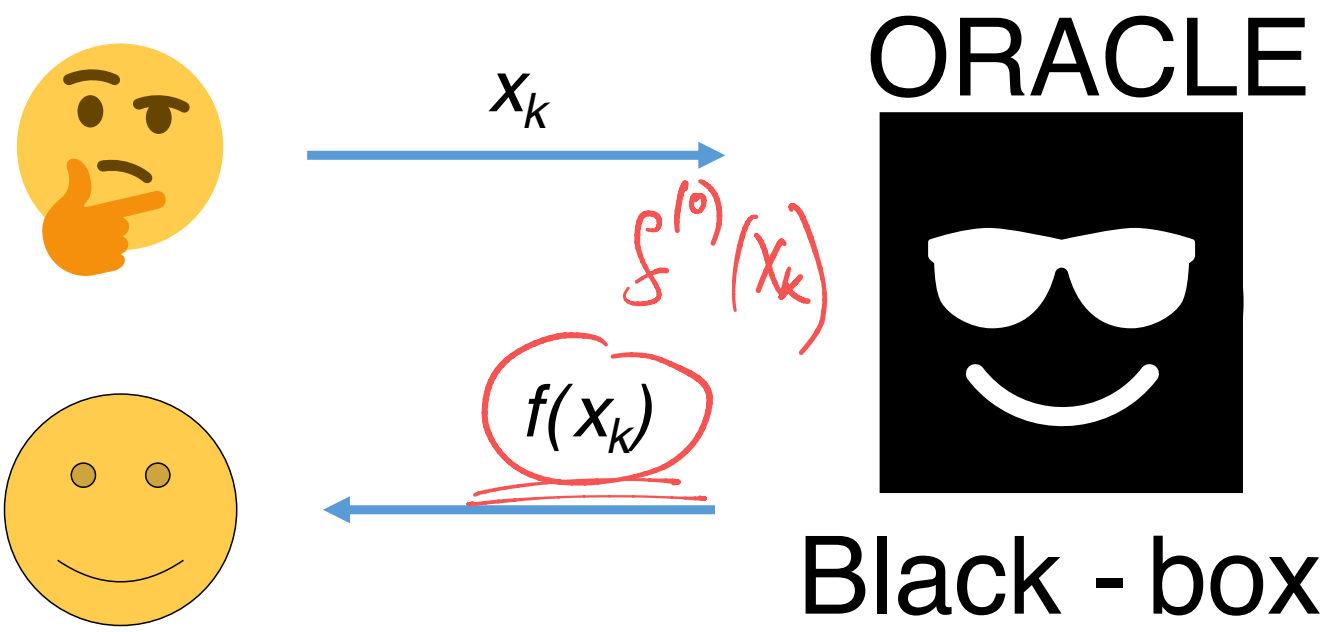
Zero order methods



Now we have only zero order information from the oracle. Typical speed of convegence of these methods is sublinear. A lot of methods are referred both to zero order methods and global optimization.

