OPTIMIZATION ALGORITHMS · MASTER'S COURSE

Stochastic & Population Methods

Lecture 4 — Randomness as a search tool: annealing, distribution-based, and evolutionary methods

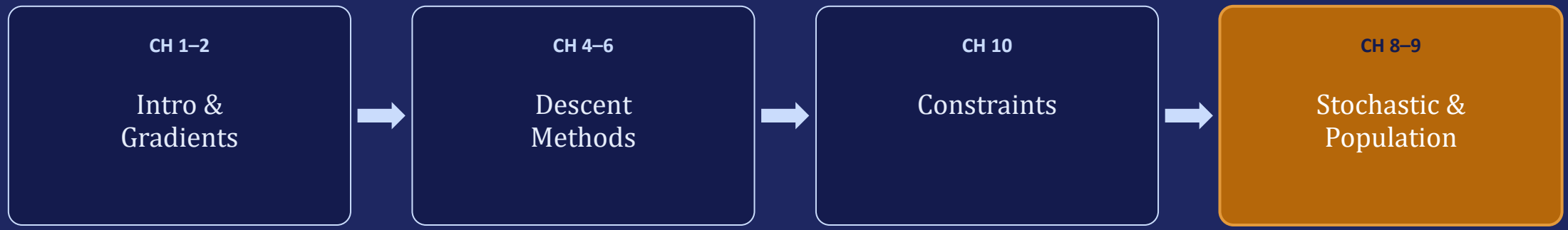
- Noisy descent & simulated annealing
- Cross-entropy, NES & CMA-ES
- Genetic algorithms, DE, PSO & friends

Based on Kochenderfer & Wheeler, Algorithms for Optimization (MIT Press) — Chapters 8 & 9

WHERE WE ARE IN THE COURSE

Course Roadmap

Gradient-based methods (and their constrained cousins) dominate Lectures 1–3. Today we relax the assumption that a gradient is available — or that a single design point is enough.



Direct (derivative-free) methods, Chapter 7, are not covered in this lecture.

PART A · CHAPTER 8

Stochastic Methods

Using randomness strategically to explore the design space

Why Add Randomness?

Stochastic methods use (pseudo-)random numbers to explore the design space. Randomness can help escape local optima and increase the chances of finding a global optimum — but a large amount of randomness is ineffective, because it prevents us from using earlier evaluations to guide the search.

Escape

Noise can push iterates off saddle points and out of shallow basins where deterministic methods stall.

Explore

Random proposals sample regions a purely local method would never visit, improving the odds of finding the global optimum.

Control

The art is controlling how much randomness: temperature schedules, shrinking noise, or an adapted sampling distribution.

Reproducibility: pseudo-random number generators look random but are deterministic — fixing the seed lets us repeat an experiment exactly.

Adding Noise to Descent

Saddle points, where the gradient is nearly zero, can make descent methods choose uselessly small steps. One remedy: add zero-mean Gaussian noise at every step.

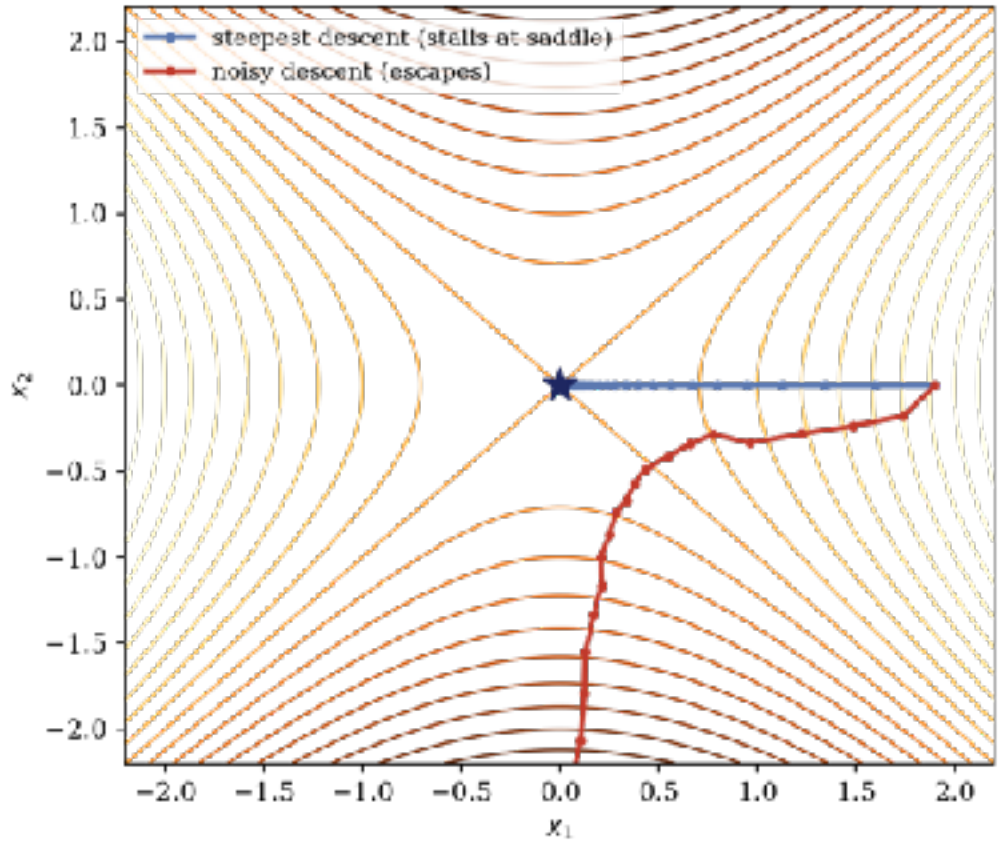
$$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k)} - \alpha \nabla f(\mathbf{x}^{(k)}) + \boldsymbol{\varepsilon}^{(k)}$$

$$\boldsymbol{\varepsilon}^{(k)} \sim \mathcal{N}(0,\ (\sigma^{(k)})^2 \mathbf{I})$$

The noise level $\sigma^{(k)}$ is typically reduced over time — for example a decreasing sequence such as 1/k — and the scheme can wrap any descent method (momentum, Adam, …), not just plain gradient descent.

Stochastic gradient descent is the same idea from another angle: the gradient is estimated from a randomly chosen mini-batch of the training data — noisier, but far cheaper than the true gradient, and the noise also helps traverse saddle points.

Escaping a Saddle Point



Test function $f(x) = x_1^2 - x_2^2$

It has a saddle at the origin: a minimum along x_1 but a maximum along x_2. The gradient vanishes there.

Started exactly on the x_1-axis, steepest descent converges to the saddle and stops — the gradient never points off the axis.

Noisy descent is nudged off the axis. Once $x_2 \neq 0$, the descent direction itself pushes x_2 away from zero, and the iterate slides down the negative-curvature direction.

Illustration: step size 0.08, noise $\sigma^{(k)} = 0.07/\sqrt{k}$.

Convergence of Stochastic Gradient Descent

Convergence guarantees for stochastic gradient descent require positive step sizes $\alpha^{(k)}$ satisfying:

$$\sum_{k=1}^{\infty} \alpha^{(k)} = \infty \qquad \sum_{k=1}^{\infty} (\alpha^{(k)})^2 < \infty$$

$\Sigma\, \alpha^{(k)} = \infty$

Steps stay large enough in total that the iterates can travel anywhere they need to — they cannot stall far from a minimum.

