Computational Physics: An Introduction to Monte Carlo Simulations of Matrix Field Theory

Badis Ydri

Department of Physics, Faculty of Sciences, BM Annaba University, Annaba, Algeria.

March 16, 2016

Abstract

This book is divided into two parts. In the first part we give an elementary introduction to computational physics consisting of 21 simulations which originated from a formal course of lectures and laboratory simulations delivered since 2010 to physics students at Annaba University. The second part is much more advanced and deals with the problem of how to set up working Monte Carlo simulations of matrix field theories which involve finite dimensional matrix regularizations of noncommutative and fuzzy field theories, fuzzy spaces and matrix geometry. The study of matrix field theory in its own right has also become very important to the proper understanding of all noncommutative, fuzzy and matrix phenomena. The second part, which consists of 9 simulations, was delivered informally to doctoral students who are working on various problems in matrix field theory. Sample codes as well as sample key solutions are also provided for convenience and completness. An appendix containing an executive arabic summary of the first part is added at the end of the book.

Contents

II Monte Carlo Simulations of Matrix Field Theory 115

Introductory Remarks

Introducing Computational Physics

Computational physics is a subfield of computational science and scientific computing in which we combine elements from physics (especially theoretical), elements from mathematics (in particular applied mathematics such as numerical analysis) and elements from computer science (programming) for the purpose of solving a physics problem. In physics there are traditionally two approaches which are followed: 1) The experimental approach and 2) The theoretical approach. Nowadays, we may consider "The computational approach" as a third approach in physics. It can even be argued that the computational approach is independent from the first two approaches and it is not simply a bridge between the two.

The most important use of computers in physics is *simulation*. Simulations are suited for nonlinear problems which can not generally solved by analytical methods. The starting point of a simulation is an idealized model of a physical system of interest. We want to check whether or not the behaviour of this model is consistent with observation. We specify an algorithm for the implementation of the model on a computer. The execution of this implementation is a simulation. Simulations are therefore virtual experiments. The comparison between computer simulations and laboratory experiments goes therefore as follows:

Laboratory experiment	Simulation
sample	model
physical apparatus	computer program (the code)
calibration	testing of code
measurement	computation
data analysis	data analysis

A crucial tool in computational physics is programming languages. In simulations as used by the majority of research physicists codes are written in a high-level compiled language such as Fortran and C/C++. In such simulations we may also use calls to routine libraries such as Lapack. The use of mathematical software packages such as Maple, Mathematica and Matlab is only suited for relatively small calculations. These packages are interpreted languages and thus the code they produce run generally far too slowly compared to compiled languages. In this book we will mainly follow the path of

developping and writing all our codes in a high-level compiled language and not call any libraries. As our programming language we will use Fortran 77 under the Linux operating system. We adopt exclusively the Ubuntu distribution of Linux. We will use the Fortran compilers f77 and gfortran. As an editor we will use mostly Emacs and sometimes Gedit and Nano while for graphics we will use mostly Gnuplot.

References

The main references which we have followed in developing the first part of this book include the following items:

1. N.J.Giordano, H. Nakanishi, Computational Physics (2nd edition), Pearson/Prentice Hall, (2006).

2. H.Gould, J.Tobochnick, W.Christian, An Introduction To Computer Simulation Methods: Applications to Physical Systems (3rd Edition), Addison-Wesley (2006).

3. R.H.Landau, M.J.Paez, C.C. Bordeianu, Computational Physics: Problem Solving with Computers (2nd edition), John Wiley and Sons (2007).

4. R.Fitzpatrick, Introduction to Computational Physics, `http://farside.ph.utexas.edu/teaching/329/329.html`.

5. Konstantinos Anagnostopoulos, Computational Physics: A Practical Introduction to Computational Physics and Scientific Computing, Lulu.com (2014).

6. J. M. Thijssen, Computational Physics, Cambridge University Press (1999).

7. M. Hjorth-Jensen,Computational Physics, CreateSpace Publishing (2015).

8. Paul L.DeVries, A First Course in Computational Physics (2nd edition), Jones and Bartlett Publishers (2010).

Codes and Solutions

The Fortran codes relevant to the problems considered in the first part of the book as well as some key sample solutions can be found at the URL:
`http://homepages.dias.ie/ydri/codes_solutions/`

Matrix Field Theory

The second part of this book, which is effectively the main part, deals with the important problem of how to set up working Monte Carlo simulations of matrix field theories in a, hopefully, pedagogical way. The subject of matrix field theory involves non-perturbative matrix regularizations, or simply matrix representations, of noncommutative field theory and noncommutative geometry, fuzzy physics and fuzzy spaces, fuzzy field theory, matrix geometry and gravity and random matrix theory. The subject of matrix field theory may

even include matrix regularizations of supersymmetry, string theory and M-theory. These matrix regularizations employ necessarily finite dimensional matrix algebras so that the problems are amenable and are accessible to Monte Carlo methods.

The matrix regulator should be contrasted with the, well established, lattice regulator with advantages and disadvantages which are discussed in their places in the literature. However, we note that only 5 simulations among the 7 simulations considered in this part of the book use the matrix regulator whereas the other 2, closely related simulations, use the usual lattice regulator. This part contains also a special chapter on the Remez and conjugate gradient algorithms which are required for the simulation of dynamical fermions. The study of matrix field theory in its own right, and not thought of as regulator, has also become very important to the proper understanding of all noncommutative, fuzzy and matrix phenomena. Naturally, therefore, the mathematical, physical and numerical aspects, required for the proper study of matrix field theory, which are found in this part of the book are quite advanced by comparison with what is found in the first part of the book.

The set of references for each topic consists mainly of research articles and is included at the end of each chapter. Sample numerical calculations are also included as a section or several sections in each chapter. Some of these solutions are quite detailed whereas others are brief. The relevant Fortran codes for this part of the book are collected in the last chapter for convenience and completeness. These codes are, of course, provided as is and no warranty should be assumed.

Appendices

We attach two appendices at the end of this book relevant to the first part of this book. In the first appendix we discuss the floating point representation of numbers, machine precision and roundoff and systematic errors. In the second appendix we give an executive summary of the simulations of part I translated into arabic.

Acknowledgments

Part I

Introduction to Computational Physics

Chapter 1

Euler Algorithm

1.1 Euler Algorithm

It is a well appreciated fact that first order differential equations are commonplace in all branches of physics. They appear virtually everywhere and some of the most fundamental problems of nature obey simple first order differential equations or second order differential equations. It is so often possible to recast second order differential equations as first order differential equations with a doubled number of unknown. From the numerical standpoint the problem of solving first order differential equations is a conceptually simple one as we will now explain.

We consider the general first order ordinary differential equation

$$y^{'} = \frac{dy}{dx} = f(x, y). \tag{1.1}$$

We impose the general initial-value boundary condition is

$$y(x_0) = y_0. \tag{1.2}$$

We solve for the function $y = y(x)$ in the unit $x-$interval starting from x_0. We make the $x-$interval discretization

$$x_n = x_0 + n\Delta x \ , \ n = 0, 1, ... \tag{1.3}$$

The Euler algorithm is one of the oldest known numerical recipe. It consists in replacing the function $y(x)$ in the interval $[x_n, x_{n+1}]$ by the straight line connecting the points (x_n, y_n) and (x_{n+1}, y_{n+1}). This comes from the definition of the derivative at the point $x = x_n$ given by

$$\frac{y_{n+1} - y_n}{x_{n+1} - x_n} = f(x_n, y_n). \tag{1.4}$$

This means that we replace the above first order differential equation by the finite difference equation

$$y_{n+1} \simeq y_n + \Delta x f(x_n, y_n). \tag{1.5}$$

This is only an approximation. The truncation error is given by the next term in the Taylor's expansion of the function $y(x)$ which is given by

$$y_{n+1} \simeq y_n + \Delta x f(x_n, y_n) + \frac{1}{2}\Delta x^2 \frac{df(x,y)}{dx}|_{x=x_n} + ... \tag{1.6}$$

The error then reads

$$\frac{1}{2}(\Delta x)^2 \frac{df(x,y)}{dx}|_{x=x_n}. \tag{1.7}$$

The error per step is therefore proportional to $(\Delta x)^2$. In a unit interval we will perform $N = 1/\Delta x$ steps. The total systematic error is therefore proportional to

$$N(\Delta x)^2 = \frac{1}{N}. \tag{1.8}$$

1.2 First Example and Sample Code

1.2.1 Radioactive Decay

It is an experimental fact that radioactive decay obeys a very simple first order differential equation. In a spontaneous radioactive decay a particle with no external influence will decay into other particles. A typical example is the nuclear isotope uranium 235. The exact moment of decay of any one particle is random. This means that the number $-d\mathcal{N}(t) = \mathcal{N}(t) - \mathcal{N}(t + dt)$ of nuclei which will decay during a time inetrval dt must be proportional to dt and to the number $\mathcal{N}(t)$ of particles present at time t, i.e.

$$- d\mathcal{N}(t) \propto \mathcal{N}(t)dt. \tag{1.9}$$

In other words the probability of decay per unit time given by $(-d\mathcal{N}(t)/\mathcal{N}(t))/dt$ is a constant which we denote $1/\tau$. The minus sign is due to the fact that $d\mathcal{N}(t)$ is negative since the number of particles decreases with time. We write

$$\frac{d\mathcal{N}(t)}{dt} = -\frac{\mathcal{N}(t)}{\tau}. \tag{1.10}$$

The solution of this first order differential equation is given by a simple exponential function, viz

$$\mathcal{N}(t) = \mathcal{N}_0 \exp(-t/\tau). \tag{1.11}$$

The number $\mathcal{N}_0$ is the number of particles at time $t = 0$. The time τ is called the mean lifetime. It is the average time for decay. For the uranium 235 the mean lifetime is around 10^9 years.

The goal now is to obtain an approximate numerical solution to the problem of radioactivity using the Euler algorithm. In this particular case we can compare to an exact solution given by the exponential decay law (1.11). We start evidently from the Taylor's expansion

$$\mathcal{N}(t + \Delta t) = \mathcal{N}(t) + \Delta t \frac{d\mathcal{N}}{dt} + \frac{1}{2}(\Delta t)^2 \frac{d^2\mathcal{N}}{dt^2} + ... \tag{1.12}$$

We get in the limit $\Delta t \longrightarrow 0$

$$\frac{d\mathcal{N}}{dt} = \text{Lim}_{\Delta t \longrightarrow 0} \frac{\mathcal{N}(t + \Delta t) - \mathcal{N}(t)}{\Delta t}. \tag{1.13}$$

We take Δt small but non zero. In this case we obtain the approximation

$$\frac{d\mathcal{N}}{dt} \simeq \frac{\mathcal{N}(t + \Delta t) - \mathcal{N}(t)}{\Delta t}. \tag{1.14}$$

Equivalently

$$\mathcal{N}(t + \Delta t) \simeq \mathcal{N}(t) + \Delta t \frac{d\mathcal{N}}{dt}. \tag{1.15}$$

By using (1.10) we get

$$\mathcal{N}(t + \Delta t) \simeq \mathcal{N}(t) - \Delta t \frac{\mathcal{N}(t)}{\tau}. \tag{1.16}$$

We will start from the number of particles at time $t = 0$ given by $\mathcal{N}(0) = \mathcal{N}_0$ which is known. We substitute $t = 0$ in (1.16) to obtain $\mathcal{N}(\Delta t) = \mathcal{N}(1)$ as a function of $\mathcal{N}(0)$. Next the value $\mathcal{N}(1)$ can be used in equation (1.16) to get $\mathcal{N}(2\Delta t) = \mathcal{N}(2)$, etc. We are thus led to the time discretization

$$t \equiv t(i) = i\Delta t \ , \ \ i = 0, ..., N. \tag{1.17}$$

In other words

$$\mathcal{N}(t) = \mathcal{N}(i). \tag{1.18}$$

The integer N determine the total time interval $T = N\Delta t$. The numerical solution (1.16) can be rewritten as

$$\mathcal{N}(i + 1) = \mathcal{N}(i) - \Delta t \frac{\mathcal{N}(i)}{\tau} \ , \ \ i = 0, ..., N. \tag{1.19}$$

This is Euler algorithm for radioactive decay. For convenience we shift the integer i so that the above equation takes the form

$$\mathcal{N}(i) = \mathcal{N}(i - 1) - \Delta t \frac{\mathcal{N}(i - 1)}{\tau} \ , \ \ i = 1, ..., N + 1. \tag{1.20}$$

We introduce $\hat{\mathcal{N}}(i) = \mathcal{N}(i - 1)$, i.e $\hat{\mathcal{N}}(1) = \mathcal{N}(0) = \mathcal{N}_0$. We get

$$\hat{\mathcal{N}}(i + 1) = \hat{\mathcal{N}}(i) - \Delta t \frac{\hat{\mathcal{N}}(i)}{\tau} \ , \ \ i = 1, ..., N + 1. \tag{1.21}$$

The corresponding times are

$$\hat{t}(i + 1) = i\Delta t \ , \ \ i = 1, ..., N + 1. \tag{1.22}$$

The initial number of particles at time $\hat{t}(1) = 0$ is $\hat{\mathcal{N}}(1) = \mathcal{N}_0$. This approximate solution should be compared with the exact solution (1.11).

1.2.2   A Sample Fortran Code

The goal in this section is to provide a sample Fortran code which implements the above algorithm (1.21). The reasons behind choosing Fortran were explained in the introduction. Any Fortran program, like any other programing language, must start with some *program statement* and conclude with an *end statement*. The program statement allows us to give a name to the program. The end statement may be preceded by a *return statement*. This looks like

```
program radioactivity


c Here is the code


return
end
```

We have chosen the name "radioactivity" for our program. The "c" in the second line indicates that the sentence "here is the code" is only a comment and not a part of the code.

After the program statement come the *declaration statements*. We state the *variables* and their *types* which are used in the program. In Fortran we have the *integer type* for integer variables and the *double precision type* for real variables. In the case of (1.21) the variables $\hat{\mathcal{N}}(i)$, $\hat{t}(i)$, τ, Δt, $\mathcal{N}_0$ are real numbers while the variables i and N are integer numbers.

An *array A* of dimension K is an ordered list of K variables of a given type called the elements of the array and denoted $A(1)$, $A(2)$,...,$A(K)$. In our above example $\hat{\mathcal{N}}(i)$ and $\hat{t}(i)$ are real arrays of dimension $N+1$. We declare that $\hat{\mathcal{N}}(i)$ and $\hat{t}(i)$ are real for all $i = 1, ..., N+1$ by writing $\hat{\mathcal{N}}(1:N+1)$ and $\hat{t}(1:N+1)$.

Since an array is declared at the begining of the program it must have a fixed size. In other words the upper limit must be a constant and not a variable. In Fortran a constant is declared with a *parameter statement*. In our above case the upper limit is $N+1$ and hence N must be declared in parameter statement.

In the Fortran code we choose to use the notation $A = \hat{\mathcal{N}}$, $A0 = \hat{\mathcal{N}}_0$, time $= \hat{t}$, $\Delta = \Delta t$ and tau $= \tau$. By putting all declarations together we get the following preliminary lines of code

```
program radioactivity
integer i,N
parameter (N=100)
doubleprecision  A(1:N+1),A0,time(1:N+1),Delta,tau


c Here is the code


return
end
```

The *input* of the computation in our case are obviously given by the parameters $\mathcal{N}_0$, τ, Δt and N.

For the radioactivity problem the main part of the code consists of equations (1.21) and (1.22). We start with the known quantities $\hat{\mathcal{N}}(1) = \mathcal{N}_0$ at $\hat{t}(1) = 0$ and generate via the successive use of (1.21) and (1.22) $\hat{\mathcal{N}}(i)$ and $\hat{t}(i)$ for all $i > 1$. This will be coded using a *do loop*. It begins with a *do statement* and ends with an *enddo statement*. We may also indicate a step size.

The *output* of the computation can be saved to a file using a *write statement* inside the do loop. In our case the output is the number of particles $\hat{\mathcal{N}}(i)$ and the time $\hat{t}(i)$. The write statement reads explicitly

$$\text{write}(10, *) \ \hat{t}(i), \hat{\mathcal{N}}(i).$$

The data will then be saved to a file called *fort.10*.

By including the initialization, the do loop and the write statement we obtain the complete code

```
program radioactivity
integer i,N
parameter (N=100)
doubleprecision  A(1:N+1),A0,time(1:N+1),Delta,tau
parameter (A0=1000,Delta=0.01d0,tau=1.0d0)

A(1)=A0
time(1)=0
do i=1,N+1,1
A(i+1)=A(i)-Delta*A(i)/tau
time(i+1)=i*Delta
write(10,*) time(i+1),A(i+1)
enddo

return
end
```

1.3   More Examples

1.3.1   Air Resistance

We consider an athlete riding a bicycle moving on a flat terrain. The goal is to determine the velocity. Newton's second law is given by

$$m\frac{dv}{dt} = F. \tag{1.23}$$

F is the force exerted by the athlete on the bicycle. It is clearly very difficult to write down a precise expression for F. Formulating the problem in terms of the power generated by

the athlete will avoid the use of an explicit formula for F. Multiplying the above equation by v we obtain

$$\frac{dE}{dt} = P. \tag{1.24}$$

E is the kinetic energy and P is the power, viz

$$E = \frac{1}{2}mv^2 \ , \ \ P = Fv. \tag{1.25}$$

Experimentaly we find that the output of well trained athletes is around $P = 400$ watts over periods of $1h$. The above equation can also be rewritten as

$$\frac{dv^2}{dt} = \frac{2P}{m}. \tag{1.26}$$

For P constant we get the solution

$$v^2 = \frac{2P}{m}t + v_0^2. \tag{1.27}$$

We remark the unphysical effect that $v \longrightarrow \infty$ as $t \longrightarrow \infty$. This is due to the absence of the effect of friction and in particular air resistance.

The most important form of friction is air resistance. The force due to air resistance (the drag force) is

$$F_{\text{drag}} = -B_1 v - B_2 v^2. \tag{1.28}$$

At small velocities the first term dominates whereas at large velocities it is the second term that dominates. For very small velocities the dependence on v given by $F_{\text{drag}} = -B_1 v$ is known as Stockes' law. For reasonable velocities the drag force is dominated by the second term, i.e. it is given for most objects by

$$F_{\text{drag}} = -B_2 v^2. \tag{1.29}$$

The coefficient B_2 can be calculated as follows. As the bicycle-rider combination moves with velocity v it pushes in a time dt a mass of air given by $dm_{\text{air}} = \rho A v dt$ where ρ is the air density and A is the frontal cross section. The corresponding kinetic energy is

$$dE_{\text{air}} = dm_{\text{air}} v^2 / 2. \tag{1.30}$$

This is equal to the work done by the drag force, i.e.

$$- F_{\text{drag}} v dt = dE_{\text{air}}. \tag{1.31}$$

From this we get

$$B_2 = C\rho A. \tag{1.32}$$

The drag coefficient is $C = \frac{1}{2}$. The drag force becomes

$$F_{\text{drag}} = -C\rho A v^2. \tag{1.33}$$

Taking into account the force due to air resistance we find that Newton's law becomes

$$m\frac{dv}{dt} = F + F_{\text{drag}}. \tag{1.34}$$

Equivalently

$$\frac{dv}{dt} = \frac{P}{mv} - \frac{C\rho A v^2}{m}. \tag{1.35}$$

It is not obvious that this equation can be solved exactly in any easy way. The Euler algorithm gives the approximate solution

$$v(i+1) \quad = \quad v(i) + \Delta t \frac{dv}{dt}(i). \tag{1.36}$$

In other words

$$v(i+1) \quad = \quad v(i) + \Delta t \left(\frac{P}{mv(i)} - \frac{C\rho A v^2(i)}{m} \right) \ , \ i = 0, ..., N. \tag{1.37}$$

This can also be put in the form (with $\hat{v}(i) = v(i-1)$)

$$\hat{v}(i+1) \quad = \quad \hat{v}(i) + \Delta t \left(\frac{P}{m\hat{v}(i)} - \frac{C\rho A \hat{v}^2(i)}{m} \right) \ , \ i = 1, ..., N+1. \tag{1.38}$$

The corresponding times are

$$t \equiv \hat{t}(i+1) = i\Delta t \ , \ i = 1, ..., N+1. \tag{1.39}$$

The initial velocity $\hat{v}(1)$ at time $t(1) = 0$ is known.

1.3.2 Projectile Motion

There are two forces acting on the projectile. The weight force and the drag force. The drag force is opposite to the velocity. In this case Newton's law is given by

$$
\begin{aligned}
m\frac{d\vec{v}}{dt} \quad &= \quad \vec{F} + \vec{F}_{\text{drag}} \\
&= \quad m\vec{g} - B_2 v^2 \frac{\vec{v}}{v} \\
&= \quad m\vec{g} - B_2 v\vec{v}. \tag{1.40}
\end{aligned}
$$

The goal is to determine the position of the projectile and hence one must solve the two equations

$$\frac{d\vec{x}}{dt} = \vec{v}. \tag{1.41}$$

$$m\frac{d\vec{v}}{dt} = m\vec{g} - B_2 v\vec{v}. \tag{1.42}$$

In components (the horizontal axis is x and the vertical axis is y) we have 4 equations of motion given by

$$\frac{dx}{dt} = v_x. \tag{1.43}$$

$$m\frac{dv_x}{dt} = -B_2 v v_x. \tag{1.44}$$

$$\frac{dy}{dt} = v_y. \tag{1.45}$$

$$m\frac{dv_y}{dt} = -mg - B_2 v v_y. \tag{1.46}$$

We recall the constraint

$$v = \sqrt{v_x^2 + v_y^2}. \tag{1.47}$$

The numerical approach we will employ in order to solve the 4 equations of motion (1.43)-(1.46) together with (1.47) consists in using Euler algorithm. This yields the approximate solution given by the equations

$$x(i+1) = x(i) + \Delta t v_x(i). \tag{1.48}$$

$$v_x(i+1) = v_x(i) - \Delta t \frac{B_2 v(i) v_x(i)}{m}. \tag{1.49}$$

$$y(i+1) = y(i) + \Delta t v_y(i). \tag{1.50}$$

$$v_y(i+1) = v_y(i) - \Delta t g - \Delta t \frac{B_2 v(i) v_y(i)}{m}. \tag{1.51}$$

The constraint is

$$v(i) = \sqrt{v_x(i)^2 + v_y(i)^2}. \tag{1.52}$$

In the above equations the index i is such that $i = 0, ..., N$. The initial position and velocity are given, i.e. $x(0)$, $y(0)$, $v_x(0)$ and $v_y(0)$ are known.

1.4　Periodic Motions and Euler-Cromer and Verlet Algorithms

As discussed above at each iteration using the Euler algorithm there is a systematic error proportional to $1/N$. Obviously this error will accumulate and may become so large that it will alter the solution drastically at later times. In the particular case of periodic motions, where the true nature of the motion can only become clear after few elapsed periods, the large accumulated error can lead to diverging results. In this section we will discuss simple variants of the Euler algorithm which perform much better than the plain Euler algorithm for periodic motions.

1.4.1   Harmonic Oscillator

We consider a simple pendulum: a particle of mass m suspended by a massless string from a rigid support. There are two forces acting on the particle. The weight and the tension of the string. Newton's second law reads

$$m\frac{d^2\vec{s}}{dt} = m\vec{g} + \vec{T}. \tag{1.53}$$

The parallel (with respect to the string) projection reads

$$0 = -mg\cos\theta + T. \tag{1.54}$$

The perpendicular projection reads

$$m\frac{d^2 s}{dt^2} = -mg\sin\theta. \tag{1.55}$$

The θ is the angle that the string makes with the vertical. Clearly $s = l\theta$. The force $mg\sin\theta$ is a restoring force which means that it is always directed toward the equilibrium position (here $\theta = 0$) opposite to the displacement and hence the minus sign in the above equation. We get by using $s = l\theta$ the equation

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin\theta. \tag{1.56}$$

For small θ we have $\sin\theta \simeq \theta$. We obtain

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\theta. \tag{1.57}$$

The solution is a sinusoidal function of time with frequency $\Omega = \sqrt{g/l}$. It is given by

$$\theta(t) = \theta_0\sin(\Omega t + \phi). \tag{1.58}$$

The constants θ_0 and ϕ depend on the initial displacement and velocity of the pendulum. The frequency is independent of the mass m and the amplitude of the motion and depends only on the length l of the string.

1.4.2   Euler Algorithm

The numerical solution is based on Euler algorithm. It is found as follows. First we replace the equation of motion (1.57) by the following two equations

$$\frac{d\theta}{dt} = \omega. \tag{1.59}$$

$$\frac{d\omega}{dt} = -\frac{g}{l}\theta. \tag{1.60}$$

We use the definition of a derivative of a function, viz

$$\frac{df}{dt} = \frac{f(t + \Delta t) - f(t)}{\Delta t} \ , \ \Delta t \longrightarrow 0. \tag{1.61}$$

We get for small but non zero Δt the approximations

$$\theta(t + \Delta t) \simeq \theta(t) + \omega(t)\Delta t$$
$$\omega(t + \Delta t) \simeq \omega(t) - \frac{g}{l}\theta(t)\Delta t. \tag{1.62}$$

We consider the time discretization

$$t \equiv t(i) = i\Delta t , \ \ i = 0, ..., N. \tag{1.63}$$

In other words

$$\theta(t) = \theta(i) , \ \omega(t) = \omega(i). \tag{1.64}$$

The integer N determine the total time interval $T = N\Delta t$. The above numerical solution can be rewritten as

$$\omega(i + 1) = \omega(i) - \frac{g}{l}\theta(i)\Delta t$$
$$\theta(i + 1) = \theta(i) + \omega(i)\Delta t. \tag{1.65}$$

We shift the integer i such that it takes values in the range $[1, N + 1]$. We obtain

$$\omega(i) = \omega(i - 1) - \frac{g}{l}\theta(i - 1)\Delta t$$
$$\theta(i) = \theta(i - 1) + \omega(i - 1)\Delta t. \tag{1.66}$$

We introduce $\hat{\omega}(i) = \omega(i-1)$ and $\hat{\theta}(i) = \theta(i-1)$. We get with $i = 1, ..., N+1$ the equations

$$\hat{\omega}(i + 1) = \hat{\omega}(i) - \frac{g}{l}\hat{\theta}(i)\Delta t$$
$$\hat{\theta}(i + 1) = \hat{\theta}(i) + \hat{\omega}(i)\Delta t. \tag{1.67}$$

By using the values of θ and ω at time i we calculate the corresponding values at time $i+1$. The initial angle and angular velocity $\hat{\theta}(1) = \theta(0)$ and $\hat{\omega}(1) = \omega(0)$ are known. This process will be repeated until the functions θ and ω are determined for all times.

1.4.3   Euler-Cromer Algorithm

As it turns out the above Euler algorithm does not conserve energy. In fact Euler's method is not good for all oscillatory systems. A simple modification of Euler's algorithm due to Cromer will solve this problem of energy non conservation. This goes as follows. We use the values of the angle $\hat{\theta}(i)$ and the angular velocity $\hat{\omega}(i)$ at time step i to calculate the angular velocity $\hat{\omega}(i+1)$ at time step $i+1$. This step is the same as before. However we use $\hat{\theta}(i)$ and $\hat{\omega}(i + 1)$ (and not $\hat{\omega}(i)$) to calculate $\hat{\theta}(i + 1)$ at time step $i + 1$. This procedure as shown by Cromer's will conserve energy in oscillatory problems. In other words equations (1.67) become

$$\hat{\omega}(i + 1) = \hat{\omega}(i) - \frac{g}{l}\hat{\theta}(i)\Delta t$$
$$\hat{\theta}(i + 1) = \hat{\theta}(i) + \hat{\omega}(i + 1)\Delta t. \tag{1.68}$$

The error can be computed as follows. From these two equations we get

$$
\begin{aligned}
\hat{\theta}(i+1) &= \hat{\theta}(i) + \hat{\omega}(i)\Delta t - \frac{g}{l}\hat{\theta}(i)\Delta t^2 \\
&= \hat{\theta}(i) + \hat{\omega}(i)\Delta t + \frac{d^2\hat{\theta}}{dt}|_i\Delta t^2. \tag{1.69}
\end{aligned}
$$

In other words the error per step is still of the order of Δt^2. However the Euler-Cromer algorithm does better than Euler algorithm with periodic motion. Indeed at each step i the energy conservation condition reads

$$
E_{i+1} = E_i + \frac{g}{2l}(\omega_i^2 - \frac{g}{l}\theta_i^2)\Delta t^2. \tag{1.70}
$$

The energy of the simple pendulum is of course by

$$
E_i = \frac{1}{2}\omega_i^2 + \frac{g}{2l}\theta_i^2. \tag{1.71}
$$

The error at each step is still proportional to Δt^2 as in the Euler algorithm. However the coefficient is precisely equal to the difference between the values of the kinetic energy and the potential energy at the step i. Thus the accumulated error which is obtained by summing over all steps vanishes since the average kinetic energy is equal to the average potential energy. In the Euler algorithm the coefficient is actually equal to the sum of the kinetic and potential energies and as consequence no cancellation can occur.

1.4.4   Verlet Algorithm

Another method which is much more accurate and thus very suited to periodic motions is due to Verlet. Let us consider the forward and backward Taylor expansions

$$
\theta(t_i + \Delta t) = \theta(t_i) + \Delta t\frac{d\theta}{dt}|_{t_i} + \frac{1}{2}(\Delta t)^2\frac{d^2\theta}{dt^2}|_{t_i} + \frac{1}{6}(\Delta t)^3\frac{d^3\theta}{dt^3}|_{t_i} + ... \tag{1.72}
$$

$$
\theta(t_i - \Delta t) = \theta(t_i) - \Delta t\frac{d\theta}{dt}|_{t_i} + \frac{1}{2}(\Delta t)^2\frac{d^2\theta}{dt^2}|_{t_i} - \frac{1}{6}(\Delta t)^3\frac{d^3\theta}{dt^3}|_{t_i} + ... \tag{1.73}
$$

Adding these expressions we get

$$
\theta(t_i + \Delta t) = 2\theta(t_i) - \theta(t_i - \Delta t) + (\Delta t)^2\frac{d^2\theta}{dt^2}|_{t_i} + O(\Delta^4). \tag{1.74}
$$

We write this as

$$
\theta_{i+1} = 2\theta_i - \theta_{i-1} - \frac{g}{l}(\Delta t)^2\theta_i. \tag{1.75}
$$

This is the Verlet algorithm for the harmonic oscillator. First we remark that the error is proportional to Δt^4 which is less than the errors in the Euler, Euler-Cromer (and even less than the error in the second-order Runge-Kutta) methods so this method is much more accurate. Secondly in this method we do not need to calculate the angular velocity $\omega = d\theta/dt$. Thirdly this method is not self-starting. In other words given the initial conditions θ_1 and ω_1 we need also to know θ_2 for the algorithm to start. We can for example determine θ_2 using the Euler method, viz $\theta_2 = \theta_1 + \Delta t\,\omega_1$.

1.5   Exercises

Exercise 1:   We give the differential equations

$$\frac{dx}{dt} = v. \tag{1.76}$$

$$\frac{dv}{dt} = a - bv. \tag{1.77}$$

- Write down the exact solutions.

- Write down the numerical solutions of these differential equations using Euler and Verlet methods and determine the corresponding errors.

Exercise 2:   The equation of motion of the solar system in polar coordinates is

$$\frac{d^2 r}{dt^2} = \frac{l^2}{r^3} - \frac{GM}{r^2}. \tag{1.78}$$

Solve this equation using Euler, Euler-Cromer and Verlet methods.

Exercise 3:   The equation of motion of a free falling object is

$$\frac{d^2 z}{dt^2} = -g. \tag{1.79}$$

- Write down the exact solution.

- Give a solution of this problem in terms of Euler method and determine the error.

- We choose the initial conditions $z = 0$, $v = 0$ at $t = 0$. Determine the position and the velocity between $t = 0$ and $t = 1$ for $N = 4$. Compare with the exact solution and compute the error in each step. Express the result in terms of $l = g\Delta t^2$.

- Give a solution of this problem in terms of Euler-Cromer and Verlet methods and determine the corresponding errors.

Exercise 4:   The equation governing population growth is

$$\frac{dN}{dt} = aN - bN^2. \tag{1.80}$$

The linear term represents the rate of birth while the quadratic term represents the rate of death. Give a solution of this problem in terms of the Euler and Verlet methods and determine the corresponding errors.

1.6   Simulation 1: Euler Algorithm- Air Resistance

The equation of motion of a cyclist exerting a force on his bicycle corresponding to a constant power P and moving against the force of air resistance is given by

$$\frac{dv}{dt} = \frac{P}{mv} - \frac{C\rho A v^2}{m}.$$

The numerical approximation of this first order differential equation which we will consider in this problem is based on Euler algorithm.

(1) Calculate the speed v as a function of time in the case of zero air resistance and then in the case of non-vanishing air resistance. What do you observe. We will take $P = 200$ and $C = 0.5$. We also give the values

$$m = 70\text{kg} \ , \ A = 0.33m^2 \ , \ \rho = 1.2\text{kg}/m^3 \ , \ \Delta t = 0.1s \ , \ T = 200s.$$

The initial speed is

$$\hat{v}(1) = 4m/s \ , \ \hat{t}(1) = 0.$$

(2) What do you observe if we change the drag coefficient and/or the power. What do you observe if we decrease the time step.

1.7   Simulation 2: Euler Algorithm- Projectile Motion

The numerical approximation based on the Euler algorithm of the equations of motion of a projectile moving under the effect of the forces of gravity and air resistance is given by the equations

$$v_x(i+1) = v_x(i) - \Delta t \frac{B_2 v(i) v_x(i)}{m}.$$

$$v_y(i+1) = v_y(i) - \Delta t g - \Delta t \frac{B_2 v(i) v_y(i)}{m}.$$

$$v(i+1) = \sqrt{v_x^2(i+1) + v_y^2(i+1)}.$$

$$x(i+1) = x(i) + \Delta t \ v_x(i).$$

$$y(i+1) = y(i) + \Delta t \ v_y(i).$$

(1) Write a Fortran code which implements the above Euler algorithm.

(2) We take the values

$$\frac{B_2}{m} = 0.00004 m^{-1} \ , \ g = 9.8 m/s^2.$$
$$v(1) = 700 m/s \ , \ \theta = 30 \text{ degree}.$$
$$v_x(1) = v(1)\cos\theta \ , \ v_y(1) = v(1)\sin\theta.$$
$$N = 10^5 \ , \ \Delta t = 0.01 s.$$

Calculate the trajectory of the projectile with and without air resistance. What do you observe.

(3) We can determine numerically the range of the projectile by means of the conditional instruction if. This can be done by adding inside the do loop the following condition

$$\text{if } (y(i+1).\text{le}.0) \text{ exit}$$

Determine the range of the projectile with and without air resistance.

(4) In the case where air resistance is absent we know that the range is maximal when the initial angle is 45 degrees. Verify this fact numerically by considering several angles. More precisely add a do loop over the initial angle in order to be able to study the range as a function of the initial angle.

(5) In the case where air resistance is non zero calculate the angle for which the range is maximal.

1.8 Simulation 3: Euler, Euler-Cromer and Verlet Algorithms

We will consider the numerical solutions of the equation of motion of a simple harmonic oscillator given by the Euler, Euler-Cromer and Verlet algorithms which take the form

$$\omega_{i+1} = \omega_i - \frac{g}{l}\theta_i \ \Delta t \ , \ \theta_{i+1} = \theta_i + \omega_i \ \Delta t \ , \ \text{Euler}.$$

$$\omega_{i+1} = \omega_i - \frac{g}{l}\theta_i \ \Delta t \ , \ \theta_{i+1} = \theta_i + \omega_{i+1} \ \Delta t \ , \ \text{Euler} - \text{Cromer}.$$

$$\theta_{i+1} = 2\theta_i - \theta_{i-1} - \frac{g}{l}\theta_i(\Delta t)^2 \ , \ \text{Verlet}.$$

(1) Write a Fortran code which implements the Euler, Euler-Cromer and Verlet algorithms for the harmonic oscillator problem.

(2) Calculate the angle, the angular velocity and the energy of the harmonic oscillator as functions of time. The energy of the harmonic oscillator is given by

$$E = \frac{1}{2}\omega^2 + \frac{1}{2}\frac{g}{l}\theta^2.$$

We take the values

$$g = 9.8 m/s^2 \ , l = 1m \ .$$

We take the number of iterations N and the time step Δt to be

$$N = 10000 \ , \ \Delta t = 0.05s.$$

The initial angle and the angular velocity are given by

$$\theta_1 = 0.1 \text{ radian} \ , \ \omega_1 = 0.$$

By using the conditional instruction if we can limit the total time of motion to be equal to say 5 periods as follows

$$\text{if } (t(i+1).ge.5*\text{period}) \text{ exit}.$$

(3) Compare between the value of the energy calculated with the Euler method and the value of the energy calculated with the Euler-Cromer method. What do you observe and what do you conclude.

(4) Repeat the computation using the Verlet algorithm. Remark that this method can not self-start from the initial values θ_1 and ω_1 only. We must also provide the angle θ_2 which can be calculated using for example Euler, viz

$$\theta_2 = \theta_1 + \omega_1 \ \Delta t.$$

We also remark that the Verlet algorithm does not require the calculation of the angular velocity. However in order to calculate the energy we need to evaluate the angular velocity which can be obtained from the expression

$$\omega_i = \frac{\theta_{i+1} - \theta_{i-1}}{2\Delta t}.$$

Chapter 2

Classical Numerical Integration

2.1 Rectangular Approximation

We consider a generic one dimensional integral of the form

$$F = \int_a^b f(x)dx. \tag{2.1}$$

In general this can not be done analytically. However this integral is straightforward to do numerically. The starting point is Riemann definition of the integral F as the area under the curve of the function $f(x)$ from $x = a$ to $x = b$. This is obtained as follows. We discretize the $x-$interval so that we end up with N equal small intervals of lenght Δx, viz

$$x_n = x_0 + n\Delta x \ , \ \Delta x = \frac{b-a}{N} \tag{2.2}$$

Clearly $x_0 = a$ and $x_N = b$. Riemann definition is then given by the following limit

$$F = \lim_{\left(\Delta x \longrightarrow 0 \ , \ N \longrightarrow \infty \ , \ b-a=\text{fixed}\right)} \left(\Delta x \sum_{n=0}^{N-1} f(x_n)\right). \tag{2.3}$$

The first approximation which can be made is to drop the limit. We get the so-called rectangular approximation given by

$$F_N = \Delta x \sum_{n=0}^{N-1} f(x_n). \tag{2.4}$$

General integration algorithms approximate the integral F by

$$F_N = \sum_{n=0}^{N} f(x_n)w_n. \tag{2.5}$$

In other words we evaluate the function $f(x)$ at $N+1$ points in the interval $[a,b]$ then we sum the values $f(x_n)$ with some corresponding weights w_n. For example in the rectangular approximation (2.4) the values $f(x_n)$ are summed with equal weights $w_n = \Delta x$, $n = 0, N-1$ and $w_N = 0$. It is also clear that the estimation F_N of the integral F becomes exact only in the large N limit.

2.2 Trapezoidal Approximation

The trapezoid rule states that we can approximate the integral by a sum of trapezoids. In the subinterval $[x_n, x_{n+1}]$ we replace the function $f(x)$ by a straight line connecting the two points $(x_n, f(x_n))$ and $(x_{n+1}, f(x_{n+1}))$. The trapezoid has as vertical sides the two straight lines $x = x_n$ and $x = x_{n+1}$. The base is the interval $\Delta x = x_{n+1} - x_n$. It is not difficult to convince ourselves that the area of this trapezoid is

$$\frac{(f(x_{n+1}) - f(x_n))\Delta x}{2} + f(x_n)\Delta x = \frac{(f(x_{n+1}) + f(x_n))\Delta x}{2}. \tag{2.6}$$

The integral F computed using the trapezoid approximation is therefore given by summing the contributions from all the N subinterval, viz

$$T_N = \sum_{n=0}^{N-1} \frac{(f(x_{n+1}) + f(x_n))\Delta x}{2} = \left(\frac{1}{2}f(x_0) + \sum_{n=1}^{N-1} f(x_n) + \frac{1}{2}f(x_N) \right)\Delta x. \tag{2.7}$$

We remark that the weights here are given by $w_0 = \Delta x/2$, $w_n = \Delta x$, $n = 1, ..., N-1$ and $w_N = \Delta x/2$.

2.3 Parabolic Approximation or Simpson's Rule

In this case we approximate the function in the subinterval $[x_n, x_{n+1}]$ by a parabola given by

$$f(x) = \alpha x^2 + \beta x + \gamma. \tag{2.8}$$

The area of the corresponding box is thus given by

$$\int_{x_n}^{x_{n+1}} dx(\alpha x^2 + \beta x + \gamma) = \left(\frac{\alpha x^3}{3} + \frac{\beta x^2}{2} + \gamma x \right)_{x_n}^{x_{n+1}}. \tag{2.9}$$

Let us go back and consider the integral

$$\int_{-1}^{1} dx(\alpha x^2 + \beta x + \gamma) = \frac{2\alpha}{3} + 2\gamma. \tag{2.10}$$

We remark that

$$f(-1) = \alpha - \beta + \gamma \ , \ f(0) = \gamma \ , \ f(1) = \alpha + \beta + \gamma. \tag{2.11}$$

Equivalently

$$\alpha = \frac{f(1) + f(-1)}{2} - f(0) \ , \ \beta = \frac{f(1) - f(-1)}{2} \ , \ \gamma = f(0). \tag{2.12}$$

Thus

$$\int_{-1}^{1} dx(\alpha x^2 + \beta x + \gamma) = \frac{f(-1)}{3} + \frac{4f(0)}{3} + \frac{f(1)}{3}. \tag{2.13}$$

In other words we can express the integral of the function $f(x) = \alpha x^2 + \beta x + \gamma$ over the interval $[-1, 1]$ in terms of the values of this function $f(x)$ at $x = -1, 0, 1$. Similarly we can express the integral of $f(x)$ over the adjacent subintervals $[x_{n-1}, x_n]$ and $[x_n, x_{n+1}]$ in terms of the values of $f(x)$ at $x = x_{n+1}, x_n, x_{n-1}$, viz

$$
\begin{aligned}
\int_{x_{n-1}}^{x_{n+1}} dx\ f(x) &= \int_{x_{n-1}}^{x_{n+1}} dx(\alpha x^2 + \beta x + \gamma) \\
&= \Delta x \left(\frac{f(x_{n-1})}{3} + \frac{4f(x_n)}{3} + \frac{f(x_{n+1})}{3} \right).
\end{aligned}
\tag{2.14}
$$

By adding the contributions from each pair of adjacent subintervals we get the full integral

$$
S_N = \Delta x \sum_{p=0}^{\frac{N-2}{2}} \left(\frac{f(x_{2p})}{3} + \frac{4f(x_{2p+1})}{3} + \frac{f(x_{2p+2})}{3} \right).
\tag{2.15}
$$

Clearly we must have N (the number of subintervals) even. We compute

$$
S_N = \frac{\Delta x}{3} \left(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + ... + 2f(x_{N-2}) + 4f(x_{N-1}) + f(x_N) \right).
\tag{2.16}
$$

It is trivial to read from this expression the weights in this approximation.

Let us now recall the trapezoidal approximation given by

$$
T_N = \left(f(x_0) + 2 \sum_{n=1}^{N-1} f(x_n) + f(x_N) \right) \frac{\Delta x}{2}.
\tag{2.17}
$$

Let us also recall that $N\Delta x = b - a$ is the length of the total interval which is always kept fixed. Thus by doubling the number of subintervals we halve the width, viz

$$
\begin{aligned}
4T_{2N} &= \left(2f(\hat{x}_0) + 4 \sum_{n=1}^{2N-1} f(\hat{x}_n) + 2f(\hat{x}_{2N}) \right) \frac{\Delta x}{2} \\
&= \left(2f(\hat{x}_0) + 4 \sum_{n=1}^{N-1} f(\hat{x}_{2n}) + 4 \sum_{n=0}^{N-1} f(\hat{x}_{2n+1}) + 2f(\hat{x}_{2N}) \right) \frac{\Delta x}{2} \\
&= \left(2f(x_0) + 4 \sum_{n=1}^{N-1} f(x_n) + 4 \sum_{n=0}^{N-1} f(\hat{x}_{2n+1}) + 2f(x_N) \right) \frac{\Delta x}{2}.
\end{aligned}
\tag{2.18}
$$

In above we have used the identification $\hat{x}_{2n} = x_n$, $n = 0, 1, ..., N-1, N$. Thus

$$
\begin{aligned}
4T_{2N} - T_N &= \left(f(x_0) + 2 \sum_{n=1}^{N-1} f(x_n) + 4 \sum_{n=0}^{N-1} f(\hat{x}_{2n+1}) + f(x_N) \right) \Delta \hat{x} \\
&= 3S_N.
\end{aligned}
\tag{2.19}
$$

2.4 Errors

The error estimates for numerical integration are computed as follows. We start with the Taylor expansion

$$f(x) = f(x_n) + (x - x_n)f^{(1)}(x_n) + \frac{1}{2!}(x - x_n)^2 f^{(2)}(x_n) + ... \tag{2.20}$$

Thus

$$\int_{x_n}^{x_{n+1}} dx \ f(x) = f(x_n)\Delta x + \frac{1}{2!}f^{(1)}(x_n)(\Delta x)^2 + \frac{1}{3!}f^{(2)}(x_n)(\Delta x)^3 + ... \tag{2.21}$$

The error in the interval $[x_n, x_{n+1}]$ in the rectangular approximation is

$$\int_{x_n}^{x_{n+1}} dx \ f(x) - f(x_n)\Delta x = \frac{1}{2!}f^{(1)}(x_n)(\Delta x)^2 + \frac{1}{3!}f^{(2)}(x_n)(\Delta x)^3 + ... \tag{2.22}$$

This is of order $1/N^2$. But we have N subintervals. Thus the total error is of order $1/N$.

The error in the interval $[x_n, x_{n+1}]$ in the trapezoidal approximation is

$$
\begin{aligned}
\int_{x_n}^{x_{n+1}} dx \ f(x) - \frac{1}{2}(f(x_n) + f(x_{n+1}))\Delta x \ &= \ \int_{x_n}^{x_{n+1}} dx \ f(x) \\
&\quad - \frac{1}{2}(2f(x_n) + \Delta x f^{(1)}(x_n) + \frac{1}{2!}(\Delta x)^2 f^{(2)}(x_n) + ...)\Delta x \\
&= \ (\frac{1}{3!} - \frac{1}{2}\frac{1}{2!})f^{(2)}(x_n)(\Delta x)^3 + ...
\end{aligned}
\tag{2.23}
$$

This is of order $1/N^3$ and thus the total error is of order $1/N^2$.

In order to compute the error in the interval $[x_{n-1}, x_{n+1}]$ in the parabolic approximation we compute

$$\int_{x_{n-1}}^{x_n} dx \ f(x) + \int_{x_n}^{x_{n+1}} dx \ f(x) \ = \ 2f(x_n)\Delta x + \frac{2}{3!}(\Delta x)^3 f^{(2)}(x_n) + \frac{2}{5!}(\Delta x)^5 f^{(4)}(x_n) + ... \tag{2.24}$$

Also we compute

$$\frac{\Delta x}{3}(f(x_{n+1}) + f(x_{n-1}) + 4f(x_n)) \ = \ 2f(x_n)\Delta x + \frac{2}{3!}(\Delta x)^3 f^{(2)}(x_n) + \frac{2}{3.4!}(\Delta x)^5 f^{(4)}(x_n) + ... \tag{2.25}$$

Hence the error in the interval $[x_{n-1}, x_{n+1}]$ in the parabolic approximation is

$$\int_{x_{n-1}}^{x_{n+1}} dx \ f(x) - \frac{\Delta x}{3}(f(x_{n+1}) + f(x_{n-1}) + 4f(x_n)) \ = \ (\frac{2}{5!} - \frac{2}{3.4!})(\Delta x)^5 f^{(4)}(x_n) + ... \tag{2.26}$$

This is of order $1/N^5$. The total error is therefore of order $1/N^4$.

2.5   Simulation 4: Numerical Integrals

(1) We take the integral

$$I = \int_0^1 f(x)dx \ ; \ f(x) = 2x + 3x^2 + 4x^3.$$

Calculate the value of this integral using the rectangular approximation. Compare with the exact result.

Hint: You can code the function using either "subroutine" or "function".

(2) Calculate the numerical error as a function of N. Compare with the theory.

(3) Repeat the computation using the trapezoid method and the Simpson's rule.

(4) Take now the integrals

$$I = \int_0^{\frac{\pi}{2}} \cos x dx \ , \quad I = \int_1^e \frac{1}{x}dx \ , \quad I = \int_{-1}^{+1} \lim_{\epsilon \longrightarrow 0} \left(\frac{1}{\pi} \frac{\epsilon}{x^2 + \epsilon^2} \right) dx.$$

Chapter 3

Newton-Raphson Algorithms and Interpolation

3.1   Bisection Algorithm

Let f be some function. We are interested in the solutions (roots) of the equation

$$f(x) = 0. \tag{3.1}$$

The bisection algorithm works as follows. We start with two values of x say x_+ and x_- such that

$$f(x_-) < 0 \ , \ f(x_+) > 0. \tag{3.2}$$

In other words the function changes sign in the interval between x_- and x_+ and thus there must exist a root between x_- and x_+. If the function changes from positive to negative as we increase x we conclude that $x_+ \leq x_-$. We bisect the interval $[x_+, x_-]$ at

$$x = \frac{x_+ + x_-}{2}. \tag{3.3}$$

If $f(x)f(x_+) > 0$ then x_+ will be changed to the point x otherwise x_- will be changed to the point x. We continue this process until the change in x becomes insignificant or until the error becomes smaller than some tolerance. The relative error is defined by

$$\text{error} = \frac{x_+ - x_-}{x}. \tag{3.4}$$

Clearly the absolute error $e = x_i - x_f$ is halved at each iteration and thus the rate of convergence of the bisection rule is linear. This is slow.

3.2   Newton-Raphson Algorithm

We start with a guess x_0. The new guess x is written as x_0 plus some unknown correction Δx, viz

$$x = x_0 + \Delta x. \tag{3.5}$$

Next we expand the function $f(x)$ around x_0, namely

$$f(x) = f(x_0) + \Delta x \frac{df}{dx}|_{x=x_0}. \tag{3.6}$$

The correction Δx is determined by finding the intersection point of this linear approximation of $f(x)$ with the x axis. Thus

$$f(x_0) + \Delta x \frac{df}{dx}|_{x=x_0} = 0 \Longrightarrow \Delta x = -\frac{f(x_0)}{(df/dx)|_{x=x_0}}. \tag{3.7}$$

The derivative of the function f is required in this calculation. In complicated problems it is much simpler to evaluate the derivative numerically than analytically. In these cases the derivative may be given by the forward-difference approximation (with some δx not necessarily equal to Δx)

$$\frac{df}{dx}|_{x=x_0} = \frac{f(x_0 + \delta x) - f(x_0)}{\delta x}. \tag{3.8}$$

In summary this method works by drawing the tangent to the function $f(x)$ at the old guess x_0 and then use the intercept with the x axis as the new hopefully better guess x. The process is repeated until the change in x becomes insignificant.

Next we compute the rate of convergence of the Newton-Raphson algorithm. Starting from x_i the next guess is x_{i+1} given by

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x)}. \tag{3.9}$$

The absolute error at step i is $\epsilon_i = x - x_i$ while the absolute error at step $i + 1$ is $\epsilon_{i+1} = x - x_{i+1}$ where x is the actual root. Then

$$\epsilon_{i+1} = \epsilon_i + \frac{f(x_i)}{f'(x)}. \tag{3.10}$$

By using Taylor expansion we have

$$f(x) = 0 = f(x_i) + (x - x_i)f'(x_i) + \frac{(x - x_i)^2}{2!}f''(x_i) + ... \tag{3.11}$$

In other words

$$f(x_i) = -\epsilon_i f'(x_i) - \frac{\epsilon_i^2}{2!}f''(x_i) + ... \tag{3.12}$$

Therefore the error is given by

$$\epsilon_{i+1} = -\frac{\epsilon_i^2}{2}\frac{f''(x_i)}{f'(x_i)}. \tag{3.13}$$

This is quadratic convergence. This is faster than the bisection rule.

3.3 Hybrid Method

We can combine the certainty of the bisection rule in finding a root with the fast convergence of the Newton-Raphson algorithm into a hybrid algorithm as follows. First we must know that the root is bounded in some interval $[a, c]$. We can use for example a graphical method. Next we start from some initial guess b. We take a Newton-Raphson step

$$b' = b - \frac{f(b)}{f'(b)}. \tag{3.14}$$

We check whether or not this step is bounded in the interval $[a, c]$. In other words we must check that

$$a \leq b - \frac{f(b)}{f'(b)} \leq c \Leftrightarrow (b - c)f'(b) - f(b) \leq 0 \leq (b - a)f'(b) - f(b). \tag{3.15}$$

Therefore if

$$\left((b - c)f'(b) - f(b)\right)\left((b - a)f'(b) - f(b)\right) < 0 \tag{3.16}$$

Then the Newton-Raphson step is accepted else we take instead a bisection step.

3.4 Lagrange Interpolation

Let us first recall that taylor expansion allows us to approximate a function at a point x if the function and its derivatives are known in some neighbouring point x_0. The lagrange interpolation tries to approximate a function at a point x if only the values of the function in several other points are known. Thus this method does not require the knowledge of the derivatives of the function. We start from taylor expansion

$$f(y) = f(x) + (y - x)f'(x) + \frac{1}{2!}(y - x)^2 f''(x) + .. \tag{3.17}$$

Let us assume that the function is known at three points x_1, x_2 and x_3. In this case we can approximate the function $f(x)$ by some function $p(x)$ and write

$$f(y) = p(x) + (y - x)p'(x) + \frac{1}{2!}(y - x)^2 p''(x). \tag{3.18}$$

We have

$$f(x_1) = p(x) + (x_1 - x)p'(x) + \frac{1}{2!}(x_1 - x)^2 p''(x)$$
$$f(x_2) = p(x) + (x_2 - x)p'(x) + \frac{1}{2!}(x_2 - x)^2 p''(x)$$
$$f(x_3) = p(x) + (x_3 - x)p'(x) + \frac{1}{2!}(x_3 - x)^2 p''(x). \tag{3.19}$$

We can immediately find

$$p(x) = \frac{1}{1 + a_2 + a_3}f(x_1) + \frac{a_2}{1 + a_2 + a_3}f(x_2) + \frac{a_3}{1 + a_2 + a_3}f(x_3). \tag{3.20}$$

The coefficients a_2 and a_3 solve the equations

$$a_2(x_2 - x)^2 + a_3(x_3 - x)^2 = -(x_1 - x)^2$$
$$a_2(x_2 - x) + a_3(x_3 - x) = -(x_1 - x). \tag{3.21}$$

We find

$$a_2 = \frac{(x_1 - x)(x_3 - x_1)}{(x_2 - x)(x_2 - x_3)} \;,\; a_3 = -\frac{(x_1 - x)(x_2 - x_1)}{(x_3 - x)(x_2 - x_3)}. \tag{3.22}$$

Thus

$$1 + a_2 + a_3 = \frac{(x_3 - x_1)(x_2 - x_1)}{(x_2 - x)(x_3 - x)}. \tag{3.23}$$

Therefore we get

$$p(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} f(x_1) + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} f(x_2) + \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} f(x_3). \tag{3.24}$$

This is a quadratic polynomial.

Let x be some independent variable with tabulated values x_i, $i = 1, 2, ..., n..$ The dependent variable is a function $f(x)$ with tabulated values $f_i = f(x_i)$. Let us then assume that we can approximate $f(x)$ by a polynomial of degree $n - 1$, viz

$$p(x) = a_0 + a_1 x + a_2 x^2 + ... + a_{n-1} x^{n-1}. \tag{3.25}$$

A polynomial which goes through the n points $(x_i, f_i = f(x_i))$ was given by Lagrange. This is given by

$$p(x) = f_1 \lambda_1(x) + f_2 \lambda_2(x) + ... + f_n \lambda_n(x). \tag{3.26}$$

$$\lambda_i(x) = \prod_{j(\neq i)=1}^{n} \frac{x - x_j}{x_i - x_j}. \tag{3.27}$$

We remark

$$\lambda_i(x_j) = \delta_{ij}. \tag{3.28}$$

$$\sum_{i=1}^{n} \lambda_i(x) = 1. \tag{3.29}$$

The Lagrange polynomial can be used to fit the entire table with n equal the number of points in the table. But it is preferable to use the Lagrange polynomial to to fit only a small region of the table with a small value of n. In other words use several polynomials to cover the whole table and the fit considered here is local and not global.

3.5   Cubic Spline Interpolation

We consider n points $(x_1, f(x_1)),(x_2, f(x_2)),...,(x_n, f(x_n))$ in the plane. In every interval $x_j \leq x \leq x_{j+1}$ we approximate the function $f(x)$ with a cubic polynomial of the form

$$p(x) = a_j(x - x_j)^3 + b_j(x - x_j)^2 + c_j(x - x_j) + d_j. \tag{3.30}$$

We assume that

$$p_j = p(x_j) = f(x_j). \tag{3.31}$$

In other words the p_j for all $j = 1, 2, ..., n - 1$ are known. From the above equation we conclude that

$$d_j = p_j. \tag{3.32}$$

We compute

$$p^{'}(x) = 3a_j(x - x_j)^2 + 2b_j(x - x_j) + c_j. \tag{3.33}$$

$$p^{''}(x) = 6a_j(x - x_j) + 2b_j. \tag{3.34}$$

Thus we get by substituting $x = x_j$ into $p^{''}(x)$ the result

$$b_j = \frac{p^{''}_j}{2}. \tag{3.35}$$

By substituting $x = x_{j+1}$ into $p^{''}(x)$ we get the result

$$a_j = \frac{p^{''}_{j+1} - p^{''}_j}{6h_j}. \tag{3.36}$$

By substituting $x = x_{j+1}$ into $p(x)$ we get

$$p_{j+1} = a_j h_j^3 + b_j h_j^2 + c_j h_j + p_j. \tag{3.37}$$

By using the values of a_j and b_j we obtain

$$c_j = \frac{p_{j+1} - p_j}{h_j} - \frac{h_j}{6}(p^{''}_{j+1} + 2p^{''}_j). \tag{3.38}$$

Hence

$$p(x) = \frac{p^{''}_{j+1} - p^{''}_j}{6h_j}(x - x_j)^3 + \frac{p^{''}_j}{2}(x - x_j)^2 + \left(\frac{p_{j+1} - p_j}{h_j} - \frac{h_j}{6}(p^{''}_{j+1} + 2p^{''}_j) \right)(x - x_j) + p_j. \tag{3.39}$$

In other words the polynomials are determined from p_j and $p^{''}_j$. The p_j are known given by $p_j = f(x_j)$. It remains to determine $p^{''}_j$. We take the derivative of the above equation

$$p^{'}(x) = \frac{p^{''}_{j+1} - p^{''}_j}{2h_j}(x - x_j)^2 + p^{''}_j(x - x_j) + \left(\frac{p_{j+1} - p_j}{h_j} - \frac{h_j}{6}(p^{''}_{j+1} + 2p^{''}_j) \right). \tag{3.40}$$

This is the derivative in the interval $[x_j, x_{j+1}]$. We compute

$$p^{'}(x_j) = \left(\frac{p_{j+1} - p_j}{h_j} - \frac{h_j}{6}(p^{''}_{j+1} + 2p^{''}_j) \right). \tag{3.41}$$

The derivative in the interval $[x_{j-1}, x_j]$ is

$$p^{'}(x) = \frac{p^{''}_j - p^{''}_{j-1}}{2h_{j-1}}(x - x_{j-1})^2 + p^{''}_{j-1}(x - x_{j-1}) + \left(\frac{p_j - p_{j-1}}{h_{j-1}} - \frac{h_{j-1}}{6}(p^{''}_j + 2p^{''}_{j-1}) \right). \tag{3.42}$$

We compute

$$p^{'}(x_j) = \frac{p^{''}_j - p^{''}_{j-1}}{2}h_{j-1} + p^{''}_{j-1}h_{j-1} + \left(\frac{p_j - p_{j-1}}{h_{j-1}} - \frac{h_{j-1}}{6}(p^{''}_j + 2p^{''}_{j-1}) \right). \tag{3.43}$$

By matching the two expressions for $p^{'}(x_j)$ we get

$$h_{j-1}p^{''}_{j-1} + 2(h_j + h_{j-1})p^{''}_j + h_j p^{''}_{j+1} = 6 \left(\frac{p_{j+1} - p_j}{h_j} - \frac{p_j - p_{j-1}}{h_{j-1}} \right). \tag{3.44}$$

These are $n - 2$ equations since $j = 2, ..., n - 1$ for n unknown $p^{''}_j$. We need two more equations. These are obtained by computing the first derivative $p^{'}(x)$ at $x = x_1$ and $x = x_n$. We obtain the two equations

$$h_1(p^{''}_2 + 2p^{''}_1) = \frac{6(p_2 - p_1)}{h_1} - 6p^{'}_1. \tag{3.45}$$

$$h_{n-1}(p^{''}_{n-1} + 2p^{''}_n) = -\frac{6(p_n - p_{n-1})}{h_{n-1}} + 6p^{'}_n. \tag{3.46}$$

The n equations (3.44), (3.45) and (3.46) correspond to a tridiagonal linear system. In general $p^{'}_1$ and $p^{'}_n$ are not known. In this case we may use natural spline in which the second derivative vanishes at the end points and hence

$$\frac{p_2 - p_1}{h_1} - p^{'}_1 = \frac{p_n - p_{n-1}}{h_{n-1}} - p^{'}_n = 0. \tag{3.47}$$

3.6   The Method of Least Squares

We assume that we have N data points $(x(i), y(i))$. We want to fit this data to some curve say a straight line $y_{\text{fit}} = mx + b$. To this end we define the function

$$\Delta = \sum_{i=1}^{N}(y(i) - y_{\text{fit}}(i))^2 = \sum_{i=1}^{N}(y(i) - mx(i) - b)^2. \tag{3.48}$$

The goal is to minimize this function with respect to b and m. We have

$$\frac{\partial \Delta}{\partial m} = 0 \ , \ \frac{\partial \Delta}{\partial b} = 0. \tag{3.49}$$

We get the solution

$$b = \frac{\sum_i x(i) \sum_j x(j)y(j) - \sum_i x(i)^2 \sum_j y(j)}{(\sum_i x(i))^2 - N \sum_i x_i^2}. \tag{3.50}$$

$$m = \frac{\sum_i x(i) \sum_j y(j) - N \sum_i x(i)y(i)}{(\sum_i x(i))^2 - N \sum_i x_i^2}. \tag{3.51}$$

3.7   Simulation 5: Newton-Raphson Algorithm

A particle of mass m moves inside a potential well of height V and length $2a$ centered around 0. We are interested in the states of the system which have energies less than V, i.e. bound states. The states of the system can be even or odd. The energies associated with the even wave functions are solutions of the transcendental equation

$$\alpha \tan \alpha a = \beta.$$

$$\alpha = \sqrt{\frac{2mE}{\hbar^2}} \ , \ \beta = \sqrt{\frac{2m(V-E)}{\hbar^2}}.$$

In the case of the infinite potential well we find the solutions

$$E_n = \frac{(n+\frac{1}{2})^2 \pi^2 \hbar^2}{2ma^2} \ , \ n = 0, 1....$$

We choose (dropping units)

$$\hbar = 1 \ , \ a = 1 \ , \ 2m = 1.$$

In order to find numerically the energies E_n we will use the Newton-Raphson algorithm which allows us to find the roots of the equation $f(x) = 0$ as follows. From an initial guess x_0, the first approximation x_1 to the solution is determined from the intersection of the tangent to the function $f(x)$ at x_0 with the $x-$axis. This is given by

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Next by using x_1 we repeat the same step in order to find the second approximation x_2 to the solution. In general the approximation x_{i+1} to the desired solution in terms of the approximation x_i is given by the equation

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}.$$

(1) For $V = 10$, determine the solutions using the graphical method. Consider the two functions

$$f(\alpha) = \tan \alpha a \ , \ g(\alpha) = \frac{\beta}{\alpha} = \sqrt{\frac{V}{\alpha^2} - 1}.$$

(2) Find using the method of Newton-Raphson the two solutions with a tolerance equal 10^{-8}. For the first solution we take the initial guess $\alpha = \pi/a$ and for the second solution we take the initial guess $\alpha = 2\pi/a$.

(3) Repeat for $V = 20$.

(4) Find the 4 solutions for $V = 100$. Use the graphical method to determine the initial step each time.

(5) Repeat the above questions using the bisection method.

Chapter 4

The Solar System-The Runge-Kutta Methods

4.1 The Solar System

4.1.1 Newton's Second Law

We consider the motion of the Earth around the Sun. Let r be the distance and M_s and M_e be the masses of the Sun and the Earth respectively. We neglect the effect of the other planets and the motion of the Sun (i.e. we assume that $M_s >> M_e$). The goal is to calculate the position of the Earth as a function of time. We start from Newton's second law of motion

$$
\begin{aligned}
M_e \frac{d^2\vec{r}}{dt^2} &= -\frac{GM_eM_s}{r^3}\vec{r} \\
&= -\frac{GM_eM_s}{r^3}(x\vec{i} + y\vec{j}).
\end{aligned}
\tag{4.1}
$$

We get the two equations

$$
\frac{d^2x}{dt^2} = -\frac{GM_s}{r^3}x.
\tag{4.2}
$$

$$
\frac{d^2y}{dt^2} = -\frac{GM_s}{r^3}y.
\tag{4.3}
$$

We replace these two second-order differential equations by the four first-order differential equations

$$
\frac{dx}{dt} = v_x.
\tag{4.4}
$$

$$
\frac{dv_x}{dt} = -\frac{GM_s}{r^3}x.
\tag{4.5}
$$

$$\frac{dy}{dt} = v_y. \tag{4.6}$$

$$\frac{dv_y}{dt} = -\frac{GM_s}{r^3}y. \tag{4.7}$$

We recall

$$r = \sqrt{x^2 + y^2}. \tag{4.8}$$

4.1.2 Astronomical Units and Initial Conditions

The distance will be measured in astronomical units (AU) whereas time will be measured in years. One astronomical unit of lenght (1 AU) is equal to the average distance between the earth and the sun, viz $1\text{AU} = 1.5 \times 10^{11}m$. The astronomical unit of mass can be found as follows. Assuming a circular orbit we have

$$\frac{M_e v^2}{r} = \frac{GM_s M_e}{r^2}. \tag{4.9}$$

Equivalently

$$GM_s = v^2 r. \tag{4.10}$$

The radius is $r = 1\text{AU}$. The velocity of the earth is $v = 2\pi r/\text{yr} = 2\pi\text{AU}/\text{yr}$. Hence

$$GM_s = 4\pi^2 \text{AU}^3/\text{yr}^2. \tag{4.11}$$

For the numerical simulations it is important to determine the correct initial conditions. The orbit of Mercury is known to be an ellipse with eccentricity $e = 0.206$ and radius (semimajor axis) $a = 0.39$ AU with the Sun at one of the foci. The distance between the Sun and the center is ea. The first initial condition is $x_0 = r_1$, $y_0 = 0$ where r_1 is the maximum distance from Mercury to the Sun,i.e. $r_1 = (1 + e)a = 0.47$ AU. The second initial condition is the velocity $(0, v_1)$ which can be computed using conservation of energy and angular momentum. For example by comparing with the point $(0, b)$ on the orbit where b is the semiminor axis, i.e $b = a\sqrt{1 - e^2}$ the velocity $(v_2, 0)$ there can be obtained in terms of $(0, v_1)$ from conservation of angular momentum as follows

$$r_1 v_1 = b v_2 \Leftrightarrow v_2 = \frac{r_1 v_1}{b}. \tag{4.12}$$

Next conservation of energy yields

$$-\frac{GM_s M_m}{r_1} + \frac{1}{2}M_m v_1^2 = -\frac{GM_s M_m}{r_2} + \frac{1}{2}M_m v_2^2. \tag{4.13}$$

In above $r_2 = \sqrt{e^2 a^2 + b^2}$ is the distance between the Sun and Mercury when at the point $(0, b)$. By substituting the value of v_2 we get an equation for v_1. This is given by

$$v_1 = \sqrt{\frac{GM_s}{a}\frac{1 - e}{1 + e}} = 8.2 \text{ AU/yr}. \tag{4.14}$$

4.1.3   Kepler's Laws

Kepler's laws are given by the following three statements:

- The planets move in elliptical orbits around the sun. The sun resides at one focus.

- The line joining the sun with any planet sweeps out equal areas in equal times.

- Given an orbit with a period T and a semimajor axis a the ratio T^2/a^3 is a constant.

The derivation of these three laws proceeds as follows. We work in polar coordinates. Newton's second law reads

$$M_e \ddot{\vec{r}} = -\frac{GM_s M_e}{r^2}\hat{r}. \tag{4.15}$$

We use $\dot{\hat{r}} = \dot{\theta}\hat{\theta}$ and $\dot{\hat{\theta}} = -\dot{\theta}\hat{r}$ to derive $\dot{\vec{r}} = \dot{r}\hat{r} + r\dot{\theta}\hat{\theta}$ and $\ddot{\vec{r}} = (\ddot{r} - r\dot{\theta}^2)\hat{r} + (r\ddot{\theta} + 2\dot{r}\dot{\theta})\hat{\theta}$. Newton's second law decomposes into the two equations

$$r\ddot{\theta} + 2\dot{r}\dot{\theta} = 0. \tag{4.16}$$

$$\ddot{r} - r\dot{\theta}^2 = -\frac{GM_s}{r^2}. \tag{4.17}$$

Let us recall that the angular momentum by unit mass is defined by $\vec{l} = \vec{r} \times \dot{\vec{r}} = r^2\dot{\theta}\hat{r} \times \hat{\theta}$. Thus $l = r^2\dot{\theta}$. Equation (4.16) is precisely the requirement that angular momentum is conserved. Indeed we compute

$$\frac{dl}{dt} = r(r\ddot{\theta} + 2\dot{r}\dot{\theta}) = 0. \tag{4.18}$$

Now we remark that the area swept by the vector $\vec{r}$ in a time interval dt is $dA = (r \times rd\theta)/2$ where $d\theta$ is the angle traveled by $\vec{r}$ during dt. Clearly

$$\frac{dA}{dt} = \frac{1}{2}l. \tag{4.19}$$

In other words the planet sweeps equal areas in equal times since l is conserved. This is Kepler's second law.

The second equation (4.17) becomes now

$$\ddot{r} = \frac{l^2}{r^3} - \frac{GM_s}{r^2} \tag{4.20}$$

By multiplying this equation with $\dot{r}$ we obtain

$$\frac{d}{dt}E = 0 \ , \ E = \frac{1}{2}\dot{r}^2 + \frac{l^2}{2r^2} - \frac{GM_s}{r}. \tag{4.21}$$

This is precisely the statement of conservation of energy. E is the energy per unit mass. Solving for dt in terms of dr we obtain

$$dt = \frac{dr}{\sqrt{2\left(E - \frac{l^2}{2r^2} + \frac{GM_s}{r}\right)}} \tag{4.22}$$

However $dt = (r^2 d\theta)/l$. Thus

$$d\theta = \frac{l dr}{r^2 \sqrt{2\left(E - \frac{l^2}{2r^2} + \frac{GM_s}{r}\right)}} \tag{4.23}$$

By integrating this equation we obtain (with $u = 1/r$)

$$
\begin{aligned}
\theta &= \int \frac{l dr}{r^2 \sqrt{2\left(E - \frac{l^2}{2r^2} + \frac{GM_s}{r}\right)}} \\
&= -\int \frac{du}{\sqrt{\frac{2E}{l^2} + \frac{2GM_s}{l^2} u - u^2}}.
\end{aligned} \tag{4.24}
$$

This integral can be done explicitly. We get

$$\theta = -\arccos\left(\frac{u - C}{eC}\right) + \theta' \ , \ e = \sqrt{1 + \frac{2l^2 E}{G^2 M_s^2}} \ , \ C = \frac{GM_s}{l^2}. \tag{4.25}$$

By inverting this equation we get an equation of ellipse with eccentricity e since $E < 0$, viz

$$\frac{1}{r} = C(1 + e\cos(\theta - \theta')). \tag{4.26}$$

This is Kepler's first law. The angle at which r is maximum is $\theta - \theta' = \pi$. This distance is precisely $(1 + e)a$ where a is the semi-major axis of the ellipse since ea is the distance between the Sun which is at one of the two foci and the center of the ellipse. Hence we obtain the relation

$$(1 - e^2)a = \frac{1}{C} = \frac{l^2}{GM_s}. \tag{4.27}$$

From equation (4.19) we can derive Kepler's third law. By integrating both sides of the equation over a single period T and then taking the square we get

$$A^2 = \frac{1}{4} l^2 T^2. \tag{4.28}$$

A is the area of the ellipse, i.e. $A = \pi ab$ where the semi-minor axis b is related the semi-major axis a by $b = a\sqrt{1 - e^2}$. Hence

$$\pi^2 a^4 (1 - e^2) = \frac{1}{4} l^2 T^2. \tag{4.29}$$

By using equation (4.27) we get the desired formula

$$\frac{T^2}{a^3} = \frac{4\pi^2}{GM_s}. \tag{4.30}$$

4.1.4   The inverse-Square Law and Stability of Orbits

Any object with mass generates a gravitational field and thus gravitational field lines will emanate from the object and radiate outward to infinity. The number of field lines N is proportional to the mass. The density of field lines crossing a sphere of radius r surrounding this object is given by $N/4\pi r^2$. This is the origin of the inverse-square law. Therefore any other object placed in this gravitational field will experience a gravitational force proportional to the number of field lines which intersect it. If the distance between this second object and the source is increased the force on it will become weaker because the number of field lines which intersect it will decrease as we are further away from the source.

4.2   Euler-Cromer Algorithm

The time discretization is

$$t \equiv t(i) = i\Delta t \ , \ \ i = 0, ..., N. \tag{4.31}$$

The total time interval is $T = N\Delta t$. We define $x(t) = x(i)$, $v_x(t) = v_x(i)$, $y(t) = y(i)$, $v_y(t) = v_y(i)$. Equations (4.4), (4.5), (4.6),(4.7) and (4.8) become (with $i = 0, ..., N$)

$$v_x(i+1) = v_x(i) - \frac{GM_s}{(r(i))^3} x(i)\Delta t. \tag{4.32}$$

$$x(i+1) = x(i) + v_x(i)\Delta t. \tag{4.33}$$

$$v_y(i+1) = v_y(i) - \frac{GM_s}{(r(i))^3} y(i)\Delta t. \tag{4.34}$$

$$y(i+1) = y(i) + v_y(i)\Delta t. \tag{4.35}$$

$$r(i) = \sqrt{x(i)^2 + y(i)^2}. \tag{4.36}$$

This is Euler algorithm. It can also be rewritten with $\hat{x}(i) = x(i-1)$, $\hat{y}(i) = y(i-1)$, $\hat{v}_x(i) = v_x(i-1)$, $\hat{v}_y(i) = v_y(i-1)$, $\hat{r}(i) = r(i-1)$ and $i = 1, ..., N+1$ as

$$\hat{v}_x(i+1) = \hat{v}_x(i) - \frac{GM_s}{(\hat{r}(i))^3} \hat{x}(i)\Delta t. \tag{4.37}$$

$$\hat{x}(i+1) = \hat{x}(i) + \hat{v}_x(i)\Delta t. \tag{4.38}$$

$$\hat{v}_y(i+1) = \hat{v}_y(i) - \frac{GM_s}{(\hat{r}(i))^3} \hat{y}(i)\Delta t. \tag{4.39}$$

$$\hat{y}(i+1) = \hat{y}(i) + \hat{v}_y(i)\Delta t. \tag{4.40}$$

$$\hat{r}(i) = \sqrt{\hat{x}(i)^2 + \hat{y}(i)^2}. \tag{4.41}$$

In order to maintain energy conservation we employ Euler-Cromer algorithm. We calculate as in the Euler's algorithm the velocity at time step $i+1$ by using the position and velocity at time step i. However we compute the position at time step $i+1$ by using the position at time step i and the velocity at time step $i+1$, viz

$$\hat{v}_x(i+1) = \hat{v}_x(i) - \frac{GM_s}{(\hat{r}(i))^3}\hat{x}(i)\Delta t. \tag{4.42}$$

$$\hat{x}(i+1) = \hat{x}(i) + \hat{v}_x(i+1)\Delta t. \tag{4.43}$$

$$\hat{v}_y(i+1) = \hat{v}_y(i) - \frac{GM_s}{(\hat{r}(i))^3}\hat{y}(i)\Delta t. \tag{4.44}$$

$$\hat{y}(i+1) = \hat{y}(i) + \hat{v}_y(i+1)\Delta t. \tag{4.45}$$

4.3 The Runge-Kutta Algorithm

4.3.1 The Method

The problem is still trying to solve the first order differential equation

$$\frac{dy}{dx} = f(x,y). \tag{4.46}$$

In the Euler's method we approximate the function $y = y(x)$ in each interval $[x_n, x_{n+1}]$ by the straight line

$$y_{n+1} = y_n + \Delta x f(x_n, y_n). \tag{4.47}$$

The slope $f(x_n, y_n)$ of this line is exactly given by the slope of the function $y = y(x)$ at the begining of the inetrval $[x_n, x_{n+1}]$.

Given the value y_n at x_n we evaluate the value y_{n+1} at x_{n+1} using the method of Runge-Kutta as follows. First the middle of the interval $[x_n, x_{n+1}]$ which is at the value $x_n + \frac{1}{2}\Delta x$ corresponds to the y-value y_{n+1} calculated using the Euler's method, viz $y_{n+1} = y_n + \frac{1}{2}k_1$ where

$$k_1 = \Delta x f(x_n, y_n). \tag{4.48}$$

Second the slope at this middle point $(x_n + \frac{1}{2}\Delta x, y_n + \frac{1}{2}k_1)$ which is given by

$$\frac{k_2}{\Delta x} = f(x_n + \frac{1}{2}\Delta x, y_n + \frac{1}{2}k_1) \tag{4.49}$$

is the value of the slope which will be used to estimate the correct value of y_{n+1} at x_{n+1} using again Euler's method, namely

$$y_{n+1} = y_n + k_2. \tag{4.50}$$

In summary the Runge-Kutta algorithm is given by

$$
\begin{aligned}
k_1 &= \Delta x f(x_n, y_n) \\
k_2 &= \Delta x f(x_n + \frac{1}{2}\Delta x, y_n + \frac{1}{2}k_1) \\
y_{n+1} &= y_n + k_2.
\end{aligned}
\tag{4.51}
$$

The error in this method is proportional to Δx^3. This can be shown as follows. We have

$$
\begin{aligned}
y(x + \Delta x) &= y(x) + \Delta x \frac{dy}{dx} + \frac{1}{2}(\Delta x)^2 \frac{d^2 y}{dx^2} + ... \\
&= y(x) + \Delta x f(x, y) + \frac{1}{2}(\Delta x)^2 \frac{d}{dx} f(x, y) + ... \\
&= y(x) + \Delta x \left(f(x, y) + \frac{1}{2}\Delta x \frac{\partial f}{\partial x} + \frac{1}{2}\Delta x f(x, y) \frac{\partial f}{\partial y} \right) + ... \\
&= y(x) + \Delta x f(x + \frac{1}{2}\Delta x, y + \frac{1}{2}\Delta x f(x, y)) + O(\Delta x^3) \\
&= y(x) + \Delta x f(x + \frac{1}{2}\Delta x, y + \frac{1}{2}k_1) + O(\Delta x^3) \\
&= y(x) + k_2 + O(\Delta x^3).
\end{aligned}
\tag{4.52}
$$

Let us finally note that the above Runge-Kutta method is strictly speaking the second-order Runge-Kutta method. The first-order Runge-Kutta method is the Euler algorithm. The higher-order Runge-Kutta methods will not be discussed here.

4.3.2   Example 1: The Harmonic Oscillator

Let us apply this method to the problem of the harmonic oscillator. We have the differential equations

$$
\begin{aligned}
\frac{d\theta}{dt} &= \omega \\
\frac{d\omega}{dt} &= -\frac{g}{l}\theta.
\end{aligned}
\tag{4.53}
$$

Euler's equations read

$$
\begin{aligned}
\theta_{n+1} &= \theta_n + \Delta t \omega_n \\
\omega_{n+1} &= \omega_n - \frac{g}{l}\theta_n \Delta t.
\end{aligned}
\tag{4.54}
$$

First we consider the function $\theta = \theta(t)$. The middle point is $(t_n + \frac{1}{2}\Delta t, \theta_n + \frac{1}{2}k_1)$ where $k_1 = \Delta t \omega_n$. For the function $\omega = \omega(t)$ the middle point is $(t_n + \frac{1}{2}\Delta t, \omega_n + \frac{1}{2}k_3)$ where $k_3 = -\frac{g}{l}\Delta t \theta_n$. Therefore we have

$$
\begin{aligned}
k_1 &= \Delta t \omega_n \\
k_3 &= -\frac{g}{l}\Delta t \theta_n.
\end{aligned}
\tag{4.55}
$$

The slope of the function $\theta(t)$ at its middle point is

$$\frac{k_2}{\Delta t} = \omega_n + \frac{1}{2} k_3. \tag{4.56}$$

The slope of the function $\omega(t)$ at its middle point is

$$\frac{k_4}{\Delta t} = -\frac{g}{l}(\theta_n + \frac{1}{2} k_1). \tag{4.57}$$

The Runge-Kutta solution is then given by

$$\theta_{n+1} = \theta_n + k_2$$
$$\omega_{n+1} = \omega_n + k_4. \tag{4.58}$$

4.3.3 Example 2: The Solar System

Let us consider the equations

$$\frac{dx}{dt} = v_x. \tag{4.59}$$

$$\frac{dv_x}{dt} = -\frac{GM_s}{r^3} x. \tag{4.60}$$

$$\frac{dy}{dt} = v_y. \tag{4.61}$$

$$\frac{dv_y}{dt} = -\frac{GM_s}{r^3} y. \tag{4.62}$$

First we consider the function $x = x(t)$. The middle point is $(t_n + \frac{1}{2}\Delta t, x_n + \frac{1}{2} k_1)$ where $k_1 = \Delta t \, v_{xn}$. For the function $v_x = v_x(t)$ the middle point is $(t_n + \frac{1}{2}\Delta t, v_{xn} + \frac{1}{2} k_3)$ where $k_3 = -\frac{GM_s}{r_n} \Delta t \, x_n$. Therefore we have

$$k_1 = \Delta t \, v_{xn}$$
$$k_3 = -\frac{GM_s}{r_n^3} \Delta t \, x_n. \tag{4.63}$$

The slope of the function $x(t)$ at the middle point is

$$\frac{k_2}{\Delta t} = v_{xn} + \frac{1}{2} k_3. \tag{4.64}$$

The slope of the function $v_x(t)$ at the middle point is

$$\frac{k_4}{\Delta t} = -\frac{GM_s}{R_n^3}(x_n + \frac{1}{2} k_1). \tag{4.65}$$

Next we consider the function $y = y(t)$. The middle point is $(t_n + \frac{1}{2}\Delta t, y_n + \frac{1}{2}k_1')$ where $k_1' = \Delta t\, v_{yn}$. For the function $v_y = v_y(t)$ the middle point is $(t_n + \frac{1}{2}\Delta t, v_{yn} + \frac{1}{2}k_3')$ where $k_3' = -\frac{GM_s}{r_n}\Delta t\, y_n$. Therefore we have

$$k_1' = \Delta t\, v_{yn}$$
$$k_3' = -\frac{GM_s}{r_n^3}\Delta t\, y_n. \tag{4.66}$$

The slope of the function $y(t)$ at the middle point is

$$\frac{k_2'}{\Delta t} = v_{yn} + \frac{1}{2}k_3'. \tag{4.67}$$

The slope of the function $v_y(t)$ at the middle point is

$$\frac{k_4'}{\Delta t} = -\frac{GM_s}{R_n^3}(y_n + \frac{1}{2}k_1'). \tag{4.68}$$

In the above equations

$$R_n = \sqrt{(x_n + \frac{1}{2}k_1)^2 + (y_n + \frac{1}{2}k_1')^2}. \tag{4.69}$$

The Runge-Kutta solutions are then given by

$$x_{n+1} = x_n + k_2$$
$$v_{x(n+1)} = v_{xn} + k_4$$
$$y_{n+1} = y_n + k_2'$$
$$v_{y(n+1)} = v_{yn} + k_4'. \tag{4.70}$$

4.4 Precession of the Perihelion of Mercury

The orbit of Mercury is elliptic. The orientation of the axes of the ellipse rotate with time. This is the precession of the perihelion (the point of the orbit nearest to the Sun) of Mercury. Mercury's perihelion makes one revolution every 23000 years. This is approximately 566 arcseconds per century. The gravitational forces of the other planets (in particular Jupiter) lead to a precession of 523 arcseconds per century. The remaining 43 arcseconds per century are accounted for by general relativity.

For objects too close together (like the Sun and Mercury) the force of gravity predicted by general relativity deviates from the inverse-square law. This force is given by

$$F = \frac{GM_sM_m}{r^2}(1 + \frac{\alpha}{r^2}) \; , \; \alpha = 1.1 \times 10^{-8}\text{AU}^2. \tag{4.71}$$

We discuss here some of the numerical results obtained with the Runge-Kutta method for different values of α. We take the time step and the number of iterations to be $N = 20000$ and $dt = 0.0001$. The angle of the line joining the Sun and Mercury with the horizontal

axis when mercury is at the perihelion is found to change linearly with time. We get the following rates of precession

$$\alpha = 0.0008 \ , \ \frac{d\theta}{dt} = 8.414 \pm 0.019$$

$$\alpha = 0.001 \ , \ \frac{d\theta}{dt} = 10.585 \pm 0.018$$

$$\alpha = 0.002 \ , \ \frac{d\theta}{dt} = 21.658 \pm 0.019$$

$$\alpha = 0.004 \ , \ \frac{d\theta}{dt} = 45.369 \pm 0.017. \tag{4.72}$$

Thus

$$\frac{d\theta}{dt} = a\alpha \ , \ \alpha = 11209.2 \pm 147.2 \text{ degrees}/(\text{yr}.\alpha). \tag{4.73}$$

By extrapolating to the value provided by general relativity, viz $\alpha = 1.1 \times 10^{-8}$ we get

$$\frac{d\theta}{dt} = 44.4 \pm 0.6 \text{ arcsec/century.} \tag{4.74}$$

4.5   Exercises

Exercise 1:   Using the Runge-Kutta method solve the following differential equations

$$\frac{d^2r}{dt^2} = \frac{l^2}{r^3} - \frac{GM}{r^2}. \tag{4.75}$$

$$\frac{d^2z}{dt^2} = -g. \tag{4.76}$$

$$\frac{dN}{dt} = aN - bN^2. \tag{4.77}$$

Exercise 2:   The Lorenz model is a chaotic system given by three coupled first order differential equations

$$\frac{dx}{dt} = \sigma(y - x)$$

$$\frac{dy}{dt} = -xz + rx - y$$

$$\frac{dz}{dt} = xy - bz. \tag{4.78}$$

This system is a simplified version of the system of Navier-Stokes equations of fluid mechanics which are relevant for the Rayleigh-Bénard problem. Write down the numercial solution of these equations according to the Runge-Kutta method.

4.6   Simulation 6: Runge-Kutta Algorithm- The Solar System

Part I   We consider a solar system consisting of a single planet moving around the Sun. We suppose that the Sun is very heavy compared to the planet that we can safely assume that it is not moving at the center of the system. Newton's second law gives the following equations of motion

$$v_x = \frac{dx}{dt} \ , \ \frac{dv_x}{dt} = -\frac{GM_s}{r^3}x \ , \ v_y = \frac{dy}{dt} \ , \ \frac{dv_y}{dt} = -\frac{GM_s}{r^3}y.$$

We will use here the astronomical units defined by $GM_s = 4\pi^2 \text{AU}^3/\text{yr}^2$.

(1) Write a Fortran code in which we implement the Runge-Kutta algorithm for the problem of solving the equations of motion of the the solar system.

(2) Compute the trajectory, the velocity and the energy as functions of time. What do you observe for the energy.

(3) According to Kepler's first law the orbit of any planet is an ellipse with the Sun at one of the two foci. In the following we will only consider planets which are known to have circular orbits to a great accuracy. These planets are Venus, Earth, Mars, Jupiter and Saturn. The radii in astronomical units are given by

$$a_{\text{venus}} = 0.72 \ , \ a_{\text{earth}} = 1 \ , \ a_{\text{mars}} = 1.52 \ , \ a_{\text{jupiter}} = 5.2 \ , \ a_{\text{saturn}} = 9.54.$$

Verify that Kepler's first law indeed holds for these planets.

In order to answer questions 2 and 3 above we take the initial conditions

$$x(1) = a \ , \ y(1) = 0 \ , \ v_x(1) = 0 \ , \ v_y(1) = v.$$

The value chosen for the initial velocity is very important to get a correct orbit and must be determined for example by assuming that the orbit is indeed circular and as a consequence the centrifugal force is balanced by the force of gravitational attraction. We get $v = \sqrt{GM_s/a}$.

We take the step and the number of iterations $\Delta t = 0.01$ yr , $N = 10^3 - 10^4$.

Part II

(1) According to Kepler's third law the square of the period of a planet is directly proportional to the cube of the semi-major axis of its orbit. For circular orbits the proportionality factor is equal 1 exactly. Verify this fact for the planets mentioned above. We can measure the period of a planet by monitoring when the planet returns to its farthest point from the sun.

(2) By changing the initial velocity appropriately we can obtain an elliptical orbit. Check this thing.

(3) The fundamental laws governing the motion of the solar system are Newton's law of universal attraction and Newton's second law of motion. Newton's law of universal attraction states that the force between the Sun and a planet is inversely proportioanl to the square of the distance between them and it is directed from the planet to the Sun. We will assume in the following that this force is inversely proportional to a different power of the distance. Modify the code accordingly and calculate the new orbits for powers between 1 and 3. What do you observe and what do you conclude.

4.7   Simulation 7: Precession of the perihelion of Mercury

According to Kepler's first law the orbits of all planets are ellipses with the Sun at one of the two foci. This law can be obtained from applying Newton's second law to the system consisting of the Sun and a single planet. The effect of the other planets on the motion will lead to a change of orientation of the orbital ellipse within the orbital plane of the planet. Thus the point of closest approach (the perihelion) will precess, i.e. rotate around the sun. All planets suffer from this effect but because they are all farther from the sun and all have longer periods than Mercury the amount of precession observed for them is smaller than that of Mercury.

However it was established earlier on that the precession of the perihelion of Mercury due to Newtonian effects deviates from the observed precession by the amount 43 arcsecond/century. As it turns out this can only be explained within general relativity. The large mass of the Sun causes space and time around it to be curved which is felt the most by Mercury because of its proximity. This spacetime curvature can be approximated by the force law

$$F = \frac{GM_s M_m}{r^2}(1 + \frac{\alpha}{r^2}) \ , \ \alpha = 1.1.10^{-8} AU^2.$$

(1) Include the above force in the code. The initial position and velocity of Mercury are

$$x_0 = (1 + e)a \ , \ y_0 = 0.$$

$$v_{x0} = 0 \ , \ v_{y0} = \sqrt{\frac{GM_s}{a} \frac{1-e}{1+e}}.$$

Thus initially Mercury is at its farthest point from the Sun since a is the semi-major axis of Mercury ($a = 0.39$ AU) and e is its eccentricity ($e = 0.206$) and hence ea is the distance between the Sun and the center of the ellipse. The semi-minor axis is defined by $b = a\sqrt{1-e^2}$. The initial velocity was calculated from applying the principles of conservation of angular momentum and conservation of energy between the above initial point and the point $(0, b)$.

(2) The amount of precession of the perihelion of Mercury is very small because α is very small. In fact it can not be measured directly in any numerical simulation with a limited amount of time. Therefore we will choose a larger value of α for example

$\alpha = 0.0008$ AU^2. We also work with $N = 20000$, $dt = 0.0001$. Compute the orbit for these values. Compute the angle θ made between the vector position of Mercury and the horizontal axis as a function of time. Compute also the distance between Mercury and the sun and its derivative with respect to time given by

$$\frac{dr}{dt} = \frac{xv_x + yv_y}{r}.$$

This derivative will vanish each time Mercury reaches its farthest point from the sun or its closest point from the sun (the perihelion). Plot the angle θ_p made between the vector position of Mercury at its farthest point and the horizontal axis as a function of time. What do you observe. Determine the slope $d\theta_p/dt$ which is precisely the amount of precession of the perihelion of Mercury for the above value of α.

(3) Repeat the above question for other values of α say $\alpha = 0.001, 0.002, 0.004$. Each time compute $d\theta_p/dt$. Plot $d\theta_p/dt$ as a function of α. Determine the slope. Deduce the amount of precession of the perihelion of Mercury for the value of $\alpha = 1.1.10^{-8}$AU^2.