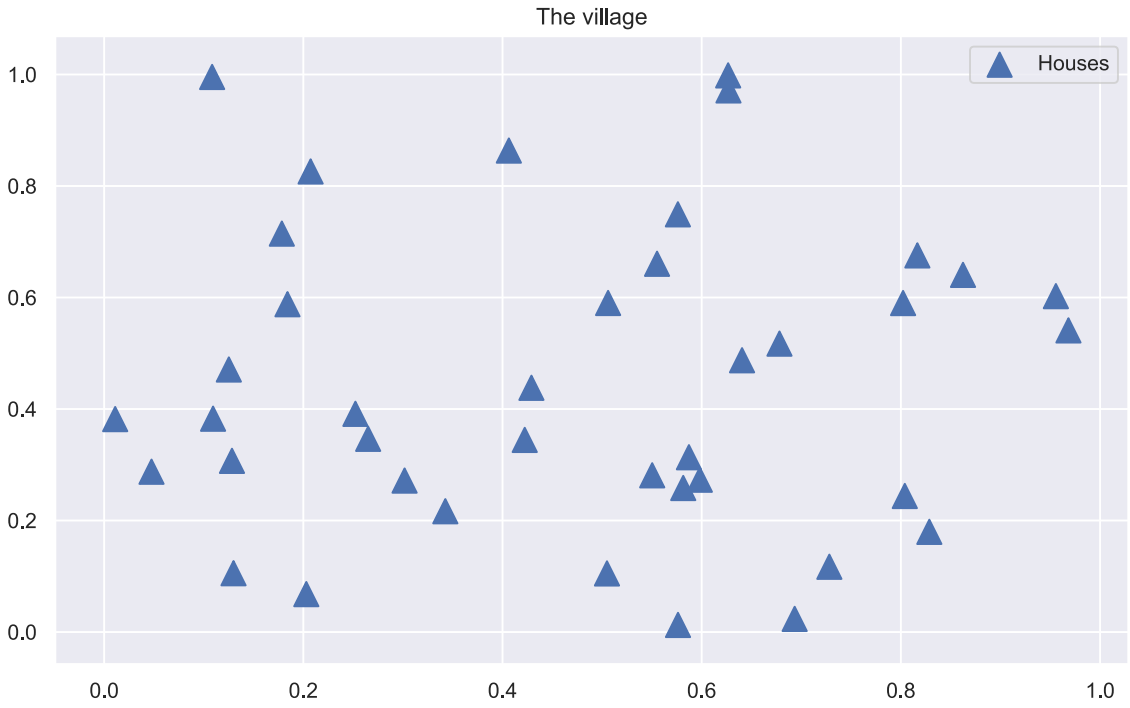
Code

- Global optimization illustration - Open in Colab
- Nevergrad library - Open in Colab
- Optuna quickstart Open in Colab

Travelling salesman problem

Problem

Suppose, we have N points in $\mathbb{R}^d$ Euclidian space (for simplicity we'll consider and plot case with $d = 2$). Let's imagine, that these points are nothing else but houses in some 2d village. Salesman should find the shortest way to go through the all houses only once.



That is, very simple formulation, however, implies NP - hard problem with the factorial growth of possible combinations. The goal is to minimize the following cumulative distance:

$$d = \sum_{i=1}^{N-1} \|x_{y(i+1)} - x_{y(i)}\|_2 \to \min_y,$$

where x_k is the k-th point from N and y stands for the N- dimensional vector of indicies, which describes the order of path. Actually, the problem could be formulated as an LP problem, which is easier to solve.

🧬Genetic (evolution) algorithm

Our approach is based on the famous global optimization algorithm, known as evolution algorithm.

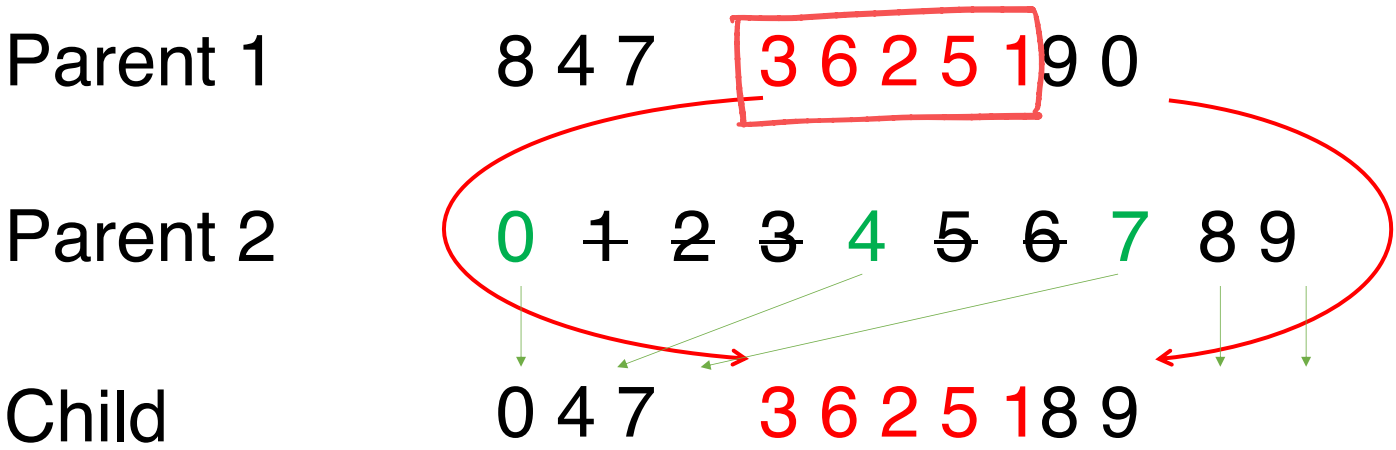
Population and individuals

Firstly we need to generate the set of random solutions as an initialization. We will call a set of solutions $\{y_k\}_{k=1}^n$ as *population*, while each solution is called *individual* (or creature).

Each creature contains integer numbers $1, \dots, N$, which indicates the order of bypassing all the houses. The creature, that reflects the shortest path length among the others will be used as an output of an algorithm at the current iteration (generation).

