# MATH3027: Optimization (UK 21/22)

## Week 12: Nature-Inspired Global Optimization

Prof. Richard Wilkinson
School of Mathematical Sciences
University of Nottingham, United Kingdom
Please send any comments or mistakes to
r.d.wilkinson@nottingham.ac.uk

## Motivation

Over the last 11 weeks, we have discussed different aspects of continuous optimization:

- How to formulate an optimization problem.
- How to characterize optimality.
- How to deal with constraints.
- How to build effective numerical methods for finding minima.
- How to include stochastic effects to deal with large-scale problems.

However, if we were asked to minimize a function such as the one in Figure **??**, there is not much we can do about it. For sure, if we would have a symbolic representation $f(\mathbf{x})$ we could compute its gradient, and eventually set a gradient descent which, in 99.9% of the cases, will get stuck in some of the many local minima of this function. Discussing about convexity here is pointless. So, what can we do to find the global minimum of a function like this? In mathematics, this is an open problem, and we can only provide a partial answer through *metaheuristics*.

Metaheuristics is just a very fancy and cool name to describe an idea that works, but for which we do not have any serious theoretical justification like the ones we have developed in the previous weeks. It's a very inefficient yet practical approach to the problem of finding this minimum. For example, what would happen if we pick 1000 marbles $\mathbf{x}_i$ and we drop them randomly over this surface? After a few seconds, their motion will stabilize, and they will all converge to different stationary points, which can be local or global minima. We could then record their final position $\mathbf{x}_i^\infty$, evaluate $f(\mathbf{x}_i^\infty)$, and find the minimum value by inspection.

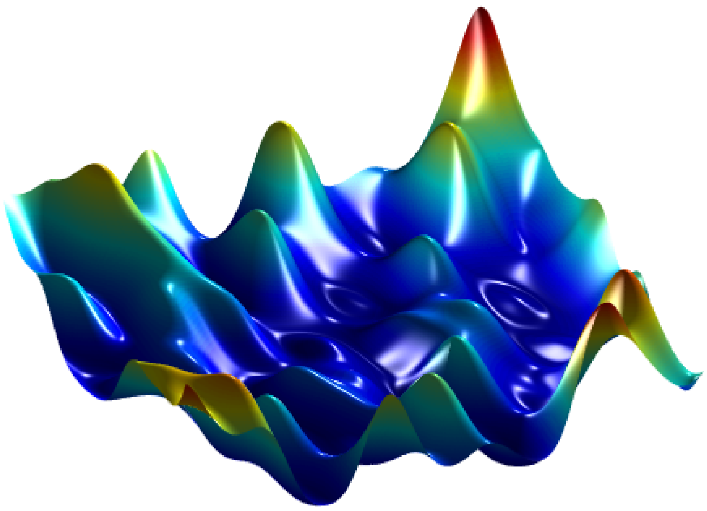

Figure 1: How do we find the global minimum of this function?

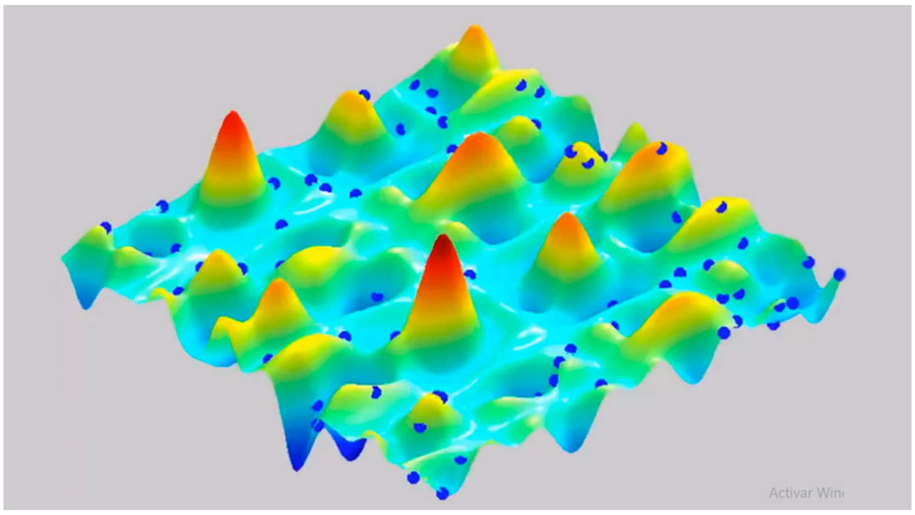

Figure 2: What if we drop 1000 marbles and measure their final location?