Chapter 5

Chaotic Pendulum

5.1 Equation of Motion

We start from a simple pendulum. The equation of motion is given by

$$ml\frac{d^2\theta}{dt^2} = -mg\sin\theta. \tag{5.1}$$

We consider the effect of air resistance on the motion of the mass m. We will assume that the force of air resistance is given by Stokes' law. We get

$$ml\frac{d^2\theta}{dt^2} = -mg\sin\theta - mlq\frac{d\theta}{dt}. \tag{5.2}$$

The air friction will drain all energy from the pendulum. In order to maintain the motion against the damping effect of air resistance we will add a driving force. We will choose a periodic force with amplitude mlF_D and frequency ω_D. This arise for example if we apply a periodic electric field with amplitude E_D and frequency ω_D on the mass m which is assumed to have an electric charge q, i.e $mlF_D = qE_D$. It can also arise from the periodic oscillations of the pendulum's pivot point. By adding the driving force we get then the equation of motion

$$ml\frac{d^2\theta}{dt^2} = -mg\sin\theta - mlq\frac{d\theta}{dt} + mlF_D\cos\omega_D t. \tag{5.3}$$

The natural frequency of the oscillations is given by the frequency of the simple pendulum, viz

$$\omega_0 = \sqrt{\frac{g}{l}}. \tag{5.4}$$

We will always take $\omega_0 = 1$, i.e. $l = g$. The equation of motion becomes

$$\frac{d^2\theta}{dt^2} + \frac{1}{Q}\frac{d\theta}{dt} + \sin\theta = F_D\cos\omega_D t. \tag{5.5}$$

The coefficient $Q = 1/q$ is known as the quality factor. It measures how many oscillations the pendulum without driving force will make before its energy is drained. We will

write the above second order differential equation as two first order differential equations, namely

$$\frac{d\theta}{dt} = \Omega$$

$$\frac{d\Omega}{dt} = -\frac{1}{Q}\Omega - \sin\theta + F_D\cos\omega_D t. \tag{5.6}$$

This system of differential equations does not admit a simple analytic solution. The linear approximation corresponds to small amplitude oscillations, viz

$$\sin\theta \simeq \theta. \tag{5.7}$$

The differential equations become linear given by

$$\frac{d\theta}{dt} = \Omega$$

$$\frac{d\Omega}{dt} = -\frac{1}{Q}\Omega - \theta + F_D\cos\omega_D t. \tag{5.8}$$

Or equivalently

$$\frac{d\theta^2}{dt^2} = -\frac{1}{Q}\frac{d\theta}{dt} - \theta + F_D\cos\omega_D t. \tag{5.9}$$

For $F_D = 0$ the solution is given by

$$\theta_{t0} = \left(\theta(0)\cos\omega_* t + \frac{1}{\omega_*}\left(\Omega(0) + \frac{\theta(0)}{2Q}\right)\sin\omega_* t\right)e^{-\frac{t}{2Q}} \;,\; \omega_* = \sqrt{1 - \frac{1}{4Q^2}}. \tag{5.10}$$

For $F_D \neq 0$ a particular solution is given by

$$\theta_\infty = F_D(a\cos\omega_D t + b\sin\omega_D t). \tag{5.11}$$

We find

$$a = \frac{1}{(1 - \omega_D^2)^2 + \frac{\omega_D^2}{Q^2}}(1 - \omega_D^2) \;,\; b = \frac{1}{(1 - \omega_D^2)^2 + \frac{\omega_D^2}{Q^2}}\frac{\omega_D}{Q}. \tag{5.12}$$

For $F_D \neq 0$ the general solution is given by

$$\theta = \theta_\infty + \theta_t. \tag{5.13}$$

$$\theta_t = \left[\left(\theta(0) - \frac{F_D(1 - \omega_D^2)}{(1 - \omega_D^2)^2 + \frac{\omega_D^2}{Q^2}}\right)\cos\omega_* t + \frac{1}{\omega_*}\left(\Omega(0) + \frac{\theta(0)}{2Q} - \frac{1}{2Q}\frac{F_D(1 - 3\omega_D^2)}{(1 - \omega_D^2)^2 + \frac{\omega_D^2}{Q^2}}\right)\sin\omega_* t\right]e^{-\frac{t}{2Q}}.$$

$$\tag{5.14}$$

The last two terms depend on the initial conditions and will vanish exponentially at very large times $t \longrightarrow \infty$, i.e. they are transients. The asymptotic motion is given by θ_∞. Thus for $t \longrightarrow \infty$ we get

$$\theta = \theta_\infty = F_D(a\cos\omega_D t + b\sin\omega_D t). \tag{5.15}$$

Also for $t \longrightarrow \infty$ we get

$$\Omega = \frac{d\theta}{dt} = F_D \omega_D (-a \sin \omega_D t + b \cos \omega_D t). \tag{5.16}$$

We compute in the limit of large times $t \longrightarrow \infty$

$$\theta^2 + \frac{\Omega^2}{\omega_D^2} = \tilde{F}_D^2 = F_D^2(a^2 + b^2) = \frac{F_D^2}{(1 - \omega_D^2)^2 + \frac{\omega_D^2}{Q^2}}. \tag{5.17}$$

In other words the orbit of the system in phase space is an ellipse. The motion is periodic with period equal to the period of the driving force. This ellipse is also called a periodic attractor because regardless of the initial conditions the trajectory of the system will tend at large times to this ellipse.

Let us also remark that the maximum angular displacement is $\tilde{F}_D$. The function $\tilde{F}_D = \tilde{F}_D(\omega_D)$ exhibits resonant behavior as the driving frequency approaches the natural frequency which is equivalent to the limit $\omega_D \longrightarrow 1$. In this limit $\tilde{F}_D = QF_D$. The width of the resonant window is proportional to $1/Q$ so for $Q \longrightarrow \infty$ we observe that $\tilde{F}_D \longrightarrow \infty$ when $\omega_D \longrightarrow 1$ while for $Q \longrightarrow 0$ we observe that $\tilde{F}_D \longrightarrow 0$ when $\omega_D \longrightarrow 1$.

In general the time-asymptotic response of any linear system to a periodic drive is periodic with the same period as the driving force. Furthermore when the driving frequency approaches one of the natural frequencies the response will exhibits resonant behavior.

The basic ingredient in deriving the above results is the linearity of the dynamical system. As we will see shortly periodic motion is not the only possible time-asymptotic response of a dynamical system to a periodic driving force.

5.2 Numerical Algorithms

The equations of motion are

$$\frac{d\theta}{dt} = \Omega$$
$$\frac{d\Omega}{dt} = -\frac{1}{Q}\Omega - \sin\theta + F(t). \tag{5.18}$$

The external force is periodic and it will be given by one of the following expressions

$$F(t) = F_D \cos \omega_D t. \tag{5.19}$$

$$F(t) = F_D \sin \omega_D t. \tag{5.20}$$

5.2.1 Euler-Cromer Algorithm

Numerically we can employ the Euler-Cromer algorithm in order to solve this system of differential equations. The solution goes as follows. First we choose the initial conditions.

For example

$$\Omega(1) = 0$$
$$\theta(1) = 0$$
$$t(1) = 0. \tag{5.21}$$

For $i = 1, ..., N + 1$ we use

$$\Omega(i + 1) = \Omega(i) + \Delta t \left(-\frac{1}{Q}\Omega(i) - \sin\theta(i) + F(i) \right)$$
$$\theta(i + 1) = \theta(i) + \Delta t\ \Omega(i + 1)$$
$$t(i + 1) = \Delta t\ i. \tag{5.22}$$

$$F(i) \equiv F(t(i)) = F_D \cos\omega_D \Delta t(i - 1). \tag{5.23}$$

$$F(i) \equiv F(t(i)) = F_D \sin\omega_D \Delta t(i - 1). \tag{5.24}$$

5.2.2   Runge-Kutta Algorithm

In order to achieve better precision we employ the Runge-Kutta algorithm. For $i = 1, ..., N + 1$ we use

$$k_1 = \Delta t\ \Omega(i)$$
$$k_3 = \Delta t \left[-\frac{1}{Q}\Omega(i) - \sin\theta(i) + F(i) \right]$$
$$k_2 = \Delta t \left(\Omega(i) + \frac{1}{2}k_3 \right)$$
$$k_4 = \Delta t \left[-\frac{1}{Q} \left(\Omega(i) + \frac{1}{2}k_3 \right) - \sin\left(\theta(i) + \frac{1}{2}k_1 \right) + F(i + \frac{1}{2}) \right]$$
$$\tag{5.25}$$

$$\theta(i + 1) = \theta(i) + k_2$$
$$\Omega(i + 1) = \Omega(i) + k_4$$
$$t(i + 1) = \Delta t\ i. \tag{5.26}$$

$$F(i) \equiv F(t(i)) = F_D \cos\omega_D \Delta t(i - 1). \tag{5.27}$$

$$F(i) \equiv F(t(i)) = F_D \sin\omega_D \Delta t(i - 1). \tag{5.28}$$

$$F(i + \frac{1}{2}) \equiv F(t(i) + \frac{1}{2}\Delta t) = F_D \cos\omega_D \Delta t(i - \frac{1}{2}). \tag{5.29}$$

$$F(i + \frac{1}{2}) \equiv F(t(i) + \frac{1}{2}\Delta t) = F_D \sin\omega_D \Delta t(i - \frac{1}{2}). \tag{5.30}$$

5.3 Elements of Chaos

5.3.1 Butterfly Effect: Sensitivity to Initial Conditions

The solution in the linear regime (small amplitude) reads

$$\theta = \theta_\infty + \theta_t. \tag{5.31}$$

The transient is of the form

$$\theta_t = f(\theta(0), \Omega(0))e^{-t/2Q}. \tag{5.32}$$

This goes to zero at large times t. The time-asymptotic is thus given by

$$\theta_\infty = F_D(a\cos\omega_D t + b\sin\omega_D t). \tag{5.33}$$

The motion in the phase space is periodic with period equal to the period of the driving force. The orbit in phase space is precisley an ellipse of the form

$$\theta_\infty^2 + \frac{\Omega_\infty^2}{\omega_D^2} = F_D^2(a^2 + b^2). \tag{5.34}$$

Let us consider a perturbation of the initial conditions. We can imagine that we have two pendulums A and B with slightly different initial conditions. Then the difference between the two trajectories is

$$\delta\theta = \delta f(\theta(0), \Omega(0))e^{-t/2Q}. \tag{5.35}$$

This goes to zero at large times. If we plot $\ln \delta\theta$ as a function of time we find a straight line with a negative slope. The time-asymptotic motion is not sensitive to initial conditions. It converges at large times to θ_∞ no matter what the initial conditions are. The curve $\theta_\infty = \theta_\infty(\Omega_\infty)$ is called a (periodic) attractor. This is because any perturbed trajectory will decay exponentially in time to the attractor.

In order to see chaotic behavior we can for example increase Q keeping everything else fixed. We observe that the slope of the line $\ln \delta\theta = \lambda t$ starts to decrease until at some value of Q it becomes positive. At this value the variation between the two pendulums increases exponentially with time. This is the chaotic regime. The value $\lambda = 0$ is the value where chaos happens. The coefficient λ is called Lyapunov exponent.

The chaotic pendulum is a deterministic system (since it obeys ordinary differential equations) but it is not predictable in the sense that given two identical pendulums their motions will diverge from each other in the chaotic regime if there is the slightest error in determining their initial conditions. This high sensitivity to initial conditions is known as the butterfly effect and could be taken as the definition of chaos itself.

However we should stress here that the motion of the chaotic pendulum is not random. This can be seen by inspecting Poincare sections.

5.3.2   Poincare Section and Attractors

The periodic motion of the linear system with period equal to the period of the driving force is called a period-1 motion. In this motion the trajectory repeats itself exactly every one single period of the external driving force. This is the only possible motion in the low amplitude limit.

Generally a period-$\mathcal{N}$ motion corresponds to an orbit of the dynamical system which repeats itself every $\mathcal{N}$ periods of the external driving force. These orbits exist in the non-linear regime of the pendulum.

The Poincare section is defined as follows. We plot in the θ-Ω phase space only one point per period of the external driving force. We plot for example (θ, Ω) for

$$\omega_D t = \phi + 2\pi n. \tag{5.36}$$

The angle ϕ is called the Poincare phase and n is an integer. For period-1 motion the Poincare section consists of one single point. For period-$\mathcal{N}$ motion the Poincare section consists of $\mathcal{N}$ points.

Thus in the linear regime if we plot (θ, Ω) for $\omega_D t = 2\pi n$ we get a single point since the motion is periodic with period equal to that of the driving force. The single point we get as a Poincare section is also an attractor since all pendulums with almost the same initial conditions will converge onto it.

In the chaotic regime the Poincare section is an attractor known as strange attractor. It is a complicated curve which could have fractal structure and all pendulums with almost the same initial conditions will converge onto it.

5.3.3   Period-Doubling Bifurcations

In the case of the chaotic pendulum we encounter between the linear regime and the emergence of chaos the so-called period doubling phenomena. In the linear regime the Poincare section is a point P which corresponds to a period-1 motion with period equal $T_D = 2\pi/\omega_D$. The θ or Ω coordinate of this point P will trace a line as we increase Q while keeping everything fixed. We will eventually reach a value Q_1 of Q where this line bifurcates into two lines. By close inspection we see that at Q_1 the motion becomes period-2 motion, i.e. the period becomes equal to $2T_D$.

In a motion where the period is T_D (below Q_1) we get the same value of θ each time $t = mT_D$ and since we are plotting θ each time $t = 2n\pi/\omega_D = nT_D$ we will get a single point in the Poincare section. In a motion where the period is $2T_D$ (at Q_2) we get the same value of θ each time $t = 2mT_D$, i.e. the value of θ at times $t = mT_D$ is different and hence we get two points in the Poincare section.

As we increase Q the motion becomes periodic with period equal $4T_D$, then with period equal $8T_D$ and so on. The motion with period $2^{\mathcal{N}} T_D$ is called period-$\mathcal{N}$ motion. The corresponding Poincare section consists of $\mathcal{N}$ distinct points.

The diagram of θ as a function of Q is called a bifurcation diagram. It has a fractal structure. Let us point out here that normally in ordinary oscillations we get harmonics with periods equal to the period of the driving force divided by $2^{\mathcal{N}}$. In this case we

obtained in some sense subharmonics with periods equal to the period of the driving force times $2^{\mathcal{N}}$. This is very characteristic of chaos. In fact chaotic behavior corresponds to the limit $\mathcal{N} \longrightarrow \infty$. In other words chaos is period-∞ (bounded) motion which could be taken as another definition of chaos.

5.3.4   Feigenbaum Ratio

Let $Q_{\mathcal{N}}$ be the critical value of Q above which the $\mathcal{N}$th bifurcation is triggered. In other words $Q_{\mathcal{N}}$ is the value where the transition to period-$\mathcal{N}$ motion happens. We define the Feigenbaum ratio by

$$F_{\mathcal{N}} = \frac{Q_{\mathcal{N}-1} - Q_{\mathcal{N}-2}}{Q_{\mathcal{N}} - Q_{\mathcal{N}-1}}. \tag{5.37}$$

It is shown that $F_{\mathcal{N}} \longrightarrow F = 4.669$ as $\mathcal{N} \longrightarrow \infty$. This is a universal ratio called the Feigenbaum ratio and it characterizes many chaotic systems which suffer a transition to chaos via an infinite series of period-doubling bifurcations. The above equation can be then rewritten as

$$Q_{\mathcal{N}} = Q_1 + (Q_2 - Q_1) \sum_{j=0}^{\mathcal{N}-2} \frac{1}{F^j} \tag{5.38}$$

Let us define the accumulation point by Q_{∞} then

$$Q_{\infty} = Q_1 + (Q_2 - Q_1) \frac{F}{F - 1} \tag{5.39}$$

This is where chaos occur. In the bifurcation diagram the chaotic region is a solid black region.

5.3.5   Spontaneous Symmetry Breaking

The bifurcation process is associated with a deep phenomenon known as spontaneous symmetry breaking. The first period-doubling bifurcation corresponds to the breaking of the symmetry $t \longrightarrow t + T_D$. The linear regime respects this symmetry. However period-2 motion and in general period-$\mathcal{N}$ motions with $\mathcal{N} > 2$ do not respect this symmetry.

There is another kind of spontaneous symmetry breaking which occurs in the chaotic pendulum and which is associated with a bifurcation diagram. This happens in the region of period-1 motion and it is the breaking of spatial symmetry or parity $\theta \longrightarrow -\theta$. Indeed there exists solutions of the equations of motion that are either left-favoring or right-favoring. In other words the pendulums in such solutions spend much of its time in the regions to the left of the pendulum's vertical ($\theta < 0$) or to the right of the pendulum's vertical ($\theta > 0$). This breaking of left-right symmetry can be achieved by a gradual increase of Q. We will then reach either the left-favoring solution or the right-favoring solution starting from a left-right symmetric solution depending on the initial conditions. The symmetry $\theta \longrightarrow -\theta$ is also spontaneously broken in period-$\mathcal{N}$ motions.

5.4   Simulation 8: The Butterfly Effect

We consider a pendulum of a mass m and a length l moving under the influence of the force of gravity, the force of air resistance and a driving periodic force. Newton's second law of motion reads

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin\theta - q\frac{d\theta}{dt} + F_D\sin 2\pi\nu_D t.$$

We will always take the angular frequency $\sqrt{g/l}$ associated with simple oscillations of the pendulum equal 1, i.e. $l = g$. The numerical solution we will consider here is based on the Euler-Cromer algorithm.

The most important property of a large class of solutions of this differential equation is hyper sensitivity to initial conditions known also as the butterfly effect which is the defining characteristic of chaos. For this reason the driven non-linear pendulum is also known as the chaotic pendulum.

The chaotic pendulum can have two distinct behaviors. In the linear regime the motion (neglecting the initial transients) is periodic with a period equal to the period of the external driving force. In the chaotic regime the motion never repeats and any error even infinitesimal in determining the initial conditions will lead to a completely different orbit in the phase space.

(1) Write a code which implements the Euler-Cromer algorithm for the chaotic pendulum. The angle θ must always be taken between $-\pi$ and π which can be maintained as follows

$$\text{if}(\theta_i.\text{lt.} \mp \pi)\ \theta_i = \theta_i \pm 2\pi.$$

(2) We take the values and initial conditions

$$dt = 0.04s \ , \ \ 2\pi\nu_D = \frac{2}{3}s^{-1} \ , \ \ q = \frac{1}{2}s^{-1} \ , \ \ N = 1000 - 2000.$$

$$\theta_1 = 0.2 \text{ radian} \ , \ \ \Omega_1 = 0 \text{ radian}/s.$$

$$F_D = 0 \text{ radian}/s^2 \ , \ \ F_D = 0.1 \text{ radian}/s^2 \ , \ \ F_D = 1.2 \text{ radian}/s^2.$$

Plot θ as a function of time. What do you observe for the first value of F_D. What is the period of oscillation for small and large times for the second value of F_D . Is the motion periodic for the third value of F_D.

5.5   Simulation 9: Poincaré Sections

In the chaotic regime the motion of the pendulum although deterministic is not predictable. This however does not mean that the motion of the pendulum is random which can clearly be seen from the Poincare sections.

A Poincare section is a curve in the phase space obtained by plotting one point of the orbit per period of the external drive. Explicitly we plot points (θ, Ω) which corresponds to times $t = n/\nu_D$ where n is an integer. In the linear regime of the pendulum the Poincare section consists of a single point. Poincare section in the chaotic regime is a curve which

does not depend on the initial conditions thus confirming that the motion is not random and which may have a fractal structure. As a consequence this curve is called a strange attractor.

(1) We consider two identical chaotic pendulums A and B with slightly different initial conditions. For example we take

$$\theta_1^A = 0.2 \text{ radian} , \ \theta_1^B = 0.201 \text{ radian}.$$

The difference between the two motions can be measured by

$$\Delta\theta_i = \theta_i^A - \theta_i^B.$$

Compute $\ln \Delta\theta$ as a function of time for

$$F_D = 0.1 \text{ radian}/s^2 , \ F_D = 1.2 \text{ radian}/s^2.$$

What do you observe. Is the two motions identical. What happens for large times. Is the motion of the pendulum predictable. For the second value of F_D use

$$N = 10000 , \ dt = 0.01s.$$

(2) Compute the angular velocity Ω as a function of θ for

$$F_D = 0.5 \text{ radian}/s^2 , \ F_D = 1.2 \text{ radian}/s^2.$$

What is the orbit in the phase space for small times and what does it represent. What is the orbit for large times. Compare between the two pendulums A and B. Does the orbit for large times depend on the initial conditions.

(3) A Poincare section is obtained numerically by plotting the points (θ, Ω) of the orbit at the times at which the function $\sin \pi\nu_D t$ vanishes. These are the times at which this function changes sign. This is implemented as follows

$$\text{if}(\sin \pi\nu_D t_i \sin \pi\nu_D t_{i+1}.\text{lt}.0)\text{then}$$

$$\text{write}(*, *)t_i, \theta_i, \Omega_i.$$

Verify that Poincare section in the linear regime is given by a single point in the phase space. Take and use $F_D = 0.5 \text{ radian}/s^2$, $N = 10^4 - 10^7$, $dt = 0.001s$. Verify that Poincare section in the chaotic regime is also an attractor. Take and use $F_D = 1.2 \text{ radian}/s^2$, $N = 10^5$, $dt = 0.04s$. Compare between Poincare sections of the pendulums A and B. What do you observe and what do you conclude.

5.6   Simulation 10: Period Doubling

Among the most important chaotic properties of the driven non-linear pendulum is the phenomena of period doubling. The periodic orbit with period equal to the period of the external driving force are called period-1 motion. There exist however other periodic orbits with periods equal twice, four times and in general $2^{\mathcal{N}}$ times the period of the external driving force. The orbit with period equal $2^{\mathcal{N}}$ times the period of the external driving force is called period-$\mathcal{N}$ motion. The period doubling observed in the driven non-linear pendulum is a new phenomena which belongs to the world of chaos. In the standard phenomena of mixing the response of a non-linear system to a single frequency external driving force will contain components with periods equal to the period of the driving force divided by $2^{\mathcal{N}}$. In other words we get "harmonics" as opposed to the "subharmonics" we observe in the chaotic pendulum.

For period-$\mathcal{N}$ motion we expect that there are $\mathcal{N}$ different values of the angle θ for every value of F_D. The function $\theta = \theta(F_D)$ is called a bifurcation diagram. Formally the transition to chaos occurs at $\mathcal{N} \longrightarrow \infty$. In other words chaos is defined as period-infinity motion.

(1) We take the values and initial conditions

$$l = g \ , \ 2\pi\nu_D = \frac{2}{3}s^{-1} \ , \ q = \frac{1}{2}s^{-1} \ , \ N = 3000 - 100000 \ , \ dt = 0.01s.$$

$$\theta_1 = 0.2 \text{ radian} \ , \ \Omega_1 = 0 \text{ radian}/s.$$

Determine the period of the motion for

$$F_D = 1.35 \text{ radian}/s^2 \ , \ F_D = 1.44 \text{ radian}/s^2 \ , \ F_D = 1.465 \text{ radian}/s^2.$$

What happens to the period when we increase F_D. Does the two second values of F_D lie in the linear or chaotic regime of the chaotic pendulum.

(2) Compute the angle θ as a function of F_D for the times t which satisfy the condition $2\pi\nu_D t = 2n\pi$. We take F_D in the interval

$$F_D = (1.34 + 0.005k) \text{ radian}/s^2 \ , \ k = 1, ..., 30.$$

Determine the interval of the external driving force in which the orbits are period-1, period-2 and period-4 motions.

In this problem it is very important to remove the initial transients before we start measuring the bifurcation diagram. This can be done as follows. We calculate the motion for $2N$ steps but then only consider the last N steps in the computation of the Poincare section for every value of F_D.

5.7   Simulation 11: Bifurcation Diagrams

Part I   The chaotic pendulum is given by the equation

$$\frac{d^2\theta}{dt^2} = -\sin\theta - \frac{1}{Q}\frac{d\theta}{dt} + F_D \cos 2\pi\nu_D t.$$

In this simulation we take the values $F_D = 1.5$ radian/s^2 and $2\pi\nu_D = \frac{2}{3}s^{-1}$. In order to achieve a better numerical precision we use the second-order Runge-Kutta algorithm.

In the linear regime the orbits are periodic with period equal to the period T_D of the external driving force and are symmetric under $\theta \longrightarrow -\theta$. There exists other solutions which are periodic with period equal T_D but are not symmetric under $\theta \longrightarrow -\theta$. In these solutions the pendulum spends the majority of its time in the region to the left of its vertical ($\theta < 0$) or in the region to the right of its vertical ($\theta > 0$).

These symmetry breaking solutions can be described by a bifurcation diagram $\Omega = \Omega(Q)$. For every value of the quality factor Q we calculate the Poincare section. We observe that the Poincare section will bifurcate at some value Q_* of Q. Below this value we get one line whereas above this value we get two lines corresponding to the two symmetry breaking solutions in which the pendulum spends the majority of its time in the regions ($\theta > 0$) and ($\theta < 0$).

(1) Rewrite the code for the chaotic pendulum using Runge-Kutta algorithm.

(2) We take two different sets of initial conditions

$$\theta = 0.0 \text{ radian} , \ \Omega = 0.0 \text{ radian}/s.$$

$$\theta = 0.0 \text{ radian} , \ \Omega = -3.0 \text{ radian}/s .$$

Study the nature of the orbit for the values $Q = 0.5$s, $Q = 1.24$s and $Q = 1.3$s. What do you observe.

(3) Plot the bifurcation diagram $\Omega = \Omega(Q)$ for values of Q in the interval $[1.2, 1.3]$. What is the value Q_* at which the symmetry $\theta \longrightarrow -\theta$ is spontaneously broken.

Part II   As we have seen in the previous simulation period doubling can also be described by a bifurcation diagram. This phenomena is also an example of a spontaneous symmetry breaking. In this case the symmetry is $t \longrightarrow t + T_D$. Clearly only orbits with period T_D are symmetric under this transformation.

Let $Q_\mathcal{N}$ be the value of Q at which the $\mathcal{N}$th bifurcation occurs. In other words this is the value at which the orbit goes from being a period-$(\mathcal{N} - 1)$ motion to a period-$\mathcal{N}$ motion. The Feigenbaum ratio is defined by

$$F_\mathcal{N} = \frac{Q_{\mathcal{N}-1} - Q_{\mathcal{N}-2}}{Q_\mathcal{N} - Q_{\mathcal{N}-1}}.$$

As we approach the chaotic regime, i.e. as $\mathcal{N} \longrightarrow \infty$ the ratio $F_\mathcal{N}$ converges rapidly to the constant value $F = 4.669$. This is a general result which holds for many chaotic systems. Any dynamical system which can exhibit a transition to chaos via an infinite series of period-doubling bifurcations is characterized by a Feigenbaum ratio which approaches 4.669 as $\mathcal{N} \longrightarrow \infty$.

(1) Calculate the orbit and Poincare section for $Q = 1.36$s. What is the period of the motion. Is the orbit symmetric under $t \longrightarrow t + T_D$. Is the orbit symmetric under $\theta \longrightarrow -\theta$.

(2) Plot the bifurcation diagram $\Omega = \Omega(Q)$ for two different sets of initial conditions for values of Q in the interval $[1.3, 1.36]$. What is the value Q at which the period gets doubled. What is the value of Q at which the symmetry $t \longrightarrow t + T_D$ is spontaneously broken.

(3) In this question we use the initial conditions

$$\theta = 0.0 \text{ radian} , \ \Omega = 0.0 \text{ radian}/s.$$

Calculate the orbit and Poincare section and plot the bifurcation diagram $\Omega = \Omega(Q)$ for values of Q in the interval $[1.34, 1.38]$. Determine from the bifurcation diagram the values $Q_{\mathcal{N}}$ for $\mathcal{N} = 1, 2, 3, 4, 5$. Calculate the Feigenbaum ratio. Calculate the accumulation point Q_∞ at which the transition to chaos occurs.

Chapter 6

Molecular Dynamics

6.1   Introduction

In the molecular dynamics approach we attempt to understand the behavior of a classical many-particle system by simulating the trajectory of each particle in the system. In practice this can be applied to systems containing 10^9 particles at most. The molecular dynamics approach is complementary to the more powerful Monte Carlo method. The Monte Carlo method deals with systems that are in thermal equilibrium with a heat bath. The molecular dynamics approach on the other hand is useful in studying how fast in real time a system moves from one microscopic state to another.

We consider a box containing a collection of atoms or molecules. We will use Newton's second law to calculate the positions and velocities of all the molecules as functions of time. Some of the questions we can answer with the molecular dynamics approach are:

- The melting transition.
- The rate of equilibration.
- The rate of diffusion.

As state above molecular dynamics allows us to understand classical systems. A classical treatment can be justified as follows. We consider the case of liquid argon as an example. The energy required to excite an argon atom is of the order of 10eV while the typical kinetic energy of the center of mass of an argon atom is 0.1eV. Thus a collision between two argon atoms will not change the electron configuration of either atoms. Hence for all practical purposes we can ignore the internal structure of argon atoms. Furthermore the wavelength of an argon atom which is of the order of 10^{-7}A is much smaller than the spacing between argon atoms typically of the order of 1A which again justifies a classical treatment.

6.2   The Lennard-Jones Potential

We consider a box containing N argon atoms. For simplicity we will assume that our argon atoms move in two dimensions. The equations of motion of the ith atom which is

located at the position (x_i, y_i) with velocity $(v_{i,x}, v_{i,y})$ read

$$\frac{dv_{i,x}}{dt} = a_{x,i} \ , \ \frac{dx_i}{dt} = v_{i,x}. \tag{6.1}$$

$$\frac{dv_{i,y}}{dt} = a_{y,i} \ , \ \frac{dy_i}{dt} = v_{i,y}. \tag{6.2}$$

Each argon atom experience a force from all other argon atoms. In order to calculate this force we need to determine the interaction potential. We assume that the interaction potential between any pair of argon atoms depend only on the distance between them. Let r_{ij} and $u(r_{ij})$ be the distance and the interaction potential between atoms i and j. The total potential is then given by

$$U = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} u(r_{ij}). \tag{6.3}$$

The precise form of u can be calculated from first principles, i.e. from quantum mechanics. However this calculation is very complicated and in most circumstances a phenomenological form of u will be sufficient.

For large separations r_{ij} the potential $u(r_{ij})$ must be weakly attractive given by the Van der Walls force which arises from electrostatic interaction between the electric dipole moments of the two argon atoms. In other words $u(r_{ij})$ for large r_{ij} is attractive due to the mutual polarization of the two atoms. The Van der Walls potential can be computed from quantum mechanics where it is shown that it varies as $1/r_{ij}^6$. For small separations r_{ij} the potential $u(r_{ij})$ must become strongly repulsive due to the overlap of the electron clouds of the two argon atoms. This repulsion known also as core repulsion is a consequence of Pauli exclusion principle. It is a common practice to choose the repulsive part of the potential u to be proportional to $1/r_{ij}^{12}$. The total potential takes the form

$$u(r) = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right]. \tag{6.4}$$

This is the Lennard-Jones potential. The parameter σ is of dimension length while ϵ is of dimension energy. We observe that at $r = \sigma$ the potential is 0 identically while for $r > 2.5\sigma$ the potential approaches zero rapidly. The minimum of the potential occurs at $r = 2^{1/6}\sigma$. The depth of the potential at the minimum is ϵ.

The force of atom k on atom i is

$$\vec{f}_{k,i} = -\vec{\nabla}_{k,i} u(r_{k,i}) = \frac{24\epsilon}{r_{ki}} \left[2 \left(\frac{\sigma}{r_{ki}} \right)^{12} - \left(\frac{\sigma}{r_{ki}} \right)^6 \right] \hat{r}_{ki}. \tag{6.5}$$

The acceleration of the ith atom is given by

$$a_{x,i} = \frac{1}{m} \sum_{k \neq i} f_{k,i} \cos \theta_{k,i} \quad = \quad \frac{1}{m} \sum_{k \neq i} f_{k,i} \frac{x_i - x_k}{r_{ki}}$$

$$= \quad \frac{24\epsilon}{m} \sum_{k \neq i} \frac{x_i - x_k}{r_{ki}^2} \left[2 \left(\frac{\sigma}{r_{ki}} \right)^{12} - \left(\frac{\sigma}{r_{ki}} \right)^6 \right]. \tag{6.6}$$

$$a_{y,i} = \frac{1}{m} \sum_{k \neq i} f_{k,i} \sin \theta_{k,i} = \frac{1}{m} \sum_{k \neq i} f_{k,i} \frac{y_i - y_k}{r_{ki}}$$

$$= \frac{24\epsilon}{m} \sum_{k \neq i} \frac{y_i - y_k}{r_{ki}^2} \left[2 \left(\frac{\sigma}{r_{ki}} \right)^{12} - \left(\frac{\sigma}{r_{ki}} \right)^6 \right]. \qquad (6.7)$$

6.3 Units, Boundary Conditions and Verlet Algorithm

Reduced Units   We choose σ and ϵ as the units of distance and energy respectively. We also choose the unit of mass to be the mass m of a single argon atom. Everything else is measured in terms of σ, ϵ and m. For example velocity is measured in units of $(\epsilon/m)^{1/2}$ and time in units of $\sigma(\epsilon/m)^{1/2}$. The reduced units are given by

$$\sigma = \epsilon = m = 1. \qquad (6.8)$$

For argon atoms we have the values

$$\sigma = 3.4 \times 10^{-10} m \ , \ \epsilon = 1.65 \times 10^{-21} J = 120 k_B J \ , \ m = 6.69 \times 10^{-26} kg. \qquad (6.9)$$

Thus

$$\sigma \sqrt{\frac{m}{\epsilon}} = 2.17 \times 10^{-12} s. \qquad (6.10)$$

Hence a molecular dynamics simulation which runs for 2000 steps with a reduced time step $\Delta t = 0.01$ corresponds to a total reduced time $2000 \times 0.01 = 20$ which is equivalent to a real time $20\sigma(\epsilon/m)^{1/2} = 4.34 \times 10^{-11} s$.

Periodic Boundary Conditions   The total number of atoms in a real physical system is huge of the order of 10^{23}. If the system is placed in a box the fraction of atoms of the system near the walls of the box is negligible compared to the total number of atoms. In typical simulations the total number of atoms is only of the order of $10^3 - 10^5$ and in this case the fraction of atoms near the walls is considerable and their effect can not be neglected.

In order to reduce edge effects we use periodic boundary conditions. In other words the box is effectively a torus and there are no edges. Let L_x and L_y be the lengths of the box in the x and y directions respectively. If an atom crosses the walls of the box in a particular direction we add or subtract the length of the box in that direction as follows

$$\text{if } (x > L_x) \text{ then } x = x - L_x$$
$$\text{if } (x < 0) \text{ then } x = x + L_x. \qquad (6.11)$$

$$\text{if } (y > L_y) \text{ then } y = y - L_y$$
$$\text{if } (y < 0) \text{ then } y = y + L_y. \qquad (6.12)$$

The maximum separation in the x direction between any two particles is only $L_x/2$ whereas the maximum separation in the y direction between any two particles is only $L_y/2$. This can be implemented as follows

$$\text{if } (x_{ij} > +L_x/2) \text{ then } x_{ij} = x_{ij} - L_x$$
$$\text{if } (x_{ij} < -L_x/2) \text{ then } x_{ij} = x_{ij} + L_x. \tag{6.13}$$

$$\text{if } (y_{ij} > +L_y/2) \text{ then } y_{ij} = y_{ij} - L_y$$
$$\text{if } (y_{ij} < -L_y/2) \text{ then } y_{ij} = y_{ij} + L_y. \tag{6.14}$$

Verlet Algorithm The numerical algorithm we will use is Verlet algorithm. Let us consider the forward and backward Taylor expansions of a function f given by

$$f(t_n + \Delta t) = f(t_n) + \Delta t \frac{df}{dt}|_{t_n} + \frac{1}{2}(\Delta t)^2 \frac{d^2 f}{dt^2}|_{t_n} + \frac{1}{6}(\Delta t)^3 \frac{d^3 f}{dt^3}|_{t_n} + \ldots \tag{6.15}$$

$$f(t_n - \Delta t) = f(t_n) - \Delta t \frac{df}{dt}|_{t_n} + \frac{1}{2}(\Delta t)^2 \frac{d^2 f}{dt^2}|_{t_n} - \frac{1}{6}(\Delta t)^3 \frac{d^3 f}{dt^3}|_{t_n} + \ldots \tag{6.16}$$

Adding these expressions we get

$$f(t_n + \Delta t) = 2f(t_n) - f(t_n - \Delta t) + (\Delta t)^2 \frac{d^2 f}{dt^2}|_{t_n} + O(\Delta t^4). \tag{6.17}$$

We remark that the error is proportional to Δt^4 which is less than the errors in the Euler, Euler-Cromer and second-order Runge-Kutta methods so this method is more accurate. We have therefore for the ith atom

$$x_{i,n+1} = 2x_{i,n} - x_{i,n-1} + (\Delta t)^2 a_{x,i,n}. \tag{6.18}$$

$$y_{i,n+1} = 2y_{i,n} - y_{i,n-1} + (\Delta t)^2 a_{y,i,n}. \tag{6.19}$$

The force and the acceleration are given by

$$f_{k,i,n} = \frac{24\epsilon}{r_{ki,n}} \left[2\left(\frac{\sigma}{r_{ki,n}}\right)^{12} - \left(\frac{\sigma}{r_{ki,n}}\right)^{6} \right]. \tag{6.20}$$

$$a_{x,i,n} = \frac{1}{m} \sum_{k \neq i} f_{k,i,n} \frac{x_{i,n} - x_{k,n}}{r_{ki,n}}. \tag{6.21}$$

$$a_{y,i,n} = \frac{1}{m} \sum_{k \neq i} f_{k,i,n} \frac{y_{i,n} - y_{k,n}}{r_{ki,n}}. \tag{6.22}$$

The separation $r_{ki,n}$ between the two atoms k and i is given by

$$r_{ki,n} = \sqrt{(x_{i,n} - x_{k,n})^2 + (y_{i,n} - y_{k,n})}. \tag{6.23}$$

In the Verlet method it is not necessary to calculate the components $dx_{i,n}/dt$ and $dy_{i,n}/dt$ of the velocity. However since the velocity will be needed for other purposes we will also compute it using the equations

$$v_{x,i,n} = \frac{x_{i,n+1} - x_{i,n-1}}{2\Delta t}. \tag{6.24}$$

$$v_{y,i,n} = \frac{y_{i,n+1} - y_{i,n-1}}{2\Delta t}. \tag{6.25}$$

Let us remark that the Verlet method is not self starting. In other words given the initial conditions $x_{i,1}$, $y_{i,1}$, $v_{x,i,1}$ and $v_{y,i,1}$ we need also to know $x_{i,2}$, $y_{i,2}$, $v_{x,i,2}$ and $v_{y,i,2}$ for the algorithm to start which can be determined using the Euler method.

6.4 Some Physical Applications

6.4.1 Dilute Gas and Maxwell Distribution

A gas in thermal equilibrium is characterized by a temperature T. Molecular dynamics allows us to study how a dilute gas approaches equilibrium. The temperature of the gas can be computed using the molecular dynamics simulations as follows. According to the equipartition theorem the average thermal energy of each quadratic degree of freedom in a gas in thermal equilibrium is equal $k_B T/2$. In other words

$$\frac{1}{2}k_B T = \frac{1}{d} < \frac{1}{2}m\vec{v}^2 > . \tag{6.26}$$

The average $<>$ can be understood in two different but equivalent ways. We can follow the motion of a single atom and take the time average of its kinetic energy. The same result can be obtained by taking the average of the kinetic energy over the different atoms. In this latter case we write

$$\frac{1}{2}k_B T = \frac{1}{dN} \sum_{i=1}^{N} \frac{1}{2}m\vec{v}_i^2. \tag{6.27}$$

Another way of measuring the temperature T of a dilute gas is through a study of the distribution of atom velocities. A classical gas in thermal equilibrium obeys Maxwell distribution. The speed and velocity distributions in two dimensions are given respectively by

$$P(v) = C\frac{v}{k_B T}e^{-\frac{mv^2}{2k_B T}}. \tag{6.28}$$

$$P(v_x) = C_x\frac{1}{\sqrt{k_B T}}e^{-\frac{mv_x^2}{2k_B T}} \ , \ \ P(v_y) = C_y\frac{1}{\sqrt{k_B T}}e^{-\frac{mv_y^2}{2k_B T}}. \tag{6.29}$$

Recall that the probability per unit v of finding an atom with speed v is equal $P(v)$ whereas the probability per unit $v_{x,y}$ of finding an atom with velocity $v_{x,y}$ is equal $P(v_{x,y})$. The

constants C and $C_{x,y}$ are determined from the normalization conditions. There are peaks in the distributions $P(v)$ and $P(v_{x,y})$. Clearly the temperature is related to the location of the peak which occurs in $P(v)$. This is given by

$$k_B T = m v_{\text{peak}}^2. \tag{6.30}$$

6.4.2 The Melting Transition

This is a very important subject which we will discuss at great length in the second lab problem of this chapter.

6.5 Simulation 12: Maxwell Distribution

We consider the motion in two dimensions of N argon atoms in an $L \times L$ box. The interaction potential u between any two atoms in the gas separated by a distance r is given by the Lennard-Jones potential. The numerical algorithm we will use is Verlet algorithm.

In this problem we will always take L odd and N a perfect square. The lattice spacing is defined by

$$a = \frac{L}{\sqrt{N}}.$$

Clearly there are N cells of area $a \times a$. We choose L and N such that $a > 2\sigma$. For simplicity we will use reduced units $\sigma = \epsilon = m = 1$. In order to reduce edge effects we use periodic boundary conditions. In other words the box is effectively a torus and there are no edges. Thus the maximum separation in the x direction between any two particles is only $L/2$ and similarly the maximum separation in the y direction between any two particles is only $L/2$.

The initial positions of the atoms are fixed as follows. The atom $k = \sqrt{N}(i-1)+j$ will be placed at the center of the cell with corners (i,j), $(i+1,j)$, $(i,j+1)$ and $(i+1,j+1)$. Next we perturb in a random way these initial positions by adding random numbers in the interval $[-a/4, +a/4]$ to the x and y coordinates of the atoms. The initial velocities can be chosen in random directions with a speed equal v_0 for all atoms.

(1) Write a molecular dynamics code along the above lines. Take $L = 15$, $N = 25$, $\Delta t = 0.02$, Time $= 500$ and $v_0 = 1$. As a first test verify that the total energy is conserved. Plot the trajectories of the atoms. What do you observe.

(2) As a second test we propose to measure the temperature by observing how the gas approaches equilibrium. Use the equipartition theorem

$$k_B T = \frac{m}{2N} \sum_{i=1}^{N} (v_{i,x}^2 + v_{i,y}^2).$$

Plot T as a function of time. Take Time $= 1000 - 1500$. What is the temperature of the gas at equilibrium.

(3) Compute the speed distribution of the argon atoms by constructing an appropriate histogram as follows. We take the value Time = 2000. We consider the speeds of all particles at all times. There are Time × N values of the speed in this sample. Construct the histogram for this sample by 1) finding the maximum and minimum, 2) dividing the interval into bins, 3) determining the number of times a given value of the speed falls in a bin and (4) properly normalizing the distribution. Compare with the Mawell distribution

$$P_{\text{Maxwell}}(v) = C\frac{v^2}{k_B T}\, e^{-\frac{mv^2}{2k_B T}}.$$

Deduce the temperature from the peak of the distribution given by $k_B T = mv^2_{\text{peak}}$. Compare with the value of the temperature obtained from the equipartition theorem. What happens if we increase the initial speed.

6.6 Simulation 13: Melting Transition

We would like to study the melting transition. First we need to establish the correct conditions for a solid phase. Clearly the temperature must be sufficiently low and the density must be sufficiently high. To make the temperature as low as possible we will start with all particles at rest. In order to obatin maximum attraction between atoms we choose a low density of approximately one particle per unit reduced area. In particular we choose $N = 16$ and $L = 4$.

(1) Show that with these conditions you obtain a crystalline solid with a triangular lattice.

(2) In order to observe melting we must heat up the system. This can be achieved by increasing the kinetic energy of the atoms by hand. A convenient way of doing this is to rescale the current and previous positions of the atoms periodically (say every 1000 steps) as follows

$$\text{hh} = \text{int}(n/1000)$$
$$\text{if } (\text{hh} * 1000.\text{eq}.n) \text{ then}$$
$$x(i, n) = x(i, n + 1) - R(x(i, n + 1) - x(i, n))$$
$$y(i, n) = y(i, n + 1) - R(y(i, n + 1) - y(i, n))$$
$$\text{endif}.$$

This procedure will rescale the velocity by the amount R. We choose $R = 1.5$. Verify that we will indeed reach the melting transition by means of this method. What happens to the energy and the temperature.

Chapter 7

Pseudo Random Numbers and Random Walks

7.1 Random Numbers

A sequence of numbers r_1, r_2,... is called random if there are no correlations between the numbers. The sequence is called uniform if all numbers have an equal probability to occur. More precisely let the probability that a number r_i in the sequence occurs between r and $r + dr$ be $P(r)dr$ where $P(r)$ is the probability distribution. A uniform distribution corresponds $P(r) = $ constant.

Most random number generators on computers generate uniform distributions between 0 and 1. These are sequences of pseudo random numbers since given r_i and its preceding elements we can compute r_{i+1}. Therefore these sequences are not really random and correlations among the numbers of the sequence exist. True random numbers can be found in tables of random numbers determined during say radioactive decay or other naturally occurring random physical phenomena.

7.1.1 Linear Congruent or Power Residue Method

In this method we generate a set of k random numbers r_1,r_2,...,r_k in the interval $[0, M - 1]$ as follows. Given a random number r_{i-1} we generate the next random number r_i by the rule

$$r_i = (ar_{i-1} + c) \bmod M = \text{remainder}\left(\frac{ar_{i-1} + c}{M}\right). \tag{7.1}$$

The notation $y = z \bmod M$ means that we subtract M from z until $0 \leq y \leq M - 1$. The first random number r_1 is supplied by the user and it is called the seed. Also supplied are the multiplier a, the increment c and the modulus M. The remainder is a built-in function in most computer languages. The largest possible integer number generated by the above rule is $M - 1$. Thus the maximum possible period is M, i.e $k \leq M$. In general the period k depends on a, c and M. To get a uniform sequence in the interval $[0, 1]$ we divide by $M - 1$.

Let us take the following example $a = 4, c = 1$ and $M = 9$ with seed $r_1 = 3$. We get a sequence of length 9 given by

$$3, 4, 8, 6, 7, 2, 0, 1, 5. \tag{7.2}$$

After the last number 5 we get 3 and therefore the sequence will repeat. In this case the period is $M = 9$.

It is clear that we need to choose the parameters a, c and M and the seed r_1 with care so that we get the longest sequence of pseudo random numbers. The maximum possible period depends on the size of the computer word. A $32-$bit machine may use $M = 2^{31} = 2 \times 10^9$. The numbers generated by (7.1) are random only in the sense that they are evenly distributed over their range. Equation (7.1) is related to the logistic map which is known to exhibit chaotic behaviour. Although chaos is deterministic it looks random. In the same way although equation (7.1) is deterministic the numbers generated by it look random. This is the reason why they are called pseudo random numbers.

7.1.2   Statistical Tests of Randomness

Period : The first obvious test is to verify that the random number generator has a sufficiently long period for a given problem. We can use the random number generator to plot the position of a random walker. Clearly the plot will repeat itself when the period is reached.

Uniformity : The kth moment of the random number distribution is

$$< x_i^k >= \frac{1}{N} \sum_{i=1}^{N} x_i^k. \tag{7.3}$$

Let $P(x)$ be the probability distribution of the random numbers. Then

$$< x_i^k >= \int_0^1 dx \; x^k P(x) + O(\frac{1}{\sqrt{N}}). \tag{7.4}$$

For a uniform distribution $P(x) = 1$ we must have

$$< x_i^k >= \frac{1}{k+1} + O(\frac{1}{\sqrt{N}}). \tag{7.5}$$

In the words

$$\sqrt{N} \left(\frac{1}{N} \sum_{i=1}^{N} x_i^k - \frac{1}{k+1} \right) = O(1). \tag{7.6}$$

This is a test of uniformity as well as of randomness. To be more precise if $< x_i^k >$ is equal to $1/(k+1)$ then we can infer that the distribution is uniform whereas if the deviation varies as $1/\sqrt{N}$ then we can infer that the distribution is random.

A direct test of uniformity is to divide the unit interval into K equal subintevals (bins) and place each random number in one of these bins. For a uniform distribution we must obtain N/K numbers in each bin where N is the number of generated random numbers.

Chi-Square Statistic   : In the above test there will be statistical fluctuations about the ideal value N/K for each bin. The question is whether or not these fluctuations are consistent with the laws of statistics. The answer is based on the so-called chi-square statistic defined by

$$\chi_\mathrm{m}^2 = \sum_{i=1}^{K} \frac{(N_i - n_\mathrm{ideal})^2}{n_\mathrm{ideal}}. \tag{7.7}$$

In the above definition N_i is the number of random numbers which fall into bin i and n_ideal is the expected number of random numbers in each bin.

The probability of finding any particular value χ^2 which is less than χ_m^2 is found to be proportional to the incomplete gamma function $\gamma(\nu/2, \chi_\mathrm{m}^2/2)$ where ν is the number of degrees of freedom given by $\nu = K - 1$. We have

$$P(\chi^2 \le \chi_\mathrm{m}^2) = \frac{\gamma(\nu/2, \chi_\mathrm{m}^2/2)}{\Gamma(\nu/2)} \equiv P(\nu/2, \chi_\mathrm{m}^2/2). \tag{7.8}$$

The most likely value of χ_m^2, for some fixed number of degrees of freedom ν, corresponds to the value $P(\nu/2, \chi_\mathrm{m}^2/2) = 0.5$. In other words in half of the measurements (bin tests), for some fixed number of degrees of freedom ν, the chi-square statistic predicts that we must find a value of χ_m^2 smaller than the maximum.

Randomness   : Let r_1, r_2,...,r_N be a sequence of random numbers. A very effective test of randomness is to make a scatterplot of $(x_i = r_{2i}, y_i = r_{2i+1})$ for many i. There must be no regularity in the plot otherwise the sequence is not random.

Short-Term Correlations   : Let us define the autocorrelation function

$$\begin{aligned} C(j) &= \frac{<x_i x_{i+j}> - <x_i><x_{i+j}>}{<x_i x_i> - <x_i>^2} \\ &= \frac{<x_i x_{i+j}> - <x_i>^2}{<x_i x_i> - <x_i>^2} \ , \ j = 1, 2, ... \end{aligned} \tag{7.9}$$

In the above equation we have used the fact that $<x_{i+j}> = <x_i>$ for a large sample, i.e. the choice of the origin of the sequence is irrelevant in that case and

$$<x_i x_{i+j}> = \frac{1}{N-j} \sum_{i=1}^{N-j} x_i x_{i+j}. \tag{7.10}$$

Again if x_i and x_{i+j} are independent random numbers which are distributed with the joint probability distribution $P(x_i, x_{i+j})$ then

$$<x_i x_{i+j}> \simeq \int_0^1 dx \int_0^1 dy\, xy P(x, y). \tag{7.11}$$

We have clearly assumed that N is large. For a uniform distribution, viz $P(x, y) = 1$ we get

$$<x_i x_{i+j}> \simeq \frac{1}{4}. \tag{7.12}$$

For a random distrubution the deviation from this result is of order $1/\sqrt{N}$. Hence in the case that the random numbers are not correlated we have

$$C(j) = 0. \tag{7.13}$$

7.2 Random Systems

Both quantum and statistical physics deal with systems that are random or stochastic. These are non deterministic systems as opposed to classical systems. The dynamics of a deterministic system is given by a unique solution to the equations of motion which describes the physics of the system at all times.

We take the case of the diffusion of fluid molecules. For example the motion of dust particles in the atmosphere, the motion of perfume molecules in the air or the motion of milk molecules in a coffee. These are all cases of a Brownian motion.

In the case of a drop of milk in a coffee the white mass of the drop of milk will slowly spread until the coffee takes on a uniform brown color. At the molecular level each milk molecule collides with molecules in the coffee. Clearly it will change direction so frequently that its motion will appear random. This trajectory can be described by a random walk. This is a system in which each milk molecule moves one step at a time in any direction with equal probability.

The trajectory of a dust, perfume or milk molecule is not really random since it can in principle be computed by solving Newton's equations of motion for all molecules which then allows us to know the evolution of the system in time. Although this is possible in principle it will not be feasible in practice. The random walk is thus effectively an approximation. However the large number of molecules and collisions in the system makes the random walk a very good approximation.

7.2.1 Random Walks

Let us consider a one dimensional random walk. It can take steps of lenght unity along a line. It begins at $s_0 = 0$ and the first step is chosen randomly to be either to the left or to right with equal probabilities. In other words there is a 50 per cent chance that the walker moves to the point $s_1 = +1$ and a 50 per cent chance that it moves to the point $s_1 = -1$. Next the walker will again move either to the right or to the left from the point s_1 to the point s_2 with equal probabilities. This process will be repeated N times and we get the position of the walker x_N as a function of the step number N. In the motion of a molecule in a solution the time between steps is a constant and hence the step number N is proportional to time. Therefore x_N is the position of the walker as a function of time.

In general a one-dimensional random walker can move to the right with probability p and to the left with probability $q = 1 - p$ with steps of equal lenght a. The direction of each step is independent of the previous one. The displacement or position of the walker after N steps is

$$x_N = \sum_{i=1}^{N} s_i. \tag{7.14}$$

The walker for $p = q = 1/2$ can be generated by flipping a coin N times. The position is increased by a for heads and decreased by a for tails.

Averaging over many walks each consisting of N steps we get

$$< x_N >= \sum_{i=1}^{N} < s_i >= N < s > . \tag{7.15}$$

In above we have used the fact that the average over every step is the same given by

$$< s_i >=< s >= p(a) + q(-a) = (p - q)a. \tag{7.16}$$

For $p = q = 1/2$ we get $< x_N >= 0$. A better measure of the walk is given by

$$x_N^2 = \left(\sum_{i=1}^{N} s_i \right)^2 . \tag{7.17}$$

The mean square net displacement Δx^2 is defined by

$$\Delta x^2 =< (x_N - < x_N >)^2 >=< x_N^2 > - < x_N >^2 . \tag{7.18}$$

We compute

$$\begin{aligned}
\Delta x^2 &= \sum_{i=1}^{N} \sum_{j=1}^{N} < (s_i - < s >)(s_j - < s >) > \\
&= \sum_{i \neq j = 1}^{N} < (s_i - < s >)(s_j - < s >) > + \sum_{i=1}^{N} < (s_i - < s >)^2 > . \tag{7.19}
\end{aligned}$$

In the first term since $i \neq j$ we have $< (s_i - < s >)(s_j - < s >) >=< (s_i - < s >) >< (s_j - < s >) >$. But $< (s_i - < s >) >= 0$. Thus

$$\begin{aligned}
\Delta x^2 &= \sum_{i=1}^{N} < (s_i - < s >)^2 > \\
&= N(< s_i^2 > - < s >^2 >) \\
&= N(a^2 - (p - q)^2 a^2) \\
&= 4Npqa^2. \tag{7.20}
\end{aligned}$$

For $p = q = 1/2$ and $a = 1$ we get

$$< x_N^2 > = N. \tag{7.21}$$

The main point is that since N is proportional to time we have $< x_N^2 > \propto t$. This is an example of a diffusive behaviour.

7.2.2 Diffusion Equation

The random walk is successful in simulating many physical systems because it is related to the solutions of the diffusion equation. To see this we start from the probability $P(i, N)$ that the random walker is at site s_i after N steps. This is given by

$$P(i, N) = \frac{1}{2}\left(P(i + 1, N - 1) + P(i - 1, N - 1)\right). \tag{7.22}$$

Let τ be the time between steps and a the lattice spacing. Then $t = N\tau$ and $x = ia$. Also we define $P(x, t) = P(i, N)/a$. We get

$$P(x, t) = \frac{1}{2}\left(P(x + a, t - \tau) + P(x - a, t - \tau)\right). \tag{7.23}$$

Let us rewrite this equation as

$$\frac{1}{\tau}\left(P(x, t) - P(x, t - \tau)\right) = \frac{a^2}{2\tau}\left[P(x + a, t - \tau) - 2P(x, t - \tau) + P(x - a, t - \tau)\right]\frac{1}{a^2}. \tag{7.24}$$

In the limit $a \longrightarrow 0$, $\tau \longrightarrow 0$ with the ratio $D = a^2/2\tau$ kept fixed we obtain the equation

$$\frac{\partial P(x, t)}{\partial t} = D\frac{\partial^2 P(x, t)}{\partial x^2}. \tag{7.25}$$

This is the diffusion equation. Generalization to $3-$dimensions is

$$\frac{\partial P(x, y, z, t)}{\partial t} = D\nabla^2 P(x, y, z, t). \tag{7.26}$$

A particular solution of (7.25) is given by

$$P(x, t) = \frac{1}{\sigma}e^{-\frac{x^2}{2\sigma^2}} \ , \ \sigma = \sqrt{2Dt}. \tag{7.27}$$

In other words the spatial distribution of the diffusing molecules is always a gaussian with half-width σ increasing with time as $\sqrt{t}$.

The average of any function f of x is given by

$$< f(x, t) >= \int f(x)P(x, t)dx. \tag{7.28}$$

Let us multiply both sides of (7.25) by $f(x)$ and then integrate over x, viz

$$\int f(x)\frac{\partial P(x, t)}{\partial t}dx = D\int f(x)\frac{\partial^2 P(x, t)}{\partial x^2}dx. \tag{7.29}$$

Clearly

$$\int f(x)\frac{\partial P(x, t)}{\partial t}dx = \int \frac{\partial}{\partial t}\left(f(x)P(x, t)\right)dx = \frac{d}{dt}\int f(x)P(x, t)dx = \frac{d}{dt} < f(x) > . \tag{7.30}$$

Thus

$$
\begin{aligned}
\frac{d}{dt} < f(x) > &= D \int f(x) \frac{\partial^2 P(x,t)}{\partial x^2} dx \\
&= D \left(f(x) \frac{\partial P(x,t)}{\partial x} \right) |_{x=-\infty}^{x=+\infty} - D \int \frac{\partial f(x)}{\partial x} \frac{\partial P(x,t)}{\partial x} dx. \quad (7.31)
\end{aligned}
$$

We have $P(x = \pm\infty, t) = 0$ and also all spatial derivatives are zero at $x = \pm\infty$. We then get

$$
\frac{d}{dt} < f(x) > = -D \int \frac{\partial f(x)}{\partial x} \frac{\partial P(x,t)}{\partial x} dx. \quad (7.32)
$$

Let us choose $f(x) = x$. Then

$$
\frac{d}{dt} < x > = -D \int \frac{\partial P(x,t)}{\partial x} dx = 0. \quad (7.33)
$$

In other words $< x >=$ constant and since $x = 0$ at $t = 0$ we must have constant $= 0$. Thus

$$
< x >= 0. \quad (7.34)
$$

Let us next choose $f(x) = x^2$. Then

$$
\begin{aligned}
\frac{d}{dt} < x^2 > &= -2D \int x \frac{\partial P(x,t)}{\partial x} dx \\
&= 2D. \quad (7.35)
\end{aligned}
$$

Hence

$$
< x^2 > = 2Dt. \quad (7.36)
$$

This is the diffusive behaviour we have observed in the random walk problem.

7.3   The Random Number Generators RAN $0, 1, 2$

Linear congruential generators are of the form

$$
r_i = (a r_{i-1} + c) \bmod M. \quad (7.37)
$$

For $c > 0$ the linear congruential generators are called mixed. They are denoted by $\mathrm{LCG}(a, c, M)$. The random numbers generated with $\mathrm{LCG}(a, c, M)$ are in the range $[0, M - 1]$.

For $c = 0$ the linear congruential generators are called multiplicative. They are denoted by $\mathrm{MLCG}(a, M)$. The random numbers generated with $\mathrm{MLCG}(a, M)$ are in the range $[1, M - 1]$.

In the case that a is a primitive root modulo M and M is a prime the period of the generator is $M - 1$. A number a is a primitive root modulo M means that for any integer n such that $\gcd(n, M) = 1$ there exists a k such that $a^k = n \bmod M$.

An example of MLCG is RAN0 due to Park and Miller which is used extensively on IBM computers. In this case

$$a = 16807 = 7^5 \ , \ M = 2^{31} - 1. \tag{7.38}$$

The period of this generator is not very long given by

$$\text{period} = 2^{31} - 2 \simeq 2.15 \times 10^9. \tag{7.39}$$

This generator can not be implemented directly in a high level language because of integer overflow. Indeed the product of a and $M-1$ exceeds the maximum value for a $32-$bit integer. Assembly language implementation using $64-$bit product register is straightforward but not portable.

A better solution is given by Schrage's algorithm. This algorithm allows the multiplication of two $32-$bit integers without using any intermediate numbers which are larger than 32 bits. To see how this works explicitly we factor M as

$$M = aq + r. \tag{7.40}$$

$$r = M \bmod a \ , \ q = [\frac{M}{r}]. \tag{7.41}$$

In the above equation $[\ ]$ denotes integer part. Remark that

$$r = M \bmod a = M - [\frac{M}{a}]a. \tag{7.42}$$

Thus by definition $r < a$. We will also demand that $r < q$ and hence

$$\frac{r}{qa} << 1. \tag{7.43}$$

We have also

$$\begin{aligned} X_{i+1} = aX_i \bmod M \ &= \ aX_i - [\frac{aX_i}{M}]M \\ &= \ aX_i - [\frac{aX_i}{aq + r}]M. \end{aligned} \tag{7.44}$$

We compute

$$\begin{aligned} \frac{aX_i}{aq + r} = \frac{X_i}{q + \frac{r}{a}} \ &= \ \frac{X_i}{q}\frac{1}{1 + \frac{r}{qa}} \\ &= \ \frac{X_i}{q}(1 - \frac{r}{qa}) \\ &= \ \frac{X_i}{q} - \frac{X_i}{aq}\frac{r}{q}. \end{aligned} \tag{7.45}$$

Clearly

$$\frac{X_i}{aq} = \frac{X_i}{M-r} \simeq \frac{X_i}{M} < 1. \tag{7.46}$$

Hence

$$[\frac{aX_i}{M}] = [\frac{X_i}{q}], \tag{7.47}$$

if neglecting $\epsilon = (rX_i)/(aq^2)$ does not affect the integer part of aX_i/M and

$$[\frac{aX_i}{M}] = [\frac{X_i}{q}] - 1, \tag{7.48}$$

if neglecting ϵ does affect the integer part of aX_i/M. Therefore we get

$$
\begin{aligned}
X_{i+1} &= aX_i - [\frac{aX_i}{M}](aq+r) \\
&= a(X_i - [\frac{aX_i}{M}]q) - [\frac{aX_i}{M}]r \tag{7.49} \\
&= a(X_i - [\frac{X_i}{q}]q) - [\frac{X_i}{q}]r \tag{7.50} \\
&= a(X_i \bmod q) - [\frac{X_i}{q}]r, \tag{7.51}
\end{aligned}
$$

if

$$a(X_i \bmod q) - [\frac{X_i}{q}]r \geq 0. \tag{7.52}$$

Also

$$
\begin{aligned}
X_{i+1} &= aX_i - [\frac{aX_i}{M}](aq+r) \\
&= a(X_i - [\frac{aX_i}{M}]q) - [\frac{aX_i}{M}]r \tag{7.53} \\
&= a(X_i - [\frac{X_i}{q}]q + q) - [\frac{X_i}{q}]r + r \tag{7.54} \\
&= a(X_i \bmod q) - [\frac{X_i}{q}]r + M, \tag{7.55}
\end{aligned}
$$

if

$$a(X_i \bmod q) - [\frac{X_i}{q}]r < 0. \tag{7.56}$$

The generator RAN0 contains serial correlations. For example $D-$dimensional vectors $(x_1, ..., x_D)$, $(x_{D+1}, ..., x_{2D})$,...which are obtained by successive calls of RAN0 will lie on a small number of parallel $(D-1)-$dimensional hyperplanes. Roughly there will be $M^{1/D}$ such hyperplanes. In particular successive points (x_i, x_{i+1}) when binned into a $2-$dimensional plane for $i = 1, ..., N$ will result in a distribution which fails the χ^2 test for $N \geq 10^7$ which is much less than the period $M - 1$.

The RAN1 is devised so that the correlations found in RAN0 is removed using the Bays-Durham algorithm. The Bays-Durham algorithm shuffles the sequence to remove low-order serial correlations. In other words it changes the order of the numbers so that the sequence is not dependent on order and a given number is not correlated with previous numbers. More precisely the jth random number is output not on the jth call but on a randomized later call which is on average the $j + 32$th call on .

The RAN2 is an improvement over RAN1 and RAN0 due to L'Ecuyer. It uses two sequences with different periods so as to obtain a new sequence with a larger period equal to the least common multiple of the two periods. In this algorithm we add the two sequences modulo the modulus M of one of them. In order to avoid overflow we subtract rather than add and if the result is negative we add $M - 1$ so as to wrap around into the inetrval $[0, M - 1]$. L'Ecuyer uses the two sequences

$$M_1 = 2147483563 \ , \ a_1 = 40014 \ , \ q_1 = 53668 \ , \ r_1 = 12211. \tag{7.57}$$

$$M_2 = 2147483399 \ , \ a_2 = 40692 \ , \ q_2 = 52774 \ , \ r_2 = 3791. \tag{7.58}$$

The period is 2.3×10^{18}. Let us also point out that RAN2 uses Bays-Durham algorithm in order to implement an additional shuffle.

We conclude this section by discussing another generator based on the linear congruential method which is the famous random number generator RAND given by

$$\text{RAND} = \text{LCG}(69069, 1, 2^{32}). \tag{7.59}$$

The period of this generator is 2^{32} and lattice structure is present for higher dimensions $D \geq 6$.

7.4   Simulation 14: Random Numbers

Part I   We consider a linear congruential pseudo-random number generator given by

$$r_{i+1} = \text{remainder}\left(\frac{ar_i + c}{M}\right).$$

We take the values

$$a = 899, c = 0, M = 32768, r_1 = 12 \ \ \text{"good"}$$
$$a = 57, c = 1, M = 256, r_1 = 10 \ , \ \text{"bad"}.$$

The function "remainder" is implemented in Fortran by

$$\text{remainder} \ \frac{a}{b} = \text{mod}(a, b).$$

(1) Compute the sequence of the random numbers r_i obtained using the above parameters. Plot r_i as a function of i. Construct a scatterplot $(x_i = r_{2i}, y_i = r_{2i+1})$.

(2) Compute the average of the random numbers. What do you observe.

(3) Let N be the number of generated random numbers. Compute the correlation functions defined by

$$\mathrm{sum}_1(k) = \frac{1}{N-k} \sum_{i=1}^{N-k} x_i x_{i+k}.$$

$$\mathrm{sum}_2 = \frac{\mathrm{sum}_1(k) - <x_i>^2}{\mathrm{sum}_1(0) - <x_i>^2}.$$

What is the behavior of these functions as a function of k.

(4) Compute the period of the above generators.

Part II We take N random numbers in the interval $[0,1]$ which we divide into K bins of length $\delta = 1/K$. Let N_i be the number of random numbers which fall in the ith bin. For a uniform sequence of random numbers the number of random numbers in each bin is $n_{\mathrm{ideal}} = N/K$.

(1) Verify this result for the generator "rand" found in the standard Fortran library with seed given by seed = 32768. We take $K = 10$ and $N = 1000$. Plot N_i as a function of the position x_i of the ith bin.

(2) The number of degrees of freedom is $\nu = K - 1$. The most probable value of the chi-square statistics χ^2 is ν. Verify this result for a total number of bin tests equal $L = 1000$ and $K = 11$. Each time calculate the number of times L_i in the $L = 1000$ bin tests we get a specific value of χ^2. Plot L_i as a function of χ^2. What do you observe.

7.5   Simulation 15: Random Walks

Part I We consider the motion of a random walker in one dimension. The walker can move with a step $s_i = a$ to the right with a probability p or with a step $s_i = -a$ to the left with a probability $q = 1 - p$. After N steps the position of the walker is $x_N = \sum_i s_i$. We take

$$p = q = \frac{1}{2} \ , \ a = 1.$$

In order to simulate the motion of a random walker we need a generator of random numbers. In this problem we work with the generator "rand" found in the standard Fortran library. We call this generator as follows

$$\mathrm{call\ srand(seed)}$$

$$\mathrm{rand()}$$

The motion of the random walker is implemented with the code

$$\text{if } (\text{rand}() < p) \text{ then}$$
$$x_N = x_N + a$$
$$\text{else}$$
$$x_N = x_N - a$$
$$\text{endif.}$$

(1) Compute the positions x_i of three different random walkers as functions of the step number i. We take $i = 1, 100$. Plot the three trajectories.

(2) We consider now the motion of $K = 500$ random walkers. Compute the averages

$$< x_N > = \frac{1}{K} \sum_{i=1}^{K} x_N^{(i)} \ , \ \ < x_N^2 > = \frac{1}{K} \sum_{i=1}^{K} (x_N^{(i)})^2.$$

In the above equations $x_N^{(i)}$ is the position of the ith random walker after N steps. Study the behavior of these averages as a function of N. Compare with the theoretical predictions.

Part II (optional)   We consider next a random walker in two dimensions on an infinite lattice of points. From any point (i, j) on the lattice the walker can reach one of the 4 possible nearest neighbor sites $(i+1, j)$, $(i-1, j)$, $(i, j+1)$ and $(i, j-1)$ with probabilities p_x, q_x, p_y and q_y respectively such that $p_x + q_x + p_y + q_y = 1$. For simplicity we will assume that $p_x = q_x = p_y = q_y = 0.25$.

(1) Compute the averages $< \vec{r}_N >$ and $< \vec{r}_N^2 >$ as function of the number of steps N for a collection of $L = 500$ two dimensional random walkers. We consider the values $N = 10, ..., 1000$.

Chapter 8

Monte Carlo Integration

8.1 Numerical Integration

8.1.1 Rectangular Approximation Revisted

As usual let us start with something simple. The approximation of one-dimensional integrals by means of the rectangular approximation. This is a topic we have already discussed before.

Let us then begin by recalling how the rectangular approximation of one dimensional integrals works. We consider the integral

$$F = \int_a^b f(x)dx. \tag{8.1}$$

We discretize the $x-$interval so that we end up with N equal small intervals of lenght Δx, viz

$$x_n = x_0 + n\Delta x \ , \ \ \Delta x = \frac{b-a}{N} \tag{8.2}$$

Clearly $x_0 = a$ and $x_N = b$. Riemann definition of the integral is given by the following limit

$$F = \lim \ \Delta x \sum_{n=0}^{N-1} f(x_n) \ , \ \ \Delta x \longrightarrow 0 \ , \ N \longrightarrow \infty \ , \ b - a = \text{fixed}. \tag{8.3}$$

The first approximation which can be made is to simply drop the limit. We get the so-called rectangular approximation given by

$$F_N = \Delta x \sum_{n=0}^{N-1} f(x_n). \tag{8.4}$$

The error can be computed as follows. We start with the Taylor expansion

$$f(x) = f(x_n) + (x - x_n)f^{(1)}(x_n) + \frac{1}{2!}(x - x_n)^2 f^{(2)}(x_n) + \ ... \tag{8.5}$$

Thus

$$\int_{x_n}^{x_{n+1}} dx \ f(x) = f(x_n)\Delta x + \frac{1}{2!}f^{(1)}(x_n)(\Delta x)^2 + \frac{1}{3!}f^{(2)}(x_n)(\Delta x)^3 + ... \tag{8.6}$$

The error in the interval $[x_n, x_{n+1}]$ is

$$\int_{x_n}^{x_{n+1}} dx \ f(x) - f(x_n)\Delta x = \frac{1}{2!}f^{(1)}(x_n)(\Delta x)^2 + \frac{1}{3!}f^{(2)}(x_n)(\Delta x)^3 + ... \tag{8.7}$$

This is of order $1/N^2$. But we have N subintervals. Thus the total error is of order $1/N$.

8.1.2 Midpoint Approximation of Multidimensional Integrals

Let us start with the two dimensional integral

$$F = \int_R dx \ dy \ f(x,y). \tag{8.8}$$

R is the domain of integration. In order to give the midpoint approximation of this integral we imagine a rectangle of sides $x_b - x_a$ and $y_b - y_a$ which encloses the region R and we divide it into squares of lenght h. The points in the x/y direction are

$$x_i = x_a + (i - \frac{1}{2})h \ , \ i = 1, ..., n_x. \tag{8.9}$$

$$y_i = y_a + (i - \frac{1}{2})h \ , \ i = 1, ..., n_y. \tag{8.10}$$

The number of points in the x/y direction are

$$n_x = \frac{x_b - x_a}{h} \ , \ n_y = \frac{y_b - y_a}{h}. \tag{8.11}$$

The number of cells is therefore

$$n = n_x n_y = \frac{(x_b - x_a)(y_b - y_a)}{h^2}. \tag{8.12}$$

The integral is then approximated by

$$F = h^2 \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} f(x_i, y_j) H(x_i, y_j). \tag{8.13}$$

The Heaviside function is defined by

$$H(x_i, y_j) = 1 \text{ if } (x_i, y_j) \in R \text{ otherwise } H(x_i, y_j) = 0. \tag{8.14}$$

The generalization to many dimensions is straightforward. We get

$$F = h^d \sum_{i_1=1}^{n_1} ... \sum_{i_d=1}^{n_d} f(x_1^{i_1}, ..., x_d^{i_d}) H(x_1^{i_1}, ..., x_d^{i_d}). \tag{8.15}$$

The meaning of the different symbols is obvious.

The midpoint approximation is an improvement over the rectangular approximation. To see this let us consider a one dimensional integral

$$F = \int_R dx \ f(x). \tag{8.16}$$

The midpoint approximation reads in this case as follows

$$F = h \sum_{i=1}^{n_x} f(x_i) H(x_i) = h \sum_{i=1}^{n_x} f(x_i). \tag{8.17}$$

Let us say that we have n_x intervals $[x_i, x_{i+1}]$ with $x_0 = a$ and $x_i = x_a + (i - 0.5)h$, $i = 1, ..., n_x - 1$. The term $hf(x_{i+1})$ is associated with the interval $[x_i, x_{i+1}]$. It is clear that we can write this approximation as

$$F = h \sum_{i=0}^{n_x-1} f(\frac{x_i + x_{i+1}}{2}) \ , \ x_i = x_a + ih. \tag{8.18}$$

The error in the interval $[x_i, x_{i+1}]$ is given by

$$\int_{x_i}^{x_{i+1}} f(x) \ dx - f(\frac{x_i + x_{i+1}}{2})\Delta x = \frac{1}{24} f''(x_i)(\Delta x)^3 + ... \tag{8.19}$$

The total error is thereore $1/n_x^2$ as opposed to the $1/n_x$ of the rectangular approximation.

Let us do this in two dimensions. We write the error as

$$\int_{x_i}^{x_{i+1}} \int_{y_j}^{y_{j+1}} f(x, y) \ dx \ dy - f(\frac{x_i + x_{i+1}}{2}, \frac{y_j + y_{j+1}}{2})\Delta x \Delta y \tag{8.20}$$

As usual we use Taylor series in the form

$$
\begin{aligned}
f(x, y) &= f(x_i, y_j) + f'_x(x_i, y_j)(x - x_i) + f'_y(x_i, y_j)(y - y_j) + \frac{1}{2}f''_x(x_i, y_j)(x - x_i)^2 \\
&+ \frac{1}{2}f''_y(x_i, y_j)(y - y_j)^2 + f''_{xy}(x_i, y_j)(x - x_i)(y - y_j) + ...
\end{aligned} \tag{8.21}
$$

We find

$$\int_{x_i}^{x_{i+1}} \int_{y_j}^{y_{j+1}} f(x, y) \ dx \ dy - f(\frac{x_i + x_{i+1}}{2}, \frac{y_j + y_{j+1}}{2})\Delta x \Delta y = \frac{1}{24} f''_x(x_i, y_j)(\Delta x)^3 \Delta y + \frac{1}{24} f''_y(x_i, y_j)\Delta x(\Delta y)^3$$
$$+ \ ... \tag{8.22}$$

Since $\Delta x = \Delta y = h$. The individual error is proportional to h^4. The total error is nh^4 where $n = n_x n_y$. Since n is proportional to $1/h^2$, the total error in dimension two is proportional to h^2 or equivalently to $1/n$. As we have already seen the same method led to an error proportional to $1/n^2$ in dimension one. Thus as we increase the number of dimensions the error becomes worse. If in one dimension the error behaves as $1/n^a$ then in dimension d it will behave as $1/n^{\frac{a}{d}}$. In other words classical numerical integration methods become impractical at sufficiently higher dimensions (which is the case of quantum mechanics and statistical mechanics).

8.1.3   Spheres and Balls in d Dimensions

The volume of a ball of radius R in d dimensions is given by

$$
\begin{aligned}
V_d &= \int_{x_1^2+\ldots+x_d^2\leq R^2} dx_1\ldots dx_d \\
&= \int_{x_1^2+\ldots+x_d^2\leq R^2} r^{d-1}\ dr\ d\Omega_{d-1} \\
&= \frac{R^d}{d}\int d\Omega_{d-1} \\
&= \frac{R^d}{d}\frac{2\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2})}.
\end{aligned}
\tag{8.23}
$$

The surface of a sphere of radius R in d dimensions is similarly given by

$$
\begin{aligned}
S_{d-1} &= \int_{x_1^2+\ldots+x_d^2=R^2} dx_1\ldots dx_d \\
&= R^{d-1}\frac{2\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2})}.
\end{aligned}
\tag{8.24}
$$

Here are some properties of the gamma function

$$
\Gamma(1)=1\ ,\ \Gamma(\frac{1}{2})=\sqrt{\pi}\ ,\ \Gamma(n+1)=n\Gamma(n).
\tag{8.25}
$$

In order to compute numerically the volume of the ball in any dimension d we need a recursion formula which relates the volume of the ball in d dimensions to the volume of the ball in $d-1$ dimensions. The derivation goes as follows

$$
\begin{aligned}
V_d &= \int_{-R}^{+R} dx_d \int_{x_1^2+\ldots+x_{d-1}^2\leq R^2-x_d^2} dx_1\ldots dx_{d-1} \\
&= \int_{-R}^{+R} dx_d \int_0^{\sqrt{R^2-x_d^2}} r^{d-2}\ dr \int d\Omega_{d-2} \\
&= \frac{V_{d-1}}{R^{d-1}}\int_{-R}^{+R} dx_d\ (R^2-x_d^2)^{\frac{d-1}{2}}.
\end{aligned}
\tag{8.26}
$$

At each dimension d we are thus required to compute only the remaining integral over x_d using, for instance, the midpoint approximation while the volume V_{d-1} is determined in the previous recursion step. The starting point of the recursion process, for example the volume in $d=2$, can be determined also using the midpoint approximation. As we will see in the lab problems this numerical calculation is very demanding with significant errors compared with the Monte Carlo method.

8.2   Monte Carlo Integration: Simple Sampling

Let us start with the one dimensional integral

$$
F = \int_a^b dx\ f(x).
\tag{8.27}
$$

A Monte Carlo method is any procedure which uses (pseudo) random numbers to compute or estimate the above integral. In the following we will describe two very simple Monte Carlo methods based on simple sampling which give an approximate value for this integral. As we progress we will be able to give more sophisticated Monte Carlo methods. First we start with the sampling (hit or miss) method then we go on to the sample mean method.

8.2.1 Sampling (Hit or Miss) Method

This method consists of the following three main steps:

- We imagine a rectangle of width $b - a$ and height h such that h is greater than the maximum value of $f(x)$, i.e the function is within the boundaries of the rectangle.

- To estimate the value F of the integral we choose n pairs of uniform random numbers (x_i, y_i) where $a \leq x_i \leq b$ and $0 \leq y_i \leq h$.

- Then we evaluate the function f at the points x_i. Let n_{in} be the number of random points (x_i, y_i) such that $y_i \leq f(x_i)$. The value F of the integral is given by

$$F = A \frac{n_{in}}{n} \ , \ A = h(b - a). \tag{8.28}$$

8.2.2 Sample Mean Method

We start from the mean-value theorem of calculus, viz

$$F = \int_a^b dx \ f(x) = (b - a) < f > . \tag{8.29}$$

$< f >$ is the average value of the function $f(x)$ in the range $a \leq x \leq b$. The sample mean method estimates the average $< f >$ as follows:

- We choose n random points x_i from the interval $[a, b]$ which are distributed uniformly.

- We compute the values of the function $f(x)$ at these point.

- We take their average. In other words

$$F = (b - a) \frac{1}{n} \sum_{i=1}^{n} f(x_i). \tag{8.30}$$

This is formally the same as the rectangular approximation. The only difference is that here the points x_i are chosen randomly from the interval $[a, b]$ whereas the points in the rectangular approximation are chosen with equal spacing. For lower dimensional integrals the rectangular approximation is more accurate whereas for higher dimensional integrals the sample mean method becomes more accurate.

8.2.3 Sample Mean Method in Higher Dimensions

We start with the two dimensional integral

$$F = \int_R dx \ dy \ f(x, y). \tag{8.31}$$

Again we consider a rectangle of sides $y_b - y_a$ and $x_b - x_a$ which encloses the region R. The Monte carlo sample mean method yields the approximation

$$F = A\frac{1}{n}\sum_{i=1}^{n} f(x_i, y_i)H(x_i, y_i). \qquad (8.32)$$

The points x_i are random and uniformly distributed in the interval $[x_a, x_b]$ whereas the points y_i are random and uniformly distributed in the interval $[y_a, y_b]$. A is the areas of the rectangle, i.e $A = (x_b - x_a)(y_b - y_a)$. The Heaviside function is defined by

$$H(x_i, y_i) = 1 \text{ if } (x_i, y_i) \in R \text{ otherwise } H(x_i, y_i) = 0. \qquad (8.33)$$

Generalization to higher dimensions is obvious. For example in three dimensions we would have

$$F = \int_R dx \ dy \ dz \ f(x, y, z) \longrightarrow F = V\frac{1}{n}\sum_{i=1}^{n} f(x_i, y_i, z_i)H(x_i, y_i, z_i). \qquad (8.34)$$

V is the volume of the parallelepiped which encloses the three dimensional region R.

8.3   The Central Limit Theorem

Let $p(x)$ be a probability distribution function. We generate (or measure) n values x_i of a certain variable x according to the probability distribution function $p(x)$. The average $y_1 = < x_i >$ is given by

$$y_1 = < x_i > = \frac{1}{n}\sum_{i=1}^{n} x_i p(x_i). \qquad (8.35)$$

We repeat this measurement N times thus obtaining N averages y_1, y_2,...,y_N. The mean z of the averages y_i is

$$z = \frac{1}{N}\sum_{i=1}^{N} y_i. \qquad (8.36)$$

The question we want to answer is: what is the probability distribution function of z.

Clearly the probability of obtaining a particular value z is the product of the probabilities of obtaining the individual averages y_i (which are assumed to be independent) with the constraint that the average of y_i is z.

Let $\tilde{p}(y)$ be the probability distribution function of the average y and let $P(z)$ be the probability distribution of the average z of the averages. We can then write $P(z)$ as

$$P(z) = \int dy_1 ... \int dy_N \ \tilde{p}(y_1)...\tilde{p}(y_N)\delta(z - \frac{y_1 + ... + y_N}{N}). \qquad (8.37)$$

The delta function expresses the constraint that z is the average of y_i. The delta function can be written as

$$\delta(z - \frac{y_1 + ... + y_N}{N}) = \frac{1}{2\pi} \int dq e^{iq(z - \frac{y_1 + ... + y_N}{N})}. \tag{8.38}$$

Let μ be the actual average of y_i, i.e.

$$\mu = <y_i> = \int dy \tilde{p}(y) y. \tag{8.39}$$

We write

$$
\begin{aligned}
P(z) &= \frac{1}{2\pi} \int dq e^{iq(z-\mu)} \int dy_1 \ \tilde{p}(y_1) e^{\frac{iq}{N}(\mu - y_1)} ... \int dy_N \ \tilde{p}(y_N) e^{\frac{iq}{N}(\mu - y_N)} \\
&= \frac{1}{2\pi} \int dq e^{iq(z-\mu)} \left[\int dy \ \tilde{p}(y) e^{\frac{iq}{N}(\mu - y)} \right]^N.
\end{aligned} \tag{8.40}
$$

But

$$
\begin{aligned}
\int dy \ \tilde{p}(y) e^{\frac{iq}{N}(\mu - y)} &= \int dy \ \tilde{p}(y) \left[1 + \frac{iq}{N}(\mu - y) - \frac{q^2(\mu - y)^2}{2N^2} + ... \right] \\
&= 1 - \frac{q^2 \sigma^2}{2N^2} + ...
\end{aligned} \tag{8.41}
$$

We have used

$$\int dy \ \tilde{p}(y)(\mu - y)^2 = <y^2> - <y>^2 = \sigma^2. \tag{8.42}$$

Hence

$$
\begin{aligned}
P(z) &= \frac{1}{2\pi} \int dq e^{iq(z-\mu)} e^{-\frac{q^2 \sigma^2}{2N}} \\
&= \frac{1}{2\pi} e^{-\frac{N}{2\sigma^2}(z-\mu)^2} \int dq e^{-\frac{\sigma^2}{2N}(q - \frac{iN}{\sigma}(z-\mu))^2} \\
&= \frac{1}{\sqrt{2\pi}} \frac{e^{-\frac{(z-\mu)^2}{2\sigma_N^2}}}{\sigma_N}.
\end{aligned} \tag{8.43}
$$

$$\sigma_N = \frac{\sigma}{\sqrt{N}}. \tag{8.44}$$

This is the normal distribution. Clearly the result does not depend on the original probability distribution functions $p(x)$ and $\tilde{p}(y)$.

The average z of N random numbers y_i corresponding to a probability distribution function $\tilde{p}(y)$ is distributed according to the normal probability distribution function with average equal to the average value of $\tilde{p}(y)$ and variance equal to the variance of $\tilde{p}(y)$ divided by $\sqrt{N}$.

8.4   Monte Carlo Errors and Standard Deviation

In any Monte Carlo approximation method the error goes as $1/\sqrt{N}$ where N is the number of samples. This behaviour is independent of the integrand and is independent of the number of dimensions. In contrast if the error in a classical numerical approximation method goes as $1/N^a$ in one dimension (where N is now the number of intervals) then the error in the same approximation method will go as $1/N^{\frac{a}{d}}$ in d dimensions. Thus as we increase the number of dimensions the error becomes worse. In other words classical numerical integration methods become impractical at sufficiently higher dimensions. This is the fundamental appeal of Monte Carlo methods in physics (quantum mechanics and statistical mechanics) where we usually and so often encounter integrals of infinite dimensionality.

Let us again consider for simplicity the one dimensional integral as an example. We take

$$F = \int_a^b dx\ f(x). \tag{8.45}$$

The Monte Carlo sample mean method gives the approximation

$$F_N = (b-a) <f> \ , \ <f> = \frac{1}{N}\sum_{i=1}^N f(x_i). \tag{8.46}$$

The error is by definition given by

$$\Delta = F - F_N. \tag{8.47}$$

However in general we do not know the exact result F. The best we can do is to calculate the probability that the approximate result F_N is within a certain range centered around the exact result F.

The starting point is the central limit theorem. This states that the average z of N random numbers y_α corresponding to a probability distribution function $\tilde{p}(y)$ is distributed according to the normal probability distribution function. Here the variable y is (we assume for simplicity that $b-a=1$)

$$y = \frac{1}{N}\sum_{i=1}^N f(x_i). \tag{8.48}$$

We make M measurements y_α of y. We write

$$y_\alpha = \frac{1}{N}\sum_{i=1}^N f(x_{i,\alpha}). \tag{8.49}$$

The mean z of the averages is given by

$$z = \frac{1}{M}\sum_{\alpha=1}^M y_\alpha. \tag{8.50}$$

According to the central limit theorem the mean z is distributed according to the normal probability distribution function with average equal to the average value $< y >$ of y_α and variance equal to the variance of y_α divided by $\sqrt{M}$, viz

$$\sqrt{\frac{M}{2\pi\tilde{\sigma}_M^2}} \exp\left(-M\frac{(z- < y >)^2}{2\tilde{\sigma}_M^2}\right). \tag{8.51}$$

The $\tilde{\sigma}_M$ is the standard deviation of the mean given by the square root of the variance

$$\tilde{\sigma}_M^2 = \frac{1}{M-1}\sum_{\alpha=1}^{M}(y_\alpha- < y >)^2. \tag{8.52}$$

The use of $M-1$ instead of M is known as Bessel's correction. The reason for this correction is the fact that the computation of the mean $< y >$ reduces the number of independent data points y_α by one. For very large M we can replace $\tilde{\sigma}_M$ with σ_M defined by

$$\tilde{\sigma}_M^2 \sim \sigma_M^2 = \frac{1}{M}\sum_{\alpha=1}^{M}(y_\alpha- < y >)^2 =< y^2 > - < y >^2 . \tag{8.53}$$

The standard deviation of the sample (one single measurement with N data points) is given by the square root of the variance

$$\tilde{\sigma}^2 = \frac{1}{N-1}\sum_{i=1}^{N}(f(x_i)- < f >)^2. \tag{8.54}$$

Again since N is large we can replace $\tilde{\sigma}$ with σ defined by

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(f(x_i)- < f >)^2 =< f^2 > - < f >^2 . \tag{8.55}$$

$$< f >= \frac{1}{N}\sum_{i=1}^{N}f(x_i) \ , \ < f^2 >= \frac{1}{N}\sum_{i=1}^{N}f(x_i)^2. \tag{8.56}$$

The standard deviation of the mean $\tilde{\sigma}_M \sim \sigma_M$ is given in terms of the standard deviation of the sample $\tilde{\sigma} \sim \sigma$ by the equation

$$\sigma_M = \frac{\sigma}{\sqrt{N}}. \tag{8.57}$$

The proof goes as follows. We generalize equations (6.80) and (8.56) to the case of M measurements each with N samples. The total number of samples is MN. We have

$$\sigma^2 = \frac{1}{NM}\sum_{\alpha=1}^{M}\sum_{i=1}^{N}(f(x_{i,\alpha})- < f >)^2 =< f^2 > - < f >^2 . \tag{8.58}$$

$$< f >= \frac{1}{NM} \sum_{\alpha=1}^{M} \sum_{i=1}^{N} f(x_{i,\alpha}) \ , \ < f^2 >= \frac{1}{NM} \sum_{\alpha=1}^{M} \sum_{i=1}^{N} f(x_{i,\alpha})^2. \tag{8.59}$$

The standard deviation of the mean $\tilde{\sigma}_M \sim \sigma_M$ is given by

$$
\begin{aligned}
\sigma_M^2 &= \frac{1}{M} \sum_{\alpha=1}^{M} (y_\alpha - < y >)^2 \\
&= \frac{1}{M} \sum_{\alpha=1}^{M} \left(\frac{1}{N} \sum_{i=1}^{N} f(x_{i,\alpha}) - < f > \right)^2 \\
&= \frac{1}{N^2 M} \sum_{\alpha=1}^{M} \sum_{i=1}^{N} \sum_{j=1}^{N} \left(f(x_{i,\alpha}) - < f > \right) \left(f(x_{i,\alpha}) - < f > \right). \tag{8.60}
\end{aligned}
$$

In above we have used the fact that $< y >=< f >$. For every set α the sum over i and j splits into two pieces. The first is the sum over the diagonal elements with $i = j$ and the second is the sum over the off diagonal elements with $i \neq j$. Clearly $f(x_{i,\alpha}) - < f >$ and $f(x_{j,\alpha}) - < f >$ are on the average equally positive and negative and hence for large numbers M and N the off diagonal terms will cancel and we end up with

$$
\begin{aligned}
\sigma_M^2 &= \frac{1}{N^2 M} \sum_{\alpha=1}^{M} \sum_{i=1}^{N} \left(f(x_{i,\alpha}) - < f > \right)^2 \\
&= \frac{\sigma^2}{N}. \tag{8.61}
\end{aligned}
$$

The standard deviation of the mean σ_M can therefore be interpreted as the probable error in the original N measurements since if we make M sets of measurements each with N samples the standard deviation of the mean σ_M will estimate how much an average over N measurements will deviate from the exact mean.

This means in particular that the original measurement F_N of the integral F has a 68 per cent chance of being within one standard deviation σ_M of the true mean and a 95 per cent chance of being within $2\sigma_M$ and a 99.7 per cent chance of being within $3\sigma_M$. In general the proportion of data values within $\kappa\sigma_M$ standard deviations of the true mean is defined by the error function

$$\int_{<y>-\kappa\sigma_M}^{<y>+\kappa\sigma_M} \frac{1}{\sqrt{2\pi\sigma_M^2}} \exp\left(-\frac{(z- < y >)^2}{2\sigma_M^2} \right) \ dz = \frac{2}{\sqrt{\pi}} \int_0^{\frac{\kappa}{\sqrt{2}}} \exp\left(-x^2\right) \ dx = \mathrm{erf}(\frac{\kappa}{\sqrt{2}}).$$

$$\tag{8.62}$$

8.5   Nonuniform Probability Distributions

8.5.1   The Inverse Transform Method

We consider two discrete events 1 and 2 which occur with probabilities p_1 and p_2 respectively such that $p_1 + p_2 = 1$. The question is how can we choose the two events

with the correct probabilities using only a uniform probability distribution. The answer is as follows. Let r be a uniform random number between 0 and 1. We choose the event 1 if $r < p_1$ else we choose the event 2.

Let us now consider three discrete events 1, 2 and 3 with probabilities p_1, p_2 and p_3 respectively such that $p_1 + p_2 + p_3 = 1$. Again we choose a random number r between 0 and 1. If $r < p_1$ then we choose event 1, if $p_1 < r < p_1 + p_2$ we choose event 2 else we choose event 3.

We consider now n discrete events with probabilities p_i such that $\sum_{i=1}^{n} p_i = 1$. Again we choose a random number r between 0 and 1. We choose the event i if the random number r satisfies the inequality

$$\sum_{j=1}^{i-1} p_j \leq r \leq \sum_{j=1}^{i} p_j. \tag{8.63}$$

In the continuum limit we replace the probability p_i with $p(x)dx$ which is the probability that the event x is found between x and $x + dx$. The condition $\sum_{i=1}^{n} p_i = 1$ becomes

$$\int_{-\infty}^{+\infty} p(x)\ dx = 1. \tag{8.64}$$

The inequality (8.63) becomes the identity

$$P(x) \equiv \int_{-\infty}^{x} p(x^{'})\ dx^{'} = r \tag{8.65}$$

Thus r is equal to the cumulative probability distribution $P(x)$, i.e the probability of choosing a value less than or equal to x. This equation leads to the inverse transform method which allows us to generate a nonuniform probability distribution $p(x)$ from a uniform probability distribution r. Clearly we must be able to 1) perform the integral analytically to find $P(x)$ then 2) invert the relation $P(x) = r$ for x.

As a first example we consider the Poisson distribution

$$p(x) = \frac{1}{\lambda} e^{-\frac{x}{\lambda}}\ ,\ 0 \leq x \leq \infty. \tag{8.66}$$

We find

$$P(x) = 1 - e^{-\frac{x}{\lambda}} = r. \tag{8.67}$$

Hence

$$x = -\lambda \ln(1 - r). \tag{8.68}$$

Thus given the uniform random numbers r we can compute directly using the above formula the random numbers x which are distributed according to the Poisson distribution $p(x) = \frac{1}{\lambda} e^{-\frac{x}{\lambda}}$.

