PHY 411-506 Computational Physics II

Chapter 8: Statistical Mechanics, Phase Transitions, and the Ising Model

Lecture 1

Monday January 14, 2008

LECTURE OUTLINE

Lecture Outline

The Ising Model and Statistical Mechanics

Magnetism

- Magnetic Fields are produced by electric currents and magnetic moments

$$\mathbf{B} = \mu_0(\mathbf{H} + \mathbf{M})$$

where $\mathbf{M}$ is the Magnetization and

$$\mathbf{J} = \nabla \times \mathbf{H}$$

is the electric current density

- The magnetic susceptibility of the material

$$\chi = \frac{\mathbf{B}}{\mu_0 \mathbf{H}}$$

- Diamagnetism – substance is repelled by a magnetic field

$$\chi < 0$$

  - ◇ Atoms have no net magnetic moment
  - ◇ Consequence of Lenz' Law – induced moment reduces $\mathbf{B}$

- Paramagnetism – substance is not magnetic and attracted by a magnetic field

$$\chi > 0$$

  - ◇ Atoms have net orbital and/or spin magnetic moment
  - ◇ Thermal motion causes $\mathbf{M} = 0$
  - ◇ Moments tend to line up with $\mathbf{H}$ – increases $\mathbf{B}$

- Ferromagnetism – substance is magnetic and attracted by a magnetic field

  - ◇ Quantum mechanical Exchange Interactions causes atomic moments to spontaneously align inside domains at temperatures below the *Curie temperature* T_{C}
  - ◇ Magnetic hysteresis – relation between $\mathbf{H}$ and $\mathbf{B}$ is not linear and depends on history of sample

- Magnetism is an inherently quantum phenomenon

  - ◇ Classical charged particles in thermal equilibrium are not diamagnetic

◇ Diamagnetic field of gyrating electrons exactly cancelled by boundary effects – physical argument by Niels Bohr

◇ Proved by H.-J. Van Leeuwen, J. Phys. Radium, **2**, 361 (1921).

Ising Model

- Magnetism in metals can be caused by local moments (orbital or spin) and by non-localized moments in electronic bands
- The Heisenberg Model assumes localized moments due to spins $\mathbf{s}_i$ at fixed lattice sites i and energy

$$E = -\sum_{i,j} J_{ij}\mathbf{s}_i \cdot \mathbf{s}_j - \mu \sum_i \mathbf{H}_i \cdot \mathbf{s}_i$$

  where J_{ij} represents the exchange interaction between spins at sites i, j, μ is the gyromagnetic ratio (magnetic moment) of the spin and $\mathbf{H}_i$ is the external magnetic field at site i

- The Ising Model assumes that the spins are classical and restricted to values

$$s_i = \pm 1$$

and that

$$J_{ij} = \begin{cases} J & \text{for nearest neighbors } \langle ij \rangle \\ 0 & \text{otherwise} \end{cases}$$

so that

$$E = -J \sum_{\langle ij \rangle} s_i s_j - \mu H \sum_i s_i$$

- The Ising Model on a one-dimensional lattice was given by Wilhelm Lenz to Ernst Ising as a PhD thesis topic and has an interesting history
  - ◇ Ising found that $T_{\mathrm{c}} = 0$ for this model – it is not ferromagnetic!
- The Ising Model on a 2-d square lattice is ferromagnetic at low temperatures
  - ◇ Kramers and Wannier found that

$$\frac{k_{\mathrm{B}} T_{\mathrm{c}}}{J} = \frac{2}{\log(1+\sqrt{2})} = 2.269\ldots$$

  - ◇ Lars Onsager and C.N. Yang found an exact analytic formula for the magnetization

