Then
          Ip = i + 1
```

Fig. 5.9 (continued)

```
        Else
           Ip = 1                        'i index of right neighbor.
        End If
        If i > 1 Then
           Im = i - 1
        Else
           Im = Nx                       'i index of left neighbor.
        End If
           SPIN = S(i, j)                                    'Spin at site (i,j).
           F = S(Ip, j) + S(Im, j) + S(i, Jp) + S(i, Jm)  'Sum of 4 neighbor spins.
            If Rnd() > R(3 + F / 2, (3 + SPIN) / 2) Then
               S(i, j) = SPIN                                'No spin flip.
            Else
                S(i, j) = -SPIN                              'Flip the spin.
            End If
Next i
      Next j
Return
'-------------------------------------------------------------------------------
Lattice2:
  For j = 1 To Ny
    For i = 1 To Nx
        If S(i, j) = 1 Then
            Cells(i + 6, j).Interior.Color = RGB(0, 0, 0)        'Black
        Else:
            Cells(i + 6, j).Interior.Color = RGB(255, 255, 255)  'White
        End If
    Next i
  Next j
Return
'-------------------------------------------------------------------------------
End Sub
```

Fig. 5.9 (continued)

in the magnetization and the susceptibility are remarkable. In particular, the susceptibility changes by a factor of 800 to 900 in the 96×96 lattice whereas it does by factor of 200 to 300. The larger size of a spin lattice better but the computation time increases exponentially. It took almost nine days to complete the superposed figures using a notebook PC!

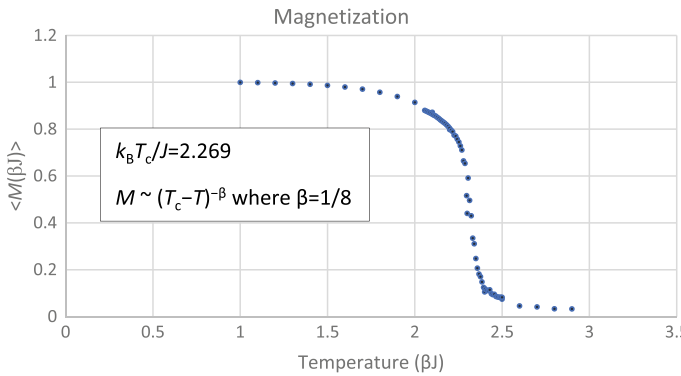


Fig. 5.10 Spontaneous magnetization of Ising square lattice (96×96) near critical temperature

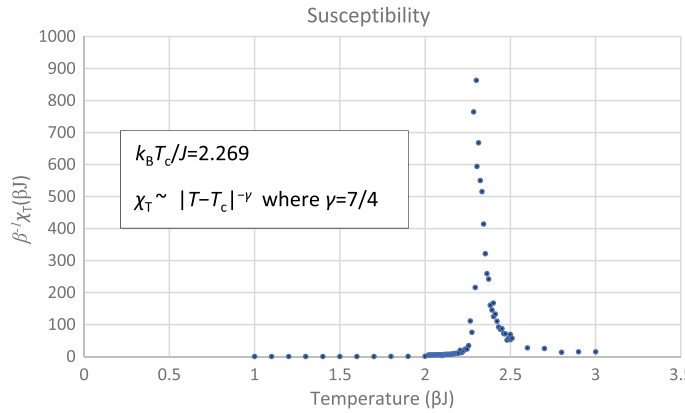


Fig. 5.11 Susceptibility of Ising model on square lattice (96×96) near critical temperature

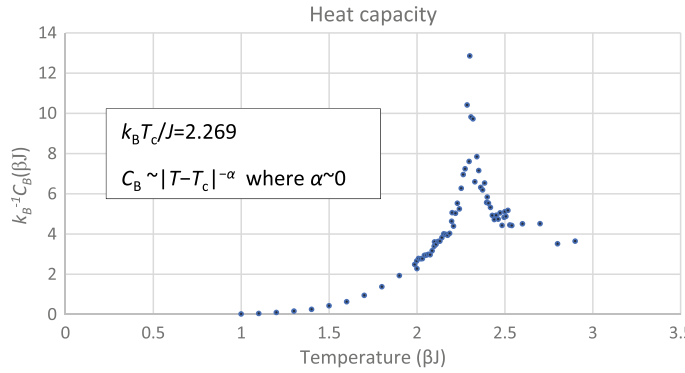


Fig. 5.12 Heat Capacity of Ising model on square lattice (96×96) near critical temperature

The exact solution of the two-dimensional Ising model shows the critical temperature $\beta_c J = 2.269$. The phase transitions of the magnetization (M), the susceptibility (χ_T), and the heat capacity (C_B) near the critical temperature are expressed as the critical exponents, α, β, and γ:

$M \sim (T_c - T)^{-\beta}$ where $\beta = 1/8$.

$\chi_T \sim |T - T_c|^{-\gamma}$ where $\gamma = 7/4$ (divergence).

$C_B \sim |T - T_c|^{-\alpha}$ where $\alpha \sim 0$ (cusp or logarithmic divergence).

Calculation of the critical exponents from the simulation results is beyond our scope. Interested readers should refer to the monumental books [3].

Figure 5.14 shows a pattern of spontaneous magnetization of a 200×200 square lattice at a temperature slightly lower than the critical temperature, $\beta J = k_B T/J = 2.30$. The thermalization cycles are 240,000. In this figure, the black cells indicate spin up states,

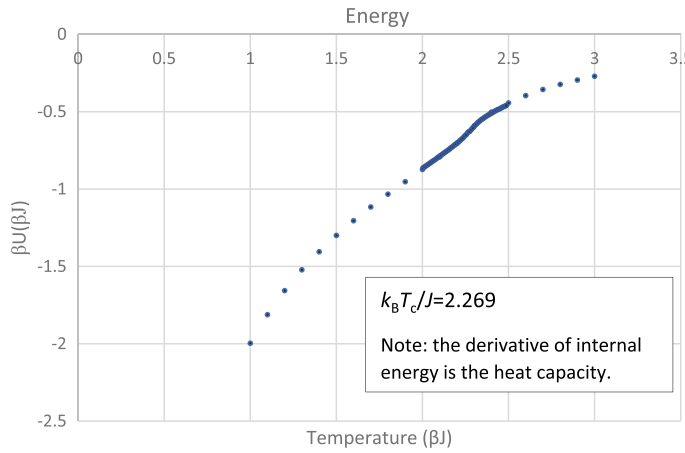


Fig. 5.13 Internal energy of Ising model on square lattice (96×96) near critical temperature

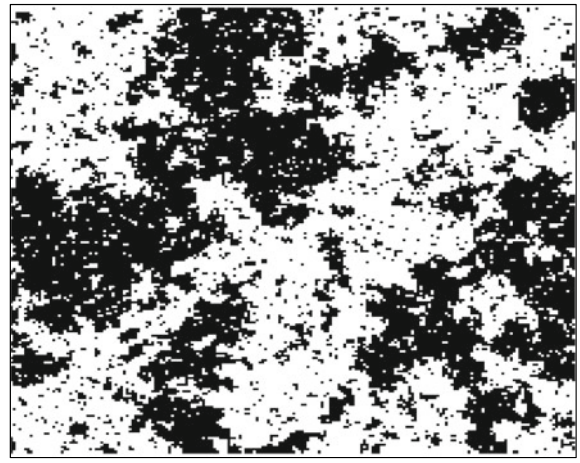


Fig. 5.14 Spin map (200×200) just below the critical point ($\beta J = 2.30$)

with the temperature still slightly below the critical temperature. There are many spin-up clusters of different sizes formed while the net magnetization is still close to zero. At the critical point, these clusters are connected together and several large clusters abruptly appear to establish the spontaneous magnetization as the theory of critical phenomena describes [2]. Recall that, in the percolation geometry phase transition, a similar sudden change in the cluster size occurs to cause the geometrical phase transition.

5.5 Quantum Optimization Using Ising Model

5.5.1 Optimization by Quantum Annealing

The Metropolis algorithm finds the minimum energy state of the Ising spins by flipping a randomly sampled spin, and then compute the system energy if the energy becomes smaller. If it does, keep the spin flipped, and the simulation repeats this flipping step to lower the temperature gradually until the energy becomes the minimum or close to the minimum. This gradual energy-minimizing step is equivalent to the temperature lowering step, and hence, it is called simulated (classical) "annealing." Recently, the Ising model can be applied various optimization models with quantum mechanical approach, which is called "quantum annealing" [9].

Quantum annealing efficiently finds the ground state energy [10]. In the classical nearest neighbor Ising model, the sets of the coupling constant $\{J_{ij}\}$, which may be a constant in some cases, and the external magnetic field is the same for all spins. In the quantum annealing, we minimize the eigenvalue of the Hamiltonian operator where the Ising spins should be expressed as an operator with a set of local interactions $\{J_{ij}\}$ and local external fields $\{h_i\}$:

$$\hat{H}(\{\sigma\}) = -\sum_{i \neq j} J_{ij} \hat{\sigma}_i^z \hat{\sigma}_j^z - \sum_i h_i \hat{\sigma}_i^z \tag{5.32}$$

where $\hat{\sigma}^z$ is a Pauli's spin matrix. The complete set of Pauli's spin matrices are:

$$\hat{\sigma}^x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \ \hat{\sigma}^y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \ \text{and } \hat{\sigma}^z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{5.33}$$

The up-spin and down-spin are respectively expressed as $|\uparrow>$ and $|\downarrow>$, which are eigen vectors the spin matrix $\hat{\sigma}^z$:

$$\hat{\sigma}^z|\uparrow\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |\uparrow\rangle \ \text{ and } \ \hat{\sigma}^z|\downarrow\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} = -|\downarrow\rangle. \tag{5.34}$$

5.5.2 Addition of Horizontal Field

In the classical Monte Carlo method, we prepare a random spin configuration at first. At each site, the spin state takes either up or down. In quantum annealing, we make a superposition of the spin up and down states at each site. The superposed up/down spin

states of a single spin is called a qbit [11]. The quantum annealing creates qbits in the following ways.

Consider the superposed spin states:

$$|+\rangle = \frac{1}{\sqrt{2}}(|\uparrow\rangle + |\downarrow\rangle) \text{ and } |-\rangle = \frac{1}{\sqrt{2}}(|\uparrow\rangle - |\downarrow\rangle). \tag{5.35}$$

They are eigen states of the Pauli's spin matrix $\hat{\sigma}^x$:

$$\hat{\sigma}^x|+\rangle = \frac{1}{\sqrt{2}}|+\rangle \text{ and } \hat{\sigma}^x|-\rangle = -\frac{1}{\sqrt{2}}|-\rangle. \tag{5.36}$$

Therefore, by applying a strong external field in the *horizontal* direction, we can create the superposed state for each spin. In other words, the superposed spin states are eigne states of a Hamiltonian that has the horizontal magnetic field. In order to create superposed spin states at start, the quantum annealing adds a strong horizontal external field term to the original Hamiltonian. The additional field term, called the quantum fluctuation term, can be expressed as

$$\hat{H}_1 = -\Gamma(t)\sum_i \hat{\sigma}_i^x. \tag{5.37}$$

Then, quantum annealing takes the following steps for optimization:

(1) We find the initial eigen values of the quantum fluctuation term, $\hat{H}_1(t=0)$, to create the superposed spin states, $|\Psi(t)>$.
(2) Apply the Schrödinger equation of the Hamiltonian, $\hat{H} = \hat{H}_0 + \hat{H}_1$

$$i\hbar\frac{\partial}{\partial t}|\Psi(t)\rangle = \hat{H}(t)|\Psi(t)\rangle \tag{5.38}$$

(3) Observe the time evolution by slowly decreasing $\Gamma(t)$ to reduce the fluctuation term $\hat{H}_1$:
(4) If the reduction of $\Gamma(t)$ is slow, we expect that the time evolution is also slow and thus adiabatic [12]. Then, the state $|\Psi(t)>$ at each time is very close to the ground state $|\Phi_t>$ of the Hamiltonian at the time.
(5) Continue the step (3) with a gradual reduction of $\Gamma(t)$, and if we find the eigne states and eigen values of $\hat{H}_0$ after vanishing $\hat{H}_1$, the annealing is completed and the eigen state is closed to the ground state of $\hat{H}_0$.

The time varying total Hamilton would be like

$$\hat{H}(t) = \Xi(t)\hat{H}_0 - \Gamma(T)\sum_i \hat{\sigma}_i^x \tag{5.39}$$

where coefficients $\Xi(t)$ and $\Gamma(t)$ determine the time dependence of the Hamiltonian (5.39). At $t = 0$, staring with $\Xi(0) = 0$ and $\Gamma(0) = 1$, $\Xi(t)$ increases to 1 while $\Gamma(t)$ decreases to zero. For example, we could set linear time dependence $\Xi(t) = \zeta t/T$ and $\Gamma(t) = \gamma(1 - t)/T$ where ζ and γ are constants, and T is a predetermined sufficiently long time. At $t = T$, $\Xi(T) = 1$ and $\Gamma(T) = 0$, and

$$\hat{H}(t) = \Xi(t) \sum_i \hat{\sigma}_i^x \text{ at } t = 0 \text{ and } \hat{H}(T) = \hat{H}_0 \text{ at end } (t = T)$$

According to the adiabatic theorem, the eigen state stays in the ground state at each time if the change of the Hamiltonian (5.39) is slow enough. Thus, the final ground state at $t = T$ will be the solution of the optimization problem expressed by the Hamiltonian (5.32).

5.5.3 Traveling Salesman

Once an optimization problem is expressed in terms of an Ising model, quantum annealing can be performed. As an example of how to obtain an Ising- Hamiltonian, a popular problem of quantum annealing, the traveling salesman problem, will be explained [13].

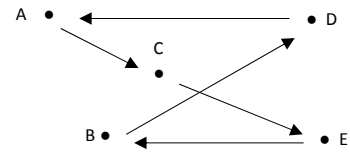
Suppose there are five cities (A, B, C, D, and E) a salesman must visit. The distances between each pair of cities are given. Each city is visited only once, and no multiple cities can be visited at the same time. How do you find the shortest traveling distance, starting from city A? For example, $A \rightarrow C \rightarrow E \rightarrow B \rightarrow D \rightarrow A$ can be charted as shown in Table 5.3.

In this table. Each element can be expressed 0 or 1. Let us denote the element to be $q_{c,i} = 0$ or 1 where the suffix c is the city name and i is the order of visiting cities. In this case, $i = 1, 2, 3, 4$, or 5. Define the distance between two successively visiting cities C and D as δ_{cd} then the total traveling distance is given by

$$L = \sum_{c,d} \sum_{i=1}^{5} \delta_{cd} q_{c,i} q_{d,i+1} \tag{5.40}$$

Table 5.3 An example route of traveling salesman visiting five cities

Visit	A	B	C	D	E
1st	1	0	0	0	0
2nd	0	0	1	0	0
3rd	0	0	0	0	1
4th	0	1	0	0	0
5th	0	0	0	1	0
Back to 1st	1	0	0	0	0



The optimization task is: "Select each $q_{c,i}$ to find minimum L." Recall that (i) each city will be visited just once; and (ii) only one city can be visited at once. These can be formulated as follows:

$$\text{(i)} \quad \sum_i \left(q_{c,i} - 1\right)^2 = 0 \quad \text{at each city c; and} \tag{5.41}$$

$$\text{(ii)} \quad \sum_c \left(q_{c,i} - 1\right)^2 = 0 \quad \text{at each } i\,.$$

Therefore, the optimization can use the method of Lagrange multipliers:

$$H_0 = \sum_{c,d} \sum_{i=1}^{5} \delta_{cd} q_{c,i} q_{d,i+1} + \alpha \sum_c \sum_i \left(q_{c,i} - 1\right)^2 + \beta \sum_i \sum_c \left(q_{c,i} - 1\right)^2 \tag{5.42}$$

where α and β are positive constants. Replacing $\{q;\ q = 0 \text{ or } 1\}$ with the Ising spins $\{s_z;\ s_z = \pm 1\}$ and using the relationship $q = (1 + s_z)/2$, this problem becomes an Ising model with the given Hamiltonian.

Now, in the quantum annealing, we minimize the eigenvalue of the Hamiltonian operator where the Ising spins should be expressed as an operator using the Pauli's spin matrix $\hat{\sigma}^z$. Excluding a constant term, the Hamiltonian is found to be

$$\hat{H}_0 = \sum_{c,d} \sum_{i=1} \delta_{cd} \hat{\sigma}^z_{c,i} \hat{\sigma}^z_{d,i+1} + \alpha \sum_c \sum_i \left(\hat{\sigma}^z_{c,i} - 1\right)^2 + \beta \sum_i \sum_c \left(\hat{\sigma}^z_{c,i} - 1\right)^2. \tag{5.43}$$

This is the Hamiltonian to be optimized for this problem.

Demonstration of actual quantum annealing is beyond the scope of this book. For details of quantum annealing, refer to [14]. The quantum optimization has been implemented by D-Wave's quantum annealing computer [15]. There are also simulation programs of quantum annealing available [16, 17]. Interested readers should visit their websites.

References

1. Ising E (1925) Beitrag zur theorie des ferromagnetismus. Z Phys 31:253–258
2. Taroni A (2015) 90 years of the Ising model. Nat Phys 11:997. https://www.nature.com/articles/nphys3595
3. Stanley HE (1982) Introduction to phase transitions and critical phenomena. Oxford University Press, New York, NY. Also, refer to Stanley H, 1972, Introduction to phase transitions and critical phenomena. Am J Phys 40:927. https://doi.org/10.1119/1.1986710
4. Robert CP, The Metropolis-Hastings algorithm. https://arxiv.org/pdf/1504.01896.pdf
5. There is an interesting animated simulation. Simulation of the Ising model. http://mattbierbaum.github.io/ising.js/

6. Kadowaki T, Nishimori H (1998) Quantum annealing in the transverses Ising model. Phys Rev E58:5255. https://arxiv.org/abs/cond-mat/9804280
7. Gould H, Tobochnik J (2010) Statistical and thermal physics. Princeton University Press, Princeton, NJ
8. Huang K (1991) Statistical mechanics. Wiley, Hoboken, NJ
9. Morita J, Nishimori H (2008) Mathematical foundation of quantum annealing. J Math Phys 49:125210. https://doi.org/10.1063/1.2995837
10. Neven H (2015) When can quantum annealing win? https://ai.googleblog.com/2015/12/when-can-quantum-annealing-win.htm
11. Cho S (2022) Quantum computation and quantum information simulation using Python. A Premier IOP Concise Physics. Morgan & Claypool, San Rafael, CA
12. Childs A (2008) The quantum adiabatic theory. http://www.cs.umd.edu/~amchilds/teaching/w08/l18.pdf
13. Little JDC, Murty KG, Sweeny DW, Krel C (1963) An algorithm for the traveling salesman problem. https://doi.org/10.1287/opre.11.6.972
14. Moriwaki K (2022) Exploration of Quantum computing: solving optimisation problem using quantum annealer. https://towardsdatascience.com/exploration-of-quantum-computing-solving-optimisation-problem-using-quantum-annealer-77c349671969
15. D-Wave (2023) Unblock the power of practical quantum computing today. https://www.dwavesys.com/
16. McCaffrey J (2022) Quantum-inspired annealing using C# or Python. https://visualstudiomagazine.com/articles/2022/01/20/quantum-inspired-annealing.aspx
17. Wildqat tutorial. https://github.com/shinmorino/wildqat

Chaos and Fractal

6

Although chaos and fractals are not stochastic processes, we find many interesting phenomena caused by their distinct behaviors. For example, chaos is relevant to resonance phenomena in a nonlinear oscillation, and fractal closely relates to the scaling theory of phase transitions. This chapter introduces several computer-generated patterns of chaos and fractals to demonstrate their characteristics without using the Monte Carlo method. Besides scientific curiosity, it's fun to create and watch chaos and fractals on screen!

6.1 Chaos

What is chaos anyway? Like its name suggests, chaos is an irregular dynamic behavior of a system that has a relatively small degree of freedom. It is a deterministic process, but a slight change of the initial condition causes dramatic variations in the following dynamics, making our prediction very difficult. Edward Lorenz found that his model of weather patterns, called Lorentz attractor, showed significantly different dynamic patterns when a set of parameters are slightly different [1]. Another example is that in the logistic mapping, which is a single simple parameter in the population variation of a species, also causes a very unpredictable change [2]. Other examples are nonlinear oscillations in mechanics and electronics that often exhibit chaotic behaviors. These almost-unpredictable changes are called the butterfly effect: the sensitive dependence on initial conditions, and a small change in one state of a deterministic nonlinear system can result in large differences in a later state.

S. Cho, *Monte Carlo Simulations Using Microsoft EXCEL®*,
Synthesis Lectures on Mathematics & Statistics,
https://doi.org/10.1007/978-3-031-33886-1_6

6.1.1 Lorentz Attractor

An attractor is a point in the phase space that is used to describe a systems' dynamic tendency to evolve from an arbitrary initial condition. Lorentz attractor forms from a set of relatively simple three nonlinear equations based on fluid dynamics. It exhibits very interesting and complicated dynamic patterns. In the coupled equations shown below, which are cited from Edward Lortentz's study on his weather model, where P is the Prandti number representing the ratio of the fluid viscosity to its thermal conductivity, R is the difference in temperature between the top and the bottom of the system, and B is the ratio of the width to height of the box where the system flows. Lorentz used $P = 10$, $R = 28$, and $B = 8/3$.

$$\begin{cases} \frac{dx}{dt} = P(y - x) \\ \frac{dy}{dt} = Rx - y - xz \\ \frac{dz}{dt} = xy - Bz \end{cases} \tag{6.1}$$

We apply the Runge–Kutter method [3] to compute $\{x(t), y(t), \text{and } z(t)\}$ to produce the trajectories of the attractor. Figure 6.1 lists the VBA code we created.

Figures 6.2, 6.3, 6.4, 6.5, 6.6 and 6.7 show traces of $x(t)$, $y(t)$, $z(t)$, $\{x(t), y(t)\}$, $\{y(t), z(t)\}$, and $\{x(t), z(t)\}$. The dynamics of the system is chaotic; they are somewhat "periodic" but oscillation amplitudes vary irregularly. Traces $x(t)$ and $y(t)$ are similar but they randomly change both periods and amplitudes. Trace $z(t)$ maintains a relatively same period while the amplitude varies periodically to some extent. As the result, they exhibit distinct oscillational behaviors, which are vividly shown in the xy, yz, and xz trajectories. Two points on the attractor that are near each other at one time will be arbitrarily far apart at later times. It is not easy to predict exactly where the system locates on the attractor without knowing the exact initial conditions. This explains why weather can differ drastically with a slight change of initial conditions.

6.1.2 Logistic Function

The logistic function models the population of a species with limited potential of growth. Examples include a bacteria growth in a test-tube, and a nuclear chain reaction in a reactor. They follow the differential equation for yield describes the population change.

$$\frac{dP}{dt} = \gamma P(1 - \beta P). \tag{6.2}$$

The second term of the above equation, $-\gamma\beta P^2$, is added to the exponential growth equation, $dP/dt = \gamma P$, limits the growth. The solution of Eq. (6.2) is given by