Have we found the global minimum? We don't know, maybe. What if we repeat the experiment, with $10^6$ marbles this time, scattered as randomly as possible. We still have no theoretical proof that we will find the global minimizer, but chances are better than for the 1000 marbles experiment. As you can see, this is a very pragmatic and inefficient way to find the global minimum, yet it is still very likely we will stumble upon it, or at least to some local minimum that is close enough in value. Welcome to the world of metaheuristics.

Figure 3: A flock of birds exhibiting some kind of swarm intelligence

Over the last decades, together with the development of computational science and more powerful machines, we have started to study a class of metaheuristic algorithms which are motivated by the swarm intelligence behaviour we observe in nature. For example, we observe flock of birds as in Figure **??** and we wonder, what kind of communication mechanism does this flock have in order to communicate and self-organize in real-time? We conjecture there are at least three features in the interactions:

- the action of *social* forces such as attraction, repulsion, and alignment, among many.
- interaction between local and global sensing of the flock, that is, an accurate measurement of what happens locally, combined with a rough sensing of what happens with the whole flock.
- a certain degree of randomness (environment -wind-, own decisions).

The combination of these factors yields to fascinating self-organization phenomena! What if instead of dropping our inanimate marbles over the surface to find the global minimum, we let a flock of 1000 birds fly over the surface, communicate to each other the values of $f(\mathbf{x})$ they observe along their trajectories $\mathbf{x}_i(t)$, and let them self-organize around the global minimizer? Of course, our *birds* are non-autonomous, in the sense that we will design their interactions in order to help them find the minimum, but our settings will be inspired from what we observe in nature. Hence the name of this week's topic: Nature-Inspired Global Optimization.

## Nature-Inspired Algorithms

A vast majority of nature-inspired algorithms for global optimization are equation-based, where all solution vectors $\mathbf{x}_i \in \mathbb{R}^d$ are represented as a population set of $n$ solutions (also known as agents) in a $d$-dimensional search space. In this sense, all different algorithms use the same type of vector representations of solutions. In addition, the selection of the solutions is mainly based on their fitness values (how good is $f(\mathbf{x}_i)$). The fittest solutions (higher objective values for maximization, or lower objective values for minimization) are most likely to be passed onto the next generation in the population.

Consequently, the main difference among different nature-inspired optimization algorithms is the way in which the trajectory of the agents is modified using diverse search mechanisms. In general, a solution vector $\mathbf{x}_i^t$ at iteration or generation $t$ is a **position** vector, and the new solution $\mathbf{x}_i^{t+1}$ is generated by a modification increment or mutation vector $\Delta\mathbf{x}_i^t$. That is

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \Delta\mathbf{x}_i^t,$$

which dictates the main differences between different algorithms. Traditionally, this increment is a step size (or a step vector). In the case of gradient-based algorithms this step is linked to the negative gradient

$$\Delta\mathbf{x}_i^t = -\eta\nabla f(\mathbf{x}_i^t)$$

where $\nabla f$ is the gradient of the objective function $f(\mathbf{x})$, and $\eta > 0$ is a stepsize.

In some nature-inspired algorithms, the modification in $\Delta\mathbf{x}_i^t$ is often related to a **velocity** such as

$$\Delta\mathbf{x}_i^t = \mathbf{v}_i^t\Delta t$$

where $\mathbf{v}_i^t$ is the velocity for the solution $i$ (particle, agent, etc.) at iteration $t$. Here, $\Delta t$ is the time increment. As all algorithms are iterative or time-discrete dynamical systems, the time step or increment $\Delta t$ is the difference in the iteration counter $t$, which means that $\Delta t = 1$ can be used for all these algorithms. Consequently, there is no need to worry about the units of these quantities and thus we consider all quantities in the same units. Let us discuss the equations for updating the motion of the agents in different algorithms.

