

Topics in Computational Physics PHY880

Fall 2004

Simon Catterall

Introduction

What this course is **not** going to be ...

- Comprehensive overview of numerical methods – nothing on solving partial differential equations, general eigensystems, spectral analysis, full matrix linear algebra. Good source for description of (some of) these is found in **Numerical Recipes** – see later. This book will also be useful to us. I recommend it for anybody whose research even occasionally demands numerical work.
- Course in C++. I expect that anyone taking this course has had at least a little programming experience (not necessarily in C++). I will give snippets of C++ code to illustrate implementations of various numerical algorithms and will expect a numerical project to be completed by the end of the course (see later for details). I will not teach C++ formally but nevertheless expect some of the structure and style of programming will rub off during the course of the semester and will expect to see that in your final projects.
- A course for particle theorists only. That is my background and certainly motivates much of what I will say but I hope that this course will prove interesting and useful to a broad audience of both experimentalists and theorists from both particle physics and condensed matter and other subjects.

What this course (hopefully) will include ...

- Fairly comprehensive treatment of a variety of Monte Carlo techniques useful for studying such systems as arise in particle physics, equilibrium statistical mechanics or indeed any system possessing an effective **stochastic dynamics**. The stochastic element may occur because of **coarse graining** or **quantum effects**. Both discrete and continuous time stochastic dynamics will be studied. Various results on the theory of stochastic processes, Markov chains and Master equations will be summarized.
- The structure and organization of simulation programs. I will use C++ as the main vehicle here as it allows a very efficient implementation of the various data structures needed to code a variety of physical system. All systems considered will consist of a very large number of (mutually interacting) degrees of freedom.
- I will try to include enough background to make the discussion sensible to a non-specialist and motivate the use of any particular algorithm. Thus I will start trying to explain why quantum field theory and equilibrium statistical mechanics are basically

the same subject. I will use scalar field theory as pedagogical example but will point out as we go along how the ideas may be generalized to cover magnetic spin systems, relativistic fermions, gluons etc. I would like to include introductions to lattice gauge theories, spin (magnetic) systems and maybe something on random systems.

- Data analysis. Once one has developed a Monte Carlo simulation of some system you need to know how to extract meaningful (and quantitative) results from it. I will cover the basic concepts of **autocorrelation time**, **error analysis** and **model fitting** in reasonable detail.

Grading etc

I will have occasion to set the odd homework which I expect to be done within one week. However, I expect the bulk of the grade from the course will come from a final project which will take the form of a simulation program and subsequent analysis of some physical system. I will supply a list of possible topics around the middle of the semester although I would encourage you to come to me if there is something else you would like to do. You must write your codes in C++.

Books

There is no required text for this course – mainly because none exist! However, I will have occasion to draw upon material in the following books. You may even find some of them in the Physics library. Because of this I will try to post lecture notes as we go.

- Quantum Fields on a Lattice, Istvan Montvay and Gernot Munster, Cambridge Monographs on Mathematical Physics, CUP.
- Stochastic Processes in Physics and Chemistry, N.G. Van Kampen, North Holland Publishing.
- Numerical Recipes in C++, Press, Teukolsky, Vetterling and Flannery, CUP
- A guide to Monte Carlo Simulations in Statistical Physics, D. Landau, CUP.

1 Jumping off point – why is quantum field theory just statistical mechanics ?

1.1 Quantum Mechanics

Let us start from the **Feynman Path Integral** representation of quantum mechanics. It is relatively easy to see (although hard to justify rigorously!) that the quantum mechanical amplitude for a particle which started at $x = x_1$ at $t = -T$ to later be found at $x = x_2$ at time $t = T$ is given by

$$\langle x_1, -T | x_2, T \rangle = \lim_{\delta t \rightarrow 0} \int \prod_{i=1}^{N-1} dx_i e^{\frac{i}{\hbar} \sum_{i=1}^{N-1} L(x_i) \delta t}$$

where the coordinate x is constrained to take on the values x_1 at $t = -T$ and $x = x_2$ at $t = T$ and the intermediate time period is broken up into N timeslices such that $N\delta t = 2T$. The function $L(x)$ is just the **Lagrangian** and the expression converges to the **action** $S = \int L(x) dt$ in the limit. The shorthand notation for this integral in the limit is

$$\int Dx e^{\frac{i}{\hbar} S}$$

If we draw a picture of x versus t it is clear that the quantum mechanical transition amplitude receives contributions from all paths $x(t)$ lying between $x = x_1$ at $t = -T$ and $x = x_2$ at $t = T$. One (or more) of these paths are distinguished by the condition that the action be stationary $\delta S = 0$ under small variations of the path $x(t) \rightarrow x(t) + \epsilon(t)$. It is a well known fact in classical mechanics that such paths correspond to the classical evolution of the coordinate $x_{cl}(t)$ (so-called principle of least action). Furthermore since the quantum amplitude is a pure phase it should be clear that these paths are uniquely picked out as $\hbar \rightarrow 0$.

A moments thought should convince you that a knowledge of the following generating function can be used to compute any such amplitudes

$$Z(J) = \int Dx e^{iS(x) + i \int J(t)x(t)}$$

where the integral is now extended to run from $T = -\infty$ to $T = +\infty$ and an arbitrary source term is added to the classical action. For example

$$\langle x(0) | x(t) \rangle = \frac{\delta}{\delta J(0)} \frac{\delta}{\delta J(t)} \ln Z(J) |_{J=0}$$

At this point one might be tempted to go away and use this formalism to compute quantum amplitudes in quantum mechanics by performing (very large) integrals. Unfortunately the presence of the e^{iS} factor renders the resulting integrals very poorly defined. It is common to avoid this problem by *performing a Wick rotation to Euclidean space*. Which means treating

the time coordinate as pure imaginary $t = -i\tau$. The Lagrangian $L = \int dt (\frac{1}{2}m(\frac{dx}{dt})^2 - V(x))$ then becomes

$$L \rightarrow iH = i \int d\tau (\frac{1}{2}m(\frac{dx}{d\tau})^2 + V(x))$$

and the quantum amplitude $Z(J)$ is given by

$$Z(J) = \int Dx e^{-\frac{1}{\hbar}(H + \int J(\tau)x(\tau))}$$

Actually it is often very convenient to work away from the limit $\delta\tau \rightarrow 0$ and treat the path integral as a (very large) ordinary multiple integral running over the values of the coordinate $x_n = x(\tau_n)$ at a series of discrete times $\tau_n = n\delta\tau$. In such an approximation the (Wick-rotated) Lagrangian looks like

$$H_L = \delta\tau \sum_{n=1}^{N-1} \left(\frac{1}{2}m \left(\frac{x_{n+1} - x_n}{\delta\tau} \right)^2 + V(x_n) \right)$$

The (Euclidean) amplitude in this discrete or **lattice** theory is now given by the multiple integral

$$Z(J_n) = \prod_i \int dx_i e^{-\frac{1}{\hbar}(H_L - \sum_i J_i x_i)}$$

Thus the Euclidean, lattice quantum amplitude can be regarded as a statistical mechanical partition function at temperature $kT = \hbar$. Equilibrium thermal fluctuations may be modeled in a way analogous to quantum fluctuations ! Notice that quantum probability amplitudes are equivalent to statistical mechanical correlation functions. This is at the heart of the use of stochastic techniques to evaluate physical observables in quantum field theory. Of course at the end of the calculation we must still *take the continuum limit* $\delta\tau \rightarrow 0$ (and also analytically continue to $t = i\tau$).

1.2 Quantum Field Theory

With the exception of String Theory most modern theories of particle physics go under the name of Quantum Field Theories. Quantum Field Theories comprise not just a single quantum mechanical coordinate at each time $x(t)$ but an entire field variable $\phi(x)$ where x ranges (usually) over three dimensional space. For simplicity we shall mostly consider toy models with only a single spatial dimension. This will simplify our notation at times and the simulations will be computational less costly but it is easy to generalize all I will say to two or three (spatial) dimensions. Physical observables in these theories are generalizations of the amplitudes discussed in the case of quantum mechanics and can be gotten from a similar path integral representation

$$\langle \phi(x_1, t_1) | \phi(x_2, t_2) \rangle = \frac{\delta}{\delta J(x_1, t_1)} \frac{\delta}{\delta J(x_2, t_2)} \ln \int D\phi e^{\frac{i}{\hbar}(S(\phi) + \int J\phi)}$$

Again to make sense of such integrals we go to Euclidean space and discretize now *both time and space*.

$$\phi(x, t) \rightarrow \phi(x_i, \tau_j)$$

where $x_n = n\Delta x$ $n = 0 \dots L - 1$ and $\tau_i = i\Delta\tau$ $i = 0, \dots T - 1$ Notice the finite spacetime volume $a^2 LT$ associated with such a system. In some cases we will want to extrapolate our final results to infinite spatial volumes and time extents. The resulting **lattice** Hamiltonian then takes the form

$$H = a^2 \sum_{i=0}^{L-1} \sum_{j=0}^{T-1} \left(\frac{1}{2a^2} \left((\phi_{i+1,j} - \phi_{i,j})^2 + (\phi_{i,j+1} - \phi_{i,j})^2 \right) + V(\phi_{i,j}) \right)$$

In this expression we have added a (spatial) gradient energy to the potential, and rescaled the field ϕ to absorb the mass parameter m . We have also set $\Delta\tau = \Delta x = a$ for simplicity. The parameter a is called the lattice spacing. As in quantum mechanics, quantum amplitudes in this theory can be reinterpreted as correlation functions in its statistical cousin and \hbar again plays the role of inverse temperature. In condensed matter systems the lattice spacing a could reflect the physical cut-off associated with the spacing between atoms or molecules in a crystal and is held fixed and finite. In a particle physics context we will want to send $a \rightarrow 0$ at the end of the calculation. Unlike the case of quantum mechanics this is in general a subtle business which involves the use of the **Renormalization Group** and connects to the theory of critical phenomena. Hopefully we will have time to address this issue later.

2 Computing Partition Functions

2.1 Direct evaluation

Now we have reduced lattice QFT to statistical mechanics we can start to address the problem of computing observables. As we have seen Green's functions (probability amplitudes) of the QFT map to correlation functions in the statistical theory. Recall the formula for Z

$$Z(J) = \prod_i^N d\phi_i e^{-H(\phi, J)}$$

In principle this is a N -dimensional multiple integral over the microscopic degrees of freedom ϕ_i which you might hope to evaluate by explicit analytic integration. Of course as you are aware this can only be done exactly for a Hamiltonian which is quadratic in ϕ_i in which cases it reduces to the determinant of a large matrix. To handle the non-gaussian interaction pieces the only systematic analytic technique boils down to power expanding these pieces. The partition function is then a (infinite) sum over moments of this Gaussian distribution. This perturbation approach only has a hope of success in the case where the appropriate interaction parameters are small. In other cases (eg QCD) we must find a more direct approach.

You might think that the lattice theory is ideal in this respect – just use an (almost) exact numerical technique to evaluate Z and its moments. Unfortunately a quick glance at the magnitude of this task reveals that it is quite impossible – even for simple systems where a finite number p of grid points are used to approximate a given integral the final integral involves a number of arithmetic operations which grows exponentially fast with volume $O(p^N)$. Even very modest resolution $p = 10$ and a very small lattice say 10×10 yields a computation time $O(10^{100})!$

2.2 Stochastic Methods

Fortunately there is a way out – the integrand involves an exponential function and typically is very close to zero for almost all configurations of the microscopic variables. Thus we may approximate the integrals rather well by devising a technique to *sample* the integrand in the vicinity of its maximum. Indeed since the integrand is positive definite (lets not worry about relativistic fermions right now) we can interpret it as a probability distribution (just as in statistical mechanics) and try to devise a scheme which generates the *most likely* configurations automatically. In the language of statistical mechanics these are just those configurations typical of *thermal equilibrium*. The question is how can I find an efficient way of generating such equilibrium configurations? The process must necessarily be *stochastic* (i.e involve random numbers) since I merely want to generate a set of configurations drawn from the probability distribution e^{-H} .

The simplest way to achieve this is to imagine constructing a transition matrix W_{ab} which acting on one configuration b of the microscopic variables produces another a . The individual matrix elements are interpreted as giving the probabilities that any individual system starting in state b will transition to state a after a single unit of discrete time. This transition matrix must possess the following properties

- It must have possess $W_{ab} > 0$. The positivity ensures it can be regarded as a probability of transition from state a to state b . The requirement that it is never zero is often called ergodicity – in principle there is a non-zero probability of going from one state to any other.
- $\sum_{a=1}^N W_{ab} = 1$. Normalization. The total probability to do something must sum to one.

Stochastic processes realized in this manner are said to be *Markovian*. To simplify matters further let us examine the case in which the dimension of the state space is finite. This could correspond to the situation in which there is a finite number of states for the microscopic degrees of freedom which live on a finite lattice. In terms of the matrix W_{ab} we can write a discrete time evolution equation for the probability $p^n(a)$ to be in state a at time n as

$$p_{n+1}(a) = \sum_b W_{ab} p_n(b)$$

Notice that the third criterion above leads to conservation of probability

$$\sum_a p_{n+1}(a) = \sum_b p_n(b)$$

It also implies that the vector v_a with $v_a = 1$, all a is a left eigenvector of W_{ab} with unit eigenvalue. As a consequence the matrix W_{ab} will have a right eigenvector which we can call $p_{eq}(a)$ also with eigenvalue 1

$$p_{eq}(a) = W_{ab}p_{eq}(b)$$

This *equilibrium* probability distribution hence represents a fixed point of the stochastic dynamics. However if we are to use this dynamics as a way of generating a given equilibrium distribution we will need it to be an *attractive* fixed point. It is clear that this will occur *provided* the equilibrium eigenvector has the largest eigenvalue. If this is true, then starting from any initial probability distribution we will naturally evolve to the equilibrium one. To see this imagine expanding the initial probability vector on the basis of right eigenvectors of W .

$$p(a) = \sum_n \alpha_n v_n(a)$$

After N steps of the dynamics we find

$$p'(a) = \sum_n \alpha_n \lambda_n^N v_n(a)$$

Since all λ are smaller than one except for the equilibrium vector the latter will dominate at large times

$$p'(a) \sim \alpha_{eq} p^{eq}(a)$$

2.3 Proof that $\lambda = 1$ is largest eigenvalue of W

If v is some (right) eigenvector of W we have

$$\sum_b W_{ab} v_b = \lambda v_a$$

Therefore

$$\sum_b v_b = \lambda \sum_a v_a$$

Thus either $\lambda = 1$ or $\sum v_a = 0$ Taking the absolute value of both sides

$$\sum_b W_{ab} |v_b| \geq |\lambda| |v_a|$$

Then summing over a again

$$\sum_b |v_b| \geq |\lambda| \sum_a |v_a|$$

Thus we see that $|\lambda| \leq 1$ Equality is possible only if $|v_a| = p_{eq}(a)$ the equilibrium distribution. The one remaining subtlety corresponds to removing the absolute value operator from around the components of v . Notice that if we operate on an initial vector which has only positive components the vector obtained after any number of applications of W still has strictly positive components – so that we may remove the absolute value operator from v and the proof is complete – only the equilibrium distribution corresponding to the (unique) maximal eigenvalue ($\lambda = 1$) of W survives in the long time limit.