```
Sub Attractor()
Cells(1,1)="Lorentz Attractor"
'Compute a set of three coupled differential equations using the Rung-Kutter method.
    'dx/dt=P(y-x), dy/dt=Rx-y-xz, and dz/dt=xy-Bz.
    'Parameters of the equations:
        P = 10: R = 28: B = 8 / 3
        t = 0                                   'Initial value of t.
        x = 1: y = 1: z = 20                    'Initial position (x and y).
        h = 0.0025                              'Time increment.
        n = 10000                               'Iteration number = n.
'Write labels and initial value in cells:
   'Labels:
      Cells(3, 2) = "Initial t"
      Cells(3, 3) = "Initial x"
      Cells(3, 4) = "Initial y"
      Cells(3, 5) = "Initial z"
   'Initial Values and increment read from cells:
      Cells(4, 2) = t
      Cells(4, 3) = x
      Cells(4, 4) = y
      Cells(4, 5) = z
      Cells(4, 9) = h
   'Parameter names:
      Cells(10, 2) = "t"
      Cells(10, 3) = "x"
      Cells(10, 4) = "y"
      Cells(10, 5) = "z"
   'Rung-Kutta parameters:
  For i = 0 To n
      Cells(i + 11, 2) = t
      Cells(i + 11, 3) = x
      Cells(i + 11, 4) = y
      Cells(i + 11, 5) = z

lx1 = gx(P, R, B, t, x, y, z)
ly1 = gy(P, R, B, t, x, y, z)
lz1 = gz(P, R, B, t, x, y, z)

lx2 = gx(P, R, B, t + h / 2, x + h * lx1 / 2, y + h * ly1 / 2, z + h * lz1 / 2)
ly2 = gy(P, R, B, t + h / 2, x + h * lx1 / 2, y + h * ly1 / 2, z + h * lz1 / 2)
lz2 = gz(P, R, B, t + h / 2, x + h * lx1 / 2, y + h * ly1 / 2, z + h * lz1 / 2)

lx3 = gx(P, R, B, t + h / 2, x + h * lx2 / 2, y + h * ly2 / 2, z + h * lz2 / 2)
ly3 = gy(P, R, B, t + h / 2, x + h * lx2 / 2, y + h * ly2 / 2, z + h * lz2 / 2)
lz3 = gz(P, R, B, t + h / 2, x + h * lx2 / 2, y + h * ly2 / 2, z + h * lz2 / 2)

lx4 = gx(P, R, B, t + h, x + h * lx3, y + h * ly3, z + h * lz3)
ly4 = gy(P, R, B, t + h, x + h * lx3, y + h * ly3, z + h * lz3)
lz4 = gz(P, R, B, t + h, x + h * lx3, y + h * ly3, z + h * lz3)

   t = t + h
   x = x + h * (lx1 + 2 * lx2 + 2 * lx3 + lx4) / 6
   y = y + h * (ly1 + 2 * ly2 + 2 * ly3 + ly4) / 6
   z = z + h * (lz1 + 2 * lz2 + 2 * lz3 + lz4) / 6

  Next i
End Sub
'-----------------------------------------------------------------------
Function gx(P, R, B, t, x, y, z)
   'dx/dt=gx
    gx = P * (y - x)
End Function
'-----------------------------------------------------------------------
```

Fig. 6.1 VBA code of strange attractor

