Lecture I

Helmut G. Katzgraber

TEXAS A&M UNIVERSITY

First steps towards Monte Carlo…

Outline

- Random numbers:
  - True vs pseudo
  - Recommended generators
  - Libraries
- Application: Random walks
  - Standard random walk
  - Simple simulation techniques
- Monte Carlo integration
  - Recap: Traditional schemes
  - Simple sampling
  - Markov chain sampling
  - Importance sampling
- Statistical mechanics
  - Concepts in a (pea)nut shell
  - Phase transitions
  - A toy model: the Ising model
  - Finite-size scaling

Literature used

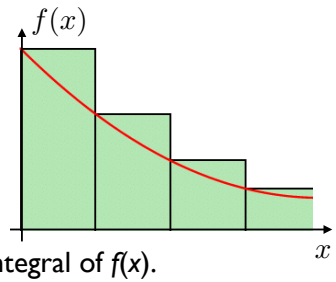
- Random numbers:
  - "*Random numbers: A survival guide*" – Mertens (arXiv:0905.4238)
  - "*Random Numbers in Scientific Computing*" – HGK (arXiv:1005.4117)
  - "*Random Numbers*" – Knuth ("*Art of Scientific Computing*" Volume 2).
  - "*Numerical Recipes*" – Press *et al.*
  - "*Numerical Analysis*" – Timothy Sauer
- Monte Carlo:
  - "*Introduction to Monte Carlo Algorithms*" – Krauth
  - "*Introduction to Monte Carlo Methods*" – HGK (arXiv:0905.1629)
  - "*Monte Carlo Methods in Statistical Physics*" – Newman & Barkema
  - "*Optimization Algorithms in Physics*" – Hartmann & Rieger
  - "*Scientific Programming*" – Zachary

Motivation: Integration schemes…

Example: Rectangle rule

- Goal:
  - Compute the following one-dimensional integral: $I = \int_a^b f(x)dx$
- Solution: Newton–Cotes–like scheme
  - Partition [*a*,*b*] into *M* slices of width $h = (b-a)/M$.
  - Perform a *k*-th order interpolation.
  - Approximate the integral as a sum.
  - Rectangle rule

$$I \approx \sum_{l=0}^{M-1} hf(x_l)$$

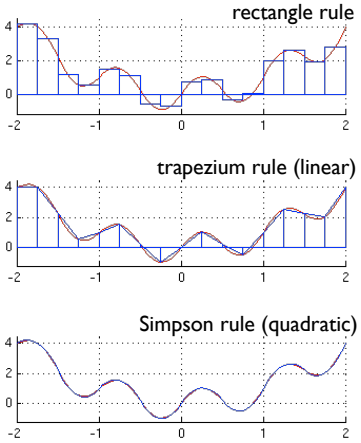


  - For $M \to \infty$ the sum converges to the integral of *f*(*x*).
  - Error ~ *O*(*h*).

What happens in high space dimensions?

Error of traditional integration schemes

- Error:
  - Rectangle rule Error ~ M^{-1}
  - Trapezium rule Error ~ M^{-2}
  - Simpson rule Error ~ M^{-4}



- What happens in *d* space dimensions?
  - One-dimensional error scales ~ M^{-k}.
  - *d*-dimensional error scales ~ $M^{-k/d}$.
  - For large *d*, convergence is slow!
- Further technicality: nested ranges

$$I = \iiint dx\,dy\,dz\,f(x,y,z) = \int_{x_1}^{x_2} dx \int_{y_1(x)}^{y_2(x)} dy \int_{z_1(x,y)}^{z_2(x,y)} dz\,f(x,y,z)$$

Can this be an issue for physics applications?

- Yes!
  - Most integration schemes *fail* for high-dimensional integrals.
  - The phase space of physical problems is generally huge.

    Examples:

    - N classical particles $d = 6N$ (3 coordinates, 3 momenta).
    - N classical Ising spins $d = 2^N$ (can take values of ± 1).
    - N spin-S quantum spins $d = (2S + 1)^N$

- Solution:
  - Find a method where the error is independent of the dimension d.
  - However, for that we need to first understand random numbers…

Random Number Generation

Poll: Which generators do you use?

- drand48( )?
- r250( )?
- r1279( )?
- Mersenne Twister?
- Random123?
- http://random.org?
- /dev/random?

- Home brew?
- I do not know but it works…
- Never used one.

Desired properties of a RNG in simulations

- Simple facts:
  - Modern computers can perform ~10^9 operations per second.
  - A typical Monte Carlo simulation needs ~10^{14} random numbers!

- Desired properties for a RNG:
  - The numbers should be "as random as possible" and not repeat.
  - It should be fast.

- Problem:
  - Excellent RNGs are typically slow, poor RNGs fast…

- Solution:
  - Use the right RNG for the right problem (*True* vs *Pseudo* RNGs).

True Random Number Generators

True RNGs

- Pros:
  - True random numbers are generated.
  - No correlations in the sequence, the numbers are unique.
- Cons:
  - Generally slow (not useful for physical simulations).
  - Because the numbers are unique, code debugging is difficult.
- Applications:
  - Cryptography.
  - Seeding of large-scale simulations (or PRNGs).

Implementations of TRNGs

- General concept:
  - Exploit unpredictable processes in nature.
  - Add post-processing to prevent any bias.

quantis PCI card

- Selected hardware implementations:
  - Coin flipping, rolling of dice, roulette, … (slow: 32 tosses for 1 int).
  - Random physical processes:
    - Noise (thermal, atmospheric, …)
    - Radioactive decay
  - Quantum interference (idQuantique)
  - Human game-play entropy in MMO games
  - LavaRand by SGI (pictures of patterns for entropy).
  - Unix `/dev/random` collects noise from device drivers.