Crossing procedure

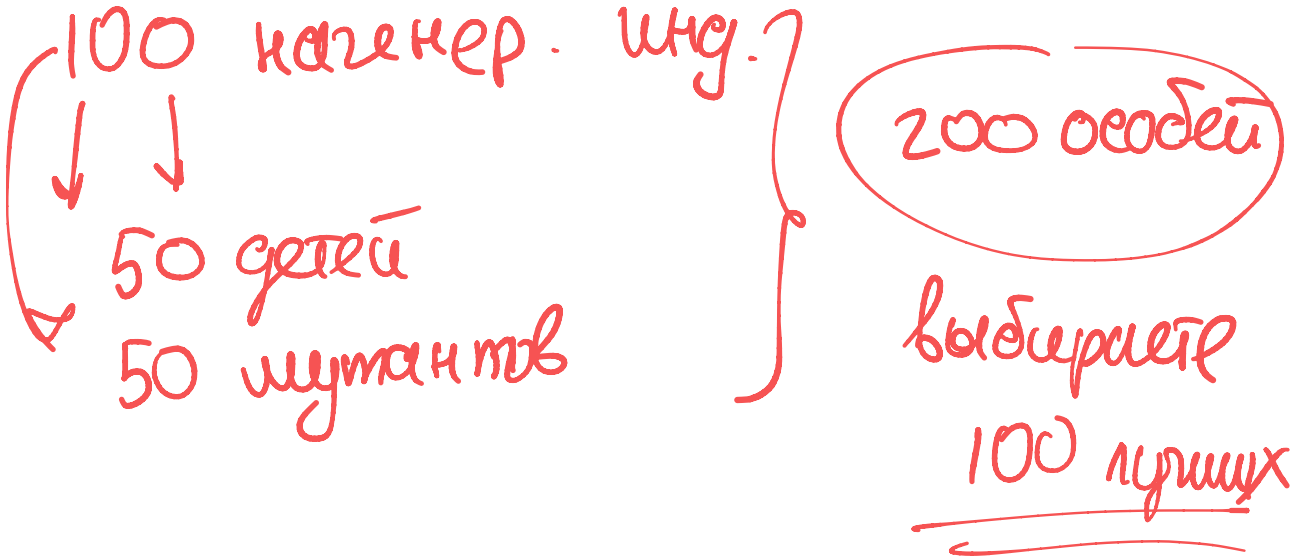
Each iteration of the algorithm starts with the crossing (breed) procedure. Formally speaking, we should formulate the mapping, that takes two creature vectors as an input and returns its offspring, which inherits parents properties, while remaining consistent. We will use ordered crossover as such procedure.



