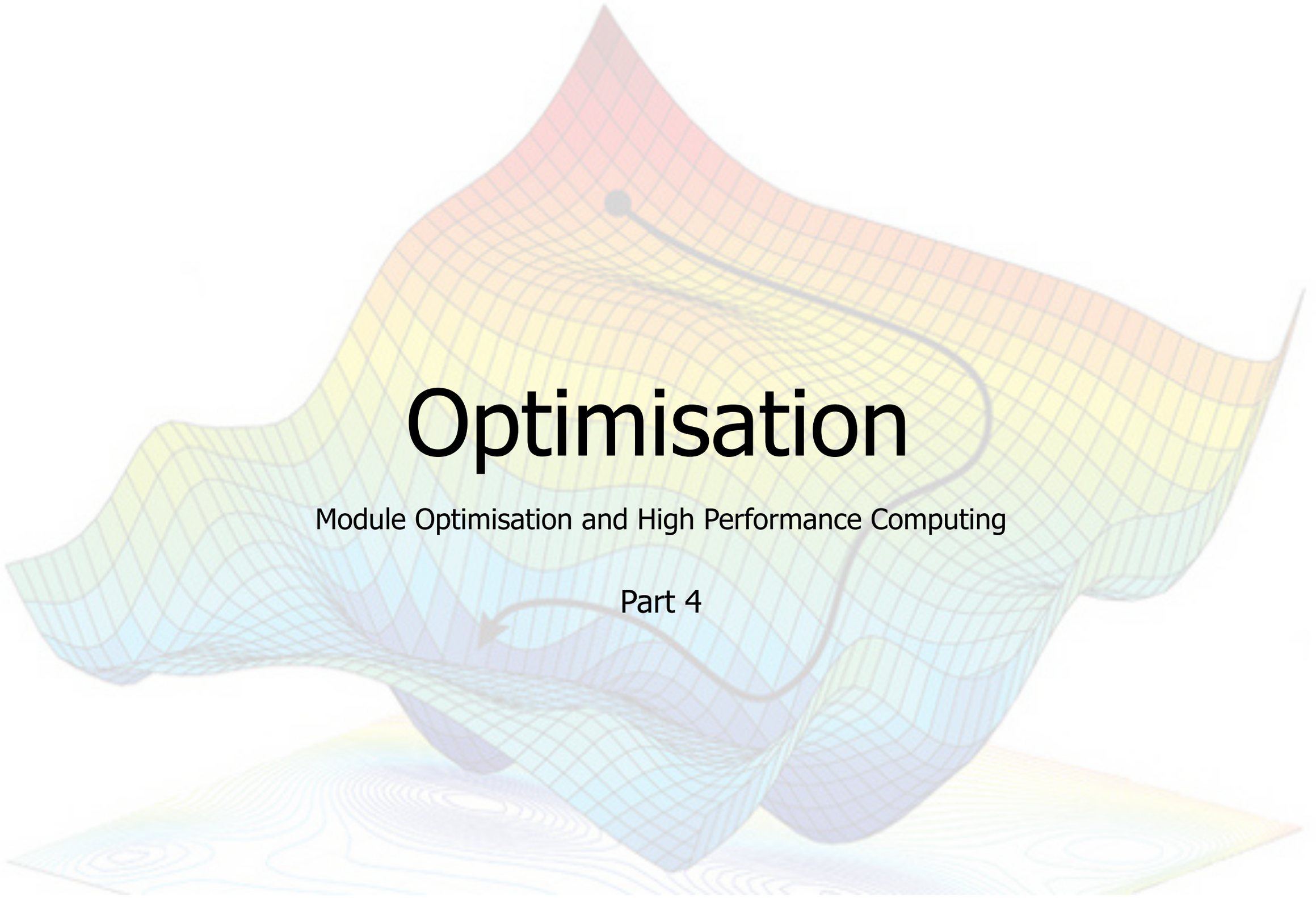
Zurich University of Applied Sciences
Institute of Computational Life Sciences



Optimisation

Module Optimisation and High Performance Computing

Part 4

Part 4

Stochastic Methods:

- Noisy Gradient Descent
- Simulated Annealing

Part 4

Stochastic Methods:

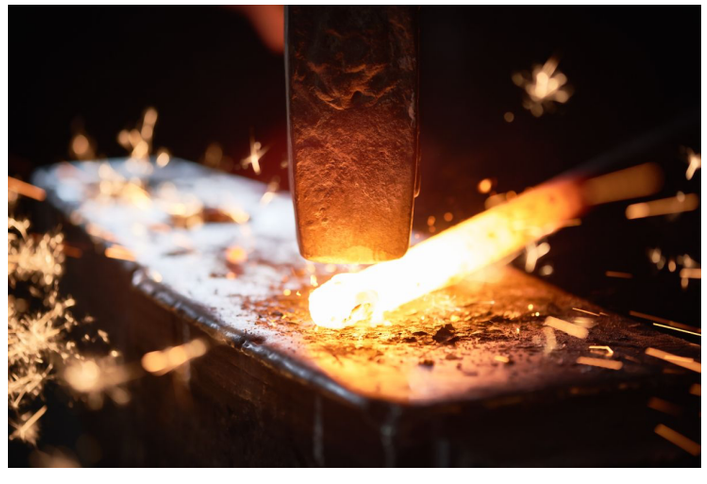
- Noisy Gradient Descent

- **Simulated Annealing**

Simulated Annealing

- **Simulated Annealing** borrows inspiration from **metallurgy**.

  Annealing is a process in which a material is heated and then cooled, making it more workable. **When hot, the atoms are more free to move around and tend to settle into better positions**. A slow cooling brings the material to an ordered, crystalline state. A fast, abrupt quenching causes defects because the material is forced to settle in its current condition.



- **Temperature** is used to control the degree of stochasticity during the randomised search.
  - The **temperature starts high**, allowing large jumps so that the process can freely move in the search space, with the hope that it will find a good region with the best local minimum.
  - The **temperature is then slowly brought down**, reducing the stochasticity and forcing the search to converge to a minimum.

- **Simulated Annealing** is often used on objective functions with many local minima due to its ability to escape such local minima.

Simulated Annealing

1. Start with an initial guess $\mathbf{x}_0$ and (high, but not too high) temperature $T = T_0$

2. Sample a candidate transition $\mathbf{x}_n \rightarrow \mathbf{x}'$ from a **transition distribution** $\mathscr{D}$

   In other words, generate a new candidate $\mathbf{x}'$ by making a small change to the current $\mathbf{x}_n$

3. Calculate the difference in the objective function $\Delta E = f(\mathbf{x}') - f(\mathbf{x}_n)$

4. Accept the move with probability:

$$\begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\Delta E/T} & \text{if } \Delta E > 0 \end{cases}$$

   if the candidate $\mathbf{x}'$ is better, accept it

   if the candidate $\mathbf{x}'$ is worse, accept it with a probability that depends on the temperature T and the difference ΔE

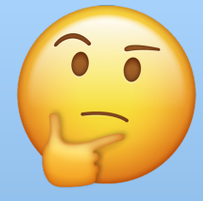
   This acceptance probability, known as **Metropolis criterion**, allows the algorithm to escape from local minima if the temperature is high enough.