**Differential evolution(DE):** In DE, the main mutation is realized by

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + F\left(\mathbf{x}_j^t - \mathbf{x}_k^t\right)$$

which can be written as

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \Delta\mathbf{x}_i^t, \quad \Delta\mathbf{x}_i^t = F\left(\mathbf{x}_j^t - \mathbf{x}_k^t\right)$$

where $\mathbf{x}_i^t$, $\mathbf{x}_j^t$ and $\mathbf{x}_k^t$ are three distinct solution vectors from the population, $\mathbf{x}_j$ and $\mathbf{x}_k$ is sampled at each iteration. The parameter $F \in (0,2)$ controls the mutation strength. The update $\mathbf{x}_i^{t+1}$ is adopted only if $f(\mathbf{x}_i^{t+1}) < f(\mathbf{x}_i^t)$.

**Particle swarm optimization (PSO):** Many nature-inspired optimization algorithm use collective behaviour or *swarm intelligince*. The main inspiration of PSO comes from the swarming behaviour of birds and fish. Particle swarm optimization was developed by Kennedy and Eberhart in 1995 and has become one of the widely used swarm-intelligence-based algorithm dues to its simplicity and flexibility. Instead of using mutation/crossover of agents, it uses randomness and global communication among the swarm particles. The position and velocity of particle $i$ at any iteration or pseudo-time $t$ can be updated iteratively using

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + \Delta\mathbf{v}_i^t$$
$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \Delta\mathbf{x}_i^t$$

with

$$\Delta\mathbf{x}_i^t = \mathbf{v}_i^{t+1}\Delta t = \mathbf{v}_i^{t+1}$$

and

$$\Delta\mathbf{v}_i^t = \alpha\varepsilon_1^t \odot \left[\mathbf{g}_* - \mathbf{x}_i^t\right] + \beta\varepsilon_2^t \odot \left[\mathbf{x}_i^* - \mathbf{x}_i^t\right]$$

where $\varepsilon_1$ and $\varepsilon_2$ are two uniformly distributed random vectors in $[0, 1]$, and $\odot$ is the componentwise product. The parameters $\alpha$ and $\beta$ are the learning parameters or acceleration constants, which can be typically take as $\alpha \approx \beta \approx 2$. Here, $\mathbf{g}_* = \text{argmin}_i\{f(\mathbf{x}_i^t)\}$ is the best solution of the population at iteration $t$, while $\mathbf{x}_i^*$ is the individual best solution for particle $i$ among its search history up to iteration $t$. The initial locations of all particles should be distributed relatively uniformly so that can cover most of the search space, and the intial velocity can be taken as 0, that is $\mathbf{v}_i^0 = 0$. The pseudocode of the PSO algorithm is presented in Algorithm **??**.

**Algorithm 1:** Particle Swarm Optimization

```
Initialization: Objective function $f(x)$,  $x = (x_1, \dots, x_d)^T$, Initialize locations $x_i$
                and velocity $v_i$ of $n$ particles, parameters $\alpha$, $\beta$.
1  Find $g^*$ from $\min\{f(x_1), \dots, f(x_n)\}$ ( at $t = 0$);
   General Step: for any $t = 0, 1, 2, \dots$ execute the following steps:
2  for loop over all $n$ particles and all $d$ dimensions:
3  Generate new velocity $\mathbf{v}_i^{t+1}$
4  Calculate new location $\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1}$
5  Evaluate objective functions at new locations $\mathbf{x}_i^{t+1}$
6  Find the current best for each particle $\mathbf{x}_i^*$
7  end for
8  Find the current global best $\mathbf{g}^*$
9  Update $t = t + 1$ (pseudo time or iteration counter)
10 end for
11 Output the final results $\mathbf{x}_i^*$ and $\mathbf{g}^*$
```