2.4 Detailed Balance

In our case we are faced with the following problem – we have an equilibrium distribution $P^{eq} = e^{-H}$ we want to generate and we want to know how to choose the matrix to achieve that distribution at large times (notice: in QFT we have a Euclidean time coordinate – this has *nothing* to do with the time we are discussing here which is an artificial dynamics time needed to generate the probability distribution for the microscopic variables). One simple way this can be achieved (which is used in very many algorithms) is to impose the so-called **detailed balance** condition on the matrix elements

$$W_{ab}P^{eq}(b) = W_{ba}P^{eq}(a)$$

This is to be true for all states a and b . Easy to show – apply W_{ab} to $P^{eq}(b)$. We find

$$P'(a) = \sum_b W_{ab}P^{eq}(b) = \sum_b W_{ba}P^{eq}(a) = P^{eq}(a)$$

Thus $P^{eq}(a)$ is indeed a right eigenvector of the transition matrix with eigenvalue 1 i.e the true equilibrium distribution.

Even detailed balance is not sufficient to determine a unique algorithm – we shall explore a variety of numerical algorithms based on choosing different W 's in the next few weeks. They will all satisfy detailed balance. Perhaps the simplest, which can be used almost universally for simulating many different types of system, is the **Metropolis** algorithm. In the case where we have a finite number r of final states the Metropolis algorithm uses the following recipe for the transition matrix.

$$\begin{aligned} W_{ab} &= \frac{1}{r} \text{ if } P^{eq}(a) \geq P^{eq}(b) \text{ and } a \neq b \\ W_{ab} &= \frac{1}{r} \frac{P^{eq}(a)}{P^{eq}(b)} \text{ if } P^{eq}(a) < P^{eq}(b) \end{aligned}$$

where $\frac{1}{r}$ reflects the chance of picking a given final state a out of r possibilities. It is easy to see that these conditions imply detailed balance. In addition normalization is assured by requiring

$$W_{bb} = 1 - \sum_{a \neq b} W_{ab} > 0$$

In practice the stochastic process following from these W 's is easily constructed – compute W_{ab} from some randomly chosen final state b . Compare this with a random number ξ drawn uniformly in the range $0 \rightarrow 1$. If $\xi < W_{ab}$ perform the transition, if not do not change the state. We will explore this algorithm further in a concrete example in the next lecture.

3 Toy Model – 2D Scalar Field Theory

The previous discussion introduced the class of models we will be most interested in – systems with many interacting d.o.f, a lattice structure and a specific action or Hamiltonian specifying the probability distribution of the system. As a first example of such a system consider 2d scalar field theory. Once we go to Euclidean space and introduce a lattice approximation the partition function of the model is given by

$$Z = \prod_{\mathbf{x}} \int d\phi_{\mathbf{x}} e^{-H(\phi)}$$

where

$$H = \sum_{\mathbf{x}} \left(\frac{1}{2} \sum_{\mu=1}^2 (\phi(\mathbf{x}) - \phi(\mathbf{x} + \mu))^2 + \frac{1}{2} m^2 \phi^2(\mathbf{x}) + \lambda \phi^4(\mathbf{x}) \right)$$

where the boldface notation for \mathbf{x} indicates it is a lattice vector in 2d space $\mathbf{x} = a(n_1, n_2)$, $n_1, n_2 = 0, \dots, L-1$ (a is the lattice spacing and L the lattice length). We have scaled the parameters m, λ by powers of a to remove an explicit dependence on the lattice spacing and render them dimensionless. Thus $m = m_{\text{phys}} a$ and $\lambda = \lambda_{\text{phys}} a^2$. To minimize finite size effects we will employ periodic boundary conditions on the fields $\phi(\mathbf{x}) = \phi(\mathbf{x} + \mathbf{L}_{\mathbf{n}, \mathbf{m}})$ where $\mathbf{L}_{\mathbf{n}, \mathbf{m}} = (nLa, mLa)$. Actually it is convenient and conventional to adopt a slightly different set of parameters and write the Hamiltonian (for any dimension D) as

$$H = \sum_{\mathbf{x}} \left(\sum_{\mu} -\kappa \phi(\mathbf{x}) \phi(\mathbf{x} + \mu) + \phi^2(\mathbf{x}) + \lambda' (\phi(\mathbf{x}) - 1)^2 \right)$$

where

$$\begin{aligned} \frac{1}{2} m^2 &= \frac{(1 - 2\lambda')}{2\kappa} - 1 \\ \lambda &= \frac{\lambda'}{\kappa^2} \end{aligned}$$

The limit $\lambda' \rightarrow \infty$ is particularly interesting as it forces ϕ to take on one of two discrete values $+1$ or -1 . This is the origin of its use to describe the famous Ising model.

3.1 Metropolis algorithm for 2D scalar field model

To simulate this model we need to provide a transition matrix which, when applied to any arbitrary initial configuration (set of ϕ values on lattice) will ultimately generate a

distribution of configurations which matches the e^{-H} factor in the partition function. We have already described, in general terms, perhaps the simplest way to do this - the Metropolis method. In practice this algorithm is almost always employed in a series of small steps. Each individual step comprises the attempted update of the value of the field at a single lattice site. Specializing our earlier general Metropolis algorithm to the specific case $P^{eq}(a) = e^{-H(a)}$ we can choose W_{ab} governing the probability of updating the field at this site from some initial value b to a final value a to be

$$W_{ab} = \frac{1}{r} \text{ if } H(a) < H(b) \text{ and } a \neq b$$

$$W_{ab} = \frac{1}{r} e^{-(H(a)-H(b))} \text{ if } H(a) > H(b)$$

Notice there is a nice physical interpretation of this algorithm. We make moves which generally speaking take the system to lower values of H . However every so often the noise in the system allows us to fluctuate to larger values of H . These entropy increasing transitions compete with the internal energy lowering moves to find a minimum of the free energy of the system - at which point we are in thermal equilibrium. We have yet to specify how to choose $Q(a)$ the distribution governing how we select the trial configuration a . The simplest version of the Metropolis algorithm assumes we are choosing the final state randomly and uniformly from some fixed distribution centered about zero. A simple way to do this is to generate a uniform random number η between zero and one, subtract 0.5 and rescale it by some factor ϵ . Thus the Metropolis algorithm will involve the following steps

1. Pick a site \mathbf{x} (at random or sequentially)
2. Pick a new trial value for the field at that site $\phi(\mathbf{x}) \rightarrow \phi^{\text{trial}} = \phi(\mathbf{x}) + \epsilon(\eta - \frac{1}{2})$
3. Compute change in Hamiltonian under this trial update $\Delta H = H(\phi^{\text{trial}}(\mathbf{x})) - H(\phi(\mathbf{x}))$
4. The Metropolis test. Generate another random number η' . If $\eta' < e^{-\Delta H}$ accept the trial update. Otherwise do nothing to the field at that site.
5. Go back to step 1 and examine another site.

Notice that, most importantly, the computation of ΔH requires only *local* information - the values of field variables neighbor to the site in question. This reflects the local nature of the microscopic theory and allows the update of a single degree of freedom to require $O(1)$ arithmetic operations. This is crucial for the success of this algorithm. There are physical models where the Hamiltonian is *not* local (eg. relativistic fermions) and we have to work much harder to come up with useful simulation algorithms.

A Metropolis sweep is defined as the attempted update of the field at every single site in the lattice. Typically many thousands of sweeps are needed to make good measurements of physical observables. Also, we need to apply this algorithm many times before it will generate the correct distribution e^{-H} . Hence, we typically discard a certain number of *thermalization sweeps* before taking measurements. The number required can be assessed in a variety of ways which we will describe later.

3.2 Measuring observables

The Markov procedure, if properly constructed, will automatically generate a distribution of field configuration matching the target equilibrium distribution e^{-H} . This makes the computation of an arbitrary observable very straightforward –

$$\langle O \rangle = \lim_{N_c \rightarrow \infty} \frac{1}{N_c} \sum_{i=1}^{N_c} O(i)$$

where $O(i)$ denotes the observable calculated on configuration i . Points to note:

- O can be anything – eg products of fields at different points (correlators of any size).
- This result must only be used after *thermalization* – see earlier.
- Typically successive configurations generated according to a Markov procedure are *correlated*. That is the new configuration is **not** drawn truly independently from the distribution e^{-H} . However this correlation decreases with successive updates and we typically consider the algorithm as generating independent configurations after a certain number τ_A sweeps. This number is called the *autocorrelation time* of the algorithm. Strictly this number can be a function of the observable also – but we usually mean by τ_A the autocorrelation time of the *slowest* observable in the system. Because of this we usually only make measurements every $N_m \sim \tau_A$ sweeps. Since all sweeps which do not yield measurements are in some sense wasted we can use the value of τ_A as one criteria of how effective a particular algorithm is when compared to other algorithms.
- This method of measurement is stochastic – it yields an *estimator* for the expectation value which only equals it in the limit of an infinite sample size N_c . For finite N_c it suffers from statistical errors. We will show that these decrease like $\frac{1}{\sqrt{N_c}}$

This combination of Markov updating procedure plus stochastic estimation of observable realized through a computer code I will call *Monte Carlo* simulation. As you will see, it is a very powerful technique for studying many diverse physical systems. It is used not only for quantitative work (computing specific numbers) but also for elucidating the underlying phase structure of a model, uncovering universal behaviors and discovering new long distance features of a given microscopic theory. It can be applied to strong coupling physics yielding answers to questions as diverse as “does QCD confine quarks inside hadrons”, “can we understand high- T_c superconductivity as a result of strongly interacting electrons”, “can the Universe evolve the observed large scale structure from a given gravitational dynamics” etc etc.

3.3 Coding the Simulation

Lets turn now to a practical implementation of these ideas. We will use C++ to write a simulation program to implement this simplest of Monte Carlo algorithms. First a little aside on why C++ and why the program will be organized in a particular way.

3.3.1 General comments

Since I want to go on to discuss more complicated algorithms and models I want to write a program which can be edited without changing many of the higher level functions and program organization. Thus I would like to specify my microscopic degrees of freedom and their interactions in one part of the code only and leave many other parts of the code ignorant as to those details. This allows us to edit the code without too much effort if I change the nature of the basic variables (and reduces bugs induced by such changes). C++ is a good programming language for scientific programming precisely because it allows us to create data hierarchies of this kind. We can define abstract data types which match precisely to our microscopic degrees of freedom and even code the natural mathematical operations which act on those variables in such a way as to render the resulting high level code as similar as possible to the corresponding mathematical expressions. For example, in the simplest case my variables are real numbers and the natural operations are addition and multiplication (needed to compute the Hamiltonian). In the code I use symbols `+` and `*` to represent these operations. However, someday I might want to consider models with complex fields. We can create an abstract complex data type to encode the variables, together with our definitions of complex addition and multiplication. Furthermore, C++ allows us to redefine the symbols `+` and `*` to mean these latter complex operations when operating on complex data. This allows a lot of the code to compute the Hamiltonian of this new model to remain unchanged when I go from the real to the complex case. This technique really starts to yield payoffs if I consider more complex systems, say where the basic variables are vectors or matrices. In the latter case we can define a matrix type (real or complex) and redefine (in C++ speak: *overload*) `+` and `*` to represent matrix addition and multiplication. Indeed, we can create data types to represent the entire set of microscopic field variables and overload the `+` symbol to say represent a global operation in which two entire fields are added together. So in the code you will see an object of type `ScalarField`. You should begin to see the power of the method.

In one final example, let me describe how I will represent the structure of the lattice itself. I will introduce an abstract data type called `LatticeVector` which will code an object of type vector of integers specifying the coordinates (in units of the lattice spacing) of all points in the lattice. I can code things so that this works for arbitrary dimension. Furthermore, in order to access field variables at neighboring sites (as needed in the kinetic piece in the Hamiltonian for example) we can define addition of two lattice vectors as resulting in another lattice vector whose components are the sums of the two input vectors. The real power of this method is that the periodic nature of the lattice can be concealed inside this `+` operation. Inside the function I can test whether any of the resultant vector components exceed $L - 1$ and if so map them back into the range $0 \rightarrow L - 1$ by subtracting appropriate multiples of L . This implements the toroidal boundary conditions without the use of multiple `if` statements in higher levels of the code. In the code you will also see that I can use the `LatticeVector` type to index the `ScalarField` object. Thus the mathematical expression

$$\phi(\mathbf{x})\phi(\mathbf{x} + \mu)$$

may be represented in the code as

```
phi.get(x) * phi.get(x + e_mu)
```

where `phi` is of type `ScalarField`, `x` and `e_mu` are both of type `LatticeVector` (with e_μ a unit vector in the μ direction) and `*` representing multiplication. The definitions of these basic data types and the operation upon them are all coded in the function `utilities.cpp`. The header file `utilities.h` is then included in any other function which uses those variables.

3.3.2 Creating abstract data types and operators

Let us examine now how these abstract data types are created. Inside `utilities.h` you will see some code which serves to define the abstract data type `LatticeVector`.

```
class LatticeVector{
private:
int coords[D];
public:
LatticeVector(void);
LatticeVector(int);
void set(int,int);
int get(int) const;
void print(void) const;
};
```

The object contains `D` pieces of data – the integer lattice coordinates of a point. These pieces of data are `private` – they cannot be accessed by any other C++ function directly. Indeed the way data is passed into and out of this abstract data object is via the `public` functions (methods) `set()` and `get()`. For example,

```
LatticeVector x;
x.set(0,2);
```

This piece of code sets the 0th component of the vector to 2. Similarly `y=x.get(2)` gets the second component of the `LatticeVector x` and assigns it to the `int` variable `y`. Finally, the first two functions within this class are called `constructor` functions. They are called when the variable is first created. Notice there are two types of constructor for `LatticeVector` (there may be as many as we wish to define). We will see presently that the first constructor creates the variable and assigns its private data to zeroes. The second takes an integer argument and creates a unit vector along the direction specified by the input integer argument. Notice all the code in `utilities.h` specifies the type of data and (some) of functions that operate on that data, but do *not* tell you what those operators do in detail. the code to *implement* the constructors, and the `set()` and `get()` functions is contained in the file `utilities.cpp`. `utilities.h` is a *header* file used to convey data and function definitions to other functions that want to use the basic (user defined) data types. Looking at

`utilities.cpp` we see this code in detail. Beside the functions we have discussed you should see pieces of code which tell the C++ compiler how to deal with addition or subtraction of objects of type `LatticeVector`. For example code for the `+` operator is given below

```
LatticeVector operator +(const LatticeVector &x, const LatticeVector &y){
LatticeVector dum;
for(int i=0;i<D;i++){
dum.set(i,(x.get(i)+y.get(i))%L);}
return(dum);}
```

You should read this in the following way: first `+` takes two arguments of type `LatticeVector` and produces another object of type `LatticeVector`. Inside the function this operation is implemented via the loop over `i` running over all components. Notice the operation `%L` acting on the sum. This is the C++ modulo operator which adds the numbers modulo `L`. This is how the periodic boundary conditions are incorporated! Finally the dummy argument `dum` of type `LatticeVector` is returned to the calling environment. A final point to note: the arguments `const LatticeVector &x` allows the C++ compiler to pass just the address in memory of the object rather than the object itself. This is a great optimization for large data types which otherwise would need to be explicitly copied around between functions. Let us now turn to the code which implements the higher level functions. This is housed (conventionally) in separate C++ source files – one for each C++ function.

