Gliwice, January 30, 2023

Fundamental Optimization procedures

Documentation of the final project

Title: Differential evolution

Author (Authors): Grzegorz Koperwas, Kamil Kowalczyk

Field of studies: Informatics (sem. V)

Theoretical description

Differential evolution (or *DE*) is a method that optimizes the problem by iteratively trying to improve the solutions. *Differential evolution* is a *Metaheuristic*, so it does guarantee that the optimal solution is ever found. It was first described by Rainer Storn and Kenneth Price in 1997.

Differential evolution does not require the problem to be differentiable, as it does not rely on the gradient of the function.

The implemented variant of *DE*, works by having a population of candidate solutions, that are called `agents`. These agents are moved around in the space using a simple mathematical formula combining values from random other agents. If the new position is an improvement on the old one, it replaces the old one.

This process is repeated for every agent multiple times, where an optimal solution might be found after a number of iterations.

Description of the algorithm

Initialization

Let:

- $f : \mathbb{R} \to \mathbb{R}^n$ - function to be minimized.
- $x \in \mathbb{R}$ - a candidate solution.

Choose parameters:

- $NP \geq 4$ - Number of agents, usually it is equal to $10n$.
- $CR \in [0, 1]$ - Crossover probability, will be used to decide if additional values in the candidate position should be adjusted. Typically, the value of 0.9 is used.

- $F \in [0, 2]$ - Differential weight, used to calculate new positions. Typically, the value of 0.8 is used.

The performance of the algorithm is highly dependent on these parameters. There is a lot of research into the optimal values of those.

Next, the initial positions of the agents x in the given search space are chosen at random.

Main loop

1. For each agent x in the population:
   (a) Pick three other, distinct agents a, b and c from the population[1].
   (b) Pick a random index $R \in \{1, \ldots, n\}$. The value under this index in x will be always changed.
   (c) Create a vector $r \in [0, 1]^n$ with random values.
   (d) Compute new candidate position $y \in \mathbb{R}^n$ where y_i is:

$$y_i = \begin{cases} a_i + F\left(b_i - c_i\right) & \text{if } r_i < CR \text{ or } i = R \\ x_i & \text{if } r_i \geq CR \text{ and } i \neq R \end{cases}$$

   (e) If $f(y) \leq f(x)$, then replace x in the population with y.
2. If maximal number of iterations has not been reached, then repeat step 1.
3. The result is the agent, for which the function f reaches the smallest value.

Exemplary calculations

Let:

- f be defined as $f(x, y) = |x| + |y|$,
- agents be a set of points: $\{(1, 1), (-1, 1), (-0.5, 0), (0, 1)\}$,
- $CR = 0.5$,
- $F = 1$,

Lets calculate one step in the algorithm, where we will consider point $x = (1, 1)$.

1. Choose three distinct points a, b, c:
   - $a = (-1, 1)$
   - $b = (0, 1)$
   - $c = (-0.5, 0)$

[1] This is the reason why $NP \geq 4$

2. Pick a random index of $R = 1$.
3. Pick a random vector $r = (0.1, 0.3)$
4. Calculate new position y.
   (a) $y_1 = -1 + (0 + 0.5)$, as $R = 1$,
   (b) $y_2 = 1$, as $0.3 \not< CR$ and $R \neq 2$
5. Compare the value of the functions for the new and old positions:
   - $f(x) = 1 + 1 = 2$
   - $f(y) = |-0.5| + 1 = 1.5$
6. As $1.5 \leq 2$, we will set x to y

After this step, the list of agents will look like this:

$$\{(-0.5, 1), (-1, 1), (-0.5, 0), (0, 1)\}$$

Supporting computer program

Inputs

- `function` - n-dimensional function to find the global minimum for.
- `NP` - number of agents to use.
- `CR` - Crossover probability.
- `F` - Differential probability.
- `domain` - Region to use when picking initial agents.
- `iterations` - How many iterations to run.

Outputs

The program returns the point, in which the function reaches it's global minimum.

Results discussion

The program returns results identical to builtin function `FindMinimum` for functions with only one local minimum.

For functions with multiple local minima the program returns the correct global minimum. When viewing snapshots of the working state of the program, we can clearly see that the points converge on the global minimum with each iteration.

Enclosures

The program is included in `Project_Koperwas.nb`.