LavaRand

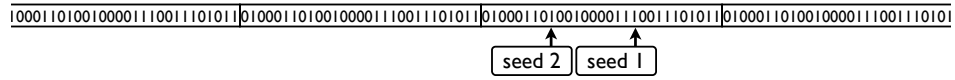
Pseudo Random Number Generators

Pseudo RNGs

- Pros:
  - Generally fast.
  - Do not require special hardware and are therefore portable.
  - Sequences can be reproduced for debugging.
- Cons:
  - Finite sequence lengths (however, some are very long).
  - The numbers can be correlated.
- Applications:
  - Computer simulations.
  - Statistical data analysis.
  - Applications that are not mission critical ("*Who pays for the beer?*").

Implementations of typical PRNGs

- General concept:
  - PRNGs are based on an algorithm and are therefore *deterministic*.



- Mathematical structure:

$$x_i = f(x_{i-1}, x_{i-2}, \ldots, x_{i-n})$$

  - The initial *n* numbers needed are called the *seed block*.
  - Goal: find a function *f* that produces "very" random numbers.
  - The seed determines the sequence of random numbers:
    - It is crucial you carefully seed your simulation.
    - Do not seed your simulation too often to prevent overlaps.
  - Some generators use the modulo operator to randomize the sequence. This, in turn, limits the length of the sequence.

PRNGs you should use

Lagged Fibonacci Generators

- The name comes from the similarity to the Fibonacci series:

$$x_i = x_{i-1} + x_{i-2} \quad \longrightarrow \quad 1, 1, 2, 3, 5, 8, 13, 21, \ldots \qquad (x_0 = x_1 = 1)$$

- Definition:

$$x_i = (x_{i-j} \odot x_{i-k}) \bmod m\,, \qquad 0 < j < k$$

- Properties:
  - $\odot$ represents either addition, multiplication, or XOR.
  - Requires a seed block of size *k* (has to be built carefully!).
  - $m = 2^M$, with $M = 32$ or 64.
  - Very fast (can be vectorized and pipelined).
  - In general, passes all statistical tests known.
  - Very long periods: $\rho(\oplus) = 2^{k-1}2^{M-1}$

$$\rho(\otimes) = 2^{k-1}2^{M-3}$$

Lagged Fibonacci generators contd.

- The quality of the generator (and length of period) highly depends on the values of *j* and *k*. The larger the lags, the better.
  - Note: the XOR version is known as *two-tap generalized shift register*.
  - Additive choices: (rarely used) $\{55, 24, \oplus\}$ $\{607, 273, \oplus\}$ $\{2281, 1252, \oplus\}$ $\{9689, 5502, \oplus\}$
  - Multiplicative choices: (commonly used) $\{250, 103, \otimes\}$ ✗ $\{1279, 418, \otimes\}$ ✓
- r1279 (multiplicative with *k* = 1279, *j* = 418):
  - Period of approximately 10^{394}.
  - Passes all statistical tests.
  - Part of the GNU Scientific Libraries (GSL).

Other commonly used PRNGs

- Mersenne Twister:
  - Generalized feedback shift register PRNG.
  - The period is given by a Mersenne prime: $M_n = 2^n - 1 \quad n \in \mathbb{N}$
  - The implementation `mt19937( )` has period ~ 10^{6001}!
  - Probably best and fastest generator at this time (passes all tests).
  - Part of many scientific software packages and libraries (R, Matlab, …).
  - Easy to checkpoint.
- WELL (Well equidistant long-period linear) generators:
  - Based on the Mersenne Twister, but with better bit mixing.
- Random123 (DE Shaw):
  - Counter based, radically different. Very good for certain applications.

Only use library implementations!

Library implementations of PRNGs

- It is *not* recommended to implement one's own PRNG.
- Many libraries have *optimized* implementations.
- Examples:
  - Boost Libraries: Generic implementations in C++.
  - GNU Scientific Library (GSL): Implementations in C.
  - TRNG: Implementations for parallel simulations.
  - Numerical Recipes: Implementations in many different languages.
    license issues
- Structure/contents of PRNG libraries:
  - Uniform PRNGs (r1279, Mersenne Twister, LCG, …).
  - Distribution functions (Gaussian, Gamma, Poisson, …).
  - Tests.

Example: Boost Libraries

- Definition of generators:

```
boost::lagged_fibonacci1279 rng1;  // r1279
boost::mt19937 rng2;               // Twister
boost::minstd_rand0 rng3;          // LCG
```

- Definition of distributions:
  - Uniform: $r = a + (b - a)u$

    ```
    boost::uniform_int<int> dist1(a,b);
    boost::uniform_real<double> dist2(a,b);
    ```
  - Exponential: $q(y) = a\exp(-ay)$

    ```
    boost::exponential_distribution<double> dist3(a);
    ```
  - Normal:

    $$q(y) = \frac{1}{\sqrt{2\pi}}\exp(-y^2/2)$$

    ```
    boost::normal distribution<double> dist4(mu,sigma);
    ```

Boost Libraries: Adding it all up…

call boost headers

uniform double distribution

engine: Mersenne Twister

create generator

seeding the generator

call the generator

```
1  #include <boost/random.hpp>
2
3  int main (void)
4  {
5
6  // define distribution
7  boost::uniform_real<double> dist(0.,1.);
8
9  // define the PRNG engine
10 boost::mt19937 engine;
11
12 // create a normally-distributed generator
13 boost::variate_generator<boost::mt19937&,
14     boost::normal_distribution<double> >
15     rng(engine,dist);
16
17  // seed it
18  engine.seed(1234u);
19
20 // use it
21 for (int i = 0; i < 100; i++){
22     std::cout << rng() << "\n";
23 }
```

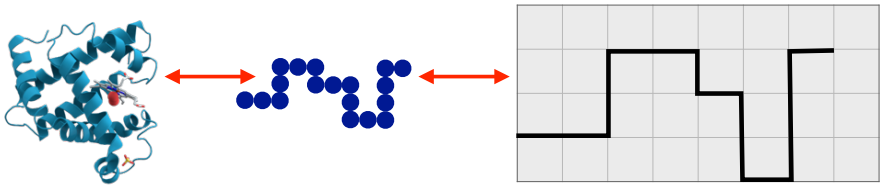
Final recommendations

- Remember:
  - Test your simulation code with *two* different PRNGs.
  - Ensure provenance: Store information about the PRNG & seed.
  - Use trusted and well-tested implementations. Avoid home-brew.
  - Know your PRNG's limits!
    - How long is the period?
    - Are there problems with certain applications?
    - Are there correlations?
  - Be careful when you use PRNGs in parallel simulations.
- Recommended generators:
  - Mersenne Twister (mt19937).
  - Lagged Fibonacci (r1279).