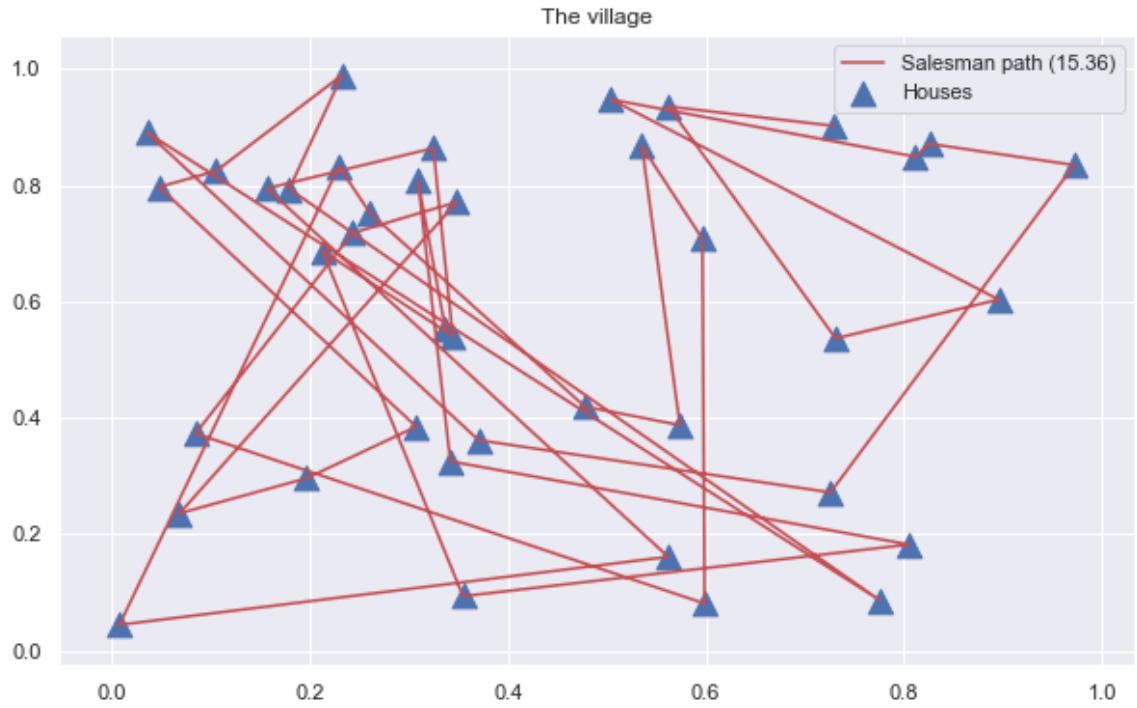
Mutation

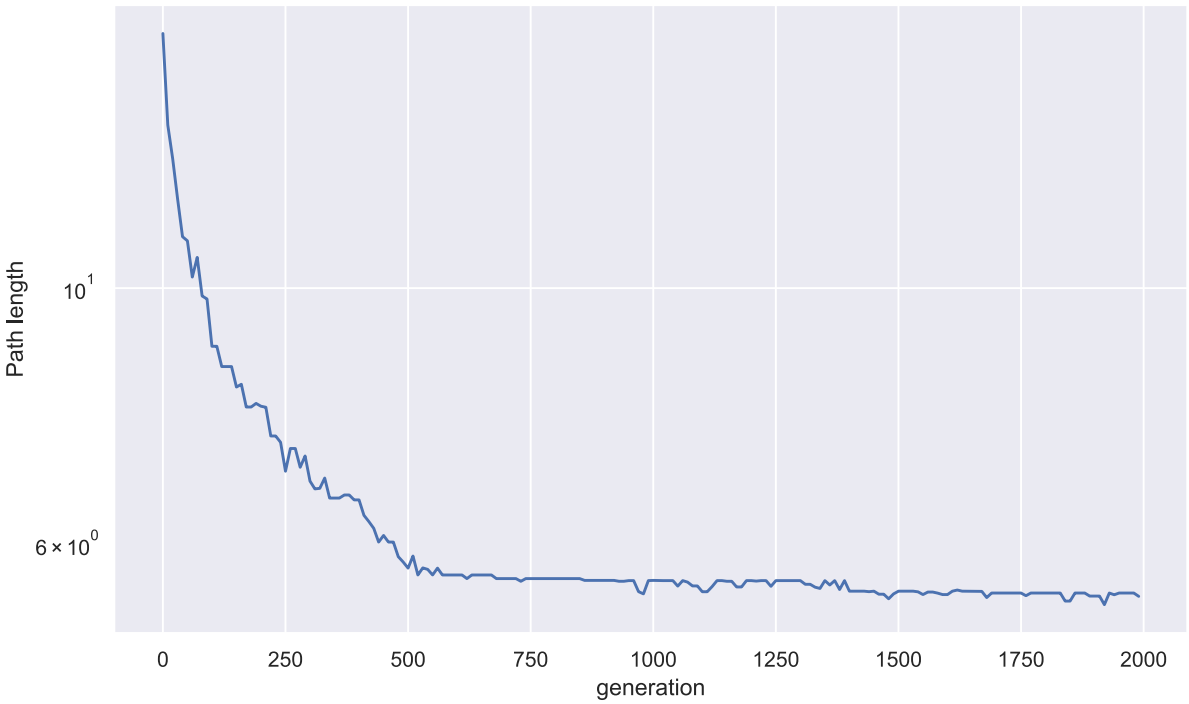
In order to give our algorithm some ability to escape local minima we provide it with mutation procedure. We simply swap some houses in an individual vector. To be more accurate, we define mutation rate (say, 0.05). On the one hand, the higher the rate, the less stable the population is, on the other, the smaller the rate, the more often algorithm gets stuck in the local minima. We choose Error: '_' allowed only in math mode individuals and in each case swap random Error: '_' allowed only in math modedigits.

Selection



At the end of the iteration we have increased populatuion (due to crossing results), than we just calculate total path distance to each individual and select top n of them.





In general, for any $c > 0$, where d is the number of dimensions in the Euclidean space, there is a polynomial-time algorithm that finds a tour of length at most $(1 + \frac{1}{c})$ times the optimal for geometric instances of TSP in

$$\mathcal{O}\left(N(\log N)^{(\mathcal{O}(c\sqrt{d}))^{d-1}}\right)$$

Code

Open in Colab

References

- General information about genetic algorithms
- Wiki

Simulated annealing

Problem

We need to optimize the global optimum of a given function on some space using only the values of the function in some points on the space.

$$\min_{x \in X} F(x) = F(x^*)$$

Simulated Annealing is a probabilistic technique for approximating the global optimum of a given function.

Algorithm

The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. Both are attributes of the material that depend on its thermodynamic free energy. Heating and cooling the material affects both the temperature and the thermodynamic free energy. The simulation of annealing can be used to find an approximation of a global minimum for a function with many variables.

Steps of the Algorithm

Step 1 Let $k = 0$ - current iteration, $T = T_k$ - initial temperature.

Step 2 Let $x_k \in X$ - some random point from our space

Step 3 Let decrease the temperature by following rule $T_{k+1} = \alpha T_k$ where $0 < \alpha < 1$ - some constant that often is closer to 1

Step 4 Let $x_{k+1} = g(x_k)$ - the next point which was obtained from previous one by some random rule. It is usually assumed that this rule works so that each subsequent approximation should not differ very much.

Step 5 Calculate $\Delta E = E(x_{k+1}) - E(x_k)$, where $E(x)$ - the function that determines the energy of the system at this point. It is supposed that energy has the minimum in desired value x^*.