```
Function gy(P, R, B, t, x, y, z)
    'dy/dt=gy
     gy = R * x - y - x * z
End Function
'-----------------------------------------------------------------------
Function gz(P, R, B, t, x, y, z)
    'dz/dt=gz
     gz = x * y - B * z
End Function
```

Fig. 6.1 (continued)

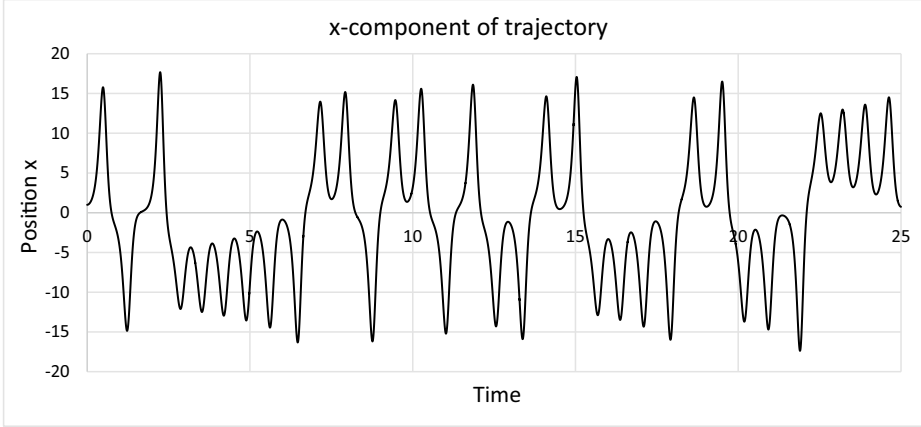


Fig. 6.2 Time dependence of x-coordinates of strange attractor

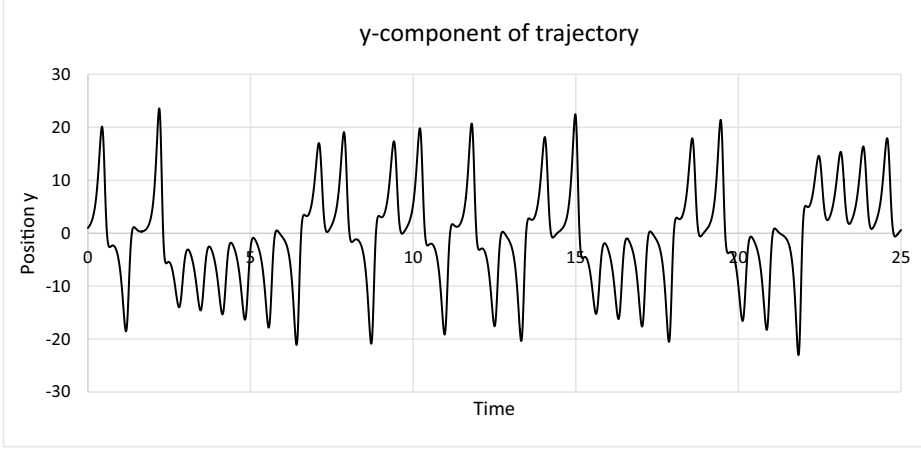


Fig. 6.3 Time dependence of *y*-coordinates of strange attractor

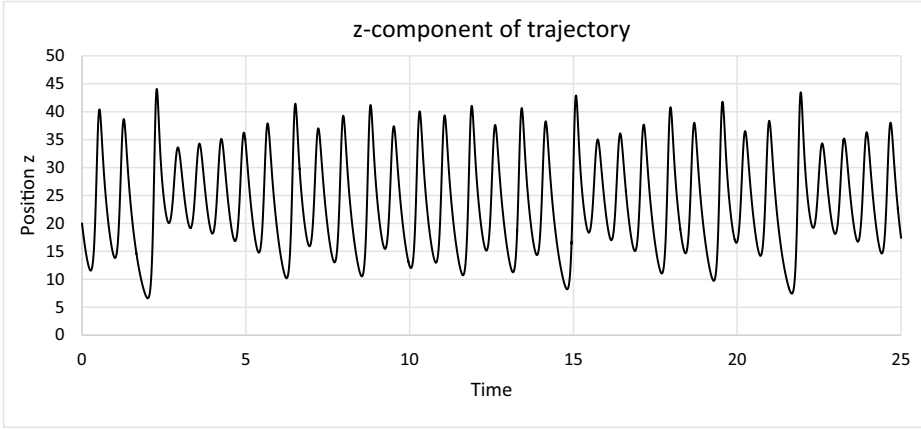


Fig. 6.4 Time dependence of z-coordinates of strange attractor

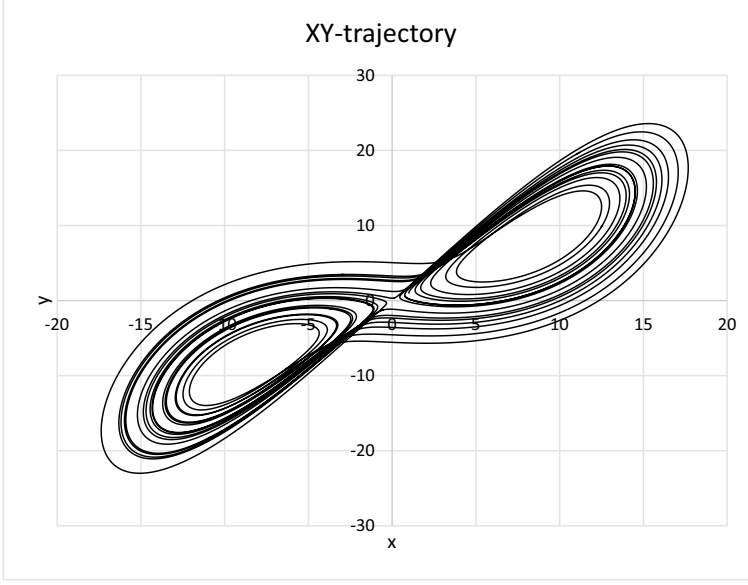


Fig. 6.5 Trajectory projected on xy-plane

$$P(t) = \frac{\varepsilon \gamma e^{\gamma t}}{1 + \gamma \beta \varepsilon e^{\gamma t}} \tag{6.3}$$

where ε is a constant to be determined by the initial condition. Notice that $P(t)$ approaches to its saturated value, $1/\beta$, i.e., $P_{\max} = 1/\beta$.

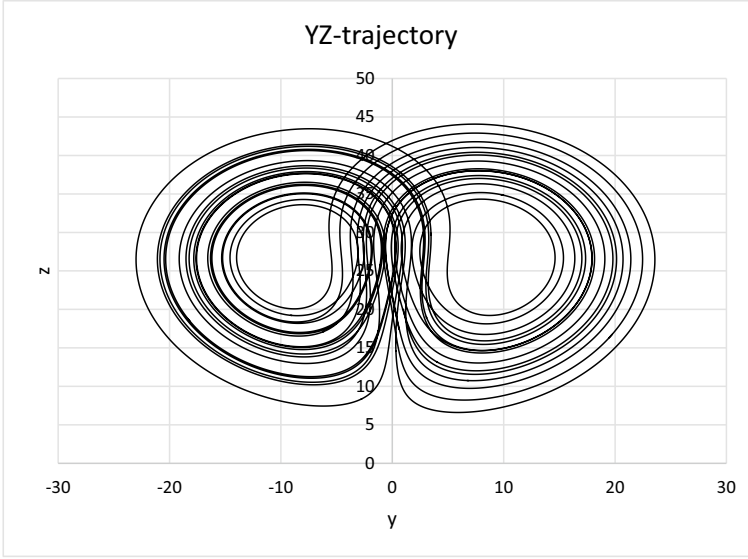


Fig. 6.6 Trajectory projected on *yz*-plane

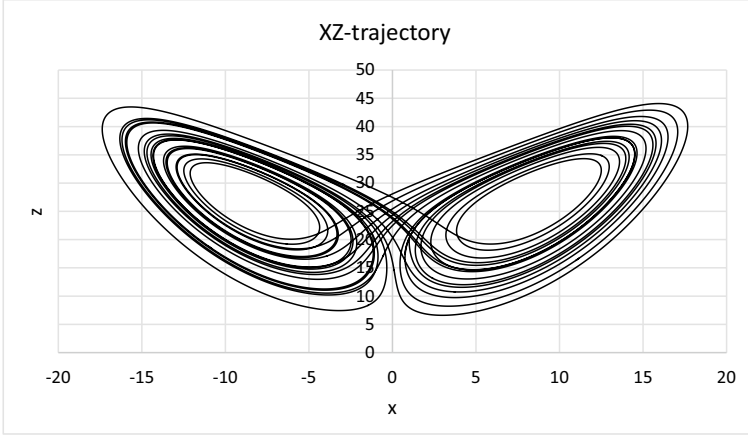


Fig. 6.7 Trajectory projected on *xz*-plane

Differential Eq. (6.2) does not show a chaotic behavior, but the following difference equation derived from Eq. (6.2) exhibits chaos.

$$\frac{\Delta P}{\Delta t} = \frac{P_{n+1} - P_n}{\Delta t} = \gamma P_n (1 - \beta P_n)$$

or

$$P_{n+1} = P_n + \gamma P_n(1 - \beta P_n)\Delta t = (1 + \gamma \Delta t) P_n \left(1 - \frac{\gamma \Delta t}{1 + \gamma \Delta t} \beta P_n\right). \tag{6.4}$$

The logistic equation can be obtained from the above Eq. (6.4). Let $\alpha = 1 + \gamma \Delta t$ and $x_n = (\gamma \Delta t \beta/\alpha) P_n$, and we obtain the logistic equation:

$$\begin{cases} x_{n+1} = \alpha x_n (1 - x_n) \\ t_n = t_0 + n\Delta t \end{cases} \tag{6.5}$$

where $0 < x < 1$ because $P_{\max} = 1/\beta$.

In order to find the dynamic behavior of $x(t)$, we define $y(x) = x$ and $f(x) = \alpha x(1 - x)$. These functions have two fixed points, 0 and x_∞, that satisfy $x_\infty = \alpha\, x_\infty(1 - x_\infty)$, and thus $x_\infty = 1 - 1/\alpha$. We also define $\varepsilon_n = x_n - x_\infty$ or $x_n = \varepsilon_n + x_\infty$ where x_n is the position at the n-steps. If $n > > 1$ and $\varepsilon_n < < 1$, then from Eq. (6.5), we obtain

$$\varepsilon_{n+1} + x_\infty = x_{n+1} = \alpha(\varepsilon_n + x_\infty)(1 - \varepsilon_n - x_\infty), \tag{6.6}$$

and the above equation becomes

$$\varepsilon_{n+1} = \alpha(1 - 2\, x_\infty - \varepsilon_n)\varepsilon_n \text{ because } x_\infty = \alpha\, x_\infty(1 - x_\infty) \text{ from equation (6.5).} \tag{6.7}$$

Since $\varepsilon_n < < 1$, we drop the ${\varepsilon_n}^2$ term from Eq. (6.7) and obtain

$$\varepsilon_{n+1} = \alpha(1 - 2\, x_\infty)\varepsilon_n = (2 - \alpha)\varepsilon_n. \tag{6.8}$$

From Eq. (6.8), if $|2 - \alpha| < 1$, i.e., $1 < \alpha < 3$, the difference between ε_{n+1} and ε_n becomes smaller and smaller, i.e., $\varepsilon_{n+1} \to 0$. Otherwise ε_n is unstable. More detail analysis finds that x will be.

(1) stably monotonic decreases if $0 < \alpha < 1$;
(2) stably monotonic increases if $0 < \alpha \leq 2$;
(3) stably damping if $2 < \alpha \leq 3$;
(4) stably oscillates between two values if $3 < \alpha \leq 1 + \sqrt{6}$; and
(5) periodically or randomly oscillates, exhibiting chaos if $1 + \sqrt{6} < \alpha < 4$.

Figure 6.8 shows the VBA code that generates $x(n = 1000)$-values as a function of the α-value ($\alpha = 0.1$ to 3.95 by step 0.01). It is called a bifurcation diagram. Figure 6.9 plots $x(n = 1000)$ generated by the VBA code. The initial value is $x(0) = 0.1$. For illustrating the α-dependence of the behavior of $x(t)$, on Fig. 6.9, we added typical $x(n)$-traces of the above cases where $0 \leq n \leq 500$ for $\alpha = 0.8$, 1.8, 2.8, 3.2, and 3.6. Those traces are produced using EXCEL's `Autofill` function as shown in Fig. 6.10.

```
Sub LogiMap()
Cells(1,1)="Bifurcation diagram with different initial values"
     Dim x(1001)
     Dim y(1001)
         x(0) = 0.1                        'Initial value
         y(0) = 0.2                        'Initial value
      k = 0                                'Index to list x(1001) and y(1001)
      N = 1000                             '1000 iterations
         For Alpha = 10 To 395             'Alpha-values
             Alpha = AA / 100
                 For i = 0 To N
                     x(i + 1) = Alpha * x(i) * (1 - x(i))
                     y(i + 1) = Alpha * y(i) * (1 - y(i))
                  Next i
             k = k + 1
                Cells(2 + k, 2) = Alpha
                Cells(2 + k, 3) = x(1001)
                Cells(2 + k, 4) = y(1001)
         Next Alpha
End Sub
```

Fig. 6.8 VBA code for generating x-vales at the 100th iteration with different initial values

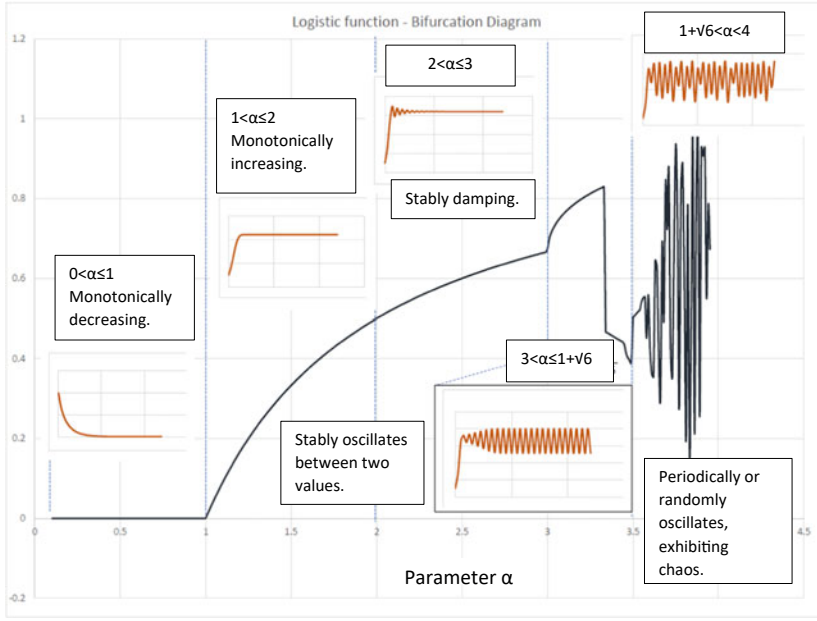


Fig. 6.9 Bifurcation diagram of α-dependence of $x(n = 1000)$

In the VBA code of Fig. 6.8, there are two curves generated with different initial values: $x(0) = 0.1$ and $y(0) = 0.2$. As shown in Fig. 6.11 below, both $x(1000)$ and $y(1000)$ are the same if $\alpha \leq 3$, independent of their initial conditions. On the other hand, they take

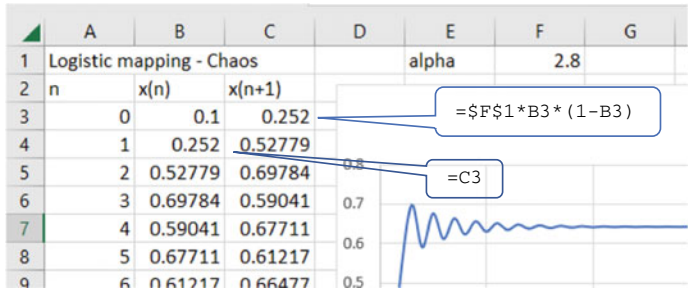


Fig. 6.10 Iterative computation of $x(n)$ using autofill

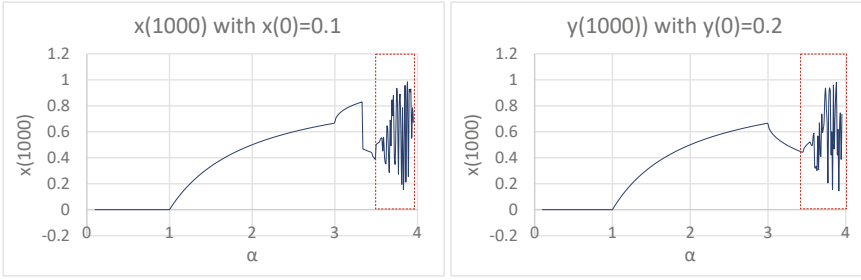


Fig. 6.11 α-value dependence of x-values starting with different initial values

quite different values in the chaotic region of $\alpha > 3$. When $\alpha > 1 + \sqrt{6}$, they oscillate chaotically.

6.1.3 Nonlinear Pendulum

Non-linear oscillations exhibit interesting chaotic oscillatory phenomena in mechanics and electronics. For example, when we add the external driving force and a damping term to a single pendulum, its dynamics become very complicated. Let us investigate the pendulum described by the following equation:

$$\frac{d^2\theta}{dt^2} = -\frac{g}{\ell}\sin\theta - D\frac{d\theta}{dt} + F_D \sin(\Omega_D t) \tag{6.9}$$

where g is the gravitational constant, ℓ is the pendulum length, D is the damping factor, and F_D is the external driving force [4]. We apply the Runge–Kutter method to analyze the dynamics. Figure 6.12 lists the VBA code.

As shown in Figs. 6.13 and 6.14, with different setting of the external force, F_D, the angular frequency, ω, dramatically varies as the angular displacement, θ, varies, and the

```
Sub NonlinearPendulum()
Cells(1,1)="Nonlinear forced oscillation"
'd(2)tehta/dt(2)=-(g/lng)*sin (theta)-D*sin(theta)/dt+F*sin(Omega*t).
   g = 9.8           'Gravitational constant.
   Leng = 9.8        'String length.
   Damping = 0.5     'Damping factor =D.
   Force = 1.2       'External driving force = F.
   Omega = 2 / 3     'Frequency of external driving force.
   pi = 3.141592654  'Period = 2 * pi
'Initial time:
    t = 0
'Initial values of x:
    theta1 = 0.2
    theta2 = theta
'Initial value of angular frequency w=d(theta)dt:
     w = 0
'Time increment:
     h = 0.02
'Iteration n (n*h = range of x; 0 to 5 by step h=0.02):
     n = 5000
'Writing labels and initial value in cells:
'Labels:
     Cells(3, 1) = "Initial t"
     Cells(3, 2) = "Initial angle"
     Cells(3, 3) = "Initial angl.freq"
     Cells(3, 4) = "Delta t"
     Cells(3, 5) = "D"
     Cells(3, 6) = "Ext F"
     Cells(3, 7) = "Ext freq"
'Initial values and increment:
     Cells(4, 1) = t
     Cells(4, 2) = theta1
     Cells(4, 3) = w
     Cells(4, 4) = h
     Cells(4, 5) = Damping
     Cells(4, 6) = Force
     Cells(4, 7) = Omega
     Cells(6, 2) = "t"
     Cells(6, 3) = "theta"
     Cells(6, 4) = "w"           'Angular frequency.
 'Rung-Kutta parameters:
    For i = 0 To n
       Cells(i + 7, 2) = t
       Cells(i + 7, 3) = theta1
       Cells(i + 7, 5) = theta2
       Cells(i + 7, 4) = w
     k1 = d(t, theta, w)
     l1 = f(Damping, Force, Omega, t, theta, w)

     k2 = d(t + h / 2, theta + h * k1 / 2, w + h * l1 / 2)
     l2 = f(Damping, Force, Omega, t + h / 2, theta + h * k1 / 2, w + h * l1 / 2)

     k3 = d(t + h / 2, theta + h * k2 / 2, w + h * l2 / 2)
     l3 = f(Damping, Force, Omega, t + h / 2, theta + h * k2 / 2, w + h * l2 / 2)

     k4 = d(t + h, theta + h * k3, w + h * l3)
     l4 = f(Damping, Force, Omega, t + h, theta + h * k3, w + h * l3)

          t = t + h
             theta1 = theta1 + h * (k1 + 2 * k2 + 2 * k3 + k4) / 6
             w = w + h * (l1 + 2 * l2 + 2 * l3 + l4) / 6