Statistical Mechanics

- The Ising Model energy (Hamiltonian) has no kinetic term – the energy is all potential
- The model has no *dynamics* – cannot study it as a function of time t
- Study its *equlibrium statistical mechanics*
  - ◇ System has a fixed volume – spins are at fixed lattice sites
  - ◇ Fix the temperature T by placing it in contact with a large heat reservoir at constant temperature
    - ○ Individual spins can flip by exchanging energy with the reservoir
    - ○ The total energy of the system is not constant
  - ◇ An external magnetic field H is applied to each spin
- The Canonical Ensemble is a set of configurations or *microstates* α of the system with probability

$$P_\alpha \sim e^{-E_\alpha/(k_\mathrm{B}T)}$$

  - ◇ Observables are averages over this ensemble – for example the

magnetization

$$M = \sum_{\alpha} P_{\alpha} M_{\alpha}$$

where $M_{\alpha} = \sum_i s_i$

- For an Ising Model with N spins, there are 2^N microstates or configurations
  - ◇ For the 2-d model with $N = 20 \times 20 = 400$ spins

$$\text{No. of configs} = 2^{N_s} = 2^{400} = 2.58 \times 10^{120} .$$

    To sum all configurations at 1 billion per second would take 2.58×10^{111} seconds $= 8.8 \times 10^{103}$ years
  - ◇ If analytic solutions are not available, random sampling of configurations (Monte Carlo method) must be used

Mean Field Theory

- Mean Field Theory is an approximate way of finding analytic solutions
- The basic idea is to reduce a many-particle problem to an effective one-particle problem
  - ◇ The spin-spin interaction terms are replaced by an effective magnetic field

$$-J\sum_{\langle ij\rangle} s_i s_j \equiv -\mu H_{\text{eff}} \sum_i s_i$$

  - ◇ Compute the effective field approximately by replacing s_j by the thermal average

$$\langle s_j\rangle \equiv \langle s\rangle$$

  - ◇ The

$$H_{\text{eff}} = \frac{J}{\mu} \sum_{\text{nearest neighbors}} \langle s\rangle = \frac{zJ}{\mu}\langle s\rangle$$

    where z is the number of nearest neighbors or *coordination number of the lattice* ($z = 2d$ for a d-dimensional hypercubic lattice)

- The one-spin problem can easily be solved because there are only two configurations $s = \pm 1$ with canonical ensemble probabilities

$$P_{\pm} \sim e^{\pm \mu (H_{\text{eff}} + H)/(k_{\text{B}} T)}$$

  - ◇ The thermal average of this spin is

$$\langle s \rangle = \frac{e^{\mu (H_{\text{eff}} + H)/(k_{\text{B}} T)} - e^{-\mu (H_{\text{eff}} + H)/(k_{\text{B}} T)}}{e^{\mu (H_{\text{eff}} + H)/(k_{\text{B}} T)} + e^{-\mu (H_{\text{eff}} + H)/(k_{\text{B}} T)}} = \tanh \left(\frac{z J \langle s \rangle + \mu H}{k_{\text{B}} T} \right)$$

- This is an implicit equation for $\langle s \rangle$ – solving for $\langle s \rangle$ is a *root finding* problem

$$f(\langle s \rangle) \equiv \langle s \rangle - \tanh \left(\frac{z J \langle s \rangle + \mu H}{k_{\text{B}} T} \right) = 0$$

- For $H = 0$ the equation has a non-zero solution for

$$\frac{k_{\text{B}} T}{z J} \leq 1$$

  This means the system is ferromagnetic below a Curie temperature

$$T_{\text{c}} = \frac{z J}{k_{\text{B}}}$$

- ◇ Wrong for the 1-d model ($z = 2$)
- ◇ Too high for the 2-d model

- The spontaneous magnetization $\langle s \rangle$ can be found numerically – Exercise 8.1

Root Finding Algorithms

- Finding zeros (roots) of a function needed in many applications – see Appendix B of the textbook
- Numerical Recipes has Chapter 9 on Root Finding and Nonlinear Sets of Equations
- Finding extrema of $f(x)$ equivalent to finding roots of $f'(x)$ – see Chapter 10 Minimization or Maximization of Functions
- Finding roots can be tricky!
  - $\diamond$ $f(x) = 1 + |x|$ does not have any roots
  - $\diamond$ $1 + x^2$ and $\cosh(x)$ have roots, but not for real values of x
  - $\diamond$ $\sin(1/x)$ has ∞ number of roots – which one do you want?
  - $\diamond$ Make rough plot of function – where (approximately) are the roots? which root do you want to find?
- We will consider the simplest algorithms