5. Reduce the temperature according to a **cooling schedule**

6. Iterate steps 2-5 until the system is frozen or another stopping criterion is met

Simulated Annealing

1. Start with an initial guess $\mathbf{x}_0$ and (high, but not too high) temperature $T = T_0$



2. Sample a candidate transition $\mathbf{x}_n \rightarrow \mathbf{x}'$ from a **transition distribution** $\mathscr{D}$

   In other words, generate a new candidate $\mathbf{x}'$ by making a small change to the current $\mathbf{x}_n$

3. Calculate the difference in the objective function $\Delta E = f(\mathbf{x}') - f(\mathbf{x}_n)$

4. Accept the move with probability:

$$\begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\Delta E/T} & \text{if } \Delta E > 0 \end{cases}$$

   if the candidate $\mathbf{x}'$ is better, accept it

   if the candidate $\mathbf{x}'$ is worse, accept it with a probability that depends on the temperature T and the difference ΔE

   This acceptance probability, known as **Metropolis criterion**, allows the algorithm to escape from local minima if the temperature is high enough.

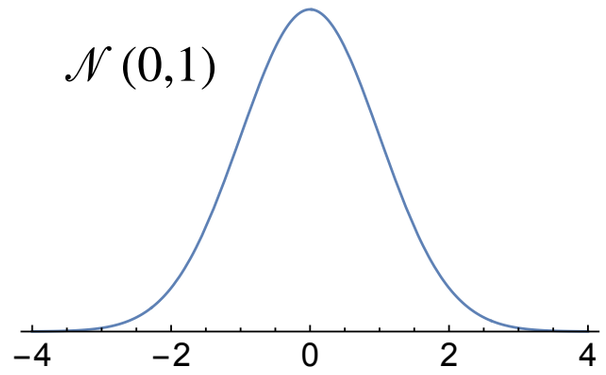
5. Reduce the temperature according to a **cooling schedule**

6. Iterate steps 2-5 until the system is frozen or another stopping criterion is met

Simulated Annealing

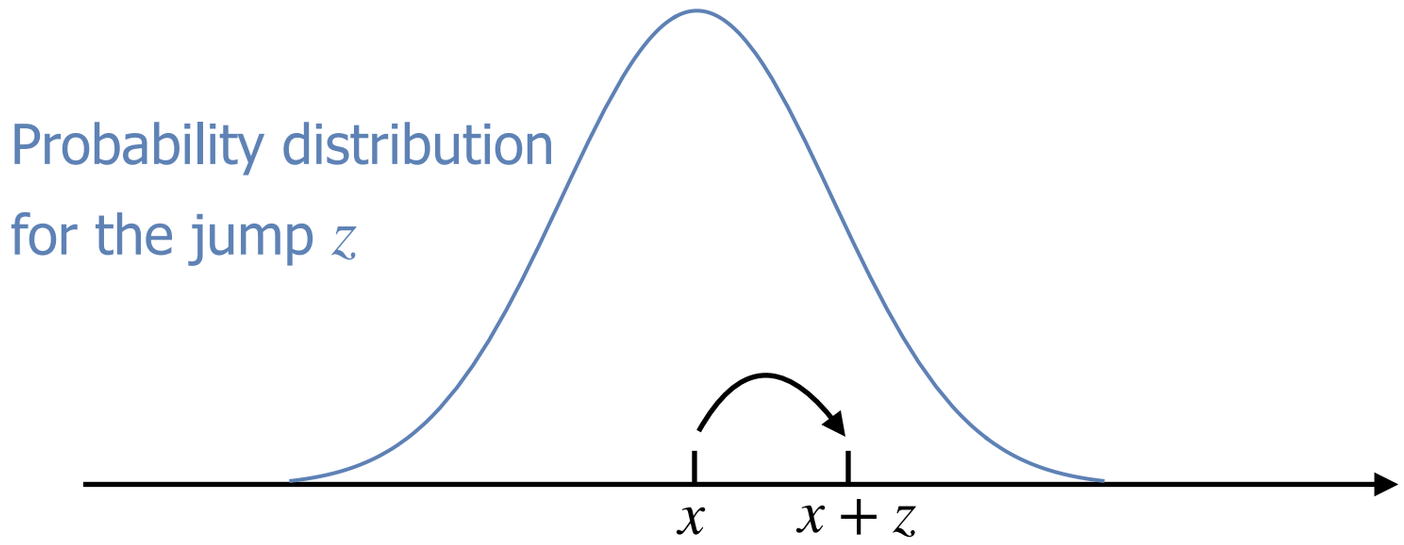
Let us consider first the simplest **univariate case**.

A quite popular transition distribution is a
zero-mean Gaussian distribution: $\mathscr{D} = \mathscr{N}(0,\sigma)$



We are in x and want to make a **random jump** from x to x':

$$x' = x + z \qquad z \sim \mathscr{D}$$

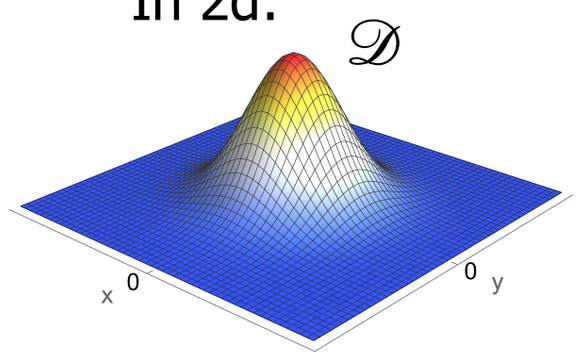


Simulated Annealing

The **multivariate** case is just a generalisation to multiple dimensions:

$$\mathscr{D} = \mathcal{N}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right) \quad \text{with} \quad \boldsymbol{\mu} = \left[0, 0 \ldots, 0\right] \quad \text{and} \quad \boldsymbol{\Sigma} = \sigma\mathbf{I} = \begin{bmatrix} \sigma & 0 & \ldots & 0 \\ 0 & \sigma & \ldots & 0 \\ \vdots & \vdots & \ldots & \vdots \\ 0 & 0 & \ldots & \sigma \end{bmatrix}$$

In 2d:

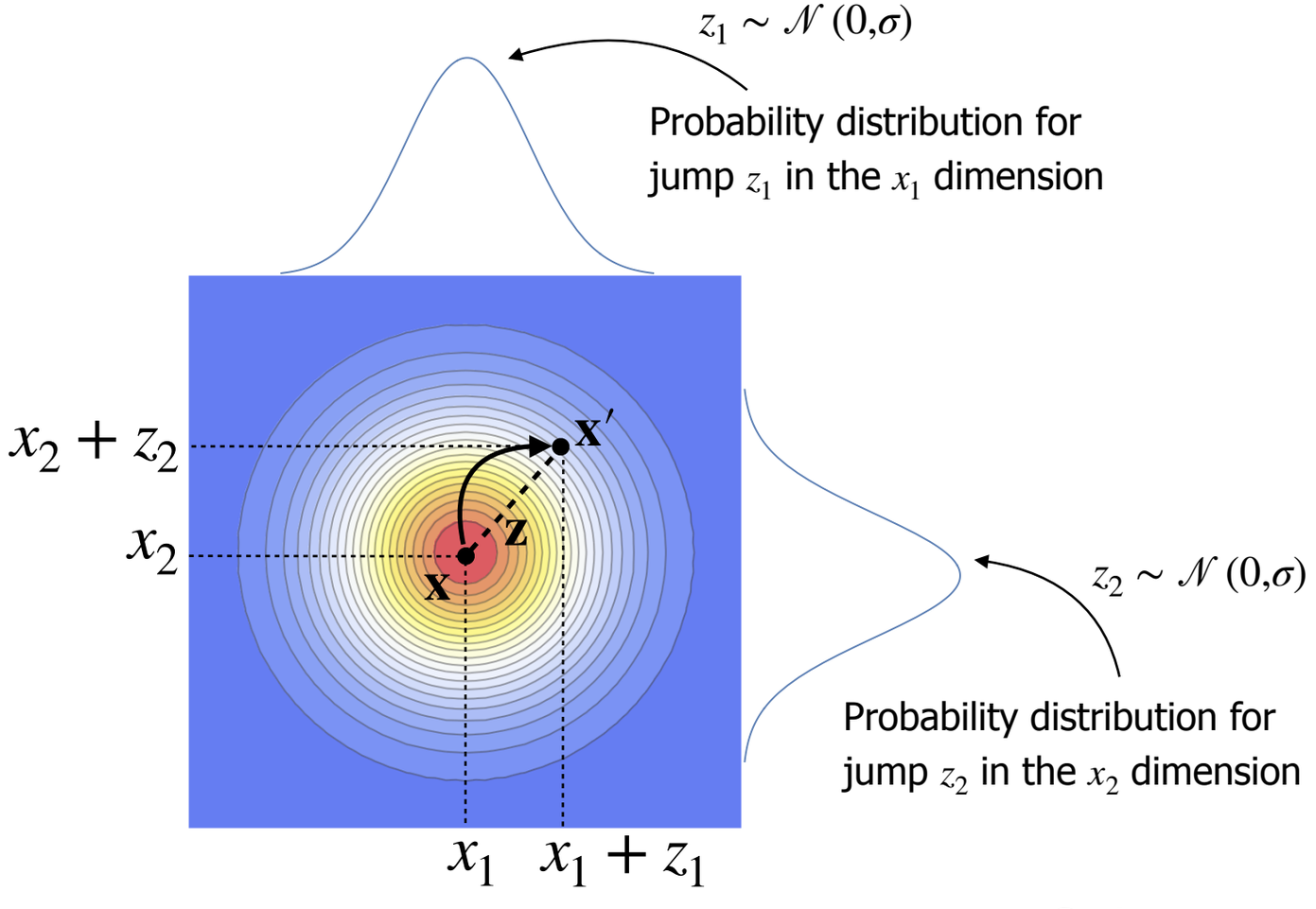


Random jump from $\mathbf{x}$ to $\mathbf{x}'$: $\mathbf{x}' = \mathbf{x} + \mathbf{z} \qquad \mathbf{z} \sim \mathscr{D}$

Example in 2d:

Current state: $\mathbf{x} = \left[x_1, x_2\right]$

Random jump: $\mathbf{z} = \left[z_1, z_2\right]$



Simulated Annealing

- The parameter σ specifies **how far one can jump**. **Critical parameter:** one should stay close to the current $\mathbf{x}$, but not too close.

- In Python, numpy provides an easy way to sample from multivariate distributions:

```python
import numpy as np
# --- Construct covariance matrix --- #
n_dims = 2                    # number of dimensions
means = np.full(n_dims, 0)    # array of mean values = [0,0,...,0]
sigma = 1                     # variance (the same for all dimensions)
covariance_matrix = np.diag(np.full(n_dims, sigma))  # diagonal covariance matrix (sigma*I)
# --- Sample from distribution --- #
z = np.random.multivariate_normal(means, covariance_matrix)
```

Simulated Annealing

1. Start with an initial guess $\mathbf{x}_0$ and (high, but not too high) temperature $T = T_0$

2. Sample a candidate transition $\mathbf{x}_n \rightarrow \mathbf{x}'$ from a **transition distribution** $\mathscr{D}$

   In other words, generate a new candidate $\mathbf{x}'$ by making a small change to the current $\mathbf{x}_n$

3. Calculate the difference in the objective function $\Delta E = f(\mathbf{x}') - f(\mathbf{x}_n)$

4. Accept the move with probability:

$$\begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\Delta E/T} & \text{if } \Delta E > 0 \end{cases}$$

   if the candidate $\mathbf{x}'$ is better, accept it

   if the candidate $\mathbf{x}'$ is worse, accept it with a probability that depends on the temperature T and the difference ΔE

   This acceptance probability, known as **Metropolis criterion**, allows the algorithm to escape from local minima if the temperature is high enough.

5. Reduce the temperature according to a **cooling schedule**

6. Iterate steps 2-5 until the system is frozen or another stopping criterion is met

Simulated Annealing

1. Start with an initial guess $\mathbf{x}_0$ and (high, but not too high) temperature $T = T_0$

2. Sample a candidate transition $\mathbf{x}_n \rightarrow \mathbf{x}'$ from a **transition distribution** $\mathscr{D}$

   In other words, generate a new candidate $\mathbf{x}'$ by making a small change to the current $\mathbf{x}_n$

3. Calculate the difference in the objective function $\Delta E = f(\mathbf{x}') - f(\mathbf{x}_n)$

4. Accept the move with probability:

$$\begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\Delta E/T} & \text{if } \Delta E > 0 \end{cases}$$

if the candidate $\mathbf{x}'$ is better

if the candidate $\mathbf{x}'$ is worse

temperature T and the differe

s on the

In physics the objective function to be minimised is an ENERGY

This acceptance probability, known as **Metropolis criterion**, a s the rithm to escape from local minima if the temperature is high enough.

5. Reduce the temperature according to a **cooling schedule**

6. Iterate steps 2-5 until the system is frozen or another stopping criterion is met

Simulated Annealing

1. Start with an initial guess $\mathbf{x}_0$ and (high, but not too high) temperature $T = T_0$

2. Sample a candidate transition $\mathbf{x}_n \rightarrow \mathbf{x}'$ from a **transition distribution** $\mathscr{D}$

   In other words, generate a new candidate $\mathbf{x}'$ by making a small change to the current $\mathbf{x}_n$

3. Calculate the difference in the objective function $\Delta E = f(\mathbf{x}') - f(\mathbf{x}_n)$

4. Accept the move with probability:

$$\begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\Delta E/T} & \text{if } \Delta E > 0 \end{cases}$$

   if the candidate $\mathbf{x}'$ is better, accept it

   if the candidate $\mathbf{x}'$ is worse, accept it with a probability that depends on the temperature T and the difference ΔE

This acceptance probability, known as **Metropolis criterion**, allows the algorithm to escape from local minima if the temperature is high enough.

5. Reduce the temperature according to a **cooling schedule**

6. Iterate steps 2-5 until the system is frozen or another stopping criterion is met

GUEST EDITORS' INTRODUCTION

the Top 10 Algorithms of the 20th century

… the 10 algorithms with the greatest influence on the development and practice of science and engineering in the 20th century.

- Metropolis Algorithm for Monte Carlo
- Simplex Method for Linear Programming
- Krylov Subspace Iteration Methods
- The Decompositional Approach to Matrix Computations
- The Fortran Optimizing Compiler
- QR Algorithm for Computing Eigenvalues
- Quicksort Algorithm for Sorting
- Fast Fourier Transform
- Integer Relation Detection
- Fast Multipole Method

(in chronological order)

JACK DONGARRA
University of Tennessee and Oak Ridge National Laboratory
FRANCIS SULLIVAN
IDA Center for Computing Sciences

Simulated Annealing

4. Accept the move with probability:

$$\begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\Delta E/T} & \text{if } \Delta E > 0 \end{cases}$$