The next example is the Gaussian distribution in two dimensions

$$p(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}. \tag{8.69}$$

We can immediately compute that

$$\frac{1}{2\pi\sigma^2}\int_{-\infty}^{+\infty} dx \int_{-\infty}^{+\infty} dy \ e^{-\frac{x^2+y^2}{2\sigma^2}} = \int_0^1 dw \int_0^1 dv. \tag{8.70}$$

$$x = r\cos\phi \ , \ y = r\sin\phi. \tag{8.71}$$

$$r^2 = -2\sigma^2 \ln v \ , \ \phi = 2\pi w. \tag{8.72}$$

The random numbers v and w are clearly uniformly distributed between 0 and 1. The random numbers x (or y) are distributed according to the Gaussian distribution in one dimension. This method is known as the Box-Muller method.

8.5.2   The Acceptance-Rejection Method

This was proposed by Von Neumann. The goal is to generate a sequence of random numbers distributed according to some normalized probability density $y = p(x)$. This method consists of the following steps:

- We start by generating a uniform random number r_x in the range of interest $x_{\min} \leq r_x \leq x_{\max}$ where $[x_{\min}, x_{\max}]$ is the interval in which $y = p(x)$ does not vanish.

- We evaluate $p(r_x)$.

- Then we generate another uniform random number r_y in the range $[0, y_{\max}]$ where $y_{\max}$ is the maximum value of the distribution $y = p(x)$.

- If $r_y < p(r_x)$ then we accept the random number r_x else we reject it.

- We repeat this process a sufficient number of times.

It is not difficult to convince ourselves that the accepted random numbers r_x will be distributed according to $y = p(x)$.

8.6   Simulation 16: Midpoint and Monte Carlo Approximations

Part I   The volume of a ball of radius R in d dimensions is given by

$$\begin{aligned} V_d &= \int_{x_1^2+...+x_d^2 \leq R^2} dx_1...dx_d \\ &= 2\int dx_1...dx_{d-1}\sqrt{R^2 - x_1^2 - ... - x_{d-1}^2} \\ &= \frac{R^d}{d}\frac{2\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2})}. \end{aligned}$$

(1) Write a program that computes the three dimensional integral using the midpoint approximation. We take the stepsize $h = 2R/N$, the radius $R = 1$ and the number of steps in each direction to be $N = N_x = N_y = 2^p$ where $p = 1, 15$.

(2) Show that the error goes as $1/N$. Plot the logarithm of the absolute value of the absolute error versus the logarithm of N.

(3) Try out the two dimensional integral. Work in the positive quadrant and again take the stepsize $h = R/N$ where $R = 1$ and $N = 2^p$, $p = 1, 15$. We know that generically the theoretical error goes at least as $1/N^2$. What do you actually find? Why do you find a discrepancy?

   Hint: the second derivative of the integrand is singular at $x = R$ which changes the dependence from $1/N^2$ to $1/N^{1.5}$.

Part II   In order to compute numerically the volume of the ball in any dimension d we use the recursion formula

$$V_d = \frac{V_{d-1}}{R^{d-1}} \int_{-R}^{+R} dx_d \ (R^2 - x_d^2)^{\frac{d-1}{2}}.$$

(1) Find the volumes in $d = 4, 5, 6, 7, 8, 9, 10, 11$ dimensions. Compare with the exact result given above.

Part III

(1) Use the Monte Carlo sampling (hit or miss) method to find the integrals in $d = 2, 3, 4$ and $d = 10$ dimensions. Is the Monte Carlo method easier to apply than the midpoint approximation?

(2) Use the Monte Carlo sample mean value method to find the integrals in $d = 2, 3, 4$ and $d = 10$ dimensions. For every d we perform M measurements each with N samples. We consider $M = 1, 10, 100, 150$ and $N = 2^p$, $p = 10, 19$. Verify that the exact error in this case goes like $1/\sqrt{N}$.

   Hint: Compare the exact error which is known in this case with the standard deviation of the mean σ_M and with $\sigma/\sqrt{N}$ where σ is the standard deviation of the sample, i.e. of a single measurement. These three quantities must be identical.

Part IV

(1) The value of π can be given by the integral

$$\pi = \int_{x^2 + y^2 \leq R^2} dx \ dy.$$

   Use the Monte Carlo sampling (hit or miss) method to give an approximate value of π.

(2) The above integral can also be put in the form

$$\pi = 2 \int_{-1}^{+1} dx \ \sqrt{1 - x^2}.$$

   Use the Monte Carlo sample mean value method to give another approximate value of π.

8.7   Simulation 17: Nonuniform Probability Distributions

Part I   The Gaussian distribution is given by

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \, \exp -\frac{(x-\mu)^2}{2\sigma}.$$

The parameter μ is the mean and σ is the variance, i.e the square root of the standard deviation. We choose $\mu = 0$ and $\sigma = 1$.

(1) Write a program that computes a sequence of random numbers x distributed according to $P(x)$ using the inverse transform method (Box-Muller algorithm) given by the equations

$$x = r\cos\phi.$$

$$r^2 = -2\sigma^2 \ln v \ , \ \ \phi = 2\pi w.$$

The v and w are uniform random numbers in the interval $[0,1]$.

(2) Draw a histogram of the random numbers obtained in the previous question. The steps are as follows:

   a- Determine the range of the points x.

   b- We divide the interval into u bins. The lenght of each bin is $h = \text{interval}/u$. We take for example $u = 100$.

   c- We determine the location of every point x among the bins. We increase the counter of the corresponding bin by a unit.

   d- We plot the fraction of points as a function of x. The fraction of point is equal to the number of random numbers in a given bin divided by hN where N is the total number of random numbers. We take $N = 10000$.

(3) Draw the data on a logarithmic scale, i.e plot log(fraction) versus x^2. Find the fit and compare with theory.

Part II

(1) Apply the acceptance-rejection method to the above problem.

(2) Apply the Fernandez-Criado algorithm to the above problem. The procedure is as follows

   a- Start with N points x_i such that $x_i = \sigma$.

   b- Choose at random a pair (x_i, x_j) from the sequence and make the following change

$$x_i \longrightarrow \frac{x_i + x_j}{\sqrt{2}}$$

$$x_j \longrightarrow -x_i + \sqrt{2}x_j.$$

c- Repeat step 2 until we reach equilibrium. For example try it M times where $M = 10, 100, ...$

Chapter 9

The Metropolis Algorithm and The Ising Model

9.1   The Canonical Ensemble

We consider physical systems which are in thermal contact with an environment. The environment is usually much larger than the physical system of interest and as a consequence energy exchange between the two of them will not change the temperature of the environement. The environement is called heat bath or heat reservoir. When the system reaches equilibrium with the heat bath its temperature will be given by the temperature of the heat bath.

A system in equilibrium with a heat bath is described statistically by the canonical ensemble in which the temperature is fixed. In contrast an isolated system is described statistically by the microcanonical ensemble in which the energy is fixed. Most systems in nature are not isolated but are in thermal contact with the environment. It is a fundamental result of statistical mechanics that the probability of finding a system in equilibrium with a heat bath at temperature T in a microstate s with energy E_s is given by the Boltzmann distribution

$$P_s = \frac{1}{Z}e^{-\beta E_s} \ , \ \ \beta = \frac{1}{k_B T}. \tag{9.1}$$

The normalization connstant Z is the partition function. It is defined by

$$Z = \sum_s e^{-\beta E_s}. \tag{9.2}$$

The sum is over all the microstates of the system with a fixed N and V. The Helmholtz free energy F of a system is given by

$$F = -k_B T \ln Z. \tag{9.3}$$

In equilibrium the free energy is minimum. All other thermodynamical quantities can be given by various derivatives of F. For example the internal energy U of the system which

is the expectation value of the energy can be expressed in terms of F as follows

$$U = <E> = \sum_s E_s P_s = \frac{1}{Z} \sum_s E_s e^{-\beta E_s} = -\frac{\partial}{\partial \beta} \ln Z = \frac{\partial}{\partial \beta}(\beta F). \quad (9.4)$$

The specific heat is given by

$$C_v = \frac{\partial}{\partial T} U. \quad (9.5)$$

In the definition of the partition function (9.2) we have implicitly assumed that we are dealing with a physical system with configurations (microstates) which have discrete energies. This is certainly true for many quantum systems. However for many other systems especially classical ones the energies are not discrete. For example the partition function of a gas of N distinguishable classical particles is given by

$$Z = \int \prod_{i=1}^{N} \frac{d^3 p_i d^3 q_i}{h^3} \ e^{-\beta H(\vec{p}_i, \vec{q}_i)}. \quad (9.6)$$

For quantum dynamical field systems (in Euclidean spacetimes) which are of fundamental importance to elementary particles and their interactions the partition function is given by the so-called path integral which is essentially of the same form as the previous equation with the replacement of the Hamiltonian $H(\vec{p}_i, \vec{q}_i)$ by the action $S[\Phi]$ where Φ stands for the field variables and the replacement of the measure $\prod_{i=1}^{N}(d^3 p_i d^3 q_i)/h^3$ by the relevant (infinite dimensional) measure $\mathcal{D}\Phi$ on the space of field configurations. We obtain therefore

$$Z = \int \mathcal{D}\Phi \ e^{-\beta S[\Phi]}. \quad (9.7)$$

Similarly to what happens in statistical mechanics where all observables can be derived from the partition function the observables of a quantum field theory can all be derived from the path integral. The fundamental problem therefore is how to calculate the partition function or the path integral for a given physical system. Normally an analytic solution will be ideal. However finding such a solution is seldom possible and as a consequence only the numerical approach remains available to us. The partition function and the path integral are essentially given by multidimensional integrals and thus one should seek numerical approaches to the problem of integration.

9.2   Importance Sampling

In any Monte Carlo integration the numerical error is proportional to the standard deviation of the integrand and is inversely proportional to the number of samples. Thus in order to reduce the error we should either reduce the variance or increase the number of samples. The first option is preferable since it does not require any extra computer time. Importance sampling allows us to reduce the standard deviation of the integrand and hence the error by sampling more often the important regions of the integral where

the integrand is largest. Importance sampling uses also in a crucial way nonuniform probability distributions.

Let us again consider the one dimensional integral

$$F = \int_a^b dx \ f(x). \tag{9.8}$$

We introduce the probability distribution $p(x)$ such that

$$1 = \int_a^b dx \ p(x). \tag{9.9}$$

We write the integral as

$$F = \int_a^b dx \ p(x) \ \frac{f(x)}{p(x)}. \tag{9.10}$$

We evaluate this integral by sampling according to the probability distribution $p(x)$. In other words we find a set of N random numbers x_i which are distributed according to $p(x)$ and then approximate the integral by the sum

$$F_N = \frac{1}{N} \sum_{i=1}^N \frac{f(x_i)}{p(x_i)}. \tag{9.11}$$

The probability distribution $p(x)$ is chosen such that the function $f(x)/p(x)$ is slowly varying which reduces the corresponding standard deviation.

9.3 The Ising Model

We consider a $d-$dimensional periodic lattice with n points in every direction so that there are $N = n^d$ points in total in this lattice. In every point (lattice site) we put a spin variable s_i $(i = 1, ..., N)$ which can take either the value $+1$ or -1. A configuration of this system of N spins is therefore specified by a set of numbers $\{s_i\}$. In the Ising model the energy of this system of N spins in the configuration $\{s_i\}$ is given by

$$E_I\{s_i\} = -\sum_{<ij>} \epsilon_{ij} s_i s_j - H \sum_{i=1}^N s_i. \tag{9.12}$$

The parameter H is the external magnetic field. The symbol $< ij >$ stands for nearest neighbor spins. The sum over $< ij >$ extends over $\gamma N/2$ terms where γ is the number of nearest neighbors. In $2, 3, 4$ dimensions $\gamma = 4, 6, 8$. The parameter ϵ_{ij} is the interaction energy between the spins i and j. For isotropic interactions $\epsilon_{ij} = \epsilon$. For $\epsilon > 0$ we obtain ferromagnetism while for $\epsilon < 0$ we obtain antiferromagnetism. We consider only $\epsilon > 0$. The energy becomes with these simplifications given by

$$E_I\{s_i\} = -\epsilon \sum_{<ij>} s_i s_j - H \sum_{i=1}^N s_i. \tag{9.13}$$

The partition function is given by

$$Z = \sum_{s_1}\sum_{s_2}...\sum_{s_N} e^{-\beta E_I\{s_i\}}. \tag{9.14}$$

There are 2^N terms in the sum and $\beta = 1/k_B T$.

In $d = 2$ we have $N = n^2$ spins in the square lattice. The configuration $\{s_i\}$ can be viewed as an $n \times n$ matrix. We impose periodic boundary condition as follows. We consider $(n+1) \times (n+1)$ matrix where the $(n+1)$th row is identified with the first row and the $(n+1)$th column is identified with the first column. The square lattice is therefore a torus.

9.4   The Metropolis Algorithm

The internal energy $U = < E >$ can be put into the form

$$< E >= \frac{\sum_s E_s e^{-\beta E_s}}{\sum_s e^{-\beta E_s}}. \tag{9.15}$$

Generally given any physical quantity A its expectation value $< A >$ can be computed using a similar expression, viz

$$< A >= \frac{\sum_s A_s e^{-\beta E_s}}{\sum_s e^{-\beta E_s}}. \tag{9.16}$$

The number A_s is the value of A in the microstate s. In general the number of microstates N is very large. In any Monte Carlo simulation we can only generate a very small number n of the total number N of the microstates. In other words $< E >$ and $< A >$ will be approximated with

$$< E > \simeq < E >_n = \frac{\sum_{s=1}^n E_s e^{-\beta E_s}}{\sum_{s=1}^n e^{-\beta E_s}}. \tag{9.17}$$

$$< A > \simeq < A >_n = \frac{\sum_{s=1}^n A_s e^{-\beta E_s}}{\sum_{s=1}^n e^{-\beta E_s}}. \tag{9.18}$$

The calculation of $< E >_n$ and $< A >_n$ proceeds therefore by 1) choosing at random a microstate s, 2) computing E_s, A_s and $e^{-\beta E_s}$ then 3) evaluating the contribution of this microstate to the expectation values $< E >_n$ and $< A >_n$. This general Monte Carlo procedure is however highly inefficient since the microstate s is very improbable and therefore its contribution to the expectation values is negligible. We need to use importance sampling. To this end we introduce a probability distribution p_s and rewrite the expectation value $< A >$ as

$$< A >= \frac{\sum_s \frac{A_s}{p_s} e^{-\beta E_s} p_s}{\sum_s \frac{1}{p_s} e^{-\beta E_s} p_s}. \tag{9.19}$$

Now we generate the microstates s with probabilities p_s and approximate $< A >$ with $< A >_n$ given by

$$< A >_n = \frac{\sum_{s=1}^{n} \frac{A_s}{p_s} e^{-\beta E_s}}{\sum_{s=1}^{n} \frac{1}{p_s} e^{-\beta E_s}}. \qquad (9.20)$$

This is importantce sampling. The Metropolis algorithm is importance sampling with p_s given by the Boltzmann distribution, i.e.

$$p_s = \frac{e^{-\beta E_s}}{\sum_{s=1}^{n} e^{-\beta E_s}}. \qquad (9.21)$$

We get then the arithmetic average

$$< A >_n = \frac{1}{n} \sum_{s=1}^{n} A_s. \qquad (9.22)$$

The Metropolis algorithm in the case of spin systems such as the Ising model can be summarized as follows:

(1) Choose an initial microstate.

(2) Choose a spin at random and flip it.

(3) Compute $\Delta E = E_{\text{trial}} - E_{\text{old}}$. This is the change in the energy of the system due to the trial flip.

(4) Check if $\Delta E \leq 0$. In this case the trial microstate is accepted.

(5) Check if $\Delta E > 0$. In this case compute the ratio of probabilities $w = e^{-\beta \Delta E}$.

(6) Choose a uniform random number r in the inetrval $[0, 1]$.

(7) Verify if $r \leq w$. In this case the trial microstate is accepted, otherwise it is rejected.

(8) Repeat steps 2) through 7) until all spins of the system are tested. This sweep counts as one unit of Monte Carlo time.

(9) Repeat setps 2) through 8) a sufficient number of times until thermalization, i.e. equilibrium is reached.

(10) Compute the physical quantities of interest in n thermalized microstates. This can be done periodically in order to reduce correlation between the data points.

(11) Compute averages.

The proof that this algorithm leads indeed to a sequence of states which are distributed according to the Boltzmann distribution goes as follows.

It is clear that the steps 2) through 7) corresponds to a transition probability between the microstates $\{s_i\}$ and $\{s_j\}$ given by

$$W(i \longrightarrow j) = \min(1, e^{-\beta \Delta E}) , \ \ \Delta E = E_j - E_i. \qquad (9.23)$$

Since only the ratio of probabilities $w = e^{-\beta \Delta E}$ is needed it is not necessary to normalize the Boltzmann probability distribution. It is clear that this probability function satisfies the detailed balance condition

$$W(i \longrightarrow j) \ e^{-\beta E_i} = W(j \longrightarrow i) \ e^{-\beta E_j}. \tag{9.24}$$

Any other probability function W which satisfies this condition will generate a sequence of states which are distributed according to the Boltzmann distribution. This can be shown by summing over the index j in the above equation and using $\sum_j W(i \longrightarrow j) = 1$. We get

$$e^{-\beta E_i} = \sum_j W(j \longrightarrow i) \ e^{-\beta E_j}. \tag{9.25}$$

The Boltzmann distribution is an eigenvector of W. In other words W leaves the equilibrium ensemble in equilibrium. As it turns out this equation is also a sufficient condition for any ensemble to approach equilibrium.

9.5   The Heat-Bath Algorithm

The heat-bath algorithm is generally a less efficient algorithm than the Metropolis algorithm. The acceptance probability is given by

$$W(i \longrightarrow j) = \min(1, \frac{1}{1 + e^{\beta \Delta E}}) \ , \ \Delta E = E_j - E_i. \tag{9.26}$$

This acceptance probability satisfies also detailed balance for the Boltzmann probability distribution. In other words the detailed balance condition which is sufficient but not necessary for an ensemble to reach equilibrium does not have a unique solution.

9.6   The Mean Field Approximation

9.6.1   Phase Diagram and Critical Temperature

We consider $N = L^2$ spins on a square lattice where L is the number of lattice sites in each direction. Each spin can take only two possible values $s_i = +1$ (spin up) and $s_i = -1$ (spin down). Each spin interacts only with its 4 neigbhors and also with a magnetic field H. The Ising model in 2 dimensions is given by the energy

$$E\{s\} = -J \sum_{<ij>} s_i s_j - H \sum_i s_i. \tag{9.27}$$

The system is assumed to be in equilibrium with a heat bath with temperature T. Thermal equilibrium of the Ising model is described by the canonical ensemble. The probability of finding the Ising model in a configuration $\{s_1, ..., s_{2N}\}$ is given by Boltzmann distribution

$$P\{s\} = \frac{e^{-\beta E\{s\}}}{Z}. \tag{9.28}$$

The partition function is given by

$$Z = \sum_{\{s\}} e^{-\beta E\{s\}} = \sum_{s_1} \cdots \sum_{s_{2N}} e^{-\beta E\{s\}}. \tag{9.29}$$

The magnetization M in a configuration $\{s_1, ..., s_{2N}\}$ is the order parameter of the system. It is defined by

$$M = \sum_i s_i. \tag{9.30}$$

The average of M is given by

$$< M >= \sum_i < s_i >= N < s > . \tag{9.31}$$

In above $< s_i >=< s >$ since all spins are equivalent. We have

$$< M >= \frac{1}{\beta} \frac{\partial \log Z}{\partial H} = -\frac{\partial F}{\partial H}. \tag{9.32}$$

In order to compute $< M >$ we need to compute Z. In this section we use the mean field approximation. First we rewrite the energy $E\{s\}$ in the form

$$\begin{aligned} E\{s\} &= (-J \sum_{<ij>} s_j) s_i - H \sum_i s_i \\ &= \sum_i H^i_{\text{eff}} s_i - H \sum_i s_i. \end{aligned} \tag{9.33}$$

The effective magnetic field H^i_{eff} is given by

$$H^i_{\text{eff}} = -J \sum_{j(i)} s_{j(i)}. \tag{9.34}$$

The index $j(i)$ runs over the four nearest neighbors of the spin i. In the mean field approximation we replace the spins $s_{j(i)}$ by their thermal average $< s >$. We obtain

$$H^i_{\text{eff}} = -J\gamma < s > , \ \gamma = 4. \tag{9.35}$$

In other words

$$E\{s\} = -(H + J\gamma < s >) \sum_i s_i = H_{\text{eff}} \sum_i s_i \tag{9.36}$$

The partition function becomes

$$\begin{aligned} Z &= \left(\sum_{s_1} e^{-\beta H_{\text{eff}} s_i} \right)^N \\ &= \left(e^{-\beta H_{\text{eff}}} + e^{\beta H_{\text{eff}}} \right)^N \tag{9.37} \\ &= \left(2 \cosh \beta H_{\text{eff}} \right)^N. \tag{9.38} \end{aligned}$$

The free energy and magnetization are then given by

$$F = -kT \ln Z \quad = \quad -kTN \ln \left(2 \cosh \beta H_{\text{eff}} \right). \tag{9.39}$$

$$< M >= N < s >= N \tanh \beta H_{\text{eff}}. \tag{9.40}$$

Thus for zero magnetic field we get from the second equation the constraint

$$< s >= \tanh \gamma \beta J < s > . \tag{9.41}$$

Clearly $< s >= 0$ is always a solution. This is the high temperature paramagnetic phase. For small temperature we have also a solution $< s > \neq 0$. This is the ferromagnetic phase. There must exist a critical temperature T_c which separates the two phases. We expect $< s >$ to approach $< s >= 0$ as T goes to T_c from below. In other words near T_c we can treat $< s >$ as small and as a consequence we can use the expansion $\tanh x = x - \frac{1}{3}x^3$. We obtain

$$< s >= \gamma \beta J < s > -\frac{1}{3} \left(\gamma \beta J < s > \right)^3. \tag{9.42}$$

Equivalently

$$< s > \left(< s >^2 -\frac{3}{T} \frac{1}{(\gamma \beta J)^3} \left(\frac{\gamma J}{k_B} - T \right) \right) = 0. \tag{9.43}$$

We get the two solutions

$$< s >= 0 \ , \ \text{paramagnetic phase}$$
$$< s >= \pm \sqrt{\frac{3}{T} \frac{1}{(\gamma \beta J)^3} (T_c - T)^\beta} \ , \ \text{ferromagnetic phase.} \tag{9.44}$$

The critical temperature T_c and the critical exponent β are given by

$$T_c = \frac{\gamma J}{k_B} \ , \ \beta = \frac{1}{2}. \tag{9.45}$$

The ferromagnetic solution can only exist for $T < T_c$.

9.6.2 Critical Exponents

The free energy for zero magnetic field is

$$F = -kTN \ln \left(2 \cosh \gamma \beta J < s > \right). \tag{9.46}$$

We see that for $T < T_c$ the ferromagnetic solution has a lower free energy than the paramagnetic solution $< s >= 0$. The phase $T < T_c$ is indeed ferromagnetic. The transition at $T = T_c$ is second order. The free energy is continuous at $T = T_c$, i.e. there is

no latent heat while the specific heat is logarithmically divergent. The mean field theory yields the correct value 0 for the critical exponent α although it does not reproduce the logarithmic divergence. The susceptibility diverges at $T = T_c$ with critical exponent $\gamma = 1$. These latter statements can be seen as follows.

The specific heat is given by

$$
\begin{aligned}
C_v &= -\frac{\partial}{\partial T}\left(k_B T^2 \frac{\partial}{\partial T}(\beta F)\right) \\
&= -2k_B T \frac{\partial}{\partial T}(\beta F) - k_B T^2 \frac{\partial^2}{\partial T^2}(\beta F).
\end{aligned}
\tag{9.47}
$$

Next we use the expression $\beta F = -N \ln(e^x + e^{-x})$ where $x = \gamma \beta J < s >$. We find

$$
\frac{C_v}{N} = 2k_B T \tanh x \frac{\partial x}{\partial T} + k_B T^2 \tanh^2 x \frac{\partial^2 x}{\partial T^2} + k_B T^2 \frac{1}{\cosh^2 x}\left(\frac{\partial x}{\partial T}\right)^2.
\tag{9.48}
$$

We compute

$$
x = \pm\sqrt{\frac{3k_B}{\gamma J}}(T_c - T)^{\frac{1}{2}} \ , \ \frac{\partial x}{\partial T} = \mp\frac{1}{2}\sqrt{\frac{3k_B}{\gamma J}}(T_c - T)^{-\frac{1}{2}} \ , \ \frac{\partial^2 x}{\partial T^2} = \mp\frac{1}{4}\sqrt{\frac{3k_B}{\gamma J}}(T_c - T)^{-\frac{3}{2}}.
\tag{9.49}
$$

It is not difficult to show that the divergent terms cancel and as a consequence

$$
\frac{C_v}{N} \sim (T_c - T)^{-\alpha} \ , \ \alpha = 0.
\tag{9.50}
$$

The susceptibility is given by

$$
\chi = \frac{\partial}{\partial H} < M > .
\tag{9.51}
$$

To compute the behavior of χ near $T = T_c$ we consider the equation

$$
< s >= \tanh(\gamma \beta J < s > +\beta H).
\tag{9.52}
$$

For small magnetic field we can still assume that $\gamma \beta J < s > +\beta H$ is small near $T = T_c$ and as a consequence we can expand the above equation as

$$
< s >= (\gamma \beta J < s > +\beta H) - \frac{1}{3}(\gamma \beta J < s > +\beta H)^3.
\tag{9.53}
$$

Taking the derivative with respect to H of both sides of this equation we obtain

$$
\hat{\chi} = (\gamma \beta J \hat{\chi} + \beta) - (\gamma \beta J \hat{\chi} + \beta)(\gamma \beta J < s > +\beta H)^2.
\tag{9.54}
$$

$$
\hat{\chi} = \frac{\partial}{\partial H} < s > .
\tag{9.55}
$$

Setting the magnetic field to zero we get

$$
\hat{\chi} = (\gamma \beta J \hat{\chi} + \beta) - (\gamma \beta J \hat{\chi} + \beta)(\gamma \beta J < s >)^2.
\tag{9.56}
$$

In other words

$$\left(1 - \gamma\beta J + \gamma\beta J(\gamma\beta J < s >)^2\right)\hat{\chi} = \beta - \beta(\gamma\beta J < s >)^2. \tag{9.57}$$

$$2\frac{T_c - T}{T}\hat{\chi} = \frac{1}{k_B T}(1 - (\gamma\beta J < s >)^2). \tag{9.58}$$

Hence

$$\hat{\chi} = \frac{1}{2k_B}(T_c - T)^{-\gamma} \ , \ \gamma = 1. \tag{9.59}$$

9.7   Simulation of The Ising Model and Numerical Results

9.7.1   The Fortran Code

We choose to write our code in Fortran. The reason is simplicity and straightforwardness. A person who is not well versed in programming languages, who has a strong background in physics and maths, and who wants to get up and running quickly with the coding so that she starts doing physics (almost) immediately the choice of Fortran for her is ideal and thus it is only natural. The potential superior features which may be found in C are peripheral to our purposes here.

The spin found in the intersection point of the ith row and jth column of the lattice will be represented with the matrix element $\phi(i,j)$. The energy will then read (with $N = n^2$ and $n \equiv L$)

$$E = -\sum_{i,j=1}^{n} \left[\frac{J}{2}\phi(i,j)\left(\phi(i+1,j) + \phi(i-1,j) + \phi(i,j+1) + \phi(i,j-1)\right) + H\phi(i,j)\right]. \tag{9.60}$$

We impose periodic boundary condition in order to reduce edge and boundary effects. This can be done as follows. We consider $(n+1) \times (n+1)$ matrix where the $(n+1)$th row is identified with the first row and the $(n+1)$th column is identified with the first column. The square lattice is therefore a torus. The toroidal boundary condition will read explicitly as follows

$$\phi(0,j) = \phi(n,j) \ , \ \phi(n+1,j) = \phi(1,j) \ , \ \phi(i,0) = \phi(i,n) \ , \ \phi(i,n+1) = \phi(i,1).$$

The variation of the energy due to the flipping of the spin $\phi(i,j)$ is an essential ingredient in the Metropolis algorithm. This variation is explicitly given by

$$\Delta E = 2J\phi(i,j)\left(\phi(i+1,j) + \phi(i-1,j) + \phi(i,j+1) + \phi(i,j-1)\right) + 2H\phi(i,j). \tag{9.61}$$

The Fortran code contains the following pieces:

- A subroutine which generates pseudo random numbers. We prefer to work with well established suboutines such as the RAN 2 or the RANLUX.

- A subroutine which implements the Metropolis algorithm for the Ising model. This main part will read (with some change of notation such as $J = \text{exch}$)

```
do i=1,L
    ip(i)=i+1
    im(i)=i-1
enddo
    ip(L)=1
    im(1)=L

do  i=1,L
    do   j=1,L
      deltaE=2.0d0*exch*phi(i,j)*(phi(ip(i),j)+phi(im(i),j)+phi(i,ip(j))+phi(i,im(j)))
      deltaE=deltaE + 2.0d0*H*phi(i,j)
            if (deltaE.ge.0.0d0)then
                  probability=dexp(-beta*deltaE)
               call ranlux(rvec,len)
               r=rvec(1)
               if (r.le.probability)then
                   phi(i,j)=-phi(i,j)
               endif
            else
                  phi(i,j)=-phi(i,j)
            endif
    enddo
enddo
```

- We compute the energy $< E >$ and the magnetization $< M >$ of the Ising model in a separate subroutine.

- We compute the errors using for example the Jackknife method in a separate subroutine.

- We fix the parameters of the model such as L, J, $\beta = 1/T$ and H.

- We choose an initial configuration. We consider both cold and hot starts which are given respectively by

$$\phi(i,j) = +1. \tag{9.62}$$

$$\phi(i,j) = \text{random signs}. \tag{9.63}$$

- We run the Metropolis algorithm for a given thermalization time and study the history of the energy and the magnetization for different values of the temperature.

- We add a Monte Carlo evolution with a reasonably large number of steps and compute the averages of E and M.

- We compute the specific heat and the susceptibility of the system.

9.7.2   Some Numerical Results

Energy: The energy is continuous through the transition point and as a consequence there is no latent heat. This indicates a second order behavior.

Specific Heat: The critical exponent associated with the specific heat is given by $\alpha = 0$. However the specific heat diverges logarithmically at $T = T_c$. This translates into the fact that the peak grows with n logarithmically, namely

$$\frac{C_v}{n^2} \sim \log n. \tag{9.64}$$

Magnetization: The magnetization near but below the critical temperature in the two-dimensional Ising model scales as

$$\frac{<M>}{n^2} \sim (T_c - T)^{-\beta} \ , \ \ \beta = 1/8. \tag{9.65}$$

Susceptibility: The susceptibility near the critical temperature in the two-dimensional Ising model scales as

$$\frac{\chi}{n^2} \sim |T - T_c|^{-\gamma} \ , \ \ \gamma = 7/4. \tag{9.66}$$

Critical Temperature: From the behavior of the above observable we can measure the critical temperature, which marks the point where the second order ferromagnetic phase transition occurs, to be given approximately by

$$k_B T_c = \frac{2J}{\ln(\sqrt{2}+1)}. \tag{9.67}$$

Critical Exponents and $2-$Point Correlation Function: The $2-$point correlation function of the two-dimensional Ising model is defined by the expression

$$\begin{aligned} f(x) &= <s_0 s_x> \\ &= <\frac{1}{4n^2} \sum_{i,j} \phi(i,j) \Big(\phi(i+x,j) + \phi(i-x,j) + \phi(i,j+x) + \phi(i,j-x) \Big) > . \end{aligned} \tag{9.68}$$

We can verify numerically the following statements:

- At $T = T_c$ the behaviour of $f(x)$ is given by

$$f(x) \simeq \frac{1}{x^\eta} \ , \ \ \eta = 1/4. \tag{9.69}$$

- At T less than T_c the behavior of $f(x)$ is given by

$$f(x) = < M >^2 .$$

(9.70)

- At T larger than T_c the behaviour of $f(x)$ is given by

$$f(x) \simeq a \, \frac{1}{x^\eta} e^{-\frac{x}{\xi}}.$$

(9.71)

- Near T_c the correlation lenght diverges as

$$\xi \simeq \frac{1}{|T - T_c|^\nu} \; , \; \nu = 1.$$

(9.72)

Note that near-neighbor lattice sites which are a distance x away in a given direction from a given index i are given by

```
do x=1,nn
   if (i+x .le. n) then
      ipn(i,x)=i+x
   else
      ipn(i,x)=(i+x)-n
   endif
   if ((i-x).ge.1)then
      imn(i,x)=i-x
   else
      imn(i,x)=i-x+n
   endif
enddo
```

For simplicity we consider only odd lattices, viz $n = 2nn + 1$. Clearly because of the toroidal boundary conditions the possible values of the distance x are $x = 1, 2, ..., nn$.

First Order Transition and Hysteresis: We can also consider the effect of a magnetic field H on the physics of the Ising model. We observe a first order phase transition at $H = 0$ or H near 0 and a phenomena of hysteresis. We observe the following:

- For $T < T_c$ we can observe a first order phase transition. Indeed we observe a discontinuity in the energy and the magnetization which happens at a non-zero value of H due to hysteresis. The jumps in the energy and the magnetization are typical signal for a first order phase transition.

- For $T > T_c$ the magnetization becomes a smooth function of H near $H = 0$ which means that above T_c there is no distinction between the ferromagnetic states with $M \geq 0$ and $M \leq 0$.

- We recompute the magnetization as a function of H for a range of H back and fourth. We observe the following:

- A hysteresis loop.
- The hysteresis window shrinks with increasing temperature or accumulating more Monte Carlo time.
- The hysteresis effect is independent of the size of the lattice.

The phenomena of hysteresis indicates that the behaviour of the system depends on its initial state and history. Equivalently we say that the system is trapped in a metastable state.

9.8   Simulation 18: The Metropolis Algorithm and The Ising Model

Part I   We consider $N = L^2$ spins on a square lattice where L is the number of lattice sites in each direction. Each spin can take only two possible values $s_i = +1$ (spin up) and $s_i = -1$ (spin down). Each spin interacts only with its 4 neigbhors and also with a magnetic field H. The Ising model in 2 dimensions is given by the energy

$$E = -J \sum_{<ij>} s_i s_j - H \sum_i s_i.$$

We will impose toroidal boundary condition. The system is assumed to be in equilibrium with a heat bath with temperature T. Thermal fluctuations of the system will be simulated using the Metropolis algorithm.

(1) Write a subroutine that computes the energy E and the magnetization M of the Ising model in a configuration ϕ. The magnetization is the order parameter of the system. It is defined by

$$M = \sum_i s_i. \tag{9.73}$$

(2) Write a subroutine that implements the Metropolis algorithm for this system. You will need for this the variation of the energy due to flipping the spin $\phi(i, j)$.

(3) We choose $L = 10$, $H = 0$, $J = 1$, $\beta = 1/T$. We consider both a cold start and a hot start.

Run the Metropolis algorithm for a thermalization time TTH $= 2^6$ and study the history of the energy and the magnetization for different values of the temperature. The energy and magnetization should approach the values $E = 0$ and $M = 0$ when $T \longrightarrow \infty$ and the values $E = -2JN$ and $M = +1$ when $T \longrightarrow 0$.

(4) Add a Monte Carlo evolution with TTM $= 2^{10}$ and compute the averages of E and M.

(5) Compute the specific heat and the susceptibility of the system. These are defined by

$$C_v = \frac{\partial}{\partial \beta} < E > = \frac{\beta}{T}(< E^2 > - < E >^2) \ , \ \chi = \frac{\partial}{\partial H} < M > = \beta(< M^2 > - < M >^2).$$

(6) Determine the critical point. Compare with the theoretical exact result

$$k_B T_c = \frac{2J}{\ln(\sqrt{2}+1)}.$$

Part II Add to the code a separate subroutine which implements the Jackknife method for any set of data points. Compute the errors in the energy, magnetization, specific heat and susceptibility of the Ising model using the Jackknife method.

9.9   Simulation 19: The Ferromagnetic Second Order Phase Transition

Part I The critical exponent associated with the specific heat is given by $\alpha = 0$, viz

$$\frac{C_v}{L^2} \sim (T_c - T)^{-\alpha} , \ \alpha = 0.$$

However the specific heat diverges logarithmically at $T = T_c$. This translates into the fact that the peak grows with L logarithmically, namely

$$\frac{C_v}{L^2} \sim \log L.$$

Verify this behaviour numerically. To this end we take lattices between $L = 10 - 30$ with TTH $= 2^{10}$, TMC $= 2^{13}$. The temperature is taken in the range

$$T = T_c - 10^{-2} \times \text{step} , \ \text{step} = -50, 50.$$

Plot the maximum of C_v/L^2 versus $\ln L$.

Part II The magnetization near but below the critical temperature in 2D Ising model scales as

$$\frac{<M>}{L^2} \sim (T_c - T)^{-\beta} , \ \beta = \frac{1}{8}.$$

We propose to study the magnetization near T_c in order to determine the value of β numerically. Towards this end we plot $| <M> |$ versus $T_c - T$ where T is taken in the the range

$$T = T_c - 10^{-4} \times \text{step} , \ \text{step} = 0, 5000.$$

We take large lattices say $L = 30 - 50$ with TTH $=$ TMC $= 2^{10}$.

Part III The susceptibility near the critical temperature in 2D Ising model scales as

$$\frac{\chi}{L^2} \sim |T - T_c|^{-\gamma} , \ \gamma = \frac{7}{4}.$$

Determine γ numerically. Use TTH $= 2^{10}$, TMC $= 2^{13}$, $L = 50$ with the two ranges

$$T = T_c - 5 \times 10^{-4} \times \text{step} , \ \text{step} = 0, 100.$$

$$T = T_c - 0.05 - 4.5 \times 10^{-3} \text{step} , \ \text{step} = 0, 100.$$

9.10 Simulation 20: The $2-$Point Correlator

In this exercise we will continue our study of the ferromagnetic second order phase transition. In particular we will calculate the $2-$point correlator defined by the expression

$$f(n) = < s_0 s_n > = < \frac{1}{4L^2} \sum_{i,j} \phi(i,j) \Big(\phi(i+n,j) + \phi(i-n,j) + \phi(i,j+n) + \phi(i,j-n) \Big) > .$$

(1) Verify that at $T = T_c$ the behaviour of $f(n)$ is given by

$$f(n) \simeq \frac{1}{n^\eta} \ , \ \eta = \frac{1}{4}.$$

(2) Verify that at T less than T_c the behaviour of $f(n)$ is given by

$$f(n) = < M >^2 .$$

(3) Verify that at T larger than T_c the behaviour of $f(n)$ is given by

$$f(n) \simeq a \ \frac{1}{n^\eta} e^{-\frac{n}{\xi}}.$$

In all the above questions we take odd lattices say $L = 2LL + 1$ with $LL = 20 - 50$. We also consider the parameters TTH $= 2^{10}$, TTC $= 2^{13}$.

(4) Near T_c the correlation lenght diverges as

$$\xi \simeq \frac{1}{|T - T_c|^\nu} \ , \ \nu = 1.$$

In the above question we take $LL = 20$. We also consider the parameters TTH $= 2^{10}$, TTC $= 2^{15}$ and the temperatures

$$T = T_c + 0.1 \times \text{step} \ , \ \text{step} = 0, 10.$$

9.11 Simulation 21: Hysteresis and The First Order Phase Transition

In this exercise we consider the effect of the magnetic field on the physics of the Ising model. We will observe a first order phase transition at $H = 0$ or H near 0 and a phenomena of hysteresis .

(1) We will compute the magnetization and the energy as functions of H for a range of temperatures T. The initialization will be done once for all H. The thermalization will be performed once for the first value of the magnetic field H say $H = -5$. After we compute the magnetization for $H = -5$, we start slowly (adiabatically) changing the magnetic field with small steps so we do not loose the thermalization of the Ising system of spins. We try out the range $H = -5, 5$ with step equal 0.25.

a- For $T < T_c$ say $T = 0.5$ and 1.5 determine the first order transition point from the discontinuity in the energy and the magnetization. The transition should happen at a non-zero value of H due to hysteresis. The jump in the energy is associated with a non-zero latent heat. The jumps in the energy and the magnetization are the typical signal for a first order phase transition.

b- For $T > T_c$ say $T = 3$ and 5 the magnetization becomes a smooth function of H near $H = 0$ which means that above T_c there is no distinction between the ferromagnetic states with $M \geq 0$ and $M \leq 0$.

(2) We recompute the magnetization as a function of H for a range of H from -5 to 5 and back. You should observe a hysteresis loop.

a- Verify that the hysteresis window shrinks with increasing temperature or accumulating more Monte Carlo time.

b- Verify what happens if we increase the size of the lattice.

The phenomena of hysteresis indicates that the behaviour of the system depends on its initial state and history or equivalently the system is trapped in metastable states.

Part II

Monte Carlo Simulations of Matrix Field Theory

Chapter 1

Metropolis Algorithm for Yang-Mills Matrix Models

1.1 Dimensional Reduction

1.1.1 Yang-Mills Action

In a four dimensional Minkowski spacetime with metric $g_{\mu\nu} = (+1, -1, -1, -1)$, the Yang-Mills action with a topological theta term is given by

$$S = -\frac{1}{2g^2} \int d^4x \, \mathrm{Tr} F_{\mu\nu} F^{\mu\nu} - \frac{\theta}{16\pi^2} \int d^4x \, \mathrm{Tr} F_{\mu\nu} \tilde{F}^{\mu\nu}. \tag{1.1}$$

We recall the definitions

$$D_\alpha = \partial_\alpha - i[A_\alpha, ...]. \tag{1.2}$$

$$F_{\mu\nu} = \partial_\mu A_\nu - \partial_\nu A_\mu - i[A_\mu, A_\nu]. \tag{1.3}$$

$$\tilde{F}^{\mu\nu} = \frac{1}{2} \epsilon^{\mu\nu\alpha\beta} F_{\alpha\beta}. \tag{1.4}$$

The path integral of interest is

$$Z = \int DA_\mu \, \exp(iS). \tag{1.5}$$

This is invariant under the finite gauge transformations $A_\mu \longrightarrow g^{-1} A_\mu g + i g^{-1} \partial_\mu g$ with $g = e^{i\Lambda}$ in some group G (we will consider mostly $SU(N)$).

We Wick rotate to Euclidean signature as $x^0 \longrightarrow x^4 = ix^0$ and as a consequence $d^4x \longrightarrow d_E^4 x = id^4x$, $\partial_0 \longrightarrow \partial_4 = -i\partial_0$ and $A_0 \longrightarrow A_4 = -iA_0$. We compute $F_{\mu\nu} F^{\mu\nu} \longrightarrow (F_{\mu\nu}^2)_E$ and $F_{\mu\nu} \tilde{F}^{\mu\nu} \longrightarrow i(F_{\mu\nu} \tilde{F}_{\mu\nu})_E$. We get then

$$Z_E = \int DA_\mu \ \exp(-S_E). \tag{1.6}$$

$$S_E = \frac{1}{2g^2} \int (d^4x)_E \text{Tr}(F_{\mu\nu}^2)_E + \frac{i\tilde\theta}{16\pi^2} \int (d^4x)_E \text{Tr}(F_{\mu\nu}\tilde F_{\mu\nu})_E. \tag{1.7}$$

We remark that the theta term is imaginary. In the following we will drop the subscript E for simplicity. Let us consider first the $\theta = 0$ (trivial) sector. The pure Yang-Mills action is defined by

$$S_{\text{YM}} = \frac{1}{2g^2} \int d^4x \text{Tr} F_{\mu\nu}^2. \tag{1.8}$$

The path integral is of the form

$$\int DA_\mu \ \exp(-\frac{1}{2g^2} \int d^4x \text{Tr} F_{\mu\nu}^2). \tag{1.9}$$

First we find the equations of motion. We have

$$\begin{aligned}
\delta S_{\text{YM}} &= \frac{1}{g^2} \int d^4x \ \text{Tr} F_{\mu\nu}\delta F_{\mu\nu} \\
&= \frac{2}{g^2} \int d^4x \ \text{Tr} F_{\mu\nu}D_\mu\delta A_\nu \\
&= -\frac{2}{g^2} \int d^4x \ \text{Tr} D_\mu F_{\mu\nu}.\delta A_\nu + \frac{2}{g^2} \int d^4x \ \text{Tr} D_\mu(F_{\mu\nu}\delta A_\nu) \\
&= -\frac{2}{g^2} \int d^4x \ \text{Tr} D_\mu F_{\mu\nu}.\delta A_\nu + \frac{2}{g^2} \int d^4x \ \text{Tr} \partial_\mu(F_{\mu\nu}\delta A_\nu). \tag{1.10}
\end{aligned}$$

The equations of motion for variations of the gauge field which vanish at infinity are therefore given by

$$D_\mu F_{\mu\nu} = 0. \tag{1.11}$$

Equivalently

$$\partial_\mu F_{\mu\nu} - i[A_\mu, F_{\mu\nu}] = 0. \tag{1.12}$$

We can reduce to zero dimension by assuming that the configurations A_a are constant configurations, i.e. are $x-$independent. We employ the notation $A_a = X_a$. We obtain immediately the action and the equations of motion

$$S_{\text{YM}} = -\frac{V_{R^4}}{2g^2} \text{Tr}[X_\mu, X_\nu]^2. \tag{1.13}$$

$$[X_\mu, [X_\mu, X_\nu]] = 0. \tag{1.14}$$

1.1.2   Chern-Simons Action: Myers Term

Next we consider the general sector $\theta \neq 0$. First we show that the second term in the action S_E does not affect the equations of motion. In other words, the theta term is only a surface term. We define

$$\mathcal{L}_\theta = \frac{1}{16\pi^2} \mathrm{Tr} F_{\mu\nu} \tilde{F}_{\mu\nu}. \tag{1.15}$$

We compute the variation

$$
\begin{aligned}
\delta \mathcal{L}_\theta &= \frac{1}{16\pi^2} \epsilon_{\mu\nu\alpha\beta} \mathrm{Tr} F_{\mu\nu} \delta F_{\alpha\beta} \\
&= \frac{1}{8\pi^2} \epsilon_{\mu\nu\alpha\beta} \mathrm{Tr} F_{\mu\nu} D_\alpha \delta A_\beta.
\end{aligned} \tag{1.16}
$$

We use the Jacobi identity

$$
\begin{aligned}
\epsilon_{\mu\nu\alpha\beta} D_\alpha F_{\mu\nu} &= \epsilon_{\mu\nu\alpha\beta} (\partial_\alpha F_{\mu\nu} - i[A_\alpha, F_{\mu\nu}]) \\
&= -\epsilon_{\mu\nu\alpha\beta} [A_\alpha, [A_\mu, A_\nu]] \\
&= 0.
\end{aligned} \tag{1.17}
$$

Thus

$$
\begin{aligned}
\delta \mathcal{L}_\theta &= \frac{1}{8\pi^2} \epsilon_{\mu\nu\alpha\beta} \mathrm{Tr} D_\alpha (F_{\mu\nu} \delta A_\beta) \\
&= \frac{1}{8\pi^2} \epsilon_{\mu\nu\alpha\beta} \mathrm{Tr} \left(\partial_\alpha (F_{\mu\nu} \delta A_\beta) - i[A_\alpha, F_{\mu\nu} \delta A_\beta] \right) \\
&= \partial_\alpha \delta \mathcal{K}_\alpha.
\end{aligned} \tag{1.18}
$$

$$\delta \mathcal{K}_\alpha = \frac{1}{8\pi^2} \epsilon_{\alpha\mu\nu\beta} \mathrm{Tr} F_{\mu\nu} \delta A_\beta. \tag{1.19}$$

This shows explicitly that the theta term will not contribute to the equations of motion for variations of the gauge field which vanish at infinity.

In order to find the current $\mathcal{K}_\alpha$ itself we adopt the method of [1]. We consider a one-parameter family of gauge fields $A_\mu(x, \tau) = \tau A_\mu(x)$ with $0 \leq \tau \leq 1$. By using the above result we have immediately

$$
\begin{aligned}
\frac{\partial}{\partial \tau} \mathcal{K}_\alpha &= \frac{1}{8\pi^2} \epsilon_{\alpha\mu\nu\beta} \mathrm{Tr} F_{\mu\nu}(x, \tau) \frac{\partial}{\partial \tau} A_\beta \\
&= \frac{1}{8\pi^2} \epsilon_{\alpha\mu\nu\beta} \mathrm{Tr} \left(\tau \partial_\mu A_\nu - \tau \partial_\nu A_\mu - i\tau^2 [A_\mu, A_\nu] \right) . A_\beta(x).
\end{aligned} \tag{1.20}
$$

By integrating both sides with respect to τ between $\tau = 0$ and $\tau = 1$ and setting $\mathcal{K}_\alpha(x, 1) = \mathcal{K}_\alpha(x)$ and $\mathcal{K}_\alpha(x, 0) = 0$ we get

$$\mathcal{K}_\alpha = \frac{1}{8\pi^2} \epsilon_{\alpha\mu\nu\beta} \mathrm{Tr} \left(\frac{1}{2} \partial_\mu A_\nu - \frac{1}{2} \partial_\nu A_\mu - \frac{i}{3} [A_\mu, A_\nu] \right) . A_\beta(x). \tag{1.21}$$

The theta term is proportional to an integer k (known variously as the Pontryagin class, the winding number, the instanton number and the topological charge) defined by

$$
\begin{aligned}
k &= \int d^4x \mathcal{L}_\theta \\
&= \int d^4x \partial_\alpha \mathcal{K}_\alpha.
\end{aligned}
\tag{1.22}
$$

Now we imagine that the four-dimensional Euclidean spacetime is bounded by a large three-sphere S^3 in the same way that we can imagine that the plane is bounded by a large S^1, viz

$$
\partial R^4 = S^3_\infty.
\tag{1.23}
$$

Then

$$
\begin{aligned}
k &= \int_{\partial R^4 = S^3_\infty} d^3\sigma_\alpha \mathcal{K}_\alpha \\
&= \frac{1}{16\pi^2}\epsilon_{\alpha\mu\nu\beta} \int_{\partial R^4 = S^3_\infty} d^3\sigma_\alpha \mathrm{Tr}\left[F_{\mu\nu}A_\beta + i\frac{2}{3}A_\mu A_\nu A_\beta \right].
\end{aligned}
\tag{1.24}
$$

The Chern-Simons action is defined by

$$
S_{\mathrm{CS}} = i\theta k.
\tag{1.25}
$$

A Yang-Mills instanton is a solution of the equations of motion which has finite action. In order to have a finite action the field strength $F_{\mu\nu}$ must approach 0 at infinity at least as $1/x^2$, viz[1]

$$
F^I_{\mu\nu}(x) = o(1/x^2) \ , \ x \longrightarrow \infty.
\tag{1.26}
$$

We can immediately deduce that the gauge field must approach a pure gauge at infinity, viz

$$
A^I_\mu(x) = ig^{-1}\partial_\mu g + o(1/x) \ , \ x \longrightarrow \infty.
\tag{1.27}
$$

This can be checked by simple substitution in $F_{\mu\nu} = \partial_\mu A_\nu - \partial_\nu A_\mu - i[A_\mu, A_\nu]$. Now a gauge configuration $A^I_\mu(x)$ at infinity (on the sphere S^3_∞) defines a group element g which satisfies (from the above asymptotic behavior) the equation $\partial_\mu g^{-1} = iA^I_\mu g^{-1}$ or equivalently

$$
\frac{d}{ds}g^{-1}(x(s), x_0) = i\frac{dx^\mu}{ds}A^I_\mu(x(s))g^{-1}(x(s), x_0).
\tag{1.28}
$$

The solution is given by the path-ordered Wilson line

$$
g^{-1}(x, x_0) = \mathcal{P}\exp\left(i\int_0^1 ds\frac{dy^\mu}{ds}A^I_\mu(y(s)) \right).
\tag{1.29}
$$

---

[1]The requirement of finite action can be neatly satisfied if we compactify R^4 by adding one point at ∞ to obtain the four-sphere S^4.

The path is labeled by the parameter s which runs from $s = 0$ ($y = x_0$) to $s = 1$ ($y = x$) and the path-ordering operator $\mathcal{P}$ is defined such that terms with higher values of s are always put on the left in every order in the Taylor expansion of the exponential .

In the above formula for g^{-1} the points x and x_0 are both at infinity, i.e. on the sphere S^3_∞. In other words gauge configurations with finite action (the instanton configurations $A^I_\mu(x)$) define a map from S^3_∞ into G, viz

$$g^{-1} : S^3_\infty \longrightarrow G. \tag{1.30}$$

These maps are classified by homotopy theory.

As an example we take the group $G = SU(2)$. The group $SU(2)$ is topologically a three-sphere since any element $g \in SU(2)$ can be expanded (in the fundamental representation) as $g = n_4 + i\vec{n}\vec{\tau}$ and as a consequence the unitarity condition $g^+g = 1$ becomes $n_4^2 + \vec{n}^2 = 1$. In this case we have therefore maps from the three-sphere to the three-sphere, viz

$$g^{-1} : S^3_\infty \longrightarrow SU(2) = S^3. \tag{1.31}$$

These maps are characterized precisely by the integer k introduced above. This number measures how many times the second S^3 (group) is wrapped (covered) by the first sphere S^3_∞ (space). In fact this is the underlying reason why k must be quantized. In other words k is an element of the third homotopy group $\pi_3(S^3)$, viz [2]

$$k \in \pi_3(SU(2)) = \pi_3(S^3) = Z. \tag{1.32}$$

For general $SU(N)$ we consider instanton configurations obtained by embedding the $SU(2)$ instanton configurations into $SU(N)$ matrices as

$$A^{SU(N)}_\mu = \begin{pmatrix} 0 & 0 \\ 0 & A^{SU(2)}_\mu \end{pmatrix}. \tag{1.33}$$

We can obviously use any spin j representation of $SU(2)$ provided it fits inside the $N \times N$ matrices of $SU(N)$. The case $N = 2j + 1$ is equivalent to choosing the generators of $SU(2)$ in the spin j representation as the first 3 generators of $SU(N)$ and hence $A^{SU(N)a}_\mu$, $a = 1, 2, 3$ are given by the $SU(2)$ instanton configurations whereas the other components $A^{SU(N)a}_\mu$, $a = 4, ..., N^2 - 1$ are zero identically. The explicit constructions of all these instanton solutions will not be given here.

The story of instanton calculus is beautiful but long and complicated and we can only here refer the reader to the vast literature on the subject. See for example the pedagogical lectures [2].

We go back to the main issue for us which is the zero dimensional reduction of the Chern-Simons term. By using the fact that on S^3_∞ we have $F_{\mu\nu} = 0$ we can rewrite (1.24) as

$$k = \frac{i}{24\pi^2} \epsilon_{\alpha\mu\nu\beta} \int_{\partial R^4 = S^3_\infty} d^3\sigma_\alpha \text{Tr} A_\mu A_\nu A_\beta. \tag{1.34}$$

---

[2]In general $\pi_n(S^n) = Z$. It is obvious that $\pi_1(S^1) = \pi_2(S^2) = Z$.

By using also the fact that $A_\mu = A_\mu^I = ig^{-1}\partial_\mu g = iX_\mu$ on S_∞^3 we have

$$k = \frac{1}{24\pi^2}\epsilon_{\alpha\mu\nu\beta}\int_{\partial R^4=S_\infty^3} d^3\sigma_\alpha \mathrm{Tr}X_\mu X_\nu X_\beta. \tag{1.35}$$

By introducing now a local parametrization $\xi_a = \xi_a(x)$ of the G group elements we can rewrite k as (with $X_a = g^{-1}\partial_a g$)

$$k = \frac{1}{24\pi^2}\epsilon_{\alpha\mu\nu\beta}\int_{\partial R^4=S_\infty^3} d^3\sigma_\alpha \frac{\partial\xi_a}{\partial x_\mu}\frac{\partial\xi_b}{\partial x_\nu}\frac{\partial\xi_c}{\partial x_\beta}\mathrm{Tr}X_a X_b X_c. \tag{1.36}$$

Next we use

$$d^3\sigma_\alpha = \frac{1}{6}\epsilon_{\alpha\mu\nu\beta}dx_\mu \wedge dx_\nu \wedge dx_\beta. \tag{1.37}$$

$$\epsilon_{\alpha\mu\nu\beta}\epsilon_{\alpha\mu'\nu'\beta'} = \delta_{[\mu\nu\beta]}^{\mu'\nu'\beta'} = \delta_\mu^{\mu'}(\delta_\nu^{\nu'}\delta_\beta^{\beta'} - \delta_\beta^{\nu'}\delta_\nu^{\beta'}) + \delta_\nu^{\mu'}(\delta_\beta^{\nu'}\delta_\mu^{\beta'} - \delta_\mu^{\nu'}\delta_\beta^{\beta'}) + \delta_\beta^{\mu'}(\delta_\mu^{\nu'}\delta_\nu^{\beta'} - \delta_\nu^{\nu'}\delta_\mu^{\beta'}) \tag{1.38}$$

We get

$$\begin{aligned} k &= \frac{1}{24\pi^2}\frac{1}{6}\delta_{[\mu\nu\beta]}^{\mu'\nu'\beta'}\int_{\partial R^4=S_\infty^3} dx_{\mu'} \wedge dx_{\nu'} \wedge dx_{\beta'} \frac{\partial\xi_a}{\partial x_\mu}\frac{\partial\xi_b}{\partial x_\nu}\frac{\partial\xi_c}{\partial x_\beta}\mathrm{Tr}X_a X_b X_c \\ &= \frac{1}{24\pi^2}\int_{\partial R^4=S_\infty^3} d\xi_a \wedge d\xi_b \wedge d\xi_c \mathrm{Tr}X_a X_b X_c \\ &= \frac{1}{24\pi^2}\int_{\partial R^4=S_\infty^3} d^3\xi\epsilon_{abc}\mathrm{Tr}X_a X_b X_c. \end{aligned} \tag{1.39}$$

The trace Tr is generically $(2j+1)-$dimensional, and not $N-$dimensional, corresponding to the spin j representation of $SU(2)$. The Chern-Simons action becomes

$$S_{\mathrm{CS}} = \frac{i\theta}{24\pi^2}\int_{\partial R^4=S_\infty^3} d^3\xi\epsilon_{abc}\mathrm{Tr}X_a X_b X_c. \tag{1.40}$$

As before we can reduce to zero dimension by assuming that the configurations X_a are constant. We obtain immediately

$$S_{\mathrm{CS}} = \frac{i\theta V_{S^3}}{24\pi^2}\epsilon_{abc}\mathrm{Tr}X_a X_b X_c. \tag{1.41}$$

By putting (1.13) and (1.41) we obtain the matrix action

$$S_E = -\frac{V_{R^4}}{2g^2}\mathrm{Tr}[X_\mu, X_\nu]^2 + \frac{i\theta V_{S^3}}{24\pi^2}\epsilon_{abc}\mathrm{Tr}X_a X_b X_c. \tag{1.42}$$

We choose to perform the scaling

$$X_\mu \longrightarrow \left(\frac{Ng^2}{2V_{R^4}}\right)^{1/4}X_\mu. \tag{1.43}$$

The action becomes

$$S_E = -\frac{N}{4}\mathrm{Tr}[X_\mu, X_\nu]^2 + i\frac{2N\alpha}{3}\epsilon_{abc}\mathrm{Tr}X_a X_b X_c. \tag{1.44}$$

The new coupling constant α is given by

$$\alpha = \frac{1}{16\pi^2}\frac{\theta V_{S^3}}{N}\left(\frac{Ng^2}{2V_{R^4}}\right)^{3/4}. \tag{1.45}$$

1.2 Metropolis Accept/Reject Step

In the remainder we only consider the basic Yang-Mills matrix action to be of interest. This is given by

$$
\begin{aligned}
S_{\mathrm{YM}}[X] &= -\frac{N}{4} Tr[X_\mu, X_\nu]^2 \\
&= -N \sum_{\mu=1}^{d} \sum_{\nu=\mu+1}^{d} (X_\mu X_\nu X_\mu X_\nu - X_\mu^2 X_\nu^2).
\end{aligned}
\tag{1.46}
$$

The path integral or partition function of this model is given by

$$
Z = \int \prod_\mu dX_\mu \exp(-S_{\mathrm{YM}}).
\tag{1.47}
$$

The meaning of the meausre is obvious since X_μ are $N \times N$ matrices. The corresponding probability distribution for the matrix configurations X_μ is given by

$$
P(X) = \frac{1}{Z} \exp(-S_{\mathrm{YM}}[X]).
\tag{1.48}
$$

We want to sample this probability distribution in Monte Carlo using the Metropolis algorithm. Towards this end, we need to compute the variation of the action under the following arbitrary change

$$
X_\lambda \longrightarrow X_\lambda^{'} = X_\lambda + \Delta X_\lambda,
\tag{1.49}
$$

where

$$
(\Delta X_\lambda)_{nm} = d\delta_{ni}\delta_{mj} + d^*\delta_{nj}\delta_{mi}.
\tag{1.50}
$$

The corresponding variation of the action is

$$
\Delta S_{\mathrm{YM}} = \Delta S_1 + \Delta S_2.
\tag{1.51}
$$

The two pieces ΔS_1 and ΔS_2 are given respectively by

$$
\begin{aligned}
\Delta S_1 &= -N \sum_\sigma Tr[X_\sigma, [X_\lambda, X_\sigma]]\Delta X_\lambda \\
&= -Nd \sum_\sigma [X_\sigma, [X_\lambda, X_\sigma]]_{ji} - Nd^* \sum_\sigma [X_\sigma, [X_\lambda, X_\sigma]]_{ij}.
\end{aligned}
\tag{1.52}
$$

$$
\begin{aligned}
\Delta S_2 &= -\frac{N}{2} \sum_{\sigma \neq \lambda} [\Delta X_\lambda, X_\sigma]^2 \\
&= -\frac{N}{2}d \sum_{\sigma \neq \lambda} [X_\sigma, [\Delta X_\lambda, X_\sigma]]_{ji} - \frac{N}{2}d^* \sum_{\sigma \neq \lambda} [X_\sigma, [\Delta X_\lambda, X_\sigma]]_{ij} \\
&= -N \sum_{\sigma \neq \lambda} \Big[d^2 (X_\sigma)_{ji}(X_\sigma)_{ji} + (d^*)^2 (X_\sigma)_{ij}(X_\sigma)_{ij} + 2dd^*(X_\sigma)_{ii}(X_\sigma)_{jj} - dd^*\big((X_\sigma^2)_{ii} + (X_\sigma^2)_{jj}\big) \\
&\quad - \frac{1}{2}(d^2 + (d^*)^2)\big((X_\sigma^2)_{ii} + (X_\sigma^2)_{jj}\big)\delta_{ij} \Big].
\end{aligned}
\tag{1.53}
$$

The Metropolis accept/reject step is based on the probability distribution

$$P[X] = \min(1, \exp(-\Delta S_{\mathrm{YM}}). \tag{1.54}$$

It is not difficult to show that this probability distribution satisfies detailed balance, and as a consequence, this algorithm is exact, i.e. free from systematic errors.

1.3   Statistical Errors

We use the Jacknife method to estimate statistical errors. Given a set of $T = 2^P$ (with P some integer) data points $f(i)$ we proceed by removing z elements from the set in such a way that we end up with $n = T/z$ sets (or bins). The minimum number of data points we can remove is $z = 1$ and the maximum number is $z = T - 1$. The average of the elements of the ith bin is

$$< y(j) >_i = \frac{1}{T-z} \left(\sum_{j=1}^{T} f(j) - \sum_{j=1}^{z} f((i-1)z+j) \right) , \ i = 1, n. \tag{1.55}$$

For a fixed partition given by z the corresponding error is computed as follows

$$e(z) = \sqrt{\frac{n-1}{n} \sum_{i=1}^{n} (< y(j) >_i - < f >)^2} \ , \ < f > = \frac{1}{T} \sum_{j=1}^{T} f(j). \tag{1.56}$$

We start with $z = 1$ and we compute the error $e(1)$ then we go to $z = 2$ and compute the error $e(2)$. The true error is the largest value. Then we go to $z = 3$, compute $e(3)$, compare it with the previous error and again retain the largest value and so on until we reach $z = T - 1$.

1.4   Auto-Correlation Time

In any given ergodic process we obtain a sequence (Markov chain) of field/matrix configurations ϕ_1, ϕ_2,....,ϕ_T. We will assume that ϕ_i are thermalized configurations. Let f some (primary) observable with values $f_i \equiv f(\phi_i)$ in the configurations ϕ_i respectively. The average value $< f >$ of f and the statistical error δf are given by the usual formulas

$$< f > = \frac{1}{T} \sum_{i=1}^{T} f_i. \tag{1.57}$$

$$\delta f = \frac{\sigma}{\sqrt{T}}. \tag{1.58}$$

The standard deviation (the variance) is given by

$$\sigma^2 = < f^2 > - < f >^2 . \tag{1.59}$$

The above theoretical estimate of the error is valid provided the thermalized configurations ϕ_1, ϕ_2,....,ϕ_T are statistically uncorrelated, i.e. independent. In real simulations, this is certainly not the case. In general, two consecutive configurations will be dependent, and the average number of configurations which separate two really uncorrelated configurations is called the auto-correlation time. The correct estimation of the error must depend on the auto-correlation time.

We define the auto-correlation function Γ_j and the normalized auto-correlation function ρ_j for the observable f by

$$\Gamma_j = \frac{1}{T-j} \sum_{i=1}^{T-j} (f_i - <f>)(f_{i+j} - <f>). \tag{1.60}$$

$$\rho_j = \frac{\Gamma_j}{\Gamma_0}. \tag{1.61}$$

These function vanish if there is no auto-correlation. Obviously Γ_0 is the variance σ^2, viz $\Gamma_0 = \sigma^2$. In the generic case, where the auto-correlation function is not zero, the statistical error in the average $<f>$ will be given by

$$\delta f = \frac{\sigma}{\sqrt{T}} \sqrt{2\tau_{\text{int}}}. \tag{1.62}$$

The so-called integrated auto-correlation time τ_{int} is given in terms of the normalized auto-correlation function ρ_j by

$$\tau_{\text{int}} = \frac{1}{2} + \sum_{j=1}^{\infty} \rho_j. \tag{1.63}$$

The auto-correlation function Γ_j, for large j, can not be precisely determined, and hence, one must truncate the sum over j in τ_{int} at some cut-off M, in order to not increase the error $\delta\tau_{\text{int}}$ in τ_{int} by simply summing up noise. The integrated auto-correlation time τ_{int} should then be defined by

$$\tau_{\text{int}} = \frac{1}{2} + \sum_{j=1}^{M} \rho_j. \tag{1.64}$$

The value M is chosen as the first integer between 1 and T such that

$$M \geq 4\tau_{\text{int}} + 1. \tag{1.65}$$

The error $\delta\tau_{\text{int}}$ in τ_{int} is given by

$$\delta\tau_{\text{int}} = \sqrt{\frac{4M+2}{T}} \tau_{\text{int}}. \tag{1.66}$$

This formalism can be generalized to secondary observables F which are functions of n primary observables f^α, viz $F = F(f^1, f^2, ..., f^n)$. See for example [3].

In general two among the three parameters of the molecular dynamics (the time step dt, the number of iterations n and the time interval $T = ndt$) should be optimized in such a way that the acceptance rate is fixed, for example, between 70 and 90 per cent. We fix n and optimize dt along the line discussed in previous chapters. We make, for every N, a reasonable guess for the value of the number of iterations n, based on trial and error, and then work with that value throughout. For example, for N between $N = 4$ and $N = 8$, we found the value $n = 10$, to be sufficiently reasonable.

1.5   Code and Sample Calculation

Typically, we run $T_{\text{ther}} + T_{\text{meas}}$ Monte Carlo steps where thermalization is supposed to occur within the first T_{ther} steps, which are then discarded, while measurements are performed on a sample consisting of the subsequent T_{meas} configurations. We choose, for $N = 4 - 8$, $T_{\text{ther}} = 2^{11}$ and $T_{\text{meas}} = 2^{11}$. The interval from which we draw the variations d and d^* is updated after each Metropolis step by requiring that the acceptance rate is fixed between 25 and 30 per cent. We generate our random numbers using the algorithm ran2. We do not discuss auto-correlations while error bars are estimated using the jackknife method as discussed above. A FORTRAN code along these lines is included in the last chapter for illustrative purposes. This seems to go as fast as N^4.

Some thermalized results for $N = 8, 10$, for dimensions between $d = 2$ and $d = 10$, are shown on figure (1.1). The observed linear fit for the average action is in excellent agreement with the exact analytic result

$$\frac{<S>}{N^2 - 1} = \frac{d}{4}. \tag{1.67}$$

This identity follows from the invariance of the path integral under the translations $X_\mu \longrightarrow X_\mu + \epsilon X_\mu$.

Figure 1.1:

Bibliography

[1] A. M. Polyakov, "Gauge Fields and Strings," Contemp. Concepts Phys. **3**, 1 (1987).

[2] S. Vandoren and P. van Nieuwenhuizen, arXiv:0802.1862 [hep-th].

[3] S. Schaefer, "Simulations with the Hybrid Monte Carlo Algorithm: implementation and data analysis" .

Chapter 2

Hybrid Monte Carlo Algorithm for Yang-Mills Matrix Models

2.1 The Yang-Mills Matrix Action

The hybrid Monte Carlo algorithm is a combination of the molecular dynamics method and the Metropolis algorithm. In this section we will follow [1, 2] and [3–5].

We are still interested in the Euclidean Yang-Mills matrix model

$$S_{\text{YM}} = -\frac{N\gamma}{4} \sum_{\mu,\nu=1}^{d} Tr[X_\mu, X_\nu]^2 + V(X). \tag{2.1}$$

γ is some parameter, and V is some $U(N)-$invariant potential in the d matrices X_μ. In this chapter we will take a potential consisting of a harmonic oscillator term and a Chern-Simons term in the three directions X_1, X_2 and X_3 given by

$$V = \frac{1}{2}m^2 Tr X_\mu^2 + \frac{2Ni\alpha}{3}\epsilon_{abc} Tr X_a X_b X_c. \tag{2.2}$$

The path integral we wish to sample in Monte Carlo simulation is

$$Z_{\text{YM}} = \int \prod_{\mu=1}^{d} dX_\mu \ \exp(-S_{\text{YM}}[X]). \tag{2.3}$$

Firstly, we will think of the gauge configurations X_μ as evolving in some fictitious time-like parameter t, viz

$$X_\mu \equiv X_\mu(t). \tag{2.4}$$

The above path integral is then equivalent to the Hamiltonian dynamical system

$$Z_{\text{YM}} = \int \prod_{\mu} dP_\mu \prod_{\mu} dX_\mu \ \exp(-\frac{1}{2}\sum_{\mu=1}^{d} Tr P_\mu^2 - S_{\text{YM}}[X]). \tag{2.5}$$

In other words, we have introduced d Hermitian matrices P_μ which are obviously $N \times N$, and which are conjugate to X_μ. The Hamiltonian is clearly given by

$$H = \frac{1}{2} \sum_{\mu=1}^{d} Tr P_\mu^2 + S_{\text{YM}}[X]. \tag{2.6}$$

In summary, we think of the matrices X_μ as fields in one dimension with corresponding conjugate momenta P_μ. The Hamiltonian equations of motion read

$$\frac{\partial H}{\partial (P_\mu)_{ij}} = (\dot{X}_\mu)_{ij} \ , \ \frac{\partial H}{\partial (X_\mu)_{ij}} = -(\dot{P}_\mu)_{ij}. \tag{2.7}$$

We have then the equations of motion

$$(P_\mu)_{ji} = (\dot{X}_\mu)_{ij}. \tag{2.8}$$

$$\frac{\partial S_{\text{YM}}}{\partial (X_\mu)_{ij}} = -N\gamma \sum_{\nu=1}^{d} [X_\nu, [X_\mu, X_\nu]]_{ji} + \frac{\partial V}{\partial (X_\mu)_{ij}} = -(\dot{P}_\mu)_{ij}. \tag{2.9}$$

We will define

$$
\begin{aligned}
(V_\mu)_{ij}(t) &= \frac{\partial S_{\text{YM}}}{\partial (X_\mu)_{ij}(t)} \\
&= -N\gamma \sum_{\nu=1}^{d} [X_\nu, [X_\mu, X_\nu]]_{ji} + \frac{\partial V}{\partial (X_\mu)_{ij}} \\
&= -N\gamma \left(2X_\nu X_\mu X_\nu - X_\nu^2 X_\mu - X_\mu X_\nu^2 \right)_{ji} + m^2 (X_\mu)_{ji} \\
&+ 2i\alpha N [X_2, X_3]_{ji}\delta_{\mu 1} + 2i\alpha N [X_3, X_1]_{ji}\delta_{\mu 2} + 2i\alpha N [X_1, X_2]_{ji}\delta_{\mu 3}. \tag{2.10}
\end{aligned}
$$

2.2   The Leap Frog Algorithm

The first task we must face up with is to solve the above differential equations.

The numerical solution of these differential equations is formulated as follows. We consider Taylor expansions of $(X_\mu)_{ij}(t + \delta t)$ and $(P_\mu)_{ij}(t + \delta t)$ up to order δt^2 given by

$$(X_\mu)_{ij}(t + \delta t) = (X_\mu)_{ij}(t) + \delta t (\dot{X}_\mu)_{ij}(t) + \frac{\delta t^2}{2} (\ddot{X}_\mu)_{ij}(t) + \ldots \tag{2.11}$$

$$(P_\mu)_{ij}(t + \delta t) = (P_\mu)_{ij}(t) + \delta t (\dot{P}_\mu)_{ij}(t) + \frac{\delta t^2}{2} (\ddot{P}_\mu)_{ij}(t) + \ldots \tag{2.12}$$

We calculate that

$$
\begin{aligned}
(\ddot{X}_\mu)_{ij} = (\dot{P}_\mu)_{ji} &= -\frac{\partial S_{\text{YM}}}{\partial (X_\mu)_{ji}} \\
&= N \sum_{\nu=1}^{d} [X_\nu, [X_\mu, X_\nu]]_{ij} - \frac{\partial V}{\partial (X_\mu)_{ji}}. \tag{2.13}
\end{aligned}
$$

$$(\ddot{P}_\mu)_{ij} = -\sum_{kl,\nu} \frac{\partial^2 S_{\text{YM}}}{\partial(X_\nu)_{kl}\partial(X_\mu)_{ij}}(\dot{X}_\nu)_{kl}$$

$$= N\sum_{\nu=1}^{d}\left([P_\nu^T,[X_\mu,X_\nu]] + [X_\nu,[P_\mu^T,X_\nu]] + [X_\nu,[X_\mu,P_\nu^T]]\right)_{ji} - \sum_{kl,\nu}\frac{\partial^2 V}{\partial(X_\nu)_{kl}\partial(X_\mu)_{ij}}(\dot{X}_\nu)_{kl}.$$

$$(2.14)$$

For generic non-local potentials V the second equation will be approximated by

$$(\ddot{P}_\mu)_{ij} = \frac{(\dot{P}_\mu)_{ij}(t+\delta t) - (\dot{P}_\mu)_{ij}(t)}{\delta t}$$

$$= -\frac{1}{\delta t}\left(\frac{\partial S_{\text{YM}}}{\partial(X_\mu)_{ij}(t+\delta t)} - \frac{\partial S_{\text{YM}}}{\partial(X_\mu)_{ij}(t)}\right). \tag{2.15}$$

Taylor expansions of $(X_\mu)_{ij}(t+\delta t)$ and $(P_\mu)_{ij}(t+\delta t)$ become

$$(X_\mu)_{ij}(t+\delta t) = (X_\mu)_{ij}(t) + \delta t(P_\mu)_{ji}(t) - \frac{\delta t^2}{2}\frac{\partial S_{\text{YM}}}{\partial(X_\mu)_{ji}(t)} + \dots \tag{2.16}$$

$$(P_\mu)_{ij}(t+\delta t) = (P_\mu)_{ij}(t) - \frac{\delta t}{2}\frac{\partial S_{\text{YM}}}{\partial(X_\mu)_{ij}(t)} - \frac{\delta t}{2}\frac{\partial S_{\text{YM}}}{\partial(X_\mu)_{ij}(t+\delta t)} + \dots \tag{2.17}$$

We write these two equations as the three equations

$$(P_\mu)_{ij}(t+\frac{\delta t}{2}) = (P_\mu)_{ij}(t) - \frac{\delta t}{2}\frac{\partial S_{\text{YM}}}{\partial(X_\mu)_{ij}(t)}. \tag{2.18}$$

$$(X_\mu)_{ij}(t+\delta t) = (X_\mu)_{ij}(t) + \delta t(P_\mu)_{ji}(t+\frac{\delta t}{2}). \tag{2.19}$$

$$(P_\mu)_{ij}(t+\delta t) = (P_\mu)_{ij}(t+\frac{\delta t}{2}) - \frac{\delta t}{2}\frac{\partial S_{\text{YM}}}{\partial(X_\mu)_{ij}(t+\delta t)}. \tag{2.20}$$

By construction $(X_\mu)_{ij}(t+\delta t)$ and $(P_\mu)_{ij}(t+\delta t)$ solve Hamilton equations.

What we have done here is to integrate Hamilton equations of motion according to the so-called leap-frog algorithm. The main technical point to note is that the coordinates $(X_\mu)_{ij}$ at time $t+\delta t$ are computed in terms of the coordinates $(X_\mu)_{ij}$ at time t and the conjugate momenta $(P_\mu)_{ij}$ not at time t but at time $t+\delta t/2$. The conjugate momenta $(P_\mu)_{ij}$ at time $t+\delta t$ are then computed using the new coordinates $(X_\mu)_{ij}$ at time $t+\delta t$ and the conjugate momenta $(P_\mu)_{ij}$ at time $t+\delta t/2$. The conjugate momenta $(P_\mu)_{ij}$ at time $t+\delta t/2$ are computed first in terms of the coordinates $(X_\mu)_{ij}$ and the conjugate momenta $(P_\mu)_{ij}$ at time t.

We consider a lattice of points $t = n\delta t$, $n = 0,1,2,...,\nu-1,\nu$ where $(X_\mu)_{ij}(t) = (X_\mu)_{ij}(n)$ and $(P_\mu)_{ij}(t) = (P_\mu)_{ij}(n)$. The point $n = 0$ corresponds to the initial configuration $(X_\mu)_{ij}(0) = (X_\mu)_{ij}$ whereas $n = \nu$ corresponds to the final configuration $(X_\mu)_{ij}(T) = (X_\mu)'_{ij}$ where $T = \nu\delta t$. The momenta $(P_\mu)_{ij}(t)$ at the middle points $n+1/2$,

$n = 0, ..., \nu - 1$ will be denoted by $(P_\mu)_{ij}(n + 1/2)$. The above equations take then the form

$$(P_\mu)_{ij}(n + \frac{1}{2}) = (P_\mu)_{ij}(n) - \frac{\delta t}{2}(V_\mu)_{ij}(n). \tag{2.21}$$

$$(X_\mu)_{ij}(n + 1) = (X_\mu)_{ij}(n) + \delta t (P_\mu)_{ji}(n + \frac{1}{2}). \tag{2.22}$$

$$(P_\mu)_{ij}(n + 1) = (P_\mu)_{ij}(n + \frac{1}{2}) - \frac{\delta t}{2}(V_\mu)_{ij}(n + 1). \tag{2.23}$$

This algorithm applied to the solution of the equations of motion is essentially the molecular dynamics method.

2.3   Metropolis Algorithm

Along any classical trajectory we know that:

- 1) The Hamiltonian is invariant.
- 2) The motion is reversible in phase space.
- 3) The phase space volume is preserved defined by the condition

$$\frac{\partial(X(\tau), P(\tau))}{\partial(X(0), P(0))} = 1. \tag{2.24}$$

In other words detailed balance holds along a classical trajectory . The leap-frog method used to solve the above differential equations maintains only the last two properties. The violation of the first property introduces systematic errors and as a consequence detailed balance is violated. It is a well established fact that introducing a Metropolis accept/reject step at the end of each classical trajectory will eliminate the systematic error completely. The algorithm becomes therefore exact and it is known-together with the initial generation of the P's according to the Gaussian distribution-as the hybrid Monte Carlo algorithm. The hybrid algorithm is the hybrid Monte Carlo algorithm in which the Metropolis accept/reject step is omitted.

The difference between the hybrid algorithm and the ordinary molecular dynamics algorithm is that in the hybrid algorithm we refresh the momenta $(P_\mu)_{ij}(t)$ at the beginning of each molecular dynamics trajectory in such a way that they are chosen from a Gaussian ensemble. In this way we avoid the ergodicity problem.

The hybrid Monte Carlo algorithm can be summarized as follows:

- 1) Choose an initial configuration $X_\mu = X_\mu(0)$.
- 2)Choose $P_\mu = P_\mu(0)$ according to the Gaussian probability distribution $\exp(-\frac{1}{2}Tr P_\mu^2)$.
- 3)Find the configuration (X'_μ, P'_μ) by solving the above differential equations of motion, i.e. $(X'_\mu, P'_\mu) = (X_\mu(T), P_\mu(T))$.

- 4)Accept the configuration (X'_μ, P'_μ) with a probability $\min(1, e^{-\Delta H[X,P]})$ where ΔH is the change in the Hamiltonian..

- 5) Go back to step 2 and repeat.

Steps $2 - 4$ consists one sweep or one unit of Hybrid Monte Carlo time. The Metropolis accept/reject step guarantees detailed balance of this algorithm and absence of systematic errors which are caused by the non-invariance of the Hamiltonian due to the discretization.

2.4 Gaussian Distribution

We have

$$\int dP_\mu \ e^{-\frac{1}{2}TrP_\mu^2} = \int d(P_\mu)_{ii} e^{-\frac{1}{2}\sum_\mu \sum_i (P_\mu)_{ii}^2} \int d(P_\mu)_{ij} d(P_\mu)_{ij}^* \ e^{-\sum_\mu \sum_i \sum_{j=i+1} (P_\mu)_{ij}(P_\mu)_{ij}^*} \quad (2.25)$$

We are therefore interested in the probability distribution

$$\int dx \ e^{-\frac{1}{2}ax^2}, \quad (2.26)$$

where $a = 1/2$ for diagonal and $a = 1$ for off-diagonal. By squaring and including normalization we have

$$\frac{a}{\pi} \int dxdy \ e^{-\frac{1}{2}a(x^2+y^2)} = \int_0^1 dt_1 \int_0^1 dt_2. \quad (2.27)$$

$$t_1 = \frac{\phi}{2\pi} \ , \ t_2 = e^{-ar^2}. \quad (2.28)$$

We generate therefore two uniform random numbers t_1 and t_2 and write down for diagonal elements $(P_\mu)_{ii}$ the following equations

$$\phi = 2\pi t_1$$
$$r = \sqrt{-2\ln(1-t_2)}$$
$$(P_\mu)_{ii} = r\cos\phi. \quad (2.29)$$

For off-diagonal elements P_{ij} we write the following equations

$$\phi = 2\pi t_1$$
$$r = \sqrt{-\ln(1-t_2)}$$
$$(P_\mu)_{ij} = r\cos\phi + ir\sin\phi$$
$$(P_\mu)_{ji} = (P_\mu)_{ij}^*. \quad (2.30)$$

2.5 Physical Tests

The following tests can be conducted to verify the reliability of the written code based on the above algorithm:

- **Test 1:**For $\gamma = \alpha = 0$ the problem reduces to a harmonic oscillator problem. Indeed the system in this case is equivalent to $N^2 d$ independent harmonic oscillators with frequency and period given by

$$\omega = m \ , \ T = \frac{2\pi}{m}. \tag{2.31}$$

The Hamiltonian is conserved with error seen to be periodic with period

$$T_H = \frac{T}{2} = \frac{\pi}{m}. \tag{2.32}$$

- **Test 2:**In the harmonic oscillator problem we know that the X's are distributed according to the Gaussian distribution

$$\int dX_\mu \ e^{-\frac{m^2}{2} Tr X_\mu^2}. \tag{2.33}$$

The Metropolis must generate this distribution.

- **Test 3:**On general ground we must have

$$
\begin{aligned}
< e^{-\Delta H} > \ &= \ \frac{1}{Z} \int dP dX \ e^{-H[X,P]} \ e^{-\Delta H} \\
&= \ \frac{1}{Z} \int dP dX \ e^{-H[X',P']} \\
&= \ \frac{1}{Z} \int dP' dX' \ e^{-H[X',P']} \\
&= \ 1.
\end{aligned}
\tag{2.34}
$$

- **Test 4:**On general ground we must also have the Schwinger-Dyson identity (exact result) given by

$$4\gamma < \text{YM} > +3\alpha < \text{CS} > +2m^2 < \text{HO} >= d(N^2 - 1). \tag{2.35}$$

$$\text{YM} = -\frac{N}{4} \sum_{\mu,\nu=1}^{d} Tr[X_\mu, X_\nu]^2. \tag{2.36}$$

$$\text{CS} = \frac{2Ni}{3} \epsilon_{abc} Tr X_a X_b X_c. \tag{2.37}$$

$$\text{HO} = \frac{1}{2} Tr X_\mu^2. \tag{2.38}$$

- **Test 5:** We compute $< S_{\text{YM}} >$ and $C_{\text{v}} =< S_{\text{YM}}^2 > - < S_{\text{YM}} >^2$ for $\gamma = 1$ and $m = 0$. There must be an emergent geometry phase transition in α for $d = 3$ and $d = 4$.

- **Test 6:** We compute the eigenvalues distributions of the X's in $d = 3$ and $d = 4$ for $\gamma = 1$ and $\alpha = m = 0$.

- **Test 7:** The Polyakove line is defined by

$$P(k) = \frac{1}{N} Tr e^{ikX_1}. \tag{2.39}$$

We compute $< P(k) >$ as a function of k for $m = \alpha = 0$.

2.6   Emergent Geometry: An Exotic Phase Transition

As a concrete example we consider the Bosonic $d = 3$ Yang-Mills matrix model with only a Chern-Simons term, i.e. $\gamma = 1$, $\alpha \neq 0$ and $m = 0$. This model depends on a single (scaled) parameter

$$\tilde{\alpha} = \alpha\sqrt{N}. \tag{2.40}$$

The order parameter in this problem is given by the observable radius defined by

$$\text{radius} = TrX_a^2. \tag{2.41}$$

The radius of the sphere is related to this observable by

$$r = \frac{\tilde{\alpha}^2 c_2}{\text{radius}} \ , \ c_2 = \frac{N^2 - 1}{4}. \tag{2.42}$$

A more powerful set of order parameters is given by the eigenvalues distributions of the matrices X_3, $i[X_1, X_2]$, and X_a^2. Other useful observables are

$$S_3 = \text{YM} + \text{CS} \ , \ \text{YM} = -\frac{N}{4}[X_\mu, X_\nu]^2 \ , \ \text{CS} = \frac{2iN\alpha}{3}\epsilon_{abc}TrX_aX_bX_c. \tag{2.43}$$

The specific heat is

$$C_v = <S_3^2> - <S_3>^2 \ . \tag{2.44}$$

An exact Schwinger-Dyson identity is given by

$$\text{identity} = 4 < \text{YM} > + 3 < \text{CS} > \equiv dN^2. \tag{2.45}$$

For this so-called ARS model it is important that we remove the trace part of the matrices X_a after each molecular dynamics step because this mode can never be thermalized. In other words, we should consider in this case the path integral (partition function) given by

$$Z = \int dX_a \ \exp(-S_3)\delta(TrX_a). \tag{2.46}$$

The corresponding hybrid Monte Carlo code is included in the last chapter. We skip here any further technical details and report only few physical results.

The ARS model is characterized by two phases: the fuzzy sphere phase and the Yang-Mills phase. Some of the fundamental results are:

1. **The Fuzzy Sphere Phase:**

   - This appears for large values of $\tilde{\alpha}$. It corresponds to the class of solutions of the equations of motion given by

$$[X_a, X_b] = i\alpha\phi\epsilon_{abc}X_c \ , \ \phi = 1. \tag{2.47}$$

The global minimum is given by the largest irreducible representation of $SU(2)$ which fits in $N \times N$ matrices. This corresponds to the spin $l = (N - 1)/2$ irreducible representation, viz

$$X_a = \phi \alpha L_a. \tag{2.48}$$

$$[L_a, L_b] = i\epsilon_{abc} L_c \ , \ c_2 = \sum_a L_a^2 = l(l + 1).\mathbf{1}_N = \frac{N^2 - 1}{4}.\mathbf{1}_N. \tag{2.49}$$

The values of the various observables in these configurations are

$$S_3 = \phi^3 \tilde{\alpha}^4 c_2 (\frac{\phi}{2} - \frac{2}{3}) \ , \ \mathrm{YM} = \frac{\phi^4 \tilde{\alpha}^4 c_2}{2} \ , \ \mathrm{CS} = -\frac{2\phi^3 \tilde{\alpha}^4 c_2}{3} \ , \ \mathrm{radius} = \phi^2 \tilde{\alpha}^2 c_2 \tag{2.50}$$

- The eigenvalues of $D_3 = X_3/\alpha$ and $i[D_1, D_2] = i[X_1, X_2]/\alpha^2$ are given by

$$\lambda_i = -\frac{N - 1}{2}, ..., +\frac{N - 1}{2}. \tag{2.51}$$

The spectrum of $[D_1, D_2]$ is a better measurement of the geometry since all fluctuations around L_3 are more suppressed. Some illustrative data for $\tilde{\alpha} = 3$ and $N = 4$ is shown on figure (2.1).

2. **The Yang-Mills (Matrix) Phase:**

- This appears for small values of $\tilde{\alpha}$. It corresponds to the class of solutions of the equations of motion given by

$$[X_a, X_b] = 0. \tag{2.52}$$

This is the phase of almost commuting matrices. It is characterized by the eigenvalues distribution

$$\rho(\lambda) = \frac{3}{4R^3}(R^2 - \lambda^2). \tag{2.53}$$

It is believed that $R = 2$. We compute

$$
\begin{aligned}
< \mathrm{radius} > &= 3 < TrX_3^2 > \\
&= 3N \int_{-R}^{R} d\lambda \rho(\lambda)\lambda^2 \\
&= \frac{3}{5}R^2 N.
\end{aligned} \tag{2.54}
$$

- The above eigenvalues distribution can be derived by assuming that the joint eigenvalues distribution of the the three commuting matrices X_1, X_2 and X_3 is uniform inside a solid ball of radius R. This can be actually proven by quantizing the system in the Yang-Mills phase around commuting matrices [6].
- The value of the radius R is determined numerically as follows:
  - The first measurement R_1 is obtained by comparing the numerical result for $< \mathrm{radius} >$, for the biggest value of N, with the formula (2.54).

- We use R_1 to restrict the range of the eigenvalues of X_3.
- We fit the numerical result for the density of eigenvalues of X_3, for the biggest value of N, to the parabola (2.53) in order to get a second measurement R_2.
- We may take the average of R_1 and R_2.

Example: For $\alpha = 0$, we find the values $R_1 = 2.34(N = 6)$, $R_1 = 2.15(N = 8)$, $R_1 = 2.08(N = 10)$, and $R_2 = 2.05 \pm 0.01(N = 10)$. Sample data for $\tilde{\alpha} = 0$ with $N = 6, 8$ and 10 is shown on figure (2.2).

- It is found that the eigenvalues distribution, in the Yang-Mills phase, is independent of $\tilde{\alpha}$. Sample data for $\tilde{\alpha} = 0 - 2$ and $N = 10$ is shown on figure (2.3).

3. **Critical Fluctuations:** The transition between the two phases occur at $\tilde{\alpha} = 2.1$. The specific heat diverges at this point from the Yang-Mills side while it remains constant from the fuzzy sphere side. This indicates a second order behaviour with critical fluctuations only from one side of the transition. The Yang-Mills and Chern-Simons actions, and as a consequence the total action, as well as the radii radius and r suffer a discontinuity at this point reminiscent of a first order behavior. The different phases of the model are characterized by

fuzzy sphere ($\tilde{\alpha} > \tilde{\alpha}_*$)	matrix phase ($\tilde{\alpha} << \tilde{\alpha}_*$)
$r = 1$	$r = 0$
$C_v = 1$	$C_v = 0.75$

The Monte Carlo results of [7], derived using the Metropolis algorithm of the previous chapter and shown on figure (2.4), should be easily obtainable using the attached hybrid Monte Carlo code.

Figure 2.1:

Figure 2.2:

Figure 2.3:

Figure 2.4:

Bibliography

[1] I. Montvay and G. Munster, "Quantum fields on a lattice," Cambridge, UK: Univ. Pr. (1994), 491 p, Cambridge monographs on mathematical physics.

[2] H. J. Rothe, "Lattice gauge theories: An Introduction," World Sci. Lect. Notes Phys. **74**, 1 (2005).

[3] J. Ambjorn, K. N. Anagnostopoulos, W. Bietenholz, T. Hotta and J. Nishimura, "Large N dynamics of dimensionally reduced 4D SU(N) super Yang-Mills theory," JHEP **0007**, 013 (2000) [arXiv:hep-th/0003208].

[4] J. Ambjorn, K. N. Anagnostopoulos, W. Bietenholz, T. Hotta and J. Nishimura, "Monte Carlo studies of the IIB matrix model at large N," JHEP **0007**, 011 (2000) [arXiv:hep-th/0005147].

[5] K. N. Anagnostopoulos, T. Azuma, K. Nagao and J. Nishimura, "Impact of supersymmetry on the nonperturbative dynamics of fuzzy spheres," JHEP **0509**, 046 (2005) [arXiv:hep-th/0506062].

[6] V. G. Filev and D. O'Connor, "On the Phase Structure of Commuting Matrix Models," arXiv:1402.2476 [hep-th].

[7] R. Delgadillo-Blando, D. O'Connor and B. Ydri, "Geometry in transition: A model of emergent geometry," Phys. Rev. Lett. **100**, 201601 (2008) [arXiv:0712.3011 [hep-th]].