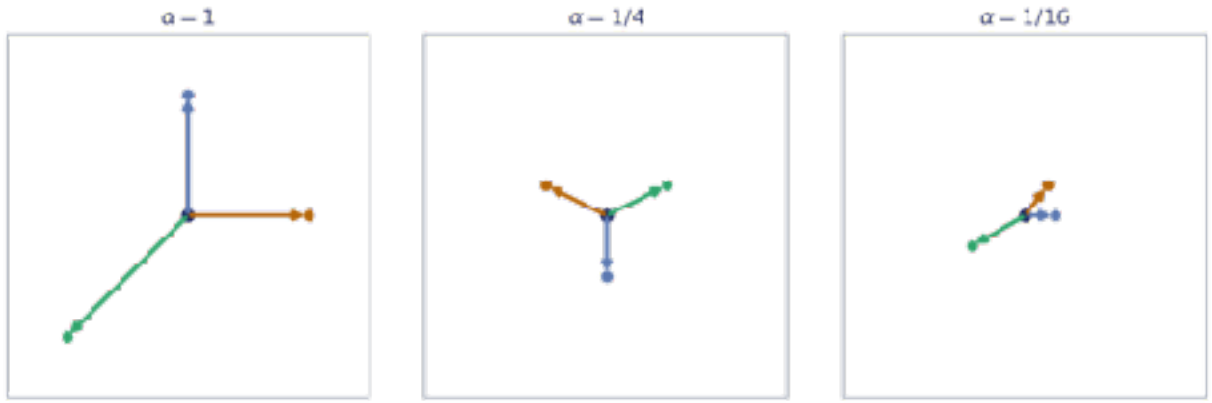
$\Sigma\, (\alpha^{(k)})^2 < \infty$

Steps shrink fast enough that the noise averages out, so the sequence settles instead of jittering forever.

A sequence like $\alpha^{(k)} = 1/k$ satisfies both: not too aggressive, but not decaying so quickly that progress stops early.

Random Positive Spanning Sets

Generalized pattern search restricts local exploration to a fixed set of directions. Mesh adaptive direct search (MADS) instead samples random positive spanning directions — n+1 of them in n dimensions — on a mesh whose resolution adapts.



Directions for three step sizes (2-D, n+1 = 3): as α shrinks, the mesh gets finer.

1. Draw a random lower-triangular matrix with diagonal ±1/√α and random off-diagonal entries; shuffle rows and columns.
2. Append the negative sum of the columns to obtain n+1 directions.
3. Try x + αd for each direction. On an improvement, accept it, then also try the longer step x + 3αd.
4. Success → α ← min(4α, 1). Failure → α ← α/4.

Randomness here is in the search directions: the method explores freely in all directions over time, yet needs only function evaluations — no gradients.

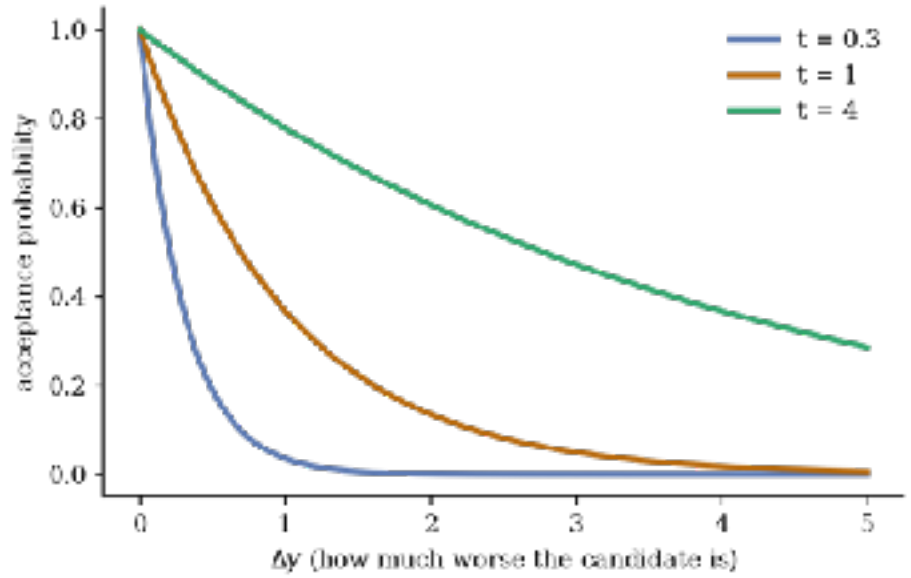
Borrowing from Metallurgy

Annealing heats a material, then cools it slowly: hot atoms move freely and, as the temperature falls, settle into an ordered low-energy state. Quenching too fast freezes in defects. Simulated annealing uses a temperature t to control the degree of stochasticity: high at first for exploration, then lowered to converge.

At each iteration, a candidate x′ is sampled from a transition distribution T and accepted with the Metropolis criterion:

$$P(\text{accept } \mathbf{x} \to \mathbf{x}') = \min\left(1,\ e^{-\Delta y / t}\right) \qquad \Delta y = f(\mathbf{x}') - f(\mathbf{x})$$

Improvements are always accepted; worse points are accepted with a probability that shrinks as they get worse — or as t falls.



Hot (t = 4): almost any move accepted. Cold (t = 0.3): essentially greedy.

Annealing Schedules

Logarithmic

$$t^{(k)} = \frac{t^{(1)} \ln 2}{\ln(k+1)}$$

Guaranteed to reach the global optimum asymptotically under certain conditions — but very slow in practice.

Exponential

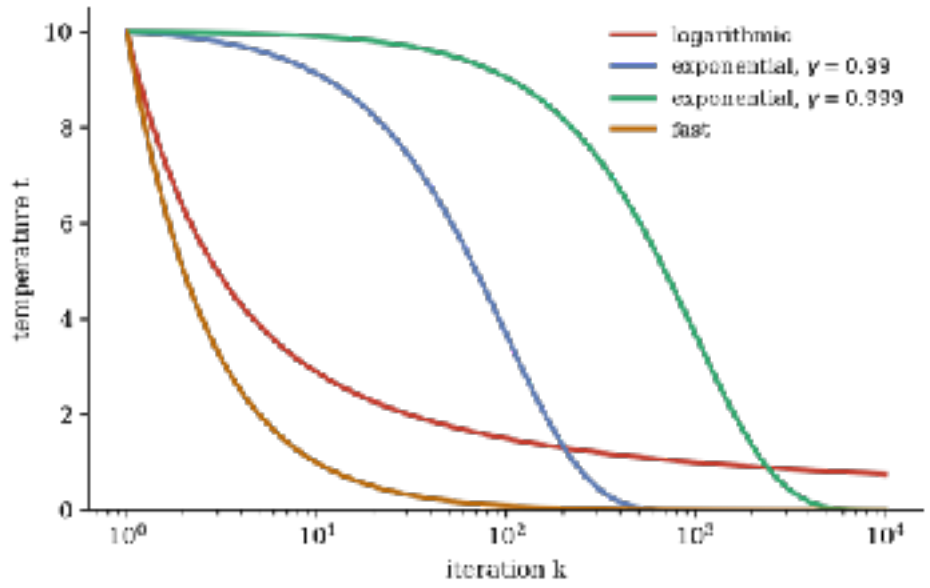
$$t^{(k+1)} = \gamma t^{(k)}, \quad \gamma \in (0,1)$$

The most common choice: multiply the temperature by a decay factor γ each step.