Random numbers – A first application…

Random walks

- Motivation:
  - Likely the simplest physical application of random numbers.
- Applications:
  - *Economics*: used to model shares prices.
  - *Genetics*: used to simulate genetic drift in genetic populations.
  - *Physics*: simplified models for Brownian motion.
  - *Biology*: motile bacteria typically perform random walks.
  - *Polymers*: simple polymer/protein properties can be modeled.



What is the typical size of a random walk?

- Simplest setup:
  - The walk can cut across itself.
  - There are no interactions.
  - The angles are random and uncorrelated.
- Model:
  - The vectors $\mathbf{r}_i$ connecting the steps can be treated as random.
  - The vectors $\mathbf{r}_i$ connecting the steps are uncorrelated.
- What does this mean? Averaging over multiple configurations yields…

$$\langle \mathbf{r}_i \rangle = 0 \qquad \langle \mathbf{r}_i \cdot \mathbf{r_j} \rangle = \langle \mathbf{r}_i \rangle \cdot \langle \mathbf{r_j} \rangle = 0 \quad i \neq j \qquad \langle \mathbf{r}_i^2 \rangle = a^2$$

What is the typical size? contd.

- Compute the vector **R** between beginning and end:

$$\mathbf{R} = \sum_{i=1}^{N} \mathbf{r}_i$$

- Average over many configurations:

$$\langle \mathbf{R} \rangle = \sum_{i=1}^{N} \langle \mathbf{r}_i \rangle = 0$$

$$\langle \mathbf{R}^2 \rangle = \sum_{i=1}^{N} \sum_{j=1}^{N} \langle \mathbf{r}_i \cdot \mathbf{r}_j \rangle = \sum_{i=1}^{N} \langle \mathbf{r}_i^2 \rangle + \sum_{i \neq j} \langle \mathbf{r}_i \cdot \mathbf{r}_j \rangle = N a^2$$

- The typical linear size of the walk is therefore $\sqrt{\langle \mathbf{R}^2 \rangle} \propto a\sqrt{N}$.
- Note:
  - This expression is independent of the space dimension *d*!
  - Random walks are *D* = 2–dimensional fractal objects with $N \sim \ell^D$.

Modeling *d*-dimensional random walks

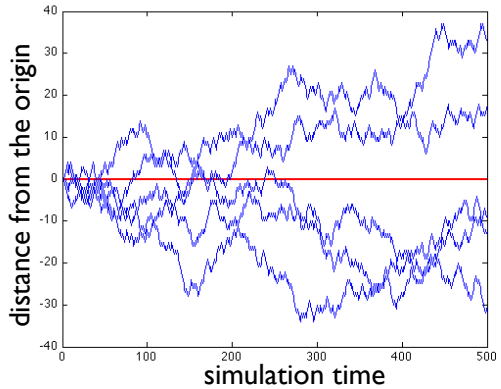
- Algorithm (*d*-dimensional lattice):
  - Place a *walker* on the origin.
  - Draw a uniform random number in [1,2*d*].
  - Move the walker to the new position.
  - Treat this new position as the new origin.
  - Iterate…

- How to determine the typical distance:
  - Perform *N* steps.
  - Measure the geometric distance from the origin.
  - Average over many runs.
  - Vary *N* and verify that $\sqrt{\langle \mathbf{R}^2 \rangle} \propto a\sqrt{N}$.

Example: One-dimensional random walk

- Simple algorithm:

```
while(i < N){
   if(rand_bit()){
      dist++;
   }
   else{
      dist--;
   }
   i++;
   x2_ave += dist*dist;
}
```



- Note:
  - The above code snippet is for one run of *N* steps.
  - To compute error bars, you need to average over runs.
  - Higher dimensions can be implemented with a case statement.

Manhattan?

Monte Carlo!

Historical motivation

- Manhattan Project at Los Alamos Natl. Lab:
  - Simulations of nuclear weapons.
  - The term "Monte Carlo" was coined 1940 by Ulam, Fermi, von Neumann, Metropolis and others thinking of casinos when using random numbers.

Ulam Feynman von Neumann

- Monte Carlo method:
  - One of the most important methods in computational physics.
- Idea:
  - Randomly sample a volume in a *d*-dimensional space to obtain an estimate of an integral at the price of a statistical error.
  - This works best when the problem has a *large space dimension*.

Monte Carlo integration

Motivation

- Recall…
  - Most integration schemes *fail* for high-dimensional integrals.
  - The phase space of physical problems is generally huge.

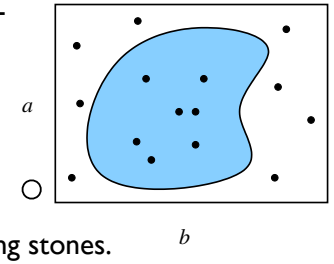
  Examples:

  - N classical particles — $d = 6N$ (3 coordinates, 3 momenta).
  - N classical Ising spins — $d = 2^N$ (can take values of ±1).
  - N spin-S quantum spins — $d = (2S + 1)^N$

- Solution:
  - A method where the error is independent of the space dimension…

Simple sampling Monte Carlo

- So far: Sample the function $f(x)$ at evenly-spaced points.
- Now: Sample $f(x)$ at random points.



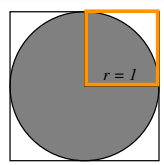
- Analogy:
  - Determine the area of a pond by throwing stones.
  - Enclose the pond by a known area $A = ab$.
  - *Randomly* throw stones in the rectangular area.

