

# **PHYS 5319-001: Math Methods in Physics III**

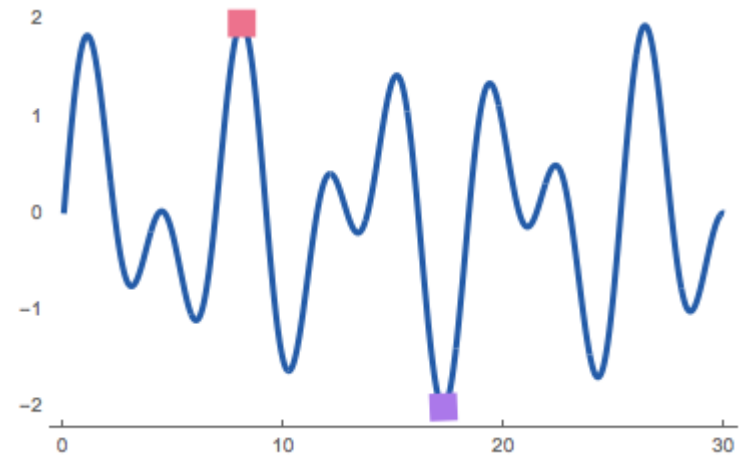
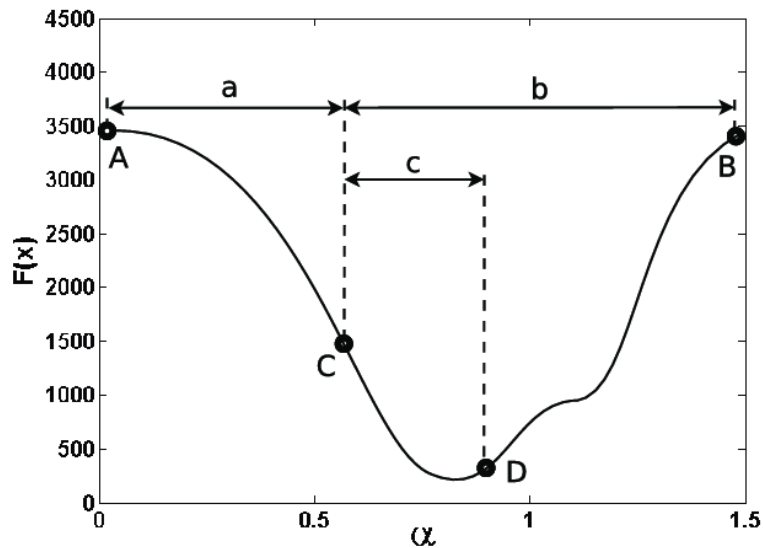
## **Optimization Methods**

---

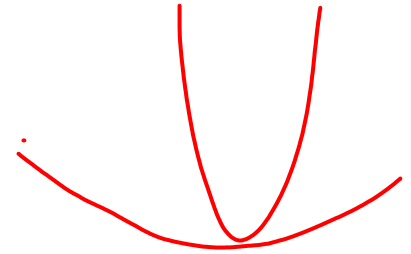
Instructor:	Dr. Qiming Zhang
Office:	CPB 336
Phone:	817-272-2020
Email:	<a href="mailto:zhang@uta.edu">zhang@uta.edu</a>

# Optimization

- Object function with  $N$  parameters  $f(x_1, x_2, \dots)$
- Minima/maxima search on a  $N$ -d space surface
- local vs. global
- We focus on minima here
- Golden section search



# Steepest descent method



- Object function with  $N$  parameters  $f(x_1, x_2, \dots)$  is a surface in  $N$ -dimension space. Imaging it as a potential surface ( $V$ ).