Fast

$$t^{(k)} = \frac{t^{(1)}}{k}$$

Temperature falls as 1/k — aggressive cooling, risk of freezing too early.



The temperature must come down slowly enough that the search still covers the region containing the global minimum — cool too quickly and it may freeze in the wrong basin.

The Algorithm

```
x_best, y_best ← x, f(x);   y ← f(x)
for k = 1 … k_max
    x′ ← x + (random sample from T)
    Δy ← f(x′) − y
    if Δy ≤ 0   or   rand() < exp(−Δy / t(k))
        x, y ← x′, f(x′)            # accept
    if y < y_best:   x_best, y_best ← x, y
return x_best
```

Transition distribution T

e.g. Gaussian steps; its width sets how far a single move can jump.

Schedule t(k)

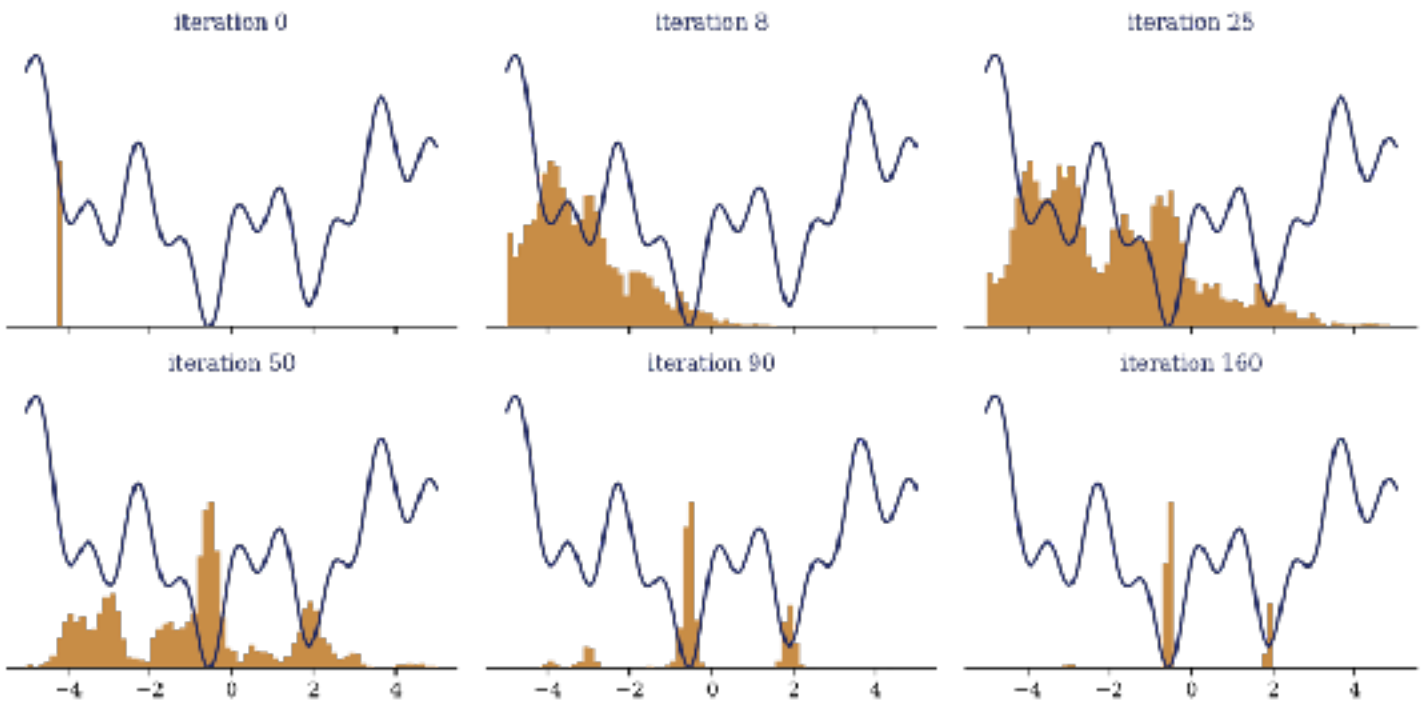
Chosen from the previous slide; sets how fast uphill moves become unlikely.

Budget k_max

Also track the best point seen — the chain itself may wander uphill.

A more sophisticated variant (Corana et al., 1987) lets the step size change during the search rather than staying fixed. Annealing is often used on functions with many local minima precisely because of its ability to escape them.

Probability Mass Settles into the Valleys



What the histograms show

Where 4,000 independent annealing runs are, over time. All start at x = −4.2.

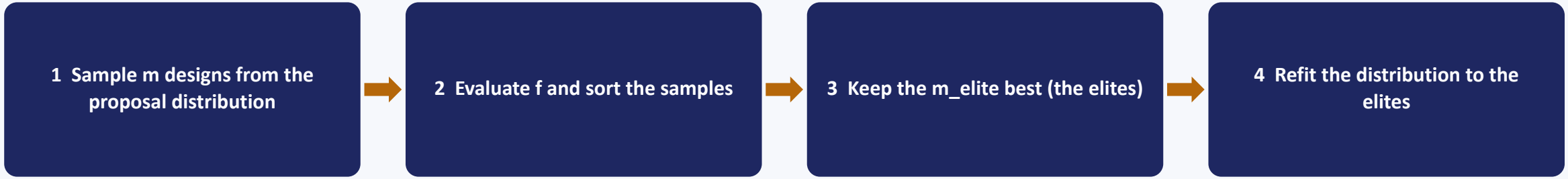
While the temperature is high the population spreads across the whole domain.

As it cools, mass concentrates in the low-lying minima — mostly the global minimum, with some runs trapped in a local one.

Exponential schedule, γ = 0.965, Gaussian transitions (σ = 0.7). Slower cooling would trap fewer runs — at the price of more iterations.

Evolving a Sampling Distribution

The methods so far move a single design point. The cross-entropy method instead maintains an explicit probability distribution over the design space — the proposal distribution — and improves the distribution itself.



Goal

The proposal distribution gradually concentrates around the global optima.

Choices

A distribution family parameterized by θ (often a multivariate normal: mean vector + covariance matrix) and the elite count m_elite.

Why it works

Fitting to the best samples shifts probability mass toward better regions, so the next samples are better on average.

Fitting the Distribution to the Elites

For a multivariate normal, fitting means taking the maximum likelihood estimate from the m_elite best samples:

$$\boldsymbol{\mu}^{(k+1)} = \frac{1}{m_{\text{elite}}} \sum_{i=1}^{m_{\text{elite}}} \mathbf{x}^{(i)}$$