Bisection method

- Algorithm assumes
  - ◇ $f(x)$ changes sign at root (won't work for $f(x) = (x-1)^2$)
  - ◇ root is *bracketed* by x_0 and x_1, i.e., $x_0 < \text{root} < x_1$
  - ◇ only one root in $[x_0, x_1]$
- Algorithm repeatedly bisects interval
  - ◇ compute $x_{\frac{1}{2}} = (x_0 + x_1)/2$
  - ◇ compute product $f(x_0) \times f(x_{\frac{1}{2}})$
    - ○ if product is positive replace $x_0 \leftarrow x_{\frac{1}{2}}$
    - ○ else replace $x_1 \leftarrow x_{\frac{1}{2}}$
  - ◇ repeat above steps until $|x_0 - x_1| \leq \epsilon$ (desired accuracy) or $f(x_{\frac{1}{2}}) = 0$
- The following program finds the root of $e^x \log(x) - x^2$

bisection.cpp

```
#include <cmath>
#include <iostream>
#include <iomanip>
using namespace std;

double f(double x) {
    return exp(x) * log(x) - x * x;
}

void print(int step, double x, double dx) {
    cout.setf(ios::right, ios::adjustfield);
    cout << " " << setw(4) << step << "    ";
    cout.precision(15);
    cout.setf(ios::left, ios::adjustfield);
    cout.setf(ios::showpoint | ios::fixed);
    cout << setw(20) << x << "  " << setw(20) << dx << '\n';
}

int main() {

    cout << " Bisection search for root of exp(x)*log(x) - x*x\n"
         << " ---------------------------------------------\n"
         << " Enter bracketing guesses x_0, x_1, and desired accuracy: ";
    double x0, x1, acc;
    cin >> x0 >> x1 >> acc;
```

```
cout << " Step            x                   dx\n"
     << " ----    ----------------    ----------------\n";
int step = 0;
double xHalf = (x0 + x1) / 2;
double dx = x1 - x0;
print(step, xHalf, dx);
double f0 = f(x0);
while (abs(dx) > abs(acc)) {
    double fHalf = f(xHalf);
    if (fHalf == 0) {
        dx = 0;
    } else {
        if (f0 * fHalf > 0) {
            x0 = xHalf;
            f0 = fHalf;
        } else {
            x1 = xHalf;
        }
        xHalf = (x0 + x1) / 2;
        dx = x1 - x0;
    }
    ++step;
    print(step, xHalf, dx);
}
```

```
}
```

Convergence rate

- Consider bisection after n steps

$$|dx_n| = |x_1 - x_0| \qquad \text{after } n \text{ iterations}$$
$$= \frac{1}{2}|dx_{n-1}| = \frac{1}{2^2}|dx_{n-2}| = \cdots = \frac{1}{2^n}|dx_0|$$

- Number of steps for accuracy ϵ

$$\frac{1}{2^n}|dx_0| \leq \epsilon$$

solve for

$$n \geq \log_2 \left[\frac{|dx_0|}{\epsilon}\right] = \frac{\log_{10}\left[\frac{|dx_0|}{\epsilon}\right]}{0.3010\ldots}$$

Example: $|dx_0| = 0.1$ and $\epsilon = 10^{-6}$ requires $n \geq 17$

- General definition of convergence rate

$$|dx_n| \simeq C_F\,|dx_{n-1}|^{\alpha}\ ,$$

where *order of convergence* $(= \alpha\ )$ and *convergence factor* $(= C_F)$

- For bisection $\alpha = 1$ – convergence is *linear* (rather slow)