        If theta1 > pi Then theta2 = theta - 2 * pi   'Keep the theta value.
        If theta1 < -pi Then theta2 = theta + 2 * pi  'within the range -pi to +pi.
  Next i
 End Sub
 '-------------------------------------------------------
```

Fig. 6.12 VBA code for nonlinear pendulum

```
Function f(Damping, Force, Omega, t, theta, w)
  'f=dw/dt
   f = -Sin(theta) - Damping * w + Force * Sin(Omega * t)
End Function
'-------------------------------------------------------
Function d(t, theta, w)
  'd=d(theta)/dt
   d = w
End Function
```

Fig. 6.12 (continued)

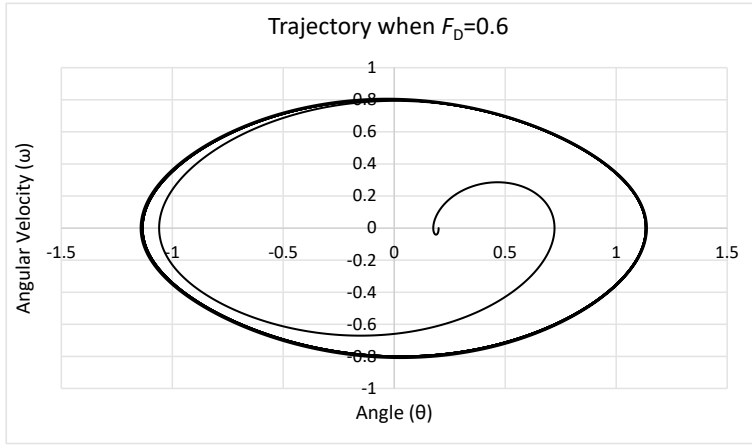


Fig. 6.13 Trajectory on the phase space when $F_D = 0.6$

trajectories in the phase space are considerably different. When $F_D = 0.6$, after the short transient that depends on the initial condition, $\theta(0)$ and $\omega(0)$, the pendulum settles into a stable periodic trajectory in phase space (ω vs θ), and the final trajectory is independent of the initial condition.

When $F_D = 1.2$, the trajectory in the phase space is random but deterministic. It is very much in the region of chaos. The trajectory shown in Fig. 6.14 has multiple orbits that are nearly closed but persists for only a few cycles or so, and then jumps to a different region of the phase space. This chaotic behavior is also seen in Figs. 6.5, 6.6 and 6.7 of the Lorentz attractor. The dynamic range of this chaotic trajectory is much wider than that of the non-chaotic case, indicating that how the dynamic of chaos is wild.

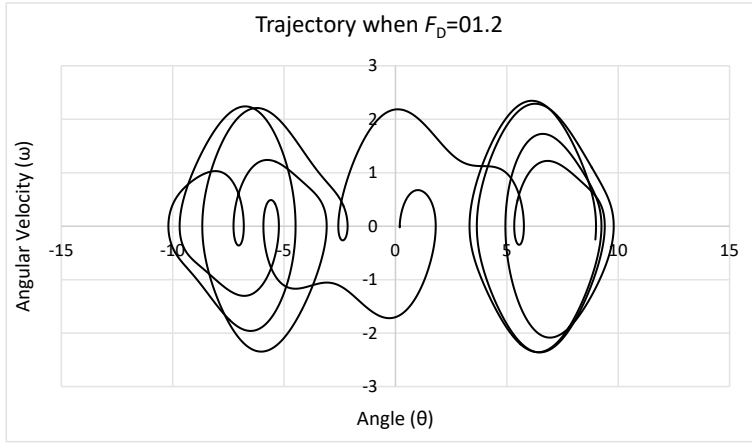


Fig. 6.14 Trajectory on the phase space when $F_D = 01.2$

6.1.4 Nonlinear Double Pendulum

A nonlinear double pendulum reveals much more complexity in its oscillation pattern. Figure 6.15 shows the double pendulum used to demonstrate its dynamics.

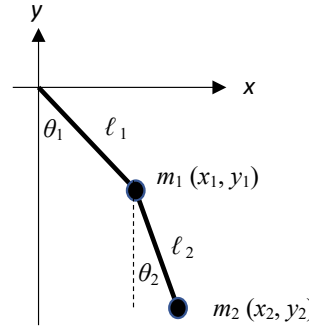
The equations of motion can be obtained from the Lagrangian [5]. Referring to the above figure, the coordinates of mass points m_1 and m_2 are

$$\begin{aligned}(x_1, y_1) &= (\ell_1 \text{sing}\theta_1, \ell_1(1 - \cos\theta_1)), \text{ and} \\ (x_2, y_2) &= (\ell_1 \text{sing}\theta_1 + \ell_2 \sin\theta_2, \ell_1(1 - \cos\theta_1) + \ell_2(1 - \cos\theta_2)).\end{aligned} \tag{6.10}$$

The Lagrangian of the system is given by

$$L = \frac{1}{2}m_1\left(\dot{x}_1^2 + \dot{y}_1^2\right) + \frac{1}{2}m_2\left(\dot{x}_2^2 + \dot{y}_2^2\right) - (m_1 g y_1 + m_2 g y_2)$$

Fig. 6.15 Double pendulum



$$= \frac{1}{2}m_1\ell_1^2\dot{\theta}_1^2 + \frac{1}{2}m_2\{\ell_1^2\dot{\theta}_1^2 + \ell_2^2\dot{\theta}_2^2 + 2\ell_1\ell_2\dot{\theta}_1\dot{\theta}_2\cos(\theta_1 - \theta_2)\}$$
$$- (m_1 + m_2)\ell_1 g(1 - \cos\theta_1) - m_2\ell_2 g(1 - \cos\theta_2). \tag{6.11}$$

From Lagrange equations:

$$\frac{d}{dt}\left[\frac{\partial L}{\partial\dot{\theta}_1}\right] - \frac{\partial L}{\partial\theta_1} = 0 \text{ and } \frac{d}{dt}\left[\frac{\partial L}{\partial\dot{\theta}_2}\right] - \frac{\partial L}{\partial\theta_2} = 0, \tag{6.12}$$

we obtain a set of coupled equations:

$$\begin{cases} \ddot{\theta}_1 + M\ell_{21}\ddot{\theta}_2\cos(\theta_1 - \theta_2) + M\ell_{21}\dot{\theta}_2^2\sin(\theta_1 - \theta_2) + \Omega\sin\theta_1 = 0 \\ \ddot{\theta}_2 + \ell_{21}\ddot{\theta}_1\cos(\theta_1 - \theta_2) - \ell_{21}\dot{\theta}_1^2\sin(\theta_1 - \theta_2) + \frac{\Omega}{\ell_{21}}\sin\theta_2 = 0, \end{cases} \tag{6.13}$$

where $M = \frac{m_2}{m_1+m_2}$, $\ell_{21} = \frac{\ell_2}{\ell_1}$, and $\Omega = \frac{g}{\ell_1}$.

For computational purpose, we express Eqs. (6.13) in simple forms:

$$\begin{cases} \ddot{\theta}_1 = \alpha_1\ddot{\theta}_2 + \beta_1 \\ \ddot{\theta}_2 = \alpha_2\ddot{\theta}_1 + \beta_2 \end{cases} \tag{6.14}$$

where

$$\begin{cases} \alpha_1 = -M\ell_{21}\cos(\theta_1 - \theta_2) \text{ and } \beta_1 = -M\ell_{21}\dot{\theta}_2^2\sin(\theta_1 - \theta_2) - \Omega\sin\theta_1. \\ \alpha_2 = -\ell_{21}\cos(\theta_1 - \theta_2) \text{ and } \beta_2 = \ell_{21}\dot{\theta}_1^2\sin(\theta_1 - \theta_2) - \frac{\Omega}{\ell_{21}}\sin\theta_2. \end{cases} \tag{6.15}$$

Separating $\ddot{\theta}_1$ and $\ddot{\theta}_2$ of Eqs. (6.14), we obtain

$$\begin{cases} \ddot{\theta}_1 = \frac{\alpha_1\beta_2 + \beta_1}{1 - \alpha_1\alpha_2} \\ \ddot{\theta}_2 = \frac{\alpha_2\beta_1 + \beta_2}{1 - \alpha_1\alpha_2} \end{cases}. \tag{6.16}$$

Figure 6.16 lists the VBA code of the dynamics of the double pendulum of Eq. (6.16). The double pendulum exhibits very complicated trajectories with variations of parameters of the pendulum, including lengths, the mass ratio, and the initial conditions.

Figure 6.17 shows two trajectories of the upper and the lower masses with slightly different initial conditions of the lower pendulum. The trajectory in black is for the upper mass and the one in broken orange is for the lower mass. The parameter values are: $\ell_1 = 9.8$, $\ell_2 = 0.98$, $M = 0.5$, $\Omega = 1$, $\theta_1(0) = 0.95\pi$, $\theta_2(0) = 0.60\pi$, and $\omega_1(0) = 0$. The $\omega_2(0)$-values are shown in the Fig. 6.17. Both figures are drawn for the same time interval, and we can see the trajectories with $\omega_2(0) = 0.2$ change very rapidly.

```
Sub DoublePendRK()
Cells(1,1)="Chaos of the Nonlinear double pendulum using the Rung-Kutter method"
  lng1 = 9.8
  lng21 = 0.1
  lng2 = lng1 * lng21
  M = 0.5          'M=m2/(m1+m2)=0.5 when m1=m2.
  omega0 = 1       'omega1=g/lng1.
  h = 0.0066       'For increment in RK-method.
  cn = 1000        'Total # of steps of RK-method.
  n = 1            'Index for outputting all coordinates at every 20-steps.
  pi = 3.141592654
'Initial conditions:
  t = 0
  theta1 = pi * 0.95
  theta2 = pi * 0.7
  omega1 = 0
  omega2 = 0.25
  x1 = lng1 * Sin(theta1)
  y1 = lng1 * Cos(theta1)
  x2 = x1 + lng2 * Sin(theta2)
  y2 = y1 + lng2 * Cos(theta2)
     Cells(9, 1) = "Time": Cells(10, 1) = t
     Cells(9, 2) = "Theta1": Cells(10, 2) = theta1
     Cells(9, 3) = "Theta2": Cells(10, 3) = theta2
     Cells(9, 4) = "x1": Cells(9, 10) = "x1"
     Cells(9, 5) = "y1": Cells(9, 11) = "y1"
     Cells(9, 6) = "x2": Cells(9, 12) = "x2"
     Cells(9, 7) = "y2": Cells(9, 13) = "y2"
     Cells(9, 4) = "x1": Cells(10, 4) = x1: Cells(10, 10) = x1:
     Cells(9, 5) = "y1": Cells(10, 5) = -y1: Cells(10, 11) = -y1:
     Cells(9, 6) = "x2": Cells(10, 6) = x2: Cells(10, 12) = x2:
     Cells(9, 7) = "y2": Cells(10, 7) = -y2: Cells(10, 13) = -y2:
'g1=d(omega1)/dt=(A1B2+B1)/D
'g2=d(omega2)/dt=(A2B1+B2)/D
'D=1-A1A2
'f1=d(theta1)/dt = omega1
'f2=d(theta2)/dt = omega2
For i = 1 To cn
   L11 = g1(t, theta1, theta2, omega1, omega2, M, lng21, omega0)
   K11 = f1(t, theta1, theta2, omega1, omega2, M, lng21, omega0)

L21 = g2(t, theta1, theta2, omega1, omega2, M, lng21, omega0)
K21 = f2(t, theta1, theta2, omega1, omega2, M, lng21, omega0)

L12 = g1(t + h / 2, theta1 + h * L11 / 2, theta2 + h * L21 / 2, omega1 + h * K11 /
2, omega2 + h * K21 / 2, M, lng21, omega0)
K12 = f1(t + h / 2, theta1 + h * L11 / 2, theta2 + h * L21 / 2, omega1 + h * K11 /
2, omega2 + h * K21 / 2, M, lng21, omega0)

L22 = g2(t + h / 2, theta1 + h * L11 / 2, theta2 + h * L21 / 2, omega1 + h * K11 /
2, omega2 + h * K21 / 2, M, lng21, omega0)
K22 = f2(t + h / 2, theta1 + h * L11 / 2, theta2 + h * L21 / 2, omega1 + h * K11 /
2, omega2 + h * K21 / 2, M, lng21, omega0)

L13 = g1(t + h / 2, theta1 + h * L12 / 2, theta2 + h * L22 / 2, omega1 + h * K12 /
2, omega2 + h * K22 / 2, M, lng21, omega0)
K13 = f1(t + h / 2, theta1 + h * L12 / 2, theta2 + h * L22 / 2, omega1 + h * K12 /
2, omega2 + h * K22 / 2, M, lng21, omega0)

L23 = g2(t + h / 2, theta1 + h * L12 / 2, theta2 + h * L22 / 2, omega1 + h * K12 /
2, omega2 + h * K22 / 2, M, lng21, omega0)
K23 = f2(t + h / 2, theta1 + h * L12 / 2, theta2 + h * L22 / 2, omega1 + h * K12 /
2, omega2 + h * K22 / 2, M, lng21, omega0)

L14 = g1(t + h, theta1 + h * L13, theta2 + h * L23, omega1 + h * K13, omega2 + h *
```

Fig. 6.16 VBA code for dynamics of double pendulum

```
K23, M, lng21, omega0)
K14 = f1(t + h, theta1 + h * L13, theta2 + h * L23, omega1 + h * K13, omega2 + h *
K23, M, lng21, omega0)