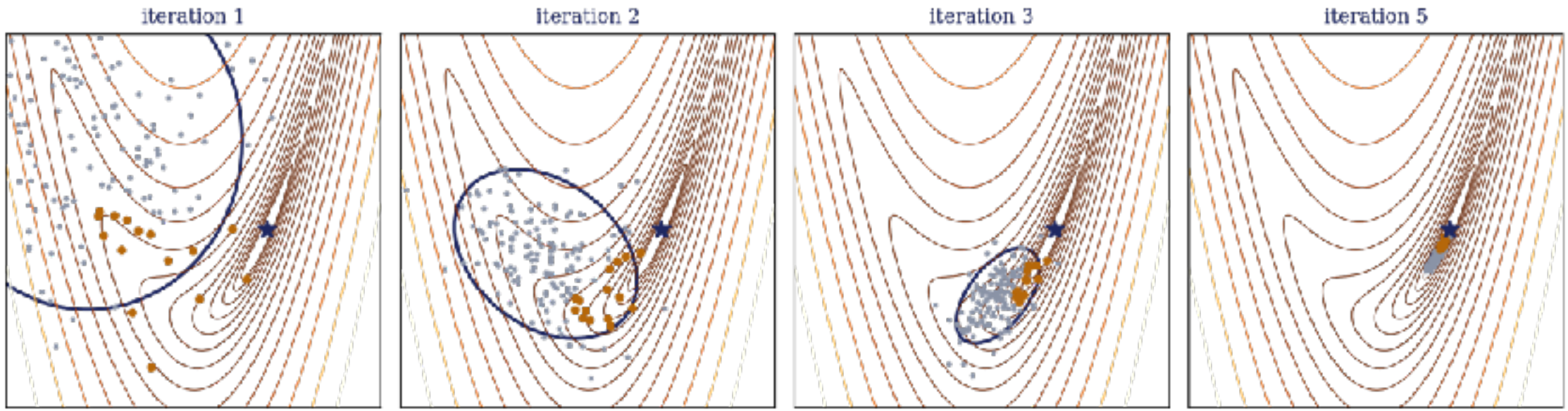
$$\Sigma^{(k+1)} = \frac{1}{m_{\text{elite}}} \sum_{i=1}^{m_{\text{elite}}} \left(\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(k+1)}\right)\left(\mathbf{x}^{(i)} - \boldsymbol{\mu}^{(k+1)}\right)^{\top}$$

```
repeat k_max times:
    samples ← draw m points from P
    sort samples by f(x)
    P ← fit(family of P, best m_elite samples)
return P
```

Typical settings: m = 100 samples, m_elite = 10. The name comes from the fitting step: minimizing cross-entropy (Kullback–Leibler divergence) corresponds to maximum likelihood.

Note: only the ranking of the samples matters — no gradient of f is ever needed.

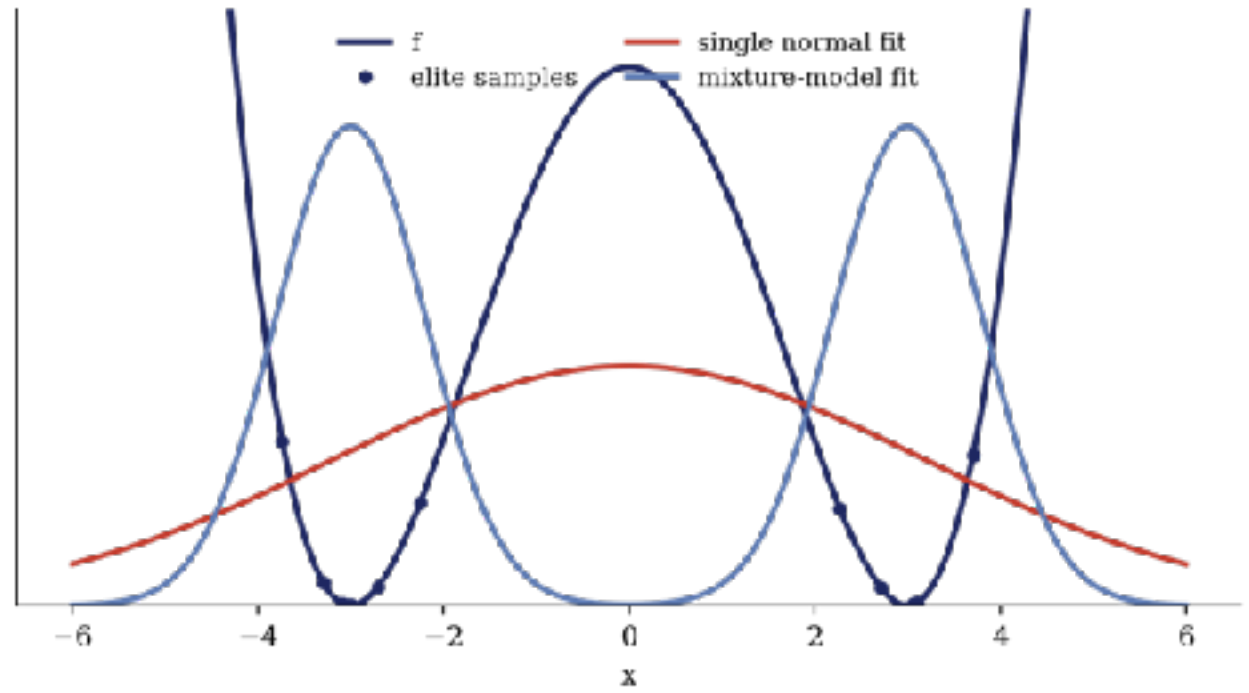
Cross-Entropy in Action



Grey: 150 samples from the current proposal. Amber: the 15 elites. Navy ellipse: proposal covariance (2σ contour). Star: minimum of a banana-shaped test function.

The distribution quickly stretches along the curved valley and then contracts onto the minimum — in about five iterations here.

When a Single Gaussian Is Not Enough



The distribution family must be flexible enough to capture the relevant features of f.

On a two-minimum function, the elite samples cluster around both minima. A single normal fitted to them centers between the minima, where f is actually high.

A mixture model can center a component on each minimum and keep both alive — at the cost of a more complex fitting step.

Design choice: simplicity of the family (fast, analytic fit) versus its ability to represent a multimodal landscape.

Gradient Descent on the Distribution

Like cross-entropy, natural evolution strategies optimize a proposal distribution p(x | θ). But instead of fitting to elite samples, they apply gradient descent to θ, minimizing the expected objective value:

$$\text{minimize}_{\boldsymbol{\theta}} \; \mathbb{E}_{\mathbf{x} \sim p(\cdot \mid \boldsymbol{\theta})}[f(\mathbf{x})]$$

The log-likelihood trick gives the gradient of the expectation without ever differentiating f:

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}[f(\mathbf{x})] = \mathbb{E}[f(\mathbf{x}) \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x} \mid \boldsymbol{\theta})]$$

$$\approx \frac{1}{m} \sum_{i=1}^{m} f(\mathbf{x}^{(i)}) \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}^{(i)} \mid \boldsymbol{\theta})$$

— estimated from m samples drawn from the current distribution.

What You Need — and What You Don't

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{1}{m} \sum_{i=1}^{m} f(\mathbf{x}^{(i)}) \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}^{(i)} \mid \boldsymbol{\theta})$$

Not needed: ∇f

The objective can be a black box. Only function values $f(x^{(i)})$ of the samples enter the update.

Needed: ∇θ log p(x | θ)

The gradient of the log-likelihood — available in closed form for a Gaussian. For the mean it is:

$$\nabla_{\boldsymbol{\mu}} \log p(\mathbf{x} \mid \boldsymbol{\mu}, \Sigma) = \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu})$$

To keep the covariance valid (positive semidefinite), write $\Sigma = A^T A$ and update A instead. A step size $\alpha = 0.01$ and $m = 100$ samples are typical defaults.

CMA-ES: A Smarter Gaussian

CMA-ES (covariance matrix adaptation evolution strategy) is robust and sample-efficient. Like cross-entropy and NES, it improves a distribution from samples — but it adapts a full covariance matrix, plus a separate step-size scalar.