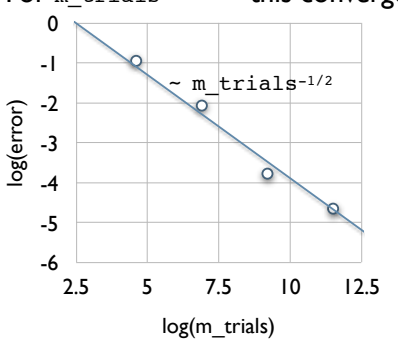
$$A_{pond} = A\, N_{in}/N_{tot}$$

  - We obtain a *simple sampling* statistical estimate of A_{pond}.
  - Note: get lots of Kölsch to properly randomize the process…

Simple sampling Monte Carlo contd.

- How can we compute π using Monte Carlo integration?
  - Integrate part of the unit circle $A_\circ = \pi r^2$ enclosed by a box of unit side length $A_\square = r^2$.
  - For `m_trials` $\to \infty$ this converges to π.



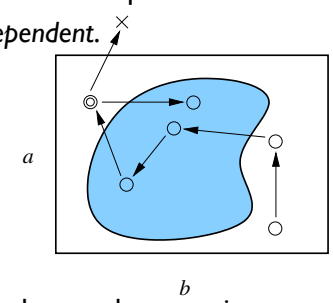



```
algorithm simple_pi
    initialize n_hits      0
    initialize m_trials    10000
    initialize counter     0

    while(counter < m_trials) do
        x = rand(0,1)
        y = rand(0,1)
        if(x**2 + y**2 < 1)
            n_hits++
        fi
        counter++
    done

return pi = 4*n_hits/m_trials
```

  - Error ~ $M^{-1/2}$ *independent* of d.

Markov chains & pebbles

- So far:
  - The pebbles are independent and thrown from one place.
  - π estimate: the random numbers are *independent*.



- Problem:
  - If the pond is large, we cannot reach all corners from one point only!
- Solution:
  - Use a bucket of pebbles. Throw the first, relocate, throw again, …
  - If you throw outside the rectangle, get the pebble and place it on your current location. The move is rejected, the last one counted twice. This ensures the *Markov chain* is reversible (*detailed balance*).

Markov chains: estimating π

- Start at {0,0} and "wander" around phase space.
- Select p carefully:
  - *too small*: slow convergence.
  - *too large*: many rejections.
  - ensure ~50% of the moves are accepted.

```
algorithm markov_pi
    initialize n_hits      0
    initialize m_trials    10000
    initialize x           0
    initialize y           0
    initialize counter     0

    while(counter < m_trials) do
        dx = rand(-p,p)
        dy = rand(-p,p)
        if(|x + dx| < 1 and |y + dy| < 1)
            x = x + dx
            y = y + dy
        fi
        if(x**2 + y**2 < 1)
            n_hits++
        fi
        counter++
    done

return pi = 4*n_hits/m_trials
```

Simple sampling vs Markov chain sampling?

- Simple sampling Monte Carlo:
  - Advantage: No correlations between states (pebbles).
  - Disadvantage: At every step a new state from a given distribution needs to be generated *from scratch*.
- Markov chain Monte Carlo:
  - Disadvantage: There are (auto)correlations between states. Uncorrelated measurements are only possible every *autocorrelation-time* steps.
  - Advantage: Slightly randomly change the existing state to generate a new one from a given distribution.
- So… What do we do?
  - Surprisingly, it is easier to sample from an existing distribution.

Back to simple sampling of integrals…

- Example: $f(x) = x^n \quad (n > -1) \quad \rightarrow \quad I = \int_0^1 f(x)dx$
  - The integral is given by:

$$I \approx \frac{1}{M}\sum_i^M f(x_i)$$

with x_i random in [0,1].

  - Estimating the error: variance

$$\delta I = \sqrt{\frac{\mathrm{Var} f}{M-1}} \sim M^{-1/2}$$

$$\mathrm{Var} f = \langle f^2 \rangle - \langle f \rangle^2$$

moments:

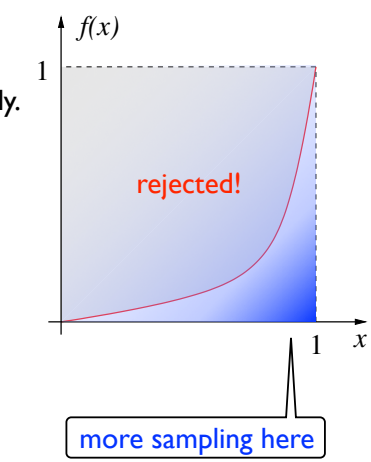
$$\langle f^k \rangle = \int_0^1 [f(x)]^k dx \approx \frac{1}{M}\sum_i^M [f(x_i)]^k$$

```
algorithm simple_integrate
    initialize integral    0
    initialize m_trials    10000
    initialize counter     0

    while(counter < m_trials) do
        x = rand(0,1)
        integral += x**n
        counter++
    done

return integral/m_trials
```

```
integral_sq += x**(2n)
```

Simple sampling: When does it fail?

- Problem:
  - $n \sim -1$ and $n \gg 1$: Var(*f*) is large.
  - The interval [0,1] is sampled uniformly.
  - The error converges slowly.
- Solution:
  - Select the random numbers such that places of *f*(*x*) with a larger support are visited more frequently.



Importance sampling

- When the variance of $f(x)$ is large, the error is also large.