Secant method

- Use *secant* (latin *secare* cut) line

$$s(x) = f(x_1) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_1)$$

- Next point determined by

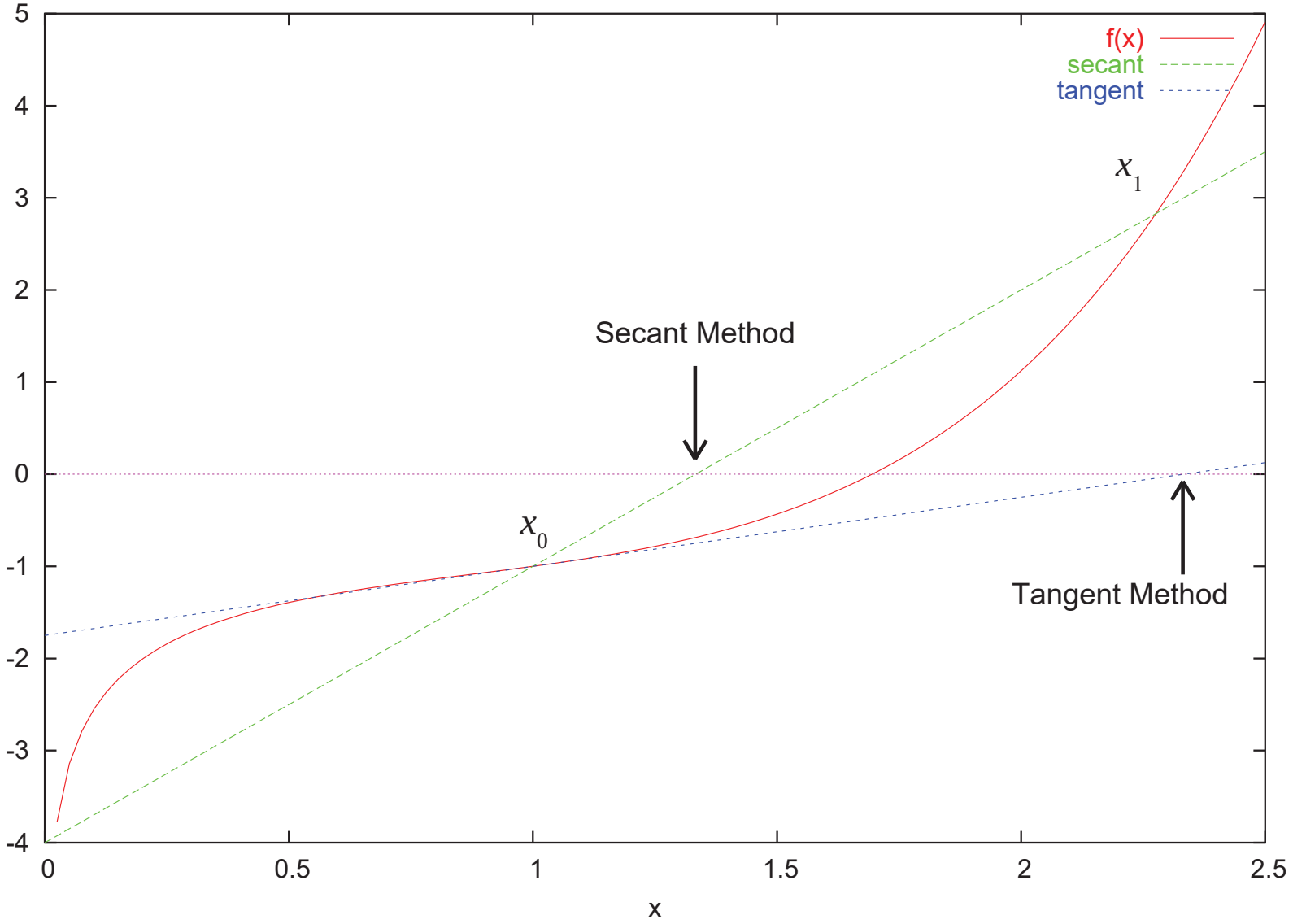
$$s(x_{\rm new}) = 0 \quad \Rightarrow \quad x_{\rm new} = x_1 - (x_1 - x_0)\frac{f(x_1)}{f(x_1) - f(x_0)} \equiv x_1 + dx_{\rm new} \ .$$