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2 \Sigma)$$

State: mean μ, covariance Σ, step size σ. At each iteration m designs are drawn from this Gaussian and sorted so that $f(x^{(1)}) \le f(x^{(2)}) \le \dots$

$$w_i' = \ln \frac{m+1}{2} - \ln i \quad \Rightarrow \quad w_i = \frac{w_i'}{\sum_j w_j'}$$

$$\boldsymbol{\mu}^{(k+1)} \leftarrow \sum_{i=1}^{m} w_i \, \mathbf{x}^{(i)}$$

The new mean is a weighted average of the best samples, with weights that sum to 1 and decrease with rank. Setting the first m_elite weights to 1/m_elite and the rest to zero recovers cross-entropy — CMA-ES instead gives better designs more weight, so the mean moves more aggressively.

Recommended: m = 4 + ⌊3 ln n⌋ samples per iteration, with m_elite = ⌊m/2⌋.

Adapting Step Size and Shape

Step size σ — evolution path

$$\sigma^{(k+1)} \leftarrow \sigma^{(k)} \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\mathbf{p}_\sigma\|}{\mathbb{E}\|\mathcal{N}(0,\mathbf{I})\|} - 1\right)\right)$$

The path pσ accumulates recent steps. If it is longer than random Gaussian steps would give, steps keep pointing the same way → grow σ. If shorter, steps cancel → shrink σ.

Shape Σ — two updates

$$\Sigma^{(k+1)} \leftarrow (1 - c_1 - c_\mu)\,\Sigma^{(k)} + c_1\,\mathbf{p}_\Sigma \mathbf{p}_\Sigma^\top + c_\mu \sum_i w_i \boldsymbol{\delta}^{(i)} \boldsymbol{\delta}^{(i)\top}$$

Simplified from the full update (a small correction term, typically zero, is omitted).

Previous Σ

Memory: most of the old covariance is kept at each step.

Rank-one update

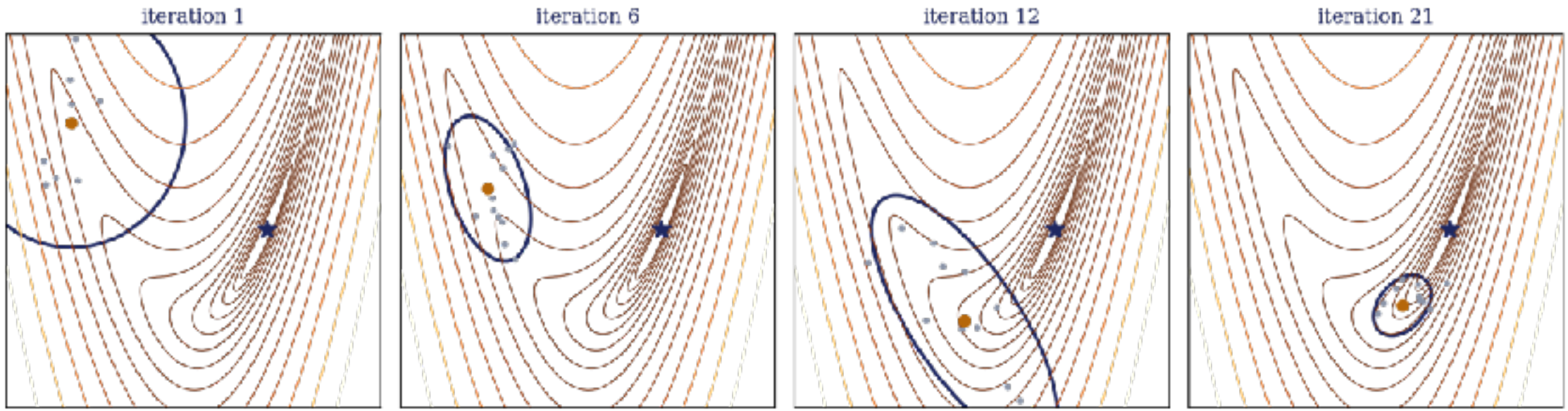
Uses the path pΣ — correlations between consecutive steps — so Σ can elongate quickly along a favorable axis.

Rank-μ update

Uses the best sampled steps $\delta^{(i)}$, measured about the old mean — estimating variances of steps, not of the designs.

Together: σ sets the overall scale of the search; Σ sets its shape and orientation.

CMA-ES on a Curved Valley



Grey: the 12 samples drawn each iteration. Amber: the distribution mean. Navy ellipse: 2σ contour of $\sigma^2\Sigma$. Star: the minimum at (1, 1), reached to four decimals within 45 iterations.

Watch the ellipse: it first grows and stretches along the valley, then contracts around the optimum — the distribution learns the landscape's geometry.

METHOD COMPARISON

Stochastic Methods at a Glance

Method	Where the randomness is	State carried between iterations	Worth remembering
Noisy descent	Noise added to each step	One point (+ descent state)	Cheap add-on to any gradient method; needs a gradient
Mesh adaptive direct search	Random positive spanning directions	One point + step size α	Gradient-free; mesh refines on failure, coarsens on success
Simulated annealing	Candidate moves and acceptance	One point + temperature	Escapes local minima; needs a slow enough schedule
Cross-entropy method	Samples from a proposal distribution	Distribution parameters θ	Rank-based refit; a single Gaussian can miss multiple minima
Natural evolution strategies	Samples from a proposal distribution	Distribution parameters θ	Gradient step on θ via log-likelihood; no ∇f needed
CMA-ES	Samples from $N(\mu, \sigma^2\Sigma)$	μ, Σ, σ (+ evolution paths)	Robust, sample-efficient; most machinery

RECAP

Part A — Key Takeaways

1. Randomness helps escape local optima and saddle points — but must be controlled, or it wastes earlier evaluations.
2. Noisy descent perturbs each step; SGD does so through mini-batches. Step sizes must decay appropriately.
3. Simulated annealing accepts worse points with probability $\exp(-\Delta y/t)$; the cooling schedule is the crucial design choice.
4. Cross-entropy, NES and CMA-ES evolve a sampling distribution — by refitting to elites, or by gradient steps on its parameters.
5. CMA-ES adapts both the step size and the full covariance, learning the geometry of the landscape from samples.

PART B · CHAPTER 9

Population Methods

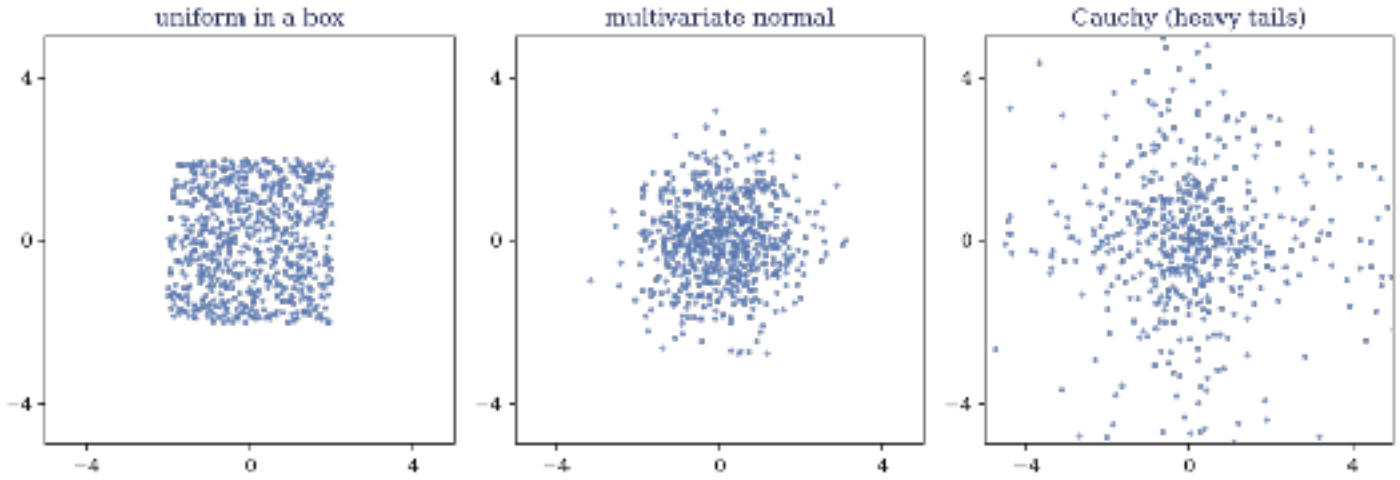
Many individuals, shared information, global exploration