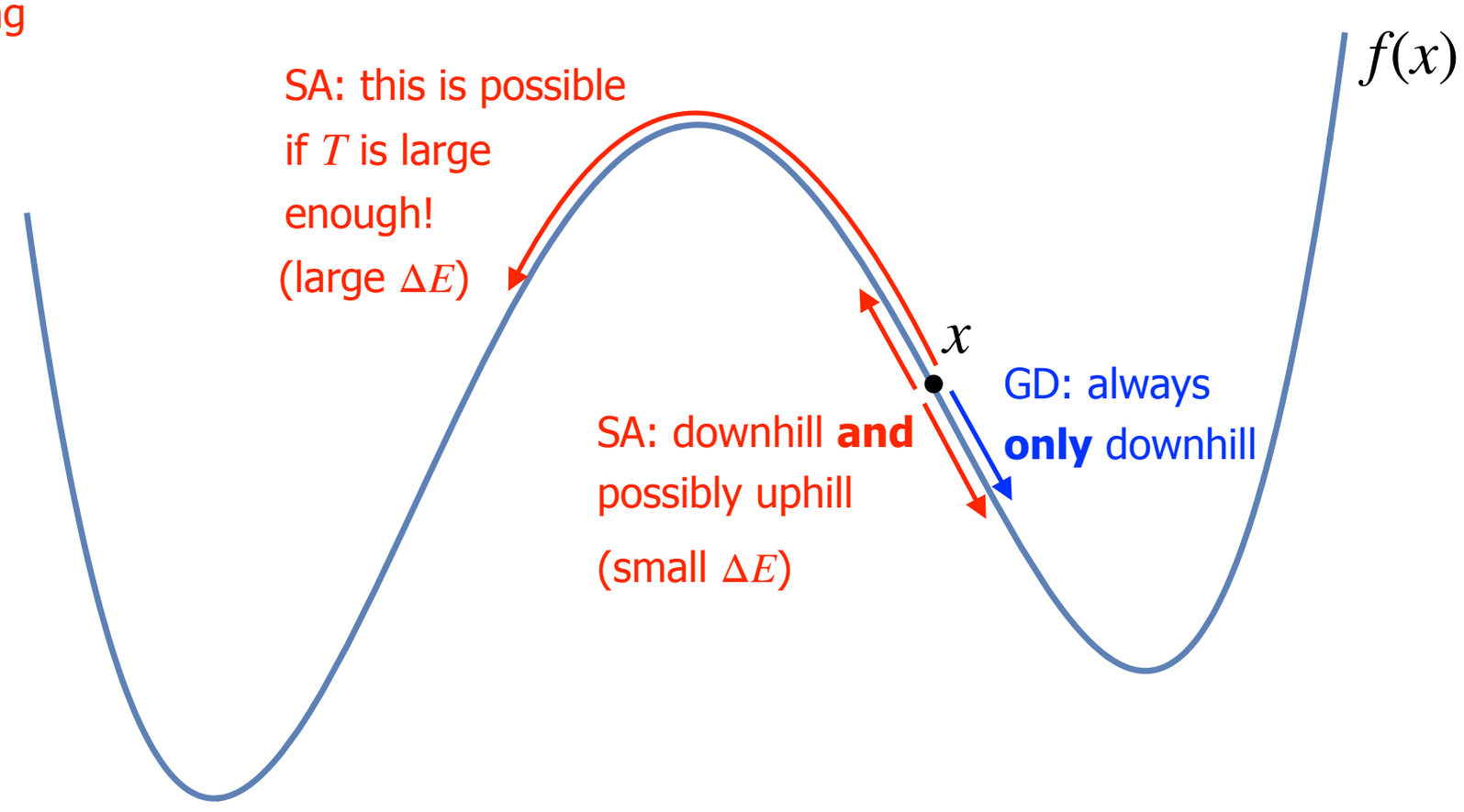
if the candidate $\mathbf{x}'$ is better, accept it

if the candidate $\mathbf{x}'$ is worse, accept it with a probability that depends on the temperature T and the difference ΔE

This acceptance probability, known as **Metropolis criterion**, allows the algorithm to escape from local minima if the temperature is high enough.

SA = Simulated Annealing

GD: Gradient Descent



Simulated Annealing

4. Accept the move with probability:

$$\begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\Delta E/T} & \text{if } \Delta E > 0 \end{cases}$$

if the candidate $\mathbf{x}'$ is better, accept it

if the candidate $\mathbf{x}'$ is worse, accept it with a probability that depends on the temperature T and the difference ΔE

This acceptance probability, known as **Metropolis criterion**, allows the algorithm to escape from local minima if the temperature is high enough.

Practically, how can it be implemented?

- Given current x_n and the proposal x', calculate $\Delta E = f(\mathbf{x}') - f(\mathbf{x}_n)$
- Generate a random number $s \in [0, 1)$

```
s = np.random.rand()
```

- `If` $e^{-\Delta E/T_n} \geq s$ accept the move: $x_{n+1} = x'$
- `else` reject it (and stay where you are): $x_{n+1} = x_n$

Simulated Annealing

1. Start with an initial guess $\mathbf{x}_0$ and (high, but not too high) temperature $T = T_0$

2. Sample a candidate transition $\mathbf{x}_n \rightarrow \mathbf{x}'$ from a **transition distribution** $\mathscr{D}$

   In other words, generate a new candidate $\mathbf{x}'$ by making a small change to the current $\mathbf{x}_n$

3. Calculate the difference in the objective function $\Delta E = f(\mathbf{x}') - f(\mathbf{x}_n)$

4. Accept the move with probability:

$$\begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\Delta E/T} & \text{if } \Delta E > 0 \end{cases}$$

   if the candidate $\mathbf{x}'$ is better, accept it

   if the candidate $\mathbf{x}'$ is worse, accept it with a probability that depends on the temperature T and the difference ΔE

This acceptance probability, known as **Metropolis criterion**, allows the algorithm to escape from local minima if the temperature is high enough.

The algorithm will not converge if we keep the temperature high

5. Reduce the temperature according to a **cooling schedule**

6. Iterate steps 2-5 until the system is frozen or another stopping criterion is met

Simulated Annealing

- The temperature parameter T controls the acceptance probability. An **annealing/cooling schedule** is used to slowly bring down the temperature as the algorithm progresses.
- The temperature T must be brought down to **ensure convergence**.
- If it is brought down too quickly, the search may not cover adequately the search space.
- Common annealing schedules are:
  - $T_n = T_0\left(1 - \frac{n}{n_{\max}}\right)$ $n_{\max}$ = max number of iterations
  - Logarithmic: $T_n = \frac{T_0 \ln 2}{\ln(n+1)}$ Efficient at finding global minima, but slow

    B. Hajek, "Cooling Schedules for Optimal Annealing", Mathematics of Operations Research, 1988
  - Exponential: $T_{n+1} = \gamma T_n$ $\gamma \in (0,1)$ Practically: $0.8 \leq \gamma \leq 0.99$
  - Fast annealing: $T_n = \frac{T_1}{n}$ H. Szu and R. Hartley, "Fast Simulated Annealing", Physics Letters A, 1987

Simulated Annealing

1. Start with an initial guess $\mathbf{x}_0$ and (high, but not too high) temperature $T = T_0$

2. Sample a candidate transition $\mathbf{x}_n \rightarrow \mathbf{x}'$ from a **transition distribution** $\mathscr{D}$

   In other words, generate a new candidate $\mathbf{x}'$ by making a small change to the current $\mathbf{x}_n$

3. Calculate the difference in the objective function $\Delta E = f(\mathbf{x}') - f(\mathbf{x}_n)$

4. Accept the move with probability:

$$\begin{cases} 1 & \text{if } \Delta E \leq 0 \\ e^{-\Delta E/T} & \text{if } \Delta E > 0 \end{cases}$$

   if the candidate $\mathbf{x}'$ is better, accept it

   if the candidate $\mathbf{x}'$ is worse, accept it with a probability that depends on the temperature T and the difference ΔE

This acceptance probability, known as **Metropolis criterion**, allows the algorithm to escape from local minima if the temperature is high enough.