3.3.3 `main()` function

The code contains a `main()` function (inside `simulation.cpp`) which is where execution starts and which controls the overall logical flow of the program. Here, the field variable `phi` is created. Subsequently, `main()` calls the function `ReadParameters()` to input simulation parameters like the total number of sweeps `SWEEPS` etc. It then starts the simulation loop with updates being performed by the function `UpdateMetropolis()`. The latter takes a single *argument* `phi` the current field configuration. `main` also calls the measurement function `measure()` every `GAP` sweeps.

3.3.4 `ReadParameters()`

If you look at this you will see how to do simple I/O in C++ including opening a data file (called `parameters` here).

3.3.5 `UpdateMetropolis()`

The bulk of the code in this function consists of a loop over all lattice sites. Notice that this is achieved by means of a call to the function `SumOverLattice()` which takes a `LatticeVector` argument holding the current site to be updated. Each call of `SumOverLattice()` returns a new lattice point coordinate until all have been exhausted. This routine is defined in `utilities.cpp` and works on a lattice of arbitrary size in arbitrary dimension (these are

defined in `utilities.h`. Hopefully the code implementing the calculation of the change in the Hamiltonian is pretty clear. Notice that the parameters `EPS,KAPPA` and `LAMBDA` are global variables – defined inside `simulation.cpp` (although **not** inside `main()`) with values allocated inside of `ReadParameters()`. The function `RandomNumber()` returns a uniformly distributed (pseudo)-random number on the interval $0 \rightarrow 1$. Notice the *member* function `set()` is used to change the value of `phi` at position `x`. Finally notice the use of `static` variables (which retain their values on function exit) to track the number of accepted updates performed. Similar static variables are used in `Measure()` to let the program know when to open the data file used for results.

3.3.6 Measure()

Note again use of C++ I/O routines. We use similar code to `MetropolisUpdate()` to compute the Hamiltonian and average value of the field. The static integer variable `firsttime` allows us to open the data file just the first time the function is called.

3.3.7 Compiling and Running the Code

Under Unix go to the directory containing the source code (both `.cpp` and `.h` files) and just type (at the command prompt)

```
g++ -O *.cpp
```

This will produce a binary executable file called `a.out`. If you want to run this, time the run and write output messages to a log file type

```
(time ./a.out) >& log &
```

The final ampersand ensures the code will run "in the background" which means that you are presented with another prompt allowing you to do other things while the code executes. In Windows, fire up your C++ IDE, load the files as some sort of project and hit compile or build. This will usually compile the code and then automatically run it. You will probably need to make it run within a win32 type console for compatibility with unix.

4 Errors

These are assessed from a powerful result in statistics called the Central Limit Theorem. Consider a set of N *independent* stochastic variables Q_i (eg. Monte Carlo measurements of some quantity). These can be drawn from an (almost) arbitrary distribution (it must have a finite variance $\sigma^2 = \langle f^2 \rangle - \langle f \rangle^2$). The central limit theorem states that their average $\frac{1}{N} \sum_{i=1}^N Q_i$ is a Gaussian distribution with variance $\frac{1}{N} \sigma^2$. The width of this Gaussian distribution reflects the statistical uncertainty or error in the Monte Carlo estimator. Thus

if all our measurements were truly independent (the autocorrelation time τ_A were zero), we could compute all our errors by simply using the formula

$$\delta Q = \frac{1}{\sqrt{N}} \sqrt{\langle Q^2 \rangle - \langle Q \rangle^2}$$

where $\langle Q \rangle = \frac{1}{N} \sum_{i=1}^N Q_i$. Unfortunately our Monte Carlo sequences are correlated. Instead of N independent measurements we only have N/τ_A . If we knew τ_A we could correct for this in the above formula by essentially multiplying the naive error by $\sqrt{\tau_A}$. Unfortunately we don't usually know τ_A very well and we must find another method. There are many techniques that can be used to assess good statistical errors in Monte Carlo data. I will just describe the simplest which will be more than good enough for our purposes. This is called the **binning** method.

4.1 Binning

Consider a set of 2^P measurements of some quantity $Q_i, i = 1 \dots 2^P$. Compute the naive error estimate for this quantity using the above formula with $N = 2^P$. Call this δQ^0 . Now form a new data set from the old one by averaging the original measurements in consecutive pairs (eg. new Q_1 will be average of old Q_1 and Q_2 , new Q_2 will be average of old Q_3 and Q_4). Thus the new data set will contain 2^{P-1} elements. Compute the new naive errors on this set and call them δQ^1 . Now average the new data set on consecutive pairs to produce a set with 2^{P-2} elements and a new error estimate δQ^2 . Continue in this way until you reach a data set with say just 16 elements and stop. Now make a plot of the error estimate δQ^k against the level number k (set length 2^{P-k}). If all is well you should see the error estimate rise to plateau (approximately) near some value. The plateau value is the true error (and the ratio of naive error when $k = 0$ to true error is a measure of $\sqrt{\tau_a}$). It should be clear why this is so; as we continue to average or *bin* the data eventually our bin samples will span data from more than one autocorrelation time which will ensure it is uncorrelated and the naive formula will apply. As a bonus we learn something about the magnitude of the autocorrelation time for this quantity!

5 Critical Phenomena

Now we have a working Monte Carlo code for a physically interesting model and know how to measure observables and give meaningful error estimates we will spend a little time exploring some general features of it. In practice we can fix the coupling λ and analyze the model as a function of the hopping parameter κ . We will focus on 4 primary observables

- Mean Energy $E = \langle H \rangle$ (or the density $h = E/N$)
- Specific Heat. This is just the fluctuations in the mean H per unit volume $C = \frac{1}{N} (\langle H^2 \rangle - \langle H \rangle^2)$

- Average field $M = \sum_x \phi(x)$ (or the mean value per site $m = M/N$)
- Susceptibility $\chi = \frac{1}{N} \sum_x (\langle M^2 \rangle - \langle M \rangle^2)$

Let us examine these quantities as a function of κ , initially for a small lattice 4×4 and at fixed coupling $\lambda = 0.5$. The figures (see links from lecture page) reveal interesting structure in these quantities around $\kappa \sim 0.6$. For small κ the average field is close to zero while for large field it appears to take on non-zero values. Since the behavior of the system exhibits qualitatively different features in the low and high κ regimes we speak of them as different *phases* of the system. Clearly the average value of the field serves as an *order parameter* to distinguish these two regimes of the model. We also note that the fluctuations in the average field are small in either *phase* but exhibit a strong peak structure near this critical value of κ . This peak is mirrored in the specific heat also. At $\kappa = \kappa_c \sim 0.6$ we talk of the system undergoing a *phase transition*. Typically at such a critical point (and strictly on an infinite system) a variety of physical observable will exhibit (usually) power law divergences

$$Q(\kappa) \sim (\kappa - \kappa_c)^x$$

where x is called a *critical exponent*. These divergences are intimately linked to the famous U.V divergences of quantum field theories and the framework needed to understand and manipulate them is called the renormalization group or RG. In the language of QFT the critical exponents are called *anomalous dimensions*. One output from the RG treatment is an approximate calculation of the critical exponents. These turn out to have universal features – they do not depend on the details of the lattice, the value of λ or even whether the microscopic variables are discrete or continuous valued (eg they are the same in the Ising limit $\lambda \rightarrow \infty$). Typically they depend only on symmetries of the theory, the dimensionality and perhaps also on the details of the couplings to a set of so-called *relevant interactions*. This universality is very attractive to physicists since it renders certain predictions of these theories insensitive to details of the model. It is also crucial to the problem of taking $a \rightarrow 0$ in the QFT case. To do this we will see that we need to tune to this critical region of the parameter space where $\kappa \rightarrow \kappa_c$.

However, in our case we can bypass most of the technicalities associated with the RG and try to determine the critical exponents by fitting the numerical data for say the susceptibility to a power form in the vicinity of κ_c . In practice they are most often derived not by this way but by a technique called *finite size scaling*. In essence one replaces the divergence in κ with a divergence with (linear) system size L

$$\langle Q \rangle \sim L^{\frac{x}{\nu}}$$

The parameter ν is another critical exponent – the so-called **correlation length critical exponent** governing the dependence of the *correlation length* ξ on κ

$$\xi \sim |\kappa - \kappa_c|^{-\nu}$$

The correlation length is defined from the behavior of the two-point correlation function of the fields

$$\langle \phi(0)\phi(x) \rangle \sim e^{-|x|/\xi} |x|^{-\eta+2-d}$$

See pictures of correlator at different κ 's. Notice the periodicity of the correlator – this is a consequence of the periodic boundary conditions. In general the correlator of a scalar field on a finite lattice of length L is (neglecting power corrections)

$$\langle \phi(0)\phi(x) \rangle \sim \cosh \frac{(x - L/2)}{\xi}$$

In QFT it corresponds to the (inverse) mass of lightest particle state created by the field and η is the anomalous dimension of the field. For a free field $\eta = 0$. Inverting the above relationship to give $\delta\kappa$ as a function of ξ and substituting into our earlier relation leads to the result

$$Q \sim \xi^{\frac{x}{\nu}}$$

Finally (as will shortly see explicitly) the correlation length grows as $\kappa \rightarrow \kappa_c$. This growth is ultimately limited only by the lattice length L and we can set $\xi \sim L$ for $\kappa \sim \kappa_c$. In this way we arrive at the finite size scaling result. Thus by measuring the L dependence of the height of the susceptibility peak we can determine the ratio x/ν . We will try this out. In the case of the specific heat the critical exponent x is usually called α while the corresponding quantity in the case of the susceptibility is named γ i.e $\chi \sim L^{\frac{\gamma}{\nu}}$.

Notice also the following relationship between the integral of the two point function and the susceptibility

$$\chi = \frac{1}{N} \left(\sum_{x,y} \langle \phi(y)\phi(x) \rangle - \langle \phi(x) \rangle \langle \phi(y) \rangle \right)$$

Thus we expect

$$\chi = \sum_x \langle \phi(0)\phi(x) \rangle_c$$

Using the above analytic form for the two point function and replacing the sum by an integral cut-off at the correlation length leads to

$$\chi = \xi^{2-\eta}$$

Comparing this with our earlier result $\chi = \xi^{\gamma/\nu}$ leads to a relationship ship between critical exponents

$$\nu(2 - \eta) = \gamma$$

5.1 Symmetry Breaking

We have seen that the two phases of the model are characterized by zero and non-zero values for the mean field. Actually this is wrong. If we were to run our simulations much

longer we would find that the mean field was always zero independent of κ . This is simply a consequence of the $\phi \rightarrow -\phi$ symmetry of the Hamiltonian in **finite volume**. The fact that this is not seen reveals that our algorithm is rather poor at large κ . In this region all the fields are highly correlated and to realize $\langle \phi \rangle = 0$ requires coherent flipping of the sign of all field variables simultaneously. Our local algorithm is very inefficient at generating such changes which is why the system seems to freeze in a single state. We will return to this issue when we discuss improved algorithms. Actually true symmetry breaking of continuous variable systems can only take place in *infinite volume* i.e the thermodynamic limit. Actually even this statement can only be made in dimensions greater than 2 (for **continuous valued fields**). This is the Mermin-Wagner theorem. Let us see how this goes. Formally global symmetry breaking is defined from the behavior of the following expectation value

$$\langle \phi(0) \rangle = \lim_{V \rightarrow \infty} \lim_{H \rightarrow 0} \int D\phi \phi(0) e^{-S(\phi) + H \int \phi}$$

where H is a (small) external field. Consider the change of variables $\phi \rightarrow \phi' = -\phi$. This expression may be rewritten as

$$\langle \phi \rangle = \lim_{V \rightarrow \infty} \lim_{H \rightarrow 0} \int D\phi \phi'(0) e^{-S(\phi') - H \int \phi'}$$

where we have assumed S is invariant under $\phi \rightarrow -\phi$. Expanding the exponentials to leading order in H and adding these two expressions leads to

$$\langle \phi \rangle = \lim_{V \rightarrow \infty} \lim_{H \rightarrow 0} H \langle \int dx \phi(0) \phi(x) \rangle$$

On a finite lattice the expectation value is some finite number and $\langle \phi \rangle \rightarrow 0$ as $H \rightarrow 0$ corresponding to the absence of symmetry breaking. However for $d > 2$ we expect that $\int \langle \phi(0) \phi(x) \rangle$ diverges as a (fractional) power of the volume V^p . We can then take a limit in which $V^p H$ is held fixed as $H \rightarrow 0$ and $V \rightarrow \infty$. In such a limit $\langle \phi(0) \rangle \neq 0$ and spontaneous symmetry breaking occurs. In $d = 2$ the situation is more subtle since the expectation value is not extensive varying only as the log of the system volume. This means that $\langle \phi(0) \rangle = 0$ once again. This log behavior reflects the *I.R* divergence of the free scalar correlation function in 2d. Notice the 2d correlator we have studied so far is an effective 1d correlation function obtained by averaging over one direction transverse to the position space argument x . In any dimension this is an exponential (cosh with periodic boundary conditions) function.