From One Point to Many

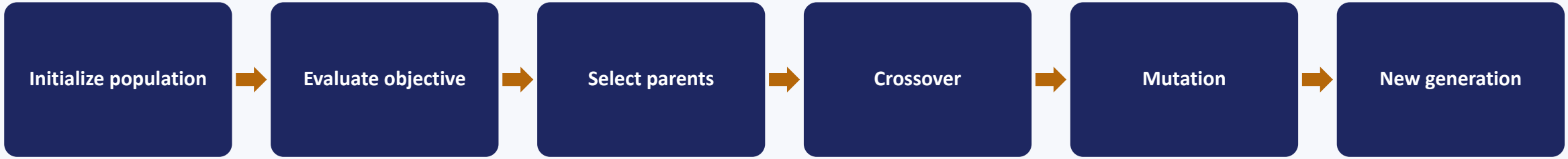
Population methods optimize with a collection of design points called individuals. A large number of individuals spread across the design space reduces the risk of becoming stuck in a local minimum; information is shared between individuals; most such methods are stochastic and easy to parallelize.

Initialization matters: like a descent method needs a start point, a population needs an initial population — spread over the design space so samples fall near the best regions.

$$x_i^{(j)} \sim U(a_i, b_i)$$



The Genetic Algorithm Loop



↺ *repeat until the budget is spent or the population has converged*

Binary chromosomes

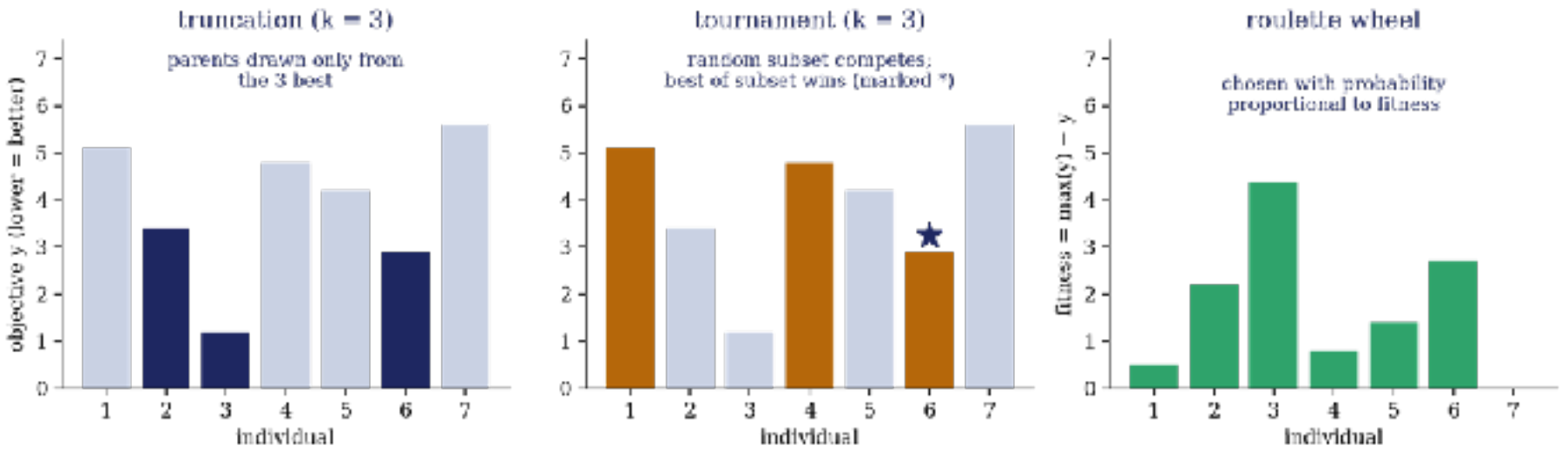
A bit string, like DNA. Crossover and mutation are trivial to express — but decoding a bit string into a design point is not always straightforward, and some strings may not represent valid points.

Real-valued chromosomes

A vector in R^d that is directly a design point. More natural for continuous problems; initialized with the samplers from the previous slide.

Each generation, selection builds m parent pairs for the m children of the next generation — duplicates are allowed.

Selection: Biasing Toward the Fittest



Truncation

Choose parents from the best k chromosomes of the population.

Tournament

Each parent is the fittest of k randomly chosen chromosomes — run separately for each parent.

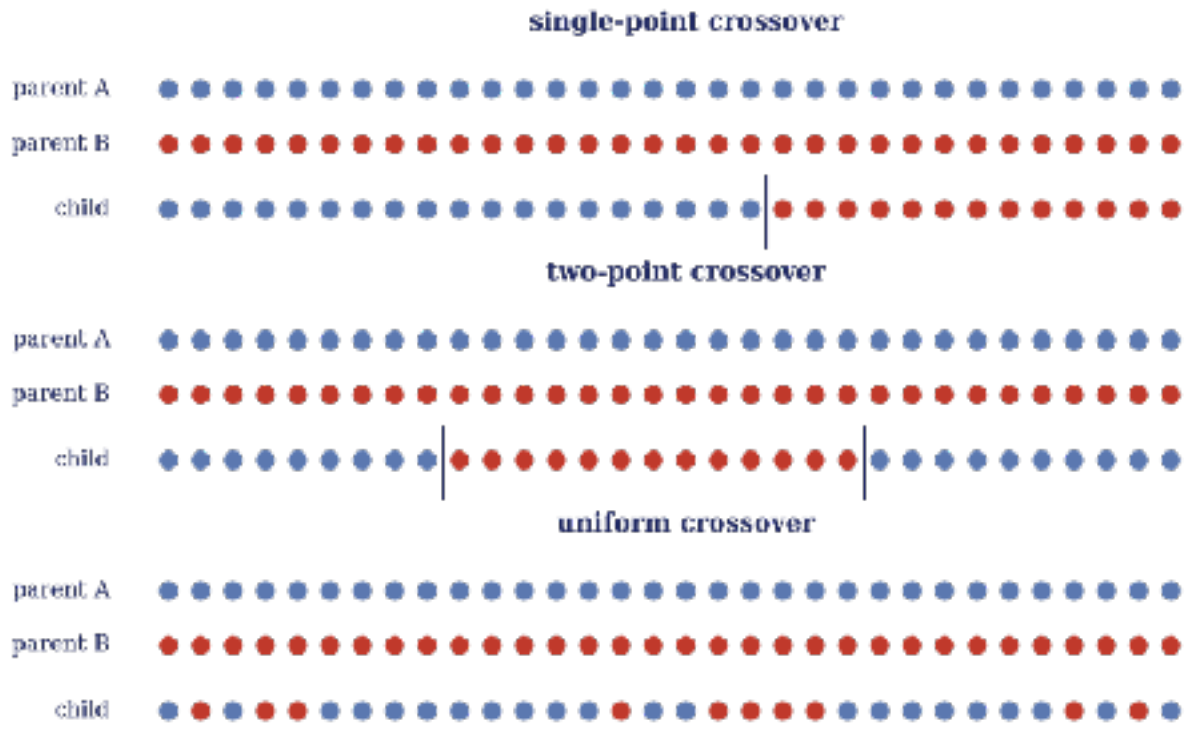
Roulette wheel

Pick with probability proportional to fitness (fitness-proportionate selection).

$$\text{fitness}_i = \max\{y^{(1)}, \ldots, y^{(m)}\} - y^{(i)}$$

since we minimize f, fitness must be inversely related to y

Crossover: Combining Parents



Single-point: the first part of the child comes from parent A, the rest from B, split at a uniformly random crossover point.