Chapter 3

Hybrid Monte Carlo Algorithm for Noncommutative Phi-Four

3.1 The Matrix Scalar Action

The hybrid Monte Carlo algorithm is a combination of the molecular dynamics method and the Metropolis algorithm. In this section we will apply this algorithm to matrix Φ^4 on the fuzzy sphere. This problem was studied using other techniques in [1–4]. We will follow here [5, 6].

We are interested in the Euclidean matrix model

$$S = \text{Tr}\left(-a[L_a, \Phi]^2 + b\Phi^2 + c\Phi^4\right). \tag{3.1}$$

The scaled (collapsed) parameters are given by

$$\tilde{b} = \frac{b}{aN^{\frac{3}{2}}} \; , \; \tilde{c} = \frac{c}{a^2 N^2}. \tag{3.2}$$

The path integral we wish to sample in Monte Carlo simulation is

$$Z = \int d\Phi \; \exp(-S[\Phi]). \tag{3.3}$$

As before, we will first think of the configurations Φ as evolving in some fictitious time-like parameter t, viz

$$\Phi \equiv \Phi(t). \tag{3.4}$$

The above path integral is then equivalent to the Hamiltonian dynamical system

$$Z = \int dP d\Phi \; \exp(-\frac{1}{2}TrP^2 - S[\Phi]). \tag{3.5}$$

In other words, we have introduced a Hermitian $N \times N$ matrix P which is conjugate to Φ. The Hamiltonian is clearly given by

$$H = \frac{1}{2}TrP^2 + S[\Phi]. \tag{3.6}$$

In summary, we think of the matrix Φ as a field in one dimension with corresponding conjugate momentum P. The Hamiltonian equations of motion read

$$\frac{\partial H}{\partial P_{ij}} = (\dot{\Phi})_{ij} = P_{ji} \ , \ \frac{\partial H}{\partial \Phi_{ij}} = -(\dot{P})_{ij} = \frac{\partial S}{\partial \Phi_{ij}}. \tag{3.7}$$

We will define the scalar force by

$$\begin{aligned} V_{ij}(t) &= \frac{\partial S}{\partial \Phi_{ij}(t)} \\ &= a\Big(- 4L_a\Phi L_a + 2L_a^2\Phi + 2\Phi L_a^2 \Big)_{ji} + 2b\Phi_{ji} + 4c(\Phi^3)_{ji}. \end{aligned} \tag{3.8}$$

3.2 The Leap Frog Algorithm

The numerical solution of the above differential equations can be given by the leap frog equations

$$(P)_{ij}(t + \frac{\delta t}{2}) = (P)_{ij}(t) - \frac{\delta t}{2}V_{ij}(t). \tag{3.9}$$

$$\Phi_{ij}(t + \delta t) = \Phi_{ij}(t) + \delta t P_{ji}(t + \frac{\delta t}{2}). \tag{3.10}$$

$$P_{ij}(t + \delta t) = P_{ij}(t + \frac{\delta t}{2}) - \frac{\delta t}{2}V_{ij}(t + \delta t). \tag{3.11}$$

Let us recall that $t = n\delta t$, $n = 0, 1, 2, ..., \nu - 1, \nu$ where the point $n = 0$ corresponds to the initial configuration $\Phi_{ij}(0)$ whereas $n = \nu$ corresponds to the final configuration $\Phi_{ij}(T)$ where $T = \nu\delta t$.

3.3 Hybrid Monte Carlo Algorithm

The hybrid Monte Carlo algorithm can be summarized as follows:

- 1) Choose $P(0)$ such that $P(0)$ is distributed according to the Gaussian probability distribution $\exp(-\frac{1}{2}TrP^2)$.

- 2)Find the configuration $(\Phi(T), P(T))$ by solving the above differential equations of motion.

- 3)Accept the configuration $(\Phi(T), P(T))$ with a probability

$$\min(1, e^{-\Delta H[\Phi, P]}), \tag{3.12}$$

  where ΔH is the corresponding change in the Hamiltonian when we go from $(\Phi(0), P(0))$ to $(\Phi(T), P(T))$.

- 4) Repeat.

3.4   Optimization

3.4.1   Partial Optimization

We start with some general comment which is not necessarily a part of the optimization process. The scalar field Φ is a hermitian matrix, i.e. the diagonal elements are real, while the off diagonal elements are complex conjugate of each other. We find it crucial that we implement, explicitly in the code, the reality of the diagonal elements by subtracting from Φ_{ii} the imaginary part (error) which in each molecular dynamics iteration is small but can accumulate. The implementation of the other condition is straightforward.

In actual simulations we can fix ν, for example we take $\nu = 20$, and adjust the step size δt, in some interval $[\delta t_{\min}, \delta t_{\max}]$, in such a way that the acceptance rate pa is held fixed between some target acceptance rates say $\text{pa}_{\text{low}} = 70$ and $\text{pa}_{\text{high}} = 90$ per cents. If the acceptance rate becomes larger than the target acceptance rate pa_{high}, then we increase the step size δt by a factor inc $= 1.2$ if the outcome is within the interval $[\delta t_{\min}, \delta t_{\max}]$. Similarly, if the acceptance rate becomes smaller than the target acceptance rate pa_{low}, we decrease the step size by a factor dec $= 0.8$ if the outcome is within the interval $[\delta t_{\min}, \delta t_{\max}]$. The adjusting of δt can be done at each Monte Carlo step, but it can also be performed only each L simulations. We take $L = 1$. A sample pseudo code is attached below. A sample of the results is shown in figure (3.1).

```
pa=(Accept)/(Rejec+Accept)
cou=mod(tmc,L)
  if (cou.eq.0)then
    if (pa.ge.target_pa_high) then
       dtnew=dt*inc
         if (dtnew.le.dt_max)then
            dt=dtnew
         else
         dt=dt_max
         endif
    endif
    if (pa.le.target_pa_low) then
       dtnew=dt*dec
         if (dtnew.ge.dt_min)then
            dt=dtnew
         else
            dt=dt_min
         endif
    endif
  endif
```

Figure 3.1:

3.4.2   Full Optimization

A more thourough optimization of the algorithm can also be done as follows [1–3]. We take $\delta\tau$ small so that the acceptance rate pa is kept sufficiently large. Then we fix ν and look for the value of $\delta \times \tau$ where the speed of motion in the phase space defined by $\delta\tau \times$ pa is maximum. Then we fix $\delta\tau$ at its optimal value and look for the value of ν where the autocorrelation time T_{au} is minimum. The number of iterations ν must also be kept relatively small so that the systematic error (which is of order $\nu \times \delta\tau^2$ for every hybrid Monte Carlo unit of time) is kept small. Clearly a small value of ν is better for the effeciency of the algorithm.

3.5   The Non-Uniform Order: Another Exotic Phase

3.5.1   Phase Structure

The theory (3.1) is a three-parameter model with the following three known phases:

- The usual 2nd order Ising phase transition between disordered $< \Phi >= 0$ and uniform ordered $< \Phi >\sim \mathbf{1}$ phases. This appears for small values of c. This is the only transition observed in commutative phi-four.

- A matrix transition between disordered $< \Phi >= 0$ and non-uniform ordered $< \Phi >\sim \gamma$ phases with $\gamma^2 = \mathbf{1}$. This transition coincides, for very large values of c, with the 3rd order transition of the real quartic matrix model, i.e. the model with $a = 0$, which occurs at $b = -2\sqrt{N}c$. See next chapter.

- A transition between uniform ordered $< \Phi >\sim \mathbf{1}$ and non-uniform ordered $< \Phi >\sim \gamma$ phases. The non-uniform phase, in which translational/rotational invariance is spontaneously broken, is absent in the commutative theory. The non-uniform phase is essentially the stripe phase observed originally on Moyal-Weyl spaces in [7, 8].

The above three phases are already present in the pure potential model $V = \mathrm{Tr}(b\Phi^2 + c\Phi^4)$. The ground state configurations are given by the matrices

$$\Phi_0 = 0. \tag{3.13}$$

$$\Phi_\gamma = \sqrt{-\frac{b}{2c}}U\gamma U^+ \ , \ \gamma^2 = \mathbf{1}_N \ , \ UU^+ = U^+U = \mathbf{1}_N. \tag{3.14}$$

We compute $V[\Phi_0] = 0$ and $V[\Phi_\gamma] = -b^2/4c$. The first configuration corresponds to the disordered phase characterized by $< \Phi >= 0$. The second solution makes sense only for $b < 0$, and it corresponds to the ordered phase characterized by $< \Phi >\neq 0$. As mentioned above, there is a non-perturbative transition between the two phases which occurs quantum mechanically, not at $b = 0$, but at $b = b_* = -2\sqrt{N}c$, which is known as the one-cut to two-cut transition. The idempotent γ can always be chosen such that $\gamma = \gamma_k = \mathrm{diag}(\mathbf{1}_k, -\mathbf{1}_{N-k})$. The orbit of γ_k is the Grassmannian manifold $U(N)/(U(k) \times U(N-k))$ which is d_k−dimensional where $d_k = 2kN - 2k^2$. It is not difficult to show that this

dimension is maximum at $k = N/2$, assuming that N is even, and hence from entropy argument, the most important two-cut solution is the so-called stripe configuration given by $\gamma = \text{diag}(\mathbf{1}_{N/2}, -\mathbf{1}_{N/2})$.

In this real quartic matrix model, we have therefore three possible phases characterized by the following order parameters:

$$< \Phi >= 0 \quad \text{disordered phase.} \tag{3.15}$$

$$< \Phi >= \pm\sqrt{-\frac{b}{2c}}\mathbf{1}_N \quad \text{Ising (uniform) phase.} \tag{3.16}$$

$$< \Phi >= \pm\sqrt{-\frac{b}{2c}}\gamma \quad \text{matrix (nonuniform or stripe) phase.} \tag{3.17}$$

However, as one can explicitly check by calculating the free energies of the respective phases, the uniform ordered phase is not stable in the real quartic matrix model $V = \text{Tr}(b\Phi^2 + c\Phi^4)$.

The above picture is expected to hold for noncommutative/fuzzy phi-four theory in any dimension, and the three phases are all stable and are expected to meet at a triple point. This structure was confirmed in two dimensions by means of Monte Carlo simulations on the fuzzy sphere in [1, 2].

3.5.2   Sample Simulations

We run simulations for every N by running T_{th} thermalization steps, and then measuring observables in a sample containing T_{mc} thermalized configurations Φ, where each two successive configurations are separated by T_{co} Monte Carlo steps in order to reduce auto-correlation effects. Most of the detail of the simulations have already been explained. We only mention again that we estimate error bars using the jackknife method and use the random number generator ran2. A sample code is attached in the last chapter.

We measure the action $< S >$, the specific heat C_v, the magnetization m and the associated susceptibility χ, the total power P_T, and the power in the zero modes P_0 defined respectively by

$$C_v =< S^2 > - < S >^2 . \tag{3.18}$$

$$m =< |Tr\Phi| > . \tag{3.19}$$

$$\chi =< |Tr\Phi|^2 > - < |Tr\Phi| >^2 . \tag{3.20}$$

$$P_T = \frac{1}{N}Tr\Phi^2. \tag{3.21}$$

$$P_0 = \frac{1}{N^2}(Tr\Phi)^2. \tag{3.22}$$

We will also compute the eigenvalues of the matrix Φ by calling the library LAPACK and then construct appropriate histograms using known techniques.

Ising: The Ising transition appears for small values of $\tilde{c}$ and is the easiest one to observe in Monte Carlo simulations. We choose, for $N = 8$, the Monte Carlo times $T_{\text{th}} = 2^{11}$, $T_{\text{mc}} = 2^{11}$ and $T_{\text{co}} = 2^0$, i.e. we ignore to take into account auto-correlations for simplicity. The data for $\tilde{c} = 0.1, 0.2$ is shown on figure (3.2). The transition, marked by the peak of the susceptibility, occurs, for $\tilde{c} = 0.1, 0.2, 0.3$ and 0.4, at $\tilde{b} = -0.5, -0.9, -1.4$ and -1.75 respectively. The corresponding linear fit which goes through the origin is given by

$$\tilde{c} = -0.22\tilde{b}_*. \tag{3.23}$$

Matrix: The disorder-to-non-uniform phase transition appears for large values of $\tilde{c}$ and is quite difficult to observe in Monte Carlo simulations due to the fact that configurations, which have slightly different numbers of pluses and minuses, strongly competes for finite N, with the physically relevant stripe configuration with an equal numbers of pluses and minuses. In principle then we should run the simulation until a symmetric eigenvalues distribution is reached which can be very difficult to achieve in practice. We choose, for $N = 8$, the Monte Carlo times $T_{\text{th}} = 2^{11}$, $T_{\text{mc}} = 2^{12}$ and $T_{\text{co}} = 2^4$. The data for the specific heat for $\tilde{c} = 1 - 4$ is shown on figure (3.3). We also plot the data for the pure quartic matrix model for $\tilde{c} = 1$ for comparison. The transition for smaller value of $\tilde{c}$ is marked, as before, by the peak in specific heat. However, this method becomes unreliable for larger values of $\tilde{c}$ since the peak disappears. Fortunately, the transition is always marked by the point where the eigenvalues distribution splits at $\lambda = 0$. The corresponding eigenvalues distributions are shown on (3.4). We include symmetric and slightly non-symmetric distributions since both were taken into account in the data of the specific heat. The non-symmetric distributions cause typically large fluctuations of the magnetization and peaks in the susceptibility which are very undesirable finite size effects. But, on the other hand, as we increase the value of $|\tilde{b}|$ we are approaching the non-symmetric uniform phase and thus the appearance of these non-symmetric distributions is very natural. This makes the determinantion of the transition point very hard from the behavior of these observables.

We have determined instead the transition point by simulating, for a given $\tilde{c}$, the pure matrix model with $a = 0$, in which we know that the transition occurs at $\tilde{b}_* = -2\sqrt{\tilde{c}}$, and then searching in the full model with $a = 1$ for the value of $\tilde{b}$ with an eigenvalues distribution similar to the eigenvalues distribution found for $a = 0$ and $\tilde{b}_* = -2\sqrt{\tilde{c}}$. This exercise is repeated for $\tilde{c} = 4, 3, 2$ and 1 and we found the transition points given respectively by $\tilde{b}_* = -5, -4.5, -4$, and -2.75. See graphs on figure (3.5). The corresponding linear fit is given by

$$\tilde{c} = -1.3\tilde{b}_* - 2.77. \tag{3.24}$$

Two more observations concerning this transition are in order:

- The eigenvalues distribution for the pure matrix model with $a = 0$ is such that it depends only on a single parameter given by $g = 4Nc/b^2$. See next chapter for more detail. From the Monte Carlo data the same statement seems to hold in the full model with $a = 1$ along the disorder-to-non-uniform boundary. See last graph on figure (3.5).

- The disorder-to-non-uniform transition line seems to be better approximated by a shift of the result $\tilde{b}_* = -2\sqrt{\tilde{c}}$ by a single unit in the $-\tilde{b}$ direction. This is roughly in accord with the analytic result for the critical point found in [9] for the multitrace approximation (see next chapter) which is given, for $a = 1$, by

$$\tilde{b}_* = -\frac{\sqrt{N}}{2} - 2\sqrt{\tilde{c}} + \frac{N}{6\sqrt{\tilde{c}}}. \tag{3.25}$$

Stripe: The uniform-to-non-uniform phase transition is even more difficult to observe in Monte Carlo simulations but it is expected, according to [1,2], to only be a continuation of the disorder-to-uniform transition line (3.23). The intersection point between the above two fits (3.23) and (3.24) is therefore an estimation of the triple point. This is given by

$$(\tilde{c}, \tilde{b}) = (0.56, -2.57). \tag{3.26}$$

However, this is not really what we observe using our code here. The uniform-to-non-uniform phase transition is only observed for small values of $\tilde{c}$ from the uniform phase to the non-uniform phase as we increase $-\tilde{b}$. The transition for these small values of $\tilde{c}$, such as $\tilde{c} = 0.1, 0.2, 0.3, 0.4$, corresponds to a second peak in the susceptibility and the specific heat. It corresponds to a transition from a one-cut eigenvalues distribution symmetric around 0 to a one-cut eigenvalues distribution symmetric around a non-zero value. The eigenvalues distributions for $\tilde{c} = 0.3$ are shown on the first two graphs of figure (3.7). In this case we have found it much easier to determine the transition points from the behavior of the magnetization and the powers. In particular, we have determined the transition point from the broad maximum of the magnetization which corresponds to the discontinuity of the power in the zero modes. The magnetization and the powers, for $\tilde{c} = 0.1, 0.2, 0.3, 0.4$, are shown on figure (3.8). The transition points were found to be -1.5, -1.7, -2 and -2.1 respectively.

The uniform phase becomes narrower as we approach the value $\tilde{c} = 0.5$. The specific heat and the susceptibility have a peak around $\tilde{b} = -2.25$ which is consistent with the Ising transition but the powers and the magnetization show the behavior of the disorder-to-non-uniform-order transition. The eigenvalues distribution is also consistent with the disorder-to-non-uniform-order transition. See last graph of figure (3.7). The value $\tilde{c} = 0.5$ is roughly the location of the triple point.

The phase diagram is shown on figure (3.6).

Figure 3.2:

Figure 3.3:

Figure 3.4:

Figure 3.5:

Figure 3.6:

Figure 3.7:

Figure 3.8:

Bibliography

[1] F. Garcia Flores, X. Martin and D. O'Connor, "Simulation of a scalar field on a fuzzy sphere," Int. J. Mod. Phys. A **24**, 3917 (2009) [arXiv:0903.1986 [hep-lat]].

[2] F. Garcia Flores, D. O'Connor and X. Martin, "Simulating the scalar field on the fuzzy sphere," PoS LAT **2005**, 262 (2006) [hep-lat/0601012].

[3] X. Martin, "A matrix phase for the phi**4 scalar field on the fuzzy sphere," JHEP **0404**, 077 (2004) [hep-th/0402230].

[4] M. Panero, "Numerical simulations of a non-commutative theory: The Scalar model on the fuzzy sphere," JHEP **0705**, 082 (2007) [hep-th/0608202].

[5] J. Ambjorn, K. N. Anagnostopoulos, W. Bietenholz, T. Hotta and J. Nishimura, "Large N dynamics of dimensionally reduced 4D SU(N) super Yang-Mills theory," JHEP **0007**, 013 (2000) [arXiv:hep-th/0003208].

[6] J. Ambjorn, K. N. Anagnostopoulos, W. Bietenholz, T. Hotta and J. Nishimura, "Monte Carlo studies of the IIB matrix model at large N," JHEP **0007**, 011 (2000) [arXiv:hep-th/0005147].

[7] S. S. Gubser and S. L. Sondhi, "Phase structure of noncommutative scalar field theories," Nucl. Phys. B **605**, 395 (2001) [hep-th/0006119].

[8] J. Ambjorn and S. Catterall, "Stripes from (noncommutative) stars," Phys. Lett. B **549**, 253 (2002) [hep-lat/0209106].

[9] B. Ydri, "A Multitrace Approach to Noncommutative Φ_2^4," arXiv:1410.4881 [hep-th].

Chapter 4

Lattice HMC Simulations of Φ_2^4: A Lattice Example

References for this chapter include the elegant quantum field theory textbook [1] and the original articles [2–4].

4.1 Model and Phase Structure

The Euclidean ϕ^4 action with $O(N)$ symmetry is given by

$$S[\phi] = \int d^d x \left(\frac{1}{2}(\partial_\mu \phi^i)^2 + \frac{1}{2}m^2 \phi^i \phi^i + \frac{\lambda}{4}(\phi^i \phi^i)^2 \right). \tag{4.1}$$

We will employ lattice regularization in which $x = an$, $\int d^d x = a^d \sum_n$, $\phi^i(x) = \phi_n^i$ and $\partial_\mu \phi^i = (\phi_{n+\hat{\mu}}^i - \phi_n^i)/a$. The lattice action reads

$$S[\phi] = \sum_n \left(-2\kappa \sum_\mu \Phi_n^i \Phi_{n+\hat{\mu}}^i + \Phi_n^i \Phi_n^i + g(\Phi_n^i \Phi_n^i - 1)^2 \right). \tag{4.2}$$

The mass parameter m^2 is replaced by the so-called hopping parameter κ and the coupling constant λ is replaced by the coupling constant g where

$$m^2 a^2 = \frac{1-2g}{\kappa} - 2d \ , \ \frac{\lambda}{a^{d-4}} = \frac{g}{\kappa^2}. \tag{4.3}$$

The fields ϕ_n^i and Φ_n^i are related by

$$\phi_n^i = \sqrt{\frac{2\kappa}{a^{d-2}}} \Phi_n^i. \tag{4.4}$$

The partition function is given by

$$Z = \int \prod_{n,i} d\Phi_n^i \ e^{-S[\phi]}$$

$$= \int d\mu(\Phi) \ e^{2\kappa \sum_n \sum_\mu \Phi_n^i \Phi_{n+\hat{\mu}}^i}. \tag{4.5}$$

The measure $d\mu(\phi)$ is given by

$$
\begin{aligned}
d\mu(\Phi) &= \prod_{n,i} d\Phi_n^i \; e^{-\sum_n \left(\Phi_n^i \Phi_n^i + g(\Phi_n^i \Phi_n^i - 1)^2\right)} \\
&= \prod_n \left(d^N \vec{\Phi}_n \; e^{-\vec{\Phi}_n^2 - g(\vec{\Phi}_n^2 - 1)^2} \right) \\
&\equiv \prod_n d\mu(\Phi_n).
\end{aligned}
\tag{4.6}
$$

This is a generalized Ising model. Indeed in the limit $g \longrightarrow \infty$ the dominant configurations are such that $\Phi_1^2 + ... + \Phi_N^2 = 1$, i.e. points on the sphere S^{N-1}. Hence

$$
\frac{\int d\mu(\Phi_n) f(\vec{\Phi}_n)}{\int d\mu(\Phi_n)} = \frac{\int d\Omega_{N-1} f(\vec{\Phi}_n)}{\int d\Omega_{N-1}} \;,\; g \longrightarrow \infty.
\tag{4.7}
$$

For $N = 1$ we obtain

$$
\frac{\int d\mu(\Phi_n) f(\vec{\Phi}_n)}{\int d\mu(\Phi_n)} = \frac{1}{2}(f(+1) + f(-1)) \;,\; g \longrightarrow \infty.
\tag{4.8}
$$

Thus the limit $g \longrightarrow \infty$ of the $O(1)$ model is precisely the Ising model in d dimensions. The limit $g \longrightarrow \infty$ of the $O(3)$ model corresponds to the Heisenberg model in d dimensions. The $O(N)$ models on the lattice are thus intimately related to spin models.

There are two phases in this model. A disordered (paramagnetic) phase characterized by $< \Phi_n^i >= 0$ and an ordered (ferromagnetic) phase characterized by $< \Phi_n^i >= v_i \neq 0$. This can be seen in various ways. The easiest way is to look for the minima of the classical potential

$$
V[\phi] = -\int d^d x \left(\frac{1}{2} m^2 \phi^i \phi^i + \frac{\lambda}{4} (\phi^i \phi^i)^2 \right).
\tag{4.9}
$$

The equation of motion reads

$$
[m^2 + \frac{\lambda}{2} \phi^j \phi^j] \phi^i = 0.
\tag{4.10}
$$

For $m^2 > 0$ there is a unique solution $\phi^i = 0$ whereas for $m^2 < 0$ there is a second solution given by $\phi^j \phi^j = -2m^2/\lambda$.

A more precise calculation is as follows. Let us compute the expectation value $< \Phi_n^i >$ on the lattice which is defined by

$$
\begin{aligned}
< \phi_n^i > &= \frac{\int d\mu(\Phi) \; \Phi_n^i e^{2\kappa \sum_n \sum_\mu \Phi_n^i \Phi_{n+\hat{\mu}}^i}}{\int d\mu(\Phi) \; e^{2\kappa \sum_n \sum_\mu \Phi_n^i \Phi_{n+\hat{\mu}}^i}} \\
&= \frac{\int d\mu(\Phi) \; \Phi_n^i e^{\kappa \sum_n \Phi_n^i \sum_\mu (\Phi_{n+\hat{\mu}}^i + \Phi_{n-\hat{\mu}}^i)}}{\int d\mu(\Phi) \; e^{\kappa \sum_n \Phi_n^i \sum_n \sum_\mu (\Phi_{n+\hat{\mu}}^i + \Phi_{n-\hat{\mu}}^i)}}.
\end{aligned}
\tag{4.11}
$$

Now we approximate the spins Φ_n^i at the $2d$ nearest neighbors of each spin Φ_n^i by the average $v^i =< \Phi_n^i >$, viz

$$
\frac{\sum_\mu (\Phi_{n+\hat{\mu}}^i + \Phi_{n-\hat{\mu}}^i)}{2d} = v^i.
\tag{4.12}
$$

This is a crude form of the mean field approximation. Equation (4.11) becomes

$$
\begin{aligned}
v^i &= \frac{\int d\mu(\Phi)\ \Phi_n^i e^{4\kappa d \sum_n \Phi_n^i v^i}}{\int d\mu(\Phi)\ e^{4\kappa d \sum_n \Phi_n^i v^i}} \\
&= \frac{\int d\mu(\Phi_n)\ \Phi_n^i e^{4\kappa d \Phi_n^i v^i}}{\int d\mu(\Phi_n^i)\ e^{4\kappa d \Phi_n^i v^i}}.
\end{aligned}
\tag{4.13}
$$

The extra factor of 2 in the exponents comes from the fact that the coupling between any two nearest neighbor spins on the lattice occurs twice. We write the above equation as

$$
v^i = \frac{\partial}{\partial J^i} \ln Z[J]|_{J^i = 4\kappa d v^i}.
\tag{4.14}
$$

$$
\begin{aligned}
Z[J] &= \int d\mu(\Phi_n)\ e^{\Phi_n^i J^i} \\
&= \int d^N \Phi_n^i\ e^{-\Phi_n^i \Phi_n^i - g(\Phi_n^i \Phi_n^i - 1)^2 + \Phi_n^i J^i}.
\end{aligned}
\tag{4.15}
$$

The limit $g \longrightarrow 0$: In this case we have

$$
Z[J] = \int d^N \Phi_n^i\ e^{-\Phi_n^i \Phi_n^i + \Phi_n^i J^i} = Z[0]\ e^{\frac{J^i J^i}{4}}.
\tag{4.16}
$$

In other words

$$
v^i = 2\kappa_c d v^i \Rightarrow \kappa_c = \frac{1}{2d}.
\tag{4.17}
$$

The limit $g \longrightarrow \infty$: In this case we have

$$
\begin{aligned}
Z[J] &= \mathcal{N} \int d^N \Phi_n^i\ \delta(\Phi_n^i \Phi_n^i - 1)\ e^{\Phi_n^i J^i} \\
&= \mathcal{N} \int d^N \Phi_n^i\ \delta(\Phi_n^i \Phi_n^i - 1) \left[1 + \Phi_n^i J^i + \frac{1}{2} \Phi_n^i \Phi_n^j J^i J^j + ... \right].
\end{aligned}
\tag{4.18}
$$

By using rotational invariance in N dimensions we obtain

$$
\int d^N \Phi_n^i\ \delta(\Phi_n^i \Phi_n^i - 1)\ \Phi_n^i = 0.
\tag{4.19}
$$

$$
\int d^N \Phi_n^i\ \delta(\Phi_n^i \Phi_n^i - 1)\ \Phi_n^i \Phi_n^j = \frac{\delta^{ij}}{N} \int d^N \Phi_n^i\ \delta(\Phi_n^i \Phi_n^i - 1)\ \Phi_n^k \Phi_n^k = \frac{\delta^{ij}}{N} \frac{Z[0]}{\mathcal{N}}.
\tag{4.20}
$$

Hence

$$
Z[J] = Z[0] \left[1 + \frac{J^i J^i}{2N} + ... \right].
\tag{4.21}
$$

Thus

$$
v^i = \frac{J^i}{N} = \frac{4\kappa_c d v^i}{N} \Rightarrow \kappa_c = \frac{N}{4d}.
\tag{4.22}
$$

The limit of The Ising Model: In this case we have

$$N = 1 \ , \ g \longrightarrow \infty. \tag{4.23}$$

We compute then

$$
\begin{aligned}
Z[J] &= \mathcal{N} \int d\Phi_n \ \delta(\Phi_n^2 - 1) \ e^{\Phi_n J} \\
&= Z[0] \cosh J.
\end{aligned} \tag{4.24}
$$

Thus

$$v = \tanh 4\kappa dv. \tag{4.25}$$

A graphical sketch of the solutions of this equation will show that for $\kappa < \kappa_c$ there is only one intersection point at $v = 0$ whereas for $\kappa > \kappa_c$ there are two intersection points away from the zero, i.e. $v \neq 0$. Clearly for κ near κ_c the solution v is near 0 and thus we can expand the above equation as

$$v = 4\kappa dv - \frac{1}{3}(4\kappa d)^3 v^2 + ... \tag{4.26}$$

The solution is

$$\frac{1}{3}(4d)^2 \kappa^3 v^2 = \kappa - \kappa_c. \tag{4.27}$$

Thus only for $\kappa > \kappa_c$ there is a non zero solution.

In summary we have the two phases

$$\kappa > \kappa_c \ : \ \text{broken, ordered, ferromagnetic} \tag{4.28}$$

$$\kappa < \kappa_c \ : \ \text{symmetric, disordered, paramagnetic}. \tag{4.29}$$

The critical line $\kappa_c = \kappa_c(g)$ interpolates in the $\kappa - g$ plane between the two lines given by

$$\kappa_c = \frac{N}{4d} \ , \ g \longrightarrow \infty. \tag{4.30}$$

$$\kappa_c = \frac{1}{2d} \ , \ g \longrightarrow 0. \tag{4.31}$$

For $d = 4$ the critical value at $g = 0$ is $\kappa_c = 1/8$ for all N. This critical value can be derived in a different way as follows. We know that the renormalized mass at one-loop order in the continuum ϕ^4 with $O(N)$ symmetry is given by the equation

$$
\begin{aligned}
m_R^2 &= m^2 + (N+2)\lambda I(m^2, \Lambda) \\
&= m^2 + \frac{(N+2)\lambda}{16\pi^2}\Lambda^2 + \frac{(N+2)\lambda}{16\pi^2}m^2 \ln \frac{m^2}{\Lambda^2} + \frac{(N+2)\lambda}{16\pi^2}m^2 \mathbf{C} + \text{finite terms}.
\end{aligned} \tag{4.32}
$$

This equation reads in terms of dimensionless quantities as follows

$$a^2 m_R^2 \;=\; am^2 + \frac{(N+2)\lambda}{16\pi^2} + \frac{(N+2)\lambda}{16\pi^2} a^2 m^2 \ln a^2 m^2 + \frac{(N+2)\lambda}{16\pi^2} a^2 m^2 \mathbf{C} + a^2 \times \text{finite terms.}$$
(4.33)

The lattice space a is formally identified with the inverse cut off $1/\Lambda$, viz

$$a = \frac{1}{\Lambda}.$$
(4.34)

Thus we obtain in the continuum limit $a \longrightarrow 0$ the result

$$a^2 m^2 \longrightarrow -\frac{(N+2)\lambda}{16\pi^2} + \frac{(N+2)\lambda}{16\pi^2} a^2 m^2 \ln a^2 m^2 + \frac{(N+2)\lambda}{16\pi^2} a^2 m^2 \mathbf{C} + a^2 \times \text{finite terms.}$$
(4.35)

In other words (with $r_0 = (N+2)/8\pi^2$)

$$a^2 m^2 \longrightarrow a^2 m_c^2 = -\frac{r_0}{2}\lambda + O(\lambda^2).$$
(4.36)

This is the critical line for small values of the coupling constant as we will now show. Expressing this equation in terms of κ and g we obtain

$$\frac{1 - 2g}{\kappa} - 8 \longrightarrow -\frac{r_0}{2}\frac{g}{\kappa^2} + O(\lambda^2).$$
(4.37)

This can be brought to the form

$$\left[\kappa - \frac{1}{16}(1 - 2g)\right]^2 \longrightarrow \frac{1}{256}\left[1 + 16 r_0 g - 4g\right] + O(g^2/\kappa^2).$$
(4.38)

We get the result

$$\kappa \longrightarrow \kappa_c = \frac{1}{8} + (\frac{r_0}{2} - \frac{1}{4})g + O(g^2).$$
(4.39)

This result is of fundamental importance. The continuum limit $a \longrightarrow 0$ corresponds precisely to the limit in which the mass approaches its critical value. This happens for every value of the coupling constant and hence the continuum limit $a \longrightarrow 0$ is the limit in which we approach the critical line. The continuum limit is therefore a second order phase transition.

4.2   The HM Algorithm

We start by considering the Hamiltonian

$$H[\phi, P] \;=\; \frac{1}{2}\sum_n P_n^i P_n^i + \sum_n \left(-2\kappa \sum_\mu \Phi_n^i \Phi_{n+\hat\mu}^i + \Phi_n^i \Phi_n^i + g(\Phi_n^i \Phi_n^i - 1)^2 \right).(4.40)$$

The Hamilton equations of motion are

$$\frac{\partial H}{\partial P_n^i} = \dot{\Phi}_n^i = P_n^i$$

$$\frac{\partial H}{\partial \Phi_n^i} = -\dot{P}_n^i = V_n^i. \tag{4.41}$$

The force is given by

$$V_n^i = \frac{\partial S}{\partial \Phi_n^i}$$

$$= -2\kappa \sum_\mu (\Phi_{n+\hat{\mu}}^i + \Phi_{n-\hat{\mu}}^i) + 2\Phi_n^i + 4g\Phi_n^i(\Phi_n^j \Phi_n^j - 1). \tag{4.42}$$

The leap frog, or Stormer-Verlet, algorithm, which maintains the symmetry under time reversible and the conservation of the phase space volume of the above Hamilton equations, is then given by the equations

$$P_n^i(t + \frac{\delta t}{2}) = (P)_n^i(t) - \frac{\delta t}{2} V_n^i(t). \tag{4.43}$$

$$\Phi_n^i(t + \delta t) = \Phi_n^i(t) + \delta t P_n^i(t + \frac{\delta t}{2}). \tag{4.44}$$

$$P_n^i(t + \delta t) = P_n^i(t + \frac{\delta t}{2}) - \frac{\delta t}{2} V_n^i(t + \delta t). \tag{4.45}$$

We recall that $t = n\delta t$, $n = 0, 1, 2, ..., \nu - 1, \nu$ where the point $n = 0$ corresponds to the initial configuration $\Phi_n^i(0)$ whereas $n = \nu$ corresponds to the final configuration $\Phi_n^i(T)$ where $T = \nu\delta t$. This algorithm does not conserve the Hamiltonian due to the systematic error associated with the discretization, which goes as $O(\delta t^2)$, but as can be shown the addition of a Metropolis accept-reject step will nevertheless lead to an exact algorithm.

The hybrid Monte Carlo algorithm in this case can be summarized as follows:

- 1) Choose $P(0)$ such that $P(0)$ is distributed according to the Gaussian probability distribution $\exp(-\frac{1}{2}\sum_n P_n^i P_n^i)$. In particular we choose P_n^i such that

$$P_n^i = \sqrt{-2\ln(1 - x_1)} \cos 2\pi(1 - x_2), \tag{4.46}$$

  where x_1 and x_2 are two random numbers uniformly distributed in the interval $[0, 1]$. This step is crucial if we want to avoid ergodic problems.

- 2)Find the configuration $(\Phi(T), P(T))$ by solving the above differential equations of motion.

- 3)Accept the configuration $(\Phi(T), P(T))$ with a probability

$$\min(1, e^{-\Delta H[\Phi, P]}), \tag{4.47}$$

  where ΔH is the corresponding change in the Hamiltonian when we go from $(\Phi(0), P(0))$ to $(\Phi(T), P(T))$.

- 4) Repeat.

4.3 Renormalization and Continuum Limit

The continuum and lattice actions for Φ^4 theory in two dimensions with $N = 1$ are given, with some slight change of notation, by

$$S[\phi] = \int d^2 x \left(\frac{1}{2}(\partial_\mu \phi)^2 + \frac{1}{2}\mu_0^2 \phi^2 + \frac{\lambda}{4}\phi^4 \right). \tag{4.48}$$

$$S[\phi] = \sum_n \left(-2\kappa \sum_\mu \Phi_n \Phi_{n+\hat{\mu}} + \Phi_n^2 + g(\Phi_n^2 - 1)^2 \right). \tag{4.49}$$

$$\mu_0^2 = m^2. \tag{4.50}$$

$$\mu_{0l}^2 \equiv \mu_0^2 a^2 = \frac{1 - 2g}{\kappa} - 4 \ , \ \ \lambda_l \equiv \lambda a^2 = \frac{g}{\kappa^2}. \tag{4.51}$$

In the simulations we will start by fixing the lattice quartic coupling λ_l and the lattice mass parameter μ_{0l}^2 which then allows us to fix κ and g as

$$\kappa = \frac{\sqrt{8\lambda_l + (\mu_{0l}^2 + 4)^2} - (\mu_{0l}^2 + 4)}{4\lambda_l}. \tag{4.52}$$

$$g = \kappa^2 \lambda_l. \tag{4.53}$$

The phase diagram will be drawn originally in the $\mu_{0l}^2 - \lambda_l$ plane. This is the lattice phase diagram. This should be extrapolated to the infinite volume limit $L = Na \longrightarrow \infty$.

The Euclidean quantum field theory phase diagram should be drawn in terms of the renormalized parameters and is obtained from the lattice phase diagram by taking the limit $a \longrightarrow 0$. In two dimensions the Φ^4 theory requires only mass renormalization while the quartic coupling constant is finite. Indeed, the bare mass μ_0^2 diverges logarithmically when we remove the cutoff, i.e. in the limit $\Lambda \longrightarrow \infty$ where $\Lambda = 1/a$ while λ is independent of a. As a consequence, the lattice parameters will go to zero in the continuum limit $a \longrightarrow 0$.

We know that mass renormalization is due to the tadpole diagram which is the only divergent Feynman diagram in the theory and takes the form of a simple reparametrization given by

$$\mu_0^2 = \mu^2 - \delta\mu^2, \tag{4.54}$$

where μ^2 is the renormalized mass parameter and $\delta\mu^2$ is the counter term which is fixed via an appropriate renormalization condition. The unltraviolet divergence $\ln \Lambda$ of μ_0^2 is contained in $\delta\mu^2$ while the renormalization condition will split the finite part of μ_0^2 between μ^2 and $\delta\mu^2$. The choice of the renormalization condition can be quite arbitrary. A convenient choice suitable for Monte Carlo measurements and which distinguishes between the two phases of the theory is given by the usual normal ordering prescription [2] .

Quantization at one-loop gives explicitly the $2-$point function

$$\Gamma^{(2)}(p) = p^2 + \mu_0^2 + 3\lambda \int \frac{d^2k}{(2\pi)^2} \frac{1}{k^2 + \mu_0^2}. \tag{4.55}$$

A self-consistent Hartree treatment gives then the result

$$
\begin{aligned}
\Gamma^{(2)}(p) &= p^2 + \mu_0^2 + 3\lambda \int \frac{d^2k}{(2\pi)^2} \frac{1}{\Gamma^{(2)}(k)} \\
&= p^2 + \mu^2 + 3\lambda \int \frac{d^2k}{(2\pi)^2} \frac{1}{\Gamma^{(2)}(k)} - \delta\mu^2 \\
&= p^2 + \mu^2 + 3\lambda \int \frac{d^2k}{(2\pi)^2} \frac{1}{k^2 + \mu^2} - \delta\mu^2 + \text{two} - \text{loop}
\end{aligned}
\tag{4.56}
$$

This should certainly work in the symmetric phase where $\mu^2 > 0$. We can also write this as

$$\Gamma^{(2)}(p) = p^2 + \mu^2 + \Sigma(p) \ , \ \Sigma(p) = 3\lambda A_{\mu^2} - \delta\mu^2 + \text{two} - \text{loop}. \tag{4.57}$$

A_{μ^2} is precisely the value of the tadpole diagram given by

$$A_{\mu^2} = \int \frac{d^2k}{(2\pi)^2} \frac{1}{k^2 + \mu^2}. \tag{4.58}$$

The renormalization condition which is equivalent to normal ordering the interaction in the interaction picture in the symmetric phase is equivalent to the choice

$$\delta\mu^2 = 3\lambda A_{\mu^2}. \tag{4.59}$$

A dimensionless coupling constant can the be defined by

$$f = \frac{\lambda}{\mu^2}. \tag{4.60}$$

The action becomes

$$S[\phi] = \int d^2x \left(\frac{1}{2}(\partial_\mu\phi)^2 + \frac{1}{2}\mu^2(1 - 3fA_{\mu^2})\phi^2 + \frac{f\mu^2}{4}\phi^4 \right). \tag{4.61}$$

For sufficiently small f the exact effective potential is well approximated by the classical potential with a single minimum at $\phi_{\text{cl}} = 0$. For larger f, the coefficient of the mass term in the above action can become negative and as a consequence a transition to the broken symmetry phase is possible, although in this regime the effective potential is no longer well approximated by the classical potential. Indeed, a transition to the broken symmetry phase was shown to be present in [4], where a duality between the strong coupling regime of the above action and a weakly coupled theory normal ordered with respect to the broken phase was explicitly constructed.

The sites on the lattice are located at $x_\mu = n_\mu a$ where $n_\mu = 0, ..., N - 1$ with $L = Na$. The plane waves on a finite volume lattice with periodic boundary conditions are $\exp(ipx)$

with $p_\mu = m_\mu 2\pi/L$ where $m_\mu = -N/2+1, -N/2+2, ..., N/2$ for N even. This means that the zero of the $x-$space is located at the edge of the box while the zero of the $p-$space is located in the middle of the box. We have therefore the normalization conditions $\sum_x \exp(-i(p-p')x) = \delta_{p,p'}$ and $\sum_p \exp(-i(x-x')p) = \delta_{x,x'}$ where, for example, $\sum_p = \sum_m /L^2$. In the infinite volume limit defined by $L = Na \longrightarrow \infty$ with a fixed we have $\sum_p \longrightarrow \int_{-\pi/a}^{\pi/a} d^2p/(2\pi)^2$. It is not difficult to show that on the lattice the propagator $1/(p^2 + \mu^2)$ becomes $a^2/(4\sum_\mu \sin^2 ap_\mu/2 + \mu_l^2)$ [1]. Thus on a finite volume lattice with periodic boundary conditions the Feynman diagram A_{μ^2} takes the form

$$
\begin{aligned}
A_{\mu^2} &= \sum_{p_1,p_2} \frac{a^2}{4\sin^2 ap_1/2 + 4\sin^2 ap_2/2 + \mu_l^2} \\
&= \frac{1}{N^2} \sum_{m_1=1}^{N} \sum_{m_2=1}^{N} \frac{1}{4\sin^2 \pi m_1/N + 4\sin^2 \pi m_2/N + \mu_l^2}.
\end{aligned}
\tag{4.62}
$$

In the last line we have shifted the integers m_1 and m_2 by $N/2$. Hence on a finite volume lattice with periodic boundary conditions equation (4.54), together with equation (4.59), becomes

$$
F(\mu_l^2) = \mu_l^2 - 3\lambda_l A_{\mu^2} - \mu_{0l}^2 = 0.
\tag{4.63}
$$

Given the critical value of μ_{0l}^2 for every value of λ_l we need then to determine the corresponding critical value of μ_l^2. This can be done numerically using the Newton-Raphson algorithm. The continuum limit $a \longrightarrow 0$ is then given by extrapolating the results into the origin, i.e. taking $\lambda_l = a^2\lambda \longrightarrow 0$, $\mu_l^2 = a^2\mu^2 \longrightarrow 0$ in order to determine the critical value

$$
f_c = \lim_{\lambda_l,\mu_l^2 \longrightarrow 0} \frac{\lambda_l}{\mu_{lc}^2}.
\tag{4.64}
$$

4.4   HMC Simulation Calculation of The Critical Line

We measure as observables the average value of the action, the specific heat, the magnetization, the susceptibility and the Binder cumulant defined respectively by

$$
< S > .
\tag{4.65}
$$

$$
C_v = < S^2 > - < S >^2 .
\tag{4.66}
$$

$$
M = \frac{1}{N^2} < m > , \; m = |\sum_n \phi_n|.
\tag{4.67}
$$

$$
\chi = < m^2 > - < m >^2 .
\tag{4.68}
$$

$$U = 1 - \frac{<m^4>}{3<m^2>^2}. \tag{4.69}$$

We note the use of the absolute value in the definition of the magnetization since the usual definition $M =< \sum_n \phi_n > /N^2$ is automatically zero on the lattice because of the symmetry $\phi \longrightarrow -\phi$. The specific heat diverges at the critical point logarithmically as the lattice size is sent to infinity. The susceptibility shows also a peak at the critical point whereas the Binder cumulant exhibits a fixed point for all values of N.

We run simulations with $T_{\text{th}} + T_{\text{mc}} \times T_{\text{co}}$ steps with $T_{\text{th}} = 2^{13}$ thermalization steps and $T_{\text{mc}} = 2^{14}$ measurement steps. Every two successive measurements are separated by $T_{\text{co}} = 2^3$ steps to reduce auto-correlations. We use ran2 as our random numbers generator and the Jackknife method to estimate error bars. The hybrid Monte Carlo code used in these simulations can be found in the last chapter.

We have considered lattices with $N = 16, 32$ and 49 and values of the quartic coupling given by $\lambda_l = 1, 0.7, 0.5, 0.25$. Some results are shown on figure (4.1). The critical value μ_{0l*}^2 for each value of λ_l is found from averaging the values at which the peaks in the specific heat and the susceptibility occur. The results are shown on the second column of table (4.1). The final step is take the continuum limit $a \longrightarrow 0$ in order to find the critical value μ_{l*}^2 by solving the renormalization condition (4.63) using the Newton-Raphson method. This is an iterative method based on a single iteration given by $\mu_{l*}^2 = \mu_{l*}^2 - F/F'$. The corresponding results are shown on the third column of table (4.1). The critical line is shown on figure (4.2) with a linear fit going through the origin given by

$$\lambda_l = (9.88 \pm 0.22)\mu_{l*}^2. \tag{4.70}$$

This should be compared with the much more precise result $\lambda_l = 10.8\mu_{l*}^2$ published in [3]. The above result is sufficient for our purposes here.

λ_l	μ_{0l*}^2	μ_{l*}^2
1.0	-1.25 ± 0.05	1.00×10^{-2}
0.7	-0.95 ± 0.05	6.89×10^{-2}
0.5	-0.7 ± 0.00	5.52×10^{-2}
0.25	-0.4 ± 0.00	2.53×10^{-2}

Table 4.1:

Figure 4.1:

Figure 4.2:

Bibliography

[1] J. Smit, "Introduction to quantum fields on a lattice: A robust mate," Cambridge Lect. Notes Phys. **15**, 1 (2002).

[2] W. Loinaz and R. S. Willey, "Monte Carlo simulation calculation of critical coupling constant for continuum phi**4 in two-dimensions," Phys. Rev. D **58**, 076003 (1998) [hep-lat/9712008].

[3] D. Schaich and W. Loinaz, "An Improved lattice measurement of the critical coupling in phi(2)**4 theory," Phys. Rev. D **79**, 056008 (2009) [arXiv:0902.0045 [hep-lat]].

[4] S. J. Chang, "The Existence of a Second Order Phase Transition in the Two-Dimensional phi**4 Field Theory," Phys. Rev. D **13**, 2778 (1976) [Phys. Rev. D **16**, 1979 (1977)].

Chapter 5

(Multi-Trace) Quartic Matrix Models

5.1 The Pure Real Quartic Matrix Model

This is a very well known, and a very well studied, model which depends on a single hermitian matrix M. This is given by

$$
\begin{aligned}
V &= BTrM^2 + CTrM^4 \\
&= \frac{N}{g}(-TrM^2 + \frac{1}{4}TrM^4).
\end{aligned}
\tag{5.1}
$$

The model depends actually on a single coupling g such that

$$
B = -\frac{N}{g} \ , \ C = \frac{N}{4g}.
\tag{5.2}
$$

There are two stable phases in this model:

Disordered phase (one-cut) for $g \geq g_c$: This is characterized by the eigenvalues distribution of the matrix M given by

$$
\begin{aligned}
\rho(\lambda) &= \frac{1}{N\pi}(2C\lambda^2 + B + C\delta^2)\sqrt{\delta^2 - \lambda^2} \\
&= \frac{1}{g\pi}(\frac{1}{2}\lambda^2 - 1 + r^2)\sqrt{4r^2 - \lambda^2}.
\end{aligned}
\tag{5.3}
$$

This is a single cut solution with the cut defined by

$$
-2r \leq \lambda \leq 2r.
\tag{5.4}
$$

$$
r = \frac{1}{2}\delta.
\tag{5.5}
$$

$$
\begin{aligned}
\delta^2 &= \frac{1}{3C}(-B + \sqrt{B^2 + 12NC}) \\
&= \frac{1}{3}(1 + \sqrt{1 + 3g}).
\end{aligned}
\tag{5.6}
$$

Non-uniform ordered phase (two-cut) for $g \leq g_c$**:** This is characterized by the eigenvalues distribution of the matrix M given by

$$
\begin{aligned}
\rho(\lambda) &= \frac{2C|\lambda|}{N\pi}\sqrt{(\lambda^2 - \delta_1^2)(\delta_2^2 - \lambda^2)} \\
&= \frac{|\lambda|}{2g\pi}\sqrt{(\lambda^2 - r_-^2)(r_+^2 - \lambda^2)}.
\end{aligned}
\tag{5.7}
$$

Here there are two cuts defined by

$$
r_- \leq |\lambda| \leq r_+.
\tag{5.8}
$$

$$
r_- = \delta_1 \ , \ r_+ = \delta_2.
\tag{5.9}
$$

$$
\begin{aligned}
r_{\mp}^2 &= \frac{1}{2C}(-B \mp 2\sqrt{NC}) \\
&= 2(1 \mp \sqrt{g}).
\end{aligned}
\tag{5.10}
$$

A third order transition between the above two phases occurs at the critical point

$$
g_c = 1 \leftrightarrow B_c^2 = 4NC \leftrightarrow B_c = -2\sqrt{NC}.
\tag{5.11}
$$

There is a third phase in this model: the so-called Ising or uniform ordered phase, which despite the fact that it is not stable, plays an important role in generalizations of this model, such as the one discussed in the next section, towards noncommutative Φ^4.

5.2 The Multi-Trace Matrix Model

Our primary interest here is the theory of noncommutative Φ^4 on the fuzzy sphere given by the action

$$
S = \frac{4\pi R^2}{N+1} Tr\left(\frac{1}{2R^2}\Phi\Delta\Phi + \frac{1}{2}m^2\Phi^2 + \frac{\lambda}{4!}\Phi^4\right).
\tag{5.12}
$$

The Laplacian is $\Delta = [L_a, [L_a, ...]]$. Equivalently with the substitution $\Phi = \mathcal{M}/\sqrt{2\pi\theta}$, where $\mathcal{M} = \sum_{i,j=1}^{N} M_{ij}|i><j|$, this action reads

$$
S = Tr\left(a\mathcal{M}\Delta\mathcal{M} + b\mathcal{M}^2 + c\mathcal{M}^4\right).
\tag{5.13}
$$

The parameters are[1]

$$
a = \frac{1}{2R^2} \ , \ b = \frac{1}{2}m^2 \ , \ c = \frac{\lambda}{4!}\frac{1}{2\pi\theta}.
\tag{5.14}
$$

_____

[1]The noncommutativity parameter on the fuzzy sphere is related to the radius of the sphere by $\theta = 2R^2/\sqrt{N^2 - 1}$.

In terms of the matrix M the action reads

$$S[M] \;=\; r^2 K[M] + Tr\big[bM^2 + cM^4\big]. \tag{5.15}$$

The kinetic matrix is given by

$$K[M] \;=\; Tr\left[-\Gamma^+ M\Gamma M - \frac{1}{N+1}\Gamma_3 M\Gamma_3 M + EM^2\right]. \tag{5.16}$$

The matrices Γ, Γ_3 and E are given by

$$(\Gamma_3)_{lm} = l\delta_{lm} \ , \ (\Gamma)_{lm} = \sqrt{(m-1)(1-\frac{m}{N+1})}\delta_{lm-1} \ , \ (E)_{lm} = (l-\frac{1}{2})\delta_{lm}. \tag{5.17}$$

The relationship between the parameters a and r^2 is given by

$$r^2 = 2aN \tag{5.18}$$

We start from the path integral

$$
\begin{aligned}
Z \;&=\; \int dM \ \exp\big(-S[M]\big) \\
&=\; \int d\Lambda \ \Delta^2(\Lambda) \ \exp\big(-Tr(b\Lambda^2 + c\Lambda^4)\big) \int dU \ \exp\big(-r^2 K[U\Lambda U^{-1}]\big).
\end{aligned} \tag{5.19}
$$

The second line involves the diagonalization of the matrix M (more on this below). The calculation of the integral over $U \in U(N)$ is a very long calculation done in [2,3]. The end result is a multi-trace effective potential given by (assuming the symmetry $M \longrightarrow -M$)

$$
\begin{aligned}
S_{\text{eff}} \;=\; & \sum_i (b\lambda_i^2 + c\lambda_i^4) - \frac{1}{2}\sum_{i\neq j}\ln(\lambda_i - \lambda_j)^2 \\
& + \left[\frac{r^2}{8}v_{2,1}\sum_{i\neq j}(\lambda_i - \lambda_j)^2 + \frac{r^4}{48}v_{4,1}\sum_{i\neq j}(\lambda_i - \lambda_j)^4 - \frac{r^4}{24N^2}v_{2,2}\big[\sum_{i\neq j}(\lambda_i - \lambda_j)^2\big]^2 + ...\right].
\end{aligned} \tag{5.20}
$$

The coefficients v will be given below. If we do not assume the symmetry $M \longrightarrow -M$ then obviously there will be extra terms with more interesting consequences for the phase structure as we will discuss briefly below.

This problem (5.20) is a generalization of the quartic Hermitian matrix potential model. Indeed, this effective potential corresponds to the matrix model given by

$$V \;=\; \left(b + \frac{aN^2 v_{2,1}}{2}\right)TrM^2 + \left(c + \frac{a^2 N^3 v_{4,1}}{6}\right)TrM^4 - \frac{2\eta a^2 N^2}{3}\Big[TrM^2\Big]^2. \tag{5.21}$$

This can also be solved exactly as shown in [2]. The strength of the multi-trace term η is given by

$$\eta = v_{2,2} - \frac{3}{4}v_{4,1}. \tag{5.22}$$

The coefficients $v_{2,1}$, $v_{4,1}$ and $v_{2,2}$ are given by the following two competing calculations of [2] and [3] given respectively by

$$v_{2,1} = 1 \ , \ v_{4,1} = 0 \ , \ v_{2,2} = \frac{1}{8}. \tag{5.23}$$

$$v_{2,1} = -1 \ , \ v_{4,1} = \frac{3}{2} \ , \ v_{2,2} = 0. \tag{5.24}$$

This discrepancy is discussed in [2].

5.3 Model and Algorithm

We thus start from the potential and the partition function

$$V \ = \ Tr\Big(BM^2 + CM^4\Big) + D\Big(TrM^2\Big)^2. \tag{5.25}$$

We may include the odd terms found in [2] without any real extra effort. We will not do this here for simplicity, but we will include them for completeness in the attached code. The partition function (path integral) is given by

$$Z = \int dM \ \exp\big(-V\big). \tag{5.26}$$

The relationship between the two sets of parameters $\{a, b, c\}$ and $\{B, C, D\}$ is given by

$$B = b + \frac{aN^2 v_{2,1}}{2} \ , \ C = c + \frac{a^2 N^3 v_{4,1}}{6} \ , \ D = -\frac{2\eta a^2 N^2}{3}. \tag{5.27}$$

The collpased parameters are

$$\tilde{B} = \frac{B}{N^{\frac{3}{2}}} = \tilde{b} + \frac{\tilde{a} v_{2,1}}{2} \ , \ \tilde{C} = \frac{C}{N^2} = \tilde{c} + \frac{\tilde{a}^2 v_{4,1}}{6} \ , \ D = -\frac{2\eta \tilde{a}^2 N}{3}. \tag{5.28}$$

Only two of these three parameters are independent. For consistency of the large N limit, we must choose $\tilde{a}$ to be any fixed number. We then choose for simplicity $\tilde{a} = 1$ or equivalently $D = -2\eta N/3^2$.

We can now diagonalize the scalar matrix M as

$$M = U\Lambda U^{-1}. \tag{5.29}$$

We compute

$$\delta M = U\Big(\delta\Lambda + [U^{-1}\delta U, \Lambda]\Big)U^{-1}. \tag{5.30}$$

---

[2]The authors of [1] chose instead $a = 1$.

Thus (with $U^{-1}\delta U = i\delta V$ being an element of the Lie algebra of SU(N))

$$
\begin{aligned}
Tr(\delta M)^2 &= Tr(\delta\Lambda)^2 + Tr[U^{-1}\delta U, \Lambda]^2 \\
&= \sum_i (\delta\lambda_i)^2 + \sum_{i\neq j}(\lambda_i - \lambda_j)^2 \delta V_{ij}\delta V_{ij}^*.
\end{aligned}
\tag{5.31}
$$

We count N^2 real degrees of freedom as there should be. The measure is therefore given by

$$
\begin{aligned}
dM &= \prod_i d\lambda_i \prod_{i\neq j} dV_{ij}dV_{ij}^* \sqrt{\det(\text{metric})} \\
&= \prod_i d\lambda_i \prod_{i\neq j} dV_{ij}dV_{ij}^* \sqrt{\prod_{i\neq j}(\lambda_i - \lambda_j)^2}.
\end{aligned}
\tag{5.32}
$$

We write this as

$$
dM = d\Lambda dU \Delta^2(\Lambda).
\tag{5.33}
$$

The dU is the usual Haar measure over the group SU(N) which is normalized such that $\int dU = 1$, whereas the Jacobian $\Delta^2(\Lambda)$ is precisely the so-called Vandermonde determinant defined by

$$
\Delta^2(\Lambda) = \prod_{i>j}(\lambda_i - \lambda_j)^2.
\tag{5.34}
$$

The partition function becomes

$$
Z = \int d\Lambda\ \Delta^2(\Lambda)\ \exp\left(-Tr\left(B\Lambda^2 + C\Lambda^4\right) - D\left(Tr\Lambda^2\right)^2\right).
\tag{5.35}
$$

We are therefore dealing with an effective potential given by

$$
V_{\text{eff}} = B\sum_{i=1}\lambda_i^2 + C\sum_{i=1}\lambda_i^4 + D\left(\sum_{i=1}\lambda_i^2\right)^2 - \frac{1}{2}\sum_{i\neq j}\ln(\lambda_i - \lambda_j)^2.
\tag{5.36}
$$

We will use the Metropolis algorithm to study this model. Under the change $\lambda_i \longrightarrow \lambda_i + h$ of the eigenvalue λ_i the above effective potential changes as $V_{\text{eff}} \longrightarrow V_{\text{eff}} + \Delta V_{i,h}$ where

$$
\Delta V_{i,h} = B\Delta S_2 + C\Delta S_4 + D(2S_2\Delta S_2 + \Delta S_2^2) + \Delta S_{\text{Vand}}.
\tag{5.37}
$$

The monomials S_n are defined by $S_n = \sum_i \lambda_i^n$ while the variations ΔS_n and ΔS_{Vand} are given by

$$
\Delta S_2 = h^2 + 2h\lambda_i.
\tag{5.38}
$$

$$
\Delta S_4 = 6h^2\lambda_i^2 + 4h\lambda_i^3 + 4h^3\lambda_i + h^4.
\tag{5.39}
$$

$$
\Delta S_{\text{Vand}} = -2\sum_{j\neq i}\ln\left|1 + \frac{h}{\lambda_i - \lambda_j}\right|.
\tag{5.40}
$$

5.4   The Disorder-to-Non-Uniform-Order Transition

The pure quartic matrix model (5.1) is characterized by a third-order phase transition between a disordered phase characterized by $< M >= 0$ and a non-uniform ordered phase characterized by $< M >= -B\gamma/2C$ where γ is an $N-$dimensional idempotent, viz $\gamma^2 = 1$. This transition is also termed one-cut-to-two-cut transition. Thus the eigenvalues distribution of the scalar field M will go from a one-cut solution centered around 0 in the disordered phase to a two-cut solution with two peaks symmetric around 0 in the uniform ordered phase. The transition should occur around $g = g_c = 1$. This transition is critical since the two different eigenvalues distributions in the two phases become identical at the transition point.

Monte Carlo tests of the above effects, and other physics, can be done using the code found in the last chapter. An illustration with 2^{20} thermalized configurations, where each two successive configurations are separated by 2^5 Monte Carlo steps to reduce autocorrelation effects, and with $N = 10$ and $g = 2, 1.5, 1, 0.5$, is shown on figure (5.1). The pure quartic matrix model is obtained from the multitrace matrix model by setting the kinetic parameter $\tilde{a}$ zero. We observe an excellent with the theoretical predictions (5.3) and (5.7).

The above transition is third-order, as we said, since the first derivative of the specific heat has a finite discontinuity at $\bar{r} = B/|B_c| = -1$ as is obvious from the exact analytic result

$$\frac{C_v}{N^2} = \frac{1}{4} \ , \ \bar{r} < -1. \tag{5.41}$$

$$\frac{C_v}{N^2} = \frac{1}{4} + \frac{2\bar{r}^4}{27} - \frac{\bar{r}}{27}(2\bar{r}^2 - 3)\sqrt{\bar{r}^2 + 3} \ , \ \bar{r} > -1. \tag{5.42}$$

This behavior is also confirmed in Monte Carlo simulation as shown for $\tilde{c} = 4$ and $N = 8$ and $N = 10$ on figure (5.2).

The above one-cut-to-two-cut transition persists largely unchanged in the quartic multitrace matrix model (5.21). On the other hand, and similarly to the above pure quartic matrix model, the Ising phase is not stable in this case and as a consequence the transition between non-uniform order and uniform-order is not observed in Monte Carlo simulations. The situation is drastically different if odd multitrace terms are included.

Figure 5.1:

Figure 5.2:

5.5   Other Suitable Algorithms

5.5.1   Over-Relaxation Algorithm

In the case of scalar Φ^4 matrix models two more algorithms are available to us. The first is the over-relaxation algorithm which is very useful in the case of noncommutative Φ^4 on the fuzzy sphere given by the action

$$S = \frac{4\pi R^2}{N+1} Tr\left(\frac{1}{2R^2}\Phi\Delta\Phi + \frac{1}{2}m^2\Phi^2 + \frac{\lambda}{4!}\Phi^4 \right). \tag{5.43}$$

We define

$$S_2 = \frac{4\pi R^2}{N+1} Tr\left(\frac{1}{2R^2}\Phi\Delta\Phi + \frac{1}{2}m^2\Phi^2 \right) \ , \ S_4 = \frac{4\pi R^2}{N+1} Tr\left(\frac{\lambda}{4!}\Phi^4 \right). \tag{5.44}$$

Let Φ_0 be some initial configuration obtained at the end of some ergodic procedure such as the Metropolis algorithm or the hybrid Monte Carlo algorithm. Let Φ_* be some new completely random configuration and thus completely independent configuration from Φ_0. If $S_* = S[\Phi_*] < S_0 = S[\Phi_0]$ then Φ_* will be accepted as the new configuration. We want to devise an algorithm in which the system is forced to accept the new configuration Φ_* even if $S_* \geq S_0$. This is equivalent to heating up the system again and then letting it cool down slowly. Towards this end, we scale the configuration Φ_* as

$$\Phi_1 = \alpha\Phi_*. \tag{5.45}$$

The scale α is chosen such that

$$S_1 = S[\Phi_1] = S_0. \tag{5.46}$$

Equivalently

$$S_{4*}\alpha^4 + S_{2*}\alpha^2 - S_0 = 0. \tag{5.47}$$

The solution is given by

$$\text{if } S_0 > 0 \; : \; \alpha^2 = \frac{\sqrt{S_{2*}^2 + 4S_0 S_{4*}} - S_{2*}}{2S_{4*}}. \tag{5.48}$$

$$\text{if } S_0 < 0 \text{ and } \{S_{2*} < -\sqrt{-4S_0 S_{4*}} < 0\} : \; \alpha^2 = \frac{\pm\sqrt{S_{2*}^2 + 4S_0 S_{4*}} - S_{2*}}{2S_{4*}}. \tag{5.49}$$

If the conditions in the above two equations are not met then we should redefine the matrix Φ_* iterativley as

$$\Phi_* \longrightarrow \frac{\Phi_* + \Phi_0}{2}. \tag{5.50}$$

Then repeat. This iterative procedure will obviously create unwanted autocorrelations due to the fact that Φ_* becomes closer in each iteration to Φ_0. However, the process will terminate in a finite number of steps and the obtained final configuration Φ_1 has a greater probability in falling in a different orbit than the original Φ_0.

The claim of [5] is that this algorithm solves the ergodic problem observed in Monte Carlo simulations of noncommutative Φ^4 on the fuzzy sphere.

5.5.2 Heat-Bath Algorithm

The second algorithm is the heat-bath algorithm which works very nicely for the unbounded Φ^4 potential

$$V \; = \; \frac{N}{g}(TrM^2 - \frac{1}{4}TrM^4). \tag{5.51}$$

Remark the minus sign in front of the quartic term. Although this potential is unbounded from below it has a well defined large N limit due to the metastability of the origin. The path integral is given by

$$\begin{aligned} Z \; &= \; \int dM \exp(-\frac{N}{g}TrM^2)\exp(\frac{N}{4g}TrM^4) \\ &= \; \int dM dQ \exp(-\frac{N}{g}TrM^2 - TrQ^2 + \sqrt{\frac{N}{g}}TrQM^2). \end{aligned} \tag{5.52}$$

The matrices M and Q are fully Gaussian. Let us then consider a Gaussian distribution

$$\sqrt{\frac{a}{\pi}} \int dx \exp(-ax^2). \tag{5.53}$$

The Gaussian random number x must be chosen, in any Monte Carlo routine, as

$$\begin{aligned} R &= \sqrt{-\frac{1}{a}\ln(1 - r_1)} \\ \phi &= 2\pi r_2 \\ x &= R\cos\phi. \end{aligned} \tag{5.54}$$

The r_1 and r_2 are two uniform random numbers between 0 and 1.

The part of the above path integral which depends on Q is Gaussian given by

$$\int dQ \exp(-Tr(Q - \frac{1}{2}\sqrt{\frac{N}{g}} M^2)^2). \tag{5.55}$$

The diagonal element Q_{ii} comes with a factor $a = 1$ while the off diagonal elements comes with a factor $a = 2$. Thus we choose

$$Q_{ii} = z_{ii}|_{a=1} + \frac{1}{2}\sqrt{\frac{N}{g}}(M^2)_{ii} \ , \ Q_{ij} = \frac{x_{ij} + iy_{ij}}{\sqrt{2}}|_{a=1} + \frac{1}{2}\sqrt{\frac{N}{g}}(M^2)_{ij}. \tag{5.56}$$

The x, y and z are Gaussian random numbers with $a = 1$.

The part of the path integral which depends on the diagonal element M_{ii} is given by

$$\int \prod_i dM_{ii} \exp \sum_i \left(-\frac{N}{g}(1 - \sqrt{\frac{g}{N}}Q_{ii})(M_{ii})^2 + \frac{1}{2}\sqrt{\frac{N}{g}}\sum_{j \neq i}(Q_{ij}M_{ji} + Q_{ji}M_{ij})M_{ii} \right) =$$

$$\int \prod_i dM_{ii} \exp \sum_i \left(-l_i(M_{ii} - \frac{h_i}{2l_i})^2 + ... \right). \tag{5.57}$$

$$l_i = \frac{N}{g}(1 - \sqrt{\frac{g}{N}}Q_{ii}) \ , \ h_i = \frac{1}{2}\sqrt{\frac{N}{g}}\sum_{j \neq i}(Q_{ij}M_{ji} + Q_{ji}M_{ij}). \tag{5.58}$$

Thus the diagonal elements M_{ii} are Gaussian numbers which come with factors $a = l_i$. Thus we choose

$$M_{ii} = \frac{x_{ii}}{\sqrt{l_i}}|_{a=1} + \frac{h_i}{2l_i}. \tag{5.59}$$

Finally, the part of the path integral which depends on the off diagonal element M_{ij} is given by

$$\int \prod_{i \neq j} dM_{ij}dM_{ij}^* \exp \sum_{i \neq j} \left(-l_{ij}M_{ij}^*M_{ij} + h_{ij}M_{ij}^* + h_{ij}^*M_{ij} \right) =$$

$$\int \prod_{i \neq j} dM_{ij}dM_{ij}^* \exp \sum_{i \neq j} \left(-l_{ij}|M_{ij} - \frac{h_{ij}}{l_{ij}}|^2 + ... \right). \tag{5.60}$$

$$l_{ij} = \frac{N}{g}\left(1 - \frac{1}{2}\sqrt{\frac{g}{N}}(Q_{ii} + Q_{jj})\right) \ , \ h_{ij} = \frac{1}{4}\sqrt{\frac{N}{g}}\left(\sum_{k \neq i}Q_{ik}M_{kj} + \sum_{k \neq j}Q_{kj}M_{ik}\right). \tag{5.61}$$

Hence the off diagonal elements M_{ij} are Gaussian numbers which come with factors $a = l_{ij}$. Thus we choose

$$M_{ij} = \frac{x_{ij} + iy_{ij}}{\sqrt{l_{ij}}}|_{a=1} + \frac{h_{ij}}{l_{ij}}. \tag{5.62}$$

This algorithms can also be applied quite effectively to simple Yang-Mills matrix models as done for example in [6, 7].

Bibliography

[1] F. Garcia Flores, X. Martin and D. O'Connor, "Simulation of a scalar field on a fuzzy sphere," Int. J. Mod. Phys. A **24**, 3917 (2009) [arXiv:0903.1986 [hep-lat]].

[2] B. Ydri, "A Multitrace Approach to Noncommutative Φ_2^4," arXiv:1410.4881 [hep-th].

[3] D. O'Connor and C. Saemann, "Fuzzy Scalar Field Theory as a Multitrace Matrix Model," JHEP **0708**, 066 (2007) [arXiv:0706.2493 [hep-th]].

[4] N. Kawahara, J. Nishimura and A. Yamaguchi, "Monte Carlo approach to nonperturbative strings - Demonstration in noncritical string theory," JHEP **0706**, 076 (2007) [hep-th/0703209].

[5] M. Panero, "Numerical simulations of a non-commutative theory: The Scalar model on the fuzzy sphere," JHEP **0705**, 082 (2007) [hep-th/0608202].

[6] T. Hotta, J. Nishimura and A. Tsuchiya, "Dynamical aspects of large N reduced models," Nucl. Phys. B **545**, 543 (1999) [hep-th/9811220].

[7] T. Azuma, S. Bal, K. Nagao and J. Nishimura, "Nonperturbative studies of fuzzy spheres in a matrix model with the Chern-Simons term," JHEP **0405**, 005 (2004) [hep-th/0401038].

Chapter 6

The Remez Algorithm and The Conjugate Gradient Method

6.1 Minimax Approximations

The rational hybrid Monte Carlo algorithm (RHMC) uses in an essential way a rational approximation to the fermionic determinant. Thus in this section we will first review the issue of rational and polynomial approximations of functions. We will follow [4, 5].

6.1.1 Minimax Polynomial Approximation and Chebyshev Polynomials

Chebyshev norm: We start by introducing the Chebyshev norm (also called uniform, infinity, supremum norm) of a continuous function f over the unit interval $[0,1]$ by the relation

$$
\begin{aligned}
||f||_\infty &= \lim_{n\longrightarrow\infty}||f||_n \\
&= \lim_{n\longrightarrow\infty}\left(\int_0^1 dx|f(x)|^n\right)^{1/n} \\
&= \max_x|f(x)|.
\end{aligned}
\tag{6.1}
$$

Minimax approximation: A minimax polynomial (or rational) approximation of f is a polynomial (or rational) function p which minimizes the Chebyshev norm of $p - f$, viz

$$
||p - f||_\infty = \min_p\max_x|p(x) - f(x)|.
\tag{6.2}
$$

Weierstrass theorem: The fundamental theorem of approximation theorem is Weierstrass theorem. This can be stated as follows. For every continuous function $f(x)$ over a closed interval $[a, b]$, and for every specified tolerance $\epsilon > 0$, there exists a polynomial $p_n(x)$ of some degree n such that for all $x \in [a, b]$, we have $||f(x) - p_n(x)||_\infty < \epsilon$. Thus any

continuous function can be arbitrarily well approximated by a polynomial. This means in particular that the space of polynomials is dense in the space of continuous functions with respect to the topology induced by the Chebyshev norm.

Chebyshev theorem (minimax polynomial approximation): We consider a function f defined on the unit interval. For any given degree n, there exists always a unique polynomial p_n of degree n which minimizes the error function

$$||e||_\infty = \max_{0 \le x \le 1} |e(x)| = \max_{0 \le x \le 1} |p_n(x) - f(x)|, \tag{6.3}$$

iff the error function $e(x)$ takes its maximum absolute value at at least $n+2$ points on the unit interval, which may include the end points, and furthermore the sign of the error alternate between the successive extrema.

We can go from the function $f(x)$ defined in the interval $[-1, +1]$ to a function $f(y)$ defined in a generic interval $[a, b]$ by considering the transformation $x \longrightarrow y$ given by

$$x = \frac{y - \frac{1}{2}(b+a)}{\frac{1}{2}(b-a)}. \tag{6.4}$$

A simple proof of this theorem can be found in [4]. This goes as follows:

- **Chebyshev's criterion is necessary: If the error has fewer than $n+2$ alternating extrema then the approximation can be improved.** Let $p(x)$ be a polynomial for which the error $e(x) = p(x) - f(x)$ has fewer than $n+2$ alternating extrema. The next largest extremum of the error, corresponding to a local extremum, is therefore smaller by some non zero gap Δ. Between any two successive alternating extrema the error obviously will pass by zero at some point z_i. If we assume that we have $d+1$ alternating extrema, then we will d zeros z_i. We can trivially construct the polynomial

$$u(x) = A \prod_i (x - z_i). \tag{6.5}$$

  We choose A such that the sign of $u(x)$ is opposite to the sign of $e(x)$ and its magnitude Δ' is less than Δ, viz

$$u(x_i)e(x_i) < 0 \ , \ \Delta' = \max_{0 \le x \le 1} |u(x)| < \Delta. \tag{6.6}$$

  We consider now the polynomial $p'(x) = p(x) + u(x)$ with corresponding error function $e'(x) = e(x) + u(x)$. The first condition $u(x_i)e(x_i) < 0$ yields directly to the conclusion that the error $e'(x)$ is less than $e(x)$ in the domain of the alternating extrema, whereas it is the condition $\Delta' < \Delta$ that yields to the conclusion that $e'(x)$ is less than $e(x)$ in the domain of the next largest extremum. Thus $e'(x) < e(x)$ throughout and hence $p'(x)$ is a better polynomial approximation.

- **Chebyshev's criterion is sufficient: If the error is extremal at exactly $n+2$ alternating points then the approximation is optimal.** Let us assume

that there is another polynomial $p'(x)$ which provides a better approximation. This means that the uniform norm $||e'||_\infty = \max_{0 \le x \le 1} |e'(x)| = \max_{0 \le x \le 1} |p'(x) - f(x)|$ is less than $||e||_\infty = \max_{0 \le x \le 1} |e(x)| = \max_{0 \le x \le 1} |p(x) - f(x)|$. Equivalently we must have at the $n + 2$ extrema of $e(x_i)$ the inequalities

$$|e'(x_i)| < |e(x_i)|. \tag{6.7}$$

By the requirement of continuity there must therefore exist $n + 1$ points z_i between the extrema at which we have

$$e'(z_i) = e(z_i). \tag{6.8}$$

This leads immediately to

$$p'(z_i) = p(z_i). \tag{6.9}$$

In other words, the polynomial $p'(x) - p(x)$ has $n + 1$ zeros, but since this polynomial is of degree n, it must vanish identically. Hence $p'(x) = p(x)$.

Chebyshev polynomials: The Chebyshev polynomial of degree n is defined by

$$T_n(\cos \theta) = \cos n\theta \leftrightarrow T_n(x) = \cos(n \cos^{-1} x). \tag{6.10}$$

We have the explicit expressions

$$T_0 = 1 \ , \ T_1 = x \ , \ T_2 = 2x^2 - 1 \ , \ ... \tag{6.11}$$

From the results $T_{n\pm1} = \cos n\theta \cos \theta \mp \sin n\theta \sin \theta$ we deduce the recursion relation

$$T_{n+1} = 2xT_n - T_{n-1}. \tag{6.12}$$

These polynomials are orthogonal in the interval $[-1, 1]$ with a weight $1/(1 - x^2)^{1/2}$, viz

$$\int_{-1}^{+1} \frac{dx}{\sqrt{1 - x^2}} T_i(x) T_j(x) = \frac{\pi}{2} \delta_{ij}. \tag{6.13}$$

$$\int_{-1}^{+1} \frac{dx}{\sqrt{1 - x^2}} T_0(x) T_0(x) = \pi. \tag{6.14}$$

The zeros of the polynomial $T_n(x)$ are given by

$$T_n(\cos \theta) = 0 \Rightarrow \cos n\theta = 0 \Rightarrow n\theta = (2k - 1)\frac{\pi}{2} \Rightarrow x = \cos \frac{(2k - 1)\pi}{2n} \ , \ k = 1, 2, ..., n. \tag{6.15}$$

Since the angle θ is in the interval between 0 and π. There are therefore n zeros.

The derivative of T_n is given by

$$\begin{aligned} \frac{d}{dx} T_n &= -n \frac{d}{dx} \cos^{-1} x . \sin(n \cos^{-1} x) \\ &= \frac{n}{\sqrt{1 - x^2}} \sin(n \cos^{-1} x). \end{aligned} \tag{6.16}$$

The extrema of the polynomial $T_n(x)$ are given by

$$\frac{d}{dx}T_n = 0 \Rightarrow \sin(n\theta) = 0 \Rightarrow n\theta = k\pi \Rightarrow x = \cos\frac{k\pi}{n} \ , \ k = 0, 2, ..., n. \tag{6.17}$$

There are $n+1$ extrema. The maxima satisfy $T_n(x) = 1$ while the minima satisfy $T_n(x) = -1$.

The Chebyshev polynomials satisfy also the following discrete orthogonality relation:

$$\sum_{k=1}^{m} T_i(x_k)T_j(x_k) = \frac{m}{2}\delta_{ij}. \tag{6.18}$$

$$\sum_{k=1}^{m} T_0(x_k)T_0(x_k) = m. \tag{6.19}$$

In the above two equations $i, j < m$ and x_k, $k = 1, ..., m$, are the m zeros of the Chebyshev polynomial $T_m(x)$.

Since $T_n(x)$ has $n + 1$ extrema which alternate in value between -1 and $+1$ for $-1 \leq x \leq 1$, and since the leading coefficient of $T_n(x)$ is 2^{n-1}; the polynomial $p_n(x) = x^n - 2^{1-n}T_n(x)$ is the best polynomial approximation of degree $n - 1$ with uniform weight to the function x^n over the interval $[-1, 1]$. This is because by construction the error $e_n(x) = p_n(x) - x^n = 2^{1-n}T_n(x)$ satisfies Chebyshevs criterion. The magnitude of the error is just $||e_n||_\infty = 2^{1-n} = 2e^{-n\ln 2}$, i.e. the error decreases exponentially with n.

Chebyshev approximation: Let $f(x)$ be an arbitrary function in the interval $[-1, +1]$. The Chebyshev approximation of this function can be constructed as follows. Let N be some large degree and x_k, $k = 1, ..., N$, be the zeros of the Chebyshev polynomial $T_N(x)$. The function $f(x)$ can be approximated by the polynomial of order N defined by

$$f_N(x) = \sum_{k=1}^{N} c_k T_{k-1}(x) - \frac{1}{2}c_1. \tag{6.20}$$

The coefficients c_k are given by

$$c_j = \frac{2}{N}\sum_{k=1}^{N} f(x_k)T_{j-1}(x_k). \tag{6.21}$$

This approximation is exact for x equal to all of the N zeros of $T_N(x)$. Indeed, we can show

$$\sum_{k=1}^{N} T_{l-1}(x_k)f_N(x_k) = \sum_{k=1}^{N} c_k \sum_{k=1}^{N} T_{l-1}(x_k)T_{k-1}(x_k) - \frac{1}{2}c_1\sum_{k=1}^{N} T_{l-1}(x_k)$$
$$= \frac{N}{2}c_l \ , \ l = 1, ..., N. \tag{6.22}$$

In other words,

$$f_N(x_k) = f(x_k). \tag{6.23}$$

For very large N, the polynomial f_N becomes very close to the function f. The polynomial f_N can be "gracefully", by using the words of [5], truncated to a lower degree $m << N$ by considering

$$f_m(x) = \sum_{k=1}^{m} c_k T_{k-1}(x) - \frac{1}{2}c_1. \tag{6.24}$$

The error for rapidly decreasing c_k, which is given by the difference between f_N and f_m, is dominated by $c_{m+1}T_m$ which has $m+1$ equal extrema distributed smoothly and uniformly in the interval $[-1, +1]$. Since the T's are bounded between -1 and $+1$ the total error is the sum of the neglected c_k, $k = m + 1, ..., N$. The Chebyshev approximation $f_m(x)$ is very close to the minimax polynomial which has the smallest maximum deviation from the function $f(x)$. Although the calculation of the Chebyshev polynomial $f_m(x)$ is very easy, finding the actual minimax polynomial is very difficult in practice.

Economization of power series: This will be explained by means of a specific example. We consider the function $f(x) = \sin x$. A quintic polynomial approximation of this function is given by the Taylor expansion

$$\sin x = x - \frac{x^3}{6} + \frac{x^5}{120}. \tag{6.25}$$

The domain of definition of $\sin x$ can be taken to be the interval $[-\pi, \pi]$. By making the replacement $x \longrightarrow x/\pi$ we convert the domain of definition $[-\pi, \pi]$ into the domain $[-1, 1]$, viz

$$\sin x = \pi x - \frac{\pi^3 x^3}{6} + \frac{\pi^5 x^5}{120}. \tag{6.26}$$

The error in the above quintic approximation is estimated by the first neglected term evaluated at the end points $x = \pm 1$, viz

$$\frac{\pi^7 x^7}{7!}|_{x=\pi} = 0.6. \tag{6.27}$$

The error in the 7th degree polynomial approximation can be found in the same way. We get in this case $\pi^9 x^9/9!|_{x=\pi} = 0.08$.

The monomials x^k can be given in terms of Chebyshev polynomials by the formulas

$$x^k = \frac{1}{2^{k-1}}\left[T_k(x) + \frac{k!}{1!(k-1)!}T_{k-2}(x) + \frac{k!}{2!(k-2)!}T_{k-4}(x) + ... + \frac{k!}{\frac{k-1}{2}!(k - \frac{k-1}{2})!}T_1(x)\right], \quad k \text{ odd} \tag{6.28}$$

$$x^k = \frac{1}{2^{k-1}}\left[T_k(x) + \frac{k!}{1!(k-1)!}T_{k-2}(x) + \frac{k!}{2!(k-2)!}T_{k-4}(x) + ... + \frac{k!}{\frac{k}{2}!(k - \frac{k}{2})!}T_0(x)\right], \quad k \text{ even} \tag{6.29}$$

For example

$$x = T_1(x). \tag{6.30}$$

$$x^3 = \frac{1}{4}[T_3(x) + 3T_1(x)].\tag{6.31}$$

$$x^5 = \frac{1}{16}[T_5(x) + 5T_3(x) + 10T_1(x)].\tag{6.32}$$

By substitution we get the result

$$
\begin{aligned}
\sin x &= \pi x - \frac{\pi^3 x^3}{6} + \frac{\pi^5 x^5}{120} \\
&= \frac{\pi(192 - 24\pi^2 + \pi^3)}{192}T_1 - \frac{\pi^3(16 - \pi^2)}{384}T_3 + \frac{\pi^5}{1920}T_5.
\end{aligned}\tag{6.33}
$$

Since $|T_n| \leq 1$, the last term is of the order of 0.16. This is smaller than the error found in the quintic approximation above. By truncating this term we obtain a cubic approximation of the sine function given by

$$\sin x = \frac{\pi(192 - 24\pi^2 + \pi^3)}{192}T_1 - \frac{\pi^3(16 - \pi^2)}{384}T_3\tag{6.34}$$

By substituting the Chebyshev polynomials by their expressions in terms of the x^k, and then changing back to the interval $[-\pi, +\pi]$, we obtain the cubic polynomial

$$\sin x = \frac{383}{384}x - \frac{5x^3}{32}.\tag{6.35}$$

By construction this cubic approximation is better than the above considered quintic approximation.

6.1.2 Minimax Rational Approximation and Remez Algorithm

Chebyshev theorem revisited: Chebyshev theorem can be extended to the case of minimax rational approximation of functions as follows. Again we consider a function f defined on the unit interval. For any given degree (n, d), there exists always a unique rational function $r_{n,d}$ of degree (n, d) which minimizes the error function given by

$$||e||_\infty = \max_{0 \leq x \leq 1}|e(x)| = \max_{0 \leq x \leq 1}|r_{n,d}(x) - f(x)|,\tag{6.36}$$

iff the error function $e(x)$ takes its maximum absolute value at at least $n + d + 2$ points on the unit interval, which may include the end points, and furthermore the sign of the error alternate between the successive extrema.

A simple proof of this theorem can be found in [4]. As it can be shown rational approximations are far more superior to polynomial ones since, for some functions and some intervals, we can achieve substantially higher accuracy with the same number of coefficients. However, it should also be appreciated that constructing the rational approximation is much more difficult than the polynomial one.

We will further explain this very important theorem following the discussion of [5]. The rational function $r_{n,d}$ is the ratio of two polynomials p_n and q_d of degrees n and d respectively, viz

$$r_{n,d}(x) = \frac{p_n(x)}{q_d(x)}. \tag{6.37}$$

The polynomials p_n and q_d can be written as

$$p_n(x) = \alpha_0 + \alpha_1 x + ... + \alpha_n x^n \ , \ q_d(x) = 1 + \beta_1 x + ... + \beta_d x^d. \tag{6.38}$$

We will assume that $r_{n,d}$ is non degenerate, i.e. it has no common polynomial factors in numerator and denominator. The error function $e(x)$ is the deviation of $r_{n,d}$ from $f(x)$ with a maximum absolute value e, viz

$$e(x) = r_{n,d}(x) - f(x) \ , \ e = \max_{0 \le x \le 1} |e(x)|. \tag{6.39}$$

Equation (6.37) can be rewritten as

$$\alpha_0 + \alpha_1 x + ... + \alpha_n x^n = (f(x) + e(x)) \left(1 + \beta_1 x + ... + \beta_d x^d \right). \tag{6.40}$$

There are $n + d + 1$ unknowns α_i and β_i plus one which is the error function $e(x)$. We can choose the rational approximation $r_{n,x}(x)$ to be exactly equal to the function $f(x)$ at $n + d + 1$ points x_i in the interval $[-1, 1]$, viz

$$f(x_i) = r_{n,d}(x_i) \ , \ e(x_i) = 0. \tag{6.41}$$

As a consequence the $n + d + 1$ unknowns α_i and β_i will be given by the $n + d + 1$ linear equations

$$\alpha_0 + \alpha_1 x_i + ... + \alpha_n x_i^n = f(x_i) \left(1 + \beta_1 x_i + ... + \beta_d x_i^d \right). \tag{6.42}$$

This can be solved any standard method such as LU decomposition.

The points x_i which are chosen in the interval $[-1, 1]$ will generically be such that there exists an extremum of the error function $e(x)$ in each subinterval $[x_i, x_{i+1}]$ plus two more extrema at the endpoints $\pm - 1$ for a total of $n + d + 1$ extrema. In general, the magnitudes of $r(x)$ at the extrema are not the same.

Alternatively, we can choose the rational approximation $r_{n,x}(x)$, at $n + d + 1$ points x_i, to be equal to $f(x) + y_i$ with some fixed values y_i of the error function $e(x)$. Equation (6.42) becomes

$$\alpha_0 + \alpha_1 x_i + ... + \alpha_n x_i^n = (f(x_i) + y_i) \left(1 + \beta_1 x_i + ... + \beta_d x_i^d \right). \tag{6.43}$$

If we choose the x_i to be the extrema of the error function $e(x)$ then the y_i will be exactly $\pm e$ where e is the maximal value of $|e(x)|$. We get then $n + d + 2$ (not $n + d + 1$) equations for the unknowns α_i, β_i and e given by

$$\alpha_0 + \alpha_1 x_i + ... + \alpha_n x_i^n = (f(x_i) \pm e) \left(1 + \beta_1 x_i + ... + \beta_d x_i^d \right). \tag{6.44}$$

The $\pm$ signs are due to the fact that successive extrema are alternating between $-e$ and $+e$. Although, this is not exactly a linear system since e enters non linearly, it can still be solved using for example methods such as Newton-Raphson.

Remez algorithm: A practical constructive approach to the minimax rational approximation of functions is given by Remez (or Remes) algorithm. This is a very difficult algorithm to get to work completely and properly and some people such as the authors [5] dislike it.

The Remez algorithm involves two nested iterations; the first on e and the second on the x_i's. Explicitly, it goes through the following steps:

- We choose or guess $n + d + 2$ initial values of the points x_i in the interval $[0, 1]$. The goal is to make these points converge to the alternating extrema discussed above.

- **The first iteration:** We keep the x_i's fixed and find the best rational approximation which goes through the points $(x_i, f(x_i) + (-1)^i \Delta)$. Towards this end, we need to solve the $n + d + 2$ equations

$$\alpha_0 + \alpha_1 x_i + ... + \alpha_n x_i^n = (f(x_i) + (-1)^i \Delta)) \left(1 + \beta_1 x_i + ... + \beta_d x_i^d \right). \qquad (6.45)$$

  The unknowns are α_i, β_i and Δ. We write this equation as

$$Mv = 0. \qquad (6.46)$$

  The $(n + d + 2)-$dimensional vector v is formed from the coefficients α_i, $i = 0, ..., n$ and β_j, $j = 0, ..., d$ with $\beta_0 = 1$. This linear system has a non trivial solution iff $\det M = 0$. This condition is a polynomial in Δ. The real roots of this polynomial are the allowed values of Δ and each one of them will correspond to a solution α_i and β_j. Each solution (α_i, β_j) corresponds to a certain rational approximation $r_{n,d}(x)$. We pick the solution which minimizes the error function.

- **The second iteration:** We keep e or Δ fixed and choose a new set of points x_i's which is the best alternating set for $e(x)$. This is done as follows. We choose an arbitrary partition $\{I_i\}$ of the interval $[0, 1]$ where I_i is such that $x_i \in I_i$. Then we choose a new set of points $x_i^{'}$ such that

$$x_i^{'} \in I_i \ , \ (-1)^i e(x_i^{'}) = \max_{x \in I_i} (-1)^i e(x_i). \qquad (6.47)$$

Several drawbacks of this algorithm are noted in [4,5]. Among these, we mention here the slow rate of convergence and the necessity of multiple precision arithmetic.

Zolotarevs Theorem: The case of rational approximations of the sign function, the square root and the inverse square root are known analytically in the sense that the coefficients of the optimal and unique Chebyshev rational approximations are known exactly. This result is due to Zolotarev.

The Numerical Recipes algorithm: A much simpler but very sloppy approximation, which is claimed in [5] to be "within a fraction of a least significant bit of the minimax one", and in which we try to bring the error not to zero as in the minimax case but to $\pm$ some consistent value, can be constructed as follows:

- We start from $n + d + 1$ values of x_i, or even a larger number of x_i, which are spaced approximately like the zeros of a higher order Chebyshev polynomials.

- We solve for α_i and β_j the linear system:

$$\alpha_0 + \alpha_1 x_i + ... + \alpha_n x_i^n = f(x_i)\left(1 + \beta_1 x_i + ... + \beta_d x_i^d\right). \tag{6.48}$$

In the case that the number of x_i's is larger than $n + d + 1$ we can use the singular value decomposition method to solve this system. The solution will provide our starting rational approximation $r_{n,d}(x)$. Compute $e(x_i)$ and e.

- We solve for α_i and β_j the linear system:

$$\alpha_0 + \alpha_1 x_i + ... + \alpha_n x_i^n = (f(x_i) \pm e)\left(1 + \beta_1 x_i + ... + \beta_d x_i^d\right). \tag{6.49}$$

The $\pm$ is chosen to be the sign of the observed error function $e(x_i)$ at each point x_i.

- We repeat the second step several times.

6.1.3 The Code "AlgRemez"

This code can be found in [6].

6.2 Conjugate Gradient Method

6.2.1 Construction

Our presentation of the conjugate gradient method in this section will follow the pedagogical note [1]. See also [2,3].

The basic problem: We consider a symmetric and positive definite $n \times n$ matrix A and an $n-$dimensional vector $\vec{v}$. The basic problem here is to solve for the $n-$dimensional vector $\vec{x}$ which satisfies the equation

$$A\vec{x} = \vec{v}. \tag{6.50}$$

We will find the solution by means of the conjugate gradient method which is an iterative algorithm suited for large sparse matrices A.