Modulo these caveats about the absence of true symmetry breaking in two dimensions we can nevertheless get a handle rather easily on how symmetry breaking is realized in the Hamiltonian. We have seen (homework 1.) that the κ parameter is related to the usual mass parameter m^2 via the relation

$$\frac{1}{2} m^2 = \frac{(1 - 2\lambda')}{\kappa} - 1$$

If we assume the ground state configuration is a constant we can attempt to find it (classically) by minimizing the Hamiltonian. For positive m^2 (small κ) this occurs for $\phi = 0$. But

at large κ the mass term can become negative leading to $\phi \neq 0$. There are two possible values differing only in their sign. The system in infinite volume will pick one at random and the symmetry will be broken. For $\lambda = 0.5$ we see that $\kappa_c = 0$. Why then do we observe $\kappa_c \sim 0.6$? The answer is that we have ignored fluctuations in this analysis. These *renormalize* the values of the parameters in the Hamiltonian and hence change the critical κ_c .

6 Non-linear fitting

We have seen that an important quantity the correlation length ξ is buried inside the two point function. We would like to know how to extract this from the MC data. This brings us to the subject of model fitting. That is given a theoretical model for some quantity eg.

$$\langle \phi(0)\phi(x) \rangle = A \cosh\left(\frac{x - L/2}{\xi}\right)$$

and some Monte Carlo estimate for this correlator (plus errors), how can we extract the “best fit” value of ξ (together with an error) and also some measure of “goodness of fit”. That is we want to know whether the theoretical functional form is a good way of summarizing the data. If not it may point to a problem with our theory. This is a big subject and we will only touch on the simplest aspect of this – how to fit statistical data to some theoretical curve where the model is some (in general) non-linear function of the fitting parameters (eg. A and ξ in the example above). This is accomplished by first constructing a goodness-of-fit function or χ^2 -function given by

$$\chi^2(a) = \sum_{i=1}^N \left[\frac{y_i - y(x_i, a)}{\sigma_i} \right]^2$$

where the data is some discretely sampled curve $\{y_i\}$ with N points and the theoretical model is given by $y(x, a)$ where a represents a set of fitting parameters $a \equiv a_j, j = 1 \dots P$. Notice the errors σ_i in the denominator. Poorly determined points weigh less than well-determined points in the fit.

An optimal fit is given by setting $\frac{\partial \chi^2}{\partial a_j} = 0$ Now let us suppose that close to the minimum of $\chi^2(a^*)$ we can treat the merit function as a quadratic function of the parameters

$$\chi^2(a^* - a) = \gamma \frac{\partial \chi^2}{\partial a_j} \Big|_{a^*} (a^* - a)_j + \frac{1}{2} (a^* - a)_i \frac{\partial^2 \chi^2}{\partial a_j \partial a_k} \Big|_{a^*} (a^* - a)_j (a^* - a)_k$$

If we are close to the optimal set a^* we can use this formula to jump to the optimal set in one go

$$a_i^* = a_i - H_{ij}^{-1} d_j$$

where d is the gradient of χ^2 with respect to the parameters and H , the Hessian, is the second derivative of χ^2 . If we are not close enough to the minimum for this formula to be good we can always choose to lower the χ^2 by simply performing a “gradient descent” step

$$a_i^{\text{next}} = a_i^{\text{current}} - \text{constant} \times d_i$$

These two updates to the parameters can be use to find the optimal fit. In detail they give

$$d_k = -2 \sum_{i=1}^N \frac{[y_i - y(x_i, a)]}{\sigma_i^2} \frac{\partial y(x_i, a)}{\partial a_k}$$

$$H_{kl} = 2 \sum_{i=1}^N \frac{1}{\sigma_i^2} \left[\frac{\partial y(x_i, a)}{\partial a_k} \frac{\partial y(x_i, a)}{\partial a_l} + X \right]$$

where X represents 2nd derivative pieces which will be neglected. In practice we combine the two types of update to solve for the optimal parameters by iteratively solving the equation

$$\sum_{j=1}^P H'_{ij} \delta a_j = d_i \quad (1)$$

where $H'_{ii} = H_{ii}(1 + \lambda)$. If $\lambda \rightarrow \infty$ we get the steepest descent update. If $\lambda \rightarrow 0$ we have the one step algorithm. By changing the value of λ we can thus interpolate between them. The algorithm runs as follows

1. Compute $\chi^2(a)$ for some initial guess
2. Set $\lambda = 0.001$ say
3. Solve eqn. 1 for new a 's and evaluate $\chi^2(a + \delta a)$
4. If $\chi^2(a + \delta a) \geq \chi^2(a)$, increase λ by say factor of 10 and go back to solver again.
5. If $\chi^2(a + \delta a) < \chi^2(a)$ decrease λ by factor of 10, update a and go back to solver stage.

Stop when percentage change in parameters is small. The errors in the fit parameters are given by the diagonal elements in the inverse Hessian H^{-1} . See C code (corrfit.c). Rule of thumb. For *uncorrelated* data a $\chi^2 \sim O(1)$ implies theoretical model is good. For correlated data it may be much smaller since errors are effectively overestimates of the amount of point to point fluctuation. There are techniques to avoid this and produce a sensible χ^2 but they will take us too long to discuss. In practice the fitted parameters themselves are given well by this approach although not necessarily the value of χ^2 .

7 Critical Slowing Down

If we using the binning code to estimate the autocorrelation time τ_A for the average field as we vary κ we notice that τ_A increases as $\kappa \rightarrow \kappa_c$. This parallels the increase in the correlation length ξ . Indeed we can observe that

$$\tau_A \sim \xi^z$$

where z is called the **dynamic critical exponent**. For *local* algorithms like Metropolis we will typically find $z \sim 2$. One hand waving way to see why this is true goes as follows.

Consider updating some site. Other sites that are many lattice spacings away from the site in question do not immediately know that the field has changed. In fact it takes a certain of updates before that information propagates across the lattice. This information propagates as a random walk. To generate a statistically independent configuration requires that the information propagate a distance ξ away so that all correlated fields feel the effect of the update. The random walk character guarantees that this will take a time proportional to the square of the distance hence $\tau \sim \xi^2$. Thus as we approach the phase transition the algorithm gets less and less effective at generating new configurations. This is termed **critical slowing down**. What we would like is to come up with different algorithms with smaller z 's. The ideal situation $z = 0$ is almost realized with so-called *cluster algorithms* which we will discuss later. Unfortunately these very efficient algorithms are not generally applicable to all cases and so we will start by considering other local algorithms which while they may not reduce z as much are nevertheless are interest in practical situations.

7.1 Heatbath

Let us return to the detailed balance condition

$$W_{ab}P_b^{eq} = W_{ba}P_a^{eq}$$

One particularly simple solution puts

$$W_{ab} \sim P_a^{eq}$$

we thus pick a final state according to its equilibrium probability independent of its current state. This seems easy – the difficulty is normalizing this factor to get a true probability. In the context of our scalar field model the following algorithm implements this heat bath method

- Pick a site y to update
- Thinking of all other fields as fixed compute the probability for the field to take a value $\phi(y)$

$$p(\phi) = \frac{1}{\int d\phi(x) e^{-H(\phi(x))}} e^{-H(\phi(y))}$$

- Generate the new field $\phi(y)$ according to this distribution

In practice the ability to compute the integral in the denominator severely limits this method *except for discrete systems* where it amounts to a series of finite sums for all configurations of the neighbor variables which may computed once and for all and the possible values tabulated in a lookup table. One other case which may handled in our model corresponds to $\lambda = 0$. In this case the Hamiltonian looks like

$$-\kappa\phi(y) \sum_{\text{neighbors } x} \phi(x) + \phi^2(y)$$

Completing the square lead to

$$\left(\phi(y) - \frac{1}{2}\kappa \sum_x \phi(x)\right)^2$$

This is gaussian distribution with mean $\frac{\kappa}{2} \sum_x \phi(x)$ and variance $1/2$. This distribution can be normalized and can be easily simulated and leads to an efficient way of simulating the model for $\lambda = 0$. In detail

$$\phi(y) = \frac{1}{2}\kappa \sum_x \phi(x) + \hat{\eta}$$

where $\hat{\eta}$ is a gaussian distributed variable with width $1/2$.

7.2 Overrelaxation

This is not strictly a stochastic update but may be combined with Metropolis and heatbath moves to generate an efficient method. The idea (which is best implemented in the $\lambda = 0$ model again) relies on making an update which in some sense moves a maximal distance in the local configuration space while not changing the action. In the context of the previous Hamiltonian it corresponds to the reflection operation

$$\phi(y) \rightarrow \kappa \sum_x \phi(x) - \phi(y)$$

Clearly H is same after this move but the field variable has changed significantly. For certain spin models the use of this technique can reduce z to 1.

7.3 Cluster Methods

This class of algorithm is rather different than those based on local updating. They are most often used for spin models but can as we will see be used in combination with Metropolis to produce efficient methods for continuous field systems. Let us examine this case in some detail. Let us rewrite the Hamiltonian in the following fashion

$$H = \sum_x -\kappa s(x) |\phi(x)| \sum_{\mu} s(x + \mu) |\phi(x + \mu)| + \phi^2(x) + \lambda(\phi^2(x) - 1)^2$$

where s_x is the sign of the field at x . We now imagine freezing the magnitude of the field at x i.e $|\phi(x)|$ and concentrate on updating the sign field $s(x)$

$$H = \sum_x - \sum_{\mu} \kappa |\phi(x)\phi(x + \mu)| s(x)s(x + \mu)$$

This looks like a Ising model with locally varying couplings $\beta_{x,x+\mu} = \kappa |\phi(x)\phi(x + \mu)|$. To perform a cluster update of these spin degrees of freedom we do the following

1. Pick a site x at random.

2. Draw bonds to all neighbors y of same sign s with probability $1 - e^{-2\beta_{xy}}$
3. If bonds have been drawn to any nearest neighbor site j , draw bonds to all nearest neighbors z of site y with probability $1 - e^{-\beta_{yz}}$
4. Repeat last step until no new bonds are created.
5. Flip all spins in the cluster $s \rightarrow -s$.
6. Go back to first step and pick another site.

The proof that this satisfies detailed balance will be given later. Notice it will be very efficient in flipping the sign of the field at large κ i.e deep in the broken symmetry phase. See pictures for average field at large κ .

This cluster flipping technique works well in the case of spin models. These models play an important role in condensed matter/magnetic systems. The simplest type of continuous spin model consists of a field ϕ^i carrying an additional index i and satisfying the constraint $\sum_{i=1}^N \phi^i(x)\phi^i(x) = 1$. The field is thus like a N -dimensional unit vector. The action for such a model is invariant under $O(N)$ rotations in this internal field space and typically takes the form

$$S = \sum_{\langle xy \rangle} \phi^i(x)\phi^i(y) = \sum_{\langle xy \rangle} \phi(x) \cdot \phi(y)$$

To implement a cluster update for such a system we need to generalize the notion of spin flip. The concept of spin flip is replaced by a reflection in the plane orthogonal to some randomly chosen unit vector r .

$$R(r)\phi(x) = \phi(x) - 2(\phi(x) \cdot r)r$$

Clearly $R^2 = I$ and the action $\sum_{\langle xy \rangle} \phi(x) \cdot \phi(y)$ is invariant under R .

The cluster algorithm now proceeds as follows.

1. Choose a direction r at random. Also choose a site at random as the first point x of a cluster. Mark x . Flip $\phi(x) \rightarrow R(r)\phi(x)$
2. Examine every link from x . Draw in a bond with probability

$$\begin{aligned} p(\phi(x), \phi(y)) &= 1 - e^{\min(0, \kappa\phi(x) \cdot [1 - R(r)]\phi(y))} \\ p(\phi(x), \phi(y)) &= 1 - e^{\min(0, 2\kappa(\phi(x) \cdot r)(\phi(y) \cdot r))} \end{aligned}$$

3. If the bond occurs add mark y , flip it and add it to the cluster.
4. Proceed iteratively activating bonds according to this probabilistic rule to *unmarked* sites, subsequently marking them and adding them to the cluster.
5. Stop when no new unmarked sites can be reached.

Notice for $N = 1$ we get back the Ising model while for $N = 2$ we obtain the XY-model and for $N = 3$ an $O(3)$ vector model. One can compute the autocorrelation time for all these models near their critical points and verify that the dynamic critical exponent z is actually reduced to values close to zero! We remark in passing that the XY-model is particularly interesting as it possesses non-power law critical behavior, the correlation length diverging exponentially as we approach the critical point. This model also possesses topologically non-trivial configurations of the fields called vortices which bind together in vortex-antivortex pairs at low temperature. Indeed the transition can be interpreted as an unbinding of these bound pairs as the temperature is raised.

7.3.1 Proof of detailed balance

Consider two configurations $\{\phi(x)\}$ and $\{\phi'(x)\}$ that differ by a flip R on a cluster c . The transition probabilities obey

$$\frac{W(\{\phi(x)\} \rightarrow \{\phi'(x)\})}{W(\{\phi'(x)\} \rightarrow \{\phi(x)\})} = \prod_{xy \text{ in } \delta c} \frac{1 - p(R\phi(x), \phi(y))}{1 - p(R\phi'(x), \phi'(y))}$$

This latter expression can be rewritten

$$\exp \left(\kappa \sum_{xy \text{ in } \delta c} (\phi'(x)\phi'(y) - \phi(x)\phi(y)) \right)$$

where δc denotes sites on the boundary of the cluster c (i.e all links in which x is in c but y is not). To grow a specific cluster c its boundary links must *not* be activated which accounts for the factors $1 - p$ in this expression.

7.4 Hybrid Monte Carlo

There is another algorithm that can be generally used to implement global updates on field configurations and which can reduce critical slowing down substantially. The basic idea is to imagine that the field variables $\phi(x)$ are promoted to be functions of a new (continuous) time variable t i.e $\phi(x) \rightarrow \phi(x, t)$. We also introduce a conjugate momentum variable $p(x, t)$ and write a Hamiltonian

$$H = S(\phi) + \frac{1}{2}p(x)^2$$

Notice that to avoid confusion we have called our original Hamiltonian function S rather than H . Since we have a Hamiltonian we can imagine evolving the system in time t according to Hamilton's classical equations

$$\begin{aligned} \frac{\partial \phi(x)}{\partial t} &= \frac{\partial H}{\partial p} = p(x, t) \\ \frac{\partial p(x)}{\partial t} &= -\frac{\partial H}{\partial \phi} = -\frac{\partial S}{\partial \phi} \end{aligned}$$

In practice we integrate these equations by first replacing them by a discrete time step approximation $t \rightarrow t_n = n\Delta t$. The simplest (called Euler) is just

$$\begin{aligned}\phi_{n+1} &= \phi_n + \Delta t p_n \\ p_{n+1} &= p_n - \Delta t \left(\frac{\partial S}{\partial \phi} \right)_n\end{aligned}$$

Thus, knowing the initial values of ϕ and p we can evolve the fields in time along a trajectory of approximately constant H . In practice we use a leapfrog method to do this evolution

$$\begin{aligned}\phi_{n+1} &= \phi_n + \Delta t p_n + \frac{(\Delta t)^2}{2} f_n \\ p_{n+1} &= p_n + \frac{\Delta t}{2} (f_n + f_{n+1})\end{aligned}$$

where $f_n = -\frac{\partial S}{\partial \phi_n}$ is the force. This is accurate to $O(\Delta t^2)$ (as opposed to Euler which is only good to $O(\Delta t)$). Moreover it satisfies a reversibility condition given by

$$(\phi(T), -p(T)) \rightarrow (\phi(0), -p(0))$$

where $\phi(T)$ denotes the field after T time steps and the above condition expresses the fact that if we reverse the momentum and do T further time steps we will reach our starting configuration again. This will be needed to prove detailed balance for the final algorithm.