Step 6 If $\Delta E < 0$ then the approximation found is better than it was. So accept x_{k+1} as new started point at the next step and go to the step **Step 3**

Step 7 If $\Delta E >= 0$, then we accept x_{k+1} with the probability of $P(\Delta E) = \exp^{-\Delta E / T_k}$. If we don't accept x_{k+1}, then we let $k = k + 1$. Go to the step **Step 3**

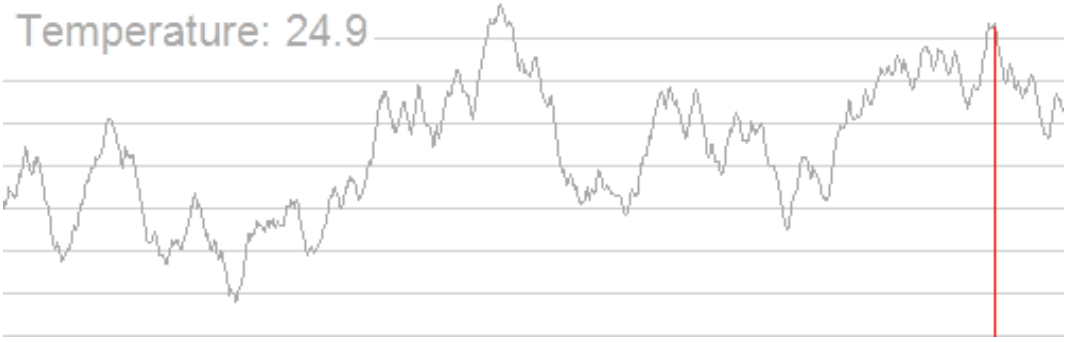
The algorithm can stop working according to various criteria, for example, achieving an optimal state or lowering the temperature below a predetermined level T_{min}.

Convergence

As it mentioned in Simulated annealing: a proof of convergence the algorithm converges almost surely to a global maximum.

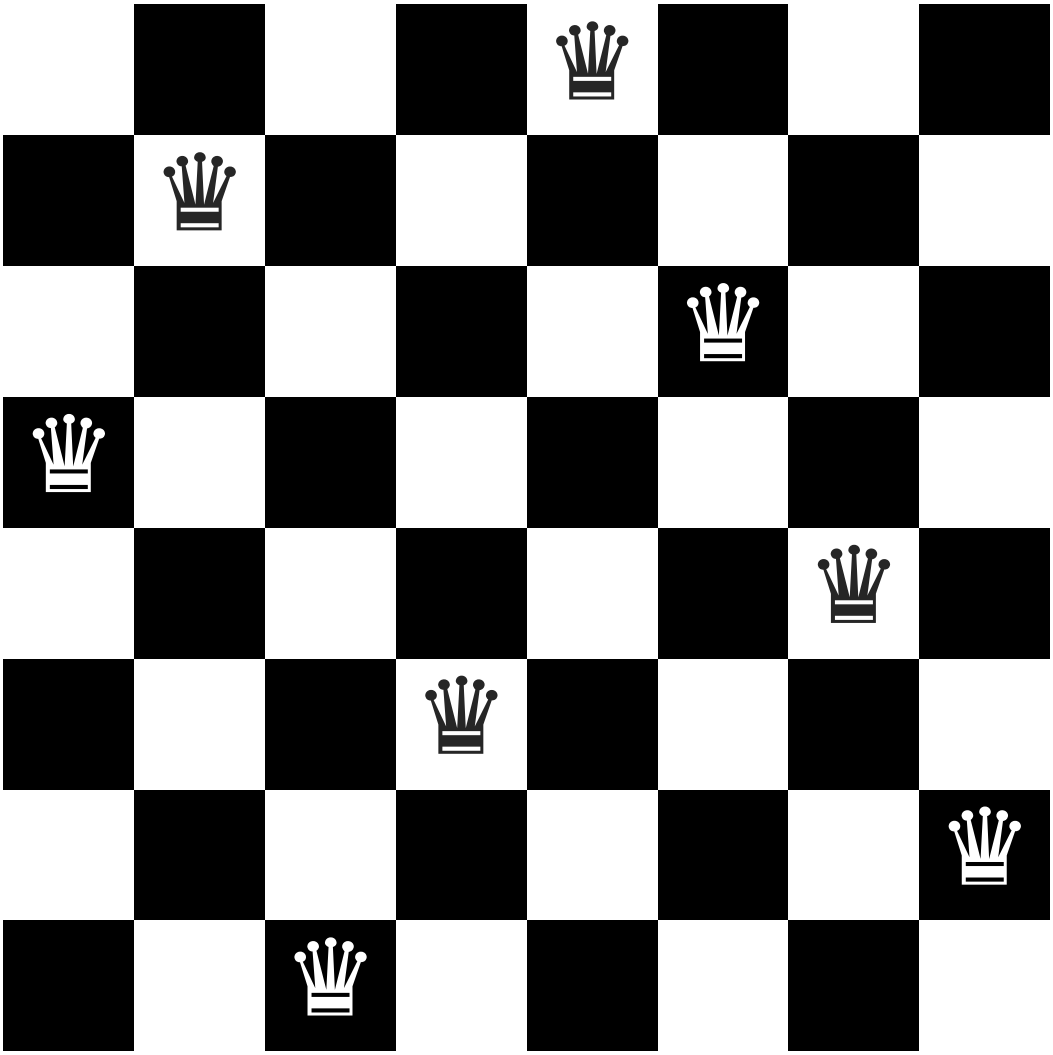
Illustration

A gif from Wikipedia:



Example

In our example we solve the N queens puzzle - the problem of placing N chess queens on an N×N chessboard so that no two queens threaten each other.