There are many variants that extend the standard PSO algorithm, the most noticeable improvement is probably the use of an inertia function $\theta(t)$ so that $\mathbf{v}_i(t)$ is replaced by

$\theta(t)\mathbf{v}_i^t$, leading to

$$\mathbf{v}_i^{t+1} = \theta(t)\mathbf{v}_i^t + \Delta\mathbf{v}_i^t$$
$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \Delta\mathbf{x}_i^t \,,$$

where $\theta$ takes values between 0 and 1. Typically, $\theta \approx 0.5 \sim 0.9$. This is equivalent to introduce some damping to stabilize the motion of the particles, leading to faster convergence.

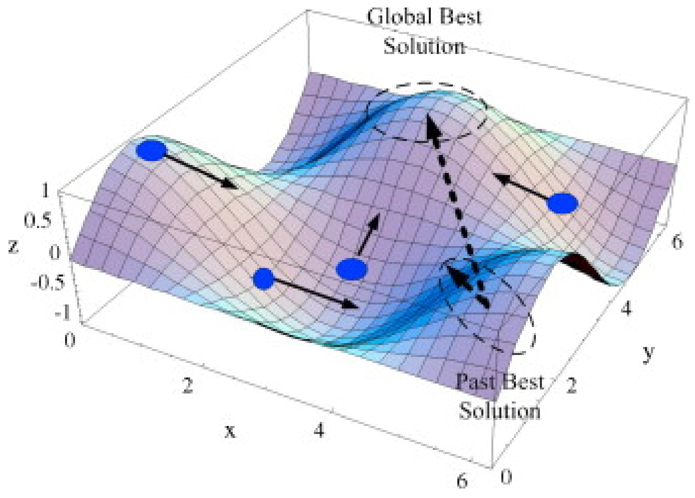

Figure 4: Schematic description of the Particle Swarm Optimization, smart trajectories recording individual and global best.

**Cuckoo search (CS):** CS was based on the aggressive reproduction strategy of some cuckoo species and their interactions with host species such as warblers. The eggs laid by cuckoos can be discovered and thus abandoned with a probability $p_a$, realized by a Heaviside step function $H$ with the use of a random vector $\varepsilon$ in $[0, 1]$. The similarity of two eggs (solutions $\mathbf{x}_j$ and $\mathbf{x}_k$ ) can be roughly measured by their difference $\left(\mathbf{x}_j - \mathbf{x}_k\right)$. Thus, the position at iteration $t$ can be updated by

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \Delta\mathbf{x}_i^t$$

where $\Delta\mathbf{x}_i^t$ can be either a local random walk given by

$$\Delta\mathbf{x}_i^t = \alpha H\left(p_a - \epsilon\right) \odot \left(\mathbf{x}_j^t - \mathbf{x}_k^t\right) ,$$

or

$$\Delta\mathbf{x}_i^t = \alpha L(\lambda) \,,$$

where $\odot$ denotes componentwise vector product, $\alpha$ is a scaling factor, $\mathbf{x}_j$ and $\mathbf{x}_k$ are randomly selected, and $L(\lambda)$ is used to represent a Lévy flight. The generation of this step size can be realized by some sophisticated algorithms such as the Mantegna's algorithm.

**Algorithm 2:** Cuckoo Search via Lévy Flights

```
Initialization: Objective function $f(x)$,   $x = (x_1, \dots, x_d)^T$, Initialize locations $x_i$
                and velocity $v_i$ of $n$ particles, parameters $\alpha$, $\beta$.
General Step: for any $t = 0, 1, 2, \dots$ execute the following steps:
1  Get a cuckoo randomly
2  Generate an update by Lévy flights
3  Evaluate its solution quality or objective value $f_i = f(\mathbf{x}_i)$
4  Choose another cuckoo randomly (say, $\mathbf{x}_j$)
5  if $(f_i < f_j)$
6  Replace $\mathbf{x}_j$ by the new solution $\mathbf{x}_i$
7  end if
8  A fraction $(p_a)$ of the worse nests are abandoned
9  New nests/solutions are built/generated by a local random walk
10 Rank the solutions and find the current best
11 Update $t \longleftarrow t + 1$
12 end
```