The basic idea is to evolve the field along finite length classical trajectories. To make sure we sample the correct distribution e^{-S} we need to restart the evolution periodically, choosing our initial momenta from a gaussian distribution. Notice that I can always add such momenta to the action – they integrate out to merely rescale the partition function. If our integration was exact this would be the end of the story. Of course in practice the Hamiltonian is not exactly conserved and so Δt^2 errors are induced. However, because of the reversibility criteria we can choose to view the entire classical trajectory as merely a way of providing a global move on the fields. We then render the algorithm exact by imposing a Metropolis test on the final change in the Hamiltonian. If it fails we go back to the beginning of the trajectory and choose new gaussian distributed momenta. This Metropolis step eliminates the Δt dependence. The resulting algorithm is called Hybrid Monte Carlo HMC (it's a hybrid of Metropolis and classical dynamics algorithms). As we will discuss later it can be adapted to combat critical slowing down by performing the update in momentum space with a time step that depends on the momentum of the field component allowing long wavelength (critical) fluctuations to be updated faster than short wavelength modes. This requires an efficient way to go back and forth between real space and momentum space – such fast fourier transforms exist (see Numerical Recipes).

7.5 Langevin Equation

There is an interesting limit of this dynamics in which just a single step of the classical dynamics is carried out (and the Metropolis test is omitted). In this case we need not evolve

the momentum field explicitly (since it is thrown away after 1 step anyway) and the evolution of the field $\phi(x)$ can be written

$$\phi(x)_{n+1} = \phi(x)_n + \epsilon \frac{\partial S}{\partial \phi_n} + \sqrt{2\epsilon} \eta$$

where $\epsilon = \frac{1}{2}\Delta t^2$ and η is gaussian random noise. This update gives an exact simulation of the model in the limit $\epsilon \rightarrow 0$ where it is called the **Langevin equation**. This equation is actually used to model the dynamics of many complex systems with an underlying stochastic dynamics. For short times the fields suffer random forces due to the pure noise term but over large times they feel the effect of the drift force given by the gradient of S . Brownian motion and a variety of diffusive phenomena can be handled this way.

At finite step size observables will pick up errors of $O(\epsilon)$ and so in principle to use this algorithm requires running for several step sizes and extrapolating all results to zero stepsize. In practice once one understands the magnitude of the systematic ϵ errors one can take a step size which generates systematic errors well below the statistical errors and not worry further. Again this update may be done in momentum space to help alleviate critical slowing down using Fourier acceleration techniques in a way analogous to HMC. Both of these algorithms really come into their own when discussing simulation of relativistic fermions which are associated with non-local effective Hamiltonians/actions. We will discuss this further later.

7.6 Fourier Acceleration

Both HMC and the Langevin algorithm suffer from the usual problems of critical slowing down. In this case this manifests itself in the fact that the classical dynamics is rather poor at evolving the long wavelength components of the field – and it is precisely these which are important for long distance observables such as the field averaged over the lattice. To see this explicitly consider the case of $\lambda = 0$ in our 2d scalar field model. The discrete time equations for the case of HMC look like

$$\begin{aligned} \phi_{n+1}(x) &= \phi_n(x) + \epsilon p_n(x) + \frac{1}{2}\epsilon^2 f(\phi_n(x)) \\ p_{n+1}(x) &= p_n(x) + \frac{\epsilon}{2}(f(\phi_n(x)) + f(\phi_{n+1}(x))) \end{aligned}$$

where for $\lambda = 0$

$$f_n(\phi_n(x)) = -\frac{\partial S}{\partial \phi_n(x)} = \kappa \sum_{\mu} (\phi_n(x + \mu) + \phi_n(x - \mu)) - 2\phi_n(x)$$

Eliminating the momentum yields

$$\phi_{n+1}(x) - 2\phi_n(x) + \phi_{n-1}(x) = \epsilon^2 f(\phi_n(x))$$

When f is linear in ϕ (free case) this equation may be solved by Fourier transforming to momentum space leading to the mode equation

$$\phi_{n+1}(p) - 2\phi_n(p) + \phi_{n-1}(p) = \epsilon^2 \left(2\kappa \sum_{\mu} \cos(p_{\mu}) - 2 \right) \phi_n(p)$$

where

$$\phi(p) = \frac{1}{N} \sum_x e^{ip \cdot x} \phi(x)$$

and N is the total number of sites. The lattice momenta are given by $p = \frac{2\pi}{La}(m, n)$ in two dimensions where $m, n = 0 \dots L-1$. The LHS of the evolution equation is an approximation for $\frac{d^2\phi(p,t)}{dt^2}$ and this equation can be recast as

$$\frac{d^2\phi(p,t)}{dt^2} = (2\kappa \sum_{\mu} \cos(p_{\mu}) - 2)\phi(p,t)$$

Examine this equation in the vicinity of $\kappa = \frac{1}{2}$ (the critical point of the free theory where $m^2 = 0$) Notice that modes with small $p \sim 0$ are updated **very** slowly in this limit – the frequency being $\omega^2 = \sum_{\mu} \cos(p_{\mu}) - 2 \rightarrow 0$ for $p_{\mu} \rightarrow 0$. This is the origin of the problem in this context. The effective time step for update of the long wavelength modes tends to zero. It is now also obvious how to fix the problem – do the update in momentum space and use a time step $\epsilon(p)$ in the discrete time evolution equations that cancels out the *intrinsic* frequency dependence of the update. Thus we modify the update equations to look like

$$\begin{aligned} \phi_{n+1}(p) &= \phi_n(p) + \epsilon(p)p_n(p) + \frac{1}{2}\epsilon(p)^2 f(\phi_n(p)) \\ p_{n+1}(p) &= p_n(p) + \frac{\epsilon(p)}{2}(f(\phi_n(p)) + f(\phi_{n+1}(p))) \end{aligned}$$

We choose a function $\epsilon(p)$ which is peaked near $p = 0$. One possible choice is

$$\epsilon(p) = \epsilon_0 \sqrt{(1+m^2)} \sqrt{\left(\sum_{\mu} (1 - \cos(p_{\mu})) + m^2\right)}$$

which will remove all p -dependence from the update at $m = 0$ ($\kappa_c = 1/2$) in the free theory. Typically in the interacting theory a choice of $m = 1/\xi$ the inverse correlation length is close to optimal. The one missing piece in all this is a method to find the Fourier components of the original field. In practice the forces are all computed in real space so we must have a technique for going back and forth between momentum space and real space efficiently. However such an algorithm exists – the fast Fourier transform (see Numerical Recipes) and can be employed as a *black box* routine to do the transformation. The operation count scales only as $N \ln N$ where N is the number of lattice sites. The algorithm proceeds as follows:

1. Using current field configuration $\phi(x)$ compute the forces needed for one step of the classical dynamics
2. Fourier transform the fields to (lattice) momentum space. Update the field $\phi(p)$ using $\epsilon(p)$.
3. Inverse transform back to find new $\phi(x)$. Use it to compute the updated forces.

4. Fourier transform the new forces back to momentum space
5. Update the momentum $p(p)$ (sorry - horrible notation!)
6. Repeat for next time step

Notice that this Fourier acceleration technique can also be used for the Langevin equation.

7.7 Application – relativistic fermions

The HMC algorithm we have described was initially invented to handle simulations of QFT's containing relativistic fermions. Such systems can be represented by *non-local* effective actions or Hamiltonians. To appreciate why these actions take the form they do takes a little digression into the representation of fermions in QFT. A state containing a single fermion in state r is denoted $|1_r\rangle$. We can imagine that such a state is created by acting on the ground state $|0\rangle$ by a fermion creation operator a_r^\dagger in complete analogy to the raising operators used to create states in the simple harmonic oscillator (a similar description can be made for bosons although our initial introduction to QFT made no use of it). Thus a state with 2 fermions in states r and s would be written

$$|1_r 1_s\rangle = a_r^\dagger a_s^\dagger |0\rangle$$

But we learn in ordinary QM that such a fermion state must be antisymmetric under exchange of the two fermions. Thus

$$|1_r 1_s\rangle = -|1_s 1_r\rangle$$

This in turn requires

$$a_r^\dagger a_s^\dagger = -a_s^\dagger a_r^\dagger$$

The dof used to describe fermions are *anticommuting* variables. Notice that the Pauli principle is automatically a consequence of this assumption

$$a_r^\dagger a_r^\dagger = 0$$

Anticommuting variables are often called *grassmann* variables. Functions of grassmann variables are considerably simpler than their commuting cousins. For example if we have N grassmann variables $\theta_i, i = 1 \dots N$ they obey the algebra

$$[\theta_i, \theta_j]_+ = 0$$

where the bracket denotes *anticommutator*. A function of such variables has a finite Taylor expansion

$$f(\theta) = a + b_i \theta_i + c_{ij} \theta_i \theta_j + \dots + \epsilon \theta_1 \theta_2 \dots \theta_N$$

Integration is defined as

$$\int d\theta_i = 0, \quad \int d\theta_i \theta_i = 1$$

We now state (we will not prove) that the Feynman amplitude to evolve from one fermion field configuration $\theta(x)$ to another $\theta'(x)$ is given by the grassmann integral

$$Z = \int D\theta e^{-S_F(\theta)}$$

In practice the fermion action $S_F(\theta)$ (now defined on a lattice) most often takes the quadratic form

$$S_F = \sum_x \theta(x) M_{x,y} \theta(y)$$

where M is a local $N \times N$ (antisymmetric) matrix operator. Consider the case $N = 2$. The most general form of M is

$$M = \begin{pmatrix} 0 & m \\ -m & 0 \end{pmatrix}$$

Taylor expanding the exponential function and using the rules of grassmann integration leads to

$$I = \int d\theta_1 d\theta_2 (1 - m\theta_1\theta_2 + m\theta_2\theta_1) = 2m$$

But $2m = 2\sqrt{\det(M)}$. This generalizes and in general the effect of integrating out the fermion dof in a relativistic field theory can be encapsulated by simulating the (square root) determinant of a fermion operator.

$$Z = \int D\phi \sqrt{\det(M(\phi))} e^{-S(\phi)} = \int D\phi e^{S_{\text{eff}}(\phi)}$$

where the effective action is given by

$$S_{\text{eff}} = S(\phi) - \frac{1}{2} \text{Tr} \ln (M(\phi))$$

Notice this action is *non-local*. This in turn may be represented through **commuting** *pseudofermion* variables $\sigma(x)$ via the identity

$$\sqrt{\det M} = \int D\sigma(x) e^{\sigma M^{-1} \sigma}$$

where we have assumed the operator M is real positive definite so that the σ integral is well-defined. This result is a simple generalization of the usual rule for gaussian integrals $\int dx e^{-x\alpha^{-1}x} = \sqrt{\pi} \sqrt{\alpha}$ (these restrictions on M can be gotten around in some physically interesting cases)

The non-locality of S_{eff} renders simple algorithms like Metropolis highly inefficient – the update of a single site variable will take a computation time proportional to the system volume cubed ! HMC type algorithms are the only solution since they update all the degrees of freedom at once. Conjugate momenta for both the original field $\phi(x)$ and the pseudofermion field $\sigma(x)$ are introduced and the combined system evolved according to the usual HMC

algorithm. This evolution relies on computing the derivatives $\frac{\partial S_{\text{eff}}}{\partial \sigma(x)}$ and $\frac{\partial S_{\text{eff}}}{\partial \phi(x)}$. The latter is given by

$$-f(x) \frac{\partial M}{\partial \phi(x)} f(x)$$

where $f(x)$ the pseudofermion force is given as the solution of the (very large) matrix equation

$$M_{xx'}(\phi) f(x') = \sigma(x)$$

While computationally costly this linear system can be solved in $O(V)$ arithmetic operations using *conjugate gradient* techniques (see later) giving an total update time per dof that (in best case scenarios) does not scale with system volume.

7.8 Solving large (sparse) linear systems

We have seen that the HMC algorithms require a force calculation associated with the pseudofermion action of the form $f(x) = M_{xx'}^{-1} \sigma(x')$. In the continuum M is a derivative operator while on the lattice it takes the form of a matrix of size $V \times V$ where V is the total number of lattice sites (actually it can be even larger if $\phi(x)$ contains other “internal” degrees of freedom). Naively one might think that one simply inverts the matrix at each step to calculate the forces. You should *never* do this. The standard matrix inverter routines cost V^3 arithmetic operations which is prohibitively expensive for realistic lattice sizes. Remember this inversion has to be done *at each step* of the classical evolution. Even if you could stomach this CPU load the storage of such an inverse matrix would rapidly become impossible – for example in simulations of QCD one typically encounters $V \sim O(10^7)$ leading to storage of 10^{14} real numbers (100-1000 Terabytes of RAM required!). Thus we need algorithms to solve the linear system $Mf = \sigma$ which

- Scale (much) more slowly with size than V^3 . Optimally like V .
- Require storage $O(V)$

The key fact that allows us to come up with such an algorithm is that most of the matrix elements of M are in fact zero. The matrix is said to be *sparse*. For example, simple quantum mechanical “fermions” would require an M of the form

$$M_{xx'} = \delta_{x+a,x'} - \delta_{xx'} + \delta_{xx'} P(\phi)$$

where the first term corresponds to a single (forward) difference operator (discretization of the 1D Dirac operator) and the second a coupling to a scalar potential. This is a tridiagonal matrix. These sparseness properties can be exploited by a wide class of linear solver which attempt to find solutions of the force equation by iteration. The simplest, most robust and the only one we will discuss here is called *conjugate gradient*. This requires that the matrix is positive definite (that is possesses only real, positive eigenvalues). This turns out often to be the case in physically interesting situations (and the algorithm can be generalized to cover other situations also).