- Secant algorithm
  - ◇ choose x_0, x_1 near root – need not bracket
  - ◇ if $x(x_0) = f(x_1)$ algorithm fails – abort and retry
  - ◇ otherwise replace $x_0 \leftarrow x_1$ and $x_1 \leftarrow x_{\rm new}$
  - ◇ repeat above steps until $|x_0 - x_1| \leq \epsilon$ (desired accuracy) or $f(x_{\rm new}) = 0$

- Convergence rate is the famous Golden Ratio

$$\alpha = \frac{1+\sqrt{5}}{2} = 1.618033988\ldots$$

- Convergence is *supelinear* – much faster than linear



- The following program finds the root of $e^x \log(x) - x^2$

`secant.cpp`

```
#include <cmath>
#include <iostream>
#include <iomanip>
using namespace std;

double f(double x) {
    return exp(x) * log(x) - x * x;
}

void print(int step, double x, double dx) {
    cout.setf(ios::right, ios::adjustfield);
    cout << " " << setw(4) << step << "    ";
    cout.precision(15);
    cout.setf(ios::left, ios::adjustfield);
    cout.setf(ios::showpoint | ios::fixed);
    cout << setw(20) << x << "  " << setw(20) << dx << '\n';
}

int main() {

    cout << " Secant search for root of exp(x)*log(x) - x*x\n"
         << " ---------------------------------------\n"
         << " Enter guesses x_0, x_1, and desired accuracy: ";
    double x0, x1, acc;
    cin >> x0 >> x1 >> acc;
```

```
    cout << " Step             x                    dx\n"
         << " ----    ------------------    ------------------\n";
    int step = 0;
    double dx = x1 - x0;
    print(step, x1, dx);
    double f0 = f(x0);
    while (abs(dx) > abs(acc)) {
        double f1 = f(x1);
        if (f0 == f1) {
            cerr << " Secant horizontal ... try again!\n";
            return 1;
        } else {
            dx *= - f1 / (f1 - f0);
            x0 = x1;
            f0 = f1;
            x1 += dx;
        }
        ++step;
        print(step, x1, dx);
    }
}
```

Newton's tangent method

- Requires only one intial guess x_0 sufficiently close to root

- Construct tangent (latin *tangere* touch) line

$$t(x) = f(x_0) + f'(x_0)(x - x_0)$$

- Next point is intersection with x axis

$$x_{\text{new}} = x_0 - \frac{f(x_0)}{f'(x_0)} \equiv x_0 + dx$$

- Convergence rate $\alpha = 2$ is *quadratic* – very fast
- Requires derivative $f'(x)$ either analytically or as finite difference

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

- The following program finds the root of $e^x \log(x) - x^2$

tangent.cpp

```
#include <cmath>
#include <iostream>
#include <iomanip>
using namespace std;
```

```
double f(double x) {
    return exp(x) * log(x) - x * x;
}

double fPrime(double x) {
    return exp(x) * (log(x) + 1/x) - 2 * x;
}

void print(int step, double x, double dx) {
    cout.setf(ios::right, ios::adjustfield);
    cout << " " << setw(4) << step << "    ";
    cout.precision(15);
    cout.setf(ios::left, ios::adjustfield);
    cout.setf(ios::showpoint | ios::fixed);
    cout << setw(20) << x << "  " << setw(20) << dx << '\n';
}

int main() {

    cout << " Tangent search for root of exp(x)*log(x) - x*x\n"
         << " ---------------------------------------\n"
         << " Enter guess x_0, and desired accuracy: ";
    double x, acc;
    cin >> x >> acc;
```

```
    cout << " Step              x                    dx\n"
         << " ----    ------------------    ------------------\n";
    int step = 0;
    double dx = 1;
    print(step, x, dx);
    while (abs(dx) > abs(acc)) {
        double slope = fPrime(x);
        if (slope == 0) {
            cerr << " Tangent horizontal ... try again!\n";
            return 1;
        } else {
            dx = - f(x) / slope;
            x += dx;
        }
        ++step;
        print(step, x, dx);
    }
}
```

Computational Physics Library

- The root finding routines outlined above are available in the Computational Physics Library, see the header file findroot.hpp