Principles of the method: The above problem is equivalent to finding the minimum $\vec{x}$ of the function $\Phi(\vec{x})$ defined by

$$\Phi(\vec{x}) = \frac{1}{2}\vec{x}A\vec{x} - \vec{x}\vec{v}. \tag{6.51}$$

The gradient of Φ is given by

$$\vec{\nabla}\Phi(\vec{x}) = A\vec{x} - \vec{v}. \tag{6.52}$$

This vanishes at the minimum. If not zero, it gives precisely the direction of steepest ascent of the surface Φ. The residual of the above set of equations is defined by

$$\vec{r} = -\vec{\nabla}\Phi(\vec{x}) = \vec{v} - A\vec{x}. \tag{6.53}$$

We will denote the n linearly independent vectors in the vector space to which $\vec{x}$ belongs by $\vec{p}^{(i)}$, $i = 1, ..., n$. They form a basis in this vector space. The vector $\vec{x}$ can be expanded as

$$\vec{x} = \sum_{i=1}^{n} s_i \vec{p}^{(i)} = P\vec{s}. \tag{6.54}$$

P is the $n \times n$ matrix of the linearly independent vectors $\vec{p}^{(i)}$, i.e. $P_{ij} = p_i^{(j)}$, and $\vec{s}$ is the vector of the coefficients s_i. Typically, we will start from a reference vector $\vec{x}_0$. Thus we write

$$\vec{x} = \vec{x}_0 + P\vec{s}. \tag{6.55}$$

The vectors $\vec{p}^{(i)}$ are $A-$conjugate to each other iff

$$\vec{p}^{(i)} A \vec{p}^{(j)} = 0 \ , \ i \neq j. \tag{6.56}$$

Thus we can write

$$P^T A P = D. \tag{6.57}$$

D is a diagonal matrix with elements given by

$$d_i = \vec{p}^{(i)} A \vec{p}^{(i)}. \tag{6.58}$$

The gradient of Φ takes the form

$$\vec{\nabla}\Phi = AP\vec{s} - \vec{r}_0 \ , \ \vec{r}_0 = \vec{v} - A\vec{x}_0. \tag{6.59}$$

Next, multiplication with the transpose P^T yields

$$\begin{aligned} P^T \vec{\nabla}\Phi &= P^T A P\vec{s} - P^T \vec{r}_0 \\ &= D\vec{s} - P^T \vec{r}_0. \end{aligned} \tag{6.60}$$

The solution to $\vec{\nabla}\Phi = 0$ is then

$$D\vec{s} - P^T \vec{r}_0 = 0 \Rightarrow s_i = \frac{\vec{p}^{(i)} \vec{r}_0}{\vec{p}^{(i)} A \vec{p}^{(i)}}. \tag{6.61}$$

The solution s_i found by globally minimizing Φ, also locally minimizes Φ along the direction $\vec{p}^{(i)}$. Thus starting from a vector $\vec{x}_0$ we obtain the solution

$$\vec{x}_1 = \vec{x}_0 + s_1 \vec{p}^{(1)} \ , \ s_1 = \frac{\vec{p}^{(1)} \vec{r}_0}{\vec{p}^{(1)} A \vec{p}^{(1)}} \ , \ \vec{r}_0 = \vec{v} - A\vec{x}_0. \tag{6.62}$$

This is the local minimum of Φ along a line from $\vec{x}_0$ in the direction $\vec{p}^{(1)}$. Indeed, we can check that

$$\vec{p}^{(1)}\vec{\nabla}\Phi = 0 \Rightarrow s_1 = \frac{\vec{p}^{(1)}\vec{r}_0}{\vec{p}^{(1)}A\vec{p}^{(1)}}. \tag{6.63}$$

The vector $\vec{r}_0$ is the first residual at the point $\vec{x}_0$ given by

$$\vec{\nabla}\Phi|_{\vec{x}_0} = -\vec{r}_0. \tag{6.64}$$

Next, starting from the vector $\vec{x}_1$ we obtain the solution

$$\vec{x}_2 = \vec{x}_1 + s_2\vec{p}^{(2)} \ , \ s_2 = \frac{\vec{p}^{(2)}\vec{r}_1}{\vec{p}^{(2)}A\vec{p}^{(2)}} \ , \ \vec{r}_1 = \vec{v} - A\vec{x}_1. \tag{6.65}$$

This is the local minimum of Φ along a line from $\vec{x}_1$ in the direction $\vec{p}^{(2)}$. The vector $\vec{r}_1$ is the new residual at the point $\vec{x}_1$, viz

$$\vec{\nabla}\Phi|_{\vec{x}_1} = -\vec{r}_1. \tag{6.66}$$

In general starting from the vector $\vec{x}_i$ we obtain the solution

$$\vec{x}_{i+1} = \vec{x}_i + s_{i+1}\vec{p}^{(i+1)} \ , \ s_{i+1} = \frac{\vec{p}^{(i+1)}\vec{r}_i}{\vec{p}^{(i+1)}A\vec{p}^{(i+1)}} \ , \ \vec{r}_i = \vec{v} - A\vec{x}_i. \tag{6.67}$$

This is the local minimum of Φ along a line from $\vec{x}_i$ in the direction $\vec{p}^{(i+1)}$. The vector $\vec{r}_i$ is the residual at the point $\vec{x}_i$, viz

$$\vec{\nabla}\Phi|_{\vec{x}_i} = -\vec{r}_i. \tag{6.68}$$

The residual vectors provide the directions of steepest descent of the function Φ at each iteration step. Thus if we know the conjugate vectors $\vec{p}^{(i)}$ we can compute the coefficients s_i and write down the solution $\vec{x}$. Typically, a good approximation of the true minimum of Φ may be obtained only after a small subset of the conjugate vectors are visited.

Choosing the conjugate vectors: The next step is to choose a set of conjugate vectors. An obvious candidate is the set of eigenvectors of the symmetric matrix A. However, in practice this choice is made as follows. Given that we have reached the iteration step i, i.e. we have reached the vector $\vec{x}_i$ which minimizes Φ in the direction $\vec{p}^{(i)}$, the search direction $\vec{p}^{(i+1)}$ will be naturally chosen in the direction of steepest descent of the function Φ at the point $\vec{x}_i$, which since A is positive definite is given by the direction of the residual $\vec{r}_i$, but conjugate to the previous search direction $\vec{p}^{(i)}$. We start then from the ansatz

$$\vec{p}^{(i+1)} = \vec{r}_i - \lambda\vec{p}^{(i)}. \tag{6.69}$$

This must be $A-$conjugate to $\vec{p}^{(i)}$, viz

$$\vec{p}^{(i)}A\vec{p}^{(i+1)} = 0. \tag{6.70}$$

This yields the value

$$\lambda = \frac{\vec{p}^{(i)} A \vec{r}_i}{\vec{p}^{(i)} A \vec{p}^{(i)}}. \tag{6.71}$$

The gradient $\vec{\nabla}\Phi$ at the point $\vec{x}_i$ is orthogonal to all previous search directions $\vec{p}^{(j)}$, $j < i$. Indeed, we compute

$$
\begin{aligned}
\vec{p}^{(j)}\vec{\nabla}\Phi|_{\vec{x}_i} &= \vec{p}^{(j)}\big(A\vec{x}_i - \vec{v}\big) \\
&= \vec{p}^{(j)}\big(A\vec{x}_0 + \sum_{k=1}^{i} s_k A\vec{p}^{(k)} - \vec{v}\big) \\
&= \vec{p}^{(j)}\big(\sum_{k=1}^{i} s_k A\vec{p}^{(k)} - \vec{r}_0\big) \\
&= \sum_{k=1}^{i} s_k \vec{p}^{(j)} A\vec{p}^{(k)} - \vec{p}^{(j)}\vec{r}_0 \\
&= s_j \vec{p}^{(j)} A\vec{p}^{(j)} - \vec{p}^{(j)}\vec{r}_0 \\
&= 0.
\end{aligned}
\tag{6.72}
$$

This formula works also for $j = i$. The gradients $\vec{\nabla}\Phi|_{\vec{x}_i}$ is also orthogonal to all previous gradients $\vec{\nabla}\Phi|_{\vec{x}_j}$, $j < i$. Indeed, we have

$$
\begin{aligned}
\vec{\nabla}\Phi|_{\vec{x}_j}\vec{\nabla}\Phi|_{\vec{x}_i} &= -\vec{r}_j \vec{\nabla}\Phi|_{\vec{x}_i} \\
&= -(\lambda \vec{p}^{(j)} + \vec{p}^{(j+1)})\vec{\nabla}\Phi|_{\vec{x}_i} \\
&= 0.
\end{aligned}
\tag{6.73}
$$

The first search direction can be chosen arbitrarily. We can for example choose $\vec{p}^{(1)} = \vec{r}_0 = -\vec{\nabla}\Phi|_{\vec{x}_0}$. The next search direction $\vec{p}^{(2)}$ is by construction $A-$conjugate to $\vec{p}^{(1)}$. At the third iteration step we obtain $\vec{p}^{(3)}$ which is $A-$conjugate to $\vec{p}^{(2)}$. The remaining question is whether $\vec{p}^{(3)}$ is $A-$conjugate to $\vec{p}^{(1)}$ or not. In general we would like to show that the search direction $\vec{p}^{(i)}$ generated at the ith iteration step, which is $A-$conjugate to $\vec{p}^{(i-1)}$, is also $A-$conjugate to all previously generated search directions $\vec{p}^{(j)}$, $j < i - 1$. Thus we need to show that

$$\vec{p}^{(j)} A\vec{p}^{(i)} = 0 \ , \ j < i - 1. \tag{6.74}$$

We compute

$$
\begin{aligned}
\vec{p}^{(j)} A\vec{p}^{(i)} &= \vec{p}^{(j)} A(\vec{r}_{i-1} - \lambda \vec{p}^{(i-1)}) \\
&= \vec{p}^{(j)} A\vec{r}_{i-1} - \lambda \vec{p}^{(j)} A\vec{p}^{(i-1)} \\
&= \frac{1}{s_j}(\vec{x}_j - \vec{x}_{j-1})A\vec{r}_{i-1} - \lambda \vec{p}^{(j)} A\vec{p}^{(i-1)} \\
&= \frac{1}{s_j}(-\vec{r}_j + \vec{r}_{j-1})\vec{r}_{i-1} - \lambda \vec{p}^{(j)} A\vec{p}^{(i-1)} \\
&= -\lambda \vec{p}^{(j)} A\vec{p}^{(i-1)} \\
&= 0.
\end{aligned}
\tag{6.75}
$$

Summary: Let us now summarize the main ingredients of the above algorithm. We have the following steps:

1) We choose a reference vector $\vec{x}_0$. We calculate the initial residual $\vec{r}_0 = \vec{v} - A\vec{x}_0$.

2) We choose the first search direction as $\vec{p}^{(1)} = \vec{r}_0$.

3) The first iteration towards the solution is

$$\vec{x}_1 = \vec{x}_0 + s_1\vec{p}^{(1)} \ , \ \ s_1 = \frac{\vec{p}^{(1)}\vec{r}_0}{\vec{p}^{(1)}A\vec{p}^{(1)}}. \tag{6.76}$$

4) The above three steps are iterated as follows:

$$\vec{r}_i = \vec{v} - A\vec{x}_i. \tag{6.77}$$

$$\vec{p}^{(i+1)} = \vec{r}_i - \lambda\vec{p}^{(i)} \ , \ \ \lambda = \frac{\vec{p}^{(i)}A\vec{r}_i}{\vec{p}^{(i)}A\vec{p}^{(i)}}. \tag{6.78}$$

$$s_{i+1} = \frac{\vec{p}^{(i+1)}\vec{r}_i}{\vec{p}^{(i+1)}A\vec{p}^{(i+1)}}. \tag{6.79}$$

$$\vec{x}_{i+1} = \vec{x}_i + s_{i+1}\vec{p}^{(i+1)}. \tag{6.80}$$

By using equations (6.77) and (6.80) we can show that equation (6.77) can be replaced by the equation

$$\vec{r}_i = \vec{r}_{i-1} - s_i A\vec{p}^{(i)} \tag{6.81}$$

Also we can derive the more efficient formulas

$$s_{i+1} = \frac{\vec{r}_i\vec{r}_i}{\vec{p}^{(i+1)}A\vec{p}^{(i+1)}} \ , \ \ \lambda = -\frac{\vec{r}_i\vec{r}_i}{\vec{r}_{i-1}\vec{r}_{i-1}}. \tag{6.82}$$

5) The above procedure continues as long as $|\vec{r}| \geq \epsilon$ where ϵ is some tolerance, otherwise stop.

6.2.2   The Conjugate Gradient Method as a Krylov Space Solver

We start this section by introducing some slight change of notation. By making the replacements $\vec{p}^{(i+1)} \longrightarrow \vec{p}_i$, $s_{i+1} \longrightarrow -\beta_i$, $\lambda \longrightarrow -\alpha_i$ the conjugate gradient algorithm will read

$$\vec{x}_{i+1} = \vec{x}_i - \beta_i\vec{p}_i \ , \ \ \beta_i = -\frac{\vec{r}_i\vec{r}_i}{\vec{p}_iA\vec{p}_i}. \tag{6.83}$$

$$\vec{r}_{i+1} = \vec{r}_i + \beta_iA\vec{p}_i. \tag{6.84}$$

$$\vec{p}_{i+1} = \vec{r}_{i+1} + \alpha_{i+1}\vec{p}_i \ , \ \alpha_{i+1} = \frac{\vec{r}_{i+1}\vec{r}_{i+1}}{\vec{r}_i\vec{r}_i}. \tag{6.85}$$

We start iterating from

$$\vec{x}_0 = 0 \ , \ \vec{r}_0 = \vec{v} - A\vec{x}_0 = \vec{v} \ , \ \vec{p}_0 = \vec{r}_0 = \vec{v}. \tag{6.86}$$

Remark now the following. We have

$$\vec{r}_0 = \vec{v} - A\vec{x}_0 \in \text{span}\{\vec{r}_0\}. \tag{6.87}$$

$$\vec{r}_1 = \vec{r}_0 + \beta_0 A\vec{r}_0 \in \text{span}\{\vec{r}_0, A\vec{r}_0\}. \tag{6.88}$$

$$\vec{r}_2 = \vec{r}_0 + \beta_0 A\vec{r}_0 + \beta_1 A(\vec{r}_0 + \beta_0 A\vec{r}_0) + \alpha_1\beta_1 A\vec{r}_0 \in \text{span}\{\vec{r}_0, A\vec{r}_0, A^2\vec{r}_0\}. \tag{6.89}$$

In general we will have

$$\vec{r}_n = P_n(A)\vec{r}_0 \in \text{span}\{\vec{r}_0, A\vec{r}_0, A^2\vec{r}_0, ..., A^n\vec{r}_0\}. \tag{6.90}$$

The $P_n(A)$ is a polynomial of degree n which obviously satisfy $P_n(0) = 1$. It is called the residual polynomial. On the other hand, the space $\text{span}\{\vec{r}_0, A\vec{r}_0, ..., A^n\vec{r}_0\}$ is called a Krylov subspace. Since the residues $\vec{r}_n$ are orthogonal the polynomials $P_n(A)$ are also orthogonal.

Similarly, we observe that

$$\vec{p}_0 = \vec{r}_0 \in \text{span}\{\vec{r}_0\}. \tag{6.91}$$

$$\vec{p}_1 = \vec{r}_1 + \alpha_1\vec{r}_0 \in \text{span}\{\vec{r}_0, A\vec{r}_0\}. \tag{6.92}$$

$$\vec{p}_2 = \vec{r}_2 + \alpha_2\vec{r}_1 + \alpha_1\alpha_2\vec{r}_0 \in \text{span}\{\vec{r}_0, A\vec{r}_0, A^2\vec{r}_0\}. \tag{6.93}$$

Thus in general

$$\vec{p}_n \in \text{span}\{\vec{r}_0, A\vec{r}_0, A^2\vec{r}_0, ..., A^n\vec{r}_0\}. \tag{6.94}$$

Also

$$\vec{x}_n \ = \ \vec{x}_0 - \sum_{i=0}^{n-1} \beta_i\vec{p}_i. \tag{6.95}$$

Thus

$$\vec{x}_n - \vec{x}_0 = Q_{n-1}(A)\vec{r}_0 \in \text{span}\{\vec{r}_0, A\vec{r}_0, A^2\vec{r}_0, ..., A^{n-1}\vec{r}_0\}. \tag{6.96}$$

The $Q_{n-1}(A)$ is a polynomial of exact degree $n - 1$. Hence both the conjugate gradient directions $\vec{p}_n$ and the solutions $\vec{x}_n - \vec{x}_0$ belong to various Krylov subspaces.

The conjugate gradient method is an example belonging to a large class of Krylov subspace methods. It is due to Hestenes and Stiefel [8] and it is the method of choice for solving linear systems that are symmetric positive definite or Hermitian positive definite. We conclude this section by the following two definitions.

Definition 1: Given a non-singular matrix $A \in \mathbf{C}^{n \times n}$ and a non-zero vector $r \in \mathbf{C}^n$, the nth Krylov (sub)space $\mathcal{K}_n(A, r)$ generated by A from r is

$$\mathcal{K}_n(A, r) = \text{span}(r, Ar, ..., A^{n-1}r). \tag{6.97}$$

Definition 2: A standard Krylov space method for solving a linear system $Ax = b$ is an iterative method which starts from some initial guess x_0 with residual $r_0 = b - Ax_0$ and then generates better approximations x_n to the exact solution x_* as follows

$$x_n - x_0 = Q_{n-1}(A)r_0 \in \mathcal{K}_n(A, r_0) = \text{span}\{r_0, Ar_0, A^2 r_0, ..., A^{n-1}r_0\}. \tag{6.98}$$

The residuals r_n of the above so-called Krylov space solver will satisfy

$$r_n = P_n(A)r_0 \in \mathcal{K}_{n+1}(A, r_0) = \text{span}\{r_0, Ar_0, A^2 r_0, ..., A^n r_0\}. \tag{6.99}$$

It is not difficult to show that

$$P_n(A) = 1 - AQ_{n-1}(A). \tag{6.100}$$

6.2.3   The Multi-Mass Conjugate Gradient Method

The goal now is to solve a multi-mass linear system of the form

$$(A + \sigma)\vec{x} = \vec{v}. \tag{6.101}$$

By a direct application of the conjugate gradient method we get the solution

$$\vec{x}_{i+1}^{\sigma} = \vec{x}_i^{\sigma} - \beta_i^{\sigma}\vec{p}_i^{\sigma} \ , \ \beta_i^{\sigma} = -\frac{\vec{r}_i^{\sigma}\vec{r}_i^{\sigma}}{\vec{p}_i^{\sigma}(A + \sigma)\vec{p}_i^{\sigma}}. \tag{6.102}$$

$$\vec{r}_{i+1}^{\sigma} = \vec{r}_i^{\sigma} + \beta_i^{\sigma}(A + \sigma)\vec{p}_i^{\sigma}. \tag{6.103}$$

$$\vec{p}_{i+1}^{\sigma} = \vec{r}_{i+1}^{\sigma} + \alpha_{i+1}^{\sigma}\vec{p}_i^{\sigma} \ , \ \alpha_{i+1}^{\sigma} = \frac{\vec{r}_{i+1}^{\sigma}\vec{r}_{i+1}^{\sigma}}{\vec{r}_i^{\sigma}\vec{r}_i^{\sigma}}. \tag{6.104}$$

$$\vec{x}_0^{\sigma} = 0 \ , \ \vec{r}_0^{\sigma} = \vec{v}^{\sigma} - (A + \sigma)\vec{x}_0^{\sigma} = \vec{v} \ , \ \vec{p}_0^{\sigma} = \vec{r}_0^{\sigma} = \vec{v}. \tag{6.105}$$

There is clearly a loop over σ which could be very expensive in practice. Fortunately we can solve, by following [7], the above multi-mass linear system using only a single set of vector-matrix operations as follows. First we note that

$$\vec{r}_{i+1}^{\sigma} = \vec{r}_i^{\sigma} + \beta_i^{\sigma}(A + \sigma)\vec{p}_i^{\sigma} = P_{i+1}^{\sigma}(A + \sigma)\vec{r}_0^{\sigma} \in \mathcal{K}_{i+2}(A + \sigma, \vec{r}_0). \tag{6.106}$$

As discussed before the polynomials P_{i+1}^{σ} are orthogonal in $A + \sigma$. This follows from the fact that $\vec{r}_{i+1}^{\sigma} \perp \vec{r}_i^{\sigma}$ and as a consequence

$$P_{i+1}^{\sigma}(A + \sigma)\vec{r}_0^{\sigma} \perp \mathcal{K}_{i+1}(A + \sigma, \vec{r}_0). \tag{6.107}$$

However, we have the obvious and fundamental fact that

$$\mathcal{K}_{i+1}(A + \sigma, \vec{r}_0) = \mathcal{K}_{i+1}(A, \vec{r}_0). \tag{6.108}$$

In other words, the polynomials P_{i+1}^σ are orthogonal in A as well. We must therefore have

$$P_{i+1}^\sigma(A + \sigma) = \zeta_{i+1}^\sigma P_{i+1}(A). \tag{6.109}$$

The polynomials P_{i+1}^σ are thus of a shifted structure. By the identity (6.100) it follows that the polynomials Q_i^σ are not of a shifted structure. This single observation will allow us to reduce the problem to a single set of vector-matrix operations.

By multiplying equation (6.104) by $\beta_{i+1}^\sigma(A + \sigma)$ and using equation (6.103) we get

$$\beta_{i+1}^\sigma(A + \sigma)\vec{p}_{i+1}^\sigma = \beta_{i+1}^\sigma(A + \sigma)\vec{r}_{i+1}^\sigma + \frac{\beta_{i+1}^\sigma \alpha_{i+1}^\sigma}{\beta_i^\sigma}(\vec{r}_{i+1}^\sigma - \vec{r}_i^\sigma). \tag{6.110}$$

By substitution in equation (6.103) we get the 3−term recurrence given by

$$\vec{r}_{i+2}^\sigma = (1 + \frac{\beta_{i+1}^\sigma \alpha_{i+1}^\sigma}{\beta_i^\sigma})\vec{r}_{i+1}^\sigma + \beta_{i+1}^\sigma(A + \sigma)\vec{r}_{i+1}^\sigma - \frac{\beta_{i+1}^\sigma \alpha_{i+1}^\sigma}{\beta_i^\sigma}\vec{r}_i^\sigma. \tag{6.111}$$

By using (6.109) we obtain

$$\zeta_{i+2}^\sigma \vec{r}_{i+2} = (1 + \frac{\beta_{i+1}^\sigma \alpha_{i+1}^\sigma}{\beta_i^\sigma})\zeta_{i+1}^\sigma \vec{r}_{i+1} + \beta_{i+1}^\sigma(A + \sigma)\zeta_{i+1}^\sigma \vec{r}_{i+1} - \frac{\beta_{i+1}^\sigma \alpha_{i+1}^\sigma}{\beta_i^\sigma}\zeta_i^\sigma \vec{r}_i. \tag{6.112}$$

However, the no-sigma recurrence reads

$$\vec{r}_{i+2} = (1 + \frac{\beta_{i+1}\alpha_{i+1}}{\beta_i})\vec{r}_{i+1} + \beta_{i+1}A\vec{r}_{i+1} - \frac{\beta_{i+1}\alpha_{i+1}}{\beta_i}\vec{r}_i. \tag{6.113}$$

By comparing the $A\vec{r}_{i+1}$ terms we obtain

$$\beta_n^\sigma = \beta_n \frac{\zeta_{n+1}^\sigma}{\zeta_n^\sigma}. \tag{6.114}$$

By comparing the $\vec{r}_i$ terms and also using the above result we obtain

$$\alpha_n^\sigma = \alpha_n \frac{\zeta_n^\sigma \beta_{n-1}^\sigma}{\zeta_{n-1}^\sigma \beta_{n-1}}. \tag{6.115}$$

By comparing the $\vec{r}_{i+1}$ terms and also using the above two results we find after some calculation

$$\zeta_{n+1}^\sigma = \frac{\zeta_n^\sigma \zeta_{n-1}^\sigma \beta_{n-1}}{\alpha_n \beta_n(\zeta_{n-1}^\sigma - \zeta_n^\sigma) + \zeta_{n-1}^\sigma \beta_{n-1}(1 - \sigma\beta_n)}. \tag{6.116}$$

Let us conclude by summarizing the main ingredients of this algorithm. These are:

1. We start from

$$\vec{x} = \vec{x}_0^\sigma = 0 \; , \; \vec{r}_0 = \vec{r}_0^\sigma = \vec{v} \; , \; \vec{p} = \vec{p}_0^\sigma = \vec{v}. \tag{6.117}$$

By setting $i = -1$ in (6.112) we see that we must also start from

$$\alpha_0 = \alpha_0^\sigma = 0 \; , \; \beta_{-1} = \beta_{-1}^\sigma = 1 \; , \; \zeta_0^\sigma = \zeta_{-1}^\sigma = 1. \tag{6.118}$$

2. We solve the no-sigma problem (we start from $n = 0$):

$$\beta_n = -\frac{\vec{r}_n \vec{r}_n}{\vec{p}_n A \vec{p}_n}$$
$$\vec{x}_{n+1} = \vec{x}_n - \beta_n \vec{p}_n. \tag{6.119}$$

$$\vec{r}_{n+1} = \vec{r}_n + \beta_n A \vec{p}_n. \tag{6.120}$$

$$\alpha_{n+1} = \frac{\vec{r}_{n+1} \vec{r}_{n+1}}{\vec{r}_n \vec{r}_n}$$
$$\vec{p}_{n+1} = \vec{r}_{n+1} + \alpha_{n+1} \vec{p}_n. \tag{6.121}$$

3. We generate solutions of the sigma problems by the relations (we start from $n = 0$):

$$\zeta_{n+1}^\sigma = \frac{\zeta_n^\sigma \zeta_{n-1}^\sigma \beta_{n-1}}{\alpha_n \beta_n (\zeta_{n-1}^\sigma - \zeta_n^\sigma) + \zeta_{n-1}^\sigma \beta_{n-1}(1 - \sigma \beta_n)}. \tag{6.122}$$

$$\beta_n^\sigma = \beta_n \frac{\zeta_{n+1}^\sigma}{\zeta_n^\sigma}. \tag{6.123}$$

$$\vec{x}_{n+1}^\sigma = \vec{x}_n^\sigma - \beta_n^\sigma \vec{p}_n^\sigma. \tag{6.124}$$

$$\vec{r}_{n+1}^\sigma = \zeta_{n+1}^\sigma \vec{r}_{n+1}. \tag{6.125}$$

$$\alpha_{n+1}^\sigma = \alpha_{n+1} \frac{\zeta_{n+1}^\sigma \beta_n^\sigma}{\zeta_n^\sigma \beta_n}. \tag{6.126}$$

$$\vec{p}_{n+1}^\sigma = \vec{r}_{n+1}^\sigma + \alpha_{n+1}^\sigma \vec{p}_n^\sigma. \tag{6.127}$$

Remark how the residues are generated directly from the residues of the no-sigma problem.

4. The above procedure continues as long as $|\vec{r}| \geq \epsilon$ where ϵ is some tolerance, otherwise stop. Thus

$$|\vec{r}| \geq \epsilon \ , \ \text{continue.} \tag{6.128}$$

We finally note that in the case of a hermitian matrix, i.e. $A^+ = A$, we must replace in the above formulas the transpose by hermitian conjugation. For example, we replace $\vec{p}_n^T A \vec{p}_n$ by $\vec{p}_n^+ A \vec{p}$. The rest remains unchanged.

Bibliography

[1] E. Thompson, "The Conjugate Gradient Method: A Tutorial Note".

[2] Martin H. Gutknecht, "A Brief Introduction to Krylov Space Methods for Solving Linear Systems".

[3] L. Chen, "Iterative Methods Based on Krylov Space".

[4] A. D. Kennedy, "Approximation theory for matrices," Nucl. Phys. Proc. Suppl. **128C**, 107 (2004) [hep-lat/0402037].

[5] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, "Numerical Recipes in FORTRAN: The Art of Scientific Computing," ISBN-9780521430647.

[6] M. A. Clark and A. D. Kennedy, https://github.com/mikeaclark/AlgRemez, 2005.

[7] B. Jegerlehner, "Krylov space solvers for shifted linear systems," hep-lat/9612014.

[8] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," J. Res. Nat. Bureau Standards, 49:409435, 1952.

Chapter 7

Monte Carlo Simulation of Fermion Determinants

As it is well known, simulation of fermion determinants and Pfaffians is crucial to lattice QCD, but as it trurns out, it is also crucial to all supersymmetric matrix models and quantum mechanical matrix models encountered or needed in matrix field theory, matrix/fuzzy geometry and matrix formulation of noncommutative geometry, supersymmetry and strings. As done before in this part of the book, the theoretical background will be kept to a minimum, otherwise we will stray too far afield, and we will mostly focus on practical problems. The main reference for this chapter is [1, 2]. See also [3, 4]. For some subtle details of the rational hybrid Monte Carlo algorithm see [5–8].

7.1   The Dirac Operator

The basic problem we want to solve in this section is to simulate the partition function of $\mathcal{N} = 1$ supersymmetric Yang-Mills matrix model in $d = 4$ dimensions given by

$$Z_{\mathrm{YM}} = \int \prod_{\mu=1}^{4} X_\mu \ d\bar{\theta}d\theta \ \exp\left(\bar{\theta}\big(i[X_4, ..] + \sigma_a[X_a, ..] + \xi\big)\theta \right) \exp(-S_{\mathrm{BYM}}[X]). \qquad (7.1)$$

$$S_{\mathrm{BYM}} \quad = \quad -\frac{N\gamma}{4} \sum_{\mu,\nu=1}^{4} Tr[X_\mu, X_\nu]^2. \qquad (7.2)$$

The parameter γ will be set to one and we may add to the bosonic Yang-Mills action a Chern-Simons term and a harmonic oscillator term with parameters α and m^2 respectively. The spinors $\bar{\theta}$ and θ are two independent complex two-component Weyl spinors. They contain the same number of degrees of Freedom as the four-component real Majorana spinors in four dimensions. The scalar curvature or fermion mass parameter is given by ξ. The above theory is only supersymmetric for a restricted set of values of the parameters γ, α, m^2 and ξ. See [11] and references therein for a discussion of this matter.

We have considered above the Dirac operator given by

$$\mathcal{D} = iX_4 - iX_4^R + \sigma_a X_a - \sigma_a X_a^R + \xi. \qquad (7.3)$$

The determinant of this Dirac operator is positive definite since the eigenvalues come in complex conjugate pairs [1]. In $d = 6$ and $d = 10$ the determinant is, however, complex valued which presents a serious obstacle to numerical evaluation. In these three cases, i.e. for $d = 4, 6, 10$, the supersymmetric path integral is well behaved. In $d = 3$ the supersymmetric path integral is ill defined and only the bosonic "quenched" approximation makes sense. The source of the divergence lies in the so-called flat directions, i.e. the set of commuting matrices. See [10] and references therein.

It is possible to rewrite the Dirac action in the following form (with $X_{34} = X_3 + iX_4$ and $X_\pm = X_1 \pm iX_2$)

$$
\begin{aligned}
Tr\bar{\theta}\mathcal{D}\theta &= Tr\left[\bar{\theta}_1(X_{34} + \xi)\theta_1 + \bar{\theta}_1 X_-\theta_2 + \bar{\theta}_2 X_+\theta_1 + \bar{\theta}_2(-X_{34}^+ + \xi)\theta_2\right] \\
&\quad - Tr\left[X_{34}\bar{\theta}_1\theta_1 + X_-\bar{\theta}_1\theta_2 + X_+\bar{\theta}_2\theta_1 - X_{34}^+\bar{\theta}_2\theta_2\right]. \qquad (7.4)
\end{aligned}
$$

We expand the $N \times N$ matrices θ_1, θ_2 and $\bar{\theta}_1, \bar{\theta}_2$ as

$$\theta_\alpha = \sum_{A=1}^{N^2} \theta_\alpha^A T^A \ , \ \bar{\theta}_\alpha = \sum_{\bar{A}=1}^{N^2} \bar{\theta}_\alpha^A T^A. \qquad (7.5)$$

The $N \times N$ matrices T^A are defined by

$$(T^A)_{ij} = \delta_{ii_A}\delta_{jj_A} \ , \ A = N(i_A - 1) + j_A. \qquad (7.6)$$

Then we find that

$$Tr\bar{\theta}\mathcal{D}\theta = \bar{\chi}_1\mathcal{M}_{11}\chi_1 + \bar{\chi}_1\mathcal{M}_{12}\chi_2 + \bar{\chi}_2\mathcal{M}_{21}\chi_2 + \bar{\chi}_2\mathcal{M}_{22}\chi_2. \qquad (7.7)$$

The N^2−dimensional vectors χ_1, χ_2 and $\bar{\chi}_1$, $\bar{\chi}_2$ are defined by $(\chi_\alpha)_A = \theta_\alpha^A$ and $(\bar{\chi}_\alpha)_A = \bar{\theta}_\alpha^A$. The matrices $\mathcal{M}_{\alpha\beta}^{AB}$ are $N^2 \times N^2$ defined by

$$(\mathcal{M}_{11})^{AB} = TrT^A(X_{34} + \xi)T^B - TrX_{34}T^AT^B. \qquad (7.8)$$

$$(\mathcal{M}_{12})^{AB} = TrT^AX_-T^B - TrX_-T^AT^B. \qquad (7.9)$$

$$(\mathcal{M}_{21})^{AB} = TrT^AX_+T^B - TrX_+T^AT^B. \qquad (7.10)$$

$$(\mathcal{M}_{22})^{AB} = TrT^A(-X_{34}^+ + \xi)T^B + TrX_{34}^+T^AT^B. \qquad (7.11)$$

We remark that

$$TrT^AXT^B - TrXT^AT^B = X_{j_Ai_B}\delta_{i_Aj_B} - X_{j_Bi_A}\delta_{j_Ai_B}. \qquad (7.12)$$

$$Tr(T^A)^+T^B = \delta_{i_A i_B}\delta_{j_A j_B} = \delta_{AB} \ , \ TrT^A T^B = \delta_{j_A i_B}\delta_{j_B i_A} = \delta_{\bar{A}B}. \qquad (7.13)$$

In the above two equations $\bar{A}$ and B are such that

$$\bar{A} = N(j_A - 1) + i_A \ , \ B = N(i_B - 1) + j_B. \qquad (7.14)$$

In summary, the Dirac operator in terms of the $2N^2-$dimensional vectors χ and $\bar{\chi}$ becomes

$$Tr\bar{\theta}\mathcal{D}\theta = \bar{\chi}\mathcal{M}\chi. \qquad (7.15)$$

Next, we observe that the trace parts of the matrices X_a drop from the partition function. Thus the measure should read $\int dX_a \delta(TrX_a)$ instead of simply $\int dX_a$. Similarly, we observe that if we write $\theta = \theta_0 + \eta\mathbf{1}$, then the trace part η will decouple from the rest since

$$Tr\bar{\theta}\Big(i[X_4,..] + \sigma_a[X_a,..] + \xi\Big)\theta = Tr\bar{\theta}_0\Big(i[X_4,..] + \sigma_a[X_a,..] + \xi\Big)\theta_0 + \xi\bar{\eta}\eta. \qquad (7.16)$$

Hence, the constant fermion modes η_α can also be integrated out from the partition function and thus we should consider the measure $\int d\theta d\bar{\theta}\delta(Tr\theta_\alpha)\delta(Tr\bar{\theta}_\alpha)$ instead of $\int d\theta d\bar{\theta}$. These facts should be taken into account in the numerical study. We are thus led to consider the partition function

$$Z_{\mathrm{YM}} = \int \prod_{\mu=1}^4 dX_\mu \ \delta(TrX_\mu) \ \det\mathcal{D} \ \exp\big(-S_{\mathrm{BYM}}[X]\big). \qquad (7.17)$$

The determinant is given by

$$\begin{aligned}
\det\mathcal{D} &= \int d\theta d\bar{\theta}\delta(Tr\theta_\alpha)\delta(Tr\bar{\theta}_\alpha)\exp\big(Tr\bar{\theta}\mathcal{D}\theta\big)\\
&= \int d\chi d\bar{\chi}\delta\Big(\sum_{A=1}^{N^2}(\chi_\alpha)_A\delta_{i_A j_A}\Big)\delta\Big(\sum_{A=1}^{N^2}(\bar{\chi}_\alpha)_A\delta_{i_A j_A}\Big)\exp\big(\bar{\chi}\mathcal{M}\chi\big)\\
&= \int d\chi' d\bar{\chi}'\exp\big(\bar{\chi}'\mathcal{M}'\chi'\big). \qquad (7.18)
\end{aligned}$$

The vectors χ'_α, $\bar{\chi}'_\alpha$ are $(N^2 - 1)-$dimensional. The matrix $\mathcal{M}'$ is $2(N^2 - 1) \times 2(N^2 - 1)$ dimensional, and it is given by

$$\mathcal{M}'^{A'B'}_{\alpha\beta} = \mathcal{M}^{A'B'}_{\alpha\beta} - \mathcal{M}^{N^2 B'}_{\alpha\beta}\delta_{i_{A'}j_{A'}} - \mathcal{M}^{A'N^2}_{\alpha\beta}\delta_{i_{B'}j_{B'}} + \mathcal{M}^{N^2 N^2}_{\alpha\beta}\delta_{i_{A'}j_{A'}}\delta_{i_{B'}j_{B'}}. \qquad (7.19)$$

We remark that

$$\mathcal{M}^{N^2 N^2}_{\alpha\beta} = \xi\delta_{\alpha\beta}. \qquad (7.20)$$

Thus we must have

$$\det\mathcal{D} = \det\mathcal{M}'. \qquad (7.21)$$

The partition function thus reads

$$Z_{\text{YM}} = \int \prod_{\mu=1}^{4} dX_\mu \ \delta(TrX_\mu) \ \exp\left(-S_{\text{YM}}[X]\right). \tag{7.22}$$

$$S_{\text{YM}}[X] = S_{\text{BYM}}[X] + V[X] \ , \ V = -\ln\det\mathcal{M}'. \tag{7.23}$$

We will need

$$\frac{\partial S_{\text{BYM}}}{\partial(X_\mu)_{ij}(t)} = -N\gamma \sum_{\nu=1}^{4} [X_\nu, [X_\mu, X_\nu]]_{ji}$$

$$= -N\gamma\left(2X_\nu X_\mu X_\nu - X_\nu^2 X_\mu - X_\mu X_\nu^2\right)_{ji}. \tag{7.24}$$

The determinant is real positive definite since the eigenvalues are paired up. Thus, we can introduce the positive definite operator Δ by

$$\Delta = (\mathcal{M}')^{+}\mathcal{M}'. \tag{7.25}$$

The action V can be rewritten as

$$V = -\frac{1}{2}\ln\det\Delta. \tag{7.26}$$

The leap-frog algorithm for this problem is given by

$$(P_\mu)_{ij}(n+\frac{1}{2}) = (P_\mu)_{ij}(n) - \frac{\delta t}{2}\left[\frac{\partial S_{\text{BYM}}}{\partial(X_\mu)_{ij}}(n) + (V_\mu)_{ij}(n)\right]. \tag{7.27}$$

$$(X_\mu)_{ij}(n+1) = (X_\mu)_{ij}(n) + \delta t(P_\mu)_{ji}(n+\frac{1}{2}). \tag{7.28}$$

$$(P_\mu)_{ij}(n+1) = (P_\mu)_{ij}(n+\frac{1}{2}) - \frac{\delta t}{2}\left[\frac{\partial S_{\text{BYM}}}{\partial(X_\mu)_{ij}}(n+1) + (V_\mu)_{ij}(n+1)\right]. \tag{7.29}$$

The effect of the determinant is encoded in the matrix

$$(V_\mu)_{ij} = \frac{\partial V}{\partial(X_\mu)_{ij}}$$

$$= -\frac{1}{2}Tr_{\text{ad}}\Delta^{-1}\frac{\partial\Delta}{\partial(X_\mu)_{ij}}. \tag{7.30}$$

From (7.23) and (7.30) we see that we must compute the inverse and the determinant of the Dirac operator at each hybrid Monte Carlo step. However, the Dirac operator is an $\mathcal{N}\times\mathcal{N}$ matrix where $\mathcal{N} = 2N^2 - 2$. This is proportional to the number of degrees of freedom. Since the computation of the determinant requires $O(\mathcal{N}^3)$ operations at best, through Gaussian elimination, we see that the computational effort of the above algorithm will be $O(N^6)$. Recall that the computational effort of the bosonic theory is $O(N^3)$[1] .

---

[1]Compare also with field theory in which the number of degrees of freedom is proportional to the volume, the computational effort of the bosonic theory is $O(V)$ while that of the full theory, which includes a determinant, is $O(V^2)$.

7.2   Pseudo-Fermions and Rational Approximations

We introduce pseudo-fermions in the usual way as follows. The determinant can be rewritten in the form

$$
\begin{aligned}
\det \mathcal{D} = \det \mathcal{M}^{'} &= (\det \Delta)^{\frac{1}{2}} \\
&= \int d\phi^+ d\phi \ \exp(-\phi^+ \Delta^{-1/2}\phi). \qquad (7.31)
\end{aligned}
$$

Since $\mathcal{D}$, $\mathcal{M}'$ and Δ are $\mathcal{N} \times \mathcal{N}$ matrices organized as 2×2 matrices, with components given by $\hat{\mathcal{N}} \times \hat{\mathcal{N}}$ matrices where $\hat{\mathcal{N}} = \mathcal{N}/2$, the vectors ϕ^+ and ϕ can be thought of as two-component spinors where each component is given by an $\hat{\mathcal{N}}-$dimensional vector. We will write

$$
\phi = \begin{pmatrix} \phi_1 \\ \phi_2 \end{pmatrix} , \ \phi^+ = \begin{pmatrix} \phi_1^+ & \phi_2^+ \end{pmatrix}. \qquad (7.32)
$$

These are precisely the pseudo-fermions. They are complex-valued instead of Grassmann-valued degrees of freedom, and that is why they are pseudo-fermions, with a positive definite Laplacian and thus they can be sampled in Monte Carlo simulations in the usual way.

Furthermore, we will use the so-called rational approximation, which is why the resulting hybrid Monte Carlo is termed rational, which allows us to write

$$
(\det \Delta)^{\frac{1}{2}} = \int d\phi^+ d\phi \ \exp(-\phi^+ r^2(\Delta)\phi). \qquad (7.33)
$$

The rational approximation $r(x)$ is given by

$$
x^{-1/4} \simeq r(x) = a_0 + \sum_{\sigma=1}^{M} \frac{a_\sigma}{x + b_\sigma}. \qquad (7.34)
$$

The parameters a_0, a_σ, b_σ and M are real positive numbers which can be optimized for any strictly positive range such as $\epsilon \leq x \leq 1$. This point was discussed at great length previously.

Thus the pseudo-fermions are given by a heatbath, viz

$$
\phi = r^{-1}(\Delta)\xi, \qquad (7.35)
$$

where ξ is given by the Gaussian noise $P(\xi) = \exp(-\xi^+\xi)$. We write

$$
\phi = \left(c_0 + \sum_{\sigma=1}^{M} \frac{c_\sigma}{\Delta + d_\sigma} \right)\xi. \qquad (7.36)
$$

By using a different rational approximation $\bar{r}(x)$, in order to avoid double inversion (see below), we rewrite the original path integral in the form

$$
Z_{\text{YM}} = \int \prod_{\mu=1}^{4} dX_\mu \int d\phi^+ d\phi \ \delta(TrX_\mu) \ \exp\left(- S_{\text{BYM}}[X] \right) \ \exp(-\phi^+ \bar{r}(\Delta)\phi). \qquad (7.37)
$$

The new rational approximation is defined by

$$x^{-1/2} \simeq \bar{r}(x) = a_0 + \sum_{\sigma=1}^{M} \frac{a_\sigma}{x + b_\sigma}. \tag{7.38}$$

The full action becomes

$$S_{\text{YM}} = S_{\text{BYM}}[X] + V[X]. \tag{7.39}$$

The potential is given in this case by

$$
\begin{aligned}
V &= \phi^+ \bar{r}(\Delta)\phi \\
&= a_0 \phi^+ \phi + \sum_{\sigma=1}^{M} a_\sigma \phi^+ (\Delta + b_\sigma)^{-1}\phi \\
&= a_0 \phi^+ \phi + \sum_{\sigma=1}^{M} a_\sigma \phi^+ G_\sigma = a_0 \phi_\alpha^+ \phi_\alpha + \sum_{\sigma=1}^{M} a_\sigma \phi_\alpha^+ G_{\sigma\alpha} \\
&= a_0 \phi^+ \phi + \sum_{\sigma=1}^{M} a_\sigma G_\sigma^+ \phi = a_0 \phi_\alpha^+ \phi_\alpha + \sum_{\sigma=1}^{M} a_\sigma G_{\sigma\alpha}^+ \phi_\alpha.
\end{aligned}
\tag{7.40}
$$

This can be rewritten compactly as

$$V = W_\alpha \phi_\alpha \ , \ W_\alpha = a_0(\phi_\alpha^*)_A + \sum_{\sigma=1}^{M} a_\sigma (G_{\sigma\alpha}^*)_A. \tag{7.41}$$

The vectors (pseudo-fermions) G_σ are defined by

$$G_\sigma = (\Delta + b_\sigma)^{-1}\phi. \tag{7.42}$$

We introduce a fictitious time parameter t and a Hamiltonian H given by

$$
\begin{aligned}
H &= \frac{1}{2} Tr P_\mu^2 + Q^+ Q + S_{\text{YM}} \\
&= \frac{1}{2} Tr P_\mu^2 + Q_\alpha^+ Q_\alpha + S_{\text{YM}}.
\end{aligned}
\tag{7.43}
$$

The equation of motion associated with the matrix ϕ is given by

$$
\begin{aligned}
-(\dot{Q}_\alpha)_A &= \frac{\partial H}{\partial (\phi_\alpha)_A} \\
&= \frac{\partial V}{\partial (\phi_\alpha)_A} \\
&= a_0(\phi_\alpha^*)_A + \sum_{\sigma=1}^{M} a_\sigma (G_{\sigma\alpha}^*)_A \\
&\equiv (W_\alpha)_A.
\end{aligned}
\tag{7.44}
$$

$$
\begin{aligned}
(\dot{\phi}_\alpha)_A &= \frac{\partial H}{\partial (Q_\alpha)_A} \\
&\equiv (Q_\alpha^*)_A.
\end{aligned}
\tag{7.45}
$$

This last equation is equivalent to

$$(\dot{\phi}^*_\alpha)_A \equiv (Q_\alpha)_A. \tag{7.46}$$

The leap-frog algorithm for this part of the problem is given by

$$(Q_\alpha)_A(n + \frac{1}{2}) = (Q_\alpha)_A(n) - \frac{\delta t}{2}(W_\alpha)_A(n). \tag{7.47}$$

$$(\phi_\alpha)_A(n + 1) = (\phi_\alpha)_A(n) + \delta t(Q^*_\alpha)_A(n + \frac{1}{2}). \tag{7.48}$$

$$(Q_\alpha)_A(n + 1) = (Q_\alpha)_A(n + \frac{1}{2}) - \frac{\delta t}{2}(W_\alpha)_A(n + 1). \tag{7.49}$$

The first set of equations of motion associated with the matrices X_μ are given by

$$
\begin{aligned}
-(\dot{P}_\mu)_{ij} &= \frac{\partial H}{\partial (X_\mu)_{ij}} \\
&= \frac{\partial S_{\mathrm{BYM}}}{\partial (X_\mu)_{ij}} + \frac{\partial V}{\partial (X_\mu)_{ij}} \\
&= \frac{\partial S_{\mathrm{BYM}}}{\partial (X_\mu)_{ij}} - \sum_{\sigma=1}^{M} a_\sigma G^+_{\sigma\alpha} \frac{\partial \Delta_{\alpha\beta}}{\partial (X_\mu)_{ij}} G_{\sigma\beta}.
\end{aligned} \tag{7.50}
$$

The effect of the determinant is now encoded in the matrix (the force)

$$(V_\mu)_{ij} = -\sum_{\sigma=1}^{M} a_\sigma G^+_{\sigma\alpha} \frac{\partial \Delta_{\alpha\beta}}{\partial (X_\mu)_{ij}} G_{\sigma\beta}. \tag{7.51}$$

The second set of equations associated with the matrices X_μ are given by

$$
\begin{aligned}
(\dot{X}_\mu)_{ij} &= \frac{\partial H}{\partial (P_\mu)_{ij}} \\
&= (P_\mu)_{ji}.
\end{aligned} \tag{7.52}
$$

The leap-frog algorithm for this part of the problem is given by the equations (7.27), (7.28) and (7.29) with the appropriate re-interpretation of the meaning of $(V_\mu)_{ij}$.

7.3   More on The Conjugate-Gradient

7.3.1   Multiplication by $\mathcal{M}'$ and $(\mathcal{M}')^+$

Typically we will need to find x', given v, which solves the linear system

$$(\Delta + b)x' = v. \tag{7.53}$$

We will use the conjugate gradient method to do this. The product $\Delta x'$ involves the products $\mathcal{M}' x'$ and $(\mathcal{M}')^+ y'$, viz

$$y' = \mathcal{M}' x' \leftrightarrow (y'_\alpha)_{A'} = \mathcal{M}'^{A'B'}_{\alpha\beta} (x'_\beta)_{B'}. \tag{7.54}$$

$$z' = (\mathcal{M}')^+ y' \leftrightarrow (z'_\alpha)_{A'} = (\mathcal{M}'^*_{\beta\alpha})^{B'A'} (y'_\beta)_{B'}. \tag{7.55}$$

Multiplication by $\mathcal{M}'$:   By using (7.19) we have

$$
\begin{aligned}
(y'_\alpha)_{A'} &= \mathcal{M}'^{A'B'}_{\alpha\beta}(x'_\beta)_{B'} \\
&= \mathcal{M}^{A'B'}_{\alpha\beta}(x'_\beta)_{B'} - \mathcal{M}^{N^2B'}_{\alpha\beta}\delta_{i_{A'}j_{A'}}(x'_\beta)_{B'} - \mathcal{M}^{A'N^2}_{\alpha\beta}\delta_{i_{B'}j_{B'}}(x'_\beta)_{B'} + \mathcal{M}^{N^2N^2}_{\alpha\beta}\delta_{i_{A'}j_{A'}}\delta_{i_{B'}j_{B'}}(x'_\beta)_{B'}.
\end{aligned}
\tag{7.56}
$$

Recall that the primed indices run from 1 to $N^2 - 1$ while unprimed indices run from 1 to N^2. We introduce then

$$
\begin{aligned}
(y_\alpha)_A &= \mathcal{M}^{AB}_{\alpha\beta}(x_\beta)_B \\
&= \mathcal{M}^{AB'}_{\alpha\beta}(x_\beta)_{B'} + \mathcal{M}^{AN^2}_{\alpha\beta}(x_\beta)_{N^2}.
\end{aligned}
\tag{7.57}
$$

We define

$$
(x_\beta)_{B'} = (x'_\beta)_{B'} \ , \ (x_\beta)_{N^2} = -(x'_\beta)_{B'}\delta_{i_{B'}j_{B'}}.
\tag{7.58}
$$

Thus

$$
(y_\alpha)_A = \mathcal{M}^{AB'}_{\alpha\beta}(x'_\beta)_{B'} - \mathcal{M}^{AN^2}_{\alpha\beta}(x'_\beta)_{B'}\delta_{i_{B'}j_{B'}}.
\tag{7.59}
$$

The next definition is obviously then

$$
(y'_\alpha)_{A'} = (y_\alpha)_{A'} - (y_\alpha)_{N^2}\delta_{i_{A'}j_{A'}}.
\tag{7.60}
$$

This leads immediately to

$$
(y'_\alpha)_{A'} = \mathcal{M}^{A'B'}_{\alpha\beta}(x'_\beta)_{B'} - \mathcal{M}^{A'N^2}_{\alpha\beta}(x'_\beta)_{B'}\delta_{i_{B'}j_{B'}} - \mathcal{M}^{N^2B'}_{\alpha\beta}(x'_\beta)_{B'} + \mathcal{M}^{N^2N^2}_{\alpha\beta}(x'_\beta)_{B'}\delta_{i_{B'}j_B}
\tag{7.61}
$$

This is precisely (7.56).

Next we introduce the $N \times N$ matrices $\hat{x}_\alpha$, $\hat{y}_\alpha$ associated with the vectors x_α and y_α by the relations

$$
\hat{x}_\alpha = \sum_{A=1}^{N^2}(x_\alpha)_A T^A \ , \ \hat{y}_\alpha = \sum_{A=1}^{N^2}(y_\alpha)_A T^A.
\tag{7.62}
$$

Thus

$$
(x_\alpha)_{\bar{A}} = Tr\hat{x}_\alpha T^A = (\hat{x}_\alpha)_{j_A i_A} \ , \ (y_\alpha)_{\bar{A}} = Tr\hat{y}_\alpha T^A = (\hat{y}_\alpha)_{j_A i_A}.
\tag{7.63}
$$

And

$$
(x_\alpha)_A = Tr\hat{x}_\alpha(T^A)^+ = (\hat{x}_\alpha)_{i_A j_A} \ , \ (y_\alpha)_A = Tr\hat{y}_\alpha(T^A)^+ = (\hat{y}_\alpha)_{i_A j_A}.
\tag{7.64}
$$

We verify that

$$
\mathcal{M}^{AB}_{\alpha\beta}(x_\beta)_B = TrT^A(\mathcal{D}\hat{x})_\alpha.
\tag{7.65}
$$

By comparing with

$$
(y_\alpha)_A = TrT^A(\hat{y})_\alpha,
\tag{7.66}
$$

we get

$$\hat{y}^T = \mathcal{D}\hat{x}. \tag{7.67}$$

We recall the Dirac operator

$$\mathcal{D} = \begin{pmatrix} X_{34} - X_{34}^R + \xi & X_- - X_-^R \\ X_+ - X_+^R & -X_{34}^+ + (X_{34}^R)^+ + \xi \end{pmatrix}. \tag{7.68}$$

Thus $\hat{y}^T = \mathcal{D}\hat{x}$ is equivalent to

$$(\hat{y}_1)_{ij} = (\mathcal{D}_{1\alpha}\hat{x}_\alpha)_{ji} = [X_{34}, \hat{x}_1]_{ji} + [X_-, \hat{x}_2]_{ji} + \xi(\hat{x}_1)_{ji}. \tag{7.69}$$

$$(\hat{y}_2)_{ij} = (\mathcal{D}_{2\alpha}\hat{x}_\alpha)_{ji} = -[X_{34}^+, \hat{x}_2]_{ji} + [X_+, \hat{x}_1]_{ji} + \xi(\hat{x}_2)_{ji}. \tag{7.70}$$

For completeness we remark

$$(y_\alpha)_A^* \mathcal{M}_{\alpha\beta}^{AB}(x_\beta)_B = Tr\hat{y}_\alpha^*(\mathcal{D}\hat{x})_\alpha. \tag{7.71}$$

Multiplication by $(\mathcal{M}')^+$: As before the calculation of

$$(z_\alpha')_{A'} = (\mathcal{M}_{\beta\alpha}'^*)^{B'A'}(y_\beta')_{B'} \tag{7.72}$$

can be reduced to the calculation of

$$(z_\alpha)_A = (\mathcal{M}_{\beta\alpha}^*)^{BA}(y_\beta)_B, \tag{7.73}$$

with the definitions

$$(y_\beta)_{B'} = (y_\beta')_{B'} \ , \ (y_\beta)_{N^2} = -(y_\beta')_{B'}\delta_{i_{B'}j_{B'}}. \tag{7.74}$$

$$(z_\alpha')_{A'} = (z_\alpha)_{A'} - (z_\alpha)_{N^2}\delta_{i_{A'}j_{A'}}. \tag{7.75}$$

The next step is to note that

$$\mathcal{M}_{\beta\alpha}^{*BA}(y_\beta)_B = TrT^A(\mathcal{D}^+\hat{y})_\alpha. \tag{7.76}$$

The hermitian conjugate of the Dirac operator is defined by the relation

$$\mathcal{D}^+ = -\begin{pmatrix} X_{34}^* - (X_{34}^R)^* + \xi & X_+^* - (X_+^R)^* \\ X_-^* - (X_-^R)^* & -X_{34}^T + (X_{34}^R)^T + \xi \end{pmatrix}. \tag{7.77}$$

Hence

$$\hat{z}^T = \mathcal{D}^+\hat{y}. \tag{7.78}$$

Equivalently

$$(\hat{z}_1)_{ij} = (\mathcal{D}_{1\alpha}^+\hat{y}_\alpha)_{ji} = -[X_{34}^*, \hat{y}_1]_{ji} - [X_+^*, \hat{y}_2]_{ji} + \xi(\hat{y}_1)_{ji}. \tag{7.79}$$

$$(\hat{z}_2)_{ij} = (\mathcal{D}_{2\alpha}^+\hat{y}_\alpha)_{ji} = [X_{34}^T, \hat{y}_2]_{ji} - [X_-^*, \hat{y}_1]_{ji} + \xi(\hat{y}_2)_{ji}. \tag{7.80}$$

7.3.2   The Fermionic Force

Also we will need to compute explicitly in the molecular dynamics part the fermionic force (with $(\mathcal{M}'^{+})_{\alpha\beta} = (\mathcal{M}'_{\beta\alpha})^{+}$)

$$
\begin{aligned}
(V_\mu)_{ij} &= -\sum_{\sigma=1}^{M} a_\sigma G_{\sigma\alpha}^{+} \frac{\partial \Delta_{\alpha\beta}}{\partial (X_\mu)_{ij}} G_{\sigma\beta} \\
&= -\sum_{\sigma=1}^{M} a_\sigma G_{\sigma\alpha}^{+} \frac{\partial (\mathcal{M}'_{\beta\alpha})^{+}}{\partial (X_\mu)_{ij}} F_{\sigma\beta} - \sum_{\sigma=1}^{M} a_\sigma F_{\sigma\beta}^{+} \frac{\partial \mathcal{M}'_{\beta\alpha}}{\partial (X_\mu)_{ij}} G_{\sigma\alpha} \\
&= -\sum_{\sigma=1}^{M} a_\sigma \left(F_{\sigma\beta}^{+} \frac{\partial \mathcal{M}'_{\beta\alpha}}{\partial (X_\mu)_{ij}^{*}} G_{\sigma\alpha} \right)^{*} - \sum_{\sigma=1}^{M} a_\sigma F_{\sigma\beta}^{+} \frac{\partial \mathcal{M}'_{\beta\alpha}}{\partial (X_\mu)_{ij}} G_{\sigma\alpha}.
\end{aligned}
\tag{7.81}
$$

The vectors $F_{\sigma\alpha}$ and $F_{\sigma\alpha}^{+}$ are defined by

$$
F_{\sigma\alpha} = \mathcal{M}'_{\alpha\beta} G_{\sigma\beta} \ , \ F_{\sigma\alpha}^{+} = G_{\sigma\beta}^{+} (\mathcal{M}'_{\alpha\beta})^{+}.
\tag{7.82}
$$

We can expand the bosonic matrices X_μ similarly to the fermionic matrices as

$$
X_\mu = \sum_{A=1}^{N^2} X_\mu^{A} T^{A}.
\tag{7.83}
$$

Equivalently

$$
(X_\mu)_{i_A j_A} = X_\mu^{A} \ , \ A = N(i_A - 1) + j_A.
\tag{7.84}
$$

Reality of the bosonic matrices gives

$$
(X_\mu)_{i_A j_A}^{*} = X_\mu^{\bar{A}} = (X_\mu^{A})^{*} \ , \ \bar{A} = N(j_A - 1) + i_A.
\tag{7.85}
$$

Hence we have

$$
\begin{aligned}
V_\mu^{A} &\equiv (V_\mu)_{i_A j_A} \\
&= -\sum_{\sigma=1}^{M} a_\sigma \left(F_{\sigma\beta}^{+} \frac{\partial \mathcal{M}'_{\beta\alpha}}{\partial X_\mu^{\bar{A}}} G_{\sigma\alpha} \right)^{*} - \sum_{\sigma=1}^{M} a_\sigma F_{\sigma\beta}^{+} \frac{\partial \mathcal{M}'_{\beta\alpha}}{\partial X_\mu^{A}} G_{\sigma\alpha} \\
&= -\sum_{\sigma=1}^{M} a_\sigma (\mathcal{T}_{\sigma\mu}^{\bar{A}})^{*} - \sum_{\sigma=1}^{M} a_\sigma \mathcal{T}_{\sigma\mu}^{A}.
\end{aligned}
\tag{7.86}
$$

The definition of $\mathcal{T}_{\sigma\mu}^{A}$ is obviously given by

$$
\mathcal{T}_{\sigma\mu}^{A} = F_{\sigma\beta}^{+} \frac{\partial \mathcal{M}'_{\beta\alpha}}{\partial X_\mu^{A}} G_{\sigma\alpha}.
\tag{7.87}
$$

For simplicity we may denote the derivations with respect to X_μ^{A} and $X_\mu^{\bar{A}}$ by ∂ and $\bar{\partial}$ respectively. As before we introduce the vectors in the full Hilbert space:

$$
(\tilde{G}_{\sigma\alpha})_{B'} = (G_{\sigma\alpha})_{B'} \ , \ (\tilde{G}_{\sigma\alpha})_{N^2} = -(G_{\sigma\alpha})_{B'} \delta_{i_{B'} j_{B'}}.
\tag{7.88}
$$

$$(\tilde{F}_{\sigma\alpha})_{B'} = (F_{\sigma\alpha})_{B'} \;,\; (\tilde{F}_{\sigma\alpha})_{N^2} = -(F_{\sigma\alpha})_{B'}\delta_{i_{B'}j_{B'}}. \tag{7.89}$$

A straightforward calculation gives

$$(F_{\sigma\beta}^*)_{A'}(\mathcal{M}_{\beta\alpha}')^{A'B'}(G_{\sigma\alpha})_{B'} = (\tilde{F}_{\sigma\beta}^*)_A(\mathcal{M}_{\beta\alpha})^{AB}(\tilde{G}_{\sigma\alpha})_B. \tag{7.90}$$

$$(F_{\sigma\beta}^*)_{A'}\partial(\mathcal{M}_{\beta\alpha}')^{A'B'}(G_{\sigma\alpha})_{B'} = (\tilde{F}_{\sigma\beta}^*)_A\partial(\mathcal{M}_{\beta\alpha})^{AB}(\tilde{G}_{\sigma\alpha})_B. \tag{7.91}$$

Thus

$$\mathcal{T}_{\sigma\mu}^A = \tilde{F}_{\sigma\beta}^+\frac{\partial\mathcal{M}_{\beta\alpha}}{\partial X_\mu^A}\tilde{G}_{\sigma\alpha}. \tag{7.92}$$

Explicitly we have

$$\mathcal{T}_{\sigma\mu}^A = (\tilde{F}_{\sigma\beta}^*)_C\frac{\partial\mathcal{M}_{\beta\alpha}^{CD}}{\partial X_\mu^A}(\tilde{G}_{\sigma\alpha})_D. \tag{7.93}$$

We use the result

$$\frac{\partial\mathcal{M}_{\beta\alpha}^{CD}}{\partial X_\mu^A} = Tr\frac{\partial M_{\beta\alpha}}{\partial X_\mu^A}[T^D,T^C], \tag{7.94}$$

where

$$M_{11} = X_{34} \;,\; M_{12} = X_- \;,\; M_{21} = X_+ \;,\; M_{22} = -X_{34}^+. \tag{7.95}$$

We also introduce the matrices $\hat{F}$ and $\hat{G}$ given by

$$\hat{F}_\alpha = \sum_{A=1}^{N^2}(\tilde{F}_\alpha)_A T^A \;,\; \hat{G}_\alpha = \sum_{A=1}^{N^2}(\tilde{G}_\alpha)_A T^A. \tag{7.96}$$

The reverse of these equations is

$$(\tilde{F}_\alpha)_A = Tr\hat{F}_\alpha(T^A)^+ \;,\; (\tilde{G}_\alpha)_A = Tr\hat{G}_\alpha(T^A)^+. \tag{7.97}$$

We use also the identity

$$\sum_A(T^A)_{ij}(T^A)_{kl}^+ = \delta_{il}\delta_{jk}. \tag{7.98}$$

A direct calculation yields then the fundamental results

$$\mathcal{T}_{\sigma\mu}^A = Tr\frac{\partial M_{\beta\alpha}}{\partial X_\mu^A}[\hat{G}_{\sigma\alpha},\hat{F}_{\sigma\beta}^*] \;,\; \mathcal{T}_{\sigma\mu}^{\bar{A}} = Tr\frac{\partial M_{\beta\alpha}}{\partial X_\mu^{\bar{A}}}[\hat{G}_{\sigma\alpha},\hat{F}_{\sigma\beta}^*]. \tag{7.99}$$

Explicitly we have

$$\mathcal{T}_{\sigma1}^A = [\hat{G}_{\sigma1},\hat{F}_{\sigma2}^*]_{j_Ai_A} + [\hat{G}_{\sigma2},\hat{F}_{\sigma1}^*]_{j_Ai_A} \;,\; \mathcal{T}_{\sigma1}^{\bar{A}} = [\hat{G}_{\sigma1},\hat{F}_{\sigma2}^*]_{i_Aj_A} + [\hat{G}_{\sigma2},\hat{F}_{\sigma1}^*]_{i_Aj_A}. \tag{7.100}$$

$$\mathcal{T}_{\sigma2}^A = -i[\hat{G}_{\sigma1},\hat{F}_{\sigma2}^*]_{j_Ai_A} + i[\hat{G}_{\sigma2},\hat{F}_{\sigma1}^*]_{j_Ai_A} \;,\; \mathcal{T}_{\sigma2}^{\bar{A}} = -i[\hat{G}_{\sigma1},\hat{F}_{\sigma2}^*]_{i_Aj_A} + i[\hat{G}_{\sigma2},\hat{F}_{\sigma1}^*]_{i_Aj_A}. \tag{7.101}$$

$$\mathcal{T}_{\sigma3}^A = [\hat{G}_{\sigma1},\hat{F}_{\sigma1}^*]_{j_Ai_A} - [\hat{G}_{\sigma2},\hat{F}_{\sigma2}^*]_{j_Ai_A} \;,\; \mathcal{T}_{\sigma3}^{\bar{A}} = [\hat{G}_{\sigma1},\hat{F}_{\sigma1}^*]_{i_Aj_A} - [\hat{G}_{\sigma2},\hat{F}_{\sigma2}^*]_{i_Aj_A}. \tag{7.102}$$

$$\mathcal{T}_{\sigma4}^A = i[\hat{G}_{\sigma1},\hat{F}_{\sigma1}^*]_{j_Ai_A} + i[\hat{G}_{\sigma2},\hat{F}_{\sigma2}^*]_{j_Ai_A} \;,\; \mathcal{T}_{\sigma4}^{\bar{A}} = i[\hat{G}_{\sigma1},\hat{F}_{\sigma1}^*]_{i_Aj_A} + i[\hat{G}_{\sigma2},\hat{F}_{\sigma2}^*]_{i_Aj_A}. \tag{7.103}$$

7.4   The Rational Hybrid Monte Carlo Algorithm

7.4.1   Statement

In summary the rational hybrid Monte Carlo algorithm in the present setting consists of the following steps:

1. **Initialization of** X: Start X (the fundamental field in the problem) from a random configuration.

2. **Initialization of Other Fields**:

   - Start P (the conjugate field to X) from a Gaussian distribution according to the probability $\exp(-TrP_\mu^2/2)$. Both X_μ and P_μ are hermitian $N \times N$ matrices.
   - Start ξ from a Gaussian distribution according to the probability $\exp(-\xi^+\xi)$.
   - Calculate ϕ (the pseudo-fermion) using the formula (7.36). This is done using the conjugate gradient method (see below). The coefficients c and d are computed using the Remez algorithm from the rational approximation of $x^{1/4}$.
   - Start Q (the conjugate field to ϕ) from a Gaussian distribution according to the probability $\exp(-Q^+Q)$. The spinors Q_α and ϕ_α, as well as ξ_α, are $(N^2 - 1)-$dimensional complex vectors.

3. **Molecular Dynamics**: This consists of two parts:

   - **Pseudo-Fermion**: We evolve the pseudo-fermion ϕ and its conjugate field Q using the Hamilton equations (7.47), (7.48) and (7.49). This is done using the conjugate gradient method which, given the input ϕ, computes as output the spinors G_σ given by equation (7.42) and the spinor W given by equation (7.44). On the other hand, in the initialization step above we call the conjugate gradient method with input ξ to obtain the output $\phi = W^*$. Here and below, the coefficients a and b are computed using the Remez algorithm from the rational approximation of $x^{-1/2}$.
   - **Gauge Field**: We evolve X_μ and P_μ using the Hamilton equations (7.27), (7.28) and (7.29). This requires the calculation of the boson contribution to the force given by equation (7.24) and the fermion contribution given by equation (7.51). The numerical evaluation of the fermion force is quite involved and uses the formula (7.86). This requires, among other things, the calculation of the spinors G_σ and $F_\sigma = \mathcal{M}'G_\sigma$ using the conjugate gradient.

4. **Metropolis Step**: After obtaining the solution $(X(T), P(T), \phi(T), Q(T))$ of the molecular dynamics evolution starting from the initial configuration $(X(0), P(0), \phi(0), Q(0))$ we compute the resulting variation ΔH in the Hamiltonian. The new configuration is accepted with probability

$$\text{probability} = \min(1, \exp(-\Delta H)). \qquad (7.104)$$

5. **Iteration:** Repeat starting from 2.

6. **Other Essential Ingredients**: The two other essential ingredients of this algorithm are:

(a) **Conjugate Gradient**: This plays a fundamental role in this algorithm. The multimass Krylov space solver employed here is based on the fundamental equations (6.117)-(6.128). This allows us to compute the G_σ for all σ given by equation (7.42) at once. The multiplication by Δ is done in two steps: first we multiply by $\mathcal{M}'$ then we multiply by $(\mathcal{M}')^+$. This is done explicitly by reducing (7.54) to (7.69)+(7.70) and reducing (7.55) to (7.79)+(7.80). Here, we obviously need to convert between a given traceless vector and its associated matrix and vice versa. The relevant equations are (7.58), (7.60) and (7.64).

(b) **Remez Algorithm**: This is discussed at length in the previous chapter. We only need to re-iterate here that the real coefficients c, d, for the rational approximation of $x^{1/4}$, and a and b, for the rational approximation of $x^{-1/2}$, as well as the integer M are obtained using the Remez algorithm of [9]. The integer M is supposed to be determined separately for each function by requiring some level of accuracy whereas the range over which the functions are approximated by their rational approximations should be determined on a trial and error basis by inspecting the spectrum of the Dirac operator.

7.4.2   Preliminary Tests

1. **The rational approximations**: The first thing we need to do is to fix the parameters a, b, c and d of the rational approximations by invoking the Remez algorithm. For a tolerance equal 10^{-4} and over the interval $[0.0004, 1]$ with precision 40 we have found that the required degrees of the rational approximations, for $x^{-1/2}$ and $x^{1/4}$, are $M = 6$ and $M_0 = 5$ respectively; M is the minimum value for which the uniform norm $|r - f|_\infty = \mathbf{max}|r - f|$ is smaller than the chosen tolerance. We can plot these rational approximations versus the actual functions to see whether or not these approximations are sufficiently good over the fixed range.

2. **The conjugate gradient**: The conjugate gradient is a core part in this algorithm and it must be checked thoroughly. A straightforward check is to verify that $(\Delta + b_\sigma)G_\sigma = \phi$ for all values of σ. We must be careful that the matrix-vector multiplication $\Delta.G_\sigma$ does not vanish. Thus the no-sigma problem should be defined, not with zero mass $b_\sigma = 0$, but with the smallest possible value of the mass b_σ which presumably corresponds to the least convergent linear system. In the results included below we fix the tolerance of the conjugate gradient at 10^{-5}.

3. **The decoupled theory**: This is the theory in which the gauge field $(X_\mu)_{ij}$ and the pseudo-fermion field ϕ_α^A are completely decoupled from each other. This is then equivalent to the bosonic theory. This is expected to be obtained for sufficiently large values of the fermion mass ξ. In this theory the fermion field behaves exactly as a harmonic oscillator. The decoupled theory can also be obtained, both in the molecular dynamics part and the hybrid Monte Carlo part which includes in addition

the metropolis step, by setting

$$c_0 = \frac{1}{\sqrt{a_0}} \ , \ a_i = c_i = 0. \tag{7.105}$$

In this case the pseudo-fermions decouple from the gauge fields and behave as harmonic oscillators with period $T = 2\pi$. The corresponding action should then be periodic with period $T = \pi$.

4. **The molecular dynamics**: We can run the molecular dynamics on its own to verify the prediction of the decoupled theory. In general, it is also useful to monitor the classical dynamics for its own interest and monitor in particular the systematic error due to the non-conservation of the Hamiltonian.

   In the molecular dynamics we need to fix the time step dt and the number of iterations n. Thus we run the molecular dynamics for a time interval $T = n.dt$. We choose $dt = 10^{-3}$ and $n = 2^{13} - 2^{14}$. Some results with $N = 4$ are included in figures (7.1) and (7.2). We remark that the drift in the Hamiltonian becomes pronounced as $\xi \longrightarrow 0$. This systematic error will be canceled by the Metropolis step (see below).

   We can use the molecular dynamics to obtain an estimation of the range of the rational approximations needed as follows. Starting from $\xi = 0$, we increase the value of ξ until the behavior of the theory becomes that of the decoupled (bosonic) theory. The value of ξ at which this happens will be taken as an estimation of the range. In the above example (figures (7.1) and (7.2)) we observe that the pseudo-fermion sector becomes essentially a harmonic oscillator around the value $\xi = 10$. Thus a reasonable range should be taken between 0 and 10.

5. **The metropolis step**: In general two among the three parameters of the molecular dynamics (the time step dt, the number of iterations n and the time interval $T = ndt$) should be optimized in such a way that the acceptance rate is fixed, for example, between 70 and 90 per cent. We fix n and optimize dt along the line discussed in previous chapters. We make, for every N, a reasonable guess for the value of the number of iterations n, based on trial and error, and then work with that value throughout. For example, for N between $N = 4$ and $N = 8$, we found the value $n = 10$, to be sufficiently reasonable.

   Typically, we run $T_{\text{ther}} + T_{\text{meas}}$ Monte Carlo steps where thermalization is supposed to occur within the first T_{ther} steps which are discarded while measurements are performed on a sample consisting of the subsequent T_{meas} configurations. We choose, for $N = 4 - 8$, $T_{\text{ther}} = 2^{11}$ and $T_{\text{meas}} = 2^{13}$. We do not discuss in the following auto-correlation issues while error bars are computed using the jackknife method. As always, we generate our random numbers using the algorithm ran2. Some thermalized results for $N = 4, 8$ and $\alpha = m^2 = \xi = 0$ are shown on figure (7.3).

   There are two powerful tests (exact analytic results) which can be used to calibrate the simulations. We must have the identities:

   - We must have on general grounds the identity:

   $$< \exp(-\Delta H) > = 1. \tag{7.106}$$

- We must also have the Schwinger-Dyson identity:

$$< 4\gamma \text{YM} > + < 3\alpha \text{CS} > + < 2m^2 \text{HO} > + < \xi \text{COND} >= (d+2)(N^2 - 1). \quad (7.107)$$

We have included for completeness the effects of a Chern-Simons term and a harmonic oscillator term in the bosonic action. This identity is a generalization of (2.35) where the definition of the condensation COND can be found in [11]. This identity follows from the invariance of the path integral (7.17) under the translations $X_\mu \longrightarrow X_\mu + \epsilon X_\mu$. For the flat space supersymmetric model for which $\xi = 0$ the above Schwinger-Dyson identity reduces to

$$< 4\gamma \text{YM} > + < 3\alpha \text{CS} > + < 2m^2 \text{HO} >= (d+2)(N^2 - 1). \quad (7.108)$$

As an illustration some expectation values as functions of α for $N = 4$ and $m^2 = \xi = 0$ are shown on figure (7.4).

6. **Emergent geometry:** We observe from the graph of TrX_μ^2 that something possibly interesting happens around $\alpha \sim 1.2$. In fact, this is the very dramatic phenomena of emergent geometry which is known to occur in these models when there is a non-zero mass term (here the Chern-Simons term) included. This can be studied in great detail using as order parameters the eigenvalues distributions of X_4 and X_a. In the matrix or Yang-Mills phase (small values of α) the matrices X_μ are nearly commuting with eigenvalues distributed uniformly inside a solid ball with a parabolic eigenvalues distributions, or a generalization thereof, whereas in the fuzzy sphere phase (large values of α) the matrix X_4 decouples from X_a and remains distributed as in the matrix phase, while the matrices X_a will be dominated by fluctuations around the $SU(2)$ generators in the spin $(N - 1)/2$ irreducible representation.

7. **Code:** The attached code can be used to study the above emergent geometry effect, and many other issues, in great detail. On an intel dual core E4600 processor (2.40GHz) running Ubuntu 14.04 LTS this codes goes as N^5.

Figure 7.1:

Figure 7.2:

Figure 7.3:

Figure 7.4:

7.5 Other Related Topics

Many other important topics, requiring techniques similar to the ones discussed in this chapter, and which have been studied extensively by the Japan group, includes:

1. **IKKT models:** The extension of the problem to higher dimensions; for example $d = 6$; but in particular $d = 10$ which is the famous IKKT model which provides a non-perturbative definition of string theory, is the first obvious generalization. However, the determinant in these cases is complex-valued which makes its numerical evaluation very involved.

2. **Cosmological Yang-Mills matrix models:** In recent years a generalization from Euclidean Yang-Mills matrix models to Minkowski signature was carried out with dramatic, interesting and novel consequences for cosmological models. The problem with the complex-valued Pfaffians and determinants is completely resolved in these cases.

3. **Quantum mechanical Yang-Mills matrix models:** The extension of Yang-Mills matrix models to quantum mechanical Yang-Mills matrix models, such as the BFSS and BMN models which also provide non-perturbative definitions of string theory and M-theory, involves the introduction of time. This new continuous variable requires obviously a lattice regularization. There is so much physics here relevant to the dynamics of black holes, gauge-gravity duality, strongly coupled gauge theory and many other fundamental problems.

4. **The noncommutative torus:** The noncommutative torus provides another, seemingly different, non-perturbative regularization of noncommutative field theory besides fuzzy spaces. The phenomena of emergent geometry is also observed here, as well as the phenomena of stripe phases, and furthermore, we can add fermions and supersymmetry in an obvious way. The connection to commutative theory and the commutative limit is more transparent in this case which is an advantage.

5. **Supersymmetry:** A non-perturbative definition of supersymmetry which allows Monte Carlo treatment is readily available from the above discussed, and much more, matrix models. These non-lattice simulations seem very promising to strongly coupled gauge theories.

Bibliography

[1] J. Ambjorn, K. N. Anagnostopoulos, W. Bietenholz, T. Hotta and J. Nishimura, "Large N dynamics of dimensionally reduced 4-D SU(N) superYang-Mills theory," JHEP **0007**, 013 (2000) [hep-th/0003208].

[2] J. Ambjorn, K. N. Anagnostopoulos, W. Bietenholz, T. Hotta and J. Nishimura, "Monte Carlo studies of the IIB matrix model at large N," JHEP **0007**, 011 (2000) [arXiv:hep-th/0005147].

[3] K. N. Anagnostopoulos, T. Azuma, K. Nagao and J. Nishimura, "Impact of supersymmetry on the nonperturbative dynamics of fuzzy spheres," JHEP **0509**, 046 (2005) [hep-th/0506062].

[4] K. N. Anagnostopoulos, T. Azuma and J. Nishimura, "Monte Carlo studies of the spontaneous rotational symmetry breaking in dimensionally reduced super Yang-Mills models," JHEP **1311**, 009 (2013) [arXiv:1306.6135 [hep-th]].

[5] A. D. Kennedy, I. Horvath and S. Sint, "A New exact method for dynamical fermion computations with nonlocal actions," Nucl. Phys. Proc. Suppl. **73**, 834 (1999) [hep-lat/9809092].

[6] M. A. Clark and A. D. Kennedy, "The RHMC algorithm for two flavors of dynamical staggered fermions," Nucl. Phys. Proc. Suppl. **129**, 850 (2004) [hep-lat/0309084].

[7] M. A. Clark, P. de Forcrand and A. D. Kennedy, "Algorithm shootout: R versus RHMC," PoS LAT **2005**, 115 (2006) [hep-lat/0510004].

[8] M. A. Clark, "The Rational Hybrid Monte Carlo Algorithm," PoS LAT **2006**, 004 (2006) [hep-lat/0610048].

[9] M. A. Clark and A. D. Kennedy, https://github.com/mikeaclark/AlgRemez, 2005.

[10] P. Austing, "Yang-Mills matrix theory," arXiv:hep-th/0108128.

[11] B. Ydri, "Impact of Supersymmetry on Emergent Geometry in Yang-Mills Matrix Models II," Int. J. Mod. Phys. A **27**, 1250088 (2012) [arXiv:1206.6375 [hep-th]].

Chapter 8

$U(1)$ Gauge Theory on the Lattice: Another Lattice Example

In this chapter we will follow the excellent pedagogical textbook [1] especially on practical detail regarding the implementation of the Metropolis and other algorithms to lattice gauge theories. The classic textbooks [2–5] were also very useful.

8.1 Continuum Considerations

A field theory is a dynamical system with N degrees of freedom where $N \longrightarrow \infty$. The classical description is given in terms of the Lagrangian and the action while the quantum description is given in terms of the Feynman path integral and the correlation functions. In a scalar field theory the basic field has spin $j = 0$ with respect to Lorentz transformations. Scalar field theories are relevant to critical phenomena. In gauge theories the basic fields have spin $j = 1$ (gauge vector fields) and spin $j = 1/2$ (fermions) and they are relevant to particle physics. The requirement of renormalizability restricts severely the set of quantum field theories to only few possible models. Quantum electrodynamics or QED is a renormalizable field theory given by the action

$$S_{\text{QED}} = \int d^4 x \left[-\frac{1}{4} F_{\mu\nu} F^{\mu\nu} + \bar{\psi}(i\gamma^\mu \partial_\mu - M)\psi - e\bar{\psi}\gamma_\mu \psi A^\mu \right]. \tag{8.1}$$

The γ^μ are the famous 4×4 Dirac gamma matrices which appear in any theory containing a spin $1/2$ field. They satisfy $\{\gamma^\mu, \gamma^\nu\} = 2\eta^{\mu\nu}$ where $\eta^{\mu\nu} = \text{diag}(1, -1, -1, -1)$. The electromagnetic field is given by the $U(1)$ gauge vector field A^μ with field strength $F_{\mu\nu} = \partial_\mu A_\nu - \partial_\nu A_\mu$ while the fermion (electron) field is given by the spinor field ψ with mass M. The spinor ψ is a $4-$component field and $\bar{\psi} = \psi^+ \gamma^0$. The interaction term is proportional to the electric charge e given by the last term $-e\bar{\psi}\gamma_\mu \psi A^\mu$. The Euler-Lagrange classical equations of motion derived from the above action are precisely the Maxwell equations $\partial_\mu F^{\mu\nu} = j^\nu$ with $j^\mu = e\bar{\psi}\gamma^\mu \psi$ and the Dirac equation $(i\gamma^\mu \partial_\mu - m - e\gamma_\mu A^\mu)\psi = 0$. The above theory is also invariant under the following $U(1)$ gauge transformations

$$A_\mu \longrightarrow A_\mu + \partial_\mu \Lambda \ , \ \psi \longrightarrow \exp(-ie\Lambda)\psi \ , \ \bar{\psi} \longrightarrow \bar{\psi}\exp(ie\Lambda). \tag{8.2}$$

The Feynman path integral is

$$Z = \int \mathcal{D}A^\mu \mathcal{D}\bar{\psi}\mathcal{D}\psi \exp(iS_{\text{QED}}). \tag{8.3}$$

Before we can study this theory numerically using the Monte Carlo method we need to:

1. Rotate to Euclidean signature in order to convert the theory into a statistical field theory.

2. Regularize the UV behavior of the theory by putting it on a lattice.

As a consequence we obtain an ordinary statistical system accessible to ordinary sampling techniques such as the Metropolis algorithm.

We start by discussing a little further the above action. The free fermion action in Minkowski spacetime is given by

$$S_F = \int d^4x \bar{\psi}(x)(i\gamma^\mu \partial_\mu - M)\psi(x). \tag{8.4}$$

This action is invariant under the global $U(1)$ transformation $\psi(x) \longrightarrow G\psi(x)$ and $\bar{\psi}(x) \longrightarrow \bar{\psi}(x)G^{-1}$ where $G = \exp(-i\Lambda)$. The symmetry $U(1)$ can be made local (i.e. G becomes a function of x) by replacing the ordinary derivative ∂_μ with the covariant derivative $D_\mu = \partial_\mu + ieA_\mu$ where the $U(1)$ gauge field A_μ is the electromagnetic $4-$vector potential. The action becomes

$$S_F = \int d^4x \bar{\psi}(x)(i\gamma^\mu D_\mu - M)\psi(x). \tag{8.5}$$

This action is invariant under

$$\psi \longrightarrow G(x)\psi \ , \ \bar{\psi} \longrightarrow \bar{\psi}G^{-1}(x), \tag{8.6}$$

provided we also transform the covariant derivative and the gauge field as follows

$$D_\mu \longrightarrow GD_\mu G^{-1} \iff A_\mu \longrightarrow G(x)A_\mu G^{-1}(x) - \frac{i}{e}G(x)\partial_\mu G^{-1}(x). \tag{8.7}$$

Since A_μ and $G(x) = \exp(-i\Lambda(x))$ commute the transformation law of the gauge field reduces to $A_\mu \longrightarrow A_\mu + \partial_\mu \Lambda/e$. The dynamics of the gauge field A_μ is given by the Maxwell action

$$S_G = -\frac{1}{4} \int d^4x F_{\mu\nu}F^{\mu\nu} \ , \ F_{\mu\nu} = \partial_\mu A_\nu - \partial_\nu A_\mu. \tag{8.8}$$

This action is also invariant under the local $U(1)$ gauge symmetry $A_\mu \longrightarrow A_\mu + \partial_\mu \Lambda/e$. The total action is then

$$S_{\text{QED}} = -\frac{1}{4} \int d^4x F_{\mu\nu}F^{\mu\nu} + \int d^4x \bar{\psi}(x)(i\gamma^\mu D_\mu - M)\psi(x). \tag{8.9}$$

This is precisely (8.1).

The Euclidean action S_F^{eucl} is obtained by i) making the replacement $x_0 \longrightarrow -ix_4$ wherever x_0 appears explicitly, ii) substituting $\psi^E(x) = \psi(\vec{x}, x_4)$ for $\psi(x) = \psi(\vec{x}, t)$, iii) making the replacements $A^0 \longrightarrow iA_4$ and $D^0 \longrightarrow iD_4$ and iv) multiplying the obtained expression by $-i$. Since in Euclidean space the Lorentz group is replaced by the $4-$dimensional rotation group we introduce new $\gamma-$matrices γ_μ^E as follows $\gamma_4^E = \gamma^0, \gamma_i^E = -i\gamma^i$. They satisfy $\{\gamma_\mu^E, \gamma_\nu^E\} = 2\delta_{\mu\nu}$. The fermion Euclidean action is then

$$S_F^{\text{Eucl}} = \int d^4x \bar{\psi}^E(x)(\gamma_\mu^E D_\mu + M)\psi^E(x). \tag{8.10}$$

Similarly the Euclidean action S_G^{eucl} is obtained by i) making the replacement $x_0 \longrightarrow -ix_4$ wherever x_0 appears explicitly, ii) making the replacement $A^0 \longrightarrow iA_4$ and iii) multiplying the obtained expression by $-i$. We can check that $F_{\mu\nu}F^{\mu\nu}$, $\mu, \nu = 0, 1, 2, 3$ will be replaced with $F_{\mu\nu}^2$, $\mu = 1, 2, 3, 4$. The gauge Euclidean action is then

$$S_G^{\text{Eucl}} = \frac{1}{4} \int d^4x F_{\mu\nu}^2. \tag{8.11}$$

The full Euclidean action is

$$S_{QED}^{\text{Eucl}} = \frac{1}{4} \int d^4x F_{\mu\nu}^2 + \int d^4x \bar{\psi}^E(x)(\gamma_\mu^E D_\mu + M)\psi^E(x). \tag{8.12}$$

We will drop the labels Eucl in the following.

8.2 Lattice Regularization

8.2.1 Lattice Fermions and Gauge Fields

Free Fermions on the Lattice: The continuum free fermion action in Euclidean 4d spacetime is

$$S_F = \int d^4x \bar{\psi}^E(x)(\gamma_\mu^E \partial_\mu + M)\psi^E(x). \tag{8.13}$$

This has the symmetry $\psi \longrightarrow e^{i\theta}\psi$ and the symmetry $\psi \longrightarrow e^{i\theta\gamma_5}\psi$ when $M = 0$. The associated conserved currents are known to be given by $J_\mu = \bar{\psi}\gamma_\mu\psi$ and $J_\mu^5 = \bar{\psi}\gamma_\mu\gamma_5\psi$ where $\gamma_5 = \gamma_1\gamma_2\gamma_3\gamma_4$. It is also a known result that in the quantum theory one can not maintain the conservation of both of these currents simultaneously in the presence of gauge fields.

A regularization which maintains exact chiral invariance of the above action can be achieved by replacing the Euclidean four dimensional spacetime by a four dimensional hypercubic lattice of N^4 sites. Every point on the lattice is specified by 4 integers which we denote collectively by $n = (n_1, n_2, n_3, n_4)$ where n_4 denotes Euclidean time. Clearly each component of the $4-$vector n is an integer in the range $-N/2 \leq n_\mu \leq N/2$ with N even. The lattice is assumed to be periodic. Thus $x_\mu = an_\mu$ where a is the lattice spacing

and $L = aN$ is the linear size of the lattice. Now to each site $x = an$ we associate a spinor variable $\psi(n) = \psi(x)$ and the derivative $\partial_\mu \psi(x)$ is replaced by

$$\partial_\mu \psi(x) \longrightarrow \frac{1}{a} \hat{\partial}_\mu \psi(n) = \frac{1}{2a} \Big[\psi(n + \hat{\mu}) - \psi(n - \hat{\mu}) \Big]. \tag{8.14}$$

The vector $\hat{\mu}$ is the unit vector in the $\mu-$direction. With this prescription the action (8.13) becomes (with $\hat{M} = aM$ and $\hat{\psi} = a^{3/2}\psi$)

$$
\begin{aligned}
S_F &= \sum_n \sum_m \sum_\alpha \sum_\beta \bar{\hat{\psi}}_\alpha(n) K_{\alpha\beta}(n, m) \hat{\psi}_\beta(m) \\
K_{\alpha\beta}(n, m) &= \frac{1}{2} \sum_\mu (\gamma_\mu)_{\alpha\beta} \Big(\delta_{m,n+\hat{\mu}} - \delta_{m,n-\hat{\mu}} \Big) + \hat{M} \delta_{\alpha\beta} \delta_{m,n}.
\end{aligned}
\tag{8.15}
$$

$U(1)$ **Lattice Gauge Fields:**   The free fermion action on the lattice is therefore given by

$$
\begin{aligned}
S_F = \ & \hat{M} \sum_n \sum_\alpha \bar{\hat{\psi}}_\alpha(n) \hat{\psi}_\alpha(n) \\
& - \frac{1}{2} \sum_n \sum_\alpha \sum_\beta \sum_\mu \Big[(\gamma_\mu)_{\alpha\beta} \bar{\hat{\psi}}_\alpha(n + \hat{\mu}) \hat{\psi}_\beta(n) - (\gamma_\mu)_{\alpha\beta} \bar{\hat{\psi}}_\alpha(n) \hat{\psi}_\beta(n + \hat{\mu}) \Big].
\end{aligned}
$$
$$\tag{8.16}$$

This action has the following global $U(1)$ symmetry

$$\hat{\psi}_\alpha(n) \longrightarrow G\hat{\psi}_\alpha(n) \; , \; \bar{\hat{\psi}}_\alpha(n) \longrightarrow \bar{\hat{\psi}}_\alpha(n) G^{-1}. \tag{8.17}$$

The phase $G = \exp(-i\Lambda)$ is an element of $U(1)$. By requiring the theory to be invariant under local $U(1)$ symmetry, i.e. allowing G to depend on the lattice site we arrive at a gauge invariant fermion action on the lattice. The problem lies in how we can make the bilinear fermionic terms (the second and third terms) in the above action gauge invariant.