L24 = g2(t + h, theta1 + h * L13, theta2 + h * L23, omega1 + h * K13, omega2 + h *
K23, M, lng21, omega0)
K24 = f2(t + h, theta1 + h * L13, theta2 + h * L23, omega1 + h * K13, omega2 + h *
K23, M, lng21, omega0)

  t = t + h

  omega1 = omega1 + h * (K11 + 2 * K12 + 2 * K13 + K14) / 6
  omega2 = omega2 + h * (K21 + 2 * K22 + 2 * K23 + K24) / 6
  theta1 = theta1 + h * (L11 + 2 * L12 + 2 * L13 + L14) / 6
  theta2 = theta2 + h * (L21 + 2 * L22 + 2 * L23 + L24) / 6
     x1 = lng1 * Sin(theta1)
     y1 = lng1 * Cos(theta1)
     x2 = x1 + lng2 * Sin(theta2)
     y2 = y1 + lng2 * Cos(theta2)
  Cells(10 + i, 1) = t
  Cells(10 + i, 2) = theta1
  Cells(10 + i, 3) = theta2
  Cells(10 + i, 4) = x1
  Cells(10 + i, 5) = -y1
  Cells(10 + i, 6) = x2
  Cells(10 + i, 7) = -y2
     If i Mod 20 = 0 Then
         Cells(10 + n, 10) = x1
         Cells(10 + n, 11) = -y1
         Cells(10 + n, 12) = x2
         Cells(10 + n, 13) = -y2
     Else
         GoTo Skip
     End If
       n = n + 1
 Skip: Next i
End Sub
'-----------------------------------------------------------------
Function A1(t, theta1, theta2, omega1, omega2, M, lng21, omega0)
  A1 = -M * lng21 * Cos(theta1 - theta2)
End Function
'-----------------------------------------------------------------
Function A2(t, theta1, theta2, omega1, omega2, M, lng21, omega0)
  A2 = -lng21 * Cos(theta1 - theta2)
End Function
'-----------------------------------------------------------------
Function B1(t, theta1, theta2, omega1, omega2, M, lng21, omega0)
  B1 = -M * lng21 * Sin(theta1 - theta2) * (omega2) ^ 2 - omega0 * Sin(theta1)
End Function
'-----------------------------------------------------------------
Function B2(t, theta1, theta2, omega1, omega2, M, lng21, omega0)
  B2 = lng21 * Sin(theta1 - theta2) * (omega1) ^ 2 - omega0 * Sin(theta2) / lng21
End Function
'-----------------------------------------------------------------
Function D(t, theta1, theta2, omega1, omega2, M, lng21, omega0)
  D = 1 - A1(t, theta1, theta2, omega1, omega2, M, lng21, omega0)
End Function
'-----------------------------------------------------------------
Function g1(t, theta1, theta2, omega1, omega2, M, lng21, omega0)
 'g1=d(omega1)/dt the second time derivative of theta1.
  g1 = (A1(t, theta1, theta2, omega1, omega2, M, lng21, omega0) * B2(t, theta1,
theta2, omega1, omega2, M, lng21, omega0) + B1(t, theta1, theta2, omega1, omega2, M,
lng21, omega0)) / D(t, theta1, theta2, omega1, omega2, M, lng21, omega0)
End Function
'-----------------------------------------------------------------
```

Fig. 6.16 (continued)

```
Function g2(t, theta1, theta2, omega1, omega2, M, lng21, omega0)
 'g2=d(omega2)/dt= the second time derivative of theta2.
  g2 = (A2(t, theta1, theta2, omega1, omega2, M, lng21, omega0) * B1(t, theta1,
theta2, omega1, omega2, M, lng21, omega0) + B2(t, theta1, theta2, omega1, omega2, M,
lng21, omega0)) / D(t, theta1, theta2, omega1, omega2, M, lng21, omega0)
End Function
'-----------------------------------------------------------------
Function f1(t, theta1, theta2, omega1, omega2, M, lng21, omega0)
 'f1=d(theta1)/dt
  f1 = omega1
End Function
'-----------------------------------------------------------------
Function f2(t, theta1, theta2, omega1, omega2, M, lng21, omega0)
 'f2=d(theta2)/dt
  f2 = omega2
End Function
```

Fig. 6.16 (continued)

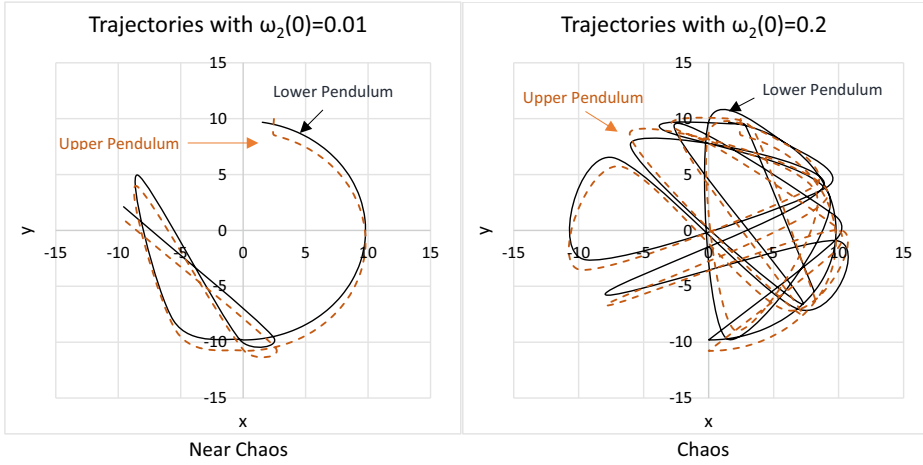


Fig. 6.17 Chaotic behaviors of double pendulum

6.2 Fractal

In nature, we notice many geometrical shapes that form themselves by similar shapes across different scales. For example, tree branches, leaf veins, lighting patterns, ria coastlines have such geometrical shapes. Mandelbrot proposed the concept of fractal to provide a parameter that characterizes the geometrical complexity [6]. Recursion of self-similar patterns is the prime characteristics of fractals. Fractals also bring interesting aspects in physics. Random walks, the percolation problem, and the Ising model we discussed in this book can be identified as fractals because their trajectories and patterns look alike the same across different time scales.

6.2.1 Triadic Koch Curve

Abstract fractals can be generated by a computer computing a simple equation repeatedly. For example, Fig. 6.18 shows a popular self-similar fractal figure called a triadic Koch curve [7]. It can be constructed in the following manner.

(1) The first stage of drawing the Koch curve starts with a straight line of length L as the first-order curve.
(2) The second-order curve is derived from the first-order curve by replacing the straight section with four segments of length $L/3$, oriented with respect to the first-order section.
(3) The third-order curve is drawn from the second-order curve by replacing each of its straight sections by four more segments with length $L/3^2$.
(4) The fourth and higher-order curves are drawn in a similar manner.

In short, at each stage, the displacement of the middle third of each segment is direction that increases the area under the curve.

The Koch curve can be generated with a VBA code listed in Fig. 6.19. The code generates x and y coordinates of the Koch curve up to the order 5. There are 4,097 data points for the order of 5, and 16,385 points for the order of 6! The chart shown in Fig. 6.18 is generated in the [`Scatter with Straight Line`] format.

A fractal is a geometrical shape that displays a property of self-similarity, a geometric shape that can be reduced to smaller parts, with each smaller part being a reduced copy of the whole. This is clearly observed in the Koch Curve. Other than the overall size of

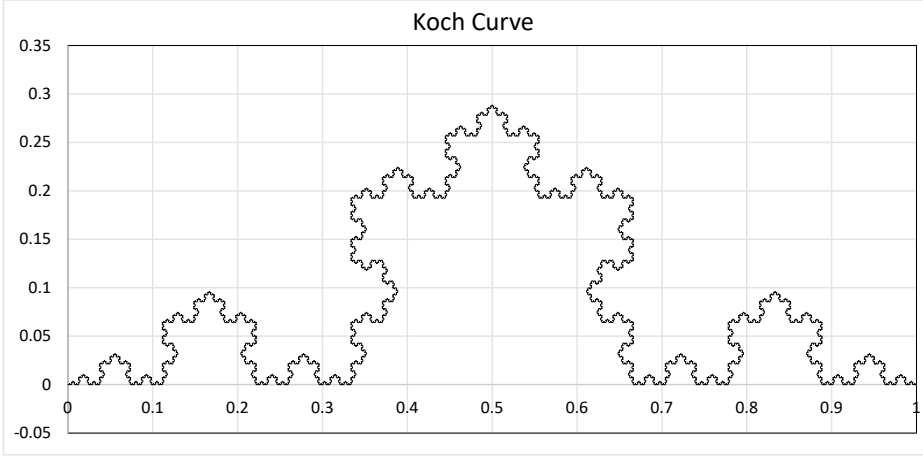


Fig. 6.18 Triadic Koch curve of orders 1 to 5

```
Sub Koch()
Cells(1, 1)="Triadic Koch Curve"
    xi = 0                                      'Stating point (xi, yi).
    yi = 0
    xf = 1                                      'Ending point (xf, yf).
    yf = 0
    Order = 5                                   'Fractal order.
    pi=3.141592654
    Range("A:B").Clear
    Cells(1, 1) = "x"
    Cells(1, 2) = "y"
        Call Draw(xi, yi, xf, yf, Order)        'Call the fractals drawing.
'Acquire the row number of the ending point (xf, yf):
    lngRow = Cells(Rows.Count, 1).End(xlUp).Row + 1
'Writing the ending point (xf, yf):
    Cells(lngRow, 1) = xf
    Cells(lngRow, 2) = yf
End Sub
'-------------------------------------------------------------------
Sub Draw(xi, yi, xf, yf, Order)
'Acquire the row number of the fractal points:
    lngRow = Cells(Rows.Count, 1).End(xlUp).Row + 1
        dx = (xf - xi) / 3
        dy = (yf - yi) / 3
            x1 = xi + dx
            y1 = yi + dy
            x2 = xf - dx
            y2 = yf - dy
' Rotate line segment(dx, dy) by 60 degrees and add to (x1, y1):
    xmid = 0.5 * dx - Sin(pi / 3) * dy + x1
    ymid = Sin(pi / 3) * dx + 0.5 * dy + y1
        If Order > 0 Then                       'Lower the Order by 1.
            Call Draw(xi, yi, x1, y1, Order - 1)
            Call Draw(x1, y1, xmid, ymid, Order - 1)
            Call Draw(xmid, ymid, x2, y2, Order - 1)
            Call Draw(x2, y2, xf, yf, Order - 1)
        Else
            Cells(lngRow, 1) = xi
            Cells(lngRow, 2) = yi
            Cells(lngRow + 1, 1) = x1
            Cells(lngRow + 1, 2) = y1
            Cells(lngRow + 2, 1) = xmid
            Cells(lngRow + 2, 2) = ymid
            Cells(lngRow + 3, 1) = x2
            Cells(lngRow + 3, 2) = y2
        End If
End Sub
```

Fig. 6.19 VBA code of Koch curve generating up to order 5

the figure, fractals have no characteristic lengths unlike a simple geometrical shape such as a triangle, a square, and a cube. How can we sort out this kind of infinitely recurring geometries?

The distinct characterization of fractals is its dimension. There are several specific definitions of fractal dimensions. In this book, we use the fractal dimension proposed by Felix Hausdorff [8]. His fractal dimension is a statistical quantity that characterizes how a fractal appears to fill space. For instance, the Hausdorff dimension of a single point is zero, of a line segment (L) is 1, of a square (A) is 2, and of a cube (V) is 3.

For the Koch curve, its dimension would be between 1 and 2 because the curve is not a simple line in the two-dimensional plane but not entirely occupy on the plane even if

the order of the curve keeps increasing indefinitely. Therefore, the fractal dimension of the Koch curve should be less than 2 but larger than 1. How can we obtain the formula to calculate the fractal dimension? The key is the length scaling.

6.2.2 Sierpinski Triangle

Another example of a fractal is Sierpinski triangle [9]. Figure 6.20 shows a Sierpinski triangle of order 7 where the scale of the spreadsheet display is reduce to 26%. The geometry looks like a two-dimensional mosaic of equilateral triangles, which starts with one large equilateral triangle, subdivided recursively into smaller (self-similar) equilateral triangles. Figure 6.21 is the VBA code to generate the Sierpinski tringle.

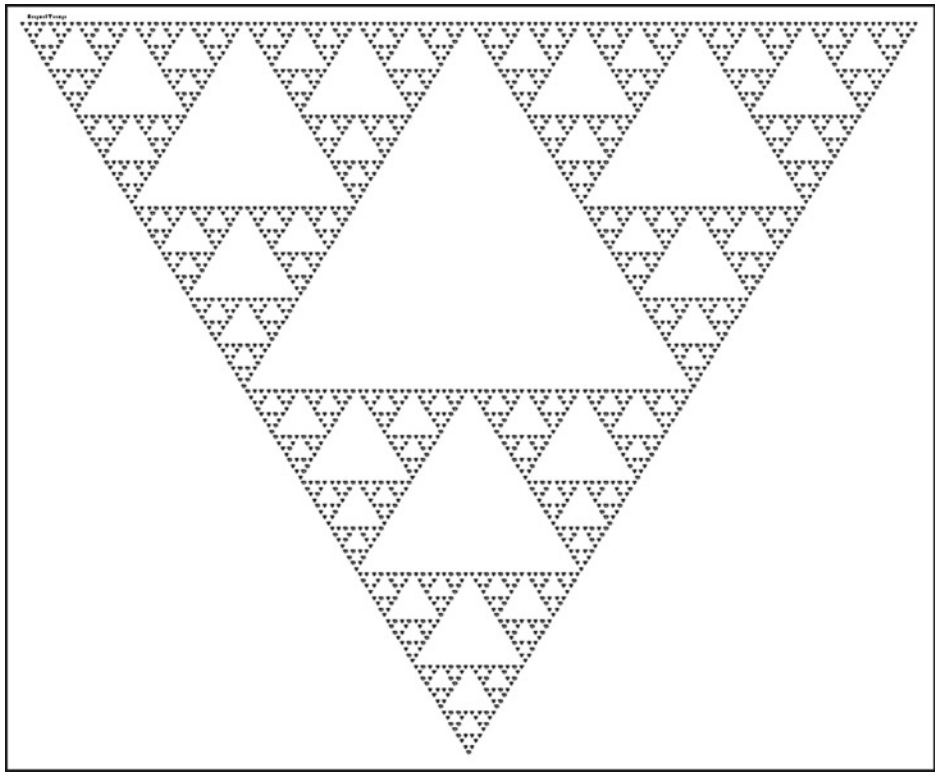


Fig. 6.20 Sierpinski triangle of order 1 to 7