7.8.1 Conjugate gradient algorithm

Consider the $N \times N$ linear system

$$Ax = b$$

It is based on the idea of minimizing the function

$$f(x) = \frac{1}{2}x \cdot Ax - b \cdot x$$

This function is minimized when its gradient

$$\nabla f = Ax - b$$

is zero which corresponds to the solution of the original system. The minimization is carried out by generating a succession of search directions p_k and improved minimizers x_k . At each stage a quantity α_k is found that minimizes $f(x_k + \alpha_k p_k)$ and x_{k+1} is set equal to the new point $x_k + \alpha_k p_k$. The p_k 's are constructed in such a way that the x_k is automatically the minimizer over all directions $p_j, j = 1 \dots k$ already examined. Thus in exact arithmetic the solution will be found after at most N iterations. Actually the presence of finite precision computer arithmetic means that this should be regarded as a genuine iterative procedure to be halted when the solution is not changing very much. Typically one can ask the norm of the residual $r_k = Ax_k - b$ is smaller than some tolerance. The algorithm proceeds as follows

1. Make an initial guess at the solution x_1 . The components of this vector can be simply calls to a random number function.
2. Compute the initial residual vector $r_1 = b - Ax_1$. Set the initial search direction equal to this $p_1 = r_1$.
3. Commence an iterative loop. For $k = 1, 2, \dots$
 - Compute $\alpha_k = \frac{r_k \cdot r_k}{p_k \cdot A p_k}$
 - Set $x_{k+1} = x_k + \alpha_k p_k$
 - Set $r_{k+1} = r_k - \alpha_k A p_k$
 - Compute $\beta_k = \frac{r_{k+1} \cdot r_{k+1}}{r_k \cdot r_k}$
 - Set $p_{k+1} = r_{k+1} + \beta_k p_k$
4. End loop when $r_k \cdot r_k < \epsilon$

Notice that this routine only requires us to be able to compute a dot product between vectors (scales like V) and a function to apply the matrix A to a vector. This latter function can exploit the sparseness properties of A in order to reduce the operation count also to $O(V)$. Thus the CPU time to solve the original system scales like VT where T is the number of iterations required. In exact arithmetic and requiring an exact solution would necessitate

$T = V$. Thus the algorithm would still beat straight matrix inversion. However, if we can tolerate only an approximate solution (and this may not hurt us in a HMC code as we only need the solution be good to $O(\Delta t^2)$) we can often find $T \sim V^x$ where $x < 1$. Away from the critical point $x \sim 0$. Actually techniques related to Fourier acceleration may also be used to speed up the linear solver close to criticality (and reduce x again). We will not discuss those further here. Notice – the solution of large sparse linear systems occurs frequently in computational physics and the conjugate gradient algorithm is generally regarded as the standard workhorse for all such problems.

8 Introduction to the Renormalization Group

Typical lattice QFTs or statistical physics theories consist of a very large number of interacting degrees of freedom. Except for free theories and a very few exactly solved two dimensional models these systems have no exact analytic solutions. While perturbation theory can sometimes be used to elucidate key features of these systems it cannot be applied across the board – for example such an approach fails for strongly coupled systems. This was part of the motivation for developing simulation methods which could access the non-perturbative physics in such systems. There is one other tool which has become very important for understanding both qualitative and quantitative aspects of such systems – the **Renormalization Group**. It originally arose in trying to deal with the infinities encountered in perturbative Feynman diagrams in continuum QFT but it was Wilson in 1971 who was the first to understand the correct physical interpretation of this approach and to apply it to condensed matter systems. This Wilsonian RG has now been applied across a very wide variety of systems from string theory to turbulence and gives a very powerful way of thinking about the physical content of the models. It has also led to new (approximate) ways of computing universal quantities like critical exponents – this is the ϵ -expansion which you may have heard of. From our point of view in this class we will look at a way of combining the ideas of the RG with the results of Monte Carlo simulation – which yields a calculational tool called the Monte Carlo renormalization group MCRG.

The basic idea of the RG is to construct a transformation which maps the system $\{\phi(x)\}$ on a lattice with spacing a and Hamiltonian $H(\phi)$ to a new coarse-grained system on a lattice with spacing ba ($b > 1$) and new variables $\{\phi'(x)\}$ and Hamiltonian $H(\phi')$ in such a way that the partition function of the primed system is conserved and such that physical correlators at distances greater than the (new) lattice spacing are preserved. Another way of saying this is that we want to integrate out the degrees of freedom at the smallest scales to produce a renormalized or effective system on a coarser lattice. This integration, sometimes called *blocking* transformation should preserve the long distance physics of the system. Thus

$$H' = R(g)H$$

where the renormalization group transformation R will depend on the coupling constants of the model. If you attempt to do such a procedure it will rapidly become apparent that one

cannot in general think of this transformation as acting on just a few coupling constants. In general blocking the system will generate new interactions not present in the original system and so the RG is best thought of as acting in an infinite dimensional coupling constant space

$$H = \sum_{i=1}^{\infty} g_i O_i$$

where the operators (terms in the Hamiltonian) would be things like

$$O_i = \sum_x \phi^2(x), \sum_{\mu} \phi(x)\phi(x + \mu), \phi^4(x), \dots$$

Notice that distances on the blocked lattice will scale like r/b . Thus the correlation length will scale as $\xi' = \xi/b$. To preserve Z you will in general need to rescale the field $\phi' = c^{-1}\phi$. Thus a two point function $\Gamma = \langle \phi(0)\phi(r) \rangle$ will transform according to

$$c^2 \Gamma(r/b, H') = \Gamma(r, H)$$

The most interesting region of the model corresponds to the critical point where the correlation length is infinite and the system hence scale invariant. In the language of the RG this corresponds to fixed points of the RG transformation where $H' = H$. We will be interested in the behavior of the RG in the vicinity of such fixed points since in that region the correlation length will be large but not actually infinite (as for finite volume systems). Under an RG transformation the coupling constants will *flow*

$$g_i = R(g)$$

Thus we see an immediate consequence of the RG assumption – the coupling constants of the renormalized theory will not be constants at all - but must change with length scale in order to keep the physical content of the theory fixed. This is the main conclusion of the renormalization procedure as seen in perturbative field theories but you see here the physical interpretation of this is quite natural and actually makes no direct reference to infinities. One can use RG methods even in situations where no infinities strictly arise. It merely corresponds to finding long distance degrees of freedom and an associated Hamiltonian in which to discuss long distance physics. The hope is that this Hamiltonian will be simpler and the blocked fields more relevant to the discussion of phenomena at that large length scale – the microscopic degrees of freedom should play no role except in renormalizing the values of couplings etc. Returning to this flow of couplings consider linearizing the flow in the vicinity of a fixed point of the RG.

$$(g_i^* + \delta g'_i) = R(g_i^*) + T_{ij} \delta g_j$$

where $T_{ij} = \left. \frac{\partial R^i}{\partial g_j} \right|_{g^*}$ is the derivative of the RG transformation vector at the fixed point. Thus we see

$$\delta g' = T \delta g$$

as a (infinite dimensional) matrix equation. Now change variables to the (right) eigenvectors of T v_i with eigenvalues $\lambda_i(b)$. Since I may consider a product of two such RG transformations to yield another RG transformation (the group property) the eigenvalues satisfy the condition

$$\lambda_i(b)\lambda_i(b') = \lambda_i(bb')$$

which constrains the eigenvalues to be of the form $\lambda_i = b^{y_i}$. The y_i turn out to be related to critical exponents. Now let us describe the flows near such a fixed point in terms of these eigenvectors

$$\delta g_i = \sum_j \alpha_j v_j$$

Under renormalization

$$\alpha'_i = b^{y_i} \alpha_i$$

Thus for positive y the coupling α will increase under renormalization driving the system away from the fixed point. Such a coupling corresponds to a *relevant* interaction. If $y < 0$ the coupling is driven to its fixed point value under renormalization. Since the value of that coupling after many blockings is independent of its value on the starting lattice it is termed *irrelevant* - long distance observables do not depend on it. Finally the case $y = 0$ is termed marginal and we must include higher terms in the Taylor expansion of the RG transformation to determine whether how such a coupling flows.

Thus the stability of the fixed point is determined by the number of relevant and irrelevant operators at that point. In general for *renormalizable* theories there are only a finite number of relevant interactions. Thus the infinite dimensional coupling constant space will contain a infinite dimension *critical surface* corresponding to the irrelevant operators. A system starting off in this surface will flow under blocking to the fixed point rendering the system insensitive to the initial values of its irrelevant couplings. Such a system will be scale invariant and possess $\xi = \infty$. The long distance physics of the system will not depend on these irrelevant directions (universality). The relevant directions out of the fixed point correspond to operators whose couplings must be fine tuned to render the system critical ($\xi = \infty$). Thus κ is a relevant coupling for the 2d scalar field model. The temperature and magnetic field are relevant couplings for the Ising model. If some system starts off close to the critical surface (large but not infinite ξ) it will initially run toward the fixed point before eventually running out from it along a relevant direction. Thus its critical behavior will be determined by the linearized RG transformation at the fixed point and independent of many details of its microscopic formulation. This is another manifestation of universality.

We see that the *concept* of an RG already allows us to understand many aspects of critical phenomena without even choosing a specific RG transformation or doing any calculations! We can take this a step further. Consider the free energy (per unit volume)

$$f(g) = b^{-d} f(g')$$

Close to a fixed point we can write

$$f(\alpha_1, \alpha_2, \dots) \sim b^{-d} f(b_1^y \alpha_1, b_2^y \alpha_2 \dots)$$

We say that the singular part of the free energy can be written in a *scaling form* Thus

$$C = \frac{\partial^2 f}{\partial \alpha_1^2} = b^{-d+2y_1} f(b_1^y \alpha_1, 0)$$

where we have set all other relevant couplings to zero. Choosing $b_1^y \alpha_1 = 1$ leads to

$$C \sim (\alpha_1)^{\frac{d-2y_1}{y_1}} f(1, 0)$$

Identifying κ with α_1 allows us to derive a power law behavior for the specific heat near the critical point where the power is indeed related to the eigenvalue of the linearized RG transformation. We can further identify the exponent y_1 by recalling that all lengths, including the correlation length scale like b . Thus $\xi \sim \alpha_1^{-\frac{1}{y_1}} = \kappa^{-\frac{1}{y_1}}$ Thus the critical exponent $\nu = \frac{1}{y_1}$. We also see that $c = b^{(d-2+\eta)}$ from the behavior of the two point function at criticality $\alpha_1 = 0$. Thus the critical exponents we introduced earlier are indeed related to the various eigenvalues of the linearized RG (we have only examined one such relevant coupling – the other would correspond to a source term for the ϕ field which breaks the symmetry $\phi \rightarrow -\phi$).

8.1 MCRG

The essence of this approach is to use Monte Carlo simulations to estimate the derivative of the RG transformation for a finite lattice close to criticality. The idea is to take some large lattice and to perform an explicit blocking or RG transformation on it to derive a sequence of *blocked* lattices. A finite basis of operators is chosen and an approximation for the matrix T is then calculated from cross correlations of this operator set between blocked and unblocked lattices. Many choices are possible for the blocking transformation. The simplest corresponding to a rescaling factor of 2 consists of grouping the original lattice fields in sets of 2^D on the original D-dimensional lattice and assigning them to a new block field on a lattice with twice the original lattice spacing.

The value of the block lattice fields can simply be the the average over the corresponding group of original lattice fields. We also choose some finite basis of operators $H = \sum_i g_i O_i$ where the lattice operators can be taken to be powers of the field, nearest neighbors products, next nearest neighbor products of fields, products of 4 fields around a square etc etc. Typical calculations may require a basis of 20-100 such operators. The matrix $T_{ij} = \frac{\partial g_i^{n+1}}{\partial g_j^n}$ is found using the chain rule

$$\frac{\partial \langle O_i^{n+1} \rangle}{\partial g_j^n} = \sum_k T_{kj} \frac{\partial \langle O_i^{n+1} \rangle}{\partial g_k^{n+1}}$$

where n and $n+1$ label successive blockings of the lattice and the derivatives may be rewritten

$$\frac{\partial \langle O_i^{n+1} \rangle}{\partial g_j^n} = \langle O_i^{n+1} O_j^n \rangle - \langle O_i^{n+1} \rangle \langle O_j^n \rangle$$

and

$$\frac{\partial \langle O_i^n \rangle}{\partial g_k^n} = \langle O_i^n O_k^n \rangle - \langle O_i^n \rangle \langle O_k^n \rangle$$

where all the expectation values from from a high statistics simulation on a large lattice close to criticality. Of course any finite lattice can only be blocked a finite number of times but the idea is that the leading eigenvalues of T should start to converge to some limiting values independent of the basis set of operators as long as it is sufficiently large. This approach can be quite accurate eg. for 3d Ising model yield $\nu = 0.629(4)$, $\eta = 0.031(5)$ which are competitive with the best analytic estimates from series expansions.

9 Introduction to Lattice Gauge Theories

9.1 Ising gauge system

We now turn to discussion of a new class of lattice model – lattice gauge theories. These arose out of attempts to write down well behaved lattice theories of QCD the strong interaction but can be formulated rather generally. Indeed, the first lattice gauge theory was invented some years earlier by Wegner from an attempt to write down an Ising-like system with no *local* order parameter. Consider the usual Ising Hamiltonian

$$H = -\beta \sum_x \sum_{\mu} \sigma(x) \sigma(x + \mu)$$

where the spins σ sit on lattice sites and are coupled along links. Let us imagine placing the spins not on the sites but on the links themselves thus $\sigma(x) \rightarrow \sigma_{\mu}(x)$ and replacing the Ising Hamiltonian with a four spin interaction (one interaction term per square or “plaquette” of the lattice)

$$H' = -\beta \sum_x \sigma_{\mu}(x) \sigma_{\nu}(x + \mu) \sigma_{\mu}^{-1}(x + \nu) \sigma_{\nu}^{-1}(x)$$

where $\sigma_{\mu}(x)$ points along the link in the positive μ -direction and $\sigma^{-1}(x)$ along the negative μ -direction. We define $\sigma^{-1}(x)$ by requiring $\sigma(x) \sigma^{-1}(x) = 1$. In the Ising case this just means $\sigma^{-1}(x) = \sigma(x)$ but in more complicated cases it may not be so. The motivation for this model may not be immediately clear. But consider in detail the form of this new Hamiltonian. Notice it is *invariant* when the σ fields are modified in the following way

$$\sigma_{\mu}(x) \rightarrow G(x) \sigma_{\mu}(x) G^{-1}(x + \mu)$$

where $G(x) = \pm 1$ and can be chosen *independently* at each lattice site. This is called a *local gauge transformation*. It is quite different from the usual $Z(2)$ spin flip symmetry of the regular Ising Hamiltonian. This latter transformation is a *global* symmetry since the same transformation must be applied at all lattice sites unlike the gauge transformation. Gauge systems such as this possess no local order order parameter – that is the expectation value

of a link is always zero – even in infinite volume. To see this set $\sigma_\mu(x) = \sigma$ and change variables on all links out of site x according to the rule

$$\sigma_\nu(x) \rightarrow \sigma \sigma_\nu(x) \quad \nu \neq \mu$$

This removes all dependence on σ from the Hamiltonian and means that the integral

$$\langle \sigma \rangle = \frac{1}{Z} \sum_{\pm} \sigma e^{-H} = \sum_{\pm} \sigma = 0$$