We go back to the continuum formulation and see how this problem is solved. In the continuum the fermionic bilinear $\bar{\psi}(x)\psi(y)$ transforms under a local $U(1)$ transformation as follows

$$\bar{\psi}(x)\psi(y) \longrightarrow \bar{\psi}(x) G^{-1}(x) G(y) \psi(y). \tag{8.18}$$

This bilinear can be made gauge covariant by inserting the Schwinger line integral

$$U(x, y) = e^{ie \int_x^y dz_\mu A_\mu(z)}, \tag{8.19}$$

which transforms as

$$U(x, y) \longrightarrow G(x) U(x, y) G^{-1}(y). \tag{8.20}$$

Therefore the fermionic bilinear

$$\bar{\psi}(x) U(x, y) \psi(y) = \bar{\psi}(x) e^{ie \int_x^y dz_\mu A_\mu(z)} \psi(y) \tag{8.21}$$

is U(1) gauge invariant. For $y = x + \epsilon$ we have

$$U(x, x + \epsilon) = e^{ie\epsilon_\mu A_\mu(x)}. \tag{8.22}$$

We conclude that in order to get local $U(1)$ gauge invariance we replace the second and third bilinear fermionic terms in the above action as follows

$$\hat{\bar{\psi}}(n)(r - \gamma_\mu)\hat{\psi}(n + \hat{\mu}) \longrightarrow \hat{\bar{\psi}}(n)(r - \gamma_\mu)U_{n,n+\hat{\mu}}\hat{\psi}(n + \hat{\mu})$$
$$\hat{\bar{\psi}}(n + \hat{\mu})(r - \gamma_\mu)\hat{\psi}(n) \longrightarrow \hat{\bar{\psi}}(n + \hat{\mu})(r - \gamma_\mu)U_{n+\hat{\mu},n}\hat{\psi}(n). \tag{8.23}$$

We obtain then the action

$$
\begin{aligned}
S_F &= \hat{M} \sum_n \sum_\alpha \hat{\bar{\psi}}_\alpha(n)\hat{\psi}_\alpha(n) \\
&- \frac{1}{2} \sum_n \sum_\alpha \sum_\beta \sum_\mu \left[(\gamma_\mu)_{\alpha\beta}\hat{\bar{\psi}}_\alpha(n + \hat{\mu})U_{n+\hat{\mu},n}\hat{\psi}_\beta(n) - (\gamma_\mu)_{\alpha\beta}\hat{\bar{\psi}}_\alpha(n)U_{n,n+\hat{\mu}}\hat{\psi}_\beta(n + \hat{\mu}) \right].
\end{aligned}
$$
$$\tag{8.24}$$

The $U(1)$ element $U_{n,n+\hat{\mu}}$ lives on the lattice link connecting the two points n and $n + \hat{\mu}$. This link variable is therefore a directed quantity given explicitly by

$$U_{n,n+\hat{\mu}} = e^{i\phi_\mu(n)} \equiv U_\mu(n) \ , \ U_{n+\hat{\mu},n} = U^+_{n,n+\hat{\mu}} = e^{-i\phi_\mu(n)} \equiv U^+_\mu(n). \tag{8.25}$$

The second equality is much clearer in the continuum formulation but on the lattice it is needed for the reality of the action. The phase $\phi_\mu(n)$ belongs to the compact interval $[0, 2\pi]$. Alternatively we can work with $A_\mu(n)$ defined through

$$\phi_\mu(n) = eaA_\mu(n). \tag{8.26}$$

Let us now consider the product of link variables around the smallest possible closed loop on the lattice, i.e. a plaquette. For a plaquette in the $\mu - \nu$ plane we have

$$U_P \equiv U_{\mu\nu}(n) = U_\mu(n)U_\nu(n + \hat{\mu})U^+_\mu(n + \hat{\nu})U^+_\nu(n). \tag{8.27}$$

The links are path-ordered. We can immediately compute

$$U_P \equiv U_{\mu\nu}(n) = e^{iea^2 F_{\mu\nu}(n)} \ , \ F_{\mu\nu} = \frac{1}{a}\left[A_\nu(n + \hat{\mu}) - A_\nu(n) - A_\mu(n + \hat{\nu}) + A_\mu(n) \right]. \tag{8.28}$$

In other words in the continuum limit $a \longrightarrow 0$ we have

$$\frac{1}{e^2} \sum_n \sum_{\mu < \nu} \left[1 - \frac{1}{2}\left(U_{\mu\nu}(n) + U^+_{\mu\nu}(n) \right) \right] = \frac{a^4}{4} \sum_n \sum_{\mu,\nu} F^2_{\mu\nu}. \tag{8.29}$$

The $U(1)$ gauge action on the lattice is therefore

$$S_G = \frac{1}{e^2} \sum_P \left[1 - \frac{1}{2}\left(U_p + U^+_p \right) \right]. \tag{8.30}$$

8.2.2   Quenched Approximation

The QED partition function on a lattice Λ is given by

$$Z = \int \mathcal{D}U \ \mathcal{D}\hat{\bar{\psi}}\mathcal{D}\hat{\psi} \ e^{-S_G[U]-S_F[U,\hat{\bar{\psi}},\hat{\psi}]}. \tag{8.31}$$

The measures are defined by

$$\mathcal{D}U = \prod_{n\in\Lambda}\prod_{\mu=1}^{4} dU_\mu(n) \ , \ \mathcal{D}\hat{\bar{\psi}} = \prod_{n\in\Lambda} d\hat{\bar{\psi}}(n) \ , \ \mathcal{D}\hat{\psi} = \prod_{n\in\Lambda} d\hat{\psi}(n). \tag{8.32}$$

The plaquette and the link variable are given by

$$U_{\mu\nu}(n) = U_\mu(n)U_\nu(n+\hat{\mu})U_\mu^+(n+\hat{\nu})U_\nu^+(n) \ , \ U_\mu(n) = e^{i\phi_\mu(n)}. \tag{8.33}$$

The action of a $U(1)$ gauge theory on a lattice is given by (with $\beta = 1/e^2$)

$$S_G[U] \ = \ \beta\sum_{n\in\Lambda}\sum_{\mu<\nu}\left[1 - \frac{1}{2}\big(U_{\mu\nu}(n) + U_{\mu\nu}^+(n)\big)\right] = \beta\sum_{n\in\Lambda}\sum_{\mu<\nu}\text{Re}\left[1 - U_{\mu\nu}(n)\right]. \tag{8.34}$$

The action of fermions coupled to a $U(1)$ gauge field on a lattice is given by

$$S_F[U,\hat{\bar{\psi}},\hat{\psi}] = \sum_\alpha\sum_\beta\sum_n\sum_m \hat{\bar{\psi}}_\alpha(n)\mathcal{D}_{\alpha\beta}(U)_{n,m}\hat{\psi}_\beta(m). \tag{8.35}$$

Where

$$\mathcal{D}_{\alpha\beta}(U)_{n,m} = \hat{M}\delta_{\alpha\beta}\delta_{n,m} - \frac{1}{2}(\gamma_\mu)_{\alpha\beta} \ \delta_{n,m+\hat{\mu}} \ U_{n+\hat{\mu},n} + \frac{1}{2}(\gamma_\mu)_{\alpha\beta} \ \delta_{m,n+\hat{\mu}} \ U_{n,n+\hat{\mu}}. \tag{8.36}$$

Using the result

$$\int \mathcal{D}\hat{\bar{\psi}}\mathcal{D}\hat{\psi} \ e^{-\sum_\alpha\sum_\beta\sum_n\sum_m \hat{\bar{\psi}}_\alpha(n)\mathcal{D}_{\alpha\beta}(U)_{n,m}\hat{\psi}_\beta(m)} \ = \ \det\mathcal{D}_{\alpha\beta}(U)_{n,m}. \tag{8.37}$$

The partition function becomes

$$Z = \int \mathcal{D}U \ \det\mathcal{D}_{\alpha\beta}(U)_{n,m} \ e^{-S_G[U]}. \tag{8.38}$$

At this stage we will make the approximation that we can set the determinal equal 1, i.e. the QED partition function will be approximated by

$$Z = \int \mathcal{D}U \ e^{-S_G[U]} \tag{8.39}$$

This is called the quenched approximation.

8.2.3   Wilson Loop, Creutz Ratio and Other Observables

The first observable we would like to measure is the expectation value of the action which after dropping the constant term is given by

$$< S_G[U] > \quad = \quad -\beta \sum_{n \in \Lambda} \sum_{\mu < \nu} < \text{Re } U_{\mu\nu}(n) > . \tag{8.40}$$

The specific heat is the corresponding second moment, viz

$$C_v \quad = \quad < S_G[U]^2 > - < S_G[U] >^2 . \tag{8.41}$$

We will also measure the expectation value of the so-called Wilson loop which has a length I in one of the spatial direction (say 1) and a width J in the temporal direction 4. This rectangular loop C is defined by

$$W_C[U] = S(n, n + I\hat{1})T(n + I\hat{1}, n + I\hat{1} + J\hat{4})S^+(n + J\hat{4}, n + I\hat{1} + J\hat{4})T^+(n, n + J\hat{4}) \tag{8.42}$$

The Wilson lines are

$$S(n, n + I\hat{1}) = \prod_{i=0}^{I-1} U_1(n + i\hat{1}) \ , \ \ S(n + J\hat{4}, n + I\hat{1} + J\hat{4}) = \prod_{i=0}^{I-1} U_1(n + i\hat{1} + J\hat{4}). \tag{8.43}$$

The temporal transporters are

$$T(n + I\hat{1}, n + I\hat{1} + J\hat{4}) = \prod_{j=0}^{J-1} U_4(n + I\hat{1} + j\hat{4}) \ , \ \ T(n, n + J\hat{4}) = \prod_{j=0}^{J-1} U_4(n + j\hat{4}). \tag{8.44}$$

The expectation value of $W_C[U]$ will be denoted by

$$W[I, J] = \frac{\int \mathcal{D}U \ W_C[U] \ e^{-S_G[U]}}{\int \mathcal{D}U \ e^{-S_G[U]}}. \tag{8.45}$$

By using the fact that under $\phi_\mu(n) \longrightarrow -\phi_\mu(n)$, the partition function is invariant while the Wilson loop changes its orientation, i.e. $W_C[U] \longrightarrow W_C[U]^+$, we obtain

$$W[I, J] = < \text{Re } W_C[U] > . \tag{8.46}$$

It is almost obvious that in the continuum limit

$$W[I, J] \longrightarrow W[R, T] = < \exp(ie \oint_C dx_\mu A_\mu) > . \tag{8.47}$$

The loop C is now a rectangular contour with spatial length $R = Ia$ and timelike length $T = Ja$. This represents the probability amplitude for the process of creating an infinitely heavy, i.e. static, quark-antiquark [1] pair at time $t = 0$ which are separated by a distance R, then allowing them to evolve in time and then eventually annihilate after a long time T.

---

[1]For $U(1)$ we should really speak of an electron-positron pair.

The precise meaning of the expectation value (8.46) is as follows

$$< \mathcal{O} >= \frac{1}{L} \sum_{i=1}^{L} \left(\frac{1}{N^3 N_T} \sum_n \text{Re } W_C[U_i] \right). \tag{8.48}$$

In other words we also take the average over the lattice which is necessary in order to reduce noise in the measurment of the Creutz ratio (see below).

The above Wilson loop is the order parameter of the pure $U(1)$ gauge theory. For large time T we expect the behavior

$$W[R, T \longrightarrow \infty] \longrightarrow e^{-V(R)T} = e^{-aV(R)J}, \tag{8.49}$$

where $V(R)$ is the static quark-antiquark potential. For strong coupling (small β) we can show that the potential is linear, viz

$$V(R) = \sigma R. \tag{8.50}$$

The constant σ is called the string tension from the fact that the force between the quark and the antiquark can be modeled by the force in a string attached to the quark and antiquark. For a linear potential the Wilson loop follows an area law $W[R, T] = \exp(-\sigma A)$ with $A = a^2 IJ$. This behavior is typical in a confining phase which occurs at high temperature.

For small coupling (large β,low temperature) the lattice $U(1)$ gauge field becomes weakly coupled and as a consequence we expect the Coulomb potential to dominate the static quark-antiquark potential, viz

$$V(R) = \frac{Z}{R}. \tag{8.51}$$

Hence for large R the quark and antiquark become effectively free and their energy is simply the sum of their self-energies. The Wilson loop in this case follows a perimeter law $W[R, T] = \exp(-2\epsilon T)$.

In summary for a rectangular $R \times T$ Wilson loop with perimeter $P = 2(R + T)$ and area $A = RT$ we expect the behavior

$$W[R, T] = e^{-\sigma A} \text{ , confinement phase.} \tag{8.52}$$

$$W[R, T] = e^{-\epsilon P} \text{ , coulomb phase.} \tag{8.53}$$

In general the Wilson loop will behave as

$$W[R, T] = e^{-B-\sigma A-\epsilon P}. \tag{8.54}$$

The perimeter piece actually dominates for any fixed size loop. To measure the string tension we must therefore eliminate the perimeter behavior which can be achieved using the so-called Creutz ratio defined by

$$\chi(I, J) = -\ln \frac{W[I, J]W[I-1, J-1]}{W[I, J-1]W[I-1, J]}. \tag{8.55}$$

For large loops clearly

$$\chi(I, J) = a^2 \sigma. \tag{8.56}$$

This should holds especially in the confinement phase whereas in the Coulomb phase we should expect $\chi(I, J) \sim 0$.

The 1×1 Wilson loop $W(1, 1)$ is special since it is related to the average action per plaquette. We have

$$W[1, 1] = < \text{Re } U_1(n)U_4(n + \hat{1})U_4^+(n)U_1^+(n + \hat{4}) > . \tag{8.57}$$

Next we compute straightforwardly

$$-\frac{\partial \ln Z}{\partial \beta} = \sum_n \sum_{\mu < \nu} < [1 - \text{Re } U_{\mu\nu}(n)] > . \tag{8.58}$$

Clearly all the planes $\mu\nu$ are equivalent and thus we should have

$$
\begin{aligned}
-\frac{\partial \ln Z}{\partial \beta} &= 6 \sum_n < [1 - \text{Re } U_{14}(n)] > \\
&= 6 \sum_n < [1 - \text{Re } U_1(n)U_4(n + \hat{1})U_4^+(n)U_1^+(n + \hat{4})] > .
\end{aligned} \tag{8.59}
$$

Remark that there are $N^3 N_T$ lattice sites. Each site corresponds to 4 plaquettes in every plane $\mu\nu$ and thus it corresponds to 4×6 plaquettes in all. Each plaquette in a plane $\mu\nu$ corresponds to 4 sites and thus to avoid overcounting we must divide by 4. In summary we have $4 \times 6 \times N^3 \times N_T/4$ plaquettes in total. Six is therefore the ratio of the number of plaquettes to the number of sites.

We have then

$$-\frac{1}{6N^3 N_T}\frac{\partial \ln Z}{\partial \beta} = 1 - \frac{1}{N^3 N_T}\sum_n < \text{Re } U_1(n)U_4(n + \hat{1})U_4^+(n)U_1^+(n + \hat{4}) > \tag{8.60}$$

We can now observe that all lattice sites n are the same under the expectation value, namely

$$-\frac{1}{6N^3 N_T}\frac{\partial \ln Z}{\partial \beta} = 1 - < \text{Re } U_1(n)U_4(n + \hat{1})U_4^+(n)U_1^+(n + \hat{4}) > . \tag{8.61}$$

This is the average action per plaquette (the internal energy) denoted by

$$P = -\frac{1}{6N^3 N_T}\frac{\partial \ln Z}{\partial \beta} = 1 - W[1, 1]. \tag{8.62}$$

8.3 Monte Carlo Simulation of Pure $U(1)$ Gauge Theory

8.3.1 The Metropolis Algorithm

The action of pure U(1) gauge theory, the corresponding partition function and the measure of interest are given on a lattice Λ respectively by (with $\beta = 1/e^2$)

$$S_G[U] = \beta \sum_{n \in \Lambda} \sum_{\mu < \nu} \text{Re}\left[1 - U_{\mu\nu}(n)\right]. \tag{8.63}$$

$$Z = \int \mathcal{D}U \; e^{-S_G[U]}. \tag{8.64}$$

$$\mathcal{D}U = \prod_{n\in\Lambda} \prod_{\mu=1}^{4} dU_\mu(n). \tag{8.65}$$

The vacuum expectation value of any observable $\mathcal{O} = \mathcal{O}(U)$ is given by

$$< \mathcal{O} > = \frac{1}{Z} \int \mathcal{D}U \; \mathcal{O} \; e^{-S_G[U]}. \tag{8.66}$$

For $U(1)$ gauge theory we can write

$$U_\mu(n) = e^{i\phi_\mu(n)}. \tag{8.67}$$

Hence

$$\mathcal{D}U = \prod_{n\in\Lambda} \prod_{\mu=1}^{4} d\phi_\mu(n). \tag{8.68}$$

We will use the Metropolis algorithm to solve this problem. This goes as follows. Starting from a given gauge field configuration, we choose a lattice point n and a direction μ, and change the link variable there, which is $U_\mu(n)$, to $U_\mu(n)^{'}$. This link is shared by 6 plaquettes. The corresponding variation of the action is

$$\Delta S_G[U_\mu(n))] \;\; = \;\; S_G[U^{'}] - S_G[U]. \tag{8.69}$$

The gauge field configurations U and $U^{'}$ differ only by the value of the link variable $U_\mu(n)$. We need to isolate the contribution of $U_\mu(n)$ to the action S_G. Note the fact that $U_{\mu\nu}^+ = U_{\nu\mu}$. We write

$$S_G[U] \;\; = \;\; \beta \sum_{n\in\Lambda} \sum_{\mu<\nu} 1 - \frac{\beta}{2} \sum_{n\in\Lambda} \sum_{\mu<\nu} \left(U_{\mu\nu}(n) + U_{\mu\nu}^+(n)\right). \tag{8.70}$$

The second term is

$$-\frac{\beta}{2} \sum_{n\in\Lambda} \sum_{\mu<\nu} U_{\mu\nu}(n) \;\; = \;\; -\frac{\beta}{2} \sum_{n\in\Lambda} \sum_{\mu<\nu} U_\mu(n)U_\nu(n+\hat{\mu})U_\mu^+(n+\hat{\nu})U_\nu^+(n). \tag{8.71}$$

In the $\mu - \nu$ plane, the link variable $U_\mu(n)$ appears twice corresponding to the two lattice points n and $n - \hat{\nu}$. For every μ there are three relevant planes. The six relevant terms are therefore given by

$$-\frac{\beta}{2} \sum_{n\in\Lambda} \sum_{\mu<\nu} U_{\mu\nu}(n) \longrightarrow \;\; - \;\; \frac{\beta}{2} \sum_{\nu\neq\mu} \Bigg(U_\mu(n)U_\nu(n+\hat{\mu})U_\mu^+(n+\hat{\nu})U_\nu^+(n)$$
$$+ \;\; U_\mu^+(n)U_\nu^+(n-\hat{\nu})U_\mu(n-\hat{\nu})U_\nu(n-\hat{\nu}+\hat{\mu}) \Bigg) + ...\tag{8.72}$$

By adding the complex conjugate terms we obtain

$$-\frac{\beta}{2}\sum_{n\in\Lambda}\sum_{\mu<\nu}(U_{\mu\nu}(n)+U_{\mu\nu}^{+}(n)) \longrightarrow - \frac{\beta}{2}\left(U_{\mu}(n)\mathcal{A}_{\mu}(n)+U_{\mu}^{+}(n)\mathcal{A}_{\mu}^{+}(n)\right)+..(8.73)$$

The $\mathcal{A}_{\mu}(n)$ is the sum over the six so-called staples which are the products over the other three link variables which together with $U_{\mu}(n)$ make up the six plaquettes which share $U_{\mu}(n)$. Explicitly we have

$$\mathcal{A}_{\mu}(n)=\sum_{\nu\neq\mu}\left(U_{\nu}(n+\hat{\mu})U_{\mu}^{+}(n+\hat{\nu})U_{\nu}^{+}(n)+U_{\nu}^{+}(n+\hat{\mu}-\hat{\nu})U_{\mu}^{+}(n-\hat{\nu})U_{\nu}(n-\hat{\nu})\right).(8.74)$$

We have then the result

$$-\frac{\beta}{2}\sum_{n\in\Lambda}\sum_{\mu<\nu}(U_{\mu\nu}(n)+U_{\mu\nu}^{+}(n)) \longrightarrow - \beta\mathrm{Re}(U_{\mu}(n)\mathcal{A}_{\mu}(n))+... \tag{8.75}$$

We compute then

$$\begin{aligned}\Delta S_{G}[U_{\mu}(n))] &= S_{G}[U^{'}]-S_{G}[U]\\ &= -\beta(U_{\mu}(n)^{'}-U_{\mu}(n))\mathcal{A}_{\mu}(n).\end{aligned} \tag{8.76}$$

Having computed the variation $\Delta S_{G}[U_{\mu}(n))]$, next we inspect its sign. If this variation is negative then the proposed change $U_{\mu}(n) \longrightarrow U_{\mu}(n)^{'}$ will be accepted (classical mechanics). If the variation is positive, we compute the Boltzmann probability

$$\exp(-\Delta S_{G}[U_{\mu}(n))]) = \exp(\beta(U_{\mu}(n)^{'}-U_{\mu}(n))\mathcal{A}_{\mu}(n)). \tag{8.77}$$

The proposed change $U_{\mu}(n) \longrightarrow U_{\mu}(n)^{'}$ will be accepted according to this probability (quantum mechanics). In practice we will pick a uniform random number r between 0 and 1 and compare it with $\exp(-\Delta S_{G}[U_{\mu}(n))])$. If $\exp(-\Delta S_{G}[U_{\mu}(n))]) < r$ we accept this change otherwise we reject it.

We go through the above steps for every link in the lattice which constitutes one Monte Carlo step. Typically equilibration (thermalization) is reached after a large number of Monte Carlo steps at which point we can start taking measurements based on the formula (8.66) written as

$$<\mathcal{O}>=\frac{1}{L}\sum_{i=1}^{L}\mathcal{O}_{i} \ , \ \mathcal{O}_{i}=\mathcal{O}(U_{i}). \tag{8.78}$$

The L configurations $U_{i}=\{U_{\mu}(n)\}_{i}$ are L thermalized gauge field configurations distributed according to $\exp(-S_{G}[U])$.

The error bars in the different measurements will be estimated using the jackknife method. We can also compute auto-correlation time and take it into account by separating the measured gauge field configurations U_{i} by at least one unit of auto-correlation time.

Let us also comment on how we choose the proposed configurations $U_{\mu}(n)^{'}$. The custom is to take $U_{\mu}(n)^{'}=XU_{\mu}(n)$ where X is an element in the gauge group (which is

here $U(1)$) near the identity. In order to maintain a symmetric selection probability, X should be drawn randomly from a set of $U(1)$ elements which contains also X^{-1}. For $U(1)$ gauge group we have $X = \exp(i\phi)$ where $\phi \in [0, 2\pi]$. In principle the acceptance rate can be maintained around at least 0.5 by tuning appropriately the angle ϕ. Reunitarization of $U_\mu(n)'$ may also be applied to reduce rounding errors.

The final technical remark is with regard to boundary conditions. In order to reduce edge effects we usually adopt periodic boundary conditions, i.e.

$$U_\mu(N, n_2, n_3, n_4) = U_\mu(0, n_2, n_3, n_4), U_\mu(n_1, N, n_3, n_4) = U_\mu(n_1, 0, n_3, n_4),$$
$$U_\mu(n_1, n_2, N, n_4) = U_\mu(n_1, n_2, n, 0, n_4), U_\mu(n_1, n_2, n_3, N_T) = U_\mu(n_1, n_2, n_3, 0) \quad (8.79)$$

This means in particular that the lattice is actually a four dimensional torus. In the actual code this is implemented by replacing $i \pm 1$ by ip(i) and im(i), ipT(i) and imT(i) respectively which are defined by

```
do i=1,N
   ip(i)=i+1
   im(i)=i-1
enddo
   ip(N)=1
   im(1)=N
do i=1,NT
   ipT(i)=i+1
   imT(i)=i-1
enddo
   ipT(NT)=1
   imT(1)=NT
```

A code written along the above lines is attached in the last chapter.

8.3.2   Some Numerical Results

1. We run simulations for $N = 3, 4, 8, 10, 12$ with the coupling constant in the range $\beta = 2, ..., 12$. We use typically 2^{14} thermalization steps and 2^{14} measurements steps.

2. We measure the specific heat (figure (8.1)). We observe a peak in the specific heat at around $\beta = 1$. The peak grows with N which signals a critical behavior typical of 2nd order transition.

3. The simplest order parameter is the action per plaquette P, defined in equation (8.62), which is shown on figure (8.2). We observe good agreement between the high-temperature and low-temperature expansions of P from one hand and the corresponding observed behavior in the strong coupling and weak coupling regions respectively from the other hand. We note that the high-temperature and low-temperature

expansions of the pure $U(1)$ gauge field are given by

$$P = 1 - \frac{\beta}{2} + O(\beta^3) \ , \ \text{high } T. \tag{8.80}$$

$$P = 1 - \frac{1}{4\beta} + O(1/\beta^2) \ , \ \text{low } T. \tag{8.81}$$

We do not observe a clear-cut discontinuity in P which is, in any case, consistent with the conclusion that this phase is second order. We note that for higher $U(N)$ the transition is first order [2].

A related object to P is the total action shown on figure (8.3).

4. A more powerful order parameters are the Wilson loops which are shown on figure (8.4). We observe that the Wilson loop in the strong coupling region averages to zero very quickly as we increase the size of the loop. This may be explained by an area law behavior. In the weak coupling region, the evolution as a function of the area is much more slower. The demarcation between the two phases becomes very sharp (possibly a jump) for large loops at $\beta = 1$.

5. Calculating the expectation value of the Wilson loop and then extracting the string tension is very difficult since the perimeter law is dominant more often. The Creutz ratios (figure (8.5)) allow us to derive the string tension in a direct way without measuring the Wilson loop. The string tension is the coefficient of the linearly rising part of the potential for large (infinite) separations of a quark-antiquark pair in the absence of pair production processes. In this way, we hope to measure the physical string tension in a narrow range of the coupling constant.

   We observe that the string tension in the weak coupling regime is effectively independent of the coupling constant and it is essentially zero. In the strong coupling regime we reproduce the strong coupling behavior

$$\sigma = -\ln\frac{\beta}{2}. \tag{8.82}$$

8.3.3   Coulomb and Confinement Phases

The physics of the compact $U(1)$ theory is clearly different in the weak- and strong-coupling regions. This can be understood from the fact that there is a phase transition as a function of the bare coupling constant. The compact $U(1)$ theory at weak coupling is not confining and contains no glueballs but simply the photons of the free Maxwell theory. One speaks of a Coulomb phase at weak coupling and a confining phase at strong coupling. In the Coulomb phase photons are massless and the static potential has the standard Coulomb form

$$V = -\frac{e^2}{4\pi r} + \text{constant}, \tag{8.83}$$

whereas in the confinement phase photons become massive and the potential is linearly confining at large distances

$$V = \sigma r. \tag{8.84}$$

There is a phase transition at a critical coupling $\beta \approx 1$ at which the string tension $\sigma(\beta)$ vanishes in the Coulomb phase. In the confinement phase topological configurations are important such as monopoles and glueballs.

The strong-coupling expansion is an expansion in powers of $1/g^2$. It has the advantage over the weak-coupling expansion that it has a non-zero radius of convergence. A lot of effort has been put into using it as a method of computation similar to the high-temperature or the hopping parameter expansion for scalar field theories. One has to be able to tune on the values of the coupling constant where the theory exhibits continuum behavior. This turns out to be difficult for gauge theories. However, a very important aspect of the strong-coupling expansion is that it gives insight into the qualitative behavior of the theory such as confinement and the particle spectrum.

The strong-coupling expansion of compact $U(1)$ theory shows explicitly that the theory is confining, i.e. the potential is linear with a string tension given by (with $a_1 = \beta/2$)

$$\sigma \quad = \quad -\ln a_1 - 2(d-2)a_1^4 + ... \tag{8.85}$$

Figure 8.1: The specific heat on a 3^4, 4^4, 10^4 and 12^4 lattices.

Figure 8.2: The action per plaquette on a 8^4 and 10^4 lattices.

Figure 8.3: The action on a 4^4 and 12^4 lattices.

Figure 8.4: The Wilson loop as a function of the inverse coupling strength β.

Figure 8.5: String tension from Creutz ratio as a function of β on a 12^4 lattice.

Bibliography

[1] C. Gattringer and C. B. Lang, "Quantum chromodynamics on the lattice," Lect. Notes Phys. **788**, 1 (2010).

[2] M. Creutz, "Quarks, Gluons And Lattices," Cambridge, Uk: Univ. Pr. (1983) 169 P. (Cambridge Monographs On Mathematical Physics).

[3] J. Smit, "Introduction to quantum fields on a lattice: A robust mate," Cambridge Lect. Notes Phys. **15**, 1 (2002).

[4] H. J. Rothe, "Lattice gauge theories: An Introduction," World Sci. Lect. Notes Phys. **74**, 1 (2005).

[5] I. Montvay and G. Munster, "Quantum fields on a lattice," Cambridge, UK: Univ. Pr. (1994) 491 p. (Cambridge monographs on mathematical physics).