```
Sub S_Triangle
Cells(2, 2) = "Sierpinski Triangle"
    x1 = 40: y1 = 40    'Staring position
    size = 1000         'Size of the largest triangle.
    n = 7               'Order of recursion.
    Call drawSierpinski(x1, y1, size, n)
End Sub
'-------------------------------------------------------------
Sub drawSierpinski(x1, y1, size, n)
    Pi = 3.141592654
    x2 = x1 + size
    x3 = x1 + size / 2
    y2 = y1
    y3 = y1 + size * Sin(Pi / 3)
        If n > 0 Then
            Call drawSierpinski(x1, y1, size / 2, n - 1)
            Call drawSierpinski(x3, y3, size / 2, n - 1)
            Call drawSierpinski(x2, y2, size / 2, n - 1)
        Else
            Call drawTriangle(x1, y1, x2, y2, x3, y3)
        End If
End Sub
'-------------------------------------------------------------
Sub drawTriangle(x1, y1, x2, y2, x3, y3)
'Connect 3 points with lines to form a triangle. Lines are black.
   ActiveSheet.Shapes.AddLine(x1, y1, x2, y2).Line.ForeColor.RGB = RGB(0, 0, 0)
   ActiveSheet.Shapes.AddLine(x2, y2, x3, y3).Line.ForeColor.RGB = RGB(0, 0, 0)
   ActiveSheet.Shapes.AddLine(x3, y3, x1, y1).Line.ForeColor.RGB = RGB(0, 0, 0)
End Sub
```

Fig. 6.21 VBA code of Sierpinski triangle of order 7

6.2.3 Determination of Fractal Dimensions

Suppose we measure the line segment L with a ruler of unit length 1. Next, use another ruler of unit length ε to measure the same line segment. Suppose the length is L' using the ruler of the unit length ε, we have a scale conversion given by $L \to L' = L/\varepsilon$. The scale conversion from 1 to ε have the following relationships for line (1-dimension), area (2-dimension), volume (3-dimension), and general geometry (*d-dimension*) as shown in Table 6.1.

From the d-dimensional geometry, we obtain $\left(\frac{1}{\varepsilon}\right)^d = \frac{V_d'}{V_d}$. Thus, the dimension d can be calculated by

Table 6.1 Scale conversions

Dimension	Scale conversion
1	$L \to \varepsilon^{-1} L = L'$
2	$A \to \varepsilon^{-2} A = A'$
3	$V \to \varepsilon^{-3}\, V = V'$
d	$V_\mathrm{d} \to \varepsilon^{-\mathrm{d}} V_\mathrm{d} = V_\mathrm{d}'$

$$d = \frac{\log\left(V_d'/V_d\right)}{\log(1/\varepsilon)}. \tag{6.17}$$

We call the dimension d of Eq. (6.17) the fractal dimension. The fractal dimension of Koch curve can be calculated in the following way:

(1) Take the line segment of the second-order as the length unit 1;
(2) Then, the total length of the Koch curve of order 2 is $L = 4^2$;
(3) If we scale the length unit to $\varepsilon = 1/3$, it means that we measure the total length with the unit of the line segment of the third order;
(4) If the length of our measuring unit is reduced by a factor of 3, the number of segments is increased by a factor of 4, i.e., $L' = 4^3$; and
(5) $d = \frac{\log(L'/L)}{\log(1/\varepsilon)} = \frac{\log 4}{\log 3} = 1.26\ldots$

As Fig. 6.20 shows, the Sierpinski triangle [9] is closer to two-dimensional geometry. However, the triangle does not fill the two-dimensional space. Therefore, its fractal dimension must be larger than that of 1 and smaller than 2. In addition, its fractal dimension is larger than that of the Koch curve. A recent article observes a fractal dimension in a resistance network which forms a Sierpinski triangle [10]. It is not a computer simulation but determines that the fractal dimension of the Sierpinski triangle is 1.585.

Figure 6.22 shows a pattern of the two-dimensional random walk that leads Eq. (2.8), taking 100,000 step width of ± 1 using the VBA code listed Appendix A2.1. The onset pattern is for the step width of ± 0.01. Observe that they are self-similar, and the patterns will eventually fill the two-dimensional plane. Therefore, the random walk has the fractal dimension 2 according to Hausdorff.

In the patterns of the percolation shown in Fig. 2.21, there are connected regions of various sizes near the critical concentration just below $p_c = 0.5928$. A large-scale computation shows that, at $p_c = 0.5928$, there are large clusters that establish percolation. These clusters are called percolation clusters. The percolation clusters are self-similar, and the fractal dimension is 1.89 for a two-dimensional space [11].

As shown in Fig. 5.13, the spin patterns of the two-dimensional Ising model near the phase transition have many of similar shapes. The two-dimensional Ising model has multiple spin-up regions just before appearing the spontaneous magnetization at the critical temperature. These "islands" are self-similar, and at the critical point, they are connected at once to cause the critical phenomena. The foundation of the scaling theory of the critical phenomena is the self-similarity of fractals. The detailed description of critical phenomena is beyond our scope and the interested readers should refer to reference herein [12, 13].

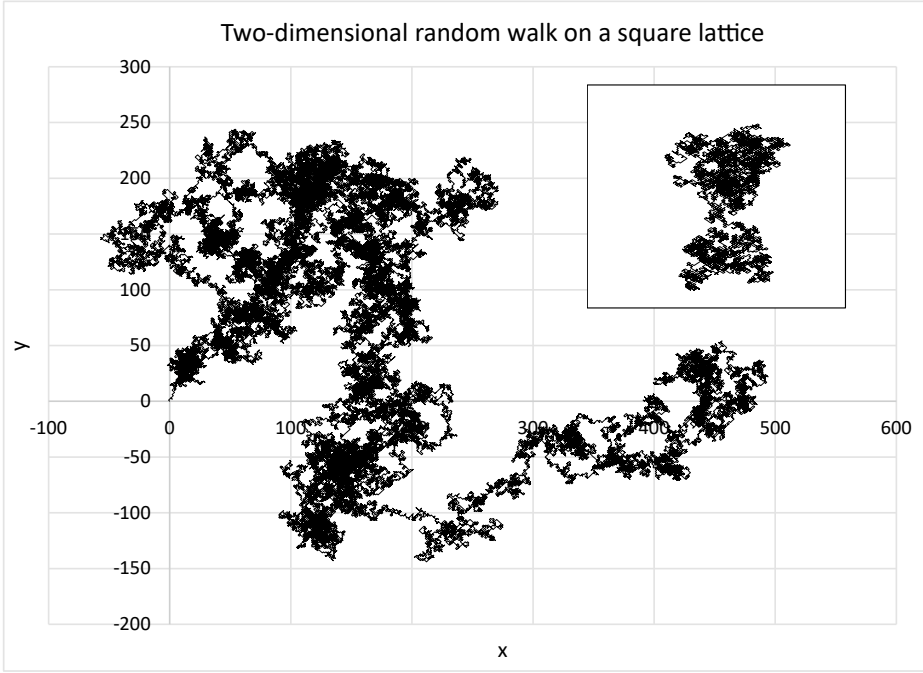


Fig. 6.22 Two-dimensional random walk of 100,000 steps

6.2.4 Note on Chaos and Fractal

Depending on the initial conditions of the recursion procedure, fractal patterns will be significantly changed. In this sense, fractals can be regarded as a chaotic system. Referring Figs. 6.5, 6.6 and 6.7, we observe that trajectories are consistent in similar patters with different sizes. We might point out that the chaos that have self-similar pattern are also fractals, although there are many systems that are chaotic but not fractals. It seems we have not seen strong enough evidence that fractals and chaos are a single entity looking from different sides.

6.2.5 Mandelbrot Figure

Last, but not least, Fig. 6.23 is an example of the Mandelbrot figure. The figure is generated by the VBA code listed in Fig. 6.24. To display the whole figure without changing the cell size in advance, the screen display is reduced to 10%, and the horizontal scale of the image is also readjusted so that each cell becomes a square. It is amazing that the complex geometry of the Mandelbrot can be generated with a small set of recursions.

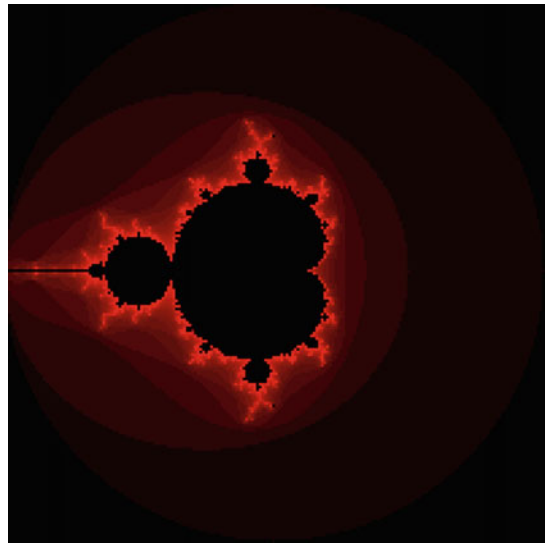


Fig. 6.23 Mandelbrot figure

```
Sub Mandelbrot1()
Cells(1,1)="Mandelbrot figure"
Dim x, y, px, py As Double
Dim BL As Boolean
    For ia = 0 To 200
        For ib = 0 To 200
            x = 0
            y = 0
            BL = True
               For ic = 0 To 200
                   px = x
                   py = y
                   x = px ^ 2 - py ^ 2 + ia / 50 - 2
                   y = 2 * px * py + ib / 50 - 2
             If x ^ 2 + y ^ 2 > 4 Then
               Cells(ib + 1, ia + 1).Interior.Color = (1 - (1 - ic / 100) ^ 9) * 255
               BL = False
               Exit For
             End If
                Next ic
             If BL Then Cells(ib + 1, ia + 1).Interior.Color = 0
           Next ib
        Next ia
End Sub
```

Fig. 6.24 VBA code of generating the Mandelbrot figure

There is a wonderful gallery of Mandelbrot gallery [14]. Create and watch these images. It's fun!

References

1. Sprott JC (2009) Simplifications of the Lorentz Attractor. Nonlinear Dyn, Psychol, Life Sci 13(3):271–278. https://sprott.physics.wisc.edu/pubs/paper327.pdf
2. Logistic Map—College of art and science. Drexel University. http://einstein.drexel.edu/~bob/PHYS750_NLD/ch2.pdf
3. Liengme BV (2016) EXCEL® VBA for physicist. A Primer IOP Concise Physics. Morgan & Claypool, San Rafael, CA
4. Giordano NJ (1977) Computational physics. Prentice Hall, Upper Saddie River, NJ
5. Landau LD, Lifshitz EM (1976) Mechanics. Elsevier, Amsterdam, Netherland
6. Fractal Foundation, What are fractals? https://fractalfoundation.org/resources/what-are-fractals/
7. Riddle LK (2022) Classic iterated function systems. https://larryriddle.agnesscott.org/ifs/ifs.htm
8. Wolfman MathWorld (2023) Hausdorff dimension. https://mathworld.wolfram.com/HausdorffDimension.html
9. The Sierpinski Triangle—Fractals—Mathigon. https://mathigon.org/course/fractals/sierpinski
10. Creffield C (2022) Fractals on a benchtop: observing fractal dimension in a resistor network. Phys Teacher 60
11. Voss RF (1984) The fractal dimension of percolation cluster hulls. J Phys A: Math Gen 17:L373. https://iopscience.iop.org/article/10.1088/0305-4470/17/7/001/pdf
12. Ma S-K (1976) Modern theory of critical phenomena. Taylor & Francis, London, UK
13. Cambier JL, Nauenberg M (1986) Distribution of fractal clusters and scaling in the Ising model. Phys Rev B 34:8071. https://journals.aps.org/prb/abstract/10.1103/PhysRevB.34.8071
14. Mandelbrot Gallery—Pretty math pictures. https://prettymathpics.com/mandelbrot-gallery/

Appendix

A1 EXCEL Options

The following description is for Microsoft® OFFICE 2019 / 365 but should be the same for other (and legacy) Windows-based MS OFFICES.

A1.1 Enabling VBA Macro

EXCEL macro is a Visual Basic (VB) programing environment. Appendix A2 lists additional VBA codes created for this book. Those who are interested in EXCEL macros, refer to IOP Concise Series [1, 2].

Take the following steps to enable EXCEL's macro capability:

(1) Go to [Trust Center].
(2) From [Option], go to [Trust Center], and click on [Trust Center Settings...] of [Excel Options] in [Microsoft Excel Trust Center] (Fig. A.1).
(3) Select [Macro Settings] and check [Enable all macro (not recommended; potentially dangerous code can run)] in order to use this capability. Click [OK] to complete the setting (Fig. A.2).
(4) In the menu, click on [View] and click on [Macros] to select [View Macros] and create a macro program (Fig. A.3).
(5) After entering a macro name, we can create its source code using a built-in editor (Fig. A.4).

S. Cho, *Monte Carlo Simulations Using Microsoft EXCEL®*,
Synthesis Lectures on Mathematics & Statistics,
https://doi.org/10.1007/978-3-031-33886-1

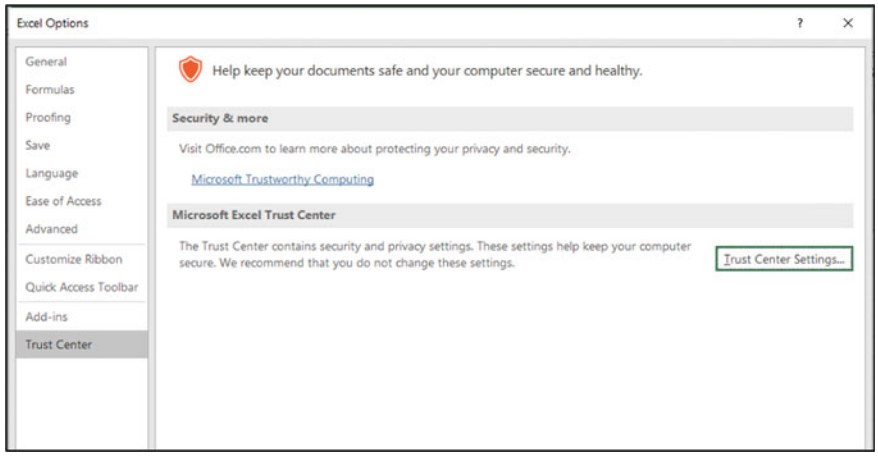


Fig. A.1 EXCEL options for enabling macro

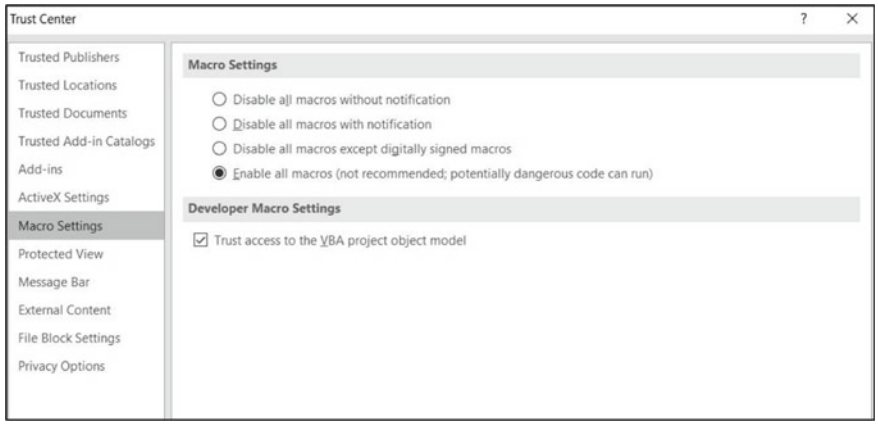


Fig. A.2 Macro settings

NOTE: When you save the EXCEL file with a VBA code, be sure to select "`Excel-Macro Enabled Workbook (*.xlsm)`" from the "`Save as type:`" box below the "`File name:`" box otherwise the VBA code cannot be re-used.

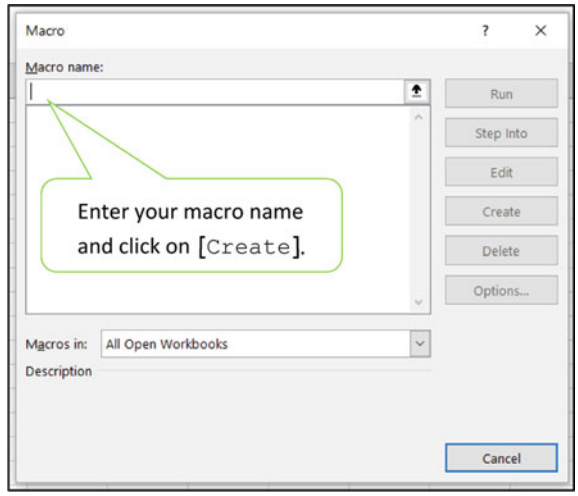


Fig. A.3 Macro entry

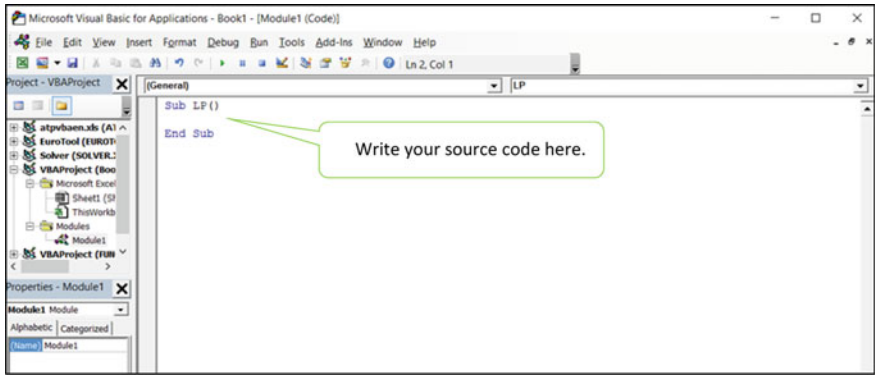


Fig. A.4 Macro editor

A1.2 Adding "Data Analysis"

For analyzing data output from simulation programs, we often need to compute the probability. EXCEL can make a tally easily by using the [Histogram] option in the [`Data Analysis Tools`]. Unfortunately, the option is not in the default setting and must be added. Here is how to do it.

From the [`File`] menu, select [`options`] to display the "`EXCEL Options`" screen. Click on [`Add-ins`] to display the following screen (Fig. A.5).

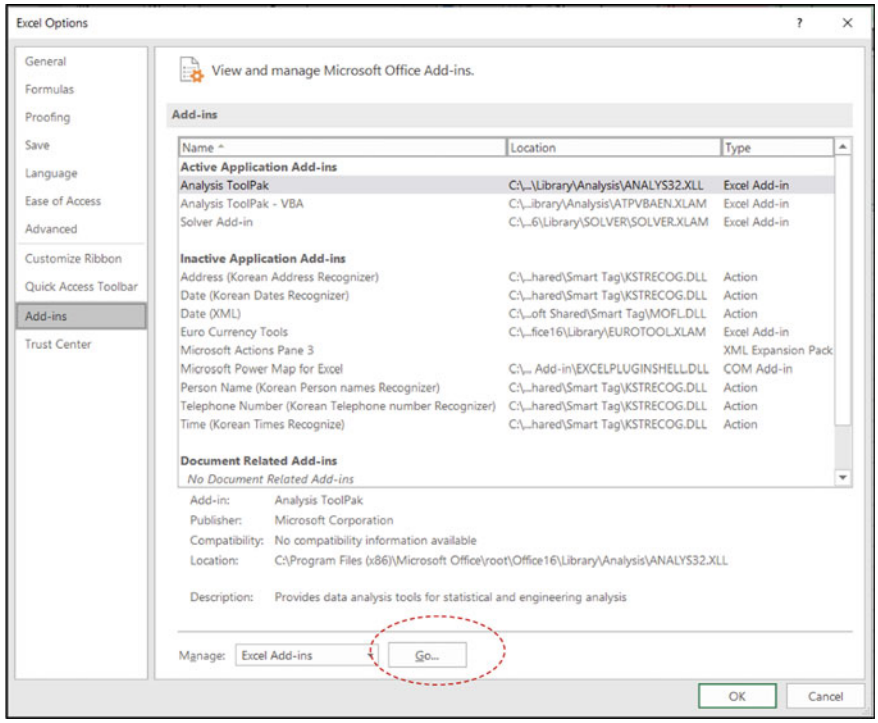


Fig. A.5 EXCEL options screen

Next, click on [Go] to display available add-ins, and then check [Analysis ToolPak], [Analysis ToolPak - VBA], and [Solver Add-in] (Fig. A.6).

A2.3 Autofill

Autofill is a very useful EXCEL tool that is frequently used for scientific calculations, we often use it to add sequential numbers and supplemental calculations. Here is a simple example that enters integer 0, 1, 2, 3, …. in Column A.

(1) Input 0 in Cell A1.
(2) Enter = **A1** + **1** in Cell A2 and press <Enter>. The value of Cell A2 becomes 1.
(3) Place the cursor in Cell A2 and hit <Enter>. The cell is emphasized.

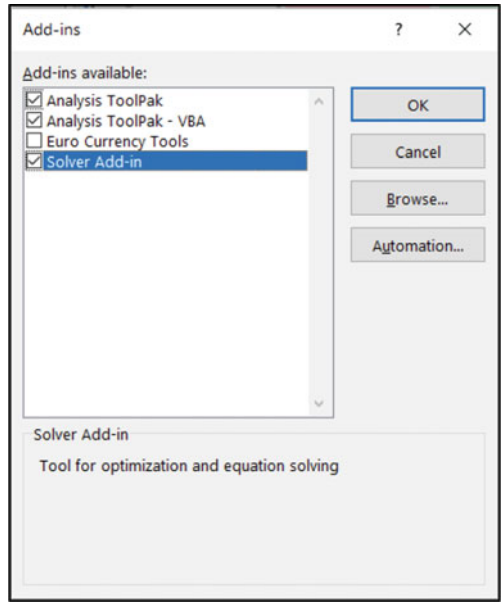


Fig. A.6 Available add-ins

(4) Click on the fill handle (a small solid square at the lower right corner). The cursor becomes a plus sign (+). While pressing the right mouse button, drag the cursor to the cells below in Column A until reaching the desired integer value (Fig. A.7).

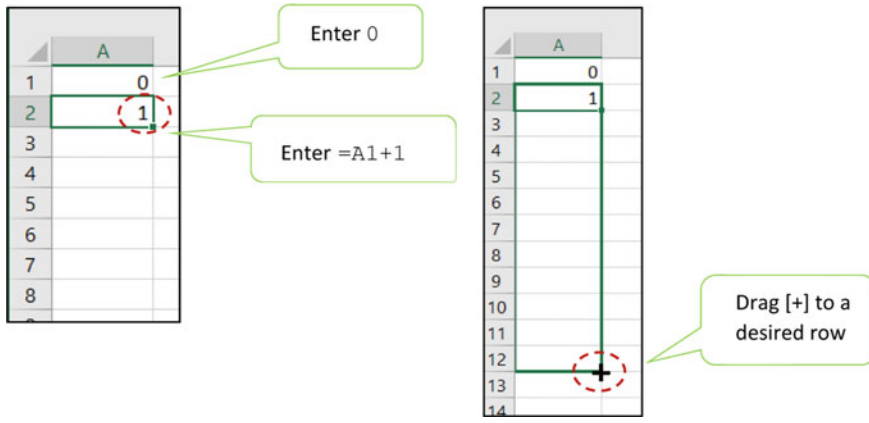


Fig. A.7 Autofill

A2 VBA Codes

This is a collection of VBA codes that not referred to.

A2.1 Two-Dimensional Random Walk on a Square Lattice

A single walker of 100,000 steps. The code outputs the coordinate (x, y) at each step. We used [`Scatter with Straight Line`] to draw the chart shown Fig. 6.20.