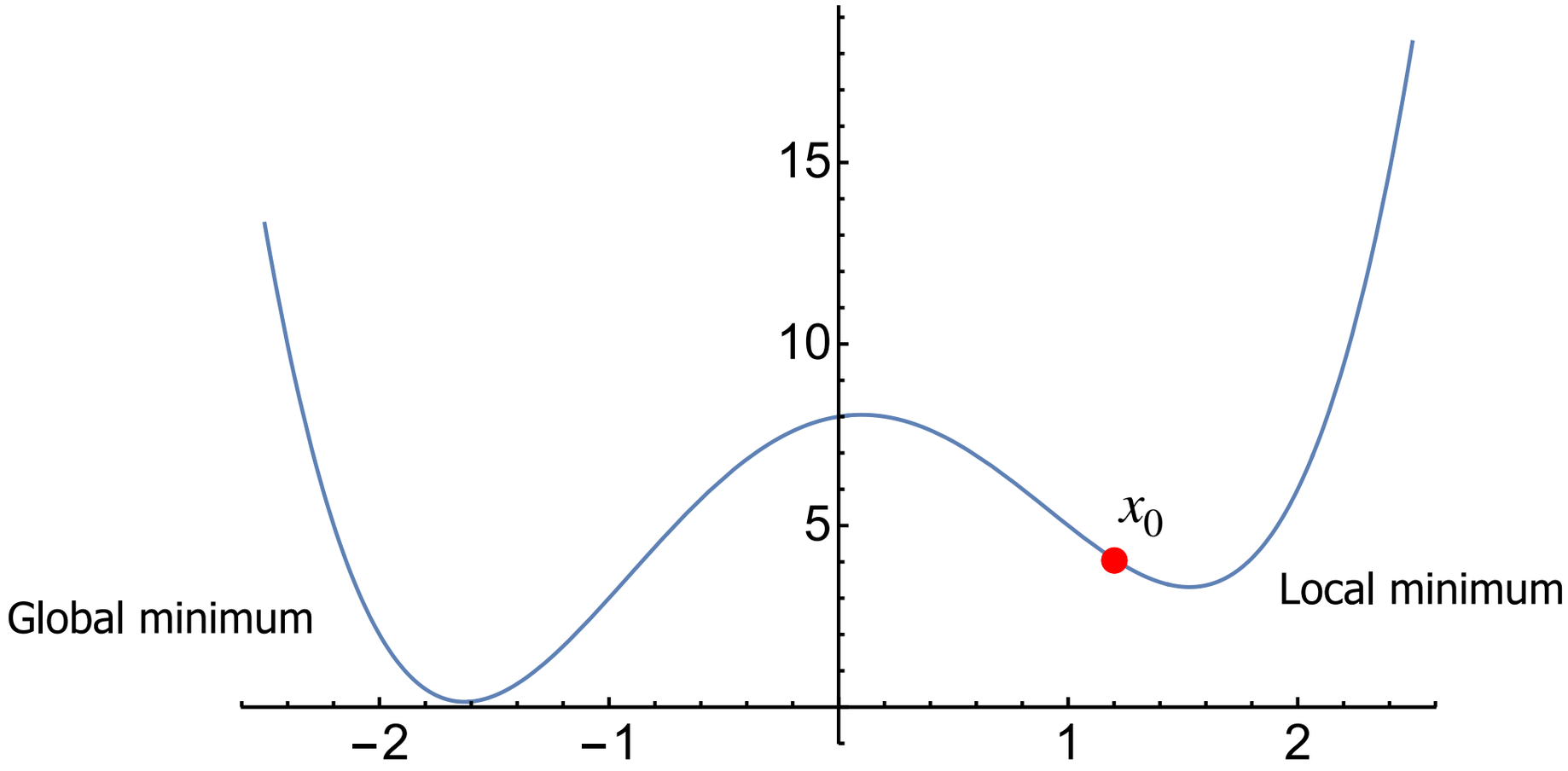
The algorithm will not converge if we keep the temperature high

5. Reduce the temperature according to a **cooling schedule**

6. Iterate steps 2-5 until the system is frozen or another stopping criterion is met

Simulated Annealing

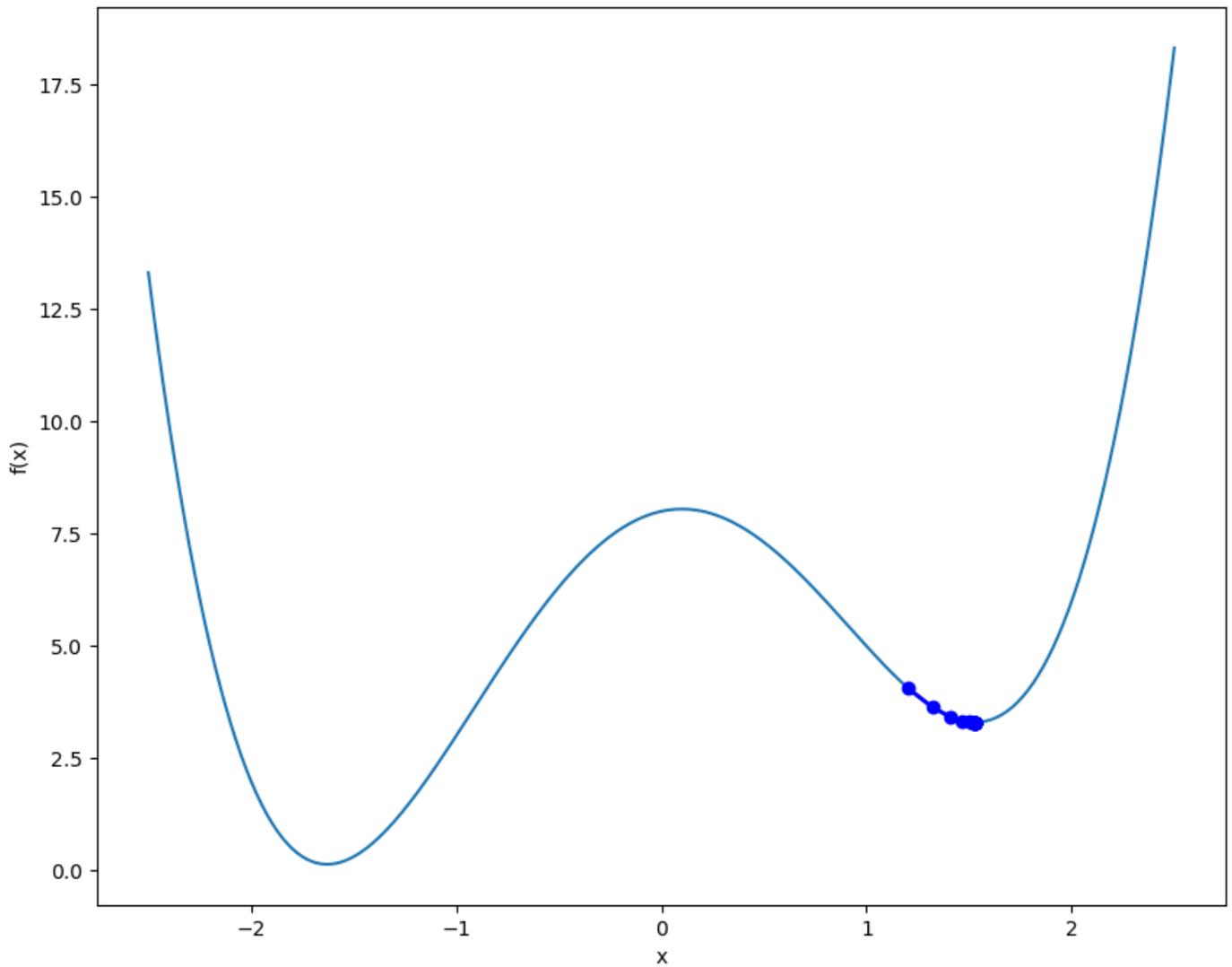
- Consider the univariate objective function $f(x) = x^4 - 5x^2 + x + 8$
- and a start point $x_0 = 1.2$
- Optimisation problem: $\min_x f(x)$