Chapter 9

Codes

```fortran
      program my_metropolis_ym
      implicit none
      integer dim,dimm,N,ther,mc,Tther,Tmc
      integer lambda,i,j,idum
      parameter (dimm=10,N=8)
      parameter (Tther=2**11,Tmc=2**11)
      double complex X(dimm,N,N)
      double precision xx,y,Accept,Reject,inn,interval,pa
      double precision act(Tmc),actio,average_act,error_act
      double precision t_1, t_2
      real x0

      call cpu_time(t_1)

      do dim=2,dimm
         if(dim.le.dimm)then

c..........initialization of random number generator...........

         idum=-148175
         x0=0.0
         idum=idum-2*int(secnds(x0))

c.......inititialization of X...............................

         inn=1.0d0
         do lambda=1,dimm
            if (lambda.le.dim)then
               do i=1,N
                  do j=i,N
                     if (j.ne.i) then
                        xx=interval(idum,inn)
                        y=interval(idum,inn)
                        X(lambda,i,j)=cmplx(xx,y)
                        X(lambda,j,i)=cmplx(xx,-y)
                     else
                        xx=interval(idum,inn)
                        X(lambda,i,j)=xx
                     endif
                  enddo
               enddo
            else
               do i=1,N
                  do j=i,N
                     if (j.ne.i) then
                        xx=0.0d0
                        y=0.0d0
                        X(lambda,i,j)=cmplx(xx,y)
                        X(lambda,j,i)=cmplx(xx,-y)
                     else
                        xx=0.0d0
                        X(lambda,i,j)=xx
                     endif
                  enddo
               enddo
            endif
         enddo

c.... accepts including flips, rejects and the acceptance rate pa...

         Reject=0.0d0
         Accept=0.0d0
         pa=0.0d0

c.............thermalization....................................

         do ther=1,Tther
```

```fortran
            call metropolis(dim,dimm,N,X,Reject,Accept,inn,idum)
            call adjust_inn(pa,inn,Reject,Accept)
            call  action(dim,dimm,N,X,actio)
            write(*,*)ther,actio,pa
            write(10+dim,*)ther,actio,pa
         enddo

c............monte carlo evolution................................

         do mc=1,Tmc
            call metropolis(dim,dimm,N,X,Reject,Accept,inn,idum)
            call adjust_inn(pa,inn,Reject,Accept)
            call  action(dim,dimm,N,X,actio)
            act(mc)=actio
            write(*,*)mc,act(mc),pa
            write(21+dim,*)mc,act(mc),pa
         enddo

c.............measurements.......................................

         call jackknife_binning(Tmc,act,average_act,error_act)
         write(*,*)dim,average_act,error_act
         write(32,*)dim,average_act,error_act
      endif
      enddo

c.........cpu time..............................................

      call cpu_time(t_2)
      write(*,*)"cpu_time", t_2-t_1

      return
      end

c...............action....................................

      subroutine action(dim,dimm,N,X,actio)
      implicit none
      integer dim,dimm,N,mu,nu,i,j,k,l
      double complex X(dimm,N,N)
      double precision actio,action0

      actio=0.0d0
      do mu =1,dimm
         do nu=mu+1,dimm
            action0=0.0d0
            do i=1,N
               do j=1,N
                  do k=1,N
                     do l=1,N
               action0=action0+X(mu,i,j)*X(nu,j,k)*X(mu,k,l)*X(nu,l,i)
     &                      -X(mu,i,j)*X(mu,j,k)*X(nu,k,l)*X(nu,l,i)
                     enddo
                  enddo
               enddo
            enddo
            action0=-N*action0
            actio=actio+action0
         enddo
      enddo

      return
      end

c..............metropolis algorithm.........................

      subroutine metropolis(dim,dimm,N,X,Reject,Accept,inn,idum)
```

```fortran
      implicit none
      integer dim,dimm,N,i,j,lambda,idum
      double precision Reject,Accept,inn,interval,deltaS,ran2,z1,p1,xx,y
      double complex X(dimm,N,N),dc,dcbar

      do lambda=1,dim
c..............diagonal..........................
         do i=1,N
            xx=interval(idum,inn)
            y=interval(idum,inn)
            dc=cmplx(xx,0)
            dcbar=cmplx(xx,-0)
            call variationYM(dim,dimm,N,lambda,i,i,dc,dcbar,X,deltaS)
            if ( deltaS .gt. 0.0d0 ) then
               z1=ran2(idum)
               p1=dexp(-deltaS)
               if ( z1 .lt. p1 ) then
                  X(lambda,i,i)=X(lambda,i,i)+dc+dcbar
                  Accept=Accept+1.0d0
               else
                  Reject=Reject+1.0d0
               endif
            else
               X(lambda,i,i)=X(lambda,i,i)+dc+dcbar
               Accept=Accept+1.0d0
            endif
         enddo
c............off diagonal..........................
         do i=1,N
            do j=i+1,N
               xx=interval(idum,inn)
               y=interval(idum,inn)
               dc=cmplx(xx,y)
               dcbar=cmplx(xx,-y)
               call variationYM(dim,dimm,N,lambda,i,j,dc,dcbar,X,deltaS)
               if ( deltaS .gt. 0.0d0 ) then
                  z1=ran2(idum)
                  p1=dexp(-deltaS)
                  if ( z1 .lt. p1 ) then
                     X(lambda,i,j)=X(lambda,i,j)+dc
                     Accept=Accept+1.0d0
                  else
                     Reject=Reject+1.0d0
                  endif
               else
                  X(lambda,i,j)=X(lambda,i,j)+dc
                  Accept=Accept+1.0d0
               endif
               X(lambda,j,i)=dconjg(X(lambda,i,j))
            enddo
         enddo
      enddo

      return
      end

c........variation of the action..........................

      subroutine variationYM(dim,dimm,N,lambda,i,j,dc,dcbar,X,deltaS)
      implicit none
      integer dim,dimm,N,i,j,lambda,sigma,k,l,p,q
      double complex delta0,delta1,del2,del3,delta2
      double precision delta11,delta22,deltaS
      double complex X(dimm,N,N),dc,dcbar

      delta0=0.0d0
      do sigma=1,dim
```

```fortran
        if (sigma.ne.lambda)then
           do k=1,N
              delta0=delta0-X(sigma,i,k)*X(sigma,k,i)
     &              -X(sigma,j,k)*X(sigma,k,j)
           enddo
        endif
     enddo
     delta1=0.0d0
     delta1=delta1+dc*dcbar*delta0
     if (i.eq.j) then
        delta1=delta1+0.5d0*(dc*dc+dcbar*dcbar)*delta0
     endif
     do sigma=1,dim
        if (sigma.ne.lambda)then
           delta1=delta1+dc*dc*X(sigma,j,i)*X(sigma,j,i)
     &           +dcbar*dcbar*X(sigma,i,j)*X(sigma,i,j)
     &           +2.0d0*dc*dcbar*X(sigma,i,i)*X(sigma,j,j)
        endif
     enddo
     delta1=-N*delta1
     delta11=real(delta1)
     del2=0.0d0
     del3=0.0d0
     do sigma=1,dim
        do k=1,N
           do l=1,N
              del2=del2+2.0d0*X(sigma,i,k)*X(lambda,k,l)*X(sigma,l,j)
     &              -1.0d0*X(sigma,i,k)*X(sigma,k,l)*X(lambda,l,j)
     &              -1.0d0*X(lambda,i,k)*X(sigma,k,l)*X(sigma,l,j)
              del3=del3+2.0d0*X(sigma,j,k)*X(lambda,k,l)*X(sigma,l,i)
     &              -1.0d0*X(sigma,j,k)*X(sigma,k,l)*X(lambda,l,i)
     &              -1.0d0*X(lambda,j,k)*X(sigma,k,l)*X(sigma,l,i)
           enddo
        enddo
     enddo
     delta2=0.0d0
     delta2=-N*dcbar*del2-N*dc*del3
     delta22=real(delta2)
     deltaS=delta11+delta22

     return
     end

c........the jackknife estimator...............................

     subroutine jackknife_binning(TMC,f,average,error)
     implicit none
     integer i,j,TMC,zbin,nbin
     double precision xm
     double precision f(1:TMC),sumf,y(1:TMC)
     double precision sig0,sig,error,average

c..............TMC is the number of data points. sig0 is the standard deviation. sumf is the sum of all
c the data points f_i whereas xm is the average of f......
     sig0=0.0d0
     sumf=0.0d0
     do i=1,TMC
        sumf=sumf+f(i)
     enddo
     xm=sumf/TMC
c.... zbin is the number of elements we remove each time from the set of TMC data points. the minimum
c number we can remove is 1 whereas the maximum number we can remove is TMC-1.each time we remove zbin
c elements we end up with  nbin sets (or bins)...........
c     do zbin=1,TMC-1
     zbin=1
     nbin=int(TMC/zbin)
     sig=0.0d0
```

```fortran
      do i=1,nbin,1
c...  y(i) is the average of the elements in the ith bin.This bin contains TMC-zbin data points after we
had removed zbin elements. for zbin=1 we have nbin=TMC.In this case there are TMC bins and y_i=sum_{j#i}
x_j/(TMC-1). for zbin=2 we have nbin=TMC/2. In this case there are TMC/2 bins and y_i=  sum_jx_j/(TMC-2)-
x_{2i}/(TMC-2)-x_{2i-1}/(TMC-2)...
         y(i)=sumf
         do j=1,zbin
            y(i)=y(i)-f((i-1)*zbin+j )
         enddo
         y(i)= y(i)/(TMC-zbin)
c..........the standard deviation computed for the ith bin..............
         sig=sig+((nbin-1.0d0)/nbin)*(y(i)-xm)*(y(i)-xm)
      enddo
c.... the standard deviation computed for the set of all bins with fixed zbin.....
      sig=sig
c.................the error.................................
      sig=dsqrt(sig)
c.... we compare the result with the error obtained for the previous zbin, if it is larger, then this is
the new value of the error...
      if (sig0 .lt. sig) sig0=sig
c     enddo
c.... the final value of the error.....................................................
      error=sig0
      average=xm

      return
      end

c.............the random number generator ran2.................

      function ran2(idum)
      implicit none
      integer idum,IM1,IM2,IMM1,IA1,IA2,IQ1,IQ2,IR1,IR2,NTAB,NDIV
      real AM,EPS,RNMX
      double precision ran2
      parameter (IM1=2147483563,IM2=2147483399,AM=1./IM1,IMM1=IM1-1,
     &    IA1=40014,IA2=40692,IQ1=53668,IQ2=52774,IR1=12211,
     &    IR2=3791,NTAB=32,NDIV=1+IMM1/NTAB,EPS=1.2E-7,RNMX=1.-EPS)
      integer idum2,j,k,iv(NTAB),iy
      SAVE iv,iy,idum2
      DATA idum2/123456789/,iv/NTAB*0/,iy/0/

      if (idum.le.0) then
         idum=max(-idum,1)
         idum2=idum
         do j=NTAB+8,1,-1
            k=idum/IQ1
            idum=IA1*(idum-k*IQ1)-k*IR1
            if (idum.lt.0) idum=idum+IM1
            if (j.le.NTAB) iv(j)=idum
         enddo
         iy=iv(1)
      endif
      k=idum/IQ1
      idum=IA1*(idum-k*IQ1)-k*IR1
      if (idum.lt.0) idum=idum+IM1
      k=idum2/IQ2
      idum2=IA2*(idum2-k*IQ2)-k*IR2
      if (idum2.lt.0) idum2=idum2+IM2
      j=1+iy/NDIV
      iy=iv(j)-idum2
      iv(j)=idum
      if (iy.lt.1) iy=iy+IMM1
      ran2=min(AM*iy,RNMX)

      return
      end
```

```fortran
c.............interval...................................

      function interval(idum,inn)
      implicit none
      double precision interval,inn,ran2
      integer idum

      interval=ran2(idum)
      interval=interval+interval-1.0d0
      interval=interval*inn

      return
      end

c.........adjusting interval..............................

      subroutine adjust_inn(pa,inn,Reject,Accept)
      implicit none
      double precision inn,pa,Reject,Accept

c.....pa acceptance rate...................................
      pa=(Accept)/(Reject+Accept)
c........fixing the acceptance rate at 30 %..................
      if (pa.ge.0.30) inn=inn*1.20d0
      if (pa.le.0.25) inn=inn*0.80d0

      return
      end
```

```fortran
      program my_hybrid_ym
      implicit none
      integer d,N,i,j,k,lambda,idum,tt,time,timeT,tther,Tth
      parameter (d=4,N=4)
      parameter (Tth=2**10)
      double precision gamma,mm,alpha,inn,dt,interval
      double complex X(d,N,N),P(d,N,N)
      double precision actio,ham,kin,variationH
      double precision Reject,Accept,pa
      double precision varH(Tth),varH_average,varH_error
      double precision ac(Tth),ac_average,ac_error
      real x0

c..........initialization of random number generator...........

      idum=-148175
      x0=0.0
c...  seed should be set to a large odd integer according to the manual. secnds(x) gives number of
c seconds-x elapsed since midnight. the 2*int(secnds(x0)) is always even so seed is always odd....
      idum=idum-2*int(secnds(x0))

c...................testing molecular dynamics.....................

c      call hot(N,d,idum,inn,X,P)
c       call cold(N,d,X)
c       time=1
c       dt=0.01d0
c       timeT=100
c       do tt=1,timeT
c           call molecular_dynamics(N,d,dt,time,gamma,mm,alpha,X,P)
c           call action(d,N,X,P,alpha,mm,gamma,actio,ham,kin)
c           write(9,*)tt,actio,ham
c           write(*,*)tt,actio,ham
c       enddo

c.......parameters of molecular dynamics...........

      time=100
      dt=0.01d0

c..................parameters..............

      mm=0.0d0
      alpha=0.0d0
      do k=0,20
          gamma=2.1d0-k*0.1d0

c...........initialization of X and P..............

          inn=1.0d0
          call hot(N,d,idum,inn,X,P)
          call cold(N,d,X)

c...............accepts including flips, rejects and the acceptance rate pa...............

          Reject=0.0d0
          Accept=0.0d0
          pa=0.0d0

c..............thermalization................

          do tther=1,Tth
              call metropolis(N,d,gamma,mm,alpha,dt,time,X,P,Reject,Accept
     &          ,variationH)
          enddo

c.................monte carlo evolution....
```

```fortran
      do tther=1,Tth
         call metropolis(N,d,gamma,mm,alpha,dt,time,X,P,Reject,Accept
     &         ,variationH)
         pa=(Accept)/(Reject+Accept)
         call action(d,N,X,P,alpha,mm,gamma,actio,ham,kin)
         ac(tther)=actio
         varH(tther)=dexp(-variationH)
         write(10,*)tther,actio,ham,kin,variationH,pa
         write(*,*)tther,actio,ham,kin,variationH,pa
      enddo

c..............measurements................

      call jackknife_binning(Tth,varH,varH_average,varH_error)
      write(*,*)gamma,alpha,mm,varH_average,varH_error
      write(11,*)gamma,alpha,mm,varH_average,varH_error
      call jackknife_binning(Tth,ac,ac_average,ac_error)
      write(*,*)gamma,alpha,mm,ac_average,ac_error
      write(12,*)gamma,alpha,mm,ac_average,ac_error
      enddo

      return
      end

c................metropolis algorithm...............

      subroutine metropolis(N,d,gamma,mm,alpha,dt,time,X,P,Reject,Accept
     &      ,variationH)
      implicit none
      integer N,d,i,j,mu,nu,k,l,idum,time
      double precision gamma,mm,alpha,inn,dt,ran2,Reject,Accept
      double complex var(d,N,N),X(d,N,N),X0(d,N,N),P(d,N,N),P0(d,N,N)
      double precision variations,variationH,probabilityS,probabilityH,r
      double precision actio,ham,kin

c........Gaussian initialization.....

      call gaussian(d,N,P)

      X0=X
      P0=P
      call action(d,N,X,P,alpha,mm,gamma,actio,ham,kin)
      variationS=actio
      variationH=ham

c............molecular dynamics evolution.....

      call molecular_dynamics(N,d,dt,time,gamma,mm,alpha,X,P)

      call action(d,N,X,P,alpha,mm,gamma,actio,ham,kin)
      variationS=actio-variationS
      variationH=ham-variationH

c........metropolis accept-reject step.................

      if(variationH.lt.0.0d0)then
         accept=accept+1.0d0
      else
         probabilityH=dexp(-variationH)
         r=ran2(idum)
         if (r.lt.probabilityH)then
            accept=accept+1.0d0
         else
            X=X0
            P=P0
            Reject=Reject+1.0d0
```

```fortran
          endif
       endif

       return
       end

c...........actions and Hamiltonians.........

       subroutine action(d,N,X,P,alpha,mm,gamma,actio,ham,kin)
       implicit none
       integer d,N,mu,nu,i,j,k,l
       double complex X(d,N,N),P(d,N,N),ii,CS,action0,ham0,action1,
     &     actio0,action2,ham1
       double precision actio,ham,kin
       double precision mm,gamma,alpha

       ii=cmplx(0,1)
       actio0=cmplx(0,0)
       do mu =1,d
          do nu=mu+1,d
             action0=cmplx(0,0)
             do i=1,N
                do j=1,N
                   do k=1,N
                      do l=1,N
                   action0=action0+X(mu,i,j)*X(nu,j,k)*X(mu,k,l)*X(nu,l,i)
     &                          -X(mu,i,j)*X(mu,j,k)*X(nu,k,l)*X(nu,l,i)
                      enddo
                   enddo
                enddo
             enddo
             actio0=actio0+action0
          enddo
       enddo
       actio=real(actio0)
       actio=-N*gamma*actio

       ham1=cmplx(0,0)
       action2=cmplx(0,0)
       do mu =1,d
       ham0=cmplx(0,0)
       action1=cmplx(0,0)
       do i=1,N
          do j=1,N
             ham0=ham0+P(mu,i,j)*P(mu,j,i)
             action1=action1+X(mu,i,j)*X(mu,j,i)
          enddo
       enddo
       action2=action2+action1
       ham1=ham1+ham0
       enddo
       ham=0.5d0*real(ham1)
       kin=ham
       actio=actio+0.5d0*mm*real(action2)

       CS=0.0d0
       do i=1,N
          do j=1,N
             do k=1,N
                CS=CS+ii*X(1,i,j)*X(2,j,k)*X(3,k,i)
     &              -ii*X(1,i,j)*X(3,j,k)*X(2,k,i)
             enddo
          enddo
       enddo
       actio=actio+2.0d0*alpha*N*real(CS)
       ham=ham+actio
```

```fortran
          return
          end

c.......the force.............

          subroutine variation(N,d,gamma,mm,alpha,X,var)
          implicit none
          integer N,d,i,j,mu,nu,k,l
          double precision gamma,mm,alpha
          double complex var(d,N,N),X(d,N,N),ii

          ii=dcmplx(0,1)
          do mu=1,d
             do i=1,N
                do j=i,N
                 var(mu,i,j)=cmplx(0,0)
                   do nu=1,d
                      do k=1,N
                         do l=1,N
             var(mu,i,j)=var(mu,i,j)+2.0d0*X(nu,j,k)*X(mu,k,l)*X(nu,l,i)
     &                         -X(nu,j,k)*X(nu,k,l)*X(mu,l,i)
     &                         -X(mu,j,k)*X(nu,k,l)*X(nu,l,i)
                         enddo
                      enddo
                   enddo
                   var(mu,i,j)=-N*gamma*var(mu,i,j)+mm*X(mu,j,i)
                   if(mu.eq.1)then
                      do k=1,N
             var(mu,i,j)=var(mu,i,j)+2.0d0*ii*alpha*N*X(2,j,k)*X(3,k,i)
     &                     -2.0d0*ii*alpha*N*X(3,j,k)*X(2,k,i)
                      enddo
                   endif
                   if(mu.eq.2)then
                      do k=1,N
             var(mu,i,j)=var(mu,i,j)+2.0d0*ii*alpha*N*X(3,j,k)*X(1,k,i)
     &                     -2.0d0*ii*alpha*N*X(1,j,k)*X(3,k,i)
                      enddo
                   endif
                   if(mu.eq.3)then
                      do k=1,N
             var(mu,i,j)=var(mu,i,j)+2.0d0*ii*alpha*N*X(1,j,k)*X(2,k,i)
     &                     -2.0d0*ii*alpha*N*X(2,j,k)*X(1,k,i)
                      enddo
                   endif
                   var(mu,j,i)=conjg(var(mu,i,j))
                enddo
             enddo
          enddo

          return
          end

c.............leap frog..............

          subroutine molecular_dynamics(N,d,dt,time,gamma,mm,alpha,X,P)
          implicit none
          integer N,d,i,j,mu,nn,time
          double precision dt,gamma,mm,alpha
          double complex X(d,N,N),P(d,N,N),var(d,N,N)

          do nn=1,time
             call variation(N,d,gamma,mm,alpha,X,var)
             do mu=1,d
                do i=1,N
                   do j=i,N
                      P(mu,i,j)=P(mu,i,j)-0.5d0*dt*var(mu,i,j)
                      X(mu,i,j)=X(mu,i,j)+dt*conjg(P(mu,i,j))
```

```fortran
                X(mu,j,i)=conjg(X(mu,i,j))
               enddo
             enddo
           enddo
           call variation(N,d,gamma,mm,alpha,X,var)
           do mu=1,d
             do i=1,N
               do j=i,N
                 P(mu,i,j)=P(mu,i,j)-0.5d0*dt*var(mu,i,j)
                 P(mu,j,i)=conjg(P(mu,i,j))
               enddo
             enddo
           enddo
        enddo

        return
        end

c.........generation of Gaussian noise for the field P..................

        subroutine gaussian(d,N,P)
        implicit none
        integer d,N,mu,i,j,idum
        double precision pi,phi,r,ran2
        double complex ii,P(d,N,N)

        pi=dacos(-1.0d0)
        ii=cmplx(0,1)
        do mu=1,d
           do i=1,N
              phi=2.0d0*pi*ran2(idum)
              r=dsqrt(-2.0d0*dlog(1.0d0-ran2(idum)))
              P(mu,i,i)=r*dcos(phi)
           enddo
           do i=1,N
              do j=i+1,N
                 phi=2.0d0*pi*ran2(idum)
                 r=dsqrt(-1.0d0*dlog(1.0d0-ran2(idum)))
                 P(mu,i,j)=r*dcos(phi)+ii*r*dsin(phi)
                 P(mu,j,i)=conjg(P(mu,i,j))
              enddo
           enddo
        enddo

        return
        end

c........the jackknife estimator...................

        subroutine jackknife_binning(TMC,f,average,error)
        implicit none
        integer i,j,TMC,zbin,nbin
        double precision xm
        double precision f(1:TMC),sumf,y(1:TMC)
        double precision sig0,sig,error,average

        sig0=0.0d0
        sumf=0.0d0
        do i=1,TMC
           sumf=sumf+f(i)
        enddo
        xm=sumf/TMC
c       do zbin=1,TMC-1
        zbin=1
        nbin=int(TMC/zbin)
        sig=0.0d0
        do i=1,nbin,1
```

```fortran
          y(i)=sumf
          do j=1,zbin
             y(i)=y(i)-f((i-1)*zbin+j )
          enddo
          y(i)= y(i)/(TMC-zbin)
          sig=sig+((nbin-1.0d0)/nbin)*(y(i)-xm)*(y(i)-xm)
       enddo
       sig=sig
       sig=dsqrt(sig)
       if (sig0 .lt. sig) sig0=sig
c      enddo
       error=sig0
       average=xm

       return
       end

c.............the random number generator ran2.........

       function ran2(idum)
       implicit none
       integer idum,IM1,IM2,IMM1,IA1,IA2,IQ1,IQ2,IR1,IR2,NTAB,NDIV
       real AM,EPS,RNMX
       double precision ran2
       parameter (IM1=2147483563,IM2=2147483399,AM=1./IM1,IMM1=IM1-1,
     &     IA1=40014,IA2=40692,IQ1=53668,IQ2=52774,IR1=12211,
     &     IR2=3791,NTAB=32,NDIV=1+IMM1/NTAB,EPS=1.2E-7,RNMX=1.-EPS)
       integer idum2,j,k,iv(NTAB),iy
       SAVE iv,iy,idum2
       DATA idum2/123456789/,iv/NTAB*0/,iy/0/

       if (idum.le.0) then
          idum=max(-idum,1)
          idum2=idum
          do j=NTAB+8,1,-1
             k=idum/IQ1
             idum=IA1*(idum-k*IQ1)-k*IR1
             if (idum.lt.0) idum=idum+IM1
             if (j.le.NTAB) iv(j)=idum
          enddo
          iy=iv(1)
       endif
       k=idum/IQ1
       idum=IA1*(idum-k*IQ1)-k*IR1
       if (idum.lt.0) idum=idum+IM1
       k=idum2/IQ2
       idum2=IA2*(idum2-k*IQ2)-k*IR2
       if (idum2.lt.0) idum2=idum2+IM2
       j=1+iy/NDIV
       iy=iv(j)-idum2
       iv(j)=idum
       if (iy.lt.1) iy=iy+IMM1
       ran2=min(AM*iy,RNMX)

       return
       end

c........hot start...................

       subroutine hot(N,d,idum,inn,X,P)
       implicit none
       integer lambda,i,j,N,d,idum
       double complex X(d,N,N),P(d,N,N)
       double precision xx,y,inn,interval

       do lambda=1,d
          do i=1,N
```

```fortran
            do j=i,N
               if (j.ne.i) then
                  xx=interval(idum,inn)
                  y=interval(idum,inn)
                  X(lambda,i,j)=cmplx(xx,y)
                  X(lambda,j,i)=cmplx(xx,-y)
                  xx=interval(idum,inn)
                  y=interval(idum,inn)
                  P(lambda,i,j)=cmplx(xx,y)
                  P(lambda,j,i)=cmplx(xx,-y)
               else
                  xx=interval(idum,inn)
                  X(lambda,i,j)=xx
                  xx=interval(idum,inn)
                  P(lambda,i,j)=xx
               endif
            enddo
         enddo
      enddo

      return
      end

c.............interval..............

      function interval(idum,inn)
      implicit none
      double precision interval,inn,ran2
      integer idum

      interval=ran2(idum)
      interval=interval+interval-1.0d0
      interval=interval*inn

      return
      end

c......cold start.....................

      subroutine cold(N,d,X)
      implicit none
      integer lambda,i,j,N,d
      double complex X(d,N,N)

      do lambda=1,d
         do i=1,N
            do j=1,N
               X(lambda,i,j)=cmplx(0,0)
            enddo
         enddo
      enddo

      return
      end
```

```fortran
      program my_hybrid_scalar_fuzzy
      implicit none
      integer N,i,j,k,idum,tt,time,tther,Tth,cou,ttco,Tco,Tmc,nn
      parameter (N=6)
      parameter (Tth=2**10,Tmc=2**10,Tco=2**0)
      double precision a,b,c,at,bt,ct
      double complex phi(N,N),P(N,N),phi0(N,N)
      double precision actio,ham,kin,quad,quar,mag,variationH,ev(1:N)
      double precision Reject,Accept,pa,inn,dt,interval,xx,y,t_1,t_2
      double precision varH(Tmc),varH_average,varH_error
      double precision acti(Tmc),acti_average,acti_error
      double precision Cv(Tmc),Cv_average,Cv_error
      double precision ma(Tmc),ma_average,ma_error
      double precision chi(Tmc),chi_average,chi_error
      double precision p0(Tmc),p0_average,p0_error
      double precision pt(Tmc),pt_average,pt_error
      double precision kinet(Tmc),k_average,k_error
      double precision ide_average,ide_error
      double precision qu(Tmc),qu_average,qu_error
      double precision target_pa_high,target_pa_low,dt_max,dt_min,inc
     &      ,dec
      real x0

      call cpu_time(t_1)

c..........initialization of random number generator...........

      idum=-148175
      x0=0.0
      idum=idum-2*int(secnds(x0))

c.............parameters...................

      at=dsqrt(1.0d0*N)!1.0d0
      a=at/dsqrt(1.0d0*N)
      ct=1.0d0
      c=N*N*ct
      do k=0,0
         bt=-5.0d0+k*0.1d0
         b=N*dsqrt(1.0d0*N)*bt

c.............initialization of phi and P.....

         inn=1.0d0
         call hot(N,idum,inn,phi,P)

c.......parameters of molecular dynamics...........

         time=10
         dt=0.01d0

c................accepts including flips, rejects and the acceptance rate pa...............

         Reject=0.0d0
         Accept=0.0d0
         pa=0.0d0

c.....the acceptance rate is fixed in [0.7,0.9] such that dt is in [0.0001,1]....

         target_pa_high=0.90d0
         target_pa_low=0.70d0
         dt_max=1.0d0
         dt_min=0.0001d0
         inc=1.2d0
         dec=0.8d0
         nn=1
```

```fortran
c............thermalization.................................

      do tther=1,Tth
         call metropolis(N,a,b,c,dt,time,phi,P,Reject,Accept
     &         ,variationH,idum)
         call  action(N,phi,P,a,b,c,kin,quad,quar,actio,ham,mag)
         cou=tther
         call adjust_inn(cou,pa,dt,time,Reject,Accept,
     &         nn,target_pa_high,target_pa_low,dt_max,dt_min,inc,dec)
         write(*,*)tther,pa,dt,actio
      enddo

c.................monte carlo evolution....................

      do tther=1,Tmc

c...............removing auto-correlations by separating data points by tco monte carlo steps.....

         do ttco=1,Tco
            call metropolis(N,a,b,c,dt,time,phi,P,Reject,Accept
     &            ,variationH,idum)
         enddo

c...........constructing thermalized obervables as vectors.......

         call  action(N,phi,P,a,b,c,kin,quad,quar,actio,ham,mag)
         acti(tther)=actio
         ma(tther)=mag
         p0(tther)=mag*mag/N**2
         pt(tther)=quad/N
         kinet(tther)=kin
         qu(tther)=quar
         varH(tther)=dexp(-variationH)

c...........adjusting the step dt.................

         cou=tther
         call adjust_inn(cou,pa,dt,time,Reject,Accept,
     &         nn,target_pa_high,target_pa_low,dt_max,dt_min,inc,dec)
         write(*,*)tther,pa,dt,actio

c.........the eigenvalues of phi.................................................

         phi0=phi
         call eigenvalues(N,phi0,ev)
         write(62,*)tther,ev
      enddo

c..............measurements..................................................

c.....................energy................................................................
      call jackknife_binning(Tmc,acti,acti_average,acti_error)
      write(*,*)"action",a,bt,ct,acti_average,acti_error
      write(10,*)a,bt,ct,acti_average,acti_error
c.........specific heat Cv=<(S_i-<S>)^2>...........................
      do tther=1,Tmc
         Cv(tther)=0.0d0
         Cv(tther)=Cv(tther)+acti(tther)
         Cv(tther)=Cv(tther)-acti_average
         Cv(tther)=Cv(tther)*Cv(tther)
      enddo
      call jackknife_binning(Tmc,Cv,Cv_average,Cv_error)
      write(*,*)"specific heat",a,bt,ct,Cv_average,Cv_error
      write(20,*)a,bt,ct,Cv_average,Cv_error
c...............magnetization..................................................
      call jackknife_binning(Tmc,ma,ma_average,ma_error)
      write(*,*)"magnetization",a,bt,ct,ma_average,ma_error
```

```fortran
            write(30,*)a,bt,ct,ma_average,ma_error
c...............susceptibility....................................................
         do tther=1,Tmc
            chi(tther)=0.0d0
            chi(tther)=chi(tther)+ma(tther)
            chi(tther)=chi(tther)-ma_average
            chi(tther)=chi(tther)*chi(tther)
         enddo
         call jackknife_binning(Tmc,chi,chi_average,chi_error)
         write(*,*)"susceptibility", a,bt,ct,chi_average,chi_error
         write(40,*)a,bt,ct,chi_average,chi_error
c..............power in the zero mode...........................................
         call jackknife_binning(Tmc,p0,p0_average,p0_error)
         write(*,*)"zero power", a,bt,ct,p0_average,p0_error
         write(50,*)a,bt,ct,p0_average,p0_error
c..............total power=quadratic term/N.....................................
         call jackknife_binning(Tmc,pt,pt_average,pt_error)
         write(*,*)"total power=quadrtic/N",a,bt,ct,pt_average,pt_error
         write(60,*)a,bt,ct,pt_average,pt_error
c..............kinetic term.....................................................
         call jackknife_binning(Tmc,kinet,k_average,k_error)
         write(*,*)"kinetic",a,bt,ct,k_average,k_error
         write(70,*)a,bt,ct,k_average,k_error
c..............quartic term....
         call jackknife_binning(Tmc,qu,qu_average,qu_error)
         write(*,*)"quartic", a,bt,ct,qu_average,qu_error
         write(80,*)a,bt,ct,qu_average,qu_error
c..............schwinger-dyson identity.....................................
         ide_average=2.0d0*a*k_average+2.0d0*b*N*pt_average
     &        +4.0d0*c*qu_average
         ide_average=ide_average/(N*N)
         ide_error=2.0d0*a*k_error+2.0d0*b*N*pt_error
     &        +4.0d0*c*qu_error
         ide_error=ide_error/(N*N)
         write(*,*)"ide", a,bt,ct,ide_average,ide_error
         write(81,*)a,bt,ct,ide_average,ide_error
c................variation of hamiltonian.................................
         call jackknife_binning(Tmc,varH,varH_average,varH_error)
         write(*,*)"exp(-\Delta H)",a,bt,ct,varH_average,varH_error
         write(11,*)a,bt,ct,varH_average,varH_error
      enddo

c......................cpu time.............................................
      call cpu_time(t_2)
      write(*,*)"cpu_time=", t_2-t_1

      return
      end


c...................metropolis algorithm..........................

      subroutine metropolis(N,a,b,c,dt,time,phi,P,Reject,Accept
     &      ,variationH,idum)
      implicit none
      integer N,i,j,mu,nu,k,l,idum,time
      double precision a,b,c,inn,dt,ran2,Reject,Accept
      double complex var(N,N),phi(N,N),phi0(N,N),P(N,N),P0(N,N)
      double precision variations,variationH,probabilityS,probabilityH,r
      double precision actio,ham,kin,quad,quar,mag

c........Gaussian initialization, molecular dynamics evolution and variation of the Hamiltonian....
      call gaussian(idum,N,P)
      phi0=phi
      P0=P
      call action(N,phi,P,a,b,c,kin,quad,quar,actio,ham,mag)
      variationS=actio
      variationH=ham
```

```fortran
      call molecular_dynamics(N,dt,time,a,b,c,phi,P)
      call action(N,phi,P,a,b,c,kin,quad,quar,actio,ham,mag)
      variationS=actio-variationS
      variationH=ham-variationH
c...........metropolis accept-reject step.................
      if(variationH.lt.0.0d0)then
         accept=accept+1.0d0
      else
         probabilityH=dexp(-variationH)
         r=ran2(idum)
         if (r.lt.probabilityH)then
            accept=accept+1.0d0
         else
            phi=phi0
            P=P0
            Reject=Reject+1.0d0
         endif
      endif

      return
      end

c....................eigenvalues............................

      subroutine eigenvalues(N,phi0,ev)
      implicit none
      integer N,inf
      double complex cw(1:2*N-1)
      double precision rw(1:3*N-2)
      double complex phi0(1:N,1:N)
      double precision ev(1:N)

c.....LAPACK's zheev diagonalizes hermitian matrices...
      call zheev('N','U',N,phi0,N,ev,cw,2*N-1,rw,inf)

      return
      end

c...............actions and Hamiltonians...............................

      subroutine action(N,phi,P,a,b,c,kin,quad,quar,actio,ham,mag)
      implicit none
      integer N,mu,i,j,k,l
      double complex phi(N,N),P(N,N)
      double precision a,b,c
      double precision kin,quad,quar,actio,ham,mag
      double complex kine,quadr,quart,ham0
      double complex Lplus(1:N,1:N),Lminus(1:N,1:N),Lz(1:N,1:N)
      double complex X(1:3,1:N,1:N)

c..................kinetic term and mass term..................
      call SU2(N,X,Lplus,Lminus)
      kine=cmplx(0,0)
      do i=1,N
         do j=1,N
            do k=1,N
               do l=1,N
                  kine=kine+X(1,i,j)*phi(j,k)*X(1,k,l)*phi(l,i)
     &                    +X(2,i,j)*phi(j,k)*X(2,k,l)*phi(l,i)
     &                    +X(3,i,j)*phi(j,k)*X(3,k,l)*phi(l,i)
               enddo
            enddo
         enddo
      enddo
      kin=-2.0d0*real(kine)
      quadr=cmplx(0,0)
      do i=1,N
```

```fortran
      do j=1,N
         quadr=quadr+phi(i,j)*phi(j,i)
      enddo
   enddo
   kin=kin+0.5d0*(N*N-1.0d0)*real(quadr)
   quad=real(quadr)
c.....................quartic term.........................
   quart=cmplx(0,0)
   do i=1,N
      do j=1,N
         do k=1,N
            do l=1,N
               quart=quart+phi(i,j)*phi(j,k)*phi(k,l)*phi(l,i)
            enddo
         enddo
      enddo
   enddo
   quar=real(quart)
c.....................action...........................
   actio=a*kin+b*quad+c*quar
c.................Hamiltonian..............................
   ham0=cmplx(0,0)
   do i=1,N
      do j=1,N
         ham0=ham0+P(i,j)*P(j,i)
      enddo
   enddo
   ham=0.5d0*real(ham0)
   ham=ham+actio
c........................magnetization............................
   mag=0.0d0
   do i=1,N
      mag=mag+phi(i,i)
   enddo
   mag=dabs(mag)

   return
   end

c.................the force..........................................

   subroutine variation(N,a,b,c,phi,var)
   implicit none
   integer N,i,j,k,l,nu
   doubleprecision a,b,c
   doublecomplex var(N,N),var1(N,N),phi(N,N)
   doublecomplex Lplus(1:N,1:N),Lminus(1:N,1:N),Lz(1:N,1:N)
   doublecomplex X(1:3,1:N,1:N)

   call SU2(N,X,Lplus,Lminus)
   do i=1,N
      do j=i,N
         var(i,j)=cmplx(0,0)
         do k=1,N
            do l=1,N
               var(i,j)=var(i,j)+X(1,j,k)*phi(k,l)*X(1,l,i)
     &                 +X(2,j,k)*phi(k,l)*X(2,l,i)
     &                 +X(3,j,k)*phi(k,l)*X(3,l,i)
            enddo
         enddo
         var1(i,j)=cmplx(0,0)
         do k=1,N
            do l=1,N
               var1(i,j)=var1(i,j)+phi(j,k)*phi(k,l)*phi(l,i)
            enddo
         enddo
         var(i,j)=-4.0d0*a*var(i,j)+(N*N-1.0d0)*a*phi(j,i)
```

```fortran
     &             +2.0d0*b*phi(j,i)+4.0d0*c*var1(i,j)
             var(j,i)=conjg(var(i,j))
          enddo
       enddo

       return
       end

c..........SU(2) generators....................

       subroutine SU2(N,L,Lplus,Lminus)
       implicit none
       integer i,j,N
       double complex Lplus(1:N,1:N),Lminus(1:N,1:N),Lz(1:N,1:N)
       double complex L(1:3,1:N,1:N)
       double complex ii

       ii=cmplx(0,1)
       do i=1,N
          do j=1,N
             if( ( i + 1 ) .eq. j )then
                Lplus(i,j)  =dsqrt( ( N - i )*i*1.0d0 )
             else
                Lplus(i,j)=0.0d0
             endif
             if( ( i - 1 ) .eq. j )then
                Lminus(i,j)=dsqrt( ( N - j )*j*1.0d0  )
             else
                Lminus(i,j)=0.0d0
             endif
             if( i.eq.j)then
                Lz(i,j) = ( N + 1 - i - i )/2.0d0
             else
                Lz(i,j) = 0.0d0
             endif
             L(1,i,j)=0.50d0*(Lplus(i,j)+Lminus(i,j))
             L(2,i,j)=-0.50d0*ii*(Lplus(i,j)-Lminus(i,j))
             L(3,i,j)=Lz(i,j)
          enddo
       enddo

       return
       end

c..............leap frog...................................

       subroutine molecular_dynamics(N,dt,time,a,b,c,phi,P)
       implicit none
       integer N,i,j,nn,time
       double precision dt,a,b,c
       double complex phi(N,N),P(N,N),var(N,N),ii

       ii=cmplx(0,1)
       do nn=1,time
          call variation(N,a,b,c,phi,var)
          do i=1,N
             do j=i,N
                if (j.ne.i)then
                   P(i,j)=P(i,j)-0.5d0*dt*var(i,j)
                   phi(i,j)=phi(i,j)+dt*conjg(P(i,j))
                   phi(j,i)=conjg(phi(i,j))
                else
                   P(i,i)=P(i,i)-0.5d0*dt*var(i,i)
                   phi(i,i)=phi(i,i)+dt*conjg(P(i,i))
                   phi(i,i)=phi(i,i)-ii*aimag(phi(1,1))
                   endif
                enddo
```

```fortran
              enddo
c...........last step of leap frog.....................................
           call variation(N,a,b,c,phi,var)
           do i=1,N
              do j=i,N
                 if(j.ne.i)then
                     P(i,j)=P(i,j)-0.5d0*dt*var(i,j)
                     P(j,i)=conjg(P(i,j))
                 else
                     P(i,i)=P(i,i)-0.5d0*dt*var(i,i)
                     P(i,i)=P(i,i)-ii*aimag(P(i,i))
                 endif
              enddo
           enddo
        enddo

        return
        end

c.........generation of Gaussian noise for the field P.................

      subroutine gaussian(idum,N,P)
      implicit none
      integer N,mu,i,j,idum
      double precision pi,phi,r,ran2
      double complex ii,P(N,N)

      pi=dacos(-1.0d0)
      ii=cmplx(0,1)
      do i=1,N
         phi=2.0d0*pi*ran2(idum)
         r=dsqrt(-2.0d0*dlog(1.0d0-ran2(idum)))
         P(i,i)=r*dcos(phi)
      enddo
      do i=1,N
         do j=i+1,N
            phi=2.0d0*pi*ran2(idum)
            r=dsqrt(-1.0d0*dlog(1.0d0-ran2(idum)))
            P(i,j)=r*dcos(phi)+ii*r*dsin(phi)
            P(j,i)=conjg(P(i,j))
         enddo
      enddo

      return
      end

c........the jackknife estimator..................

      subroutine jackknife_binning(TMC,f,average,error)
      implicit none
      integer i,j,TMC,zbin,nbin
      double precision xm
      double precision f(1:TMC),sumf,y(1:TMC)
      double precision sig0,sig,error,average

      sig0=0.0d0
      sumf=0.0d0
      do i=1,TMC
         sumf=sumf+f(i)
      enddo
      xm=sumf/TMC
c     do zbin=1,TMC-1
      zbin=1
      nbin=int(TMC/zbin)
      sig=0.0d0
      do i=1,nbin,1
         y(i)=sumf
```

```fortran
        do j=1,zbin
            y(i)=y(i)-f((i-1)*zbin+j )
        enddo
        y(i)= y(i)/(TMC-zbin)
        sig=sig+((nbin-1.0d0)/nbin)*(y(i)-xm)*(y(i)-xm)
      enddo
      sig=sig
      sig=dsqrt(sig)
      if (sig0 .lt. sig) sig0=sig
c     enddo
      error=sig0
      average=xm

      return
      end

c.............the random number generator ran2.........

      function ran2(idum)
      implicit none
      integer idum,IM1,IM2,IMM1,IA1,IA2,IQ1,IQ2,IR1,IR2,NTAB,NDIV
      real AM,EPS,RNMX
      double precision ran2
      parameter (IM1=2147483563,IM2=2147483399,AM=1./IM1,IMM1=IM1-1,
     &    IA1=40014,IA2=40692,IQ1=53668,IQ2=52774,IR1=12211,
     &    IR2=3791,NTAB=32,NDIV=1+IMM1/NTAB,EPS=1.2E-7,RNMX=1.-EPS)
      integer idum2,j,k,iv(NTAB),iy
      SAVE iv,iy,idum2
      DATA idum2/123456789/,iv/NTAB*0/,iy/0/

      if (idum.le.0) then
          idum=max(-idum,1)
          idum2=idum
          do j=NTAB+8,1,-1
              k=idum/IQ1
              idum=IA1*(idum-k*IQ1)-k*IR1
              if (idum.lt.0) idum=idum+IM1
              if (j.le.NTAB) iv(j)=idum
          enddo
          iy=iv(1)
      endif
      k=idum/IQ1
      idum=IA1*(idum-k*IQ1)-k*IR1
      if (idum.lt.0) idum=idum+IM1
      k=idum2/IQ2
      idum2=IA2*(idum2-k*IQ2)-k*IR2
      if (idum2.lt.0) idum2=idum2+IM2
      j=1+iy/NDIV
      iy=iv(j)-idum2
      iv(j)=idum
      if (iy.lt.1) iy=iy+IMM1
      ran2=min(AM*iy,RNMX)

      return
      end

c........hot start...................

      subroutine hot(N,idum,inn,phi,P)
      implicit none
      integer lambda,i,j,N,d,idum
      double complex phi(N,N),P(N,N)
      double precision xx,y,inn,interval

      do i=1,N
          do j=i,N
              if (j.ne.i) then
```

```fortran
                xx=interval(idum,inn)
                y=interval(idum,inn)
                phi(i,j)=cmplx(xx,y)
                phi(j,i)=cmplx(xx,-y)
                xx=interval(idum,inn)
                y=interval(idum,inn)
                P(i,j)=cmplx(xx,y)
                P(j,i)=cmplx(xx,-y)
            else
                xx=interval(idum,inn)
                phi(i,j)=xx
                xx=interval(idum,inn)
                P(i,j)=xx
            endif
        enddo
      enddo

      return
      end

c.............interval..............

      function interval(idum,inn)
      implicit none
      double precision interval,inn,ran2
      integer idum

      interval=ran2(idum)
      interval=interval+interval-1.0d0
      interval=interval*inn

      return
      end

c......cold start....................

      subroutine cold(N,phi)
      implicit none
      integer lambda,i,j,N
      double complex phi(N,N)

      do i=1,N
         do j=1,N
            phi(i,j)=cmplx(0,0)
         enddo
      enddo

      return
      end

c.........adjusting interval..................

      subroutine adjust_inn(cou,pa,dt,time,Rejec,Accept,
     &     nn,target_pa_high,target_pa_low,dt_max,dt_min,inc,dec)
      implicit none
      double precision dt,pa,Rejec,Accept
      integer time,cou,cou1
      integer nn
      double precision target_pa_high,target_pa_low,dt_max,dt_min,inc,
     &     dec,rho1,rho2,dtnew

c.....pa acceptance rate............
      pa=(Accept)/(Rejec+Accept)
      cou1=mod(cou,nn)
      if (cou1.eq.0)then
c........fixing the acceptance rate between 90 % 70 %..................
        if (pa.ge.target_pa_high) then
```

```fortran
         dtnew=dt*inc
         if (dtnew.le.dt_max)then
             dt=dtnew
         else
             dt=dt_max
         endif
      endif
      if (pa.le.target_pa_low) then
         dtnew=dt*dec
         if (dtnew.ge.dt_min)then
             dt=dtnew
         else
             dt=dt_min
         endif
      endif
   endif

   return
   end
```

```fortran
      program my_phi_four_on_lattice
      implicit none
      integer N,idum,time,cou,nn,kk,ith,imc,ico,Tth,Tmc,Tco
      parameter (N=16)
      parameter (Tth=2**13,Tmc=2**14,Tco=2**3)
      double precision dt,kappa,g,phi(N,N),P(N,N),lambda_l,mu0_sq_l
      double precision mass,linear,kinetic,potential,act,Ham,variationH,
     &     quartic
      double precision target_pa_high,target_pa_low,dt_max,dt_min,inc
     &     ,dec,inn,pa,accept,reject
      real x0
      double precision ac(Tmc),ac_average,ac_error,cv(Tmc),cv_average,
     &  cv_error,lin(Tmc),lin_average,lin_error,susc(Tmc),susc_average,
     &  susc_error,ac2(Tmc),ac2_av,ac2_er,ac4(Tmc),ac4_av,ac4_er,binder,
     &  binder_e

c..........initialization of random number generator...........

      idum=-148175
      x0=0.0
      idum=idum-2*int(secnds(x0))

c.............parameters..................

      lambda_l=0.5d0
      do kk=0,15
         mu0_sq_l=-1.5d0+kk*0.1d0
         kappa=dsqrt(8.0d0*lambda_l+(4.0d0+mu0_sq_l)*(4.0d0+mu0_sq_l))
         kappa=kappa/(4.0d0*lambda_l)
         kappa=kappa-(4.0d0+mu0_sq_l)/(4.0d0*lambda_l)
         g=kappa*kappa*lambda_l

c.............initialization of phi and P.....

         inn=1.0d0
         call hot(N,idum,inn,phi,P)

c.......parameters of molecular dynamics...........

         time=10
         dt=0.01d0

c...............accepts including flips, rejects and the acceptance rate pa...............

         Reject=0.0d0
         Accept=0.0d0
         pa=0.0d0

c.....the acceptance rate is fixed in [0.7,0.9] such that dt is in [0.0001,1]....

         target_pa_high=0.90d0
         target_pa_low=0.70d0
         dt_max=1.0d0
         dt_min=0.0001d0
         inc=1.2d0
         dec=0.8d0
         nn=1

c...............thermalization......

         do ith=1,Tth
            call metropolis(time,dt,N,kappa,g,idum,accept,reject,
     &           variationH,P,phi)
            call adjust_inn(cou,pa,dt,time,Reject,Accept,
     &           nn,target_pa_high,target_pa_low,dt_max,dt_min,inc,dec)
            call action(N,kappa,g,P,phi,mass,linear,kinetic,potential,
     &           act,Ham,quartic)
```

```fortran
              write(9+kk,*) ith,act,Ham,variationH,pa,dt
            enddo

c..........Monte Carlo evolution.....

        do imc=1,Tmc
           do ico=1,Tco
              call metropolis(time,dt,N,kappa,g,idum,accept,reject,
     &             variationH,P,phi)
              call adjust_inn(cou,pa,dt,time,Reject,Accept,
     &             nn,target_pa_high,target_pa_low,dt_max,dt_min,inc,dec)
           enddo
           call action(N,kappa,g,P,phi,mass,linear,kinetic,potential,
     &          act,Ham,quartic)
           ac(imc)=act
           lin(imc)=dabs(linear)
           ac2(imc)=linear*linear
           ac4(imc)=linear*linear*linear*linear
           write(9+kk,*) imc+Tth,act,Ham,variationH,pa,dt
        enddo

c....................observables.........................

c..................action......................................
        call jackknife_binning(Tmc,ac,ac_average,ac_error)
        write(50,*)mu0_sq_l,lambda_l,kappa,g,ac_average,ac_error
c.................specific heat..................................
        do imc=1,Tmc
           cv(imc)=ac(imc)-ac_average
           cv(imc)=cv(imc)**(2.0d0)
        enddo
        call jackknife_binning(Tmc,cv,cv_average,cv_error)
        write(60,*)mu0_sq_l,lambda_l,kappa,g,cv_average,cv_error
c...............magnetization....................................
        call jackknife_binning(Tmc,lin,lin_average,lin_error)
        write(70,*)mu0_sq_l,lambda_l,kappa,g,lin_average,lin_error
c...............susceptibility................................
        do imc=1,Tmc
           susc(imc)=lin(imc)-lin_average
           susc(imc)=susc(imc)**(2.0d0)
        enddo
        call jackknife_binning(Tmc,susc,susc_average,susc_error)
        write(80,*)mu0_sq_l,lambda_l,kappa,g,susc_average,susc_error
c...............Binder cumulant.............................
        call jackknife_binning(Tmc,ac2,ac2_av,ac2_er)
        write(81,*)mu0_sq_l,lambda_l,kappa,g,ac2_av,ac2_er
        call jackknife_binning(Tmc,ac4,ac4_av,ac4_er)
        write(82,*)mu0_sq_l,lambda_l,kappa,g,ac4_av,ac4_er
        binder=1.0d0-ac4_av/(3.0d0*ac2_av*ac2_av)
        binder_e=-ac4_er/(3.0d0*ac2_av*ac2_av)
     &        +2.0d0*ac4_av*ac2_er/(3.0d0*ac2_av*ac2_av*ac2_av)
        write(90,*)mu0_sq_l,lambda_l,kappa,g,binder,binder_e
      enddo

      return
      end

      subroutine metropolis(time,dt,N,kappa,g,idum,accept,reject,
     &      variationH,P,phi)
      implicit none
      integer time,N,idum
      double precision dt,kappa,g,accept,reject,P(N,N),phi(N,N),
     &      variationH,P0(N,N),phi0(N,N),r,ran2,probability
      double precision mass,linear,kinetic,potential,act,Ham,quartic

      call gaussian(N,idum,P)
      P0=P
```

```fortran
      phi0=phi

      call action(N,kappa,g,P,phi,mass,linear,kinetic,potential,act,Ham,
     &      quartic)
      variationH=-Ham
      call leap_frog(time,dt,N,kappa,g,P,phi)
      call action(N,kappa,g,P,phi,mass,linear,kinetic,potential,act,Ham,
     &      quartic)
      variationH=variationH+Ham

      if (variationH.lt.0.0d0)then
         accept=accept+1.0d0
      else
         probability=dexp(-variationH)
         r=ran2(idum)
         if (r.lt.probability)then
            accept=accept+1.0d0
         else
            P=P0
            phi=phi0
            reject=reject+1.0d0
         endif
      endif

      return
      end

      subroutine gaussian(N,idum,P)
      implicit none
      integer N,i,j,idum
      double precision P(N,N),ph,r,pi,ran2

      pi=dacos(-1.0d0)
      do i=1,N
         do j=1,N
            r=dsqrt(-2.0d0*dlog(1.0d0-ran2(idum)))
            ph=2.0d0*pi*ran2(idum)
            P(i,j)=r*dcos(ph)
         enddo
      enddo

      return
      end

      subroutine leap_frog(time,dt,N,kappa,g,P,phi)
      implicit none
      integer time,N,nn,i,j
      double precision kappa,g,phi(N,N),P(N,N),force(N,N),dt

      do nn=1,time
         call scalar_force(N,phi,kappa,g,force)
         do i=1,N
            do j=1,N
               P(i,j)=P(i,j)-0.5d0*dt*force(i,j)
               phi(i,j)=phi(i,j)+dt*P(i,j)
            enddo
         enddo
         call scalar_force(N,phi,kappa,g,force)
         do i=1,N
            do j=1,N
               P(i,j)=P(i,j)-0.5d0*dt*force(i,j)
            enddo
         enddo
      enddo

      return
      end
```

```fortran
      subroutine scalar_force(N,phi,kappa,g,force)
      implicit none
      integer N,i,j,ip(N),im(N)
      double precision phi(N,N),kappa,g,force(N,N)
      double precision force1,force2,force3

      call ipp(N,ip)
      call imm(N,im)
      do i=1,N
         do j=1,N
            force1=phi(ip(i),j)+phi(im(i),j)+phi(i,ip(j))+phi(i,im(j))
            force1=-2.0d0*kappa*force1
            force2=2.0d0*phi(i,j)
            force3=phi(i,j)*(phi(i,j)*phi(i,j)-1.0d0)
            force3=4.0d0*g*force3
            force(i,j)=force1+force2+force3
         enddo
      enddo

      return
      end

      subroutine action(N,kappa,g,P,phi,mass,linear,kinetic,potential,
     &     act,Ham,quartic)
      implicit none
      integer N,i,j,ip(N)
      double precision kappa,g
      double precision phi(N,N),P(N,N),act,potential,mass,kinetic,
     &     kineticH,ham,linear,quartic

      call ipp(N,ip)
      kinetic=0.0d0
      mass=0.0d0
      kineticH=0.0d0
      potential=0.0d0
      linear=0.0d0
      quartic=0.0d0
      do i=1,N
         do j=1,N
            linear=linear+phi(i,j)
            quartic=quartic+phi(i,j)*phi(i,j)*phi(i,j)*phi(i,j)
            kinetic=kinetic+phi(i,j)*(phi(ip(i),j)+phi(i,ip(j)))
            mass=mass+phi(i,j)*phi(i,j)
            potential=potential
     &              +(phi(i,j)*phi(i,j)-1.0d0)*(phi(i,j)*phi(i,j)-1.0d0)
            kineticH=kineticH+P(i,j)*P(i,j)
         enddo
      enddo
      kinetic=-2.0d0*kappa*kinetic
      potential=g*potential
      act=kinetic+mass+potential
      kineticH=0.5d0*kineticH
      ham=kineticH+act

      return
      end

      subroutine ipp(N,ip)
      implicit none
      integer ip(N),i,N

      do i=1,N-1
         ip(i)=i+1
      enddo
      ip(N)=1
```

```fortran
        return
        end

        subroutine imm(N,im)
        implicit none
        integer im(N),i,N

        do i=2,N
           im(i)=i-1
        enddo
        im(1)=N

        return
        end

c........the jackknife estimator.................

        subroutine jackknife_binning(TMC,f,average,error)
        implicit none
        integer i,j,TMC,zbin,nbin
        double precision xm
        double precision f(1:TMC),sumf,y(1:TMC)
        double precision sig0,sig,error,average

        sig0=0.0d0
        sumf=0.0d0
        do i=1,TMC
           sumf=sumf+f(i)
        enddo
        xm=sumf/TMC
c       do zbin=1,TMC-1
        zbin=1
        nbin=int(TMC/zbin)
        sig=0.0d0
        do i=1,nbin,1
           y(i)=sumf
           do j=1,zbin
              y(i)=y(i)-f((i-1)*zbin+j )
           enddo
           y(i)= y(i)/(TMC-zbin)
           sig=sig+((nbin-1.0d0)/nbin)*(y(i)-xm)*(y(i)-xm)
        enddo
        sig=sig
        sig=dsqrt(sig)
        if (sig0 .lt. sig) sig0=sig
c       enddo
        error=sig0
        average=xm

        return
        end

c.............the random number generator ran2.........

        function ran2(idum)
        implicit none
        integer idum,IM1,IM2,IMM1,IA1,IA2,IQ1,IQ2,IR1,IR2,NTAB,NDIV
        real AM,EPS,RNMX
        double precision ran2
        parameter (IM1=2147483563,IM2=2147483399,AM=1./IM1,IMM1=IM1-1,
     &      IA1=40014,IA2=40692,IQ1=53668,IQ2=52774,IR1=12211,
     &      IR2=3791,NTAB=32,NDIV=1+IMM1/NTAB,EPS=1.2E-7,RNMX=1.-EPS)
        integer idum2,j,k,iv(NTAB),iy
        SAVE iv,iy,idum2
        DATA idum2/123456789/,iv/NTAB*0/,iy/0/

        if (idum.le.0) then
```

```fortran
          idum=max(-idum,1)
          idum2=idum
          do j=NTAB+8,1,-1
              k=idum/IQ1
              idum=IA1*(idum-k*IQ1)-k*IR1
              if (idum.lt.0) idum=idum+IM1
              if (j.le.NTAB) iv(j)=idum
          enddo
          iy=iv(1)
      endif
      k=idum/IQ1
      idum=IA1*(idum-k*IQ1)-k*IR1
      if (idum.lt.0) idum=idum+IM1
      k=idum2/IQ2
      idum2=IA2*(idum2-k*IQ2)-k*IR2
      if (idum2.lt.0) idum2=idum2+IM2
      j=1+iy/NDIV
      iy=iv(j)-idum2
      iv(j)=idum
      if (iy.lt.1) iy=iy+IMM1
      ran2=min(AM*iy,RNMX)

      return
      end

c........hot start...................

      subroutine hot(N,idum,inn,phi,P)
      implicit none
      integer lambda,i,j,N,idum
      double precision phi(N,N),P(N,N)
      double precision inn,interval

      do i=1,N
          do j=1,N
              phi(i,j)=interval(idum,inn)
              P(i,j)=interval(idum,inn)
          enddo
      enddo

      return
      end

c.........adjusting interval..................

      subroutine adjust_inn(cou,pa,dt,time,Reject,Accept,
     &    nn,target_pa_high,target_pa_low,dt_max,dt_min,inc,dec)
      implicit none
      double precision dt,pa,Reject,Accept
      integer time,cou,cou1
      integer nn
      double precision target_pa_high,target_pa_low,dt_max,dt_min,inc,
     &    dec,rho1,rho2,dtnew

c.....pa acceptance rate............
      pa=(Accept)/(Reject+Accept)
      cou1=mod(cou,nn)
      if (cou1.eq.0)then
c........fixing the acceptance rate between 90 % 70 %..................
          if (pa.ge.target_pa_high) then
              dtnew=dt*inc
              if (dtnew.le.dt_max)then
                  dt=dtnew
              else
                  dt=dt_max
              endif
          endif
```

```fortran
      if (pa.le.target_pa_low) then
         dtnew=dt*dec
         if (dtnew.ge.dt_min)then
            dt=dtnew
         else
            dt=dt_min
         endif
      endif
      endif

      return
      end

c.............interval..............

      function interval(idum,inn)
      implicit none
      double precision interval,inn,ran2
      integer idum

      interval=ran2(idum)
      interval=interval+interval-1.0d0
      interval=interval*inn

      return
      end
```

```fortran
      program my_metropolis_scalar_multitrace
      implicit none
      integer N,i,k,idum,ither,Tther,imont,ico,tmo,Tmont,Tco,counter,
     &      Pow1,Pow2,Pow3
      parameter (N=10)
      parameter (pow1=20,pow2=20,pow3=5)
      parameter (Tther=2**pow1,Tmont=2**pow2,Tco=2**pow3)
      double precision a,b,c,d,g,at,bt,ct,eta,v22,v41,v21,ap,bp,cp,dp,e,
     &      ep,fp
      double precision ran2,inn,interval,accept,reject,pa,t_1,t_2
      double precision lambda(N)
      double precision actio,actio0,sum1,sum2,sum4,sumv,actio1,actio2,
     &      actio4
      double precision ac(Tmont),ac_average,ac_error
      double precision id,ide(Tmont),ide_average,ide_error
      double precision cv(Tmont),cv_average,cv_error
      double precision va(Tmont),va_average,va_error
      double precision p0(Tmont),p0_average,p0_error
      double precision pt(Tmont),pt_average,pt_error
      double precision p4(Tmont),p4_average,p4_error
      double precision su(Tmont),su_average,su_error
      double precision sus(Tmont),sus_average,sus_error
      real x0

      call cpu_time(t_1)

c...........initialization of the random number generator........

      idum=-148175
      x0=0.0
      idum=idum-2*int(secnds(x0))

c............parameters of the model..................

c............kinetic parameter:the pure quartic matrix model is obtained by setting at=0............
      at=1.0d0
      a=at/dsqrt(1.0d0*N)
c.........Seamann's values..................
      v21=-1.0d0
      v22=0.0d0
      v41=1.5d0
c.........Ydri's proposal...................
c      v21=1.0d0
c      v22=1.0d0/8.0d0
c      v41=0.0d0
c...........principal multitrace coupling........................
      eta=v22-0.75d0*v41
      d=-2.0d0*eta*at*at*N
      d=d/3.0d0
      e=d
c..........further multitrace couplings (odd terms).................
      ap=4.0d0*at*at*v22/3.0d0
      dp=-2.0d0*at*at*v22/3.0d0
      dp=dp/N
      cp=-2.0d0*at*at*N*v41/3.0d0
      bp=-at*dsqrt(1.0d0*N)*v21/2.0d0
c.......ep and fp are included in c and b respectively....
      ep=at*at*N*N*v41/6.0d0
      fp=at*N*dsqrt(1.0d0*N)*v21/2.0d0
c.............quartic parameter: here c is C=c+ep of note..........................
      ct=1.0d0
      c=N*N*ct
c...........mass parameter: here b is B=b+fp of note..................
      do k=0,0
         bt=-5.0d0+k*0.1d0
         b=N*dsqrt(1.0d0*N)*bt
c......the parameters b and c in terms of g: the single parameter of the quartic matrix model........
```

```fortran
c       g=1.0d0
c       b=-N/g
c       c=N
c       c=c/(4.0d0*g)

c..................initialization of the eigenvalues...

        inn=1.0d0
        do i=1,N
            lambda(i)=interval(idum,inn)
        enddo

c...............accepts including flips, rejects and the acceptance rate pa...............

        Reject=0.0d0
        Accept=0.0d0
        pa=0.0d0

c.........thermalization....................................................

        do ither=1,Tther
            call standard_metropolis(N,ap,b,bp,c,cp,d,dp,ep,fp,lambda,
     &          accept,reject,idum,inn,pa)
            call action(N,ap,b,bp,c,cp,d,dp,ep,fp,lambda,actio,actio0,
     &          sum1,sum2,sum4,sumv,id,actio1,actio2,actio4)
            write(*,*)ither,actio0,actio,dabs(sum1),sum2,sum4,id,pa,inn
            write(7,*)ither,actio0,actio,dabs(sum1),sum2,sum4,sumv,id
     &          ,pa,inn
        enddo

c.......monte carlo evolution...................
        counter=0
        do imont=1,Tmont

c........removing auto-correlations by separating data points by tco monte carlo setps................

            do ico=1,Tco
                call standard_metropolis(N,ap,b,bp,c,cp,d,dp,ep,fp,lambda
     &              ,accept,reject,idum,inn,pa)
            enddo

c.........construction of thermalized observables...................................

            call action(N,ap,b,bp,c,cp,d,dp,ep,fp,lambda,actio,actio0,
     &          sum1,sum2,sum4,sumv,id,actio1,actio2,actio4)
c       if ((id.ge.0.8d0).and.(id.le.1.2d0))then
            counter=counter+1
            ac(counter)=actio0+actio1
            ide(counter)=id
            va(counter)=sumv
            su(counter)=dabs(sum1)
            p0(counter)=sum1*sum1/(N*N)
            pt(counter)=sum2/N
            p4(counter)=sum4
            write(*,*)imont,counter,sum2,sum4,id
            write(8,*)imont,counter,sum2,sum4,id

c.....................eigenvalues.........................

            write(150+k,*)counter,lambda
c       endif
        enddo

c...............measurements............
        Tmo=counter
c................action and vandermonde...................
        call jackknife_binning(Tmo,ac,ac_average,ac_error)
```

```fortran
      write(10,*)bt,ct,d,ac_average,ac_error
      call jackknife_binning(Tmo,va,va_average,va_error)
      write(11,*)bt,ct,d,va_average,va_error
c..................identity..................
      call jackknife_binning(Tmo,ide,ide_average,ide_error)
      write(12,*)bt,ct,d,ide_average,ide_error
      write(*,*)bt,ct,d,ide_average,ide_error, "identity"
c...........power in zero modes, total power and quartic term.............
      call jackknife_binning(Tmo,p0,p0_average,p0_error)
      write(13,*)bt,ct,d,p0_average,p0_error
      call jackknife_binning(Tmo,pt,pt_average,pt_error)
      write(14,*)bt,ct,d,pt_average,pt_error
      write(*,*)bt,ct,d,pt_average,pt_error, "total power"
      call jackknife_binning(Tmo,p4,p4_average,p4_error)
      write(15,*)bt,ct,d,p4_average,p4_error
c.......magnetization and susceptibility..............
      call jackknife_binning(Tmo,su,su_average,su_error)
      write(16,*)bt,ct,d,su_average,su_error
      do i=1,Tmo
         sus(i)= (su(i)-su_average)*(su(i)-su_average)
      enddo
      call jackknife_binning(Tmo,sus,sus_average,sus_error)
      write(17,*)bt,ct,d,sus_average,sus_error
c..................specific heat...................
      do i=1,Tmo
         cv(i)=(ac(i)-ac_average)**2
      enddo
      call jackknife_binning(Tmo,cv,cv_average,cv_error)
      write(20,*)bt,ct,d,cv_average,cv_error
      enddo

c..........cpu time and detail of simulation.....................
      call cpu_time(t_2)
      write(99,*)N,d,bt,ct,tmont,tmo,tco,tther,t_2-t_1

      return
      end

c.............metropolis algorithm..........................

      subroutine standard_metropolis(N,ap,b,bp,c,cp,d,dp,ep,fp,lambda
     &     ,accept,reject,idum,inn,pa)
      implicit none
      integer N,i,idum
      double precision lambda(N),var,pro,r,b,c,d,accept,reject,ran2,
     &     h,inn,interval,pa,ap,bp,cp,dp,ep,fp

      do i=1,N
c...........variation of the action...................
         h=interval(idum,inn)
         call variation(N,ap,b,bp,c,cp,d,dp,ep,fp,i,h,lambda,Var)
c...........metropolis accept-reject step........................
         if(var.gt.0.0d0)then
            pro=dexp(-var)
            r=ran2(idum)
            if (r.lt.pro) then
               lambda(i)=lambda(i)+h
               accept=accept+1.0d0
            else
               reject=reject+1.0d0
            endif
         else
            lambda(i)=lambda(i)+h
            accept=accept+1.0d0
         endif
      enddo
c...........adjusting the interval inn...............
```

```fortran
      call adjust_inn(pa,inn,Reject,Accept)

      return
      end

c....................variation of the action...........

      subroutine variation(N,ap,b,bp,c,cp,d,dp,ep,fp,i,h,lambda,Var)
      implicit none
      integer N,i,k
      double precision lambda(N),var,b,c,d,h,ap,bp,cp,dp,ep,fp
      double precision dsum2,dsum4,sum2,dvand,dd,dvande
      double precision sum1,sum3,var1,var2,var3,var4

      dsum2=h*h+2.0d0*h*lambda(i)
      dsum4=6.0d0*h*h*lambda(i)*lambda(i)
     &      +4.0d0*h*lambda(i)*lambda(i)*lambda(i)+4.0d0*h*h*h*lambda(i)
     &      +h*h*h*h
      sum3=0.0d0
      sum2=0.0d0
      sum1=0.0d0
      do k=1,N
         sum3=sum3+lambda(k)*lambda(k)*lambda(k)
         sum2=sum2+lambda(k)*lambda(k)
         sum1=sum1+lambda(k)
      enddo
      dvand=0.0d0
      do k=i+1,N
         dd=1.0d0
         dd=dd+h/(lambda(i)-lambda(k))
         dd=dabs(dd)
         dvand=dvand+dlog(dd)
      enddo
      dvand=-dvand
      dvande=0.0d0
      do k=1,i-1
         dd=1.0d0
         dd=dd+h/(lambda(i)-lambda(k))
         dd=dabs(dd)
         dvande=dvande+dlog(dd)
      enddo
      dvande=-dvande
      dvand=dvand+dvande
      dvand=2.0d0*dvand
      var=b*dsum2+c*dsum4+2.0d0*d*dsum2*sum2+d*dsum2*dsum2+dvand
      var1=h*h+2.0d0*h*sum1
      var4=var1*var1+2.0d0*sum1*sum1*var1
      var1=bp*var1
      var4=dp*var4
      var2=h*sum2+(sum1+h)*dsum2
      var2=ap*var2
      var3=3.0d0*h*lambda(i)*lambda(i)+3.0d0*h*h*lambda(i)+h*h*h
      var3=var3*(sum1+h)
      var3=var3+h*sum3
      var3=cp*var3
      var=var+var1+var2+var3+var4

      return
      end

c...............action.......................................

      subroutine action(N,ap,b,bp,c,cp,d,dp,ep,fp,lambda,actio,actio0,
     &      sum1,sum2,sum4,sumv,id,actio1,actio2,actio4)
      implicit none
      integer N,i,j
      double precision lambda(N),b,c,d,actio,actio0,sum1,sum2,sum4,sumv,
```

```fortran
     &      id
      double precision sum3,actio1,ap,bp,cp,dp,id1,ep,fp,actio2,actio4

c.............monomial terms............
      sum1=0.0d0
      sum2=0.0d0
      sum3=0.0d0
      sum4=0.0d0
      do i=1,N
         sum1=sum1+lambda(i)
         sum2=sum2+lambda(i)*lambda(i)
         sum3=sum3+lambda(i)*lambda(i)*lambda(i)
         sum4=sum4+lambda(i)*lambda(i)*lambda(i)*lambda(i)
      enddo
c.......the multitrace model without odd terms..........
      actio0=d*sum2*sum2+b*sum2+c*sum4
      actio=actio0
c............odd multitrace terms
      actio1=bp*sum1*sum1+cp*sum1*sum3+dp*sum1*sum1*sum1*sum1
     &      +ap*sum2*sum1*sum1
c...........the multitrace model with odd terms........
      actio=actio+actio1
c........adding the vandrmonde potential..............
      sumv=0.0d0
      do i=1,N
         do j=1,N
            if (i.ne.j)then
               sumv=sumv+dlog(dabs(lambda(i)-lambda(j)))
            endif
         enddo
      enddo
      sumv=-sumv
      actio=actio+sumv
c..........the quadratic and quartic corrections explicitly....
      actio2=fp*sum2+bp*sum1*sum1
      actio4=ep*sum4+d*sum2*sum2+cp*sum1*sum3+dp*sum1*sum1*sum1*sum1
     &      +ap*sum2*sum1*sum1
c...........the schwinger-dyson identity.................
      id=4.0d0*d*sum2*sum2+2.0d0*b*sum2+4.0d0*c*sum4
      id1=2.0d0*bp*sum1*sum1+4.0d0*(cp*sum1*sum3+dp*sum1*sum1*sum1*sum1
     &      +ap*sum2*sum1*sum1)
      id=id+id1
      id=id/(N*N)

      return
      end

c........the jackknife estimator...................

      subroutine jackknife_binning(TMC,f,average,error)
      implicit none
      integer i,j,TMC,zbin,nbin
      double precision xm
      double precision f(1:TMC),sumf,y(1:TMC)
      double precision sig0,sig,error,average

      sig0=0.0d0
      sumf=0.0d0
      do i=1,TMC
         sumf=sumf+f(i)
      enddo
      xm=sumf/TMC
c     do zbin=1,TMC-1
      zbin=1
      nbin=int(TMC/zbin)
      sig=0.0d0
      do i=1,nbin,1
```

```fortran
          y(i)=sumf
          do j=1,zbin
              y(i)=y(i)-f((i-1)*zbin+j )
          enddo
          y(i)= y(i)/(TMC-zbin)
          sig=sig+((nbin-1.0d0)/nbin)*(y(i)-xm)*(y(i)-xm)
      enddo
      sig=sig
      sig=dsqrt(sig)
      if (sig0 .lt. sig) sig0=sig
c     enddo
      error=sig0
      average=xm

      return
      end

c.............the random number generator ran2.........

      function ran2(idum)
      implicit none
      integer idum,IM1,IM2,IMM1,IA1,IA2,IQ1,IQ2,IR1,IR2,NTAB,NDIV
      real AM,EPS,RNMX
      double precision ran2
      parameter (IM1=2147483563,IM2=2147483399,AM=1./IM1,IMM1=IM1-1,
     &    IA1=40014,IA2=40692,IQ1=53668,IQ2=52774,IR1=12211,
     &    IR2=3791,NTAB=32,NDIV=1+IMM1/NTAB,EPS=1.2E-7,RNMX=1.-EPS)
      integer idum2,j,k,iv(NTAB),iy
      SAVE iv,iy,idum2
      DATA idum2/123456789/,iv/NTAB*0/,iy/0/

      if (idum.le.0) then
          idum=max(-idum,1)
          idum2=idum
          do j=NTAB+8,1,-1
              k=idum/IQ1
              idum=IA1*(idum-k*IQ1)-k*IR1
              if (idum.lt.0) idum=idum+IM1
              if (j.le.NTAB) iv(j)=idum
          enddo
          iy=iv(1)
      endif
      k=idum/IQ1
      idum=IA1*(idum-k*IQ1)-k*IR1
      if (idum.lt.0) idum=idum+IM1
      k=idum2/IQ2
      idum2=IA2*(idum2-k*IQ2)-k*IR2
      if (idum2.lt.0) idum2=idum2+IM2
      j=1+iy/NDIV
      iy=iv(j)-idum2
      iv(j)=idum
      if (iy.lt.1) iy=iy+IMM1
      ran2=min(AM*iy,RNMX)

      return
      end

c.........adjusting interval inn in such a way that the acceptance rate pa is fixed at 30 per
cent...................

      subroutine adjust_inn(pa,inn,Reject,Accept)
      implicit none
      double precision inn,pa,Reject,Accept

      pa=(Accept)/(Reject+Accept)
      if (pa.ge.0.30) inn=inn*1.20d0
      if (pa.le.0.25) inn=inn*0.80d0
```

```fortran
      return
      end

c.............the interval..................................

      function interval(idum,inn)
      implicit none
      doubleprecision interval,inn,ran2
      integer idum

      interval=ran2(idum)
      interval=interval+interval-1.0d0
      interval=interval*inn

      return
      end
```

```fortran
      program my_remez
      implicit none
      integer y,z,n,d,precision,i,counter,j,n0
      parameter(n0=100)
      double precision lambda_low, lambda_high,e,tolerance
      double precision a0,a(n0),b(n0),c0,c(n0),dd(n0),coefficient(n0)
      parameter (tolerance=0.0001d0)
      character*100 degree, com
      character*50 h1
      LOGICAL THERE

c........we choose the function to approximate, the range over which the rational approximation is to be
calculated, and the precision used....

      y=1
      z=2
      lambda_low=0.0004d0
      lambda_high=1.0d0
      precision=40
      print*, "Approximating the functions x^{y/z} and x^{-y/z}:"
     &      , "y=",y,"z=",z
      print*, "Approximation bounds:", lambda_low,lambda_high
      print*, "Precision of arithmetic:", precision
      write(*,*)"...................."

c.... we start the iteration on the degree of approximation at n=d=6....

      counter=0
      i=5
 14   i=i+1
      counter=counter+1
      print*, "ITERATION:",counter
      write(degree,'("", I1 )')i
      read(degree,'(i5)')n
      read(degree,'(i5)')d
      write(*,*)"degrees of approximation", n,d

c.........we call AlgRemez by the command="./test y z n d lambda_low lambda_high precision".....

      write(com,'(a,i5," ",i5," ",i5," ",i5," ",F10.5," ",F10.5," "
     &,i5," ",a)') "./test ",y,z,d,n,lambda_low,lambda_high
     &      ,precision,""
      print*, "command:", com
      call system(com)

c........we check whether or not the uniform norm is found.......................

      inquire(file='error1.dat', exist=THERE)
 11   if ( THERE ) then
         write(*,*) "file exists!"
      else
         go to  11
      end if

c......we read the uniform norm and test whether or not it is smaller than some tolerance, if it is not,
we go back and repeat with  increased degrees of approximation, viz n=n+1 and d=d+1.............

      open(unit=50+i,file='error1.dat',status='old')
      read(50+i,555) e
      write(*,*)"uniform norm", e
      write(*,*)"...................."
 555  format(1F20.10)
      close(50+i)
      if (e.gt.tolerance) go to 14

c..............the solution for x^{y/z}.............................................................
```

```fortran
      write(*,*)"rational approximation of x^{y/z}"
      open(unit=60,file='approx.dat',status='old')
      do j=1,2*n+1
         read(60,*)coefficient(j)
      enddo
      c0=coefficient(1)
      write(*,*)"c0=",c0
      do i=2,n+1
         c(i-1)=coefficient(i)
         dd(i-1)=coefficient(i+n)
         write(*,*)"i-1=",i-1,"c(i-1)=", c(i-1),"d(i-1)=",dd(i-1)
      enddo

c................the solution for x^{-y/z}.......................................

      write(*,*)"rational approximation of x^{-y/z}"
      open(unit=61,file='approx1.dat',status='old')
      do j=1,2*n+1
         read(61,*)coefficient(j)
      enddo
      a0=coefficient(1)
      write(*,*)"a0=",a0
      do i=2,n+1
         a(i-1)=coefficient(i)
         b(i-1)=coefficient(i+n)
         write(*,*)"i-1=",i-1,"a(i-1)=", a(i-1),"b(i-1)=",b(i-1)
      enddo

      return
      end
```

```fortran
      program my_conjugate_gradient
      implicit none
      integer N,M,i,j,counter,sig
      parameter (N=3,M=2)
      double precision A(N,N),v(N),sigma(M)
      double precision x(N),r(N),p(N),q(N),product,product1,product2,
     &      residue,tolerance
      double precision  alpha,beta,alpha_previous,beta_previous,xii,xii0,
     &      beta_sigma(M),alpha_sigma(M),xi(M),xi_previous(M)
      double precision x_sigma(N,M),p_sigma(N,M),r_sigma(N,M)
      parameter(tolerance=10.0d-100)

c............example input...........................

      call input(N,M,A,v,sigma)

c...............initialization..................................................

      do i=1,N
         x(i)=0.0d0
         r(i)=v(i)
         do sig=1,M
            x_sigma(i,sig)=0.0d0
         enddo
      enddo

c..............we start with alpha(0)=0, beta(-1)=1, xi^sigma(-1)=xi^sigma(0)=1, alpha^sigma(0)=0 and
beta^sigma(-1)=1...

      alpha=0.0d0
      beta=1.0d0
      do sig=1,M
         xi_previous(sig)=1.0d0
         xi(sig)=1.0d0
         alpha_sigma(sig)=0.0d0
         beta_sigma(sig)=1.0d0
      enddo

c.............starting iteration.........

      counter=0

c..............choosing search directions.................

 13   do i=1,N
         p(i)=r(i)+alpha*p(i)
         do sig=1,M
            p_sigma(i,sig)=xi(sig)*r(i)
     &            +alpha_sigma(sig)*p_sigma(i,sig)
         enddo
      enddo

c.......solving the sigma=0 problem.........

      product=0.0d0
      product1=0.0d0
c.......the only matrix-vector multiplication in the problem..........
      do i=1,N
         q(i)=0.0d0
         do j=1,N
            q(i)=q(i)+A(i,j)*p(j)
         enddo
         product=product+p(i)*q(i)
         product1=product1+r(i)*r(i)
      enddo
      beta_previous=beta
      beta=-product1/product
```

```fortran
      product2=0.0d0
      do i=1,N
         x(i)=x(i)-beta*p(i)
         r(i)=r(i)+beta*q(i)
         product2=product2+r(i)*r(i)
      enddo
      alpha_previous=alpha
      alpha=product2/product1

c.......solving the sigma problems..............

      do sig=1,M
c......the xi coefficients..........
         xii0=alpha_previous*beta*(xi_previous(sig)-xi(sig))
     &         +xi_previous(sig)*beta_previous*(1.0d0-sigma(sig)*beta)
         xii=xi(sig)*xi_previous(sig)*beta_previous/xii0
         xi_previous(sig)=xi(sig)
         xi(sig)=xii
c........the beta coefficients......
         beta_sigma(sig)=beta*xi(sig)/xi_previous(sig)
c.........the solutions and residues...........
         do i=1,N
            x_sigma(i,sig)=x_sigma(i,sig)-beta_sigma(sig)*p_sigma(i,sig)
            r_sigma(i,sig)=xi(sig)*r(i)
         enddo
c.......the alpha coefficients.......
         alpha_sigma(sig)=alpha
         alpha_sigma(sig)= alpha_sigma(sig)*xi(sig)*beta_sigma(sig)
         alpha_sigma(sig)=alpha_sigma(sig)/(xi_previous(sig)*beta)
      enddo

c......testing whether or not the interation should be continued........

      counter=counter+1
      residue=0.0d0
      do i=1,N
         residue=residue+r(i)*r(i)
      enddo
      residue=dsqrt(residue)
      if(residue.ge.tolerance)  go to 13

c........verification 1: if we set sigma=0 then xi must be equal 1 whereas the other pairs must be
equal.........
      write(*,*)"verification 1"
      write(*,*)counter,xi(1),xi_previous(1)
      write(*,*)counter,beta,beta_sigma(1)
      write(*,*)counter,alpha,alpha_sigma(1)

c............verification 2.....
      write(*,*)"verification 2"
      do i=1,N
         q(i)=0.0d0
         do j=1,N
            q(i)=q(i)+A(i,j)*x(j)
         enddo
      enddo
      write(*,*)"v",v
      write(*,*)"q",q

c............verification 3.....
      write(*,*)"verification 3"
      sig=1
      do i=1,N
         q(i)=sigma(sig)*x_sigma(i,sig)
         do j=1,N
            q(i)=q(i)+A(i,j)*x_sigma(j,sig)
```

```fortran
          enddo
       enddo
       write(*,*)"v",v
       write(*,*)"q",q

       return
       end

c................input.......................................

       subroutine input(N,M,A,v,sigma)
       implicit none
       integer N,M
       double precision A(N,N),v(N),sigma(M)

       a(1,1)=1.0d0
       a(1,2)=2.0d0
       a(1,3)=0.0d0
       a(2,1)=2.0d0
       a(2,2)=2.0d0
       a(2,3)=0.0d0
       a(3,1)=0.0d0
       a(3,2)=0.0d0
       a(3,3)=3.0d0
       v(1)=1.0d0
       v(2)=0.0d0
       v(3)=10.0d0

       sigma(1)=1.0d0
       sigma(2)=2.0d0

       return
       end
```

```fortran
      program my_hybrid_susy_ym
      implicit none
      integer dim,N,M,M0,i,j,k,sp,A1,idum,time,timeT,tmc0,TMC,TTH,idum0,
     &     cou,nn
      parameter (dim=4,N=8,M0=5,M=6)
      parameter (timeT=2**14,TTH=2**11,TMC=2**13)
      double precision gamma,mass,alpha,zeta,alphat
      double precision a0,a(M),b(M),c0,c(M0),d(M0),coefficient(2*M+1)
     &     ,epsilon
      double complex X(dim,N,N),P(dim,N,N),phi(2,N*N-1),Q(2,N*N-1),
     &     xx(2,N*N-1)
      double complex G(M,2,N*N-1),W(2,N*N-1),W0(2,N*N-1),xi(2,N*N-1)
      double precision inn,dt,interval, Rejec,Accept,pa
      double precision ham,action,actionB,actionF,kinB,kinF,
     &     variationH,YM,CS,HO,hamB,hamF
      real x0,t_1,t_2
      double complex var(dim,N,N),varF(dim,N,N)
      double precision varH0,varH(TMC),varH_average,varH_error
      double precision h(TMC),h_average,h_error
      double precision ac(TMC),ac_average,ac_error
      double precision ac_B(TMC),acB_average,acB_error
      double precision ac_F(TMC),acF_average,acF_error
      double precision ym0(TMC),ym_average,ym_error
      double precision cs0(TMC),cs_average,cs_error
      double precision ho0(TMC),ho_average,ho_error
      double precision identity_av,identity_er
      double precision target_pa_high,target_pa_low,dt_max,dt_min,inc,
     &     dec

      call cpu_time(t_1)

c............opening output files.................................................

      open(10, action='WRITE')
      close(10)
      open(11, action='WRITE')
      close(11)
      open(12, action='WRITE')
      close(12)
      open(13, action='WRITE')
      close(13)
      open(14, action='WRITE')
      close(14)
      open(15, action='WRITE')
      close(15)
      open(16, action='WRITE')
      close(16)
      open(17, action='WRITE')
      close(17)
      open(18, action='WRITE')
      close(18)

c........calling output of AlgRemez: M, M_0, c,d,a,b...................................

c.........rational approximation of x^{1/4}...................................
      open(unit=60,file='approx_x**+0.25_dat',status='old')
      do j=1,2*M0+1
         read(60,*)coefficient(j)
      enddo
      c0=coefficient(1)
c     write(*,*)"c0=",c0
      do i=2,M0+1
         c(i-1)=coefficient(i)
         d(i-1)=coefficient(i+M0)
c     write(*,*)"i-1=",i-1,"c(i-1)=", c(i-1),"d(i-1)=",d(i-1)
      enddo
c.........rational approximation of x^{-1/2}...................................
```

```fortran
      open(unit=60,file='approx_x**-0.5_dat',status='old')
      do j=1,2*M+1
          read(60,*)coefficient(j)
      enddo
      a0=coefficient(1)
c     write(*,*)"a0=",a0
      do i=2,M+1
          a(i-1)=coefficient(i)
          b(i-1)=coefficient(i+M)
c     write(*,*)"i-1=",i-1,"a(i-1)=", a(i-1),"b(i-1)=",b(i-1)
      enddo

c.....shifting the no sigma problem of the conjugate gradient to the smallest mass which is presumably
c the least convergent mass...

      epsilon=b(1)
      if (epsilon.gt.d(1))then
          epsilon=d(1)
      endif
      do i=1,M
          b(i)=b(i)-epsilon
      enddo
      do i=1,M0
          d(i)=d(i)-epsilon
      enddo

c...................initialization of random number generator....................

      idum=-148175
      x0=0
      idum=idum-2*int(secnds(x0))

c.............parameters........................................................................

      zeta=0.0d0
      mass=0.0d0
      gamma=1.0d0
      do k=0,0
          alphat=0.0d0-k*0.25d0
          alpha=alphat/dsqrt(1.0d0*N)

c.............initialization of X.........................................................

          inn=1.0d0
          call hot(N,dim,idum,inn,X)
c         call cold(N,dim,idum,X)

c.............initialization of the other fields from Gaussian noise...........

c     call gaussian(idum,dim,N,P)
c     call gaussian_plus(idum,N,Q)
c     call gaussian_plus(idum,N,xi)
c...............here we use the coefficients c and d not the coefficients a and b..............
c     call conjugate_gradient(dim,N,M0,zeta,X,c0,c,d,xi,G,phi,W,
c     &     epsilon)

c.............molecular dynamics parameters: dt should be optimized in such a way that the acceptance
c rate pa is fixed in [0.7,0.9] and dt is fixed in [0.0001,1]....

          time=10
          dt=0.001d0
          Rejec=0.0d0
          Accept=0.0d0
          target_pa_high=0.90d0
          target_pa_low=0.70d0
          dt_max=1.0d0
          dt_min=0.0001d0
```

```
        inc=1.2d0
        dec=0.8d0
        nn=1

c..........testing the molecular dynamics part.................

c       time=1
c       dt=0.001d0
c       do tmc0=1,timeT
c       call molecular_dynamics(N,dim,M,dt,time,gamma,mass,alpha,
c      &              zeta,a0,a,b,X,P,phi,Q,var,varF,epsilon)
c       call sub_action(dim,N,M,a0,a,b,X,P,phi,Q,alpha,mass,gamma,zeta,
c      &              ham,action,actionB,actionF,kinB,kinF,YM,CS,HO,epsilon)
c       hamB=kinB+actionB
c       hamF=kinF+actionF
c       write(*,*)tmc0,ham,kinB,actionB,hamB,kinF,actionF,hamF
c       write(7,*)tmc0,ham,kinB,actionB,hamB,kinF,actionF,hamF
c       enddo

c.................thermalization..............................

        do tmc0=1,TTH
           call metropolis(N,dim,M,M0,gamma,mass,alpha,zeta,dt,time,X,
     &           P,phi,Q,a0,a,b,c0,c,d,Rejec,Accept,var,varF,variationH,
     &           epsilon,idum)
           cou=tmc0
           call adjust_inn(cou,pa,dt,time,Rejec,Accept,
     &           nn,target_pa_high,target_pa_low,dt_max,dt_min,inc,dec)
           call sub_action(dim,N,M,a0,a,b,X,P,phi,Q,alpha,mass,gamma,
     &           zeta,ham,action,actionB,actionF,kinB,kinF,YM,CS,HO,
     &           epsilon)
           varH0=dexp(-variationH)
           write(*,*)tmc0,ham,action,actionB,kinB,actionF,kinF,
     &           variationH,varH0,pa
           write(8,*)tmc0,ham,action,actionB,kinB,actionF,kinF,
     &           variationH,varH0,pa
        enddo

c...................monte carlo evolution.....................

        do tmc0=1,TMC
           call metropolis(N,dim,M,M0,gamma,mass,alpha,zeta,dt,time,X,
     &           P,phi,Q,a0,a,b,c0,c,d,Rejec,Accept,var,varF,variationH,
     &           epsilon,idum)
           cou=tmc0
           call adjust_inn(cou,pa,dt,time,Rejec,Accept,
     &           nn,target_pa_high,target_pa_low,dt_max,dt_min,inc,dec)
           call sub_action(dim,N,M,a0,a,b,X,P,phi,Q,alpha,mass,gamma,
     &           zeta,ham,action,actionB,actionF,kinB,kinF,YM,CS,HO,
     &            epsilon)
           ym0(tmc0)=YM
           cs0(tmc0)=CS
           ho0(tmc0)=HO
           ac_B(tmc0)=actionB
           ac_F(tmc0)=actionF
           ac(tmc0)=action
           h(tmc0)=ham
           varH(tmc0)=dexp(-variationH)
           write(*,*)tmc0,ham,action,actionB,kinB,actionF,kinF,
     &           variationH, varH(tmc0),pa
           write(9,*)tmc0,ham,action,actionB,kinB,actionF,kinF,
     &           variationH,varH(tmc0),pa
        enddo

c...................measurements..............................

c.................the Hamiltonian.............................
```

```fortran
          call jackknife_binning(TMC,h,h_average,h_error)
c         write(*,*)alpha,gamma,mass,zeta,h_average,h_error
          open(10, status='OLD', action='WRITE', position='APPEND')
          write(10,*)alpha,gamma,mass,zeta,h_average,h_error
          close(10)
c..................we msut have <e^(-variationH)>=1...................................
          call jackknife_binning(TMC,varH,varH_average,varH_error)
c         write(*,*)alpha,gamma,mass,zeta,varH_average,varH_error
          open(11, status='OLD', action='WRITE', position='APPEND')
          write(11,*)alpha,gamma,mass,zeta,varH_average,varH_error
          close(11)
c...............the total action...................
          call jackknife_binning(TMC,ac,ac_average,ac_error)
c         write(*,*)alpha,gamma,mass,zeta,ac_average,ac_error
          open(12, status='OLD', action='WRITE', position='APPEND')
          write(12,*)alpha,gamma,mass,zeta,ac_average,ac_error
          close(12)
c..................the bosonic and pseudo-fermion actions and the yang-mills, chern-simons and harmonic
oscillator terms ....
          call jackknife_binning(TMC,ac_B,acB_average,acB_error)
c         write(*,*)alpha,gamma,mass,zeta,acB_average,acB_error
          open(13, status='OLD', action='WRITE', position='APPEND')
          write(13,*)alpha,gamma,mass,zeta,acB_average,acB_error
          close(13)
          call jackknife_binning(TMC,ym0,ym_average,ym_error)
c         write(*,*)alpha,gamma,mass,zeta,ym_average,ym_error
          open(14, status='OLD', action='WRITE', position='APPEND')
          write(14,*)alpha,gamma,mass,zeta,ym_average,ym_error
          close(14)
          call jackknife_binning(TMC,cs0,cs_average,cs_error)
c         write(*,*)alpha,gamma,mass,zeta,cs_average,cs_error
          open(15, status='OLD', action='WRITE', position='APPEND')
          write(15,*)alpha,gamma,mass,zeta,cs_average,cs_error
          close(15)
          call jackknife_binning(TMC,ho0,ho_average,ho_error)
c         write(*,*)alpha,gamma,mass,zeta,ho_average,ho_error
          open(16, status='OLD', action='WRITE', position='APPEND')
          write(16,*)alpha,gamma,mass,zeta,ho_average,ho_error
          close(16)
          call jackknife_binning(TMC,ac_F,acF_average,acF_error)
c         write(*,*)alpha,gamma,mass,zeta,acF_average,acF_error
          open(17, status='OLD', action='WRITE', position='APPEND')
          write(17,*)alpha,gamma,mass,zeta,acF_average,acF_error
          close(17)
c............for the flat space supersymmetric model for which xi=0 the Schwinger-Dyson identity
<4*gamma*YM+3*alpha*CS+2*mass*HO>=6(N^2-1) must hold...
          identity_av=4.0d0*gamma*ym_average+3.0d0*alpha*cs_average
     &         +2.0d0*mass*ho_average
          identity_av=identity_av/(6.0d0*(N*N-1.0d0))
          identity_av=identity_av-1.0d0
          identity_er=4.0d0*gamma*ym_error+3.0d0*alpha*cs_error
     &         +2.0d0*mass*ho_error
          identity_er=identity_er/(6.0d0*(N*N-1.0d0))
c         write(*,*)alpha,gamma,mass,zeta,identity_av,identity_er
          open(18, status='OLD', action='WRITE', position='APPEND')
          write(18,*)alpha,gamma,mass,zeta,identity_av,identity_er
          close(18)
       enddo

c...............cpu time......................................................

       call cpu_time(t_2)
       write(*,*)"cpu_time=", t_2-t_1

       return
       end
```

```fortran
c............the Metropolis algorithm.....................

      subroutine metropolis(N,dim,M,M0,gamma,mass,alpha,zeta,dt,time,X,P
     &     ,phi,Q,a0,a,b,c0,c,d,Rejec,Accept,var,varF,variationH,epsilon
     &     ,idum)
      implicit none
      integer N,dim,M,M0,i,j,mu,nu,k,l,idum,time,A1,sp
      double precision gamma,mass,alpha,zeta
      double precision inn,dt,ran2,Rejec,Accept
      double precision a0,a(M),b(M),c0,c(M0),d(M0),epsilon
      double complex X(dim,N,N),X0(dim,N,N),P(dim,N,N),
     &    P0(dim,N,N),phi(2,N*N-1),phi0(2,N*N-1),Q(2,N*N-1),Q0(2,N*N-1),
     &    xi(2,N*N-1),G(M,2,N*N-1),W(2,N*N-1),W0(2,N*N-1)
      double complex var(dim,N,N),varF(dim,N,N)
      double precision variations,variationH,probabilityS,probabilityH,r
      double precision ham,action,actionB,actionF,kinB,kinF,YM,CS,HO,
     &     hamB

c............Gaussian initialization............................

      call gaussian(idum,dim,N,P)
      call gaussian_plus(idum,N,Q)
      call gaussian_plus(idum,N,xi)
      phi=xi
      call conjugate_gradient(dim,N,M,zeta,X,c0,c,d,phi,G,W0,W,
     &     epsilon)
      phi=W0

c............saving the initial configurations............................

      X0=X
      P0=P
      phi0=phi
      Q0=Q
c...............evaluation of the initial value of hamiltonian and action..............

      call sub_action(dim,N,M,a0,a,b,X,P,phi,Q,alpha,mass,gamma,zeta,
     &     ham,action,actionB,actionF,kinB,kinF,YM,CS,HO,epsilon)
      hamB=actionB+kinB
      variationS=action
      variationH=ham

c..........molecular dynamics evolution..................................

      call molecular_dynamics(N,dim,M,dt,time,gamma,mass,alpha,zeta
     &     ,a0,a,b,X,P,phi,Q,var,varF,epsilon)

c..........evaluation of the final value of hamiltonian and action and the differences...............

      call sub_action(dim,N,M,a0,a,b,X,P,phi,Q,alpha,mass,gamma,zeta,
     &     ham,action,actionB,actionF,kinB,kinF,YM,CS,HO,epsilon)
      hamB=actionB+kinB
      variationS=action-variationS
      variationH=ham-variationH

c............metropolis accept-reject step....................................

      if(variationH.lt.0.0d0)then
         accept=accept+1.0d0
      else
         probabilityH=dexp(-variationH)
         r=ran2(idum)
         if (r.lt.probabilityH)then
            accept=accept+1.0d0
         else
            X=X0
            P=P0
```

```fortran
            phi=phi0
            Q=Q0
            Rejec=Rejec+1.0d0
         endif
      endif

      return
      end

c..............the leap frog algorithm..............................

      subroutine molecular_dynamics(N,dim,M,dt,time,gamma,mass,alpha,
     &      zeta,a0,a,b,X,P,phi,Q,var,varF,epsilon)
      implicit none
      integer N,dim,M,i,j,mu,nn,time,A1,A1b,sp
      double precision dt,gamma,mass,alpha,zeta,a0,a(M),b(M),epsilon,
     &      alp
      double complex X(dim,N,N),phi(2,N*N-1),P(dim,N,N),Q(2,N*N-1),
     &      xx(2,N*N-1),var(dim,N,N),varF(dim,N,N),G(M,2,N*N-1),
     &      W(2,N*N-1),W0(2,N*N-1)

      alp=1.0d0
      do nn=1,time
         call conjugate_gradient(dim,N,M,zeta,X,a0,a,b,phi,G,W0,W,
     &         epsilon)
         call  boson_force(N,dim,gamma,mass,alpha,X,var)
         call fermion_force(N,dim,M,zeta,a0,a,b,X,G,varF)
         do i=1,N
            do j=i,N
               do mu=1,dim
                  P(mu,i,j)=P(mu,i,j)-0.5d0*alp*dt*var(mu,i,j)
     &                  -0.5d0*alp*dt*varF(mu,i,j)
                  X(mu,i,j)=X(mu,i,j)+alp*dt*conjg(P(mu,i,j))
                  X(mu,j,i)=conjg(X(mu,i,j))
               enddo
            enddo
         enddo
         do A1=1,N*N-1
            do sp=1,2
               Q(sp,A1)=Q(sp,A1)-0.5d0*alp*dt*W(sp,A1)
               phi(sp,A1)=phi(sp,A1)+alp*dt*conjg(Q(sp,A1))
            enddo
         enddo
c....................last step of the leap frog......
      call conjugate_gradient(dim,N,M,zeta,X,a0,a,b,phi,G,W0,W,
     &      epsilon)
      call  boson_force(N,dim,gamma,mass,alpha,X,var)
      call fermion_force(N,dim,M,zeta,a0,a,b,X,G,varF)

      do i=1,N
         do j=i,N
            do mu=1,dim
               P(mu,i,j)=P(mu,i,j)-0.5d0*alp*dt*var(mu,i,j)
     &               -0.5d0*alp*dt*varF(mu,i,j)
               P(mu,j,i)=conjg(P(mu,i,j))
            enddo
         enddo
      enddo
      do A1=1,N*N-1
         do sp=1,2
            Q(sp,A1)=Q(sp,A1)-0.5d0*alp*dt*W(sp,A1)
         enddo
      enddo
      enddo

      return
      end
```

```fortran
c.......the conjugate gradient method..............

      subroutine conjugate_gradient(dim,N,M,zeta,X,a0,a,b,phi,G,W0,W,
     &    epsilon)
      implicit none
      integer dim,N,M,M0,i,j,counter,A1,sig,sp
      double precision zeta,a0,a(M),b(M),tol,tol0,residue,residue0,
     &    epsilon
      double complex X(dim,N,N)
      double complex xx(2,N*N-1),phi(2,N*N-1),r(2,N*N-1),p(2,N*N-1),
     &    q(2,N*N-1),o(2,N*N-1),xx1(2,N*N-1),q_previous(2,N*N-1)
      double complex  x_traceless_vec(2,N*N-1),y_traceless_vec(2,N*N-1),
     &    z_traceless_vec(2,N*N-1)
      double complex G(M,2,N*N-1),p_sigma(M,2,N*N-1),W(2,N*N-1),
     &    W0(2,N*N-1), G0(M,2,N*N-1)
      double precision rho,rho_previous,rho_sigma(M),beta,beta_previous,
     &    beta_sigma(M),xii0,xii,xi(M),xi_previous(M)
      double precision product,product1,product2
      parameter(tol=10.0d-5,tol0=10.0d-3)

c.........initialization.................

      do A1=1,N*N-1
         do sp=1,2
            xx(sp,A1)=cmplx(0,0)
            r(sp,A1)=phi(sp,A1)
            do sig=1,M
               G(sig,sp,A1)=cmplx(0,0)
            enddo
            q(sp,A1)=cmplx(0,0)
         enddo
      enddo

c..............initialization of the coefficients...........

      rho=0.0d0
      beta=1.0d0
      do sig=1,M
         xi_previous(sig)=1.0d0
         xi(sig)=1.0d0
         rho_sigma(sig)=0.0d0
         beta_sigma(sig)=1.0d0
      enddo

c...........starting the iteration.......................................

      counter=0

c.........choosing search directions...............................

 13   do A1=1,N*N-1
         do sp=1,2
            p(sp,A1)=r(sp,A1)+rho*p(sp,A1)
            do sig=1,M
               p_sigma(sig,sp,A1)=xi(sig)*r(sp,A1)
     &              +rho_sigma(sig)*p_sigma(sig,sp,A1)
            enddo
         enddo
      enddo

c......solving the no-sigma problem.....

c........performing the only vector-matrix multiplication in the conjugate gradient method...
c     q(i)=0.0d0
c     do j=1,2*(N*N-1)
c     q(i)=q(i)+(Delta(i,j)+epsilon*delta(i,j))*p(j)
```

```fortran
c       enddo
        call multiplication(dim,N,M,zeta,X,p,y_traceless_vec)
        o=y_traceless_vec
c          write(*,*)"o",o
        call multiplication_plus(dim,N,M,zeta,X,o,z_traceless_vec)
        q_previous=q
        q=z_traceless_vec
        q=q+epsilon*p
c       write(*,*)"q",q
c................calculating the beta coefficient......
        product=0.0d0
        product1=0.0d0
        do A1=1,N*N-1
           do sp=1,2
              product=product+conjg(p(sp,A1))*q(sp,A1)
              product1=product1+conjg(r(sp,A1))*r(sp,A1)
           enddo
        enddo
        beta_previous=beta
        beta=-product1/product
c...............calculating the solution xx, its residue and the rho coefficient.....
        product2=0.0d0
        do A1=1,N*N-1
           do sp=1,2
              xx(sp,A1)=xx(sp,A1)-beta*p(sp,A1)
              r(sp,A1)=r(sp,A1)+beta*q(sp,A1)
              product2=product2+conjg(r(sp,A1))*r(sp,A1)
           enddo
        enddo
        rho_previous=rho
        rho=product2/product1

c.......solving the sigma problems..............

        do sig=1,M
c.........the xi coefficients...................
        xii0=rho_previous*beta*(xi_previous(sig)-xi(sig))+
     &        xi_previous(sig)*beta_previous*(1.0d0-b(sig)*beta)
        xii=xi(sig)*xi_previous(sig)*beta_previous/xii0
        xi_previous(sig)=xi(sig)
        xi(sig)=xii
c.........the beta coefficients...........................
        beta_sigma(sig)=beta*xi(sig)/xi_previous(sig)
c.........the solutions......................
        do A1=1,N*N-1
           do sp=1,2
           G(sig,sp,A1)=G(sig,sp,A1)-beta_sigma(sig)*p_sigma(sig,sp,A1)
           enddo
        enddo
c........the alpha coefficients:alpha=rho..
         rho_sigma(sig)=rho
         rho_sigma(sig)=rho_sigma(sig)*xi(sig)*beta_sigma(sig)
         rho_sigma(sig)=rho_sigma(sig)/(xi_previous(sig)*beta)
        enddo

c......testing whether or not we continue the iteration................

        residue=0.0d0
        do A1=1,N*N-1
           do sp=1,2
              residue=residue+conjg(r(sp,A1))*r(sp,A1)
           enddo
        enddo
        residue=dsqrt(residue)
        counter=counter+1
        if(residue.ge.tol) go to 13
c       write(*,*)counter,residue
```

```fortran
c...........computing the pseudo-fermions W and W0......................

      do A1=1,N*N-1
         do sp=1,2
            W0(sp,A1)=cmplx(0,0)
            do sig=1,M
               W0(sp,A1)=W0(sp,A1)+a(sig)*G(sig,sp,A1)
            enddo
            W0(sp,A1)=W0(sp,A1)+a0*phi(sp,A1)
            W(sp,A1)=conjg(W0(sp,A1))
         enddo
      enddo

c......verification of Delta.xx=phi.....................
c     write(*,*)"phi",phi
c     write(*,*)"........................"
c     call multiplication(dim,N,M,zeta,X,xx,y_traceless_vec)
c     o=y_traceless_vec
c     write(*,*)"o",o
c     call multiplication_plus(dim,N,M,zeta,X,o,z_traceless_vec)
c     q=z_traceless_vec
c...............we must have q=phi since Delta.xx=phi....
c     write(*,*)"q",q
c     write(*,*)".............................."

c......verification of (Delta+b(sigma)).G_sigma=phi....................
c     sig=1
c              call reverse_identification(N,M,sig,G,x_traceless_vec)
c     xx1=x_traceless_vec
c     call multiplication(dim,N,M,zeta,X,xx1,y_traceless_vec)
c     o=y_traceless_vec
c     write(*,*)"o",o
c     call multiplication_plus(dim,N,M,zeta,X,o,z_traceless_vec)
c     q=z_traceless_vec+b(sig)*xx1
c...............we must have q=phi ....
c      write(*,*)"q",q
c     write(*,*)phi(1,1),q(1,1)
c     write(*,*)".........................."

      return
      end

c.........actions and Hamiltonians............................

      subroutine sub_action(dim,N,M,a0,a,b,X,P,phi,Q,alpha,mass,gamma,
     &     zeta,ham,action,actionB,actionF,kinB,kinF,YM,CS,HO,epsilon)
      implicit none
      integer dim,N,M,mu,nu,i,j,k,l,A1,sp
      double complex X(dim,N,N),P(dim,N,N),phi(2,N*N-1),Q(2,N*N-1),
     &W(2,N*N-1),W0(2,N*N-1),G(M,2,N*N-1)
      double complex ii,action0,action1,action2,ham0,ym0,cs0,ho0,
     &     kin0,kin1
      double precision action,actionB,actionF,ham,kinB,kinF,YM,CS,HO,
     &a0,a(M),b(M),epsilon
      double precision mass,gamma,alpha,zeta

      ii=cmplx(0,1)

c................yang-mills action........................

      ym0=cmplx(0,0)
      do mu =1,dim
         do nu=mu+1,dim
            action0=cmplx(0,0)
            do i=1,N
               do j=1,N
```

```fortran
                    do k=1,N
                        do l=1,N
                    action0=action0+X(mu,i,j)*X(nu,j,k)*X(mu,k,l)*X(nu,l,i)
     &                          -X(mu,i,j)*X(mu,j,k)*X(nu,k,l)*X(nu,l,i)
                        enddo
                    enddo
                enddo
            enddo
        ym0=ym0+action0
        enddo
        enddo
        action=real(ym0)
        YM=-N*action
        action=-N*gamma*action

c...........the harmonic oscillator and the bosonic kinetic terms..........

        kin0=cmplx(0,0)
        ho0=cmplx(0,0)
        do mu =1,dim
            ham0=cmplx(0,0)
            action1=cmplx(0,0)
            do i=1,N
                do j=1,N
                    ham0=ham0+P(mu,i,j)*P(mu,j,i)
                    action1=action1+X(mu,i,j)*X(mu,j,i)
                enddo
            enddo
        kin0=kin0+ham0
        ho0=ho0+action1
        enddo
        kinB=0.5d0*real(kin0)
        ham=kinB
        HO=0.5d0*real(ho0)
        action=action+0.5d0*mass*real(ho0)

c..........the chern-simons term.............................................

        cs0=cmplx(0,0)
        do i=1,N
            do j=1,N
                do k=1,N
                    cs0=cs0+ii*X(1,i,j)*X(2,j,k)*X(3,k,i)
     &                   -ii*X(1,i,j)*X(3,j,k)*X(2,k,i)
                enddo
            enddo
        enddo
        CS=2.0d0*N*real(cs0)
        action=action+2.0d0*alpha*N*real(cs0)
        ham=ham+action
        actionB=action

c...............fermion contribution.....

        call conjugate_gradient(dim,N,M,zeta,X,a0,a,b,phi,G,W0,W,
     &      epsilon)
        action2=cmplx(0,0)
        kin1=cmplx(0,0)
        do A1=1,N*N-1
            do sp=1,2
                action2=action2+W(sp,A1)*phi(sp,A1)
                kin1=kin1+conjg(Q(sp,A1))*Q(sp,A1)
            enddo
        enddo
        actionF=real(action2)
        kinF=real(kin1)
        action=actionB+actionF
```

```fortran
      ham=ham+kinF+actionF

      return
      end

c...........the Boson force................

      subroutine boson_force(N,dim,gamma,mass,alpha,X,var)
      implicit none
      integer N,dim,i,j,mu,nu,k,l
      double precision gamma,mass,alpha
      double complex var(dim,N,N),X(dim,N,N),ii

      ii=cmplx(0,1)
      do mu=1,dim
         do i=1,N
            do j=i,N
               var(mu,i,j)=cmplx(0,0)
               do nu=1,dim
                  do k=1,N
                     do l=1,N
         var(mu,i,j)=var(mu,i,j)+2.0d0*X(nu,j,k)*X(mu,k,l)*X(nu,l,i)
     &                    -X(nu,j,k)*X(nu,k,l)*X(mu,l,i)
     &                    -X(mu,j,k)*X(nu,k,l)*X(nu,l,i)
                     enddo
                  enddo
               enddo
            var(mu,i,j)=-N*gamma*var(mu,i,j)+mass*X(mu,j,i)
              if(mu.eq.1)then
                 do k=1,N
            var(mu,i,j)=var(mu,i,j)+2.0d0*ii*alpha*N*X(2,j,k)*X(3,k,i)
     &                    -2.0d0*ii*alpha*N*X(3,j,k)*X(2,k,i)
                 enddo
              endif
              if(mu.eq.2)then
                 do k=1,N
            var(mu,i,j)=var(mu,i,j)+2.0d0*ii*alpha*N*X(3,j,k)*X(1,k,i)
     &                    -2.0d0*ii*alpha*N*X(1,j,k)*X(3,k,i)
                 enddo
              endif
              if(mu.eq.3)then
                 do k=1,N
            var(mu,i,j)=var(mu,i,j)+2.0d0*ii*alpha*N*X(1,j,k)*X(2,k,i)
     &                    -2.0d0*ii*alpha*N*X(2,j,k)*X(1,k,i)
                 enddo
              endif
             var(mu,j,i)=conjg(var(mu,i,j))
            enddo
         enddo
      enddo

      return
      end

c............the Fermion force...........................

      subroutine fermion_force(N,dim,M,zeta,a0,a,b,X,G,varF)
      implicit none
      integer N,M,dim,sig,i,j,k
      double complex X(dim,N,N),phi(2,N*N-1)
      double precision a0,a(M),b(M),zeta
      double complex T(dim),S(dim),varF(dim,N,N),ii
      double complex G(M,2,N*N-1),G_vec(2,N*N),Gm(2,N,N),F_vec(2,N*N)
     &      ,Fm(2,N,N),W(2,N*N-1),W0(2,N*N-1)
      double complex x_traceless_vec(2,N*N-1),y_traceless_vec(2,N*N-1)

      ii=cmplx(0,1)
```

```fortran
      do i=1,N
         do j=i,N
            varF(1,i,j)=cmplx(0,0)
            varF(2,i,j)=cmplx(0,0)
            varF(3,i,j)=cmplx(0,0)
            varF(4,i,j)=cmplx(0,0)
            do sig=1,M
               call reverse_identification(N,M,sig,G,x_traceless_vec)
               call conversion(N,x_traceless_vec,G_vec,Gm)
               call multiplication(dim,N,M,zeta,X,x_traceless_vec,
     &                 y_traceless_vec)
               call conversion(N,y_traceless_vec,F_vec,Fm)
               T(1)=cmplx(0,0)
               T(2)=cmplx(0,0)
               T(3)=cmplx(0,0)
               T(4)=cmplx(0,0)
               S(1)=cmplx(0,0)
               S(2)=cmplx(0,0)
               S(3)=cmplx(0,0)
               S(4)=cmplx(0,0)
            do k=1,N
      T(1)=T(1)+Gm(1,j,k)*conjg(Fm(2,k,i))-conjg(Fm(2,j,k))*Gm(1,k,i)
     &            +Gm(2,j,k)*conjg(Fm(1,k,i))-conjg(Fm(1,j,k))*Gm(2,k,i)
      S(1)=S(1)+Gm(1,i,k)*conjg(Fm(2,k,j))-conjg(Fm(2,i,k))*Gm(1,k,j)
     &            +Gm(2,i,k)*conjg(Fm(1,k,j))-conjg(Fm(1,i,k))*Gm(2,k,j)
      T(2)=T(2)-Gm(1,j,k)*conjg(Fm(2,k,i))+conjg(Fm(2,j,k))*Gm(1,k,i)
     &            +Gm(2,j,k)*conjg(Fm(1,k,i))-conjg(Fm(1,j,k))*Gm(2,k,i)
      S(2)=S(2)-Gm(1,i,k)*conjg(Fm(2,k,j))+conjg(Fm(2,i,k))*Gm(1,k,j)
     &            +Gm(2,i,k)*conjg(Fm(1,k,j))-conjg(Fm(1,i,k))*Gm(2,k,j)
      T(3)=T(3)+Gm(1,j,k)*conjg(Fm(1,k,i))-conjg(Fm(1,j,k))*Gm(1,k,i)
     &            -Gm(2,j,k)*conjg(Fm(2,k,i))+conjg(Fm(2,j,k))*Gm(2,k,i)
      S(3)=S(3)+Gm(1,i,k)*conjg(Fm(1,k,j))-conjg(Fm(1,i,k))*Gm(1,k,j)
     &            -Gm(2,i,k)*conjg(Fm(2,k,j))+conjg(Fm(2,i,k))*Gm(2,k,j)
      T(4)=T(4)+Gm(1,j,k)*conjg(Fm(1,k,i))-conjg(Fm(1,j,k))*Gm(1,k,i)
     &            +Gm(2,j,k)*conjg(Fm(2,k,i))-conjg(Fm(2,j,k))*Gm(2,k,i)
      S(4)=S(4)+Gm(1,i,k)*conjg(Fm(1,k,j))-conjg(Fm(1,i,k))*Gm(1,k,j)
     &            +Gm(2,i,k)*conjg(Fm(2,k,j))-conjg(Fm(2,i,k))*Gm(2,k,j)
            enddo
            T(2)=ii*T(2)
            S(2)=ii*S(2)
            T(4)=ii*T(4)
            S(4)=ii*S(4)
            varF(1,i,j)=varF(1,i,j)-a(sig)*(T(1)+conjg(S(1)))
            varF(2,i,j)=varF(2,i,j)-a(sig)*(T(2)+conjg(S(2)))
            varF(3,i,j)=varF(3,i,j)-a(sig)*(T(3)+conjg(S(3)))
            varF(4,i,j)=varF(4,i,j)-a(sig)*(T(4)+conjg(S(4)))
         enddo
         varF(1,j,i)=conjg(varF(1,i,j))
         varF(2,j,i)=conjg(varF(2,i,j))
         varF(3,j,i)=conjg(varF(3,i,j))
         varF(4,j,i)=conjg(varF(4,i,j))
      enddo
      enddo

      return
      end

c............multiplication by M....

      subroutine multiplication(dim,N,M,zeta,X,x_traceless_vec
     &      ,y_traceless_vec)
      implicit none
      integer i,j,k,dim,N,M
      double precision zeta
      double complex y_mat(2,N,N),y_vec(2,N*N),y_traceless_vec(2,N*N-1),
     &      x_mat(2,N,N),x_vec(2,N*N),x_traceless_vec(2,N*N-1)
      double complex ii,X(dim,N,N)
```

```fortran
      ii=cmplx(0,1)
      call conversion(N,x_traceless_vec,x_vec,x_mat)
      do j=1,N
         do i=1,N
            y_mat(1,j,i)=zeta*x_mat(1,i,j)
            y_mat(2,j,i)=zeta*x_mat(2,i,j)
            do k=1,N
               y_mat(1,j,i)=y_mat(1,j,i)
     &                +X(3,i,k)*x_mat(1,k,j)-x_mat(1,i,k)*X(3,k,j)
     &                +ii*X(4,i,k)*x_mat(1,k,j)-ii*x_mat(1,i,k)*X(4,k,j)
     &                +X(1,i,k)*x_mat(2,k,j)-x_mat(2,i,k)*X(1,k,j)
     &                -ii*X(2,i,k)*x_mat(2,k,j)+ii*x_mat(2,i,k)*X(2,k,j)
               y_mat(2,j,i)=y_mat(2,j,i)
     &                -X(3,i,k)*x_mat(2,k,j)+x_mat(2,i,k)*X(3,k,j)
     &                +ii*X(4,i,k)*x_mat(2,k,j)-ii*x_mat(2,i,k)*X(4,k,j)
     &                +X(1,i,k)*x_mat(1,k,j)-x_mat(1,i,k)*X(1,k,j)
     &                +ii*X(2,i,k)*x_mat(1,k,j)-ii*x_mat(1,i,k)*X(2,k,j)
            enddo
         enddo
      enddo
      call reverse_conversion(N,y_mat,y_vec,y_traceless_vec)

      return
      end

c.............multiplication by M^+....

      subroutine multiplication_plus(dim,N,M,zeta,X,y_traceless_vec
     &      ,z_traceless_vec)
      implicit none
      integer i,j,k,dim,N,M
      double precision zeta
      double complex z_mat(2,N,N),z_vec(2,N*N),z_traceless_vec(2,N*N-1),
     &      y_mat(2,N,N),y_vec(2,N*N),y_traceless_vec(2,N*N-1)
      double complex ii,X(dim,N,N)

      ii=cmplx(0,1)
      call conversion(N,y_traceless_vec,y_vec,y_mat)
      do j=1,N
         do i=1,N
            z_mat(1,j,i)=zeta*y_mat(1,i,j)
            z_mat(2,j,i)=zeta*y_mat(2,i,j)
            do k=1,N
               z_mat(1,j,i)=z_mat(1,j,i)
     &                -X(3,k,i)*y_mat(1,k,j)+y_mat(1,i,k)*X(3,j,k)
     &                +ii*X(4,k,i)*y_mat(1,k,j)-ii*y_mat(1,i,k)*X(4,j,k)
     &                -X(1,k,i)*y_mat(2,k,j)+y_mat(2,i,k)*X(1,j,k)
     &                +ii*X(2,k,i)*y_mat(2,k,j)-ii*y_mat(2,i,k)*X(2,j,k)
               z_mat(2,j,i)=z_mat(2,j,i)
     &                +X(3,k,i)*y_mat(2,k,j)-y_mat(2,i,k)*X(3,j,k)
     &                +ii*X(4,k,i)*y_mat(2,k,j)-ii*y_mat(2,i,k)*X(4,j,k)
     &                -X(1,k,i)*y_mat(1,k,j)+y_mat(1,i,k)*X(1,j,k)
     &                -ii*X(2,k,i)*y_mat(1,k,j)+ii*y_mat(1,i,k)*X(2,j,k)
            enddo
         enddo
      enddo
      call reverse_conversion(N,z_mat,z_vec,z_traceless_vec)

      return
      end

c.... given x_traceless_vec we construct x_vec and x_mat........

      subroutine conversion(N,x_traceless_vec,x_vec,x_mat)
      implicit none
      integer N,i,j,A1,sp
```

```fortran
      double complex x_traceless_vec(2,N*N-1),x_vec(2,N*N),x_mat(2,N,N)
     &       ,xx

      do sp=1,2
         xx=0.0d0
         do i=1,N
            do j=1,N
               A1=N*(i-1)+j
               if (A1.lt.N*N) then
                  x_vec(sp,A1)=x_traceless_vec(sp,A1)
                  if (i.eq.j) then
                     xx=xx-x_traceless_vec(sp,A1)
                  endif
               endif
               x_mat(sp,i,j)=x_vec(sp,A1)
            enddo
         enddo
         x_vec(sp,N*N)=xx
         x_mat(sp,N,N)=x_vec(sp,N*N)
      enddo

      return
      end

c......given x_mat we construct x_vec and x_traceless_vec...

      subroutine reverse_conversion(N,x_mat,x_vec,x_traceless_vec)
      implicit none
      integer N,i,j,A1,sp
      double complex x_mat(2,N,N),x_vec(2,N*N),x_traceless_vec(2,N*N-1)

      do sp=1,2
         x_vec(sp,N*N)=x_mat(sp,N,N)
         do i=1,N
            do j=1,N
               A1=N*(i-1)+j
               if (A1.lt.N*N) then
                  x_vec(sp,A1)=x_mat(sp,i,j)
                  if (i.eq.j)then
                     x_traceless_vec(sp,A1)=x_vec(sp,A1)-x_vec(sp,N*N)
                  else
                     x_traceless_vec(sp,A1)=x_vec(sp,A1)
                  endif
               endif
            enddo
         enddo
      enddo

      return
      end

c...............generation of Gaussian noise for the field P............

      subroutine gaussian(idum,dim,N,P)
      implicit none
      integer dim,N,mu,i,j,idum
      double precision pi,phi,r,ran2
      double complex ii,P(dim,N,N)

      pi=dacos(-1.0d0)
      ii=cmplx(0,1)
      do mu=1,dim
c.............diagonal.........
         do i=1,N
            phi=2.0d0*pi*ran2(idum)
            r=dsqrt(-2.0d0*dlog(1.0d0-ran2(idum)))
            P(mu,i,i)=r*dcos(phi)
```

```fortran
            enddo
c.......off diagonal............
        do i=1,N
            do j=i+1,N
                phi=2.0d0*pi*ran2(idum)
                r=dsqrt(-1.0d0*dlog(1.0d0-ran2(idum)))
                P(mu,i,j)=r*dcos(phi)+ii*r*dsin(phi)
                P(mu,j,i)=conjg(P(mu,i,j))
            enddo
        enddo
    enddo

    return
    end

c...............generation of Gaussian noise for the field Q............

    subroutine gaussian_plus(idum,N,Q)
    implicit none
    integer N,A1,sp,idum
    double precision pi,phi,r,ran2
    double complex Q(2,N*N-1),ii

    pi=dacos(-1.0d0)
    ii=cmplx(0,1)
    do A1=1,N*N-1
        do sp=1,2
            phi=2.0d0*pi*ran2(idum)
            r=dsqrt(-1.0d0*dlog(1.0d0-ran2(idum)))
            Q(sp,A1)=r*dcos(phi)+ii*r*dsin(phi)
        enddo
    enddo

    return
    end

c.........hot start.................

    subroutine hot(N,dim,idum,inn,X)
    integer mu,i,j,N,dim,idum
    double complex X(dim,N,N)
    double precision xx,y,inn,ran2

    do mu=1,dim
        do i=1,N
            do j=i,N
                if (j.ne.i) then
                    xx=(2.0d0*ran2(idum)-1.0d0)*inn
                    y=(2.0d0*ran2(idum)-1.0d0)*inn
                    X(mu,i,j)=cmplx(xx,y)
                    X(mu,j,i)=cmplx(xx,-y)
                else
                    xx=(2.0d0*ran2(idum)-1.0d0)*inn
                    X(mu,i,j)=xx
                endif
            enddo
        enddo
    enddo

    return
    end

c...........cold start.....................

    subroutine cold(N,dim,idum,X)
    integer mu,i,j,N,dim,idum
    double complex X(dim,N,N)
```

```fortran
      do mu=1,dim
         do i=1,N
            do j=1,N
               X(mu,i,j)=cmplx(0,0)
            enddo
         enddo
      enddo

      return
      end

c..........the jackknife estimator...............

      subroutine jackknife_binning(TMC,f,average,error)
      integer i,j,TMC,zbin,nbin
      double precision xm
      double precision f(1:TMC),sumf,y(1:TMC)
      double precision sig0,sig,error,average

      sig0=0.0d0
      sumf=0.0d0
      do i=1,TMC
         sumf=sumf+f(i)
      enddo
      xm=sumf/TMC
      zbin=1
      nbin=int(TMC/zbin)
      sig=0.0d0
      do i=1,nbin,1
         y(i)=sumf
         do j=1,zbin
            y(i)=y(i)-f((i-1)*zbin+j )
         enddo
         y(i)= y(i)/(TMC-zbin)
         sig=sig+((nbin-1.0d0)/nbin)*(y(i)-xm)*(y(i)-xm)
      enddo
      sig=dsqrt(sig)
      if (sig0 .lt. sig) sig0=sig
      error=sig0
      average=xm

      return
      end

c.............the random number generator ran2.............

      function ran2(idum)
      implicit none
      integer idum,IM1,IM2,IMM1,IA1,IA2,IQ1,IQ2,IR1,IR2,NTAB,NDIV
      real AM,EPS,RNMX
      double precision ran2
      parameter (IM1=2147483563,IM2=2147483399,AM=1./IM1,IMM1=IM1-1,
     &     IA1=40014,IA2=40692,IQ1=53668,IQ2=52774,IR1=12211,
     &     IR2=3791,NTAB=32,NDIV=1+IMM1/NTAB,EPS=1.2E-7,RNMX=1.-EPS)
      integer idum2,j,k,iv(NTAB),iy
      SAVE iv,iy,idum2
      DATA idum2/123456789/,iv/NTAB*0/,iy/0/

      if (idum.le.0) then
         idum=max(-idum,1)
         idum2=idum
         do j=NTAB+8,1,-1
            k=idum/IQ1
            idum=IA1*(idum-k*IQ1)-k*IR1
            if (idum.lt.0) idum=idum+IM1
            if (j.le.NTAB) iv(j)=idum
```

```fortran
         enddo
         iy=iv(1)
      endif
      k=idum/IQ1
      idum=IA1*(idum-k*IQ1)-k*IR1
      if (idum.lt.0) idum=idum+IM1
      k=idum2/IQ2
      idum2=IA2*(idum2-k*IQ2)-k*IR2
      if (idum2.lt.0) idum2=idum2+IM2
      j=1+iy/NDIV
      iy=iv(j)-idum2
      iv(j)=idum
      if (iy.lt.1) iy=iy+IMM1
      ran2=min(AM*iy,RNMX)

      return
      end

c...........defining an array from a vector....

      subroutine identification(N,M,sig,x_traceless_vec,G)
      implicit none
      integer N,M,sig,sp,A1
      double complex G(M,2,N*N-1),x_traceless_vec(2,N*N-1)

      do sp=1,2
         do A1=1,N*N-1
            G(sig,sp,A1)=x_traceless_vec(sp,A1)
         enddo
      enddo

      return
      end

c.......defining a vector from an array.......

      subroutine reverse_identification(N,M,sig,G,x_traceless_vec)
      implicit none
      integer N,M,sig,sp,A1
      double complex G(M,2,N*N-1),x_traceless_vec(2,N*N-1)

      do sp=1,2
         do A1=1,N*N-1
            x_traceless_vec(sp,A1)=G(sig,sp,A1)
         enddo
      enddo

      return
      end

c.........adjusting interval..................

      subroutine adjust_inn(cou,pa,dt,time,Rejec,Accept,
     &     nn,target_pa_high,target_pa_low,dt_max,dt_min,inc,dec)
      implicit none
      double precision dt,pa,Rejec,Accept
      integer time,cou,cou1
      integer nn
      double precision target_pa_high,target_pa_low,dt_max,dt_min,inc,
     &     dec,rho1,rho2,dtnew

c.....pa acceptance rate............
      pa=(Accept)/(Rejec+Accept)
      cou1=mod(cou,nn)
      if (cou1.eq.0)then
c........fixing the acceptance rate between 90 % 70 %..................
         if (pa.ge.target_pa_high) then
```

```fortran
            dtnew=dt*inc
            if (dtnew.le.dt_max)then
               dt=dtnew
            else
               dt=dt_max
            endif
         endif
         if (pa.le.target_pa_low) then
            dtnew=dt*dec
            if (dtnew.ge.dt_min)then
               dt=dtnew
            else
               dt=dt_min
            endif
         endif
      endif

      return
      end
```

```fortran
      program my_u_one_on_the_lattice
      implicit none
      integer dim,N,NT,i,j,k,l,mu,idum,tther,tmont,nther,nmont,counter,T
      integer tcor,ncor,betai,p,q
      double precision accept,reject,flip
      parameter (dim=4,N=4,NT=4,nther=2**(14),nmont=2**(14),ncor=2**4)
      parameter (T=100*(nther+nmont*ncor))
      double precision beta,ran2,variation,epsilon
     &     ,epsilon0,pi,acceptance,avera,erro,tau,deltau
      double complex U(dim,N,N,N,NT),ii,X,XX(0:T)
      double precision W11,W22,W33,W12,W13,W23,W21,W31,W32
      double precision acti(1:nmont),acti_mean,acti_error,
     &     action
      double precision acti_pp(1:nmont),acti_pp_mean,acti_pp_error,
     &     action_pp
      double precision cv(1:nmont),cv_mean,cv_error
      double precision plaq1(1:nmont),plaq1_mean,plaq1_error
      double precision plaq2(1:nmont),plaq2_mean,plaq2_error
      double precision plaq3(1:nmont),plaq3_mean,plaq3_error
      double precision plaq4(1:nmont),plaq4_mean,plaq4_error
      double precision plaq5(1:nmont),plaq5_mean,plaq5_error
      double precision plaq6(1:nmont),plaq6_mean,plaq6_error
      double precision plaq7(1:nmont),plaq7_mean,plaq7_error
      double precision plaq8(1:nmont),plaq8_mean,plaq8_error
      double precision plaq9(1:nmont),plaq9_mean,plaq9_error
      double precision tension1,error_tension1,tension2,error_tension2,
     &     tension3,error_tension3,tension4,error_tension4

c.........initialization of the random number generator....................

      idum=-148175
      call seed(idum)

c..........initialization of other parameters.............................

      counter=0
      accept=0
      reject=0
      flip=0
      ii=cmplx(0,1)
      pi=dacos(-1.0d0)
      epsilon=pi

c................gauge coupling constant..................

      do betai=1,1
         beta=1.9d0-betai*0.1

c............initialization of the link variables...............

      do mu=1,dim
         do i=1,N
            do j=1,N
               do k=1,N
                  do l=1,NT
c.........ordered start for coulomb phase while disordered start for confinment phase..
                     if(beta.ge.1.0d0)then
                        epsilon0=0.0d0
                     else
                        epsilon0=2.0d0*ran2(idum)-1.0d0
                        epsilon0=epsilon*epsilon0
                     endif
                     U(mu,i,j,k,l)=dcos(epsilon0)+ii*dsin(epsilon0)
                  enddo
               enddo
            enddo
         enddo
      enddo
```

```fortran
        enddo

C.................thermalization.............

        do tther=1,nther
         call metropolis(U,beta,dim,N,NT,accept,reject,flip,acceptance,
     &        epsilon,counter,XX,T)
        enddo

C..............monte carlo evolution..................................

        do tmont=1,nmont
           do tcor=1,ncor
         call metropolis(U,beta,dim,N,NT,accept,reject,flip,acceptance,
     &            epsilon,counter,XX,T)
           enddo
           call actio(U,dim,N,NT,beta,action,action_pp)
           acti(tmont)=action
           acti_pp(tmont)=action_pp
           plaq1(tmont)=0.0d0
           plaq2(tmont)=0.0d0
           plaq3(tmont)=0.0d0
           plaq4(tmont)=0.0d0
           plaq5(tmont)=0.0d0
           plaq6(tmont)=0.0d0
           plaq7(tmont)=0.0d0
           plaq8(tmont)=0.0d0
           plaq9(tmont)=0.0d0
           do i=1,N
              do j=1,N
                 do k=1,N
                    do l=1,NT
                       p=1
                       q=4
                       call Wilson_Loop(U,dim,N,NT,i,j,k,l,p,q,
     &                    W11,W22,W33,W12,W13,W23,W21,W31,W32)
                       plaq1(tmont)=plaq1(tmont)+W11
                       plaq2(tmont)=plaq2(tmont)+W22
                       plaq3(tmont)=plaq3(tmont)+W33
                       plaq4(tmont)=plaq4(tmont)+W12
                       plaq5(tmont)=plaq5(tmont)+W13
                       plaq6(tmont)=plaq6(tmont)+W23
                       plaq7(tmont)=plaq7(tmont)+W21
                       plaq8(tmont)=plaq8(tmont)+W31
                       plaq9(tmont)=plaq9(tmont)+W32
                    enddo
                 enddo
              enddo
           enddo
           plaq1(tmont)=plaq1(tmont)/(N**3*NT)
           plaq2(tmont)=plaq2(tmont)/(N**3*NT)
           plaq3(tmont)=plaq3(tmont)/(N**3*NT)
           plaq4(tmont)=plaq4(tmont)/(N**3*NT)
           plaq5(tmont)=plaq5(tmont)/(N**3*NT)
           plaq6(tmont)=plaq6(tmont)/(N**3*NT)
           plaq7(tmont)=plaq7(tmont)/(N**3*NT)
           plaq8(tmont)=plaq8(tmont)/(N**3*NT)
           plaq9(tmont)=plaq9(tmont)/(N**3*NT)
        enddo

C.....................measurements........................

C.....................action..............
        call jackknife_binning(nmont,acti,acti_mean,acti_error)
        write(11,*)beta,acti_mean,acti_error
c       write(*,*)beta,acti_mean,acti_error
```

```fortran
c..................action per plaquette..........
        call jackknife_binning(nmont,acti_pp,acti_pp_mean,acti_pp_error)
        write(12,*)beta,acti_pp_mean,acti_pp_error
c       write(*,*)beta,acti_pp_mean,acti_pp_error

c......................specific heat.............
        do tmont=1,nmont
          cv(tmont)=(acti(tmont)-acti_mean)**2
        enddo
        call jackknife_binning(nmont,cv,cv_mean,cv_error)
        write(13,*)beta,cv_mean,cv_error
c       write(*,*)beta,cv_mean,cv_error

c................Wilson loops................
        call jackknife_binning(nmont,plaq1,plaq1_mean,plaq1_error)
        write(15,*)beta,plaq1_mean,plaq1_error
c       write(*,*)beta,plaq1_mean,plaq1_error
        call jackknife_binning(nmont,plaq2,plaq2_mean,plaq2_error)
        write(16,*)beta,plaq2_mean,plaq2_error
c       write(*,*)beta,plaq2_mean,plaq2_error
        call jackknife_binning(nmont,plaq3,plaq3_mean,plaq3_error)
        write(17,*)beta,plaq3_mean,plaq3_error
c       write(*,*)beta,plaq3_mean,plaq3_error
        call jackknife_binning(nmont,plaq4,plaq4_mean,plaq4_error)
        write(18,*)beta,plaq4_mean,plaq4_error
c       write(*,*)beta,plaq4_mean,plaq4_error
        call jackknife_binning(nmont,plaq5,plaq5_mean,plaq5_error)
        write(19,*)beta,plaq5_mean,plaq5_error
c       write(*,*)beta,plaq5_mean,plaq5_error
        call jackknife_binning(nmont,plaq6,plaq6_mean,plaq6_error)
        write(20,*)beta,plaq6_mean,plaq6_error
c       write(*,*)beta,plaq6_mean,plaq6_error
        call jackknife_binning(nmont,plaq7,plaq7_mean,plaq7_error)
        write(23,*)beta,plaq7_mean,plaq7_error
c       write(*,*)beta,plaq7_mean,plaq7_error
        call jackknife_binning(nmont,plaq8,plaq8_mean,plaq8_error)
        write(24,*)beta,plaq8_mean,plaq8_error
c       write(*,*)beta,plaq8_mean,plaq8_error
        call jackknife_binning(nmont,plaq9,plaq9_mean,plaq9_error)
        write(25,*)beta,plaq9_mean,plaq9_error
c       write(*,*)beta,plaq9_mean,plaq9_error

c..............Creutz ratios:string tension.............
c...........chi22..........
        tension1=(plaq2_mean*plaq1_mean)/(plaq4_mean*plaq7_mean)
c..........chi33.....
        tension2=(plaq3_mean*plaq2_mean)/(plaq6_mean*plaq9_mean)
c..........chi23......
        tension3=(plaq6_mean*plaq4_mean)/(plaq2_mean*plaq5_mean)
c.........chi32..........
        tension4=(plaq9_mean*plaq7_mean)/(plaq2_mean*plaq8_mean)

        tension1=dabs(tension1)
        tension2=dabs(tension2)
        tension3=dabs(tension3)
        tension4=dabs(tension4)
        tension1=-dlog(tension1)
        tension2=-dlog(tension2)
        tension3=-dlog(tension3)
        tension4=-dlog(tension4)
        error_tension1=plaq2_error/plaq2_mean+plaq1_error/plaq1_mean
     &        -plaq4_error/plaq4_mean-plaq7_error/plaq7_mean
        error_tension1=dabs(error_tension1)
        error_tension2=plaq3_error/plaq3_mean+plaq2_error/plaq2_mean
     &        -plaq6_error/plaq6_mean -plaq9_error/plaq9_mean
        error_tension2=dabs(error_tension2)
        error_tension3=plaq6_error/plaq6_mean+plaq4_error/plaq4_mean
```

```fortran
     &          -plaq2_error/plaq2_mean -plaq5_error/plaq5_mean
          error_tension3=dabs(error_tension3)
          error_tension4=plaq9_error/plaq9_mean+plaq7_error/plaq7_mean
     &          -plaq2_error/plaq2_mean -plaq8_error/plaq8_mean
          error_tension4=dabs(error_tension4)

          write(22,*)beta,tension1,error_tension1,tension2,error_tension2,
     &          tension3,error_tension3,tension4,error_tension4
c        write(*,*)beta,tension1,error_tension1,tension2,error_tension2,
c     &          tension3,error_tension3,tension4,error_tension4

       enddo

       return
       end

c...............metropolis algorithm.................

       subroutine metropolis(U,beta,dim,N,NT,accept,reject,flip,
     &      acceptance,epsilon,counter,XX,T)
       implicit none
       integer dim,N,NT,nu,mu,i,j,k,l,idum,counter,counter0,nn,T
       double precision accept,reject,flip,nn0
       double precision epsilon,epsilon0,beta,variation,proba,r,ran2,pi,
     &      modulus,acceptance
       double complex U(dim,N,N,N,NT),X,ii,XX(0:T)

       pi=dacos(-1.0d0)
       ii=cmplx(0,1)

       epsilon0=2.0d0*ran2(idum)-1.0d0
       epsilon0=epsilon*epsilon0
       XX(counter)=dcos(epsilon0)+ii*dsin(epsilon0)
       XX(counter+1)=dcos(epsilon0)-ii*dsin(epsilon0)
       counter0=counter+1
       counter=counter+2

       do mu=1,dim
          do i=1,N
             do j=1,N
                do k=1,N
                   do l=1,NT
                      nn0=counter0*ran2(idum)
                      nn=nint(nn0)
                      X=XX(nn)
                    call variatio(U,X,beta,dim,N,NT,mu,i,j,k,l,variation)
                      if(variation.gt.0)then
                         proba=dexp(-variation)
                         r=ran2(idum)
                         if(proba.gt.r)then
                            U(mu,i,j,k,l)=X*U(mu,i,j,k,l)
                            accept=accept+1
                         else
                            reject=reject+1
                         endif
                      else
                         U(mu,i,j,k,l)=X*U(mu,i,j,k,l)
                         flip=flip+1
                      endif
                      modulus=U(mu,i,j,k,l)*conjg(U(mu,i,j,k,l))
                      modulus=dsqrt(modulus)
                      U(mu,i,j,k,l)=U(mu,i,j,k,l)/modulus
                   enddo
                enddo
             enddo
          enddo
       enddo
```

```fortran
c.......for the range of N and NT considered the acceptance rate is already sufficiently high so we can
simply disable the adjust subroutine....we observed that the acceptance rate decreases as we increase N
and NT......
      call adjust(epsilon,flip,accept,reject,acceptance)
c     write(*,*)flip,accept,reject,acceptance

      return
      end

c...........adjusting............................

      subroutine adjust(epsilon,flip,accept,reject,acceptance)
      implicit none
      double precision epsilon,acceptance
      double precision flip,accept,reject,ran2
      integer idum

      acceptance=(flip+accept)/(flip+accept+reject)
      if (acceptance.ge.0.5d0) then
         epsilon=epsilon*1.2d0
      endif
      if(acceptance.le.0.45d0) then
         epsilon=epsilon*0.8d0
      endif

      return
      end

c.........................variation.....................

      subroutine variatio(U,X,beta,dim,N,NT,mu,i,j,k,l,variation)
      implicit none
      integer dim,N,NT,nu,mu,i,j,k,l,idum
      double precision epsilon,epsilon0,beta,variation,ran2,pi
      double complex U(dim,N,N,N,NT),staple,ii,X

      call  stapl(U,dim,N,NT,mu,i,j,k,l,staple)
      variation=-0.5d0*beta*((X-1.0d0)*U(mu,i,j,k,l)*staple
     &      + conjg((X-1.0d0)*U(mu,i,j,k,l)*staple))

      return
      end

c.................staple...................................

      subroutine stapl(U,dim,N,NT,mu,i,j,k,l,staple)
      implicit none
      integer dim,N,NT,nu,mu,i,j,k,l,i0,ip(N),im(N),ipT(NT),imT(NT),
     &     ipn(1:N,1:N),ipnT(1:N,1:N)
      double precision beta
      double complex U(dim,N,N,N,NT),staple

      call index_array(N,NT,ip,im,ipT,imT,ipn,ipnT)

      if(mu.eq.1)then
         staple=U(2,ip(i),j,k,l)*conjg(U(mu,i,ip(j),k,l))*
     &        conjg(U(2,i,j,k,l))
     &        +conjg(U(2,ip(i),im(j),k,l))*conjg(U(mu,i,im(j),k,l))
     &        *U(2,i,im(j),k,l)
     &    +U(3,ip(i),j,k,l)*conjg(U(mu,i,j,ip(k),l))*conjg(U(3,i,j,k,l))
     &        +conjg(U(3,ip(i),j,im(k),l))*conjg(U(mu,i,j,im(k),l))
     &        *U(3,i,j,im(k),l)
     &    +U(4,ip(i),j,k,l)*conjg(U(mu,i,j,k,ipT(l)))*conjg(U(4,i,j,k,l))
     &        +conjg(U(4,ip(i),j,k,imT(l)))*conjg(U(mu,i,j,k,imT(l)))
     &        *U(4,i,j,k,imT(l))
      endif
```

```fortran
      if(mu.eq.2)then
         staple=U(1,i,ip(j),k,l)*conjg(U(mu,ip(i),j,k,l))*
     &         conjg(U(1,i,j,k,l))
     &         +conjg(U(1,im(i),ip(j),k,l))*conjg(U(mu,im(i),j,k,l))
     &         *U(1,im(i),j,k,l)
     &     +U(3,i,ip(j),k,l)*conjg(U(mu,i,j,ip(k),l))*conjg(U(3,i,j,k,l))
     &         +conjg(U(3,i,ip(j),im(k),l))*conjg(U(mu,i,j,im(k),l))
     &         *U(3,i,j,im(k),l)
     &     +U(4,i,ip(j),k,l)*conjg(U(mu,i,j,k,ipT(l)))*conjg(U(4,i,j,k,l))
     &         +conjg(U(4,i,ip(j),k,imT(l)))*conjg(U(mu,i,j,k,imT(l)))
     &         *U(4,i,j,k,imT(l))
      endif

      if(mu.eq.3)then
         staple=U(1,i,j,ip(k),l)*conjg(U(mu,ip(i),j,k,l))
     &         *conjg(U(1,i,j,k,l))
     &         +conjg(U(1,im(i),j,ip(k),l))*conjg(U(mu,im(i),j,k,l))
     &         *U(1,im(i),j,k,l)
     &     +U(2,i,j,ip(k),l)*conjg(U(mu,i,ip(j),k,l))*conjg(U(2,i,j,k,l))
     &         +conjg(U(2,i,im(j),ip(k),l))*conjg(U(mu,i,im(j),k,l))
     &         *U(2,i,im(j),k,l)
     &     +U(4,i,j,ip(k),l)*conjg(U(mu,i,j,k,ipT(l)))*conjg(U(4,i,j,k,l))
     &         +conjg(U(4,i,j,ip(k),imT(l)))*conjg(U(mu,i,j,k,imT(l)))
     &         *U(4,i,j,k,imT(l))
      endif

      if(mu.eq.4)then
         staple=U(1,i,j,k,ipT(l))*conjg(U(mu,ip(i),j,k,l))
     &         *conjg(U(1,i,j,k,l))
     &         +conjg(U(1,im(i),j,k,ipT(l)))*conjg(U(mu,im(i),j,k,l))
     &         *U(1,im(i),j,k,l)
     &     +U(2,i,j,k,ipT(l))*conjg(U(mu,i,ip(j),k,l))*conjg(U(2,i,j,k,l))
     &         +conjg(U(2,i,im(j),k,ipT(l)))*conjg(U(mu,i,im(j),k,l))
     &         *U(2,i,im(j),k,l)
     &     +U(3,i,j,k,ipT(l))*conjg(U(mu,i,j,ip(k),l))*conjg(U(3,i,j,k,l))
     &         +conjg(U(3,i,j,im(k),ipT(l)))*conjg(U(mu,i,j,im(k),l))
     &         *U(3,i,j,im(k),l)
      endif

      return
      end

c...............wilson loops.............................

      subroutine Wilson_Loop(U,dim,N,NT,i,j,k,l,p,q,
     &     W11,W22,W33,W12,W13,W23,W21,W31,W32)
      implicit none
      integer dim,N,NT,i,j,k,l,p,q,i0,j0,ipn(1:N,1:N),ipnT(1:N,1:N),
     &     ip(1:N),im(1:N),ipT(1:N),imT(1:N)
      double complex U(dim,N,N,N,NT),W1,W2,W3,W4
      double precision W11,W22,W33,W12,W13,W23,W21,W31,W32

      call index_array(N,NT,ip,im,ipT,imT,ipn,ipnT)
      if ((p.eq.1).and.(q.eq.4))then

      W1=U(p,i,j,k,l)
      W4=U(q,i,j,k,l)
c        W3=U(q,i+1,j,k,l)
      W3=U(q,ipn(i,1),j,k,l)
c        W2=U(p,i,j,k,l+1)
      W2=U(p,i,j,k,ipnT(l,1))
      W11=0.5d0*(W1*W3*conjg(W2)*conjg(W4)+
     &         conjg(W1)*conjg(W3)*W2*W4)

c        W1=U(p,i,j,k,l)*U(p,i+1,j,k,l)
      W1=U(p,i,j,k,l)*U(p,ipn(i,1),j,k,l)
c        W4=U(q,i,j,k,l)*U(q,i,j,k,l+1)
```

```fortran
            W4=U(q,i,j,k,l)*U(q,i,j,k,ipnT(l,1))
c           W3=U(q,i+2,j,k,l)*U(q,i+2,j,k,l+1)
            W3=U(q,ipn(i,2),j,k,l)*U(q,ipn(i,2),j,k,ipnT(l,1))
c           W2=U(p,i,j,k,l+2)*U(p,i+1,j,k,l+2)
            W2=U(p,i,j,k,ipnT(l,2))*U(p,ipn(i,1),j,k,ipnT(l,2))
            W22=0.5d0*(W1*W3*conjg(W2)*conjg(W4)+
     &           conjg(W1)*conjg(W3)*W2*W4)

c           W1=U(p,i,j,k,l)*U(p,i+1,j,k,l)*U(p,i+2,j,k,l)
            W1=U(p,i,j,k,l)*U(p,ipn(i,1),j,k,l)*U(p,ipn(i,2),j,k,l)
c           W4=U(q,i,j,k,l)*U(q,i,j,k,l+1)*U(q,i,j,k,l+2)
            W4=U(q,i,j,k,l)*U(q,i,j,k,ipnT(l,1))*U(q,i,j,k,ipnT(l,2))
c           W3=U(q,i+3,j,k,l)*U(q,i+3,j,k,l+1)*U(q,i+3,j,k,l+2)
            W3=U(q,ipn(i,3),j,k,l)*U(q,ipn(i,3),j,k,ipnT(l,1))*
     &           U(q,ipn(i,3),j,k,ipnT(l,2))
c           W2=U(p,i,j,k,l+3)*U(p,i+1,j,k,l+3)*U(p,i+2,j,k,l+3)
            W2=U(p,i,j,k,ipnT(l,3))*U(p,ipn(i,1),j,k,ipnT(l,3))*
     &           U(p,ipn(i,2),j,k,ipnT(l,3))
            W33=0.5d0*(W1*W3*conjg(W2)*conjg(W4)+
     &           conjg(W1)*conjg(W3)*W2*W4)


            W1=U(p,i,j,k,l)
c           W4=U(q,i,j,k,l)*U(q,i,j,k,l+1)
            W4=U(q,i,j,k,l)*U(q,i,j,k,ipnT(l,1))
c           W3=U(q,i+1,j,k,l)*U(q,i+1,j,k,l+1)
            W3=U(q,ipn(i,1),j,k,l)*U(q,ipn(i,1),j,k,ipnT(l,1))
c           W2=U(p,i,j,k,l+2)
            W2=U(p,i,j,k,ipnT(l,2))
            W12=0.5d0*(W1*W3*conjg(W2)*conjg(W4)+
     &           conjg(W1)*conjg(W3)*W2*W4)

c           W1=U(p,i,j,k,l)*U(p,i+1,j,k,l)
            W1=U(p,i,j,k,l)*U(p,ipn(i,1),j,k,l)
            W4=U(q,i,j,k,l)
c           W3=U(q,i+2,j,k,l)
            W3=U(q,ipn(i,2),j,k,l)
c           W2=U(p,i,j,k,l+1)*U(p,i+1,j,k,l+1)
            W2=U(p,i,j,k,ipnT(l,1))*U(p,ipn(i,1),j,k,ipnT(l,1))
            W21=0.5d0*(W1*W3*conjg(W2)*conjg(W4)+
     &           conjg(W1)*conjg(W3)*W2*W4)


            W1=U(p,i,j,k,l)
c           W4=U(q,i,j,k,l)*U(q,i,j,k,l+1)*U(q,i,j,k,l+2)
            W4=U(q,i,j,k,l)*U(q,i,j,k,ipnT(l,1))*U(q,i,j,k,ipnT(l,2))
c           W3=U(q,i+1,j,k,l)*U(q,i+1,j,k,l+1)*U(q,i+1,j,k,l+2)
            W3=U(q,ipn(i,1),j,k,l)*U(q,ipn(i,1),j,k,ipnT(l,1))*
     &           U(q,ipn(i,1),j,k,ipnT(l,2))
c           W2=U(p,i,j,k,l+2)
            W2=U(p,i,j,k,ipnT(l,3))
            W13=0.5d0*(W1*W3*conjg(W2)*conjg(W4)+
     &           conjg(W1)*conjg(W3)*W2*W4)


c           W1=U(p,i,j,k,l)*U(p,i+1,j,k,l)*U(p,i+2,j,k,l)
            W1=U(p,i,j,k,l)*U(p,ipn(i,1),j,k,l)*U(p,ipn(i,2),j,k,l)
            W4=U(q,i,j,k,l)
c           W3=U(q,i+3,j,k,l)
            W3=U(q,ipn(i,3),j,k,l)
c           W2=U(p,i,j,k,l+1)*U(p,i+1,j,k,l+1)*U(p,i+2,j,k,l+1)
            W2=U(p,i,j,k,ipnT(l,1))*U(p,ipn(i,1),j,k,ipnT(l,1))*
     &           U(p,ipn(i,2),j,k,ipnT(l,1))
            W31=0.5d0*(W1*W3*conjg(W2)*conjg(W4)+
     &           conjg(W1)*conjg(W3)*W2*W4)


c           W1=U(p,i,j,k,l)*U(p,i+1,j,k,l)
            W1=U(p,i,j,k,l)*U(p,ipn(i,1),j,k,l)
c           W4=U(q,i,j,k,l)*U(q,i,j,k,l+1)*U(q,i,j,k,l+2)
            W4=U(q,i,j,k,l)*U(q,i,j,k,ipnT(l,1))*U(q,i,j,k,ipnT(l,2))
```

```fortran
c        W3=U(q,i+2,j,k,l)*U(q,i+2,j,k,l+1)*U(q,i+2,j,k,l+2)
         W3=U(q,ipn(i,2),j,k,l)*U(q,ipn(i,2),j,k,ipnT(l,1))*
     &       U(q,ipn(i,2),j,k,ipnT(l,2))
c        W2=U(p,i,j,k,l+3)*U(p,i+1,j,k,l+3)
         W2=U(p,i,j,k,ipnT(l,3))*U(p,ipn(i,1),j,k,ipnT(l,3))
         W23=0.5d0*(W1*W3*conjg(W2)*conjg(W4)+
     &       conjg(W1)*conjg(W3)*W2*W4)

c        W1=U(p,i,j,k,l)*U(p,i+1,j,k,l)*U(p,i+2,j,k,l)
         W1=U(p,i,j,k,l)*U(p,ipn(i,1),j,k,l)*U(p,ipn(i,2),j,k,l)
c        W4=U(q,i,j,k,l)*U(q,i,j,k,l+1)
         W4=U(q,i,j,k,l)*U(q,i,j,k,ipnT(l,1))
c        W3=U(q,i+3,j,k,l)*U(q,i+3,j,k,l+1)
         W3=U(q,ipn(i,3),j,k,l)*U(q,ipn(i,3),j,k,ipnT(l,1))
c        W2=U(p,i,j,k,l+2)*U(p,i+1,j,k,l+2)*U(p,i+2,j,k,l+2)
         W2=U(p,i,j,k,ipnT(l,2))*U(p,ipn(i,1),j,k,ipnT(l,2))*
     &       U(p,ipn(i,2),j,k,ipnT(l,2))
         W32=0.5d0*(W1*W3*conjg(W2)*conjg(W4)+
     &       conjg(W1)*conjg(W3)*W2*W4)
      endif

      return
      end

c.........................indexing............................

      subroutine index_array(N,NT,ip,im,ipT,imT,ipn,ipnT)
      implicit none
      integer N,NT,i0,j0,ip(1:N),im(1:N),ipT(1:N),imT(1:N),
     &     ipn(1:N,1:N),ipnT(1:N,1:N)

      do i0=1,N
         ip(i0)=i0+1
         im(i0)=i0-1
      enddo
      ip(N)=1
      im(1)=N
      do i0=1,NT
         ipT(i0)=i0+1
         imT(i0)=i0-1
      enddo
      ipT(NT)=1
      imT(1)=NT
      do i0=1,N
         do j0=1,N
            if (i0+j0 .le. N) then
               ipn(i0,j0)=i0+j0
            else
               ipn(i0,j0)=(i0+j0)-N
            endif
         enddo
      enddo
      do i0=1,NT
         do j0=1,NT
            if (i0+j0 .le. NT) then
               ipnT(i0,j0)=i0+j0
            else
               ipnT(i0,j0)=(i0+j0)-NT
            endif
         enddo
      enddo

      return
      end

c.....................action............................
```

```fortran
      subroutine actio(U,dim,N,NT,beta,action,action_pp)
      implicit none
      integer dim,N,NT,i,j,k,l,ip(N),im(N),ipT(NT),imT(NT)
      double precision beta
      double precision action12,action13,action14,action23,action24,
     &     action34,action,action_pp
      double complex U(dim,N,N,N,NT)

      do i=1,N
         ip(i)=i+1
         im(i)=i-1
      enddo
      ip(N)=1
      im(1)=N
      do i=1,NT
         ipT(i)=i+1
         imT(i)=i-1
      enddo
      ipT(NT)=1
      imT(1)=NT

      i=1
      j=1
      k=1
      l=1
C.....................action per plaquette....
      action_pp=U(1,i,j,k,l)*U(2,ip(i),j,k,l)
     &     *conjg(U(1,i,ip(j),k,l))*conjg(U(2,i,j,k,l))
     &     +U(2,i,j,k,l)*U(1,i,ip(j),k,l)
     &     *conjg(U(2,ip(i),j,k,l))*conjg(U(1,i,j,k,l))
      action_pp=0.5d0*action_pp
      action_pp=1.0d0-action_pp
C...................action..........
      action12=0.0d0
      action13=0.0d0
      action14=0.0d0
      action23=0.0d0
      action24=0.0d0
      action34=0.0d0
      do i=1,N
        do j=1,N
          do k=1,N
            do l=1,NT
              action12=action12+U(1,i,j,k,l)*U(2,ip(i),j,k,l)
     &            *conjg(U(1,i,ip(j),k,l))*conjg(U(2,i,j,k,l))
     &            +U(2,i,j,k,l)*U(1,i,ip(j),k,l)
     &            *conjg(U(2,ip(i),j,k,l))*conjg(U(1,i,j,k,l))
              action13=action13+U(1,i,j,k,l)*U(3,ip(i),j,k,l)
     &            *conjg(U(1,i,j,ip(k),l))*conjg(U(3,i,j,k,l))
     &            +U(3,i,j,k,l)*U(1,i,j,ip(k),l)
     &            *conjg(U(3,ip(i),j,k,l))*conjg(U(1,i,j,k,l))
              action14=action14+U(1,i,j,k,l)*U(4,ip(i),j,k,l)
     &            *conjg(U(1,i,j,k,ipT(l)))*conjg(U(4,i,j,k,l))
     &            +U(4,i,j,k,l)*U(1,i,j,k,ipT(l))
     &            *conjg(U(4,ip(i),j,k,l))*conjg(U(1,i,j,k,l))
              action23=action23+U(2,i,j,k,l)*U(3,i,ip(j),k,l)
     &            *conjg(U(2,i,j,ip(k),l))*conjg(U(3,i,j,k,l))
     &            +U(3,i,j,k,l)*U(2,i,j,ip(k),l)
     &            *conjg(U(3,i,ip(j),k,l))*conjg(U(2,i,j,k,l))
              action24=action24+U(2,i,j,k,l)*U(4,i,ip(j),k,l)
     &            *conjg(U(2,i,j,k,ipT(l)))*conjg(U(4,i,j,k,l))
     &            +U(4,i,j,k,l)*U(2,i,j,k,ipT(l))
     &            *conjg(U(4,i,ip(j),k,l))*conjg(U(2,i,j,k,l))
              action34=action34+U(3,i,j,k,l)*U(4,i,j,ip(k),l)
     &            *conjg(U(3,i,j,k,ipT(l)))*conjg(U(4,i,j,k,l))
     &            +U(4,i,j,k,l)*U(3,i,j,k,ipT(l))
     &            *conjg(U(4,i,j,ip(k),l))*conjg(U(3,i,j,k,l))
```

```fortran
                enddo
            enddo
        enddo
    enddo
    action=action12+action13+action14+action23+action24+action34
    action=-0.5d0*beta*action
    action=action!+6.0d0*beta*N*N*N*NT

    return
    end

c........................jackknife.......................................

    subroutine jackknife_binning(TMC,f,average,error)
    implicit none
    integer i,j,TMC,zbin,nbin
    doubleprecision xm
    doubleprecision f(1:TMC),sumf,y(1:TMC)
    doubleprecision sig0,sig,error,average

    sig0=0.0d0
    sumf=0.0d0
    do i=1,TMC
        sumf=sumf+f(i)
    enddo
    xm=sumf/TMC
    zbin=1
    nbin=int(TMC/zbin)
    sig=0.0d0
    do i=1,nbin,1
        y(i)=sumf
        do j=1,zbin
            y(i)=y(i)-f((i-1)*zbin+j )
        enddo
        y(i)= y(i)/(TMC-zbin)
        sig=sig+((nbin-1.0d0)/nbin)*(y(i)-xm)*(y(i)-xm)
    enddo
    sig=dsqrt(sig)
    if (sig0 .lt. sig) sig0=sig
    error=sig0
    average=xm

    return
    end

c...............seed...................

    subroutine seed(idum)
    integer idum1,idum, n
    real x

    x=0.0
    idum=idum-2*int(secnds(x))

    return
    end

c.........the ran2 generator.................

    function ran2(idum)
    implicit none
    integer idum,IM1,IM2,IMM1,IA1,IA2,IQ1,IQ2,IR1,IR2,NTAB,NDIV
    real AM,EPS,RNMX
    doubleprecision ran2
    parameter (IM1=2147483563,IM2=2147483399,AM=1./IM1,IMM1=IM1-1,
   &    IA1=40014,IA2=40692,IQ1=53668,IQ2=52774,IR1=12211,
   &    IR2=3791,NTAB=32,NDIV=1+IMM1/NTAB,EPS=1.2E-7,RNMX=1.-EPS)
```

```fortran
      integer idum2,j,k,iv(NTAB),iy
      SAVE iv,iy,idum2
      DATA idum2/123456789/,iv/NTAB*0/,iy/0/

      if (idum.le.0) then
          idum=max(-idum,1)
          idum2=idum
          do j=NTAB+8,1,-1
              k=idum/IQ1
              idum=IA1*(idum-k*IQ1)-k*IR1
              if (idum.lt.0) idum=idum+IM1
              if (j.le.NTAB) iv(j)=idum
          enddo
          iy=iv(1)
      endif
      k=idum/IQ1
      idum=IA1*(idum-k*IQ1)-k*IR1
      if (idum.lt.0) idum=idum+IM1
      k=idum2/IQ2
      idum2=IA2*(idum2-k*IQ2)-k*IR2
      if (idum2.lt.0) idum2=idum2+IM2
      j=1+iy/NDIV
      iy=iv(j)-idum2
      iv(j)=idum
      if (iy.lt.1) iy=iy+IMM1
      ran2=min(AM*iy,RNMX)

      return
      end
```

Appendix A

Floating Point Representation, Machine Precision and Errors

Floating Point Representation: Any real number x can be put in the following binary form

$$x = \pm m \times 2^{e-\text{bias}} \ , \ 1{\leq}m < 2 \ , \ m = b_0.b_1b_2b_3... \tag{A.1}$$

We consider a $32-$bit computer. Since $1{\leq}m < 2$ we must have $b_0 = 1$. This binary expansion is called *normalized*. For single precision floating-point numbers (singles or floats) we use a $32-$bit word with one bit for the *sign*, 8 bits for the *exponent e* and 23 bits for the *significand m*. Since only 8 bits are used to store the exponent we must have e in the range $0{\leq}e{\leq}255$. The bias is chosen bias $= 127$ so that the actual exponent is in the range $-127{\leq}e - \text{bias}{\leq}128$. This way we can have very small numbers while the stored exponent is always positive. Since the first bit of the significand is 1 the stored bits of the significand are only $b_1b_2...b_{23}$. If $b_{24}, b_{25}, ..$ are not all zero the floating point representation is not exact. Strictly speaking a floating point number is a number for which $b_{24} = b_{25} = ..0$. The floating point representation of a non-zero real number is unique because of the condition $1{\leq}m < 2$. In summary the above real number is represented on the computer by

$$x_{\text{normal float}} = (-1)^s 1.f \times 2^{e-127} \ , \ 0 < e < 255. \tag{A.2}$$

These are normal numbers. The terminology floating point is now clear. The binary point can be moved (floated) to any position in the bitstring by choosing the appropriate exponent.