- Solution:
  - Produce random numbers that more efficiently sample the area.
  - Generate random numbers according to $p(x)$ with
    - $p(x)$ close to $f(x)$
    - $p(x)$-distributed random numbers are easy to generate.
  - We obtain:

$$\langle f \rangle = \langle f/p \rangle_p = \int_0^1 \frac{f(x)}{p(x)} p(x) dx \approx \frac{1}{M} \sum_i^M \frac{f(y_i)}{p(y_i)}$$

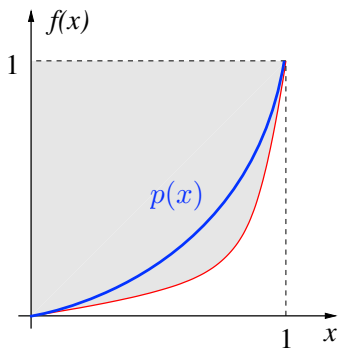
    - Notation: $\langle \cdots \rangle_p$ represents an average over p-distributed numbers and y_i are p-distributed.
  - The error is now $\mathrm{Var}(f/p)$ which is much smaller if $f(x) \sim p(x)$!

Importance sampling contd.

- Example: $f(x) = x^n \quad (n > -1)$
  - Select $p(x) \sim x^l$ with $l \geq n$
  - Power-law distributed random numbers y can be obtained from uniform numbers x via

$$y(x) = x^{1/(\ell+1)} \qquad l > -1$$

  (distribution inversion)



- We have now all ingredients to simulate a physical system:

  Markov chains + importance sampling ⟶ Metropolis algorithm

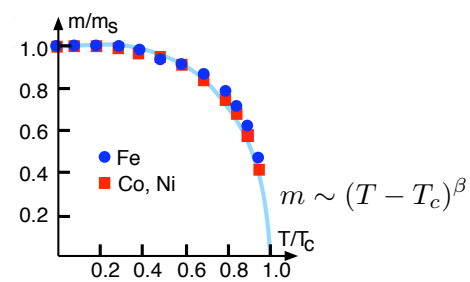
But first…

Statistical mechanics primer

Focus: Magnetic systems. Why?

- They are far easier to simulate than systems of interacting particles.
- Many nontrivial analytical results for some systems (e.g., 2D Ising model).
- Best understood models that display phase transitions.
- Simple models can describe complex materials extremely well.
  Example: 3D Heisenberg ferromagnet.

3D Heisenberg	β
Fe	0.34(4)
Ni	0.378(4)
CrB_3	0.368(5)
EuO	0.36(1)
Mean field	0.5
Monte Carlo	0.364(4)



Why statistical mechanics?

- Problem:
  - Systems of *N* particles with *N* large are hard to treat.
  - Certain types of systems have emergent collective behavior that the individual constituents do not have (e.g., phase transitions).
- Setup:
  - Consider a system of *N* entities described by a Hamiltonian *H*.
  - The system is described by a state vector $\bar{s} = \{s_1, \ldots, s_N\}$.
  - The partition function for the system is given by

$$\mathcal{Z} = \sum_s \exp[-\mathcal{H}(s)/kT]$$

    where *k* is the *Boltzmann constant* and *T* a temperature.
  - Physically measurable quantities can be computed from *Z*!

Observables

- Definition (*observable*): The expectation value of any measured quantity *O* by performing a trace over the partition function *Z*.

  see K. Huang book (87)
  - At a fixed temperature $\langle \mathcal{O} \rangle = \frac{1}{\mathcal{Z}} \sum_s \mathcal{O}(s) e^{-\mathcal{H}(s)/kT}$ sum over *all* states

$$\text{with} \quad \mathcal{Z} = \sum_s \exp[-\mathcal{H}(s)/kT]$$

  - The partition function *Z* normalizes the equilibrium *Boltzmann distribution*:

$$\mathcal{P}_{\text{eq}}(s) = \frac{1}{\mathcal{Z}} e^{-\mathcal{H}(s)/kT}$$

  - Note: It is this distribution we will statistically sample using Monte Carlo simulations.

Selected thermodynamic quantities

- Internal energy: $E = \langle \mathcal{H} \rangle = \partial_\beta \ln Z$
- Free energy: $F = -kT \ln Z = E - TS$

$$\beta = \frac{1}{kT}$$

- All thermodynamic quantities are related to *F* or *Z*:
  - Magnetization: $M = \partial_h F$
  - Specific heat: $C = \partial_T E = -T \partial_T^2 F = \beta^2(\langle \mathcal{H}^2 \rangle - \langle \mathcal{H} \rangle^2)$
  - Susceptibility: $\chi = \partial_h M = -\partial_h^2 F$
    $= \beta(\langle M^2 \rangle - \langle M \rangle^2)$
  - Entropy: $S = -\partial_T F = -k \langle \ln \mathcal{P}(s) \rangle$
  - …
- Note: *h* represents the magnetic field. *k* = 1 in the future.

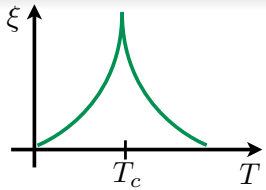
Critical behavior in magnetic systems

Continuous phase transitions (state change)

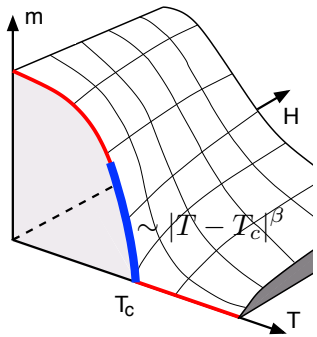
- At a *continuous* ("*2nd order*") phase transition, the correlation length diverges:

$$\xi \sim |T - T_c|^{-\nu}$$

ν critical exponent, T_c crit. temperature



- Example: Ising model in *d* = 2
  - $T_c = 2.269\ldots$
  - $\nu = 1$



- Other observables also show criticality:
  - Magnetization $m \sim |T - T_c|^{\beta}$ $\quad \beta > 0$
  - Specific heat $C \sim |T - T_c|^{-\alpha}$
  - …
- I-st order: Phase coexistence and latent heat (not discussed).