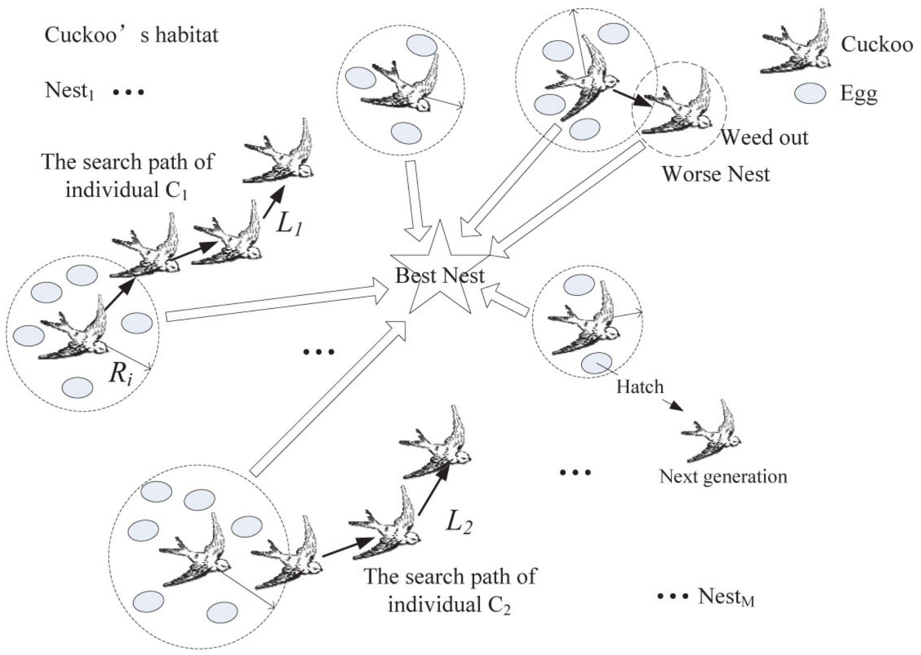

Figure 5: Schematic description of the Cuckoo Search Algorithm, with both local and global search strategies.

**Other Algorithms:** There are many other nature-inspired algorithms, such as simulated annealing, bacteria foraging optimization, biogeography-based optimization, gravitational search, charged particle system, black-hole algorithm, krill herd algorithm , eagle strategy, and others. However, their main differences are in the ways of generating $\Delta\mathbf{x}_i^t$ and $\Delta\mathbf{v}_i^t$ from the population of the existing solutions.

Though the expressions of $\Delta\mathbf{x}_i^t$ and $\Delta\mathbf{v}_i^t$ may be different and some algorithms do not use velocity at all, the detailed underlying search mechanisms may also be very different, even for seemingly similar expression of $\Delta \mathbf{x}_i^t$. For example, the mutation of differential evolution seems to be similar to update for PSO. However, the former was done by random permutation, while the latter was done by a difference vector with perturbed directions using a uniform distribution. Therefore, in order to gain better insight, we should analyze the underlying search mechanisms and their mathematical or statistical foundations.

## Benchmarking

**The last penguin of MATH3027!** E.1 Convince yourself of the magic of particle swarm optimization! Try to find the local minimum of the function:

$$f(x_1, x_2) = -\left\{\sin(x_1)\left[\sin\left(\frac{x_1^2}{\pi}\right)\right]^{20} + \sin(x_2)\left[\sin\left(\frac{2x_2^2}{\pi}\right)\right]^{20}\right\}$$

Try to code PSO by yourself, otherwise give it a try in MATLAB with its own PSO routine:

https://uk.mathworks.com/help/gads/particleswarm.html

Figure 6: It's been my pleasure to teach you Optimization, I hope you've felt challenged and motivated to learn and discover this beautiful branch of mathematics. Take 5 minutes to reflect and a appreciate how much have you learnt over the last 11 weeks, and congratulate yourself, it's hard-earned knowledge. Good luck for the future!