Simulated Annealing

- Consider the univariate objective function $f(x) = x^4 - 5x^2 + x + 8$
- and a start point $x_0 = 1.2$
- Optimisation problem: $\min_x f(x)$

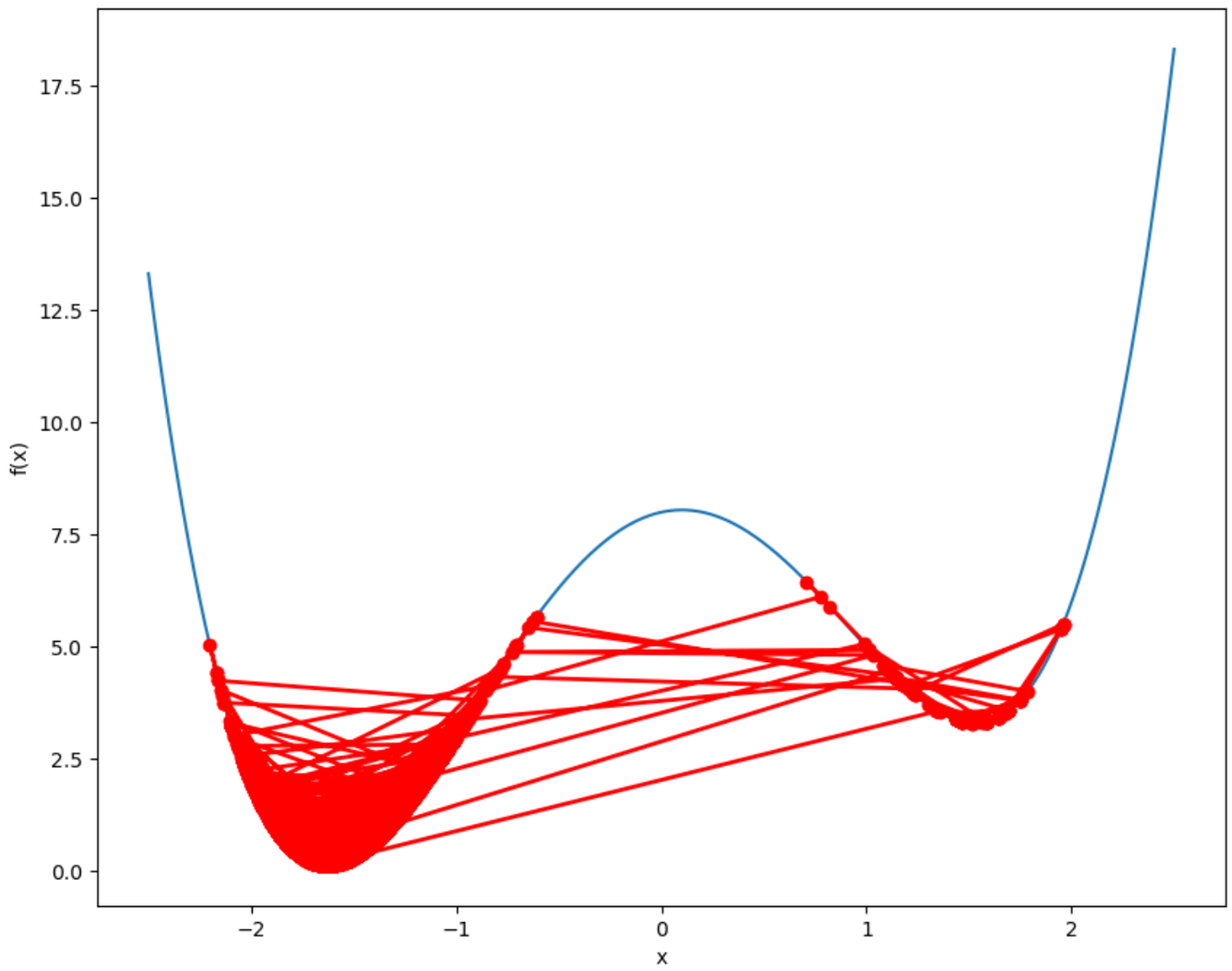
Gradient Descent finds the **local minimum**



Simulated Annealing

- Consider the univariate objective function $f(x) = x^4 - 5x^2 + x + 8$
- and a start point $x_0 = 1.2$
- Optimisation problem: $\min_x f(x)$

Simulated Annealing explores the whole space and finds the **global minimum**



$T_0 = 1.0$

$\sigma = 1.0$

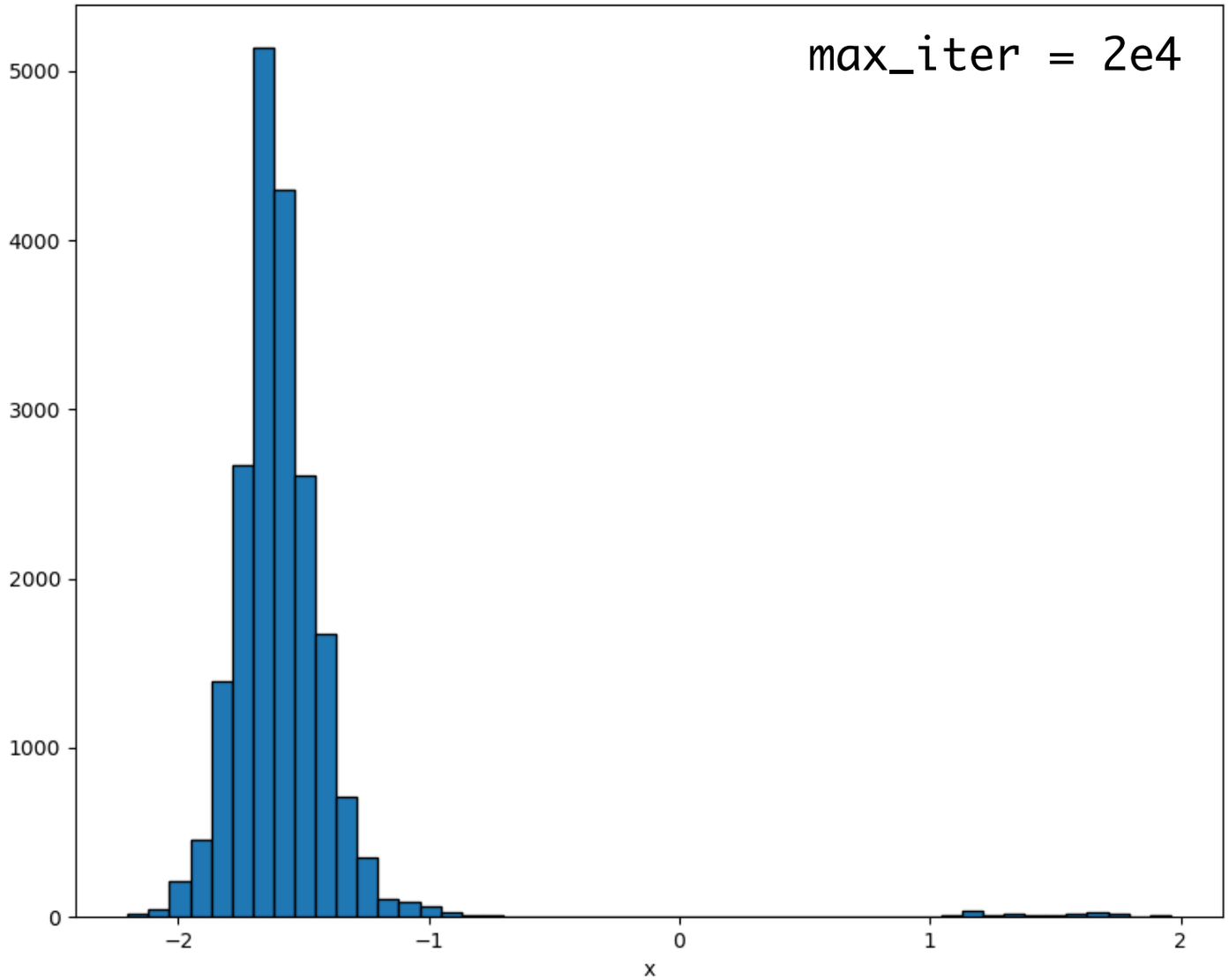
`max_iter = 2e4`

Simulated Annealing

- Consider the univariate objective function $f(x) = x^4 - 5x^2 + x + 8$
- and a start point $x_0 = 1.2$
- Optimisation problem: $\min_x f(x)$
- **Plot the histogram**!