Summary of magnetic critical exponents

Exponent	Definition	Description
α	$C_H \sim \lvert t\rvert^{-\alpha}$	specific heat at $H = 0$
β	$M \sim \lvert t\rvert^{\beta}$	magnetization at $H = 0$, $t < 0$
γ	$\chi \sim \lvert t\rvert^{-\gamma}$	isothermal susceptibility at $H = 0$
δ	$M \sim h^{\frac{1}{\delta}}$	critical isotherm
ν	$\xi \sim \lvert t\rvert^{-\nu}$	correlation length
η	$G(r) \sim \lvert r\rvert^{-(d-2+\eta)}$	correlation function

- Note:
  - In the above expressions $t = \dfrac{T - T_c}{T_c}$ and $h = \dfrac{H}{T_c}$.
  - There are relationships between the exponents.
  - Only *two* are needed to fully characterize a system! How?

Some definitions…

- Definition (*critical exponent*):
  - The critical exponent μ of a quantity *f* is defined via

$$\mu = \lim_{t \to 0} \frac{\ln f(t)}{\ln t} \qquad t = \frac{T - T_c}{T_c}$$

  - This means, that close to the transition the quantity *f* is dominated by a nonanalytic part $f(t) \sim t^{\mu}$ for $t \to 0$.
- Definition (homogenous function):
  - A function *f*(*r*) is called *homogenous* if for all values of λ

$$f(\lambda r) = g(\lambda) f(r)$$

  - The function $g(\lambda) \sim \lambda^{p}$ is called the *scaling function*.
  - For more than one space dimension:

$$\lambda f(x_1, x_2, \ldots) = f(\lambda^{y_1} x_1, \lambda^{y_2} x_2, \ldots)$$

Scaling hypothesis & exponent relations

- Scaling hypothesis:
  - The singular part of the free energy *F* is a homogenous function near the phase transition.
  - Furthermore, $f(t,h) = b^{-d} f(b^{y_t} t, b^{y_h} h)$, where *b* is some length scale and $f(t,h) = F(t,h)/V$, with $V \sim b^{d}$ a volume.
- Example derivation of the scaling relations:
  - Let $b = |t|^{-1/y_t}$ Then $f(t,h) = |t|^{d/y_t} f(\pm 1, t^{-y_h/y_t} h)$

$$\sim |t|^{d/y_t} \phi(|t|^{-y_h/y_t} h)$$

  - Recall $M \sim |t|^{\beta}$ for *H* = 0, but also $M = \dfrac{1}{T} \partial_h f|_{h \to 0} \sim |t|^{(d - y_h)/y_t}$
  - It follows: $\beta = \dfrac{d - y_h}{y_t}$

Relationships between exponents contd.

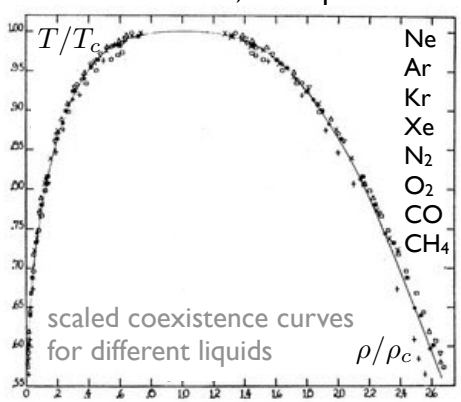
- Following the same approach as before…
  - Specific heat $\alpha = \frac{d}{y_t} - 2$
  - Magnetization $\beta = \frac{d - y_h}{y_t}$
  - Susceptibility $\gamma = \frac{d - 2y_h}{y_t}$
  - Isotherm $\delta = \frac{y_h}{d - y_h}$
- Homogenous form of the correlation function:
  - $G(r) = b^{-2(d-y_h)} G(r/b, b^{y_t} t) \sim |t|^{2(d-y_h)/y_t} \Phi(r|t|^{1/y_t})$
  - From this expression we can derive other "spatial" quantities…

Scaling & Hyperscaling

- Further exponents:
  - Correlation length $G(r) \sim e^{r/\xi} \longrightarrow \nu = \frac{1}{y_t}$
  - Correlation function $\eta = d + 2 - 2y_h$
- Scaling relations (cancel out y_t and y_h…):
  - Rushbrook $\alpha + 2\beta + \gamma = 2$
  - Widom $\beta(\delta - 1) = \gamma$
  - Josephson $2 - \alpha = d\nu$
  - *Has no name* $\gamma = \nu(2 - \eta)$
- Note: Scaling relations with the space dimension *d* are called "*hyperscaling*" relations. They break down for $d \geq d_u$.

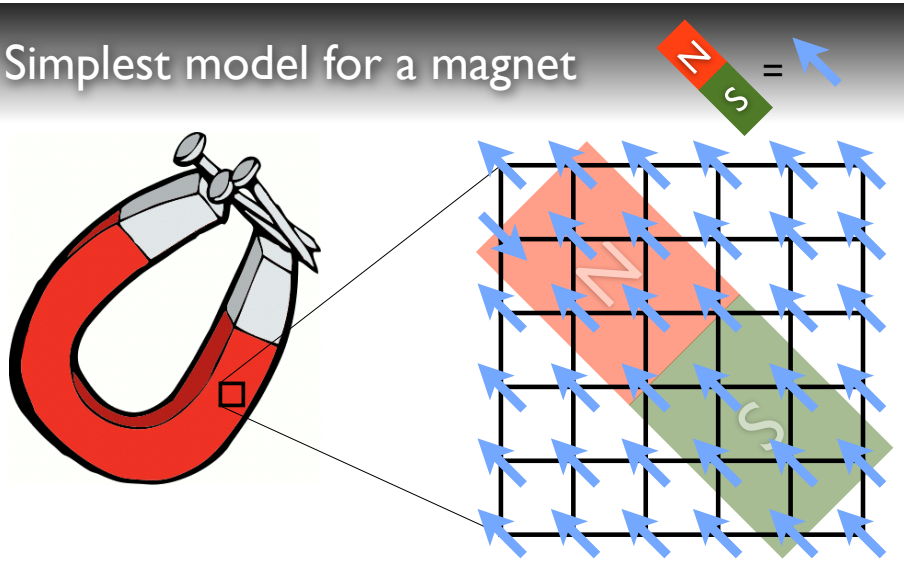
Universality

- Having defined all these exponents… Why should we care?
  - While T_c does depend on the details of the model, the exponents are *universal*.
- What do the critical exponents depend on?
  - Space dimension *d*.
  - Order parameter symmetry.
- Note:
  - For long-range interactions one has to be more careful.
  - Knowing the exponent of a simple system that has the same symmetry properties as a complex material can save years of CPU.