Two-point: two random crossover points; the middle segment comes from the other parent.

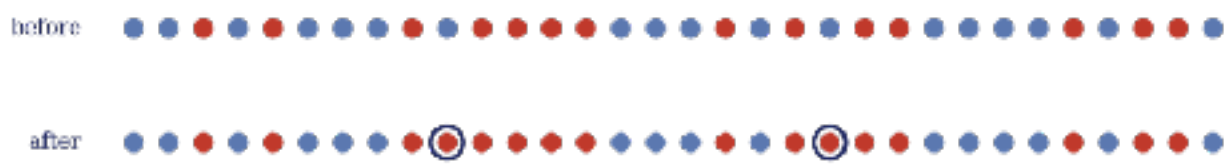
Uniform: each gene has a 50% chance of coming from either parent.

For real-valued chromosomes these still work, and we can also interpolate:

$$\mathbf{x} \leftarrow (1-\lambda)\,\mathbf{x}_a + \lambda\,\mathbf{x}_b$$

Mutation: Keeping Diversity Alive

If new chromosomes were produced only by crossover, traits absent from the initial population could never appear, and the most-fit genes could saturate the population. Mutation lets new traits appear spontaneously, so the algorithm keeps exploring.



Bit-flip mutation on a binary chromosome (flipped bits circled).

Binary

Each bit flips with a small probability. For a chromosome of m bits the mutation rate is typically 1/m — about one flip per child on average.

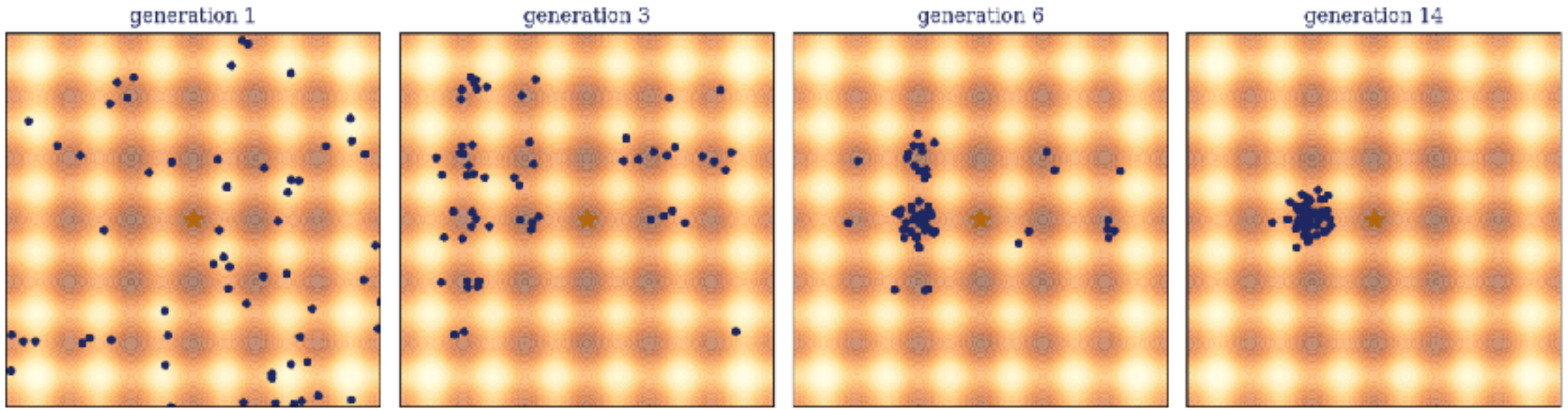
Real-valued

Bitwise flips are possible, but it is more common to add zero-mean Gaussian noise to the child.

The trade-off

Too little mutation: the population loses diversity. Too much: the search degenerates into random sampling.

A Genetic Algorithm at Work



60 individuals on the Rastrigin function (many local minima, global minimum at the star). Truncation selection of the best 12, crossover swapping coordinates between parents, Gaussian mutation (σ = 0.18).

The population contracts quickly — but here into a neighbouring local minimum: premature convergence. More diversity (larger population, more mutation) lowers that risk.

Particles with Momentum

Particle swarm optimization introduces momentum to accelerate convergence. Each particle keeps its position x, its velocity v, and the best position it has ever seen. At every iteration it is accelerated toward both its own best and the best found by any particle:

$$\mathbf{x}^{(i)} \leftarrow \mathbf{x}^{(i)} + \mathbf{v}^{(i)}$$

$$\mathbf{v}^{(i)} \leftarrow w\,\mathbf{v}^{(i)} + c_1 r_1 \left(\mathbf{x}^{(i)}_{\text{best}} - \mathbf{x}^{(i)}\right) + c_2 r_2 \left(\mathbf{x}_{\text{best}} - \mathbf{x}^{(i)}\right)$$

w — inertia

How much of the previous velocity is kept (momentum).

c_1 — cognitive

Pull toward the particle's own best position.

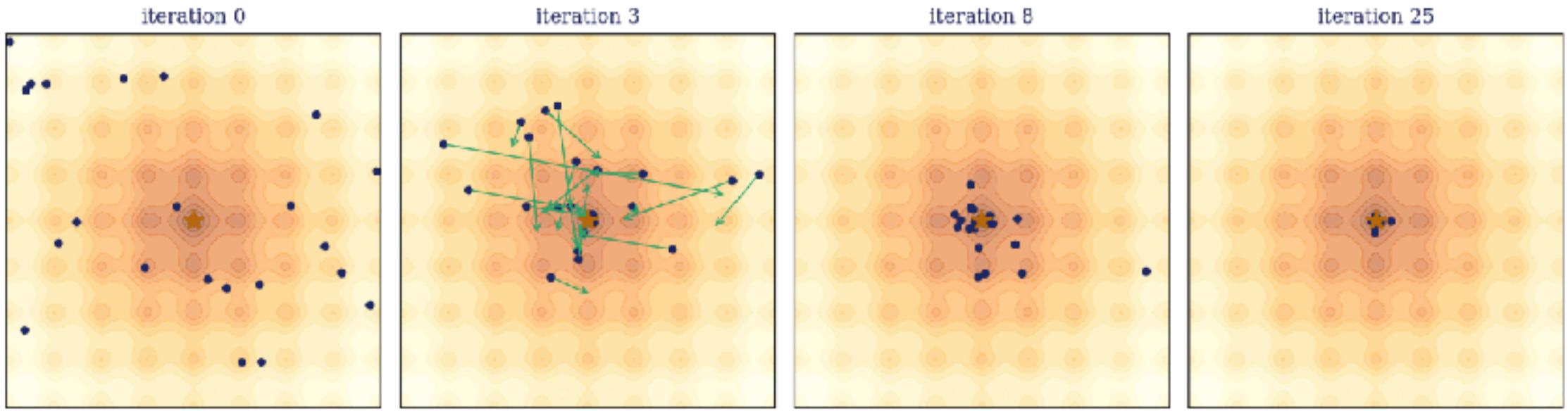
c_2 — social

Pull toward the best position found by the swarm.