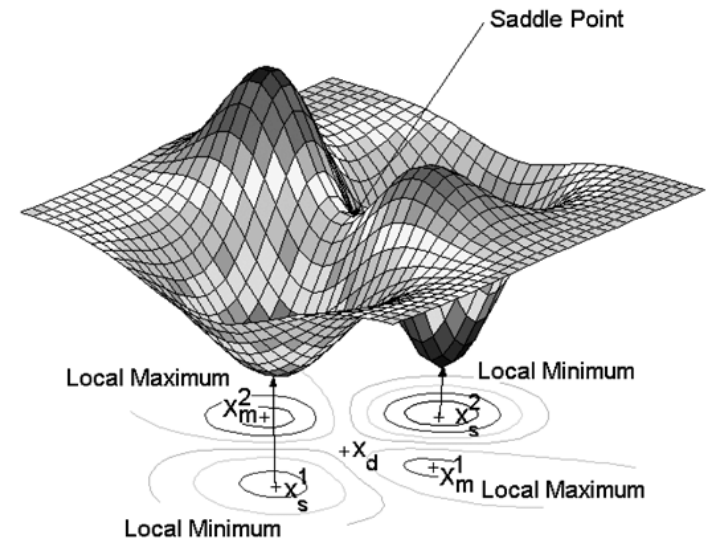
- gradient  $\nabla f = \frac{\partial f}{\partial x_1} \hat{e}_1 + \frac{\partial f}{\partial x_2} \hat{e}_2 + \dots$

- Start at a point  $\mathbf{X}^{(0)}$ , set  $k=0$

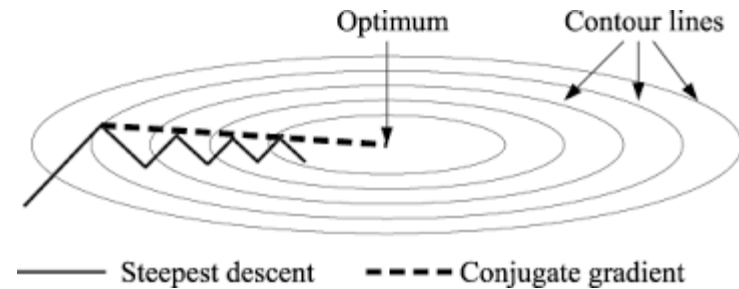
- $\mathbf{X}^{(k+1)} = \mathbf{X}^{(k)} - \nabla f(\mathbf{X}^{(k)})\Delta t$

- Until  $|\nabla f| \leq \epsilon$

- Only reach a local minimum



# Conjugate Gradient method



- Assume (by Taylor series)

$$f(\vec{x}) = f(0) + \sum \frac{\partial f}{\partial x_i} x_i + \frac{1}{2} \sum_{i,j} \frac{\partial^2 f}{\partial x_i \partial x_j} x_i x_j + \dots$$

$$\approx C - \vec{b} \cdot \vec{x} + \frac{1}{2} \vec{x} \cdot \mathbf{A} \cdot \vec{x}$$

- the gradient of the above:  $\nabla f = \mathbf{A} \cdot \vec{x} - \vec{b}$
- Variation along some direction:  $\delta(\nabla f) = \mathbf{A} \cdot (\delta \vec{x})$
- Suppose firstly along  $\mathbf{u}$  (1st use ST) to a minimum but now the proposed is along  $\mathbf{h}$ . The condition that the motion along  $\mathbf{h}$  not spoil our minimization along  $\mathbf{u}$  is just that the gradient stay perpendicular to  $\mathbf{u}$ .

$$0 = \mathbf{u} \cdot \mathbf{A} \cdot \mathbf{h}$$

Use this to determine  $\mathbf{h}$ .

## CG method

Let  $\mathbf{g}_i = -\nabla f(\mathbf{P}_i)$ , the negative gradient at  $\mathbf{P}_i$   
(start  $\mathbf{h}_0 = \mathbf{g}_0$ )

$$\mathbf{g}_{i+1} = \mathbf{g}_i - \lambda_i \mathbf{A} \cdot \mathbf{h}_i$$

$$\mathbf{h}_{i+1} = \mathbf{g}_{i+1} + \gamma_i \mathbf{h}_i$$

The new search direction!