```
Sub RandomWalk()
Cells(1, 1) = "Trace of two-dimensional random walk"
   Step = 1: Cells(2, 1) = "Step=": Cells(2, 2) = Step
   Nsteps = 100000: Cells(3, 1) = "Nsteps=": Cells(3, 2) = Nsteps  'Number of total steps.
'Initial position:
    x = 0
    y = 0
'Writing (x,y) of a single walker:
    Cells(1, 6) = "Steps"
    Cells(1, 7) = "x"
    Cells(1, 8) = "y"
For j = 1 To Nsteps
    Cells(1 + j, 6) = j
    Next j
Randomize
'Start at the origin:
    Cells(2, 6) = 0                 'Step number.
    Cells(2, 7) = 0                 'x-coordinate.
    Cells(2, 8) = 0                 'y-coordinate.
    For j = 1 To Nsteps
        Cells(4, 1) = "Current step": Cells(4, 3) = j
           If Rnd() < 0.5 Then
               x = x + 1
           Else
               x = x - 1
           End If
        If Rnd() < 0.5 Then
            y = y + 1
        Else
            y = y - 1
        End If
            Cells(2 + j, 7) = x
            Cells(2 + j, 8) = y
    Next j
End Sub
```

A2.2 Ground State of Three-Dimensional Harmonic Oscillator by QDMC Method

The initial placements of 5,000 walkers are within $-4 \leq x,\ y,\ z \leq\ +\ 4$. Each walker has 100,000 steps. The positions of each walker at 100,000 steps are analyzed using [`Histogram`].

```
Sub Quamtum2()
Cells(1, 1) = "3D Harmonic oscillator by Quantum Diffusion"
'V (i)= 0.5*(x(i)^2 + y(i)^2 + z(i)^2)
Dim x(40000) As Double
Dim y(40000) As Double
Dim z(40000) As Double
    N0 = 5000           'Preset number of walkers.
    ds = 0.1            'Step length.
    dtau = ds * ds      'Imaginary time step.
    mcs = 100000        '# of Monte Carlo steps per walker.
    N = N0              'Initial number of walkers equal to desired number.
    R = 1
    w = 4 * R           'Initial width of region for walkers.
    Vref = 0            'Reference potential.
'Display initial values:
    Cells(1, 2) = "Preset number of walkers"
    Cells(1, 5) = N0
    Cells(2, 2) = "Step length"
    Cells(2, 5) = ds
    Cells(3, 2) = "Number of Monte Carlo steps per walker"
    Cells(3, 5) = mcs
 Randomize
    For i = 1 To N                  'Define initial positions of walkers.
        x(i) = (2 * Rnd() - 1) * w  'Walkers are randomly distributed within a distance w.
        y(i) = (2 * Rnd() - 1) * w  'Walkers are randomly distributed within a distance w.
        z(i) = (2 * Rnd() - 1) * w  'Walkers are randomly distributed within a distance w.
        Vref = Verf + 0.5 * (x(i) ^ 2 + y(i) ^ 2 + z(i) ^ 2)
    Next i
        Vref = Vref / N
        Esum = 0
            Cells(5, 2) = "MC steps"
            Cells(9, 1) = "MC steps"
            Cells(5, 3) = "N"
            Cells(5, 4) = "Energy"
            Cells(9, 2) = "Energy"
            Cells(5, 5) = "Vref"
List = 0                                        'For writing MC steps and energy.
For imcs = 1 To mcs
'Calculate V-Vref and generate random number in the unit interval.
'Add/remove walkers.
'Walk sub procedure:
    GoSub QDiff3:
          Vave = Vsum / N                       'Mean potential.
          Vref = Vave - (N - N0) / (N0 * dtau)  'New reference energy.
'Data accumulation:
    Esum = Esum + Vave
        If (imcs Mod 1000) = 0 Then
            Cells(6, 2) = imcs
            Cells(6, 3) = N
            Energy = Esum / imcs
            Cells(6, 4) = Energy
            Cells(10 + List, 1) = imcs
            Cells(10 + List, 2) = Energy
    List = List + 1
            Cells(6, 5) = Vref + ER
        End If
Next imcs
'Output for histogram:
        Cells(9, 4) = "Bin"
```

A2.3 Ground State of Hydrogen Atom by QDMC Method

The initial placements of 9,000 walkers are within $-6a_0 \leq x,\ y,\ z \leq\ +\ 6a_0$ where a_0 is the Bohr radius. Each walker has 4,000,000 steps. The positions of each walker at 4,000,000 steps are analyzed using [`Histogram`] and [`Scatter`] to draw the normalized distributions of walkers shown in Fig. 4.9.

```
Sub Hatom()
Cells(1, 1) = "Hydrogen atom by Quantum Diffusion"
'V (i)= -1/r
Dim x(10000) As Double
Dim y(10000) As Double
Dim z(10000) As Double
Dim r(10000) As Double
    N0 = 9000               'Desired number of walkers.
    ds = 0.1                'Step length.
    dtau = ds * ds          'Imaginary time step.
    mcs = 4000000           '# of Monte Carlo steps per walker.
    N = N0                  'Initial number of walkers equal to desired number.
    Bohr = 1.398            'Bohr radius.
    w = 6 * Bohr            'Initial width of region for walkers.
    Vref = 0                'Reference potential.
'Display initial values:
    Cells(2, 2) = "Preset number of walkers" : Cells(2, 5) = N0
    Cells(3, 2) = "Step length" : Cells(3, 5) = ds
    Cells(4, 2) = "Number of Monte Carlo steps per walker" : Cells(4, 5) = mcs
Randomize
    For i = 1 To N                  'Define initial positions of walkers.
        x(i) = (2 * Rnd() - 1) * w    'Walkers are randomly distributed within a distance w.
        y(i) = (2 * Rnd() - 1) * w    'Walkers are randomly distributed within a distance w.
        z(i) = (2 * Rnd() - 1) * w    'Walkers are randomly distributed within a distance w.
        Vref = Vref - 1 / Sqr(x(i) ^ 2 + y(i) ^ 2 + z(i) ^ 2)
    Next i
        Vref=Vref/N
    Esum = 0
    Cells(5, 2) = "MC steps"
    Cells(9, 1) = "MC steps"
    Cells(9, 2) = "Energy"
    Cells(5, 3) = "N"
    Cells(5, 4) = "E"
    Cells(5, 5) = "Vref"
List = 1                                    'Index for listing energy at every 1000 steps.
For imcs = 1 To mcs
'Calculate V-Vref and generate random number in the unit interval
'Add/remove walkers
'Walk sub procedure
    GoSub Hatom:
        Vave = Vsum / N                         'Mean potential.
        Vref = Vave - (N - N0) / (N0 * dtau)    'New reference energy.
'Data accumulation
    Esum = Esum + Vave
        For i = 1 To N
            r(i) = Sqr(x(i) ^ 2 + y(i) ^ 2 + z(i) ^ 2)
        Next i
             If (imcs Mod 1000) = 0 Then
                Cells(6, 2) = imcs
                Cells(6, 3) = N
                Energy = Esum / imcs
                Cells(6, 4) = Energy
                Cells(9 + List, 1) = imcs
                Cells(9 + List, 2) = Energy
                Cells(6, 5) = Vref
                List = List + 1
            End If
Next imcs
'Create the radial coordinate
    For ibin = 1 To 500
        Cells(10 + ibin, 4) = ibin / 10
    Next ibin
'Output for histogram
    Cells(9, 4) = "Bin"
    Cells(9, 5) = "Walker #"
    Cells(9, 6) = "r(i)"
        For kk = 1 To N
            Cells(9 + kk, 5) = kk
            Cells(9 + kk, 6) = r(kk)
        Next kk
Exit Sub
'---------------------------------------------------------------
Hatom:
   Ninit = N                                    '# of walkers at beginning of trial.
   Vsum = 0
    For j = Ninit To 1 Step -1
        If Rnd() < 0.5 Then
            x(j) = x(j) + ds
        Else
            x(j) = x(j) - ds
        End If
            If Rnd() < 0.5 Then
                y(j) = y(j) + ds
            Else
                y(j) = y(j) - ds
            End If
                If Rnd() < 0.5 Then
                    z(j) = z(j) + ds
```

```
                Else
                    z(j) = z(j) - ds
                End If
            Potential = -1 / Sqr(x(j) ^ 2 + y(j) ^ 2 + z(j) ^ 2)
            dV = Potential - Vref
                If dV < 0 Then
                    If Rnd() < -dV * dtau Then              'Check to add walker.
                        N = N + 1                           'Add another walker.
                        x(N) = x(j)                         'New walker (x).
                        y(N) = y(j)                         'New walker (y).
                        z(N) = z(j)                         'New walker (z).
                        Vsum = Vsum + 2 * Potential         'Factor of 2 since two walkers at r(i).
                    Else
                        Vsum = Vsum + Potential             'Only do old walker.
                    End If
                Else
                    If Rnd() < dV * dtau Then               'Check to remove walker.
                        x(j) = x(N)
                        y(j) = y(N)
                        z(j) = z(N)
                        N = N - 1                           'Remove walker.
                        Vsum = Vsum - Potential
                    Else                                    'Keep the walker.
                        Vsum = Vsum + Potential
                    End If
                End If
    Next j
Return
'-------------------------------------------------------------
End Sub
```

A2.4 Ground State of Helium Atom by QDMC Method

The initial placements of 8,000 walkers are within $-20a_0 \leq x_{1,2}, y_{1,2}, z_{1,2} \leq +20a_0$ where a_0 is the Bohr radius. Each walker has 1,000,000 steps.