$r_1, r_2 \sim U(0,1)$

Random weights, drawn separately for each acceleration.

A Swarm Contracting on Ackley's Function

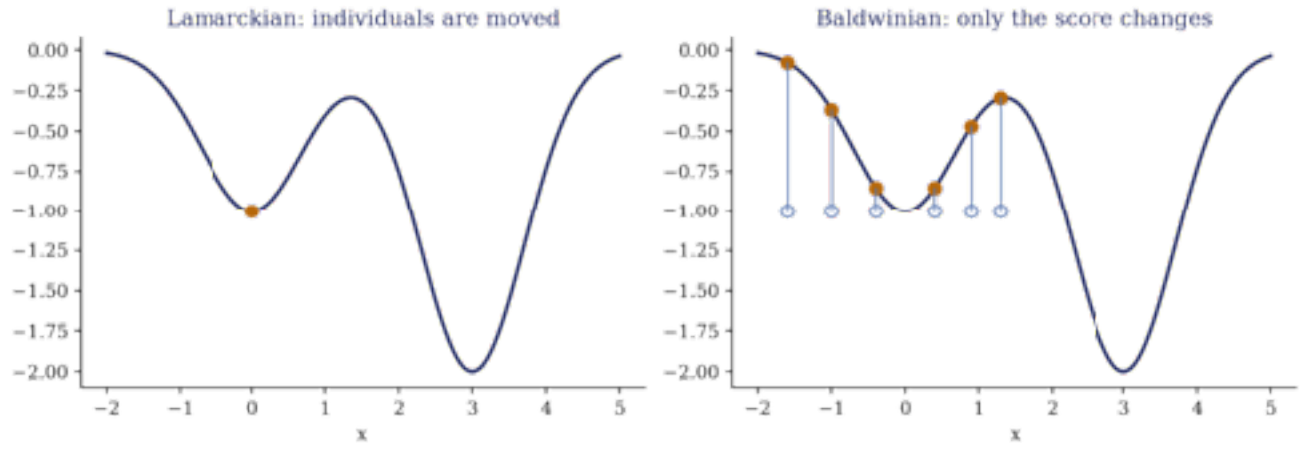


22 particles; w = 0.55, $c_1 = c_2 = 1.6$ (chosen for a clear picture). Green arrows are velocities. The swarm collapses onto the global minimum at the star.

If PSO converges quickly to a non-global minimum, the parameters can be adjusted — e.g. more inertia or a stronger random component keeps particles exploring longer; a common strategy is to let w decay over time.

Global Search Meets Local Descent

Population methods are good at global search — avoiding local minima and finding the best regions — but weaker at local refinement than descent methods. Hybrid methods (also called memetic algorithms) add a local search to each individual:



Lamarckian learning

The individual and its value are replaced by the locally optimized counterpart.

Baldwinian learning

Local search only updates the perceived objective value; the individual itself is unchanged.

Example: f(x) = −e^(−x²) − 2e^(−(x−3)²), population near x = 0. Lamarckian search collapses everyone into the local minimum, hurting the chance of finding the global one near x = 3; Baldwinian selection values each design by its local-search result while keeping the population diverse — helping to prevent premature convergence.

METHOD COMPARISON

Population Methods at a Glance

Method	Core mechanism	Key parameters	Worth remembering
Genetic algorithm	Selection, crossover and mutation of chromosomes	Selection scheme, crossover type, mutation rate	Flexible representations; diversity vs. premature convergence
Differential evolution	$z = a + w(b - c)$ from three random individuals, binary crossover, greedy replacement	w (typically 0.4–1), p	Few parameters; step size adapts to the population's spread
Particle swarm	Velocity (momentum) pulled toward personal and swarm best	w, c_1, c_2	Fast contraction; may converge prematurely
Firefly	Move toward brighter individuals + random walk	β, α, γ (intensity decay)	With $\beta = 0$ it is a random walk
Cuckoo search	Lévy-flight proposals; abandon a fraction of worst nests	p_a, flight distribution	Heavy tails give occasional long jumps

RECAP

Part B — Key Takeaways

1. Population methods optimize with many individuals, sharing information and resisting local minima; they parallelize easily.
2. Genetic algorithms combine selection, crossover and mutation — mutation is what keeps diversity alive.
3. Differential evolution, PSO, firefly and cuckoo search each define a rule that attracts individuals toward good designs while preserving exploration.
4. Premature convergence is the main risk; population size, mutation and momentum parameters tune the balance.
5. Hybrid (Lamarckian / Baldwinian) schemes add local search to population methods to refine solutions.

COURSE VIEW

Choosing Among the Algorithm Families

Family	Information used	Typical strengths	Typical limits
Gradient-based (Ch. 4–6)	Gradients, sometimes Hessians	Fast, precise local convergence; scales to many variables	Converges to local minima; needs derivatives (exact or estimated)
Direct (Ch. 7)	Function values only	No derivatives needed; simple to apply	Slower; not covered in this lecture
Stochastic (Ch. 8)	Function values + randomness	Escapes local minima and saddles; tolerates black boxes	Needs a well-chosen schedule or distribution; many evaluations
Population (Ch. 9)	Function values of many individuals	Global exploration; parallel; derivative-free	Costly; premature convergence; weak local refinement

Rule of thumb: when derivatives are available and the landscape is benign, descend. When it is rugged or a black box, search globally — then polish with a local method (the hybrid idea). No Free Lunch: no family wins on every problem.

LECTURE 4 RECAP

Randomness and Populations, Front to Back

Noise & annealing

Perturb descent, or accept worse points with probability exp(−Δy/t) under a cooling schedule.

Distribution-based

Cross-entropy, NES and CMA-ES evolve a Gaussian proposal from samples.

Evolutionary & swarm

Genetic algorithms and DE recombine individuals; PSO, firefly and cuckoo add attraction and flights.

Hybrids

Combine global population search with local descent — Lamarckian or Baldwinian.

Common thread: control the balance between exploration (randomness, diversity) and exploitation (using what has been learned) — every parameter in today's methods is a knob on that balance.

DISCUSSION

Questions to Think About

1. Why does mutation play such a fundamental role in genetic algorithms? How would you choose the mutation rate if you suspect a better optimum exists?

2. If particle swarm optimization converges quickly to a non-global minimum, how might you change its parameters?

3. Mixture proposal distributions can capture multiple minima. Why might using them in the cross-entropy method be limited?

4. When would you prefer CMA-ES to simulated annealing — and when the reverse?

Questions 1–3 are adapted from the end-of-chapter exercises in Chapters 8 and 9.

Questions?

Reading: Algorithms for Optimization, Chapters 8 and 9 (with the end-of-chapter exercises)

Kochenderfer, M. J., & Wheeler, T. A. Algorithms for Optimization. MIT Press, 2019.