The smallest normalized number is 2^{-126}. The subnormal numbers are represented by

$$x_{\text{subnormal float}} = (-1)^s 0.f \times 2^{-126}. \tag{A.3}$$

These are not normalized numbers. In fact the space between 0 and the smallest positive normalized number is filled by the subnormal numbers.

Explicitly

	s	e	f
Bit Position	31	30-23	22-0

Because only a finite number of bits is used the set of *machine numbers* (the numbers that the computer can store exactly or approximately) is much smaller than the set of real numbers. There is a maximum and a minimum. Exceeding the maximum we get the error condition known as overflow. Falling below the minimum we get the error condition known as underflow.

The largest number corresponds to the normal floating number with $s = 0$, $e = 254$ and $1.f = 1.111..1$ (with 23 1s after the binary point). We compute $1.f = 1 + 0.5 + 0.25 + 0.125 + ... = 2$. Hence $x_{\text{normal float max}} = 2 \times 2^{127} \simeq 3.4 \times 10^{38}$. The smallest number corresponds to the subnormal floating number with $s = 0$ and $0.f = 0.00...1 = 2^{-23}$. Hence $x_{\text{subnormal float min}} = 2^{-149} \simeq 1.4 \times 10^{-45}$. We get for single precision floats the range

$$1.4 \times 10^{-45} \leq \text{single precision} \leq 3.4 \times 10^{38}. \tag{A.4}$$

We remark that

$$2^{-23} \simeq 10^{-6.9}. \tag{A.5}$$

Thus single precision numbers have $6 - 7$ decimal places of significance.

There are special cases. The zero can not be normalized. It is represented by two floats ± 0. Also $\pm\infty$ are special numbers. Finally NaN (not a number) is also a special case. Explicitly we have

$$\pm 0 = (-1)^s 0.0...0 \times 2^{-126}. \tag{A.6}$$

$$\pm \infty = (-1)^s 1.0...0 \times 2^{127}. \tag{A.7}$$

$$\text{NaN} = (-1)^s 1.f \times 2^{127} \ , \ f \neq 0. \tag{A.8}$$

The double precision floating point numbers (doubles) occupy 64 bits. The first bit is for the sign, 11 bits for the exponent and 52 bits for the significand. They are stored as two $32-$bist words. Explicitly

	s	e	f	f
Bit Position	63	62-52	51-32	31-0

In this case the bias is $bias = 1023$. They correspond approximately to 16 decimal places of precision. They are in the range

$$4.9 \times 10^{-324} \leq \text{double precision} \leq 1.8 \times 10^{308}. \tag{A.9}$$

The above description corresponds to the IEEE 754 standard adopted in 1987 by the Institute of Electrical and Electronics Engineers (IEEE) and American National Standards Institute (ANSI).

Machine Precision and Roundoff Errors: The gap ϵ between the number 1 and the next largest number is called the machine precision. For single precision we get $\epsilon = 2^{-23}$. For double precision we get $\epsilon = 2^{-52}$.

Alternatively the machine precision ϵ_m is the largest positive number which if added to the number stored as 1 will not change this stored 1, viz

$$1_c + \epsilon_m = 1_c. \tag{A.10}$$

Clearly $\epsilon_m < \epsilon$. The number x_c is the computer representation of of the number x. The relative error ϵ_x in x_c is therefore such that

$$|\epsilon_x| = |\frac{x_c - x}{x}| \leq \epsilon_m. \tag{A.11}$$

All single precision numbers contain an error in their 6th decimal place and all double precision numbers contain an error in their 15th decimal place.

An operation on the computer will therefore only approximate the analytic answer since numbers are stored approximately. For example the difference $a = b - c$ is on the computer $a_c = b_c - c_c$. We compute

$$\frac{a_c}{a} = 1 + \epsilon_b \frac{b}{a} - \epsilon_c \frac{c}{a}. \tag{A.12}$$

In particular the subtraction of two very large nearly equal numbers b and c may lead to a very large error in the answer a_c. Indeed we get the error

$$\epsilon_a \simeq \frac{b}{a}(\epsilon_b - \epsilon_c). \tag{A.13}$$

In other words the large number b/a can magnify the error considerably. This is called subtractive cancellation.

Let us next consider the operation of multiplication of two numbers b and c to produce a number a, viz $a = b \times c$. This operation is represented on the computer by $a_c = b_c \times c_c$. We get the error

$$\epsilon_a = \epsilon_b + \epsilon_c. \tag{A.14}$$

Let us now consider an operation involving a large number N of steps. The question we want to ask is how does the roundoff error accumulate.

The main observation is that roundoff errors grow *slowly* and *randomly* with N. They diverge as N gets very large. By assuming that the roundoff errors in the individual steps of the operation are not correlated we can view the accumulation of error as a random walk problem with step size equal to the machine precison ϵ_m. We know from the study of the random walk problem in statistical mechanics that the total roundoff error will be proportional to $\sqrt{N}$, namely

$$\epsilon_{\mathrm{ro}} = \sqrt{N}\epsilon_m. \tag{A.15}$$

This is the most conservative estimation of the roundoff errors. The roundoff errors are analogous to the uncertainty in the measurement of a physical quantity.

Systematic (Algorithmic) Errors: This type of errors arise from the use of approximate numerical solutions. In general the algorithmic (systematic) error is inversely proportional to some power of the number of steps N, i.e.

$$\epsilon_{\text{sys}} = \frac{\alpha}{N^\beta}. \tag{A.16}$$

The total error is obtained by adding the roundoff error, viz

$$\epsilon_{\text{tot}} = \epsilon_{\text{sys}} + \epsilon_{\text{ro}} = \frac{\alpha}{N^\beta} + \sqrt{N}\epsilon_m. \tag{A.17}$$

There is a competition between the two types of errors. For small N it is the systematic error which dominates while for large N the roundoff error dominates. This is very interesting because it means that by trying to decrease the systematic error (by increasing N) we will increase the roundoff error. The best algorithm is the algorithm which gives an acceptable approximation in a small number of steps so that there will be no time for roundoff errors to grow large.

As an example let us consider the case $\beta = 2$ and $\alpha = 1$. The total error is

$$\epsilon_{\text{tot}} = \frac{1}{N^2} + \sqrt{N}\epsilon_m. \tag{A.18}$$

This error is minimum when

$$\frac{d\epsilon_{\text{tot}}}{dN} = 0. \tag{A.19}$$

For single precision calculation ($\epsilon_m = 10^{-7}$) we get $N = 1099$. Hence $\epsilon_{\text{tot}} = 4 \times 10^{-6}$. Most of the error is roundoff. In order to decrease the roundoff error and hence the total error in this example we need to decrease the number of steps. Furthermore in order for the systematic error to not increase when we decrease the number of steps we must find another algorithm which converges faster with N. For an algorithm with $\alpha = 2$ and $\beta = 4$ the total error is

$$\epsilon_{\text{tot}} = \frac{2}{N^4} + \sqrt{N}\epsilon_m. \tag{A.20}$$

This error is minimum now at $N = 67$ for which $\epsilon_{\text{tot}} = 9 \times 10^{-7}$. We have only $1/16$ as many steps with an error smaller by a factor of 4.

Appendix B

Executive Arabic Summary of Part I

اعمال تطبيقية في الفيزياء العددية

باديس يدري

معهد الفيزياء، جامعة باجي مختار، عنابة، الجزائر

جانفي 2015

الفهرس

مقدمة

الفيزياء العددية هي احد فروع العلوم العددية التي تعرف ايضا باسم الحاسوبية العلمية و التي ظهرت و تبلورت خلال 30 – 40 سنة الاخيرة مع التقدم الهائل الذي حصل في التكنولوجيا الرقمية خاصة في الولايات المتحدة الامريكية.

يمكن اعتبار الفيزياء العددية قسم من اقسام الفيزياء النظرية او يمكن اعتبارها جسر يربط بين الفيزياء النظرية و الفيزياء التجريبية و هناك حتي من يعتبرها تخصص قائم لخدمة الفيزياء التجريبية لا حسب. تقليديا هناك مقاربتان متكاملتان، علي الاقل منذ عصر نيوتن، للفيزياء. فهناك من ناحية المقاربة النظرية و من ناحية اخري هناك المقاربة التجريبية. هناك الكثير الان، خاصة من العاملين في هذا المجال و من غيرهم، من يعتبر انه توجد مقاربة ثالثة منفصلة و مختلفة للفيزياء هي المقاربة العددية. من وجهة النظر هذه فان الفيزياء العددية هي حقل منفصل بذاته غير مرتبط بالضرورة بالحقلين النظري و التجريبي. رغم هذا فان وجهة النظر التي سوف نتبناها في هذه المطوية هو الراي الاول الذي يعتبر ان الفيزياء العددية هو فرع من فروع الفيزياء النظرية.

في الفيزياء العددية يتم مزج عناصر من الفيزياء و خاصة الفيزياء النظرية و عناصر من الرياضيات التطبيقية مثل التحليل العددي مع عناصر من علوم الحاسوب مثل البرمجة من اجل هدف واحد هو حل مسالة فيزيائية معينة ليس لها حل كامل او حل معروف.

اهم استعمالات الكمبيوتر في الفيزياء هو اجراء المحاكيات (جمع محاكاة) العددية. المحاكيات العددية تلائم اكثر المسائل الفيزيائية التي تتحكم فيها معادلات رياضية غير خطية و التي لا تتوفر في معظمها علي حل تحليلي مضبوط. نقطة البدء لاي محاكاة عددية هو نموذج مثالي للجملة الفيزيائية قيد الدراسة و من الطبيعي اننا نريد ان نتأكد ما اذا كان تصرف هذا النموذج منسجم مع المشاهدة او لا في حالة توفر نتائج تجريبية للمقارنة اما في حالة عدم توفر اي نتائج تجريبية فان الهدف هو استشراف ما يمكن ان تعطيه التجربة اذا ما اجريت. الخطوة الاولي من اجل تحقيق هذا الهدف هو ايجاد خوارزمية رياضية من اجل انجاز هذا النموذج نظريا و ايضا علي الكمبيوتر. تنفيذ هذا الانجاز علي كمبيوتر هو ما نسميه بالمحاكاة العددية و هو يعتمد علي ترجمة الخوارزمية الرياضية الي شفرة مكتوبة باحدي لغات البرمجة يمكن للكمبيوتر ان يفهمها.

المحاكيات العددية هي اذن تجارب افتراضية. فمثلا يلعب النموذج الرياضي في المحاكاة العددية بالضبط دور العينة في التجربة المعملية اما الخوارزمية اوالشفرة التي تستعملها المحاكاة العددية فهي تقوم بدور جهاز القياس في التجربة المعملية. قبل البدء في استعمال المحاكاة العددية في الدراسة الفيزيائية فانه علينا اختبار او معايرة الشفرة تماما كما اننا نقوم بمعايرة جهاز القياس في التجربة المعملية قبل البدء في اجراء اي قياس. القياس الذي نقوم به في التجربة المعملية يقابله الحساب الذي تجريه المحاكاة العددية و نختتم كلا العمليتين بنفس الأمر و هو تحليل المعطيات.

من الواضح جدا و من الطبيعي ان اهم وسائل الفيزياء العددية هي لغات البرمجة. في معظم المحاكيات العددية التي نجدها في الاعمال البحثية الفيزيائية تكتب الشفرات في احدي اللغات المجمعة مثل الفرترون (Fortran) او لغة سي (C). في هذه المحاكيات يمكن ايضا عند الحاجة مناداة مكتبات الروتينات العددية مثل لاباك (Lapack) و غيرها. استعمال البرمجيات العددية الجاهزة مثل ماتلاب (Matlab) و ماثيماتيكا (Mathematica) في هذه المحاكيات العددية، خاصة التي تعتمد علي طريقة المونتي كارلو (Monte Carlo)، غير عملي بالمرة لانه يؤدي الي زمن

سير طويل جدا للشفرة علي الكمبيوتر و هذا راجع بالخصوص الي كون البرمجيات الجاهزة هي لغات مترجمة و ليست لغات مجمعة. ليس هناك ادني شك في ان البرمجيات الجاهزة مفيدة للغاية في الحسابات العددية التي لا تعتمد علي التكرار لكنها غير ملائمة تماما في المحاكيات العددية التي تعتمد بالاساس علي تكرار نفس الخطوة عدد هائل من المرات. في هذه المطوية سوف نتبع بالضبط هذا الطريق اي سوف نكتب جميع شفراتنا في لغة مجمعة و نتجنب استخدام البرمجيات الجاهزة. سوف نستخدم بالخصوص الفرترون 77 او 90 علي نظام التشغيل لينيكس (Linux) توزيع يوبنتو (Ubuntu) .

هذه المطوية تحتوي علي مجموع الاعمال التطبيقية المرفقة بمحاضرات الفيزياء العددية التي القاها المؤلف باديس ايدري في معهد الفيزياء منذ العام 2009 علي طلبة الليسانس و الماستر في اطار مقاييس التحليل العددي (ليسانس فيزياء)، الفيزياء العددية (ماستر فيزياء نظرية) و الاعلام الالي (باقي تخصصات الماستر). يمكن الحصول علي مطوية ـمحاضرات في الفيزياء العدديةـ عن طريق الاتصال بالمؤلف باديس ايدري عبر البريد الالكتروني badis.ydri@univ − annaba.org او تصفح الموقع الرسمي خاصته علي http : //homepages.dias.ie/ydri/.

في الختام يتقدم المؤلف باديس ايدري اصالة عن نفسه و عن باقي فريق العمل بالشكر الجزيل للمدير السابق لمعهد الفيزياء الاستاذ مصطفي بن شهاب و كذلك للمدير الحالي الاستاذ علاوة شيباني للمساعدات الجليلة و التسهيلات الكثيرة التي قدماها منذ البداية من اجل ادخال هذه المحاضرات و التطبيقات في البرامج الرسمية للفيزياء.

باديس يدري
سرايدي، عنابة، الجزائر
الاثنين 8 جويلية 2013

خوارزمية اولر ـ مقاومة الهواء

يقود رياضي دراجة هوائية علي طريق مستقيمة و مسطحة بسرعة v . القوة التي يطبقها الرياضي علي الدراجة تكافئ استطاعة ثابتة P تساوي 200 واط يوفرها الرياضي لفترة زمنية تقدر بساعة واحدة. قوة مقاومة الهواء (التي تعرف ايضا بقوة الجر الهوائي) تكون معاكسة للحركة و متناسبة طردا مع مربع السرعة معطاة بالعلاقة

$$F_{\text{drag}} = -C\rho A v^2.$$

في هذه المعادلة ρ هي كثافة الهواء، C هو معامل الجر و A هي مساحة المقطع العرضي لجملة الرياضي زائد الدراجة. قانون نيوتن الثاني يأخذ الشكل التالي

$$\frac{dv}{dt} = \frac{P}{mv} - \frac{C\rho A v^2}{m}.$$

المطلوب هو حساب السرعة v كدالة في الزمن. المقاربة العددية لهذه المسألة تعتمد علي خوارزمية اولر. نقطع المجال الزمني T الي N مجال زمني صغير

$$\Delta t = \frac{T}{N}$$

اي

$$t = i\Delta t \ , \ i = 0, ..., N.$$

نعرف

$$\hat{v}(i) = v(t - \Delta t).$$

الحل المعطي بتقريب اولر يأخذ الشكل

$$\hat{v}(i + 1) = \hat{v}(i) + \Delta t \left(\frac{P}{m\hat{v}(i)} - \frac{C\rho A \hat{v}^2(i)}{m} \right) . \ i = 1, ..., N + 1$$

اللحظات الزمنية المرافقة تعطي ب

$$\hat{t}(i + 1) = i\Delta t \ , \ i = 1, ..., N + 1.$$

(1) احسب السرعة كدالة في الزمن في حالة وجود مقاومة الهواء وفي حالة عدم وجود مقاومة الهواء. ماذا تلاحظ. في هذا السؤال نأخذ ثابت الجر C يساوي 0.5. نعطي ايضا القيم

$$m = 70\text{kg} \ , \ A = 0.33 m^2 \ , \ \rho = 1.2\text{kg}/m^3 \ , \ \Delta t = 0.1s \ , \ T = 200s.$$

السرعة الابتدائية تعطي ب
$$\hat{v}(1) = 4m/s \ , \ \hat{t}(1) = 0.$$

(2) ماذا تلاحظ في حالة تغيير ثابت الجر و/او الاستطاعة. ماذا تلاحظ عندما يتم تصغير الخطوة الزمنية.

حركة القذائف تحت تأثير مقاومة الهواء

نعتبر حركة قذيفة تحت تأثير قوة مقاومة الهواء التي تعمل عكس اتجاه الحركة و تكون متناسبة مع مربع السرعة. نرمز لثابت التناسب ب B. قانون نيوتن الثاني يؤدي الي معادلات الحركة التالية

$$\frac{dx}{dt} = v_x \ , \ m\frac{dv_x}{dt} = -Bvv_x.$$

$$\frac{dy}{dt} = v_y \ , \ m\frac{dv_y}{dt} = -mg - Bvv_y.$$

حل هذه المعادلات التفاضلية المعطي بخوارزمية اولر بأخذ الشكل التالي

$$v_x(i+1) = v_x(i) - \Delta t \frac{Bv(i)v_x(i)}{m}.$$

$$v_y(i+1) = v_y(i) - \Delta t g - \Delta t \frac{Bv(i)v_y(i)}{m}.$$

$$v(i+1) = \sqrt{v_x^2(i+1) + v_y^2(i+1)}.$$

$$x(i+1) = x(i) + \Delta t \ v_x(i).$$

$$y(i+1) = y(i) + \Delta t \ v_y(i).$$

القيم الابتدائية للموضع و السرعة توافق القيمة 1 ل i و i يأخذ القيم من 1 الي N.

(1) اكتب شفرة فورترون منجز فيها الحل المعطي بخوارزمية اولر لمسألة.

(2) نأخذ القيم التالية

$$\frac{B}{m} = 0.00004 m^{-1} \ , \ g = 9.8 m/s^2.$$

$$v(1) = 700 m/s \ , \ \theta = 30 \text{ degree}.$$

$$v_x(1) = v(1)\cos\theta \ , \ v_y(1) = v(1)\sin\theta.$$

$$N = 10^5 \ , \ \Delta t = 0.01 s.$$

احسب المسار بدون و ب مقاومة الهواء. ماذا تلاحظ.

(3) باستخدام التصريح الشرطي if يمكن تعيين مدي القذيفة. هذا التصريح يضاف داخل حلقة do كالتالي

$$\text{if}(y(i+1).le.0)\text{exit}.$$

عين مدي القذيفة في حالة وجود و في حالة عدم وجود مقاومة للهواء.

(4) في حالة عدم وجود مقاومة للهواء نعرف ان المدي يأخذ اعظم قيمة له لما تكون الزاوية الابتدائية تساوي 45 درجة. تحقق من هذا الامر عدديا باختبار عدة قيم للزاوية الابتدائية. يمكن ايضا اضافة حلقة do في الزاوية ثم دراسة المدي كدالة في الزاوية و البحث عن قيمته العظمي.

(5) في حالة وجود مقاومة الهواء احسب الزاوية التي يكون فيها المدي اعظمي.

الهزاز التوافقي- خوارزميات اولر- كرومر و فيرلات

نعتبر هزاز توافقي بسيط عبارة عن كتلة m مربوطة بخيط طوله l معلق في مرتكز ثابت تحت تأثير الثقالة g. نفترض ان الحركة خطية اي ان الزاوية التي يصنعها النواس مع المحور الشاقولي تبقي دائما صغيره. معادلة الاهتزاز المشتقة من قانون نيوتن الثاني تأخذ الشكل

$$\frac{d^2\theta}{dt^2} + \frac{g}{l}\theta = 0.$$

هذه المعادلة التفاضلية من الرتبه الثانيه يمكن تعويضها بمعادلتين تفاضليتين من الرتبة الاولي كالتالي

$$\frac{d\theta}{dt} = \Omega \ , \ \frac{d\Omega}{dt} = -\frac{g}{l}\theta.$$

الحل العددي الاول الذي سنعتبره هنا هو الحل المعطي بخوارزمية اولر

$$\Omega_{i+1} = \Omega_i - \frac{g}{l}\theta_i \ \Delta t.$$

$$\theta_{i+1} = \theta_i + \Omega_i \ \Delta t.$$

الحل العددي الاخر الذي سنعتبره هنا هو الحل المعطي بخوارزمية اولر- كرومر. هذا الحل يعطي بالمعادلات التالية

$$\Omega_{i+1} = \Omega_i - \frac{g}{l}\theta_i \ \Delta t.$$

$$\theta_{i+1} = \theta_i + \Omega_{i+1} \ \Delta t.$$

نعتبر هنا ايضا الحل المعطي بخوارزمية فيرلات الذي يأخذ الشكل

$$\theta_{i+1} = 2\theta_i - \theta_{i-1} - \frac{g}{l}\theta_i(\Delta t)^2.$$

(1) اكتب شفرة فورترون منجز فيها الحلول المعطاة بخوارزميات اولر و اولر- كرومر لمسألة الهزاز التوافقي.

(2) احسب الزاوية، السرعة الزاوية و الطاقة كدوال في الزمن. طاقة الهزاز تعطي ب

$$E = \frac{1}{2}\Omega^2 + \frac{1}{2}\frac{g}{l}\theta^2.$$

نأخذ القيم العددية

$$g = 9.8m/s^2 \ , l = 1m \ .$$

نأخذ عدد الخطوات و الخطوة الزمنية

$$N = 10000 \ , \ \Delta t = 0.05s.$$

نأخذ الزاوية و السرعة الزاوية الابتدائيتان

$$\theta_1 = 0.1\text{radian} \ , \ \Omega_1 = 0.$$

باستعمال التصريح الشرطي if يمكن تحديد زمن الحركة بخمسة اضعاف الدور كالتالي

$$\text{if}(t(i+1).\text{ge}.5 * \text{period}) \ \text{exit}.$$

(3) **قارن بين قيمة الطاقة المحسوبة باولر و قيمة الطاقة المحسوبة باولر- كرومر. ماذا تلاحظ و ماذا تستنتج.**

(4) **اعد الحساب باستخدام خوارزمية فيرلات. لنلاحظ ان هذه الطريقة لا يمكنها الانطلاق فقط من القيم الابتدائية θ_1 و Ω_1. يجب ايضا اعطاء الزاوية θ_2 التي يمكن حسابها باستعمال طريقة اولر اي**

$$\theta_2 = \theta_1 + \Omega_1\,\Delta t.$$

لنلاحظ ايضا ان خوارزمية فيرلات لا تحتاج الي حساب السرعة الزاوية. لكن من اجل حساب الطاقة نحتاج الي معرفة السرعة الزاوية التي نحسبها باستعمال العبارة

$$\Omega_i = \frac{\theta_{i+1} - \theta_{i-1}}{2\Delta t}.$$

التكاملات العددية

نعتبر التكاملات في بعد واحد من الشكل

$$I = \int_a^b dx f(x).$$

نعتبر الحالة العامة التي لا يمكن فيها اجراء التكامل تحليليا و يبقى الحل العددي هو الخيار الوحيد . الخوارزميات التي سنستعملها هي التقريب بالمستطيلات و بأشباه المنحرف و التقريب بقطوع المكافئ . في كل هذه الطرق نقسم مجال التعريف الي N مجال طوله Δx كالتالي

$$x_i = x_0 + i\Delta x \ , \ i = 0, ..., N \ , \ \Delta x = \frac{b-a}{N} \ , \ x_0 = a, x_N = b.$$

التقريب بالمستطيلات يعطي ب

$$F_N = \Delta x \sum_{i=0}^{N-1} f(x_i).$$

التقريب باشباه المنحرف يعطي ب

$$T_N = \Delta x \left[\frac{1}{2} f(x_0) + \sum_{i=1}^{N-1} f(x_i) + \frac{1}{2} f(x_N) \right].$$

التقريب بقطوع المكافئ (قاعدة سيمبسون) يعطي ب (هنا N يجب ان يكون زوجي)

$$S_N = \frac{\Delta x}{3} \left[f(x_0) + 4 \sum_{i=0}^{\frac{N-2}{2}} f(x_{2i+1}) + 2 \sum_{i=0}^{\frac{N-2}{2}} f(x_{2i}) + f(x_N) \right].$$

الخطأ في هذه التقريبات الثلاث متناسب مع $1/N$، $1/N^2$ و مع $1/N^4$ علي التوالي .

(1) نأخذ التكامل

$$I = \int_0^1 f(x)dx \ ; \ f(x) = 2x + 3x^2 + 4x^3.$$

احسب قيمة هذا التكامل باستعمال طريقة المستطيلات. قارن مع قيمة التكامل التحليلية. ملاحظة: شفر الدالة باستعمال subroutine او function .

(2) غير عدد المجالات N . احسب الخطأ المرتكب بدلالة N. قارن مع النظري.

(3) اعد السؤالين السابقين باستعمال طريقة اشباه المنحرف و قاعدة سيمبسون.

(4) خذ الان التكاملات التالية

$$I = \int_0^{\frac{\pi}{2}} \cos x dx \ , \ \ I = \int_1^e \frac{1}{x} dx \ , \ I = \int_{-1}^{+1} \lim_{\epsilon \longrightarrow 0} \left(\frac{1}{\pi} \frac{\epsilon}{x^2 + \epsilon^2} \right) dx.$$

خوارزمية نيوتن ـ رافسون

جسيم ذو كتلة m يتحرك في بئر كمون ارتفاعه V و طوله $2a$ يمتد من $-a$ الي $+a$. نهتم بحالات الجملة ذات الطاقات الاصغر من ارتفاع البئر اي الحالات المرتبطة. حالة الجملة قد تكون زوجية او فردية. الطاقات المسموح بها المرفقة بالدوال الموجية الزوجية تعطي بحلول المعادلة المتسامية

$$\alpha \tan \alpha a = \beta.$$

$$\alpha = \sqrt{\frac{2mE}{\hbar^2}} \ , \ \beta = \sqrt{\frac{2m(V - E)}{\hbar^2}}.$$

في حالة بئر الكمون اللانهائي نجد الحلول

$$E_n = \frac{(n + \frac{1}{2})^2 \pi^2 \hbar^2}{2ma^2} \ , \ n = 0, 1....$$

نختار(مع اهمال كتابة الوحدات)

$$\hbar = 1 \ , \ a = 1 \ , \ 2m = 1.$$

من اجل ايجاد الطاقات E نستعمل خوارزمية نيوتن ـ رافسون التي تسمح لنا بايجاد جذور اي معادلة $f(x) = 0$ كالتالي. انطلاقا من تخمين معين x_0 فاننا نقرب حل المعادلة $f(x) = 0$ بنقطة تقاطع مماس الدالة $f(x)$ في النقطة x_0 مع محور السينات. نسمي هذا التقريب الاول x_1 و هو يعطي بالمعادلة

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

انطلاقا من x_1 نقوم بنفس الخطوة من اجل ايجاد التقريب الثاني x_2 ثم نستخدم x_2 من اجل ايجاد التقريب الثالث x_3 و هكذا. التقريب x_{i+1} يعطي بدلالة التقريب x_i بالعلاقة

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}.$$

(1) **من اجل $V = 10$ بين عدد الحلول E باستعمال الطريقة البيانية عبر دراسة الدالتين**

$$f(\alpha) = \tan \alpha a \ , \ g(\alpha) = \frac{\beta}{\alpha} = \sqrt{\frac{V}{\alpha^2} - 1}.$$

(2) **جد باستعمال طريقة نيوتن ـ رافسون الحلين بدقة اقل او تساوي من 10^{-8}. من اجل ايجاد الحل الاول نأخذ التخمين الاول لطريقة نيوتن ـ رافسون في نقطة التباعد الاولي لدالة الظل اي $\alpha = \pi/a$. من اجل ايجاد الحل الثاني نأخذ التخمين الاول في نقطة التباعد الثانية اي $\alpha = 2\pi/a$.**

(3) **اعد السؤال من اجل $V = 20$.**

(4) **حدد الحلول الاربعة من اجل$V = 100$. استعن بالطريقة البيانية من اجل تحديد التخمين الاول كل مرة.**

(5) **اعد الاسئلة السابقة باستعمال طريقة التنصيف.**

خوارزمية رونج ـ كوتاـ المجموعة الشمسية

نعتبر مجموعة شمسية مشكلة من كوكب واحد يتحرك حول الشمس. نفترض ان كتلة الشمس ثقيلة جدا بالمقارنة مع كتلة الكوكب بحيث يمكن اعتبارها ساكنة في مركز النظام. قانون نيوتن الثاني يعطي معادلات الحركة التالية

$$v_x = \frac{dx}{dt} \ , \ \frac{dv_x}{dt} = -\frac{GM_s}{r^3}x \ , \ v_y = \frac{dy}{dt} \ , \ \frac{dv_y}{dt} = -\frac{GM_s}{r^3}y.$$

$$r = \sqrt{x^2 + y^2}.$$

نستخدم الوحدات الفلكية حيث

$$GM_s = 4\pi^2 AU^3/yr^2.$$

حل معادلات الحركة الانفة الذكر المعطي بخوارزمية رونج ـ كوتا يأخذ الشكل

$$k_1 = \Delta t \ v_x(i) \ , \ p_1 = \Delta t \ v_y(i).$$

$$r(i) = \sqrt{x(i)^2 + y(i)^2}.$$

$$k_3 = -\frac{GM_s}{r(i)^3}x(i)\Delta t \ , \ p_3 = -\frac{GM_s}{r(i)^3}y(i)\Delta t.$$

$$k_2 = (v_x(i) + \frac{1}{2}k_3)\Delta t \ , \ p_2 = (v_y(i) + \frac{1}{2}p_3)\Delta t.$$

$$R(i) = \sqrt{(x(i) + \frac{1}{2}k_1)^2 + (y(i) + \frac{1}{2}p_1)^2}.$$

$$k_4 = -\frac{GM_s}{R(i)^3}(x(i) + \frac{1}{2}k_1)\Delta t \ , \ p_4 = -\frac{GM_s}{R(i)^3}(y(i) + \frac{1}{2}p_1)\Delta t.$$

$$x(i + 1) = x(i) + k_2.$$

$$v_x(i + 1) = v_x(i) + k_4.$$

$$y(i + 1) = y(i) + p_2.$$

$$v_y(i + 1) = v_y(i) + p_4.$$

في المعادلات السابقة i يأخذ القيم من 1 الي N . القيم الابتدائية للموضع و السرعة توافق القيمة 1 ل i.

(1) اكتب شفرة فورترون منجز فيها الحل المعطي بخوارزمية رونج ـ كوتا لمسألة النظام الشمسي.

(2) احسب المسار و السرعة و كذلك الطاقة كدوال في الزمن. ماذا تلاحظ بالنسبة للطاقة. استخدم الوحدات الفلكية. للتذكير فان طاقة الكوكب في وحدة الكتلة تعطي ب

$$E = \frac{1}{2}v^2 - \frac{GM_s}{r}.$$

(3) حسب قانون كبلر الاول فان جميع المدارات هي قطوع ناقصة مع وجود الشمس في احد المحرقين. في ما يلي سنعتبر فقط الكواكب التي نعلم من المشاهدة ان مداراتها دائرية الي حد كبير. هذه الكواكب هي الزهرة و الارض و المريخ و المشتري و ساتورن. انصاف الاقطار تعطي في وحدة الوحدات الفلكية ب

$$a_{\text{venus}} = 0.72 \ , \ a_{\text{earth}} = 1 \ , \ a_{\text{mars}} = 1.52 \ , \ a_{\text{jupiter}} = 5.2 \ , \ a_{\text{saturn}} = 9.54.$$

تحقق من قانون كبلر الاول من اجل كل هذه الكواكب.

من اجل الاجابة علي السؤالين السابقين 2 و 3 نأخذ الشروط الابتدائية

$$x(1) = a \ , \ y(1) = 0 \ , \ v_x(1) = 0 \ , \ v_y(1) = v.$$

القيمة التي تأخذها السرعة الابتدائية مهمة جدا من اجل الحصول علي المسار الصحيح و يتم تعيينها مثلا من افتراض ان المسار هو فعلا دائري و بالتالي فان قوة الجذب الثقالي تكون متوازنة مع قوة الطرد المركزي. نحصل علي

$$v = \sqrt{\frac{GM_s}{a}}.$$

ايضانأخذ الخطوة الزمنية و عدد التكرارات كالاتي

$$\Delta t = 0.01 yr \ , \ N = 10^3 - 10^4.$$

(4) حسب قانون كبلر الثالث فان مربع الدور يكون متناسب طردا مع مكعب نصف القطر. من اجل المدارات الدائرية فان ثابث التناسب يساوي بالضبط واحد. تحقق من هذا الامر من اجل كل الكواكب المذكورة اعلاه. يتم قياس الدور مثلا بمراقبة متي يرجع الكوكب الي ابعد نقطة له عن الشمس.

(5) بتغيير السرعة الابتدائية بطريقة مناسبة فانه يمكن الحصول علي مدار قطع ناقص. جرب هذا الامر.

(6) القانونان الاساسيان الذان يحكمان حركة النظام الشمسي المبسط الذي اعتبرناه في هذا التمرين هو قانون نيوتن للجذب الثقالي بين الكتل من جهة و قانون نيوتن الثاني من جهةاخري.

قانون الجذب الثقالي ينص في اهم بنوده في ان القوة بين الشمس و الكوكب مركزية موجهة من الكوكب نحو الشمس و متناسبة عكسا مع مربع المسافة. نفترض في الاتي ان قوة الجذب الثقالي هي متناسبة عكسا مع اس اخر للمسافة مختلف عن اثنين. غير الشفرة من اجل اخذ هذا التصرف الجديد للقوة بعين الاعتبار. احسب المدارات من اجل اساسات بين ثلاثة و واحد. ماذا تلاحظ و ماذا تستنتج.

مسألة دوران الحضيض الشمسي لكوكب عطارد

حسب قانون كبلر الاول فان مدارات جميع الكواكب السيارة و من ضمنها عطارد تعطي بقطوع ناقصة مع وجود الشمس في احد المحرقين . هذا القانون يمكن اشتقاقه من تطبيق قوانين نيوتن علي تفاعل الكواكب مع الشمس مع افتراض انه يمكننا اهمال تفاعل الكواكب نفسها فيما بينها . تأثير الكواكب علي بعضها البعض يؤدي الي ظاهرة دوران محاور القطوع الناقصة حول الشمس و بالتالي الي دوران الحضيض الشمسي حول الشمس الذي هو اقرب نقطة في مسار الكوكب من الشمس . هذا الدوران للحضيض الشمسي حول الشمس يحدث لجميع الكواكب لكن مشاهدته صعبة للغاية بسبب كون اغلب المدارات هي دائرية الي حد كبير . فقط بلوتو و عطارد لها مدارات قطوع ناقصة ذات لاتراكزية كبيرة. لكن بالنسبة لبلوتو فان سرعته المدارية المنخفضة لا تسمح بمشاهدة دوران حضيضه الشمسي. يبقي عطارد الذي يمكن قياس دوران حضيضه حول الشمس بدقة معتبرة. قام الفلكيون بقياس سرعة الدوران التالية

$$566 \text{ arcsecond/century}.$$

اي ان حضيض عطارد يصنع دورة كاملة حول الشمس كل 240000 سنة. من جهة اخري فانه بتطبيق قوانين نيوتن علي تفاعل كوكب عطارد مع الشمس مع اخذ بعين الاعتبار تأثير باقي الكواكب علي عطارد نحسب سرعة الدوران

$$523 \text{ arcsecond/century}.$$

الفرق هو

$$43 \text{ arcsecond/century}.$$

هذه الكمية لا يمكن تفسيرها الا من خلال النسبية العامة اي من خلال فهمنا للثقالة علي انها قوة يتوسطها انحناء الفضاء-زمن. القوه الناجمة عن انحناء الفضاء-زمن بسبب كتلة الشمس والتي يستشعرها عطارد اكثر من غيره من الكواكب يمكن تقريبها ب

$$F = \frac{GM_s M_m}{r^2}(1 + \frac{\alpha}{r^2}) \ , \ \alpha = 1.1.10^{-8} AU^2.$$

الهدف هو التحقق عدديا من ان هذه القوة تؤدي فعلا الي كمية دوران للحضيض الشمسي لعطارد تساوي 43 قوس ثانية في القرن.

(1) عدل شفرة الفورترون التي استخدمناها في التطبيق السابق من اجل الاخذ بعين الاعتبار القوة المذكورة اعلاه .

أختيار الشروط الابتدائية مهم للغاية . الشرط الابتدائي الاول هو موضع عطارد . نختار

$$x_0 = (1 + e)a \ , \ y_0 = 0.$$

اي اننا نختار الكوكب في اللحظة الابتدائية في ابعد نقطة له عن الشمس. نصف القطر الكبير a لعطارد هو 0.39 وحدة فلكية و لاتراكزية عطارد e تساوي 0.206 . المسافة ea هي بعد الشمس التي توجد في احد المحرقين عن مركز القطع الناقص . الشرط الابتدائي الثاني هو سرعة عطارد في اللحظة الابتدائية التي تعطي ب

$$v_{x0} = 0 \ , \ v_{y0} = \sqrt{\frac{GM_s}{a}\frac{1 - e}{1 + e}}.$$

هذه السرعة يمكن حسابها من تطبيق قانوني انحفاظ العزم الحركي و انحفاظ الطاقة بين النقطة الابتدائية اعلاه و النقطة $(x = 0, y = b)$ حيث b هو نضف القطر الصغير لعطارد اي

$$b = a\sqrt{1 - e^2}.$$

(2) لان قيمه α التي تعطيها النسبية العامة صغيره جدا فان كمية دوران الحضيض الشمسي لعطارد ضئيلة يصعب ملاحظتها في اي محاكاه عددية ذات وقت محدود . نختار قيمة اكبر بكثير ل α مثلا

$$\alpha = 0.0008 AU^2.$$

نختار ايضا

$$N = 20000 \ , \ dt = 0.0001.$$

احسب المدار من اجل هذه القيم . احسب الزاوية θ التي يصنعها الشعاع الذي يربط عطارد و الشمس مع المحور الافقي يدلالة الزمن . احسب ايضا المسافة بين الشمس و عطارد و مشتقتها بالنسبة للزمن اي

$$\frac{dr}{dt} = \frac{x v_x + y v_y}{r}.$$

هذه المشتقة تغير اشارتها كلما بلغ عطارد ابعد نقطة له عن الشمس او بلغ اقرب نقطة له (اي الحضيض الشمسي) عن الشمس . استخدم هذه الملاحظة من اجل رسم الزاوية θ_p لما يكون عطارد في ابعد نقطة له عن الشمس بدلالة الزمن . ماذا تلاحظ .

عين الميل $d\theta_p/dt$ الذي هو بالضبط كمية دوران الحضيض الشمسي لعطارد حول الشمس من اجل قيمة α المختارة اعلاه .

(3) اعد السؤال السابق من اجل قيم اخري ل α . نقترح

$$\alpha = 0.001, 0.002, 0.004.$$

في كل مرة احسب $d\theta_p/dt$. ارسم $d\theta_p/dt$ بدلالة α . ماذا تلاحظ . اوجد الميل . استنتج كمية دوران الحضيض الشمسي لعطارد من اجل القيمة

$$\alpha = 1.1.10^{-8} AU^2.$$

(4) باستخدام معطيات السؤال السابق اوجد الدالة

$$\frac{d\theta_p}{dt} = f(\alpha).$$

باستعمال طريقة المربعات الاصغرية .

النواس الفوضوي 1 : تأثير الفراشة

نواس عبارة عن كتلة m مربوطة بخيط طوله l معلق في مرتكز ثابت تحت تأثير الثقالة g. حركة اي نواس هي حركة غير خطية عموما لان الزاوية التي يصنعها النواس مع المحور الشاقولي ليست بالضرورة صغيرة ويمكن ان تبلغ القيمة العظمي π او القيمة الصغري $-\pi$ و بالتالي فان النواس يمكن ان يدور دورة كاملة تساوي 360 درجة حول نقطة ارتكازه. نأخذ بعين الاعتبار تأثير قوة مقاومة الهواء علي الكتلة m ونفترض انها تعطي بقانون ستوكس الذي ينص علي ان مقاومة الهواء تكون معاكسه للحركة و متناسبة خطيا مع السرعة $d\theta/dt$ مع ثابت تناسب يساوي mlq:

$$F_{\text{drag}} = -mlq\frac{d\theta}{dt}.$$

الاحتكاك مع الهواء يؤدي الي تخامد حركة النواس و توقفه عن الحركةبعد استهلاك النواس لكامل طاقته الابتدائية . من اجل الحفاظ علي حركة النواس ضد مقاومة الهواء من الضروري اضافة قوة تحريك خارجية التي نفترض انها قوة دورية في الزمن ذات تواتر ν_D و سعة ثابتة mlF_D :

$$F_{\text{drive}} = mlF_D \sin 2\pi\nu_D t.$$

معادلة الاهتزاز المشتقة من قانون نيوتن الثاني تأخذ الشكل

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin\theta - q\frac{d\theta}{dt} + F_D \sin 2\pi\nu_D t.$$

نأخذ دائما التواتر الزاوي $\sqrt{g/l}$ المرفق بالاهتزازات البسيطة للنواس يساوي واحد اي $l = g$. الحل العددي الذي سنعتبره هنا هو الحل المعطي بخوارزمية اولر ـ كرومر:

$$\Omega_{i+1} = \Omega_i + \left(-\frac{g}{l}\sin\theta_i - q\Omega_i + F_D \sin 2\pi\nu_D t_i\right)\Delta t \;,\; \theta_{i+1} = \theta_i + \Omega_{i+1}\,\Delta t.$$

هذه الجملة الديناميكية تعرف باسم النواس الفوضوي (chaotic pendulum) و من اهم ما تتميز به الحساسية المفرطة للشروط الابتدائية . هذه الخاصية تعرف ايضا باسم تأثير الفراشة (butterfly effect) .

يمكن للهزازالفوضوي ان يتصرف بطريقتين مختلفتين . في المنطقة الخطية للهزاز الفوضوي الحركة دورية ذات دور يساوي دور قوة التحريك الخارجية اذا اهملنا الحركة الابتدائية العابرة . في المنطقة الفوضوية الحركة غير دورية لا تكرر نفسها ابدا و بالاضافة الي ذلك فان اي خطأ مهما كان متناه في الصغر في تحديد الشروط الابتدائية يؤدي الي حركة مختلفة بالكامل .

(1) اكتب شفرة منجز فيها الحل المعطي بخوارزمية اولرـ كرومر لمسألة الهزاز الفوضوي . لنلاحظ ان الزاوية θ يمكن دائما اخذها محصورة في المجال $[-\pi,\pi]$ و في الحالة التي تكون فيها خارج هذا المجال نقوم باضافة $\pm 2\pi$ من اجل اعادة حصرها في المجال و هذا كالتالي

$$\text{if}(\theta_i.\text{lt}.\mp\pi)\,\theta_i = \theta_i \pm 2\pi.$$

(2) **نأخذ القيم و الشروط الابتدائية**

$$dt = 0.04s \ , \ 2\pi\nu_D = \frac{2}{3}s^{-1} \ , \ q = \frac{1}{2}s^{-1} \ , \ N = 1000 - 2000.$$

$$\theta_1 = 0.2 \text{ radian} \ , \ \Omega_1 = 0 \text{ radian}/s.$$

$$F_D = 0 \text{ radian}/s^2 \ , \ F_D = 0.1 \text{ radian}/s^2 \ , \ F_D = 1.2 \text{ radian}/s^2.$$

ارسم الزاوية θ بدلالة الزمن. ماذا تلاحظ بالنسبة للقيمة الاولي لقوة التحريك الخارجية، ماهو تواتر الاهتزاز. ماذا تلاحظ بالنسبة للقيمة الثانية لقوة التحريك الخارجية، ماهو تواتر الاهتزاز من اجل الازمنة الصغري و ماهو تواتر الاهتزاز من اجل باقي الازمنة. ماذا تلاحظ بالنسبة للقيمة الثالثة . هل الحركة دورية .

النواس الفوضوي 2 : مقاطع بوانكري

الحركة في المنطقة الفوضوية هي حركة حتمية لان تصرف الهزاز في جميع الازمنة اللاحقة يحسب من حل معادلة الحركة اعلاه مع اعطاء شروط ابتدائية ملائمة لكن لا يمكن التنبؤ بها. لكن هذا لا يعني ان الهزاز الفوضوي هو جملة عشوائية و هي خاصية يمكن رؤيتها بوضوح في مقاطع بوانكري .

عوض رسم المدار من اجل كل الازمنة يمكن ان نرسم فقط النقاط (θ, Ω) في فضاء الطور من اجل الازمنة التي تحقق الشرط $\nu_D t = n$. مجموعة النقاط التي نحصل عليها بهذه الطريقة تسمى مقطع بوانكري .

في المنطقة الخطية للهزاز الفوضوي تتكون حركة الهزاز من حركة ابتدائية عابرة في الازمنة الصغري و حركة دورية في باقي الازمنة . الجزء الدوري لا يتعلق بالشروط الابتدائية و لذلك يسمي المدار في فضاء الطور بالجاذب الدوري للهزاز الفوضوي . يتكون مقطع بوانكري من نقطة واحدة اذا اهملنا الحركة الابتدائية العابرة للهزاز الفوضوي و من الواضح ان هذا المقطع هو جاذب لانه لا يتعلق بالشروط الابتدائية .

مقطع بوانكري في المنطقة الفوضوية هو ايضا جاذب في فضاء الطور اي مدار لا يتعلق بالشروط الابتدائية يسمي بالجاذب الغريب مما يؤكد حقيقة ان الهزاز الفوضوي رغم انه جملة حتمية لا يمكن التنبؤ بتصرفها في المنطقة الفوضوية الا انه ليس بجملة عشوائية .

(1) نعتبر الان هزازان فوضويان A و B متماثلان في كل شئ لكن شروطهما الابتدائية مختلفة اختلافا طفيفا . مثلا نأخذ

$$\theta_1^A = 0.2 \text{ radian} , \ \theta_1^B = 0.201 \text{ radian}.$$

يقاس الاختلاف بين الحركتين A و B بالفرق بين الزاويتين θ_A و θ_B:

$$\Delta \theta_i = \theta_i^A - \theta_i^B.$$

احسب $\ln \Delta \theta$ بدلالة الزمن من اجل

$$F_D = 0.1 \text{ radian}/s^2 , \ F_D = 1.2 \text{ radian}/s^2.$$

ماذا تلاحظ . هل الحركتان A و B متماثلتان . ماذا يحدث في الازمنة الكبري . هل حركة الهزاز الفوضوي هي من النوع الذي يمكن التنبؤ به . بالنسبة للقيمة الثانية استعمل

$$N = 10000 , \ dt = 0.01s.$$

(2) احسب السرعة الزاوية Ω بدلالة الزاوية θ من اجل

$$F_D = 0.5 \text{ radian}/s^2 , \ F_D = 1.2 \text{ radian}/s^2.$$

ماهو المدار في فضاء الطور من اجل الازمنة الصغري و ماذا يمثل. كيف يصبح المدار في الازمنة الكبري . قارن بين الهزازين A و B . هل يتعلق المدار في الازمنة الكبري بالشروط الابتدائية .

(3) **للحصول علي مقطع بوانكري عدديا نرسم النقاط** (θ, Ω) **في الازمنة التي تنعدم فيها الدالة** $\sin \pi \nu_D t$ **اي في الازمنة التي تغير فيها هذه الدالةاشارتها:**

$$\text{if}(\sin \pi \nu_D t_i \sin \pi \nu_D t_{i+1} . \text{lt}.0) \text{then}$$

$$\text{write}(*, *) t_i, \theta_i, \Omega_i.$$

تحقق من ان مقطع بوانكري في المنطقة الخطية هو معطي بنقطة وحيدة في فضاء الطور . خذ مثلا

$$F_D = 0.5 \text{ radian}/s^2.$$

و استعمل

$$N = 10^4 - 10^7 \ , \ dt = 0.001s.$$

تحقق من ان مقطع بوانكري في المنطقة الفوضوية هو ايضا جاذب . خذ مثلا

$$F_D = 1.2 \text{ radian}/s^2.$$

و استعمل

$$N = 10^5 \ , \ dt = 0.04s.$$

قارن بين مقطع بوانكري للهزاز A **و مقطع بوانكري للهزاز** B **. ماذا تلاحظ و ماذا تستنتج .**

النواس الفوضوي 3 : ظاهرة تضاعف الدور

من اهم الخصائص الفوضوية التي يتميز بها النواس الفوضوي هو ظاهرة تضاعف الدور. المدارات الدورية التي لها نفس دور قوة التحريك الخارجية تسمي الحركة ذات الدور واحد (period_1 motion) . لكن توجد ايضا مدارات ذات دور يساوي ضعف دور القوة الخارجية و مدارات ذات دور يساوي اربعة اضعاف دور القوة الخارجية و بصفة عامة مدارات ذات دور يساوي $2^{\mathcal{N}}$ ضعف دور القوة الخارجية . المدارات التي دورها يساوي $2^{\mathcal{N}}$ ضعف دور قوة التحريك الخارجية تسمي الحركة ذات الدور $\mathcal{N}$ (period_$\mathcal{N}$ motion) . في عالم الاهتزازات و الامواج المدارات التي نحصل عليها في العادة هي مدارات دورية ذات ادوار تساوي دور قوة التحريك الخارجية تقسيم $2^{\mathcal{N}}$ و هي ظاهرة تعرف باسم المزج (mixing). اذن ظاهرة تضاعف الدور التي تشاهد في النواس الفوضوي هي ظاهرة جديدة تنتمي الي عالم الفوضي . التحول الي الفوضي يحدث بالضبط لما $\mathcal{N} \longrightarrow \infty$.

من اجل الحركة ذات الدور $\mathcal{N}$ نتوقع ان توجد $\mathcal{N}$ قيمة مختلفة للزاوية θ من اجل كل قيمة ل F_D . الدالة θ بدلالة F_D تسمي مخطط انشطار (bifurcation) و هو منحني ذو بنية منكسرة (fractal) في المنطقة الفوضوية. من هذا المخطط يمكن حساب متي يحدث بالضبط التحول نحو الفوضي.

(1) نأخذ القيم و الشروط الابتدائية

$$l = g \ , \ 2\pi\nu_D = \frac{2}{3}s^{-1} \ , \ q = \frac{1}{2}s^{-1} \ , \ N = 3000 - 100000 \ , \ dt = 0.01s.$$

$$\theta_1 = 0.2 \text{ radian} \ , \ \Omega_1 = 0 \text{ radian}/s.$$

عين دور الحركة من اجل القيم

$$F_D = 1.35 \text{ radian}/s^2 \ , \ F_D = 1.44 \text{ radian}/s^2 \ , \ F_D = 1.465 \text{ radian}/s^2.$$

ماذا يحدث للدور عندما نزيد في قيمة F_D. هل القيمتان الثانيتان ل F_D تقعان في المنطقة الخطية ام في المنطقة الفوضوية للهزاز الفوضوي.

(2) احسب الزاوية θ بدلالة F_D من اجل الازمنة التي تحقق الشرط $2\pi\nu_D t = 2n\pi$. نأخذ F_D في المجال

$$F_D = (1.34 + 0.005k) \text{ radian}/s^2 \ , \ k = 1, ..., 30.$$

عين مجال قوة التحريك الخارجية الذي تكون فيه المدارات تنتمي الي الحركات ذات الدور واحد، اثنان و اربعة.

في هذا السؤال من المهم جدا ازالة الحركة الابتدائية العابرة قبل البدء في قياس مخطط الانشطار. يمكن انجاز هذا الامر كالتالي. نقوم بحساب الحركة لمدة $2N$ خطوة ثم نأخذ بعين الاعتبار فقط ال N خطوة الاخيرة عند حساب مقطع بوانكري من اجل كل قيمة ل F_D.

النواس الفوضوي 4: مخططات الانشطار و الانكسار التلقائي للتناظر

الهزاز الفوضوي يعطي بمعادلة الحركة

$$\frac{d^2\theta}{dt^2} = -\sin\theta - \frac{1}{Q}\frac{d\theta}{dt} + F_D \cos 2\pi\nu_D t.$$

نأخذ عبر كل هذه المحاكاة القيم التالية

$$F_D = 1.5 \text{ radian}/s^2 \ , \ 2\pi\nu_D = \frac{2}{3}s^{-1}.$$

من اجل تحري دقة عددية اعلي نستخدم هذه المرة خوارزمية رونج ـ كوتا:

$$k_1 = \Delta t \ \Omega(i).$$

$$k_3 = \Delta t \left[-\sin\theta(i) - \frac{1}{Q}\Omega(i) + F_D \cos 2\pi\nu_D \Delta t(i-1) \right].$$

$$k_2 = \Delta t \left(\Omega(i) + \frac{1}{2}k_3 \right).$$

$$k_4 = \Delta t \left[-\sin\left(\theta(i) + \frac{1}{2}k_1\right) - \frac{1}{Q}\left(\Omega(i) + \frac{1}{2}k_3\right) + F_D \cos 2\pi\nu_D \Delta t(i-\frac{1}{2}). \right].$$

$$\theta(i+1) = \theta(i) + k_2.$$

$$\Omega(i+1) = \Omega(i) + k_4.$$

$$t(i+1) = \Delta t \ i.$$

في المنطقة الخطية المدارات هي قطوع ناقصة تتميز بالتناظر

$$\theta \longrightarrow -\theta.$$

هذه المدارات بالاضافة الي كونها دورية ذات دور T_D يساوي دور القوة الخارجية فهي تتميز بتناظر كامل بين اليمين و اليسار و بالتالي فان الوقت الذي يصرفه النواس في حركته الي يمين محوره الشاقولي يساوي الوقت الذي يصرفه في حركته الي يسار محوره الشاقولي. من المثير للاهتمام وجود حلول اخري لمعادلات حركة الهزاز الفوضوي دورية ذات دور يساوي T_D لكنها لا تتميز بالتناظر $\theta \longrightarrow -\theta$. في هذه الحلول نجد ان الهزاز يصرف معظم وقته اما في المنطقة $0 > \theta$ او في المنطقة $0 < \theta$. يمكن وصف هذه الحلول غير المتناظرة بمخطط انشطار

$$\Omega = \Omega(Q).$$

من اجل كل قيمة لمعامل الجودة Q فاننا نحسب مقطع بوانكري اي قيم θ و Ω في اللحظات $t = nT_D$. نلاحظ ان مقطع بوانكري ينشطر من اجل قيمة معينة Q_* ل Q. تحت هذه القيمة نحصل علي خط واحد لان الحركة ذات دور T_D و فوق Q_* نحصل علي خطين رغم ان دورالحركة ما زال يساوي T_D. الخطان يقابلان الحلان اللذان يصرف فيهما الهزاز اغلب وقته في المنطقة اليمني $(0 < \theta)$ او المنطقة اليسري $(0 > \theta)$. الوصول الي احد الحلين انطلاقا من

الحل المتناظر يتعلق بالشروط الابتدائية و يكون عبر زيادة قيمة Q تدريجيا. هذا مثال لظاهرة الانكسار التلقائي للتناظر.

كما رأينا في المحاكاة السابقة يمكن ايضا وصف ظاهرة تضاعف الدور بمخطط انشطار. هذه الظاهرة هي ايضا مثال لظاهرة الانكسار التلقائي للتناظر. في هذه الحالة فان التناظر الذي ينكسر هو

$$t \longrightarrow t + T_D.$$

فقط الحركات التي دورها يساوي T_D تتميز بهذا التناظر. لنلاحظ ان الحركات ذات الدور $\mathcal{N}$ اي المدارات التي لها دور يساوي $2^{\mathcal{N}}T_D$ لا تتميز ايضا بالتناظر $\theta \longrightarrow -\theta$.

لتكن $Q_{\mathcal{N}}$ قيمة Q التي يحدث فيها الانشطار رقم $\mathcal{N}$. اي ان $Q_{\mathcal{N}}$ هي القيمة التي يتحول عندها المدار من مدار ذو دور يساوي $2^{\mathcal{N}-1}T_D$ الي مدار ذو دور يساوي $2^{\mathcal{N}}T_D$. نسبة فاينباوم تعرف كالتالي

$$F_{\mathcal{N}} = \frac{Q_{\mathcal{N}-1} - Q_{\mathcal{N}-2}}{Q_{\mathcal{N}} - Q_{\mathcal{N}-1}}.$$

لما نقترب من المنطقة الفوضوية اي لما $\mathcal{N} \longrightarrow \infty$ فان $F_{\mathcal{N}}$ يقترب بسرعة من القيمة الثابتة

$$F = 4.669.$$

هذه النتيجة عامة لا تختص بالهزاز الفوضوي دون غيره من الجمل الفوضوية. في اي جملة ديناميكية يمكنها ان تتحول الي الفوضي عبر سلسلة غير منتهية من الانشطارات المرفقة بتضاعف للدور فان ثابت فاينباوم يقترب من نفس القيمة 4.669 لما $\mathcal{N} \longrightarrow \infty$.

(1) اعد كتابة الشفرة باستخدام رونج ـ كوتا.

(2) نأخذ مجموعتين مختلفتين من الشروط الابتدائية

$$\theta = 0.0 \text{ radian} \ , \ \Omega = 0.0 \text{ radian}/s.$$

$$\theta = 0.0 \text{ radian} \ , \ \Omega = -3.0 \text{ radian}/s \ .$$

ادرس طبيعة المدار من اجل القيم

$$Q = 0.5s \ , \ Q = 1.24s \ , \ Q = 1.3s.$$

ماذا تلاحظ.

احسب مقطع بوانكري من اجل قيم Q في المجال

$$[1.2, 1.3].$$

ارسم مخطط الانشطار $\Omega = \Omega(Q)$. ماهي القيمة Q_* التي ينكسر فيها التناظر $\theta \longrightarrow -\theta$ تلقائيا.

(3) احسب المدار و مقطع بوانكري من اجل

$$Q = 1.36s.$$

ماهو دور الحركة. **هل المدار متناظر تحت تأثير** $t \longrightarrow t + T_D$. **هل المدار متناظر تحت تأثير** $\theta \longrightarrow -\theta$. **ارسم مخطط الانشطار** $\Omega = \Omega(Q)$ **من اجل مجموعتين مختلفتين من الشروط الابتدائية.**

ماهي القيمة Q_1 **التي يتضاعف فيها الدور اي القيمة التي ينكسر فيها التناظر** $t \longrightarrow t + T_D$

(4) **في هذا السؤال و الذي يليه نستخدم الشروط الابتدائية**

$$\theta = 0.0 \text{ radian} , \ \Omega = 0.0 \text{ radian}/s.$$

احسب المدار و مقطع بوانكري و ارسم مخطط الانشطار $\Omega = \Omega(Q)$ **من اجل قيم** Q **في المجال**

$$[1.34, 1.38].$$

عين من مخطط الانشطار القيم $Q_\mathcal{N}$ **من اجل** $\mathcal{N} = 1, 2, 3, 4, 5$. **احسب ثابت فاينباوم و نقطة التراكم** Q_∞ **التي يحدث عندها التحول نحو الفوضي.**

(5) **حتي نفهم التحول نحو الفوضي بطريقة افضل نعتبر هزازان فوضويان مختلفان اختلافا طفيفا. مثلا نأخذ**

$$\Delta\theta = 10^{-6} \text{ radian} , \ \Delta\Omega = 10^{-6} \text{ radian}/s.$$

احسب المدار و مقطع بوانكري و عين الدور و كذلك احسب $\ln|\Delta\Omega|$ **من اجل قيم** Q **التالية**

$$Q = 1.372s , \ 1.375s , \ 1.3757s , \ 1.376s.$$

ماذا تلاحظ لما نقترب من منطقة الفوضي.

الديناميك الجزيئي 1: توزيع ماكسويل

نعتبر حركة N ذرة ارغون في بعدين داخل علبة مساحتها L^2. طاقة التفاعل بين اي ذرتين مفصولتين بمسافة r تعطي بكمون لينارد-جونز u المعرف ب

$$u = 4\epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right].$$

القوة التي تطبقها الذرة k علي الذرة i هي

$$f_{k,i} = \frac{24\epsilon}{r_{ki}} \left[2 \left(\frac{\sigma}{r_{ki}} \right)^{12} - \left(\frac{\sigma}{r_{ki}} \right)^6 \right].$$

معادلات حركة الذرة i تعطي ب

$$\frac{d^2 x_i}{dt^2} = a_{x,i} = \frac{1}{m} \sum_{k \neq i} f_{k,i} \frac{x_i - x_k}{r_{ki}} \ , \ \frac{d^2 y_i}{dt^2} = a_{y,i} = \frac{1}{m} \sum_{k \neq i} f_{k,i} \frac{y_i - y_k}{r_{ki}}.$$

الخوارزمية العددية التي سنستعملها لحل هذه المعادلات التفاضلية هي خوارزمية قيرلات التي تعطي بالمعادلات

$$x_{i,n+1} = 2x_{i,n} - x_{i,n-1} + (\Delta t)^2 a_{x,i,n} \ , \ y_{i,n+1} = 2y_{i,n} - y_{i,n-1} + (\Delta t)^2 a_{y,i,n}.$$

سنحسب ايضا السرعات باستعمال المعادلات التالية

$$v_{x,i,n} = \frac{x_{i,n+1} - x_{i,n-1}}{2\Delta t} \ , \ v_{y,i,n} = \frac{y_{i,n+1} - y_{i,n-1}}{2\Delta t}.$$

من اجل التبسيط نستخدم الوحدات المختزلة $\sigma = \epsilon = m = 1$. ايضا من اجل التقليل من اثار الحواف نعمل الشروط الحدية الدورية. اي نعتبر العلبة التي تحتوي علي الغاز علي انها تورص بدون حواف و بالتالي فانه عندما تصطدم ذرة ارغون بجدران العلبة في اي اتجاه فاننا نزيد او ننقص طول العلبة في ذلك الاتجاه كما يلي

$$\text{if } (x_i > L) \text{ then } x_i = x_i - L \ , \ \text{ if } (x_i < 0) \text{ then } x_i = x_i + L$$

$$\text{if } (y_i > L) \text{ then } y_i = y_i - L \ , \ \text{ if } (y_i < 0) \text{ then } y_i = y_i + L.$$

بسبب الشروط الحدية الدورية فان ألمسافة العطمي في الاتجاه x بين اي ذرتين هو فقط $L/2$ و كذلك المسافة العظمي في الاتجاه y بين اي ذرتين هو $L/2$. يتم تنفيذ هذا الامر كالتالي

$$\text{if } (x_{ij} > L/2) \text{ then } x_{ij} = x_{ij} - L \ , \ \text{ if } (x_{ij} < -L/2) \text{ then } x_{ij} = x_{ij} + L$$

$$\text{if } (y_{ij} > L/2) \text{ then } y_{ij} = y_{ij} - L \ , \ \text{ if } (y_{ij} < -L/2) \text{ then } y_{ij} = y_{ij} + L.$$

في هذه المسألة نأخذ L فردي و N مربع تام. الشبكة تتميز بطول الخطوة

$$a = \frac{L}{\sqrt{N}}.$$

اذن الشبكة تتشكل من N خلية مساحة كل منها هي a^2. نختار L و N بحيث $a > 2\sigma$. نختار مواضع الذرات كالتالي. الذرة $j + (i - 1)\sqrt{N} = k$ توضع في مركز الخلية ذات الاركان (i, j)، $(i + 1, j)$، $(i, j + 1)$ و $(i + 1, j + 1)$. نقوم بعد ذلك بادخال اضطراب عشوائي علي هذه الوضعيات الابتدائية عن طريق اضافة اعداد عشوائية في المجال $[-a/4, +a/4]$ الي احداثيات الذرات. نختار السرعات الابتدائية في اتجاهات عشوائية لكن بطويلة تساوي v_0 من اجل جميع الذرات.

(1) **اكتب شفرة ديناميك جزيئي باتباع الخطوات اعلاه.** خذ $L = 15$، $N = 25$، $\Delta t = 0.02$، $\text{Time} = 500$ و $v_0 = 1$. كاختبار اولي تحقق من ان الطاقة الكلية للجملة منحفظة. ارسم مسارات الجسيمات. ماذا تلاحظ.

(2) كاختبار ثاني نقترح قياس درجة الحرارة عن طريق ملاحظة كيفية اقتراب الغاز من التوازن. استعمل نظرية التقسيم المتساوي للطاقة التي تعطي ب

$$k_B T = \frac{m}{2N} \sum_{i=1}^{N} (v_{i,x}^2 + v_{i,y}^2).$$

ارسم T كدالة في الزمن. خذ $\text{Time} = 1000 - 1500$. ماهي درجة حرارة الغاز عند التوازن.

(3) احسب توزيع سرعات ذرات الارغون عن طريق انشاء هيستوغرام السرعات. نأخذ القيمة $\text{Time} = 2000$. نعتبر سرعات كل الجسيمات في كل اللحظات. هناك $\text{Time}.N$ قيمة للسرعة في هذه العينة. ننشئ هيستوغرام هذه العينة عن طريق:

 ◦ ايجاد القيمة العظمي و القيمة الصغري.
 ◦ تقسيم المجال الي سلات.
 ◦ تحديد عدد المرات التي تقع فيها قيمة معينة للسرعة داخل سلة معينة.
 ◦ تنظيم التوزيع.

قارن مع توزيع ماكسويل

$$P_{\text{Maxwell}}(v) = C \frac{v^2}{k_B T} e^{-\frac{mv^2}{2k_B T}}.$$

استنتج درجة الحرارة من القيمة العظمي للتوزيع التي تعطي ب

$$k_B T = m v_{\text{peak}}^2.$$

قارن مع درجة الحرارة المحصل عليها من نظرية التقسيم المتساوي للطاقة. ماذا يحدث اذا زدنا السرعة الابتدائية.

الديناميك الجزيئي 2: الانصهار

نريد في هذه المسألة دراسة الانصهار الذي هو التحول الطوري من الحالة الصلبة الي الحالة السائلة. علينا اولا ان نحدد الشروط الصحيحة للحالة الصلبة. من الواضح ان درجة الحرارة يجب ان تكون منخفضة بما فيه الكفاية و الكثافة مرتفعة بما فيه الكفاية حتي تكون الحالة صلبة. من اجل خفض درجة الحرارة الي اقصي حد ممكن نبدأ من الحالة التي تكون فيها جميع الجسيمات في حالة سكون. من اجل الحصول علي تجاذب اعظمي بين الذرات نختار كثافة مساوية لجسيم واحد في كل وحدة مساحة مختزلة. نختار بالخصوص $N = 16$ و $L = 4$.

(1) بين انه باستعمال الشروط الابتدائية المذكورة اعلاه فاننا نحصل علي حالة صلبة بلورية ذات شبكة مثلثية.

(2) من اجل مشاهدة الانصهار يجب تسخين الجملة عن طريق زيادة الطاقة الحركية للذرات يدويا. يمكننا تحقيق هذا الامر مثلا عن طريق تغيير مواضع الجسيمات كل 1000 خطوة كالتالي

$$hh = \text{int}(n/1000)$$
$$\text{if } (hh * 1000.\text{eq}.n) \text{ then}$$
$$x(i,n) = x(i,n+1) - R(x(i,n+1) - x(i,n))$$
$$y(i,n) = y(i,n+1) - R(y(i,n+1) - y(i,n))$$
$$\text{endif}.$$

هذه العملية تؤدي الي ضرب السرعات بالقيمة R. نختار $R = 1.5$.

تحقق من اننا نحصل بالفعل علي الانصهار بهذه الطريقة. ماذا يحدث للطاقة و درجة الحرارة.

الاعداد العشوائية

الجزء الاول نعتبر مولد اعداد شبه عشوائية يعتمد علي طريقة المتبقيات المتمثلة في العلاقة

$$r_{i+1} = \text{remainder}\left(\frac{ar_i + c}{M}\right).$$

الثوابت a، c و M هم الضارب، المضاف و الطويلة علي التوالي. العدد العشوائي الابتدائي r_i يسمي البذرة. نعطي القيم

$$a = 899, c = 0, M = 32768, r_1 = 12 \quad \text{"good"}$$
$$a = 57, c = 1, M = 256, r_1 = 10 \text{ , "bad"}.$$

الدالة remainder تنفذ في الفورترن كالتالي

$$\text{remainder } \frac{a}{b} = \text{mod}(a, b).$$

(1) احسب سلسلة الاعداد العشوائية باستعمال القيم اعلاه. ارسم r_i بدلالة i. انشئ مخطط التناثر $(x_i = r_{2i}, y_i = r_{2i+1})$.

(2) احسب متوسط الاعداد العشوائية. ماذا تلاحظ.

(3) ليكن N عدد الاعداد العشوائية المولدة. احسب دوال الربط

$$\text{sum}_1(k) = \frac{1}{N - k} \sum_{i=1}^{N-k} x_i x_{i+k} \ , \ \text{sum}_2 = \frac{\text{sum}_1(k) - <x_i>^2}{\text{sum}_1(0) - <x_i>^2}.$$

ماهو تصرف هذه الدوال في k.

(4) احسب دور المولدات العشوائية اعلاه.

الجزء الثاني نأخذ N عدد عشوائي في المجال $[0,1]$ الذي نقسمه الي K مجال صغير او سلة طول كل واحدة هو $\delta = 1/K$. ليكن N_i عدد الاعداد العشوائية التي تقع في السلة i. من اجل سلسة اعداد عشوائية منتظمة عدد الاعداد العشوائية المتوقع في كل سلة هو $n_{\text{ideal}} = N/K$. تعرف احصائية χ^2- كالتالي

$$\chi^2 = \frac{1}{n_{\text{ideal}}} \sum_i (N_i - n_{\text{ideal}})^2.$$

(1) تحقق من النتيجة $n_{\text{ideal}} = N/K$ من اجل المولد rand الذي نجده في المكتبة المعيارية للفورترن. خذ القيم $K = 10$ و $N = 1000$. ارسم N_i بدلالة الموضع x_i للسلة i.

(2) عدد درجات الحرية هو $\nu = K - 1$. القيمة الاكثر احتمالا ل χ^2 هي ν. تحقق من هذه النتيجة من اجل عدد كلي من اختبارات السلة يساوي $L = 1000$ و $K = 11$. في كل مرة احسب عدد المرات L_i من بين ال $L = 1000$ اختبار سلة التي نحصل فيها علي قيمة معينة ل χ^2. ارسم L_i بدلالة χ^2. ماذا تلاحظ.

المشاء العشوائي

الجزء الاول   نعتبر حركة مشاء عشوائي في بعد واحد. المشاء يمكنه الحركة الي اليمين
خطوة تساوي $s_i = a$ باحتمال p او الي اليسار خطوة تساوي $s_i = -a$ باحتمال $q = 1 - p$. بعد N
خطوة موضع المشاء يصبح $x_N = \sum_i s_i$. نأخذ القيم

$$p = q = \frac{1}{2} \ , \ a = 1.$$

من اجل محاكاة حركة المشاء العشوائي نحتاج الي مولد للاعداد العشوائية. في هذه المسألة
نستخدم المولد rand الذي نجده في المكتبة المعيارية للفورترن. نستدعي هذا المولد عن
طريق اصدار الامر التالي

call srand(seed)

rand()

يمكن ان نستنسخ حركة المشاء العشوائي بالشفرة التالية

if $(\text{rand}() < p)$ then

$x_N = x_N + a$

else

$x_N = x_N - a$

endif.

(1) **احسب المواضع** x_i **لثلاث مشاءات عشوائية بدلالة رقم الخطوة** i. **نأخذ** $i = 1,100$. **ارسم المسارات الثلاثة.**

(2) **نعتبر الان حركة** K **مشاء عشوائي حيث** $K = 500$. **احسب المتوسطات**

$$< x_N > = \frac{1}{K} \sum_{i=1}^{K} x_N^{(i)} \ , \ < x_N^2 > = \frac{1}{K} \sum_{i=1}^{K} (x_N^{(i)})^2.$$

في المعادلات اعلاه $x_N^{(i)}$ هو موضع المشاء العشوائي i بعد N خطوة. ادرس تصرف هذه
المتوسطات كدوال في N. قارن مع الحسابات النظرية.

الجزء الثاني   نعتبر الان مشاء عشوائي في بعدين علي شبكة نقاط غير منتهية. انطلاقا من اي
نقطة (i, j) علي الشبكة يمكن للمشاء الوصول الي اي نقطة من نقاط الجوار الاقرب الاربعة
$(i + 1, j)$، $(i - 1, j)$، $(i, j + 1)$ و $(i, j - 1)$ باحتمالات p_x، q_x، p_y و q_y علي التوالي حيث
$p_x + q_x + p_y + q_y = 1$. **من اجل التبسيط نفترض ان** $p_x = q_x = p_y = q_y = 0.25$.

(1) **احسب المتوسطات** $< \vec{r}_N >$ و $< \vec{r}_N^2 >$ **كدوال في عدد الخطوات** N **من اجل** $L = 500$ **مشاء عشوائي. نعتبر القيم** $N = 10, ..., 1000$.

تقريبات النقطة الوسطي و مونتي كارلو

الجزء الاول   يعطي حجم كرة نصف قطرها R في d بعد بالعلاقة

$$V_d = \int_{x_1^2+...+x_d^2 \leq R^2} dx_1...dx_d$$

$$= 2 \int dx_1...dx_{d-1} \sqrt{R^2 - x_1^2 - ... - x_{d-1}^2}$$

$$= \frac{R^d}{d} \frac{2\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2})}.$$

(1) اكتب برنامج يحسب التكامل اعلاه في ثلاث ابعاد باستعمال طريقة النقطة الوسطي. نأخذ طول الخطوة $h = 2R/N$، نصف القطر $R = 1$ و عدد الخطوات في كل اتجاه يساوي $N = N_x = N_y = 2^p$ حيث $p = 1,15$.

(2) بين ان الخطأ يتصرف مثل $1/N$. ارسم لوغاريتم القيمة المطلقة للخطأ المطلق بدلالة لوغاريتم N.

(3) جرب حساب التكامل في بعدين. استعمل فقط الربع الموجب للمستوي الحقيقي و خذ طول الخطوة $h = R/N$ حيث $R = 1$ و $N = 2^p$، $p = 1,15$. نعلم من النظري ان الخطأ يجب ان يتصرف مثل $1/N^2$. ماهو الخطأ في هذه الحالة و لماذا الاختلاف.
ملحوظة: المشتقة الثانية للدالة داخل التكامل غير معرفة عند $x = R$ مما يغير تصرف الخطأ من $1/N^2$ الي $1/N^{1.5}$.

الجزء الثاني   من اجل حساب حجم الكرة في اي بعد d عدديا نستعمل علاقة التكرار

$$V_d = \frac{V_{d-1}}{R^{d-1}} \int_{-R}^{+R} dx_d \left(R^2 - x_d^2\right)^{\frac{d-1}{2}}.$$

(1) احسب الحجوم في الابعاد $d = 4,5,6,7,8,9,10,11$. قارن بالنتيجة المضبوطة المعطاة اعلاه.

الجزء الثالث

(1) استعمل طريقة المعاينة لمونتي كارلو المسماة طريقة الاصابة او الخطأ من اجل حساب التكاملات في الابعاد $d = 2,3,4$ و $d = 10$. هل استعمال طريقة مونتي كارلو هذه اسهل من استعمال طريقة النقطة الوسطي في اي بعد.

(2) استعمل طريقة القيمة الوسطي للعينة لمونتي كارلو من اجل حساب التكاملات في الابعاد $d = 2,3,4$ و $d = 10$. من اجل كل d نجري M قياس كل واحد مشكل من N عينة. نعتبر $M = 1,10,100,150$ و $N = 2^p$ حيث $p = 10,19$. تحقق من ان الخطأ المضبوط يتصرف مثل $1/\sqrt{N}$.

ملحوظة: قارن الخطأ المضبوط الذي هو معروف في هذه الحالة مع الانحراف المعياري للمتوسط σ_M و مع $\sigma/\sqrt{N}$ حيث σ هو الانحراف المعياري في قياس واحد. هذه الكميات الثلاثة يجب ان تكون متساوية.

الجزء الرابع

(1) يمكن ان تعطي قيمة π بالتكامل

$$\pi = \int_{x^2+y^2 \leq R^2} dx \; dy.$$

استعمل طريقة المعاينة لمونتي كارلو (طريقة الاصابة او الخطأ) لحساب قيمة تقريبية ل π.

(2) التكامل اعلاه يمكن ايضا ان يكتب علي الشكل

$$\pi = 2 \int_{-1}^{+1} dx \; \sqrt{1-x^2}.$$

استعمل طريقة القيمة الوسطي للعينة لمونتي كارلو لحساب قيمة تقريبية ل π.

توزيعات الاحتمال غير المنتظمة

الجزء الاول   توزيع غوس يعطي ب

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \, \exp -\frac{(x-\mu)^2}{2\sigma}.$$

الوسيط μ هو المتوسط و σ هو التفاوت اي الجذر التربيعي للانحراف المعياري. نختار $\mu = 0$ و $\sigma = 1$.

(1) اكتب برنامج يحسب سلسلة من الاعداد العشوائية x موزعة حسب $P(x)$ باستعمال طريقة التحويل العكسي (خوارزمية بوكس و مولر) المعطاة بالمعادلات

$$x = r \cos\phi.$$

$$r^2 = -2\sigma^2 \ln v \, , \ \phi = 2\pi w.$$

الاعداد v و w هي اعداد عشوائية منتظمة في المجال $[0,1]$.

(2) ارسم هيستوغرام للاعداد العشوائية المحصل عليها في السؤال السابق باتباع الخطوات التالية:

aـ عين مجال الاعداد العشوائية المحصل عليها في السؤال السابق.

bـ نقسم المجال الي u سلة طول كل واحدة هو $h = \text{interval}/u$. نأخذ $u = 100$.

cـ نحدد موضع كل عدد عشوائي x بين السلات. كل مرة نجد فيها عدد عشوائي في سلة معينة نزيد واحد الي العداد المرفق بهذه السلة.

dـ نرسم نسبة الاعداد العشوائية في كل سلة بدلالة الموضع x. نسبة الاعداد العشوائية في كل سلة تساوي عدد الاعداد العشوائيةالتي تقع في هذه السلة علي hN حيث N هو العدد الكلي للاعداد العشوائية. نأخذ $N = 10000$.

(3) ارسم الهيستوغرام علي سلم لوغاريتمي اي ارسم $\log(\text{fraction})$ بدلالة x^2. اوجد الفت و قارن مع النظرية.

الجزء الثاني

(1) طبق طريقة الرفض و القبول لمونتي كارلو علي المسألة اعلاه.

(2) طبق طريقة فرنانداز و كريادو علي المسالة اعلاه. الخطوات هي كالتالي:

aـ نبدأ من N نقطة x_i حيث $x_i = \sigma$.

ـb **نختار بشكل عشوائي زوج من النقاط (x_i, x_j) من السلسلة و نقوم بالتغيير**

$$x_i \longrightarrow \frac{x_i + x_j}{\sqrt{2}}$$
$$x_j \longrightarrow -x_i + \sqrt{2}x_j.$$

ـc **نكرر الخطوة الثانية حتي نصل الي التوازن. مثلا كرر الخطوة الثانية M مرة حيث $M = 10, 100, ...$**

خوارزمية ميتروبوليس و نموذج ايزينغ

الجزء الاول نعتبر N سبين علي شبكة مربعة حيث L هو عدد مواقع الشبكة في كل اتجاه اي ان $N = L^2$. كل سبين يمكنه ان يأخذ احدي القيمتين $s_i = +1$ (سبين علوي) او $s_i = -1$ (سبين سفلي). كل سبين يتفاعل فقط مع جيرانه الاربعة الاقرب و ايضا مع حقل مغناطيسي خارجي H. نموذج ايزينغ في بعدين يعطي بدالة الطاقة

$$E = -J \sum_{<ij>} s_i s_j - H \sum_i s_i.$$

السبين الموجود في نقطة تقاطع الخط i و العمود j يمثل بعنصر المصفوفة $\phi(i,j)$. الطاقة يمكن ان تعطي اذن ب

$$\begin{aligned} E = & - \frac{J}{2} \sum_{i,j=1}^{L} \phi(i,j) \Big(\phi(i+1,j) + \phi(i-1,j) + \phi(i,j+1) + \phi(i,j-1) \Big) \\ & - H \sum_{i=1} \phi(i,j). \end{aligned}$$

نفرض الشروط الحدية الموافقة للتورص اي

$$\phi(0,j) = \phi(n,j) \ , \ \phi(n+1,j) = \phi(1,j) \ , \ \phi(i,0) = \phi(i,n) \ , \ \phi(i,n+1) = \phi(i,1).$$

نفترض ايضا ان الجملة في حالة توازن حراري مع خزان حرارة ذو درجة حرارة T. التقلبات الحرارية للجملة تحاكي بخوارزمية ميتروبوليس.

(1) اكتب روتين جزئي يحسب الطاقة E و المغنظة M في التمثيلة ϕ لنموذج ايزينغ. المغنظة هو وسيط ترتيب الجملة و هو معرف كالتالي

$$M = \sum_i s_i = \sum_{i,j=1} \phi(i,j).$$

(2) اكتب روتين جزئي ينفذ خوارزمية ميتروبوليس لهذه الجملة. الفرق في الطاقة الناجم عن قلب السبين $\phi(i,j)$ يعطي ب

$$\Delta E = 2J\phi(i,j) \Big(\phi(i+1,j) + \phi(i-1,j) + \phi(i,j+1) + \phi(i,j-1) \Big) + 2H \sum_{i=1} \phi(i,j).$$

(3) نختار $L = 10$، $H = 0$، $J = 1$ و $\beta = 1/T$. نعتبر حالة الانطلاقة الباردة و كذلك حالة الانطلاقة الساخنة المعرفتان علي التوالي ب

$$\phi(i,j) = +1 \ \ \forall \ i,j : \ \text{Cold Start}.$$

$$\phi(i,j) = \text{rand}() \ : \ \text{Hot Start}.$$

شغل خوارزمية ميتروبوليس من اجل زمن موازنة $\text{TTH} = 2^6$ و ادرس تاريخ الطاقة و المغنطة من اجل درجات حرارة مختلفة. الطاقة و المغنظة تقتربان من القيم $E = 0$ و $M = 0$ لما $T \longrightarrow \infty$ و من القيم $E = -2JN$ و $M = +1$ لما $T \longrightarrow 0$.

(4) ضف $\mathrm{TTM} = 2^{10}$ خطوة مونتي كارلو و احسب متوسطات الطاقة و المغنطة.

(5) احسب السعة الحرارية و الحساسية المغناطيسية لهذه الجملة المعرفان ب

$$C_v = \frac{\partial}{\partial \beta} <E> = \frac{\beta}{T}(<E^2> - <E>^2) \ , \ \chi = \frac{\partial}{\partial H} <M> = \beta(<M^2> - <M>^2).$$

(6) احسب النقطة الحرجة و قارن بالنتيجة النظرية المضبوطة

$$k_B T_c = \frac{2J}{\ln(\sqrt{2}+1)}.$$

الجزء الثاني ضف الي الشفرة روتين جزئي اخر ينفذ طريقة المطواة من اجل اي مجموعة من القياسات. احسب الاخطاء في الطاقة، المغنطة، السعة الحرارية و الحساسية المغناطيسية لنموذج ايزينغ باستعمال طريقة المطواة.

التغير الطوري من الرتبة الثانية الفيرومغناطيسي

الجزء الاول   الاساس الحرج المرفق بالسعة الحرارية يعطي ب $\alpha = 0$، اي

$$\frac{C_v}{L^2} \sim (T_c - T)^{-\alpha} \ , \ \alpha = 0.$$

هذا يعني ان السعة الحرارية تتباعد لوغاريتميا عند $T = T_c$ وهذا يظهر علي شكل تزايد لقمة السعة الحرارية (القيمة الاعظمية) مع L لوغاريتميا اي

$$\frac{C_v}{L^2}|_{\text{peak}} \sim \log L.$$

تحقق من هذا التصرف عدديا . استخدم شبكات بين $L = 10 - 30$ و $\text{TTH} = 2^{10}$، $\text{TMC} = 2^{13}$. درجات الحرارة تؤخذ في المجال

$$T = T_c - 10^{-2}\text{step} \ , \ \text{step} = -50, 50.$$

ارسم القيمة الاعظمية ل C_v/L^2 بدلالة $\ln L$.

الجزء الثاني   المغنظة بجوار لكن تحت درجة الحرارة الحرجة تتصرف كالتالي

$$\frac{<M>}{L^2} \sim (T_c - T)^{-\beta} \ , \ \beta = \frac{1}{8}.$$

نقترح دراسة المغنطة بجوار T_c من اجل تعيين قيمة β عدديا. من اجل تحقيق هذا الهدف ارسم $|<M>|$ بدلالة $T_c - T$ حيث T يؤخذ في المجال

$$T = T_c - 10^{-4}\text{step} \ , \ \text{step} = 0, 5000.$$

نعتبر شبكات كبيرة بين $L = 30 - 50$ مع $\text{TTH} = \text{TMC} = 2^{10}$. نذكر ان درجة الحرارة الحرجة في نموذج ايزينغ في بعدين تعطي ب

$$k_B T_c = \frac{2J}{\ln(\sqrt{2}+1)}.$$

الجزء الثالث   الحساسية المغناطيسية بجوار درجة الحرارة الحرجة في نموذج ايزينغ في بعدين تتصرف كالتالي

$$\frac{\chi}{L^2} \sim |T - T_c|^{-\gamma} \ , \ \gamma = \frac{7}{4}.$$

عين γ عدديا . استعمل $\text{TTH} = 2^{10}$، $\text{TMC} = 2^{13}$، $L = 50$ و خذ درجة الحرارة في المجالين

$$T = T_c - 5.10^{-4}\text{step} \ , \ \text{step} = 0, 100.$$

$$T = T_c - 0.05 - 4.5.10^{-3}\text{step} \ , \ \text{step} = 0, 100.$$

دالة الربط (غرين) الثنائية

في هذه المسألة نواصل دراسة التغير الطوري الفيرومغناطيسي. بالخصوص سوف نحسب في هذه المسالة دالة الربط (غرين) الثنائية المعرفة بالعبارة

$$
\begin{aligned}
f(n) &= <s_0 s_n> \\
&= <\frac{1}{4L^2} \sum_{i,j} \phi(i,j)\Big(\phi(i+n,j) + \phi(i-n,j) + \phi(i,j+n) + \phi(i,j-n)\Big) > .
\end{aligned}
$$

(1) **تحقق ان تصرف الدالة $f(n)$ عند $T = T_c$ يعطي ب**

$$
f(n) \simeq \frac{1}{n^\eta} \ , \ \eta = \frac{1}{4}.
$$

(2) **تحقق ان تصرف الدالة $f(n)$ من اجل T اقل من T_c يعطي ب**

$$
f(n) = <M>^2 .
$$

(3) **تحقق ان تصرف الدالة $f(n)$ من اجل T اكبر من T_c يعطي ب**

$$
f(n) \simeq a \ \frac{1}{n^\eta} e^{-\frac{n}{\xi}}.
$$

في جميع الاسئلة اعلاه نأخذ شبكات فردية اي $L = 2LL + 1$ بين $LL = 20 - 50$. نعتبر ايضا القيم $TTH = 2^{10}$، $TTC = 2^{13}$.

(4) **بالقرب من T_c يتباعد طول الربط كالتالي**

$$
\xi \simeq \frac{1}{|T - T_c|^\nu} \ , \ \nu = 1.
$$

في هذا السؤال نأخذ $LL = 20$. نعتبر ايضا القيم $TTH = 2^{10}$، $TTC = 2^{15}$ و درجات الحرارة

$$
T = T_c + 0.1.\text{step} \ , \ \text{step} = 0, 10.
$$

لاحظ ان جيران الدليل i الذين يبعدون عنه مسافة n يمكن ان يعطون بالشفرة التالية

```
do i=1,L
do n=1,LL
if (i+n.le.L)then
ipn(i,n)=i+n
else
ipn(i,n)=(i+n)-L
endif
```

```
if ((i-n).ge.1)then
imn(i,n)=i-n
else
imn(i,n)=i-n+L
endif
enddo
enddo
```

الهستريسيس و التغير الطوري من الرتبة الاولي

في هذه المسألة نعتبر تأثير حقل مغناطيسي علي فيزياء نموذج ايزينغ. سوف نلاحظ بالخصوص تغير طوري من الرتبة الاولي بالقرب من $H = 0$ و كذلك ظاهرة هستريسيس.

(1) نحسب المغنطة و الطاقة بدلالة H من اجل درجات حرارة مختلفة. تجري الموازنة من اجل القيمة الاولي للحقل المغناطيسي و بعد حساب المغنطة المتوسطة نبدأ بتغيير الحقل المغناطيسي بشكل اديباتيكي اي بشكل بطيء جدا عبر خطوات صغيرة حتي لا نخسر موازنة الجملة. نعتبر في هذا السؤال المجال $H = -5,5$ مع خطوات تساوي 0.25.

- عين من اجل $T < T_c$ (مثلا $T = 0.5$ و $T = 1.5$) موقع التغير الطوري من الرتبة الاولي من نقطة لا استمرارية (النقطة التي تقفز عندها) الطاقة و المغنطة. هذا التغير الطوري يحدث عند قيمة غير منعدمة للحقل المغناطيسي بسبب الهستريسيس. القفزة التي نلاحظها في الطاقة عند موقع التغير الطوري توافق قيمة غير منعدمة لكمية الحرارة الكامنة.

- بين ان المغنطة من اجل $T > T_c$ (مثلا $T = 3$ و $T = 5$) تصبح دالة سلسة (دالة مستمرة و قابلة للاشتقاق عدد كيفي من المرات) بالقرب من $H = 0$ و هذا يعني انه لا يوجد اي فرق بين الحالات الفيرومغناطيسية $M \geq 0$ و الحالات الفيرومغناطيسية $M \leq 0$.

(2) نعيد حساب المغنطة كدالة في H من اجل المجال من -5 الي 5 ذهابا و ايابا. يجب ان نلاحظ حلقة هستريسيس.

- تحقق من ان نافذة الهستريسيس تضيق بزيادة درجة الحرارة او بعد تراكم عدد اكبر من خطوات مونتي كارلو.

- ماذا يحدث عند زيادة حجم الشبكة.

تشير ظاهرة الهستريسيس الي ان تصرف الجملة يتعلق بحالتها الابتدائية و تاريخها او ان الجملة عالقة في حالات شبة مستقرة.