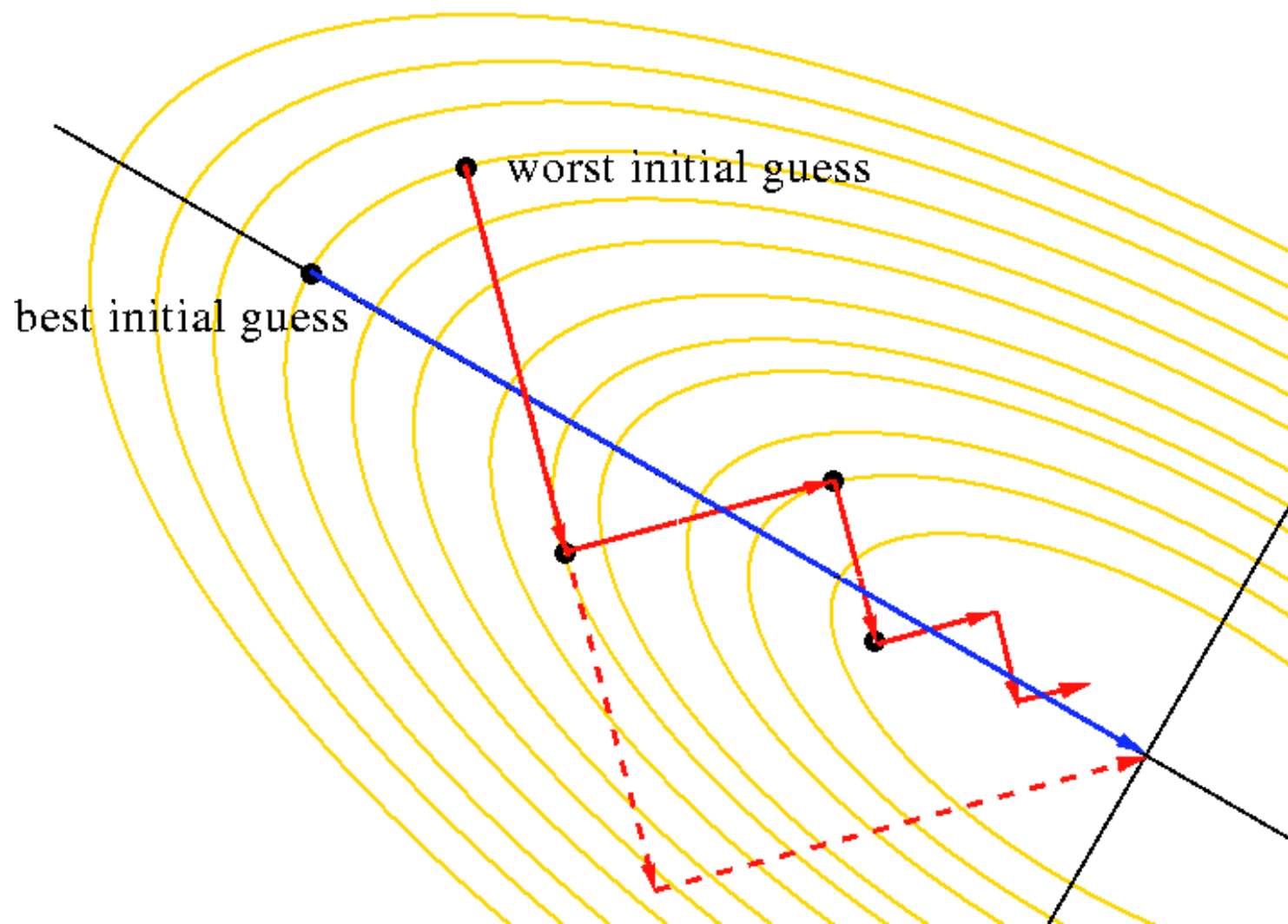
The vectors satisfy the orthogonality and conjugate conditions  $\mathbf{g}_i \cdot \mathbf{h}_j = 0$ ,  $\mathbf{h}_i \cdot \mathbf{A} \cdot \mathbf{h}_j = 0$ ,  $\mathbf{g}_i \cdot \mathbf{h}_j = 0$ ,  $j < i$

The scalars  $\lambda_i$  and  $\gamma_i$  given by

$$\lambda_i = \frac{\mathbf{g}_i \cdot \mathbf{g}_i}{\mathbf{h}_i \cdot \mathbf{A} \cdot \mathbf{h}_i} = \frac{\mathbf{g}_i \cdot \mathbf{h}_i}{\mathbf{h}_i \cdot \mathbf{A} \cdot \mathbf{h}_i}$$

$$\gamma_i = \frac{\mathbf{g}_{i+1} \cdot \mathbf{g}_{i+1}}{\mathbf{g}_i \cdot \mathbf{g}_i}$$