The Problem

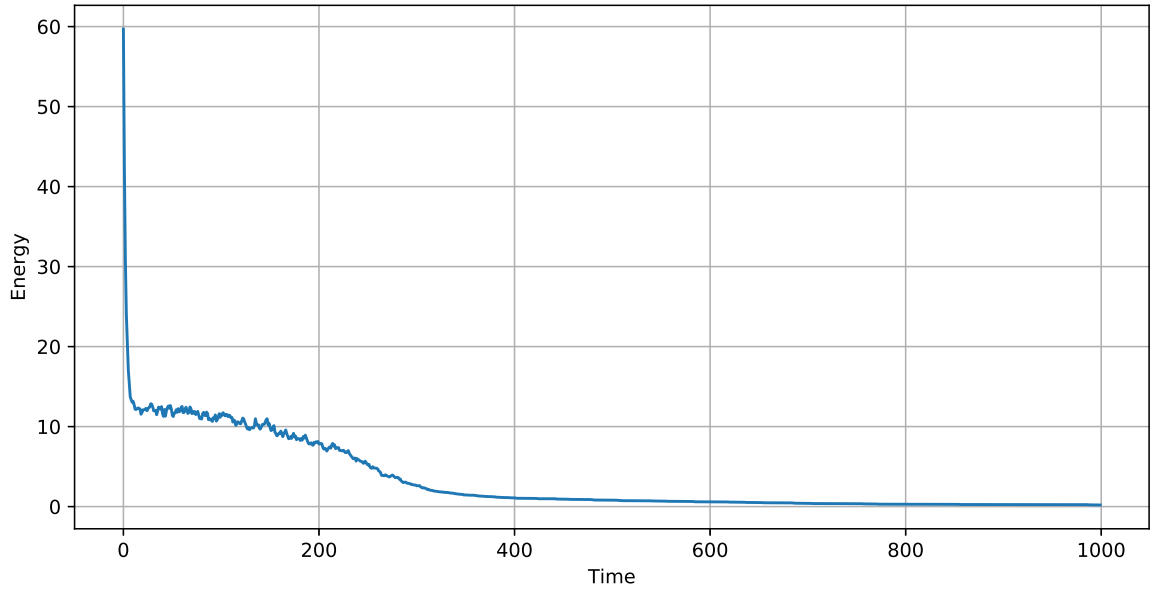
Let $E(x)$ - the number of intersections, where x - the array of placement queens at the field (the number in array means the column, the index of the number means the row).

The problem is to find x^* where $E(x^*) = \min_{x \in X} E(x)$ - the global minimum, that is predefined and equals to 0 (no two queens threaten each other).

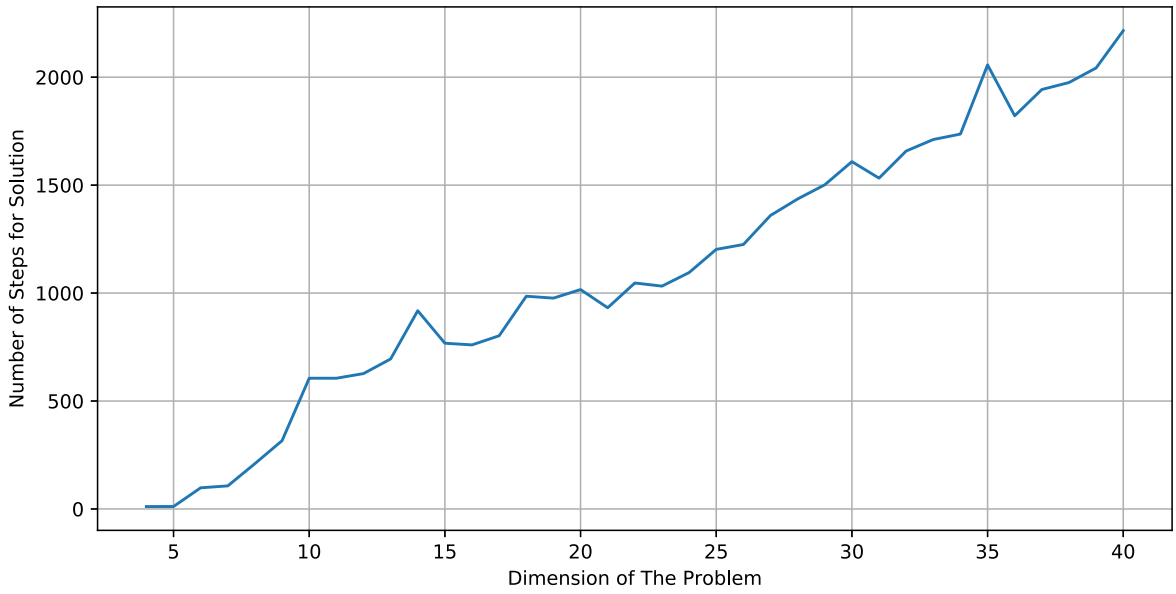
In this code $x_0 = [0, 1, 2, \ldots, N]$ that means all queens are placed at the board's diagonal . So at the beginning $E = N(N-1)$, because every queen intersects others.