Notice that the operator $\sigma_\mu(x)$ is not *gauge invariant*. And the vanishing of the expectation value of $\sigma_\mu(x)$ is an example of a wider result – that the expectation value of any operator which is not gauge invariant should vanish. Equivalently, since the configurations $\sigma_\mu(x)$ and $G(x)\sigma_\mu(x)G^{-1}(x+\mu)$ are the same (related by a symmetry) we should not be able to distinguish between them by measuring any physical observable. Thus all physical observables must be gauge invariant. The simplest example of such an observable is the plaquette term itself. This is just the local energy density of the model. But clearly the product of link fields around an arbitrary closed *Wilson loop* will also be gauge invariant. Imagine computing the expectation value of such an $R \times T$ loop in the limit $\beta \rightarrow 0$ (this is usually called *strong coupling* since for QCD $\beta \sim \frac{1}{g^2}$).

$$\begin{aligned} W(R, T) &= \langle \sigma_1(0, 0) \dots \sigma_1(R, 0) \sigma_2(R, 0) \dots \sigma_2(R, T) \\ &\quad \times \sigma_1^{-1}(R-1, T) \dots \sigma_1^{-1}(0, T) \sigma_2^{-1}(0, T-1) \dots \sigma_2^{-1}(0, 0) \rangle \end{aligned}$$

where the loop is taken in 12 plane with leftmost corner $(0, 0)$ and we simplify by considering two dimensions (the generalization to any number of dimensions is trivial) In this limit we can expand the exponential in powers of β . To get a non-vanishing result we clearly need to include a plaquette term from the Hamiltonian for each σ field appearing in the integrand. But these terms introduce new σ fields which will sum to zero unless they are compensated by further plaquette terms including that field coming from the exponential. Some thought reveals that the leading, non-zero term in this Wilson loop goes like

$$W(R, T) \sim \beta^{RT} \sim e^{-KR T}$$

corresponding to “tiling” the interior of the loop with a minimal surface of plaquettes. The coefficient in this exponential *area law* behavior is called the string tension K . If T is the temporal extent of the loop we can clearly interpret the factor of KR as the energy of a one dimensional string state. Notice this energy increases as the size of the string state R is increased – this phenomenon is called *confinement* since such a state can never reach macroscopic sizes in a phase of the model in which K remains finite. Thus the primary order parameter for distinguishing confining from free (or Coulomb) phases is the string tension. To leading order we have

$$K = -\ln \beta$$

One simple way to extract the string tension from MC data is to form *Creutz ratios*

$$\chi(R, T) = -\ln \left(\frac{W(R, T)W(R-1, T-1)}{W(R, T-1)W(R-1, T)} \right)$$

Any perimeter dependence disappears from these ratios and they give a good estimate of the string tension in an area law phase. The MC data for this model are linked from the lecture page. You should see that for $\beta \sim 0.45$ the mean energy drops very quickly and indeed in the infinite volume limit the curve exhibits a true discontinuity for $\beta = \beta_c$. This is the sign of a 1st order phase transition and the magnitude of the discontinuity measures the latent heat of the transition. Repeated MC simulations in the vicinity of such a point can reveal metastability – there are two minima of the free energy at that point – and the system can get trapped in the higher of the two as the coupling is varied – this is the analog of supercooling in liquid solid transitions which also proceeds via a 1st order transition. Large fluctuations can be seen in the MC time series corresponding to tunneling back and forth between these states at finite volume and the simulations can also exhibit *hysteresis* in which the system does not return to the same point in the energy space when its coupling is cycled quickly through the critical coupling and back. First order phase transitions cannot be used to construct continuum limits for lattice models as the correlation length *does not* go to zero in the vicinity of such a transition. As the simulations reveal they can also be problematic for local algorithms though which tend to be poor in tracking the true minima of the free energy close to the critical point. Algorithms have been devised to help with this (multicanonical algorithms) but we will not discuss these in this course. In the case of this $Z(2)$ gauge model we interpret the phase at small coupling β as a confining phase while that at large β is a free phase. This is illustrated by the plot showing the area dependence of the Wilson loop at various couplings.

9.2 General model

Instead of a simple spin with two possible values $\sigma = \pm 1$ we can consider more general objects. Since I want the product of two such generalized spins to be another spin it is natural to take the degrees of freedom as residing in some group. In practice one usually uses so-called *compact* Lie groups in which the variables

$$\sigma_\mu(x) \rightarrow U_\mu(x) = e^{iA_\mu(x)}$$

where $A_\mu(x) = \sum_a A_\mu^a(x)T^a$ is called the vector potential. The matrices T^a are traceless hermitian matrices which form a basis for the Lie algebra of the group and the functions $A_\mu^a(x)$ are real. In the case of the special unitary groups $SU(N)$ the index $a = 1 \dots N^2 - 1$. These groups are of special interest to particle physicists as they underlie QCD ($SU(3)$) and the electroweak theory ($SU(2) \times U(1)$). As for the $Z(2)$ gauge model we can write a Hamiltonian which looks like

$$H = \sum_x \sum_{\mu < \nu} \beta \left(1 - \frac{1}{N} \text{ReTr} U_\mu(x) U_\nu(x + \mu) U_\mu^\dagger(x + \nu) U_\nu^\dagger(x) \right)$$

where we have added a constant of no importance and utilized the unitary nature of the U -matrices to write the inverse in terms of the adjoint. Let us now assume that in the continuum limit $A_\mu(x)$ becomes small. This is reasonable since one usually thinks of the vector potential as being of length dimension minus 1 so that $A_\mu(x)$ must contain a factor of the lattice spacing a in it. As $\beta \rightarrow \infty$ we expect now that the matrices approach the identity and we can approximate the Wilson gauge action by power expanding in a . Thus

$$\text{Tr}U_\mu(x)U_\nu(x+\mu)U_\mu^\dagger(x+\nu)U_\nu^\dagger(x) = \text{Tr}e^{iA_\mu(x-\nu/2)}e^{iA_\nu(x+\mu/2)}e^{-iA_\mu(x+\nu/2)}e^{-iA_\nu(x-\mu/2)}$$

To leading order we find

$$\text{Tr}e^{iF_{\mu\nu}+\dots}$$

where

$$F_{\mu\nu} = \partial_\mu A_\nu - \partial_\nu A_\mu - i[A_\mu, A_\nu]$$

and the Hamiltonian becomes

$$H = \int d^d x F_{\mu\nu}^a(x) F_{\mu\nu}^a(x)$$

This is the famous Yang-Mills action. In the case of $U(1)$ the resulting equations of motion are nothing but Maxwell's equations. They reflect an underlying system whose Hamiltonian is invariant under local phase changes. For the general *non-abelian* groups $SU(N)$ they constitute a generalization of Maxwell to systems with internal degrees of freedom corresponding to representations of $SU(N)$ (eg color for QCD $N = 3$) but whose Lagrangian's are now invariant under local rotations by the non-abelian group. In the limit of small a we can also find out how the A_μ transforms. We put $G(x) = e^{i\phi(x)}$ and expand in powers of a again (we will work out the transformation for small ϕ)

$$(1 + i\phi(x) + \dots)(1 + iA_\mu(x) + \dots)(1 - i\phi(x + \mu) + \dots)$$

The leading pieces are

$$-i(\phi(x + \mu) - \phi(x) + i[A_\mu(x), \phi(x)])$$

In the continuum this looks like

$$-i(\partial_\mu \phi + i[A_\mu, \phi])$$

The quantity in brackets is called the *covariant derivative* of the field ϕ . This is the gauge transformation property of the vector potential in the small a limit.

9.3 Coupling to other fields

The Wilson action we have written down describes the gauge fields A_μ very well. But interesting theories will possess other types of field such as fermions (quarks) or Higgs scalars. Typically these fields transform in the fundamental representation of the gauge group which practically speaking means they should transform under gauge transformations like

$$\Psi(x) \rightarrow G(x)\Psi(x)$$

If we want to build gauge invariant terms out of products of neighboring fields we are forced to modify these field products in some way. It is not too hard to figure out what you must do

$$\Psi^\dagger(x)U_\mu(x)\Psi(x+\mu)$$

is gauge invariant. Using this one can write down lattice analogs of (gauged) kinetic terms

$$\Psi^\dagger(x)\gamma_\mu\left(U_\mu(x)\Psi(x+\mu)-U_\mu^\dagger(x-\mu)\Psi(x-\mu)\right)$$

Expanding this as a function of a we find

$$\Psi^\dagger\gamma_\mu(\partial_\mu+iA_\mu)\Psi(x)$$

where the quantity in brackets is again the *covariant derivative* now acting on fields in the fundamental representation of the group and explicitly couples the fermion current $J = \Psi^\dagger\gamma_\mu\Psi$ to the gauge field A_μ . Proceeding in this way we can attempt to write down gauge invariant discretizations of arbitrary continuum gauge theories. For example the following lattice action is a gauge invariant regularization of the usual Higgs sector of the standard model and describes the coupling of scalar fields to the gauge field

$$S_H = \sum_x \sum_\mu (U_\mu(x)\Phi(x+\mu) - \Phi(x)) \times \text{h.c.} + \mu\Phi^\dagger(x)\Phi(x) + (\Phi(x)\Phi^\dagger(x) - 1)^2$$

9.4 Fermion doubling

Unfortunately the fermionic sector of such models turns out to be problematic. The problems originate in the difficulty of discretizing the Dirac equation and go under the name of *fermion doubling*. Essentially the lattice theories contain more fermions than their continuum cousins. The extra states arise from replacing continuum derivatives by difference operators. And these extra states *do not* decouple as the lattice spacing is reduced. They render it impossible to define the chiral states needed by the standard model (each continuum chiral fermion gets a partner of opposite chirality). To see this simply consider the following, seemingly reasonable transcription of the continuum free Dirac equation (in one dimension!) to the lattice

$$\frac{i}{2a}(\psi(x+1) - \psi(x-1)) + m\psi(x) = 0$$

In lattice momentum space we have

$$(\sin(pa) + ma)\bar{\psi}(p) = 0$$

In the continuum limit $ma \rightarrow 0$ there are 2 solutions to this equation $pa \rightarrow 0$ and $pa \rightarrow \pi$. The first corresponds to the long wavelength continuum-like fermion we expected. But the second (which has the same energy) corresponds to a state with wavelength $\lambda = 2\pi/p =$

2a. This is a fermion doubler mode. In D dimensions there are 2^D of them (16 in four dimensions!) corresponding to the zeroes of the function

$$\sum_{\mu} \gamma_{\mu} \sin p_{\mu} a$$

These take the form

$$p_{\mu} a = \pi (n_1, \dots, n_D)$$

where $n_i = 0, 1$. Consider a mode with $n_i = 1$ and change variables from p_i to $p'_i = p_i - \pi$. Then we can see that

$$\gamma_{\mu} \sin (p_i a) = -\gamma_{\mu} \sin (p'_i a)$$

Thus this mode is associated (in the continuum limit) with a new set of γ matrices which are the negatives of the old ones. This will be important in our later discussion.

A variety of theorems show that these extra modes cannot be avoided by writing down different lattice difference operators or choosing lattices of different types. Notice also that the appearance of these states does not depend (and cannot be removed by) interactions. They will always appear in any local approximation to the derivative for translationally invariant systems which possess a *chiral symmetry*. The latter symmetry corresponds to replacing the field $\psi(x)$ by the rotated field $\psi' = e^{i\alpha\gamma_5}\psi$ and (as you can easily verify) is a good symmetry of massless continuum theories. The matrix γ_5 is just the product of the individual Dirac γ matrices $\gamma_5 = \prod_i^D \gamma_i$. Notice that lattice doubler modes in which an odd number of n_i are non-zero will possess a negative γ_5 operator and hence correspond to states of *opposite chirality*. Notice that the lattice theory we have described has exactly equal numbers of positive and negative chirality states and so *cannot* be used to describe theories possessing only fermions of one chirality. The prime example of such a theory is the electroweak sector of the standard model in which the lefthanded states (negative chirality) couple differently than the right handed states to the gauge bosons.

There is one way out – add a new term to the action which breaks chiral symmetry, and removes the doublers from the $a \rightarrow 0$ spectrum. The following Wilson fermion action shows what this looks like for (the free part of) a D -dimensional theory

$$\sum_x \sum_{\mu} \Psi^{\dagger}(x) \left(\frac{i}{2a} \gamma_{\mu} (\Psi(x + \mu) - \Psi(x - \mu)) + \frac{r}{2a} (\Psi(x + \mu) + \Psi(x - \mu) - 2\Psi(x)) + ma\Psi(x) \right)$$

In momentum space the Dirac equation now looks like

$$(\sin(pa) + ma + r(\cos(pa) - 1)) \bar{\psi}(p) = 0$$

Now as $ma \rightarrow 0$ only the mode with $pa \rightarrow 0$ survives – the doubler mode picks up an effective mass $2r/a$ and decouples in the continuum limit. The price one pays for this is that the new action is not chirally invariant and so one needs to fine tune the mass to reach a chiral continuum theory. It also *cannot* be used for chiral gauge theories. This method is a popular (although not the only way) to include fermions into lattice QCD. Thus a gauge

invariant and double free lattice fermion action for QCD is given by (we define $\gamma_{-\mu} = -\gamma_{\mu}$ and typically the Wilson parameter $r = 1$)

$$S_F = \sum_x \left(-\kappa \sum_{\mu=\pm 1}^{\pm 4} \Psi^\dagger(x + \mu) U_\mu^\dagger(x) (r + \gamma_\mu) \Psi(x) + \Psi^\dagger(x) \Psi(x) \right)$$

This can be simulated along the lines suggested earlier – namely (for 2 species of quarks) appropriate pseudofermion fields may be added and the resulting non-local action simulated using a HMC algorithm. To recover a chiral continuum theory will require tuning the hopping parameter κ with gauge coupling β . This is achieved in practice by requiring that the state with the quantum numbers of the pion be light as expected for a system with a spontaneously broken chiral symmetry. Masses of hadrons and other physical observables may be obtained from studying the correlation functions of appropriate operators. Finite volume effects are controlled by utilizing a succession of larger and larger four dimensional lattices.