  It helps visualise how the algorithm samples the search space and converges to the **global** minimum.

Monte Carlo Sampling



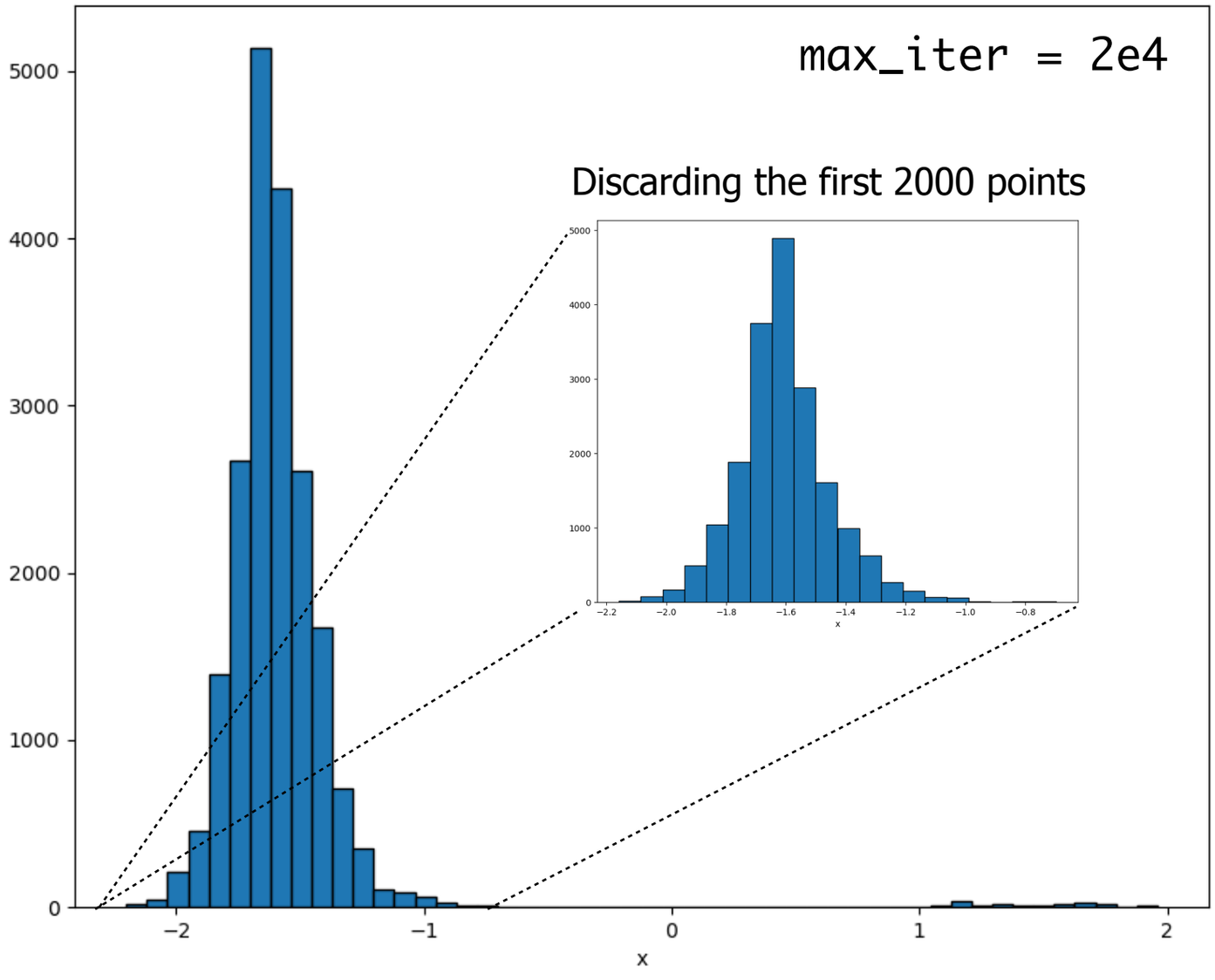
Simulated Annealing

- Consider the univariate objective function $f(x) = x^4 - 5x^2 + x + 8$
- and a start point $x_0 = 1.2$
- Optimisation problem: $\min_x f(x)$
- **Plot the histogram**!

  It helps visualise how the algorithm samples the search space and converges to the **global** minimum.

Monte Carlo Sampling

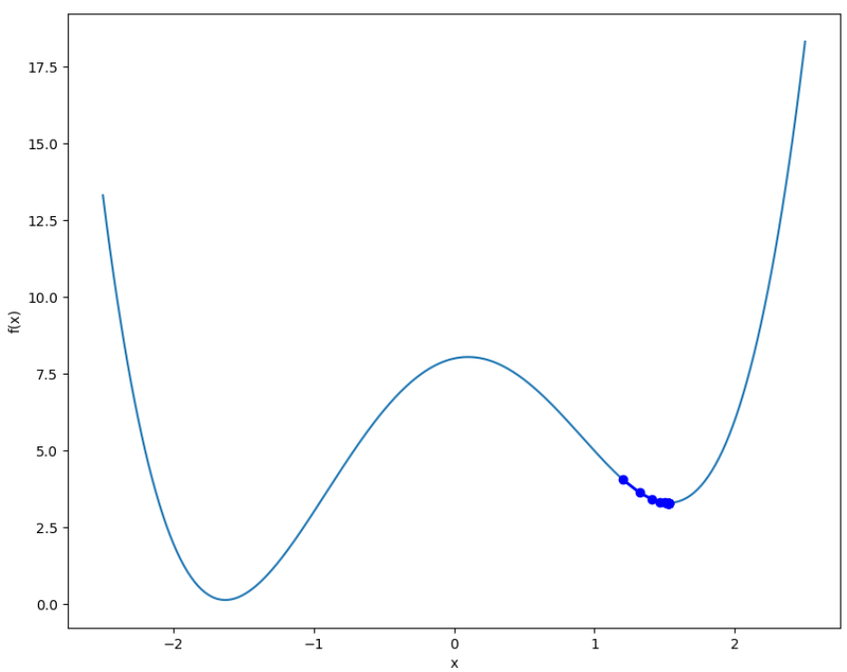
Mean over the dominant peak: $\bar{x} \approx -1.60$