Results

Results of applying this algorithm with $\alpha = 0.95$ to the N queens puzzle for $N = 10$ averaged by 100 runs are below:



Results of running the code for N from 4 to 40 and measuring the time it takes to find the solution averaged by 100 runs are below:



Open in Colab

Hengep-Mug

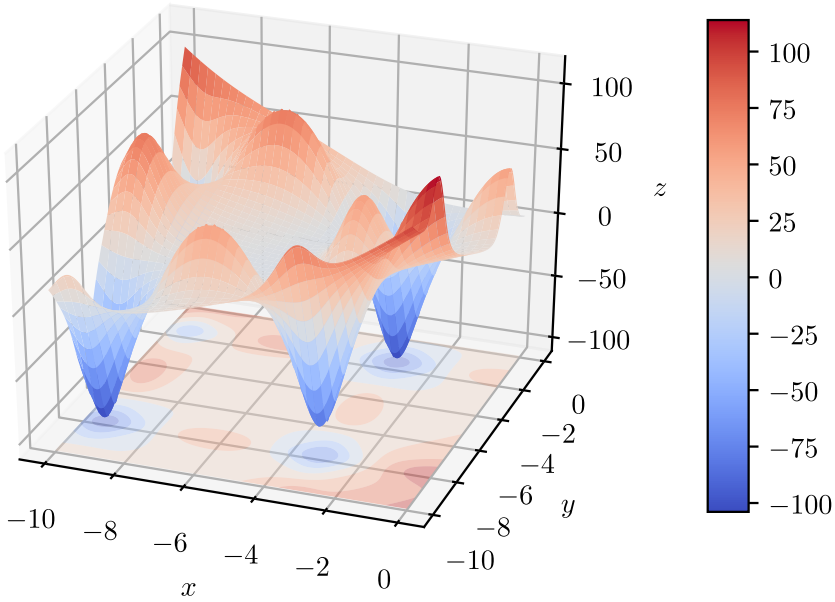
Nelder–Mead

Problem

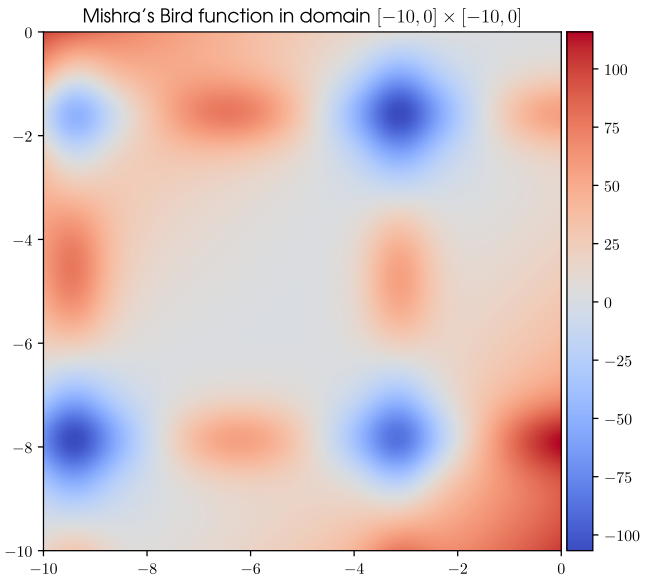
Sometimes the multidimensional function is so difficult to evaluate that even expressing the 1^{st} derivative for gradient-based methods of finding optimum becomes an impossible task.
In this case, we can only rely on the values of the function at each point. Or, in other words, on the 0 order oracle calls.

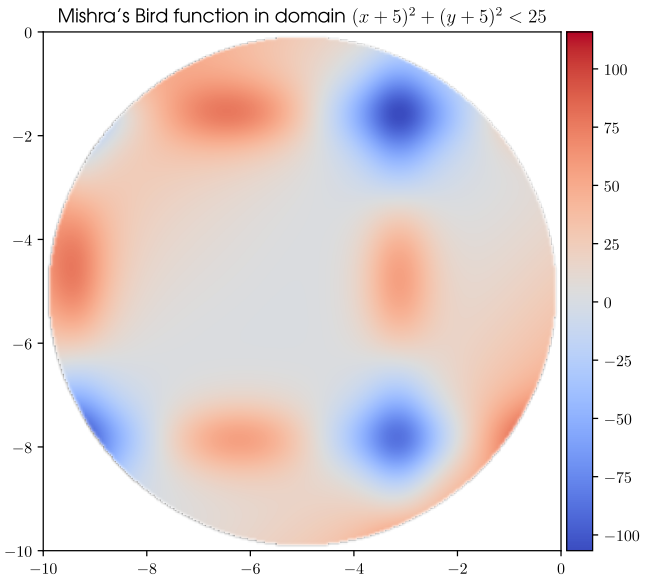
Let's take, for instance, Mishra's Bird function:

$$f(x, y) = \sin y \cdot e^{(1-\cos x)^2} + \cos x \cdot e^{(1-\sin y)^2} + (x - y)^2$$