10 Random Lattices and random systems

Many interesting statistical physics models possess Hamiltonians containing explicit random elements. Obvious examples include the random bond/field Ising model and spin glasses. In the context of particle physics the use of random rather than regular lattices allows us to keep translational symmetry whilst maintaining a U.V cut-off and also allows for study of theories on curved spaces. Often these random couplings represent the effects of disorder which is undoubtedly present in real condensed matter systems and can play an important role in the determining the physical properties of the system. This type of disorder is often called *quenched* since the couplings/randomness are independent of time. The presence of such disorder makes analytic calculations doubly difficult and most of the work on these systems has been done using numerical simulation.

There is another type of disorder in which the random elements themselves possess non-trivial dynamics and evolve on time scales comparable to the other degrees of freedom of the system. Such *annealed* disorder can appear in both condensed matter and particle physics contexts – eg fluid random surfaces or discrete quantum gravity. The physics of such systems can be particularly rich with most of these systems falling into new universality classes distinct from the cases of quenched randomness.

Here, we will provide a simple introduction to such systems again starting from the 2d scalar field model. The case of quenched disorder is most easily generating by simply formulating the model on a random lattice. A natural way to create such a lattice is to throw points at random into a two dimensional box (equipped with periodic boundary conditions) and to connect these points by links in such a way as to form a random *triangulation* of the space. The next section discusses how this is done

10.1 Creating a 2D random lattice

Choose the coordinates of a set of N lattice sites at random. Each lattice site is uniquely associated with a given cell consisting of all points closer to that chosen lattice site than any other. Consider two such points P_1, P_2 with corresponding cells C_1 and C_2 . Then the intersection $C_1 \cap C_2$ is either zero or consists of points on a line which bisects the line joining P_1 to P_2 at ninety degrees. In this case we say that P_1 and P_2 are nearest neighbors or form a 1-simplex. Similarly we can consider 3 points P_1, P_2 and P_3 and examine the mutual intersection of their cells $C_1 \cap C_2 \cap C_3$. If this is non-zero the three points are said to form a 2-simplex consisting of the triangle $P_1 P_2 P_3$. The intersection consists of a single point which is at the center of the unique circle which may be drawn through the original points. In this way the entire 2d plane may be decomposed into a set of non-overlapping triangles whose sides form links between sites considered as nearest neighbors. This decomposition is called a *triangulation*. Notice it is unique given for a fixed initial set of lattice sites.

This construction immediately supplies an algorithm for generating such a triangulation. The simplest procedure would be to check all sets of three points for possible inclusion in a 2 simplex. The set of vertices forms a 2-simplex or triangle only if the circumscribed triangle contains no other lattice site. Such a triangulation procedure is unnecessarily slow growing like V^4 in two dimensions. A simpler solution proceeds in the following way.

Suppose you have found a single bona fide triangle. Now this triangle borders 3 others which share a side with each of its three sides. Let us try and find one of these bordering triangles. It is clear that the center of the circumscribed triangle that includes a given link lies somewhere along the perpendicular bisector of that link. We can imagine gradually stepping out along that bisector and constructing a circle using the current perpendicular bisector point as center plus the two link lattice sites. If this circle contains many other lattice sites it is clear that we must decrease the distance out along the bisector. If it contains no other lattice sites we must increase our distance out along the bisector. Using these basic ingredients we may quickly locate a point at which this locator circle contains one other lattice site beside the original two lattice sites. These 3 sites now form the sought after new triangle. In this way new triangles are "grown" out from faces/links of existing triangles. This procedure may be summarized as follows:

- Locate one initial triangle by some simple procedure.
- Place its links in an array **grow**
- Loop while **grow** contains active links
 - Take link from **grow** with endpoints a and b and compute a vector v corresponding to its perpendicular bisector. Set the distance parameter λ to some initial value - say $|b - a|$.
 - * Compute a center for locator triangle $x = \frac{1}{2}(a + b) + \lambda v$
 - * Compute how many other sites S lie within locator circle. If $S = 1$ exit loop on λ .

- * If $S > 1$ decrease λ . If $S = 0$ increase λ .
- * Repeat
- Store identity of new triangle and place its links into **grow**. If link already exists in **grow** remove.
- Continue taking links off the **grow** array until empty.

Points to note:

This algorithm yields not only the triangulated lattice but also its *dual* consisting of the cell points (centers of circumscribed circles) and their neighbors.

It is not hard to generalize this algorithm to generate triangulations of arbitrary dimensional manifolds.

Notice that all distances between points should be computed taking into account the boundary conditions – thus links and triangles will in general wrap around the edges of the initial domain.

Having discussed how to generate such a lattice we now have to turn to the question of how to write down a Hamiltonian for a scalar field living on such a lattice.

10.2 Hamiltonians for random lattice scalar fields

A kinetic term for a scalar field on such a random lattice takes the form

$$K = \frac{1}{2} \sum_{\langle ij \rangle} \frac{\sigma_{ij}}{l_{ij}} (\phi_i - \phi_j)^2$$

while a mass term looks like

$$V = \frac{1}{2} \sum_i \sigma_i \phi_i^2$$

where σ_i is the volume (area) of the 2-cell dual to site i , σ_{ij} the volume (length) of the 1-cell dual to the link $\langle ij \rangle$ and l_{ij} the link length. Notice that σ_{ij} is just the distance between dual cell points. It can be shown that these formulae guarantee that solutions of the random lattice field equations converge smoothly to their continuum counterparts.

In the case of fermion fields an analogous construction can be made yielding a random lattice approximation to the Dirac equation. Notice though that in this case the system does not possess an exact translation symmetry so the no-go theorems guaranteeing the presence of fermion doubles do not apply. This was one of the motivations for studying such random lattice schemes and indeed doubler modes *do not appear* in the long distance structure of such theories. However, in general, spurious would-be doubler modes can still contribute to the short distance quantum renormalizations of various operators and typically do so in a way similar to Wilson fermions. Fine tuning is then generally needed to reach a chirally symmetric continuum limit in these models.

10.3 Random surfaces

One particularly interesting application of these ideas is to models of surfaces. Consider not one but three scalar fields living on a two dimensional lattice and set the mass of these fields to zero. I can consider these three fields as giving the coordinates of a two-dimensional surface embedded in a three dimensional (flat) space. The Hamiltonian of this system then corresponds to a series of harmonic spring energies governing the relative displacement of the points in the surface from each other - we have a model of a random surface undergoing thermal fluctuations. To make this correspondence exact we set all the parameters σ_i , σ_{ij} , l_{ij} to fixed values corresponding to a regular triangulation of the underlying surface.

However, the model we have described has some problems – to see this consider the mean size of the surface as seen from the three dimensional space. This is called the gyration radius and corresponds to the correlator

$$R^2 = \langle \phi^i(x) \phi^i(x) \rangle$$

But notice that this correlator can be computed since the two dimensional theory is free. Thus

$$\frac{1}{V} \sum_x \langle \phi(x) \phi(x) \rangle = \sum_p \langle \phi(p) \phi(p) \rangle = \sum_p \frac{1}{\sin^2(p/2)}$$

The latter sum can be approximated by an integral in the continuum limit and we find

$$R^2 \sim \int_{1/L}^1 \frac{dp}{p} \sim \ln L$$

if we have L lattice sites in each direction. In the infinite volume limit this diverges ! The physical interpretation of this is that the surface is unstable to growing spikes of arbitrary size – the surface does not look like a smooth 2d surface undergoing small undulations at all but is some kind of fractal object ! Thus the model we have written down is not physically sensible. What can have gone wrong ? Real surfaces and membranes cannot grow spikes because such structures cost a lot of *bending energy* (they also cannot self-intersect but we shall not discuss this aspect here). Thus to model realistic surfaces we should include a bending term in the Hamiltonian. The simplest thing we can do is to assign a extra curvature energy of the form

$$H_{\text{bend}} = -\beta \sum_{AB} n^A \cdot n^B$$

where n^A and n^B denote unit normals to the triangles A and B and the bending energy receives contributions from dot products of such normals across links of the triangulation. The resulting models show much more interesting phase structure – if we plot the specific heat as a function of β we find two phases separated by a continuous phase transition. For small β the model looks similar to the $\beta = 0$ case and is spiky. For large β the model enters a “flat phase” in which the normals all point in essentially the same direction. The crumpling transition that separates these phases has been the subject of much numerical and analytic work. Its existence allows the construction of continuum models describing the fluctuations

of smooth surfaces. Notice in this model that the triangulation is held fixed – the identity of each points neighbors does not change during the simulation – only the relative coordinates of those points in the three dimensional space. The models are sometimes referred to as crystalline random surfaces for this reason.

Actually another class of random surface model has been studied in which the triangulation itself can fluctuate – so-called fluid random surfaces. These models also have application to string theory since the quantum mechanical fluctuations of a string *worldsheet* can also be modeled by a discrete fluid random surface model. In this case the bending energy is referred to as extrinsic curvature and was first introduced by Polyakov as a way of generalizing the usual string model to possibly describe strings in physical dimensions and QCD. See the online Java simulations

http://simscience.org/membranes/advanced/essay/gravity_sim1.html which reveal the non-trivial phase structure of both fixed and fluctuating triangulation models as a function of curvature coupling. See also the link to the plot of the specific heat of the crystalline model as a function of bending coupling. So far we haven't discussed how to implement this fluctuating triangulation. It turns out it is not too hard to figure out how to this within the context of a Monte Carlo simulation.

10.4 Dynamical Triangulations

Any 2D random triangulation consists locally of pairs of triangles glued along their edges. Such (closed) triangulations can be classified (like continuum surfaces) into topologically inequivalent sets according to their genus (number of handles). Thus triangulations of a sphere can be distinguished from triangulations of a torus etc. The topology of a given triangulation is given in terms of the Euler characteristic χ . In 2D this is a simple function of the number of points N_0 , links N_1 and triangles N_2 via the formula

$$\chi = N_0 - N_1 + N_2$$

Thus $\chi = 2$ for a sphere, zero for a torus etc. In the case of fluid random surfaces we would like to study a system which incorporates a sum over all such random triangulations. In analogy to ordinary field theory this can be accomplished by devising a Monte Carlo simulation to stochastically estimate this (huge) summation. Such a procedure will produce a random walk in the space of all triangulations with the probability of visiting a given triangulation being proportional to its Boltzmann factor. From a practical point of view we need to be able to produce *local* changes of triangulation (since the change in action for such moves can also be computed locally). Furthermore, these moves must satisfy an ergodic property – one must be able to go using sequences of such moves from any triangulation to any other. A set of such moves exists and for a system with fixed number of triangulations is composed of just the so-called **link flip** move. This proceeds as follows:

- Select a link at random
- Find the two triangles which border that link. Extract the two additional vertices not involved in the link.

- Imagine deleting the original link and replacing it by a link which runs between the two additional vertices. This will ultimately mean the addition of two new triangles and the deletion of the initial two triangles. Compute the change in action for this process (notice that the action for a scalar field living on the vertices of the triangulation depends on the identity of the links)
- Use a simple Metropolis algorithm to update the triangulation.
- Repeat.

Notice that this process conserves the Euler characteristic and hence the topology of the triangulation. If we want to allow the triangulation to fluctuate in area (number of triangles) we need to add an additional move (and its inverse) corresponding to adding a new vertex inside an existing triangle and connecting it out to the original triangle vertices. Such a move inserts a three-fold coordinated node, three new triangles and three new links and ultimately, if accepted, deletes the original triangle. Its inverse corresponds to finding (at random) a three fold coordinated vertex and removing it from the triangulation, deleting its links and boundary triangles and adding one new triangle.

10.5 Data structures for random lattices

To implement these ideas in code we need to think about the best data structure for representing such (fluctuating) triangulations. The most elegant solution is to create a C++ class which contains information on a basic triangle. A minimal solution would look like

```

class Triangle{
private:
int vertices[3];
Triangle * neighbors[3];
public:
Triangle(void);
Triangle(int, int, int, Triangle &, Triangle &, Triangle &);
int GetVertex(int);
Triangle * GetNeighborTriangle(int);
int GetVertexFromLink(int, int);
};

```

Here, we store the 3 vertex labels of the triangle and information on the three neighbor triangles. The array `neighbors []` contains *pointers* or addresses of the triangles which surround the triangle in question. Notice the use of the `*` operator which *dereferences* the pointers in `neighbors []` to produce triangle objects. I include two constructors – a default constructor for creating an empty triangle and another that creates a triangle and assigns its vertices and neighbors. The other member functions are examples of the type of useful function you would need to code the link flip move. Thus I need to be able to extract vertices and neighbor triangles and also to locate the additional vertices in a triangle given one link. This can be accomplished using the *public* functions `GetVertex()`, `GetVertexFromLink()` and `GetNeighborTriangle()`. Finally we must be able to allocate/delete triangles dynamically as the code runs. Thus the variables must be allocated on the heap using the C++ `new` command eg.

```
new_tri_pointer=new Triangle(1,2,3,p1,p2,p3);
```

where `p1`, `p2` and `p3` are pointers (addresses in memory) of the three triangles which will be neighbor to the new triangle after it is created. Getting rid of a triangle is easy using the `delete` command eg.

```
delete new_tri_pointer;
```

You need to keep track of the total number of links, triangles and vertices separately. To embed the random triangulation into some space you need to add scalar fields to the components of the triangulation class. For example new `private` data would look like

```
double x[3],y[3],z[3]
```

You would need to implement `public` functions to retrieve this information also eg `GetScalarField()`. Finally, let us give some examples of code snippets for accessing triangle information in neighbors. The following code yields a neighbor triangle to a triangle pointed at by pointer `p`

```
*p.GetNeighborTriangle[0];
```

This can also be written

```
p->GetNeighborTriangle[0];
```

Hence a scalar field on that neighbor triangle could be accessed by the code fragment

```
p->GetNeighborTriangle[0]->GetScalarField[1];
```

A C code which uses these ideas is linked from the web page. You can take a look if you are interested in a specific implementation. Notice this code uses an *array* of pointers to objects of type `Simplex` (`Triangle` here) which makes it easy to pick up a simplex at random and do loops over all simplices in the lattice.

Let us make some final comments. First, these local retriangulation moves have a natural extension to D dimensions. In fact one can write down a code to explore the triangulation space of say a D -dimensional sphere in such a way that D is an input parameter. Since a triangulation loosely gives you an intrinsic way to measure distances between points it should be clear that models incorporating sums over triangulations look a lot like models possessing an integral over metrics. The latter operation is natural in models of quantum gravity and means that the triangulation models in say $D = 4$ dimensions have been proposed as discrete models for quantum gravity. The link shows a plot of the specific heat of such a model as a function of a bare Newton constant coupling. The rising peak is indicative of a phase transition. The latter caused some excitement when it was first found since the existence of a continuous phase transition signals a possibility of constructing a continuum quantum theory of gravity at strong bare coupling – offering a possible way of evading the usual it non-renormalizability of Einstein gravity. Unfortunately these hopes have faded with time as the transition (at least in the simplest of these models) appears to be first order in the large volume limit.