Use Golden Section search along each  $\mathbf{h}_i$  for  $\mathbf{P}_i$




Example code in **MATLAB / GNU Octave** [\[ edit \]](#)

```
function x = conjgrad(A, b, x)
    r = b - A * x;
    p = r;
    rsold = r' * r;

    for i = 1:length(b)
        Ap = A * p;
        alpha = rsold / (p' * Ap);
        x = x + alpha * p;
        r = r - alpha * Ap;
        rsnew = r' * r;
        if sqrt(rsnew) < 1e-10
            break
        end
        p = r + (rsnew / rsold) * p;
        rsold = rsnew;
    end
end
```

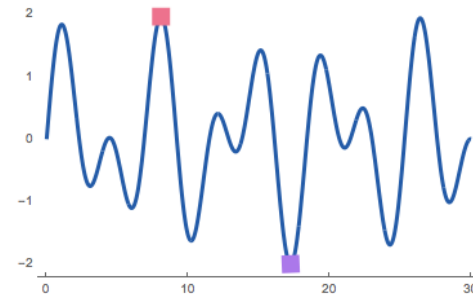
## Conjugate Gradient in Python

 conjgrad.py

```
1  def conjgrad(A, b, x):
2      """
3      A function to solve  $[A]\{x\} = \{b\}$  linear equation system with the
4      conjugate gradient method.
5      More at: http://en.wikipedia.org/wiki/Conjugate\_gradient\_method
6      ===== Parameters =====
7      A : matrix
8          A real symmetric positive definite matrix.
9      b : vector
10         The right hand side (RHS) vector of the system.
11      x : vector
12         The starting guess for the solution.
13      """
14      r = b - np.dot(A, x)
15      p = r
16      rsold = np.dot(np.transpose(r), r)
17
18      for i in range(len(b)):
19          Ap = np.dot(A, p)
20          alpha = rsold / np.dot(np.transpose(p), Ap)
21          x = x + np.dot(alpha, p)
22          r = r - np.dot(alpha, Ap)
23          rsnew = np.dot(np.transpose(r), r)
24          if np.sqrt(rsnew) < 1e-8:
25              break
26          p = r + (rsnew/rsold)*p
27          rsold = rsnew
28      return x
```