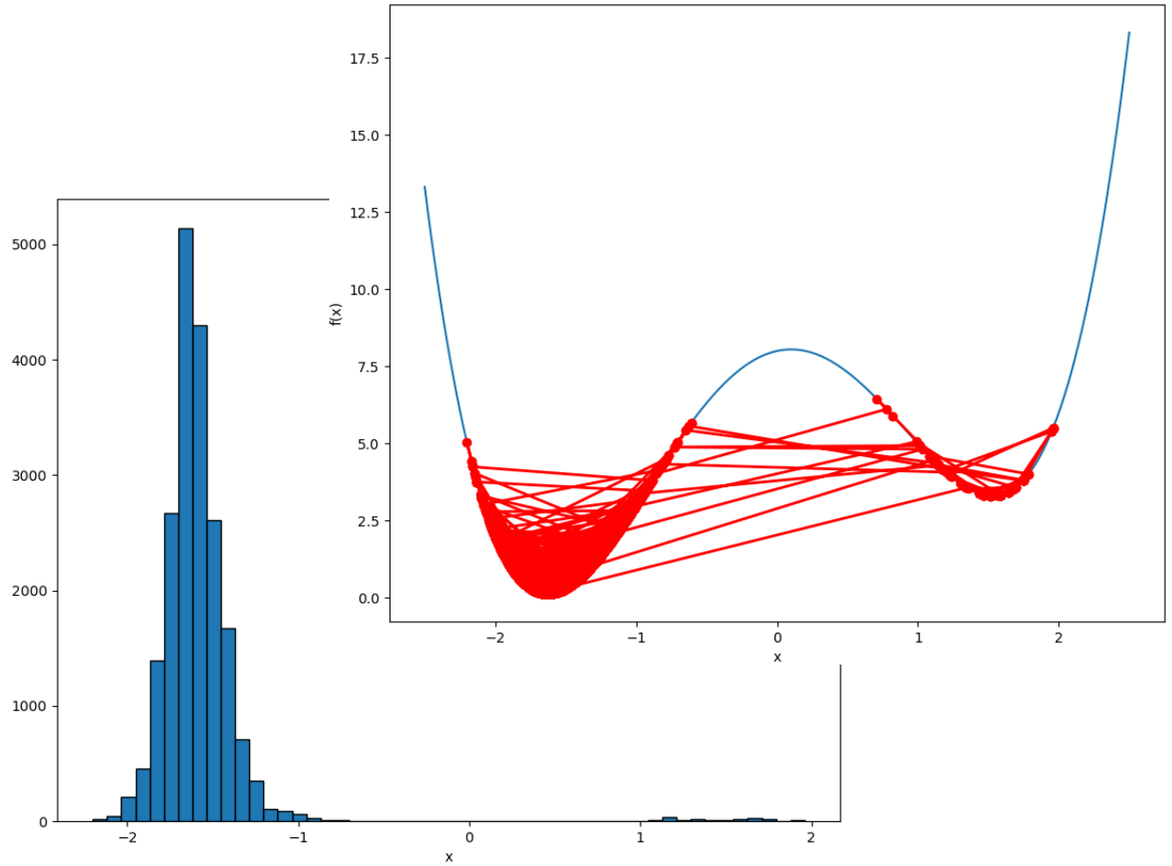


Simulated Annealing

- **Gradient Descent** is a **deterministic gradient-based** optimisation algorithm, suitable for smooth, differentiable functions, efficient for finding **local minima**, especially in convex problems.

- **Simulated Annealing** is a **probabilistic sampling-based** optimisation algorithm, suitable for a wide range of problems, including non-differentiable and non-convex functions, capable of finding **global minima** by escaping local minima through probabilistic moves.





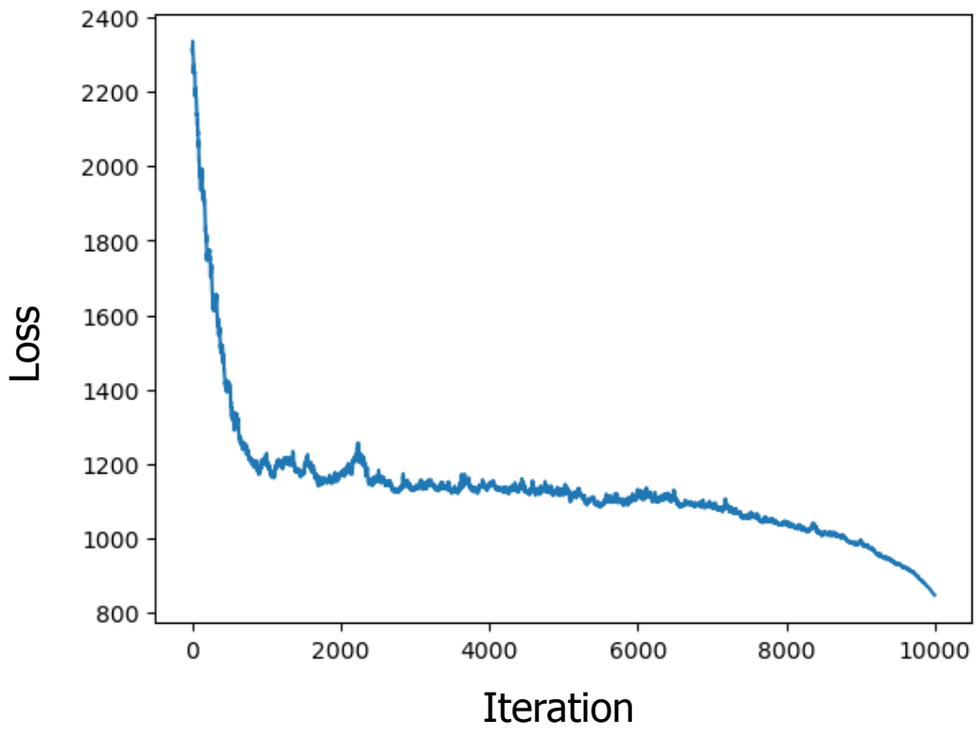
Simulated Annealing

```python
def simulated_annealing(x0, T0, sigma, f, n_iter = 2.5e5, burn_in = 2e5):

    x = x0.copy()  # initialize x
    T = T0         # initialize T
    n_params = x0.shape[0]  # number of parameters to be optimized

    # Means and covariance matrix for the jump distribution -> multivariate normal with mean 0 and standard deviation sigma
    means = np.full(n_params, 0)
    cov_matrix = np.diag(np.full(n_params, sigma))

    # Size of the output array after burn_in
    size_out = int(n_iter - burn_in)
    v = np.zeros((size_out, n_params))

    iter_counter = 0
    print("Initial loss:", f(x))
    # Start main loop
    while iter_counter < n_iter:
        iter_counter += 1;
        x_old = x;
        x_proposal = x_old + np.random.multivariate_normal(means, cov_matrix )
        DeltaE = f(x_proposal) - f(x_old)
        # Metropolis accept/reject step
        if np.exp(-np.clip(DeltaE/T,-100,100)) >= np.random.rand():
            x = x_proposal
        else:
            x = x_old
        # Update temperature according to schedule
        T = T0*(1-iter_counter/n_iter)
        # Keep track of the algorithm state
        if iter_counter%10000 == 0:
            print("Iteration ", iter_counter, " - Temperature:", T, "Loss:", f(x))
        if iter_counter > burn_in:
            v[iter_counter-int(burn_in)-1, :] = x

    return v
```

Loss function

- Run SA for different values of T_0 and σ, and look at the evolution of the loss (objective) function (e.g., MSE).

```
T0 = 1
sigma = 1
outSA = simulated_annealing(x0, T0, sigma, mse, n_iter = 2.5e5, burn_in = 2e5)

plt.figure()
mse_curve = np.apply_along_axis(mse, 1, outSA)
plt.plot(mse_curve)
plt.show(block=False)
```



Loss function

- Run SA for different values of T_0 and σ, and look at the evolution of the loss (objective) function (e.g., MSE). **Which one would you consider the "best"?**