Simplest toy: The Ising model

Simplest model for a magnet



- Imagine the system as made from small mini magnets on a lattice.
- If all mini magnets point in the same direction, the system magnetizes.

Building a model system

- Generic setup:
  - Place N magnetic moments on a d-dimensional lattice.
  - Assume the system is highly anisotropic, i.e., $S_i = \pm 1$ =
  - Most general Hamiltonian:

$$\mathcal{H} = \sum_i H_i S_i + \sum_{i,j} J_{ij} S_i S_j + \sum_{i,j,k} K_{ijk} S_i S_j S_k + \dots$$

coupling to field — 2-body spin-spin — 3-body spin-spin

- Some simplifications:
  - $H_i = H$ We assume a uniform external field
  - $K_{ijk} = 0$ We neglect n-body interactions with $n \geq 3$.
  - $J_{ij} = J$ Only isotropic nearest neighbor interactions.

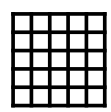
Building a model system contd.

- Is this realistic?
  - Um… No.
  - However, it is astounding that it works so well for so many materials.
- Why all the simplifications?
  - Analytically solvable in $d = 1$ (Ising, $T_c = 0$) and $d = 2$ (Onsager, $T_c > 0$).
  - What about $d = 3$? Out of luck, we must resort to simulations.
  - What about $d \geq 4$? Mean-field theory works and is exact!
- Note:
  - If $J > 0$, we obtain a ferromagnet, if $J < 0$ an antiferromagnet (spins order antiparallel).

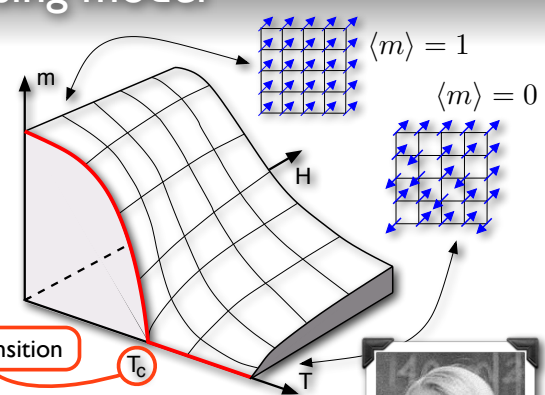
Ferromagnetic Ising model

- Final ingredients:

$= S_i = \pm 1$



$= J_{ij}$

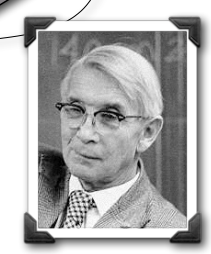


- Hamiltonian:

$$\mathcal{H} = -\sum_{\langle ij \rangle} J_{ij} S_i S_j - H \sum_i S_i \qquad J_{ij} = 1 \;\; \forall i, j$$

- Order parameter (observable):

$$m = \frac{1}{N} \sum_i S_i \text{ (magnetization)}$$



Ernst Ising 1900-98

Mean-field theory

- Idea:
  - Approximate the effects of neighboring spins by introducing a "mean field" and neglecting fluctuation effects

$$S_j \to \underbrace{\langle S_j \rangle}_{\text{mean field}} + \underbrace{(S_j - \langle S_j \rangle)}_{\text{fluctuations}}$$

- Derivation of the partition function:
  - Introduce the mean-field approximation into *H* and neglect quadratic terms:

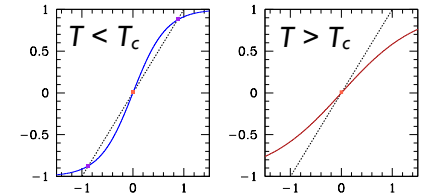
$$\mathcal{H} \approx NdJ\langle S \rangle^2 - (H + 2J\langle S \rangle) \sum_i S_i$$

  - Sum up all one-body terms in the partition function:

$$\mathcal{Z} = e^{-\beta JNd\langle S \rangle^2} \left[2\cosh\left(\beta H + 2\beta\langle S \rangle Jd\right)\right]$$

Mean-field magnetization and T_c

- Recall:
  - $M = \langle S_i \rangle$ and $M = \partial_H T \ln \mathcal{Z}$.



- Expression for the magnetization:
  - It follows: $M = \tanh\left[\beta(H + 2dJM)\right]$
  - When the external field is zero (*H* = 0) the equation has either one solution (*M* = 0) or three solutions. This defines a phase transition.
  - Critical temperature: $T_c^{\mathrm{MF}} = 2dJ$
- Note:
  - Mean-field theory implies a transition for *d* = 1, which is wrong.
  - $T_c(d=2) = 2/\ln(1+\sqrt{2}) \approx 2.26918\ldots$
  - $T_c(d=3) \approx 4.51$

Mean-field vs non-mean-field exponents

- Mean-field exponents:
  - Close to the phase transition $t = \dfrac{T - T_c}{T_c} \ll 1$. We can expand the expression for *M*:

$$H \approx Mt + M^3[1 - t + (1-t)^2 + \ldots]$$

  - *H* = 0 $\longrightarrow$ $M \sim t^{1/2} \sim |T - T_c|^{\beta}$ $\longrightarrow$ $\beta = 1/2$
  - *t* = 1 $\longrightarrow$ $M \sim H^{1/3} = H^{1/\delta}$ $\longrightarrow$ $\delta = 3$
  - *M* = 0 $\longrightarrow$ $\chi \sim 1/t \sim |T - T_c|^{-\gamma}$ $\longrightarrow$ $\gamma = 1$
  - Similarly: $\nu = 1/2$ and $\alpha = 0$.
  - Note: These exponents are valid for any *d* ≥ 4.
- Exact exponents in *d* = 2:

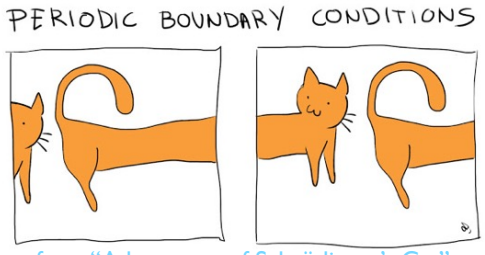
$$\alpha = 0, \quad \beta = 1/8, \quad \gamma = 7/4, \quad \delta = 15, \quad \eta = 1/4, \quad \nu = 1.$$

And for *d* = 3? Simulations…

Finite-size scaling

Finite-size effects in simulations

- Simulations:
  - The accessible system sizes are often very limited.
  - However, we can extract thermodynamic information from the data.
- Approach:
  - Use periodic boundaries to remove finite-size effects.
  - Finite-size scaling.



from "Adventures of Schrödinger's Cat"

- General philosophy:
  - Never "just" simulate a problem.
  - Check first the universality class. Has it been studied before?
  - Use "theory intuition" (finite-size scaling) to extract the information.

Finite-size scaling

- Close to the transition: $t = (T - T_c)/T_c \ll 1$
- In an infinite system the correlation length diverges $\xi \sim |t|^{-\nu}$.
- In a finite system (simulation) the correlation length cannot grow larger than the system size, i.e., $\xi \sim L$.
- We need to apply a finite-size cutoff to the scaling expressions:

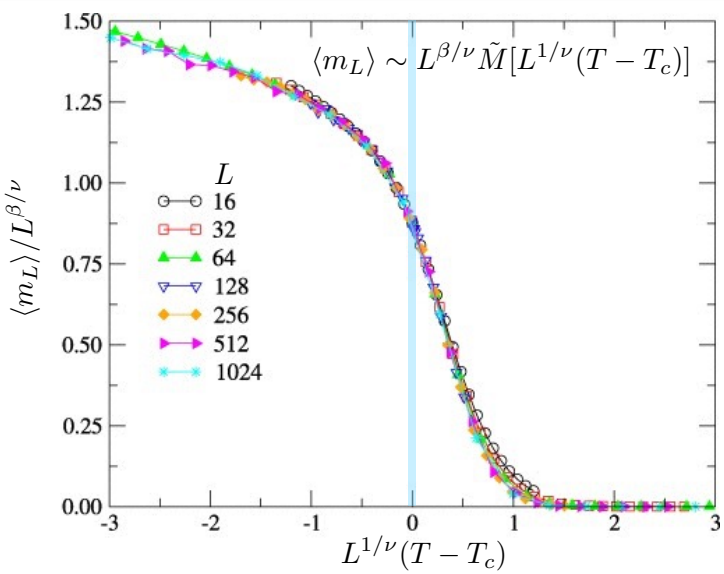
$$\mathcal{O}(t) \sim |t|^y \longrightarrow \mathcal{O}(t, L) \sim |t|^y f(L/\xi)$$

- The scaling function must satisfy:
  - $f(x) \to \text{const.}$ for $x \to \infty$ ensures correct power law for $L \to \infty$.
  - $f(x) \sim x^{y/\nu}$ for $x \to 0$ ensures $\mathcal{O}$ becomes independent of temperature when $\xi \gg L$.
- It follows:

$$\mathcal{O}(t, L) \sim L^{y/\nu} \tilde{f}[L^{1/\nu} t]$$

Example: 2D Ising model magnetization

$\beta = 1/8$
$\nu = 1$
$T_c = 2.269$



Can we do better than that?

- Scaling expression for the magnetization:

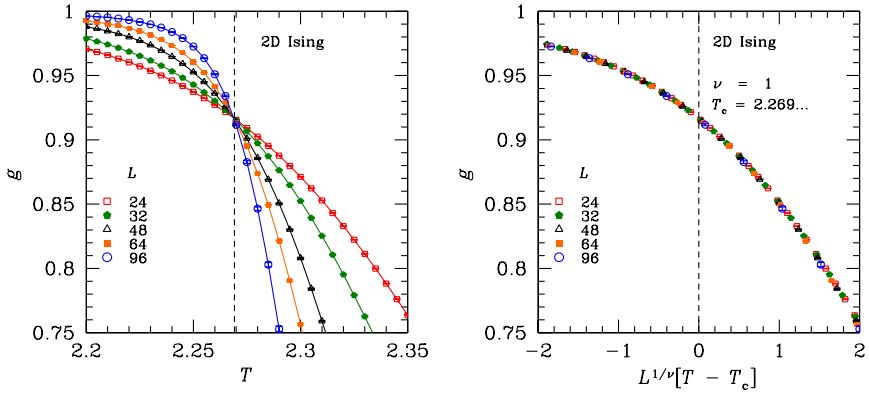
$$\langle m_L \rangle \sim L^{\beta/\nu} \tilde{M}[L^{1/\nu}(T - T_c)]$$

  - We have three unknowns (two exponents) which makes the analysis cumbersome.
- Binder ratio:
  - Use combined quantities to eliminate the metric factors:

$$g = \frac{1}{2}\left[3 - \frac{\langle m^4 \rangle}{\langle m^2 \rangle^2}\right] \sim \tilde{G}[L^{1/\nu}(T - T_c)]$$

  - The function only depends on $L^{1/\nu}(T - T_c)$. At T_c data for different L *should* cross (up to corrections…).
  - One can, in principle, derive many such dimensionless quantities.

Finite-size scaling of the Binder ratio



- The data cross at $T_c(d = 2) = 2.269\ldots$
- If we select the right value of $\nu = 1$ and T_c the data fall onto one curve.