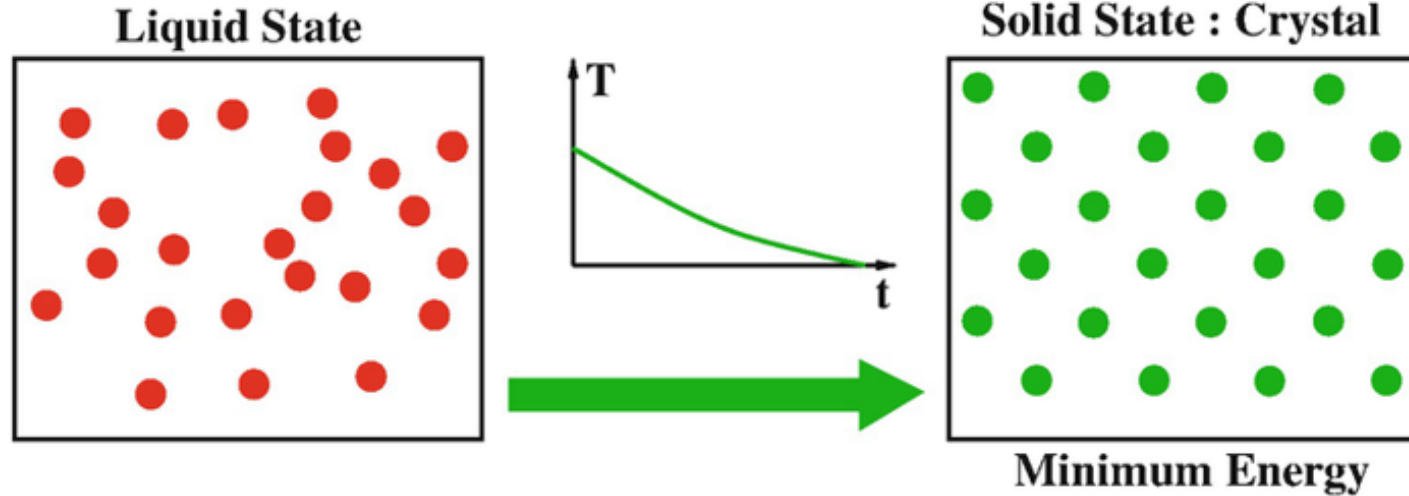


# Simulated Annealing



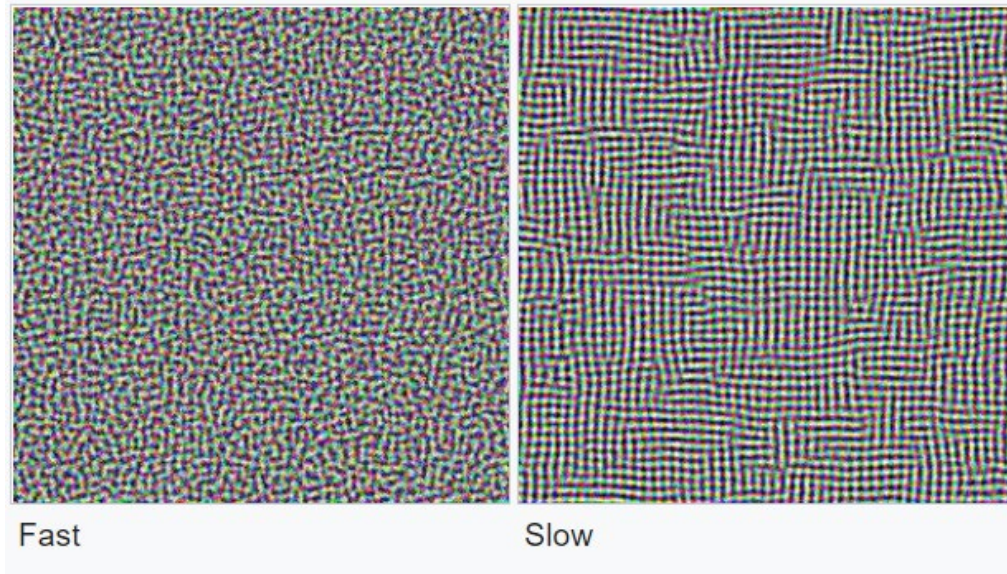
- A probabilistic technique for approximating the global minimum of a given function.
- Pick a random move. If the selected move improves the solution ( $\Delta E < 0$ ), then it is always accepted.
- Otherwise, for the hill-climbing, the algorithm makes the move *with a probability*  $\sim e^{-\Delta E/kT}$ .
- A parameter  $T$  (temperature) is also used to determine this probability. At higher  $T$ , uphill moves are more likely to occur. As  $T$  tends to zero, they become more and more unlikely, until the algorithm behaves more or less like hill-climbing. In a typical SA optimization,  $T$  starts high and is gradually decreased according to an “annealing schedule”.

# Real Matter



Molecular Dynamics  
simulations:

$$M_I \ddot{\vec{R}}_I = \vec{F}_I \equiv -\nabla_{\vec{R}_I} V$$





## ARTICLES

# Optimization by Simulated Annealing

S. Kirkpatrick<sup>1</sup>, C. D. Gelatt Jr.<sup>1</sup>, M. P. Vecchi<sup>2</sup>

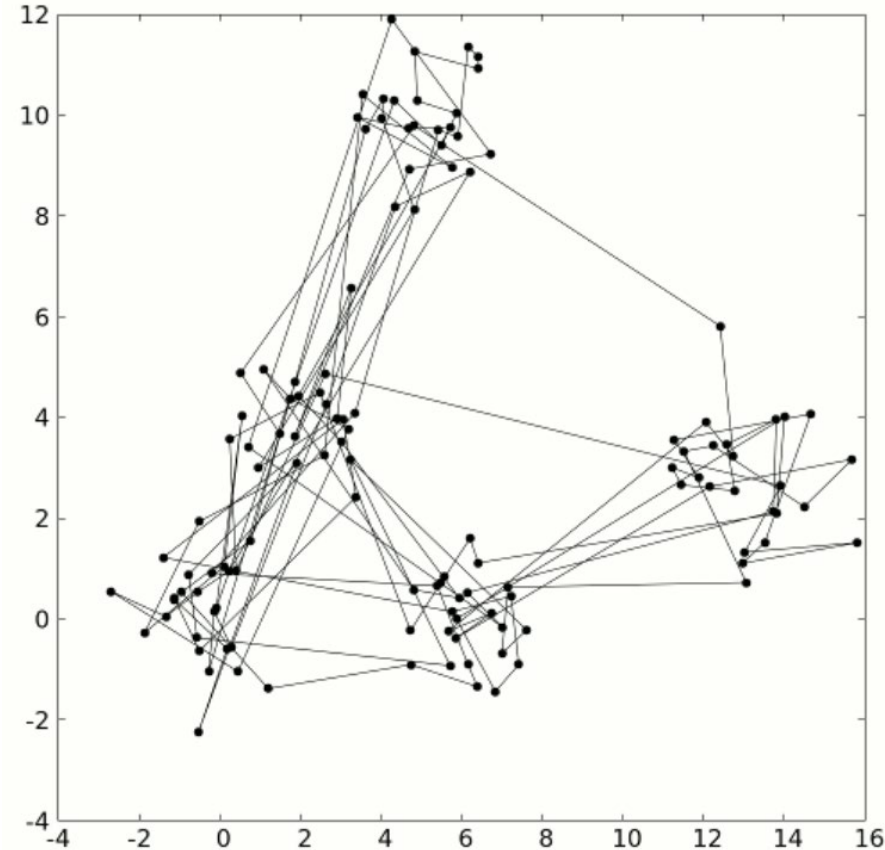
+ See all authors and affiliations

*Science* 13 May 1983:  
Vol. 220, Issue 4598, pp. 671-680  
DOI: 10.1126/science.220.4598.671

There is a deep and useful connection between statistical mechanics (the behavior of systems with many degrees of freedom in thermal equilibrium at a finite temperature) and multivariate or combinatorial optimization (finding the minimum of a given function depending on many parameters). A detailed analogy with annealing in solids provides a framework for optimization of the properties of very large and complex systems. This connection to statistical mechanics exposes new information and provides an unfamiliar perspective on traditional optimization problems and methods.

# Traveling Salesman Problem

to minimize the length of a route that connects all 125 points (cities).



[https://en.wikipedia.org/wiki/File:Travelling\\_salesman\\_problem\\_solved\\_with\\_simulated\\_annealing.gif](https://en.wikipedia.org/wiki/File:Travelling_salesman_problem_solved_with_simulated_annealing.gif)

Optimization of a solution involves evaluating the neighbours of a state of the problem, which are new states produced through conservatively altering a given state. For example, in the **travelling salesman problem** each state is typically defined as a **permutation** of the cities to be visited, and the neighbors of any state are the set of permutations produced by swapping any two of these cities. The well-defined way in which the states are altered to produce neighboring states is called a "move", and different moves give different sets of neighboring states.

From:

[https://en.wikipedia.org/wiki/Simulated\\_annealing](https://en.wikipedia.org/wiki/Simulated_annealing)

## Pseudocode

- Let  $s = s_0$
- For  $k = 0$  through  $k_{\max}$  (exclusive):
  - $T \leftarrow \text{temperature}(1 - (k+1)/k_{\max})$
  - Pick a random neighbour,  $s_{\text{new}} \leftarrow \text{neighbour}(s)$
  - If  $P(E(s), E(s_{\text{new}}), T) \geq \text{random}(0, 1)$ :
    - $s \leftarrow s_{\text{new}}$
- Output: the final state  $s$

$$E = -J \sum_{\langle i,j \rangle} s_i s_j$$

$$1 - \frac{E(s)}{E(s_{\text{new}})} < T$$