```
Sub Helium()
Cells(1, 1) = "Helium atom by Quantum Diffusion"
'V (i)= -2/r1-2/r2+1/r12
Dim x1(10000) As Double
Dim y1(10000) As Double
Dim z1(10000) As Double
Dim x2(10000) As Double
Dim y2(10000) As Double
Dim z2(10000) As Double
    N0 = 8000               'Preset number of walkers.
    ds = 0.1                'Step length.
    dtau = ds * ds          'Imaginary time step.
    mcs = 1000000           '# Monte Carlo steps per walker.
    N = N0                  'Initial number of walkers equal to desired number.
    R = 1.398               'Bohr radius in a.u.
    w = 20 * R              'Initial width of region for walkers.
    Vref = 0                'Reference potential.
'Display initial values:
    Cells(2, 2) = "Desired number of walkers": Cells(2, 5) = N0
    Cells(3, 2) = "Step width": Cells(3, 5) = ds
    Cells(4, 2) = "Number of Monte Carlo steps per walker": Cells(4, 5) = mcs
Randomize
    For i = 1 To N                      'Define initial positions of walkers.
        x1(i) = (2 * Rnd() - 1) * w     'Walkers are randomly distributed within a distance w.
        y1(i) = (2 * Rnd() - 1) * w     'Walkers are randomly distributed within a distance w.
        z1(i) = (2 * Rnd() - 1) * w     'Walkers are randomly distributed within a distance w.
        x2(i) = (2 * Rnd() - 1) * w     'Walkers are randomly distributed within a distance w.
        y2(i) = (2 * Rnd() - 1) * w     'Walkers are randomly distributed within a distance w.
        z2(i) = (2 * Rnd() - 1) * w     'Walkers are randomly distributed within a distance w.
Vref = Vref - 2 / Sqr(x1(i) ^ 2 + y1(i) ^ 2 + z1(i) ^ 2) - 2 / Sqr(x2(i) ^ 2 + y2(i) ^ 2 +
z2(i) ^ 2) + 1 / Sqr((x2(i) - x1(i)) ^ 2 + (y2(i) - y1(i)) ^ 2 + (z2(i) - z1(i)) ^ 2)
    Next i
        Vref = Vref / N
        Esum = 0
    Cells(5, 2) = "MC steps"
    Cells(9, 1) = "MC steps"
    Cells(9, 2) = "Energy"
    Cells(5, 3) = "N"
    Cells(5, 4) = "E"
    Cells(5, 5) = "Vref"
```

```
List = 1                                    'Index for listing energy at every 1000 steps.
For imcs = 1 To mcs
'Calculate V-Vref and generate random number in the unit interval
'Add/remove walkers
'Walk sub procedure
    GoSub He:
        Vave = Vsum / N                     'Mean potential.
        Vref = Vave - (N - N0) / (N0 * dtau)  'New reference energy.
'Data accumulation:
    Esum = Esum + Vave
    If (imcs Mod 1000) = 0 Then
        Cells(6, 2) = imcs
        Cells(6, 3) = N
        Energy = Esum / imcs
        Cells(6, 4) = Energy
        Cells(6, 5) = Vref
        Cells(9 + List, 1) = imcs
        Cells(9 + List, 2) = Energy
        List = List + 1
    End If
Next imcs
'Output coordinates:
    Cells(9, 5) = "x1"
    Cells(9, 6) = "y1"
    Cells(9, 7) = "z1"
    Cells(9, 9) = "x2"
    Cells(9, 10) = "y2"
    Cells(9, 11) = "z2"
        For kk = 1 To N
            Cells(9 + kk, 5) = x1(kk)
            Cells(9 + kk, 6) = y1(kk)
            Cells(9 + kk, 7) = z1(kk)
            Cells(9 + kk, 9) = x2(kk)
            Cells(9 + kk, 10) = y2(kk)
            Cells(9 + kk, 11) = z2(kk)
        Next kk
Exit Sub
'-----------------------------------------------------------
He:
Ninit = N                                   '# of walkers at beginning of trial.
Vsum = 0
    For j = Ninit To 1 Step -1
        If Rnd() < 0.5 Then
            x1(j) = x1(j) + ds
        Else
            x1(j) = x1(j) - ds
        End If
            If Rnd() < 0.5 Then
                y1(j) = y1(j) + ds
            Else
                y1(j) = y1(j) - ds
            End If
                If Rnd() < 0.5 Then
                    z1(j) = z1(j) + ds
                Else
                    z1(j) = z1(j) - ds
                End If
        If Rnd() < 0.5 Then
            x2(j) = x2(j) + ds
        Else
            x2(j) = x2(j) - ds
        End If
            If Rnd() < 0.5 Then
                y2(j) = y2(j) + ds
            Else
                y2(j) = y2(j) - ds
            End If
                If Rnd() < 0.5 Then
                    z2(j) = z2(j) + ds
                Else
                    z2(j) = z2(j) - ds
                End If
 Potential = -2 / Sqr(x1(j) ^ 2 + y1(j) ^ 2 + z1(j) ^ 2) - 2 / Sqr(x2(j) ^ 2 + y2(j) ^ 2 +
z2(j) ^ 2) + 1 / Sqr((x2(j) - x1(j)) ^ 2 + (y2(j) - y1(j)) ^ 2 + (z2(j) - z1(j)) ^ 2)
 dV = Potential - Vref
        If dV < 0 Then
            If Rnd() < -dV * dtau Then      'Check to add walker.
                N = N + 1                   'Add another walker.
                x1(N) = x1(j)               'New walker (x1).
```

```
                y1(N) = y1(j)                'New walker (y1).
                z1(N) = z1(j)                'New walker (z1).
                x2(N) = x2(j)                'New walker (x2).
                y2(N) = y2(j)                'New walker (y2).
                z2(N) = z2(j)                'New walker (z2).
                Vsum = Vsum + 2 * Potential 'Factor of 2 since two walkers at r(i).
             Else
                Vsum = Vsum + Potential     'Only do old walker.
             End If
        Else
            If Rnd() < dV * dtau Then        'Check to remove walker.
                x1(j) = x1(N)
                y1(j) = y1(N)
                z1(j) = z1(N)
                x2(j) = x2(N)
                y2(j) = y2(N)
                z2(j) = z2(N)
                N = N - 1                     'Remove walker.
                Vsum = Vsum - Potential
            Else
                Vsum = Vsum + Potential
            End If
        End If
    Next j
Return
'--------------------------------------------------------------------------
End Sub
```

A2.5 Ground State of Hydrogen Molecule by QDMC Method

The VBA code is similar to that of He-atom. The initial placements of 8,000 walkers are within $-4R \leq x_{1,2}, y_{1,2}, z_{1,2} \leq +4R$ where R is the average inter-proton distance. Each walker has 1,0000,000 steps. The inter-proton energy, ER, must be added to the potential energy. In this code, `ER = 1/R`.

```
Sub Hmolecule()
Cells(1, 1) = "Hydrogen molecule by Quantum diffusion Monte Carlo method"
'V (i)= -1/SQRT(r1-R/2)-1/SQRT(r1+R/2)-1/SQRT(r2-R/2)-1/SQRT(r2+R/2)+1/SQRT(r12).
'R = dimensionless inter-proton distance (1.398).
Dim x1(10000) As Double
Dim y1(10000) As Double
Dim z1(10000) As Double
Dim x2(10000) As Double
Dim y2(10000) As Double
Dim z2(10000) As Double
    N0 = 8000           'Initial total number of walkers.
    ds = 0.1            'Step width.
    dtau = ds * ds      'Imaginary Time step.
    mcs = 1000000       '# of Monte Carlo steps per walker.
    N = N0              'Initial number of walkers equal to desired number.
    R = 1.9783          'Average distance between two protons in a.u.
    ER = 1 / R          'Proton-proton potential energy.
    w = 4 * R           'Initial width of region for walkers.
    Vref = 0
'Display initial values:
    Cells(2, 2) = "Initial number of walkers" : Cells(2, 5) = N0
    Cells(3, 2) = "Step width" : Cells(3, 5) = ds
    Cells(4, 2) = "Number of Monte Carlo steps per walker" : Cells(4, 5) = mcs

Randomize
    For i = 1 To N                  'Define initial positions of walkers.
      x1(i) = (2 * Rnd() - 1) * w   'Walkers are randomly distributed within a distance w.
      y1(i) = (2 * Rnd() - 1) * w   'Walkers are randomly distributed within a distance w.
      z1(i) = (2 * Rnd() - 1) * w   'Walkers are randomly distributed within a distance w.
      x2(i) = (2 * Rnd() - 1) * w   'Walkers are randomly distributed within a distance w.
      y2(i) = (2 * Rnd() - 1) * w   'Walkers are randomly distributed within a distance w
      z2(i) = (2 * Rnd() - 1) * w   'Walkers are randomly distributed within a distance w.
Vref = Vref - 1 / Sqr((x1(i) - R / 2) ^ 2 + y1(i) ^ 2 + z1(i) ^ 2) - 1 / Sqr((x1(i) + R / 2) ^
2 + y1(i) ^ 2 + z1(i) ^ 2) - 1 / Sqr((x2(i) - R / 2) ^ 2 + y2(i) ^ 2 + z2(i) ^ 2) - 1 /
Sqr((x2(i) + R / 2) ^ 2 + y2(i) ^ 2 + z2(i) ^ 2) + 1 / Sqr((x2(i) - x1(i)) ^ 2 + (y2(i) -
y1(i)) ^ 2 + (z2(i) - z1(i)) ^ 2)
    Next i
    Vref = Vref / N
    Esum = 0
    Cells(5, 2) = "MC steps"
    Cells(9, 1) = "MC steps"
```

```
    Cells(5, 3) = "N"
    Cells(5, 4) = "E"
    Cells(9, 2) = "Energy"
    Cells(5, 5) = "Vref"
List = 1                                        'For listing energy at every 1000 steps.
For imcs = 1 To mcs
'Calculate V-Vref and generate random number in the unit interval.
'Add/remove walkers.
'Walk sub procedure:
    GoSub H2:
        Vave = Vsum / N                         'Mean potential.
        Vref = Vave - (N - N0) / (N0 * dtau)    'New reference energy.
'Data accumulation:
    Esum = Esum + Vave
        If (imcs Mod 1000) = 0 Then
            Cells(6, 2) = imcs
            Cells(6, 3) = N
            Energy = Esum / imcs
            Cells(6, 4) = Energy
            Cells(9 + List, 1) = imcs
            Cells(9 + List, 2) = Energy
            Cells(6, 5) = Vref
            List = List + 1
        End If
Next imcs
'Output coordinates:
    Cells(9, 5) = "x1"
    Cells(9, 6) = "y1"
    Cells(9, 7) = "z1"
    Cells(9, 9) = "x2"
    Cells(9, 10) = "y2"
    Cells(9, 11) = "z2"
        For kk = 1 To N
            Cells(9 + kk, 5) = x1(kk)
            Cells(9 + kk, 6) = y1(kk)
            Cells(9 + kk, 7) = z1(kk)
            Cells(9 + kk, 9) = x2(kk)
            Cells(9 + kk, 10) = y2(kk)
            Cells(9 + kk, 11) = z2(kk)
        Next kk
Exit Sub
'-------------------------------------------------------
H2:
Ninit = N                                       '# of walkers at beginning of trial.
Vsum = 0
    For j = Ninit To 1 Step -1
        If Rnd() < 0.5 Then
            x1(j) = x1(j) + ds
        Else
            x1(j) = x1(j) - ds
        End If
            If Rnd() < 0.5 Then
                y1(j) = y1(j) + ds
            Else
                y1(j) = y1(j) - ds
            End If
                If Rnd() < 0.5 Then
                    z1(j) = z1(j) + ds
                Else
                    z1(j) = z1(j) - ds
                End If
        If Rnd() < 0.5 Then
            x2(j) = x2(j) + ds
        Else
            x2(j) = x2(j) - ds
        End If
            If Rnd() < 0.5 Then
                y2(j) = y2(j) + ds
            Else
                y2(j) = y2(j) - ds
            End If
                If Rnd() < 0.5 Then
                    z2(j) = z2(j) + ds
                Else
                    z2(j) = z2(j) - ds
                End If
Potential = -1 / Sqr((x1(j) - R / 2) ^ 2 + y1(j) ^ 2 + z1(j) ^ 2) - 1 / Sqr((x1(j) + R / 2) ^ 2
+ y1(j) ^ 2 + z1(j) ^ 2) - 1 / Sqr((x2(j) - R / 2) ^ 2 + y2(j) ^ 2 + z2(j) ^ 2) - 1 /
```

```
Sqr((x2(j) + R / 2) ^ 2 + y2(j) ^ 2 + z2(j) ^ 2) + 1 / Sqr((x2(j) - x1(j)) ^ 2 + (y2(j) -
y1(j)) ^ 2 + (z2(j) - z1(j)) ^ 2) + ER
     dV = Potential - Vref
        If dV < 0 Then
            If Rnd() < -dV * dtau Then                'Check to add walker.
                N = N + 1                             'Add another walker.
                x1(N) = x1(j)                         'New walker.
                y1(N) = y1(j)                         'New walker.
                z1(N) = z1(j)                         'New walker.
                x2(N) = x2(j)                         'New walker.
                y2(N) = y2(j)                         'New walker.
                z2(N) = z2(j)                         'New walker.
                    Vsum = Vsum + 2 * Potential       'Factor of 2 since two walkers at r(i).
            Else
                    Vsum = Vsum + Potential           'Only do old walker.
            End If
        Else
            If Rnd() < dV * dtau Then                 'Check to remove walker.
                x1(j) = x1(N)
                y1(j) = y1(N)
                z1(j) = z1(N)
                x2(j) = x2(N)
                y2(j) = y2(N)
                z2(j) = z2(N)
                N = N - 1                              'Remove walker.
                    Vsum = Vsum - Potential
            Else
                Vsum = Vsum + Potential
            End If
        End If
    Next j
Return
'-------------------------------------------------------
End Sub
```

References

1. Liengme BV (2016) EXCEL® VBA for physicist. A Primer IOP Concise Physics Morgan & Claypool, San Rafael, CA
2. Cho S (2021) Numerical calculation for physics laboratory projects using microsoft EXCEL®. A Primer IOP Concise Physics. Morgan & Claypool, San Rafael, CA