This function is usually subjected to the domain $(x + 5)^2 + (y + 5)^2 < 25$, but for the sake of picture beauty we will mainly use domain $[-10; 0] \times [-10; 0]$.





Algorithm

Related definitions:

- **Simplex** -- polytope with the least possible number of vertices in n-dimensional space. (So, it's $(n+1)$-polytope.) In our $2D$ case it will be triangle.
- **Best point** x_1 -- vertex of the simplex, function value in which is the smallest among all vertices.
- **Worst point** x_{n+1} -- vertex of the simplex, function value in which is the largest among all vertices.
- **Other points** $x_2, \ldots, x_n$ -- vertices of the simplex, ordered in such way that $f(x_1) \leqslant f(x_2) \leqslant \ldots \leqslant f(x_n) \leqslant f(x_{n+1})$.
  This implies that $\{x_1, x_2, \ldots, x_n\}$ are best points in relation to x_{n+1} and $\{x_2, \ldots, x_n, x_{n+1}\}$ are worst points in relation to x_1.
- **Centroid** x_o -- center of mass in the polytope. In Nelder-Mead the centroid is calculated for the polytope, constituted by best vertices.
  In our $2D$ case it will be the center of the triangle side, which contains 2 best points $x_o = \dfrac{x_1 + x_2}{2}$.

Main idea

The algorithm maintains the set of test points in the form of simplex. For each point the function value is calculated and points are ordered accordingly.
Depending on those values, the simplex exchanges the worst point of the set for the new one, which is closer to the local minimum. In some sense, the simplex is crawling to the minimal value in the domain.

The simplex movements finish when its sides become too small (termination condition by sides) or its area becomes too small (termination condition by area). I prefer the second condition, because it takes into account cases when simplex becomes degenerate (three or more vertices on one axis).

Steps of the algorithm

1. Ordering

Order vertices according to values in them:

$$f(x_1) \leqslant f(x_2) \leqslant \ldots \leqslant f(x_n) \leqslant f(x_{n+1})$$

Check the termination condition. Possible exit with solution $x_{\min} = x_1$.

2. Centroid calculation

$$x_o = \frac{\sum_{k=1}^{n} x_k}{n}$$

3. Reflection

Calculate the reflected point x_r:

$$x_r = x_o + \alpha\,(x_o - x_{n+1})$$

where α -- reflection coefficient, $\alpha > 0$. (If $\alpha \leqslant 0$, reflected point x_r will not overlap the centroid)

The next step is figured out according to the value of $f(x_r)$ in dependency to values in points x_1 (best) and x_n (second worst):

- $f(x_r) < f(x_1)$: Go to step 4.
- $f(x_1) \leqslant f(x_r) < f(x_n)$: new simplex with $x_{n+1} \to x_r$. Go to step 1.
- $f(x_r) \geqslant f(x_n)$: Go to step 5.

4. Expansion

Calculate the expanded point x_e:

$$x_e = x_o + \gamma\,(x_r - x_o)$$

where γ -- expansion coefficient, $\gamma > 1$. (If $\gamma < 1$, expanded point x_e will be contracted towards centroid, if $\gamma = 1$: $x_e = x_r$)

The next step is figured out according to the ratio between $f(x_e)$ and $f(x_r)$:

- $f(x_e) < f(x_r)$: new simplex with $x_{n+1} \to x_e$. Go to step 1.
- $f(x_e) > f(x_r)$: new simplex with $x_{n+1} \to x_r$. Go to step 1.

5. Contraction

Calculate the contracted point x_c:

$$x_c = x_o + \beta\,(x_{n+1} - x_o)$$

where β -- contraction coefficient, $0 < \beta \leqslant 0.5$. (If $\beta > 0.5$, contraction is insufficient, if $\beta \leqslant 0$, contracted point x_c overlaps the centroid)

The next step is figured out according to the ratio between $f(x_c)$ and $f(x_{n+1})$:

- $f(x_c) < f(x_{n+1})$: new simplex with $x_{n+1} \to x_c$. Go to step 1.
- $f(x_c) \geqslant f(x_{n+1})$: Go to step 6.

6. Shrinkage

Replace all points of simplex x_i with new ones, except for the best point x_1:

$$x_i = x_1 + \sigma\,(x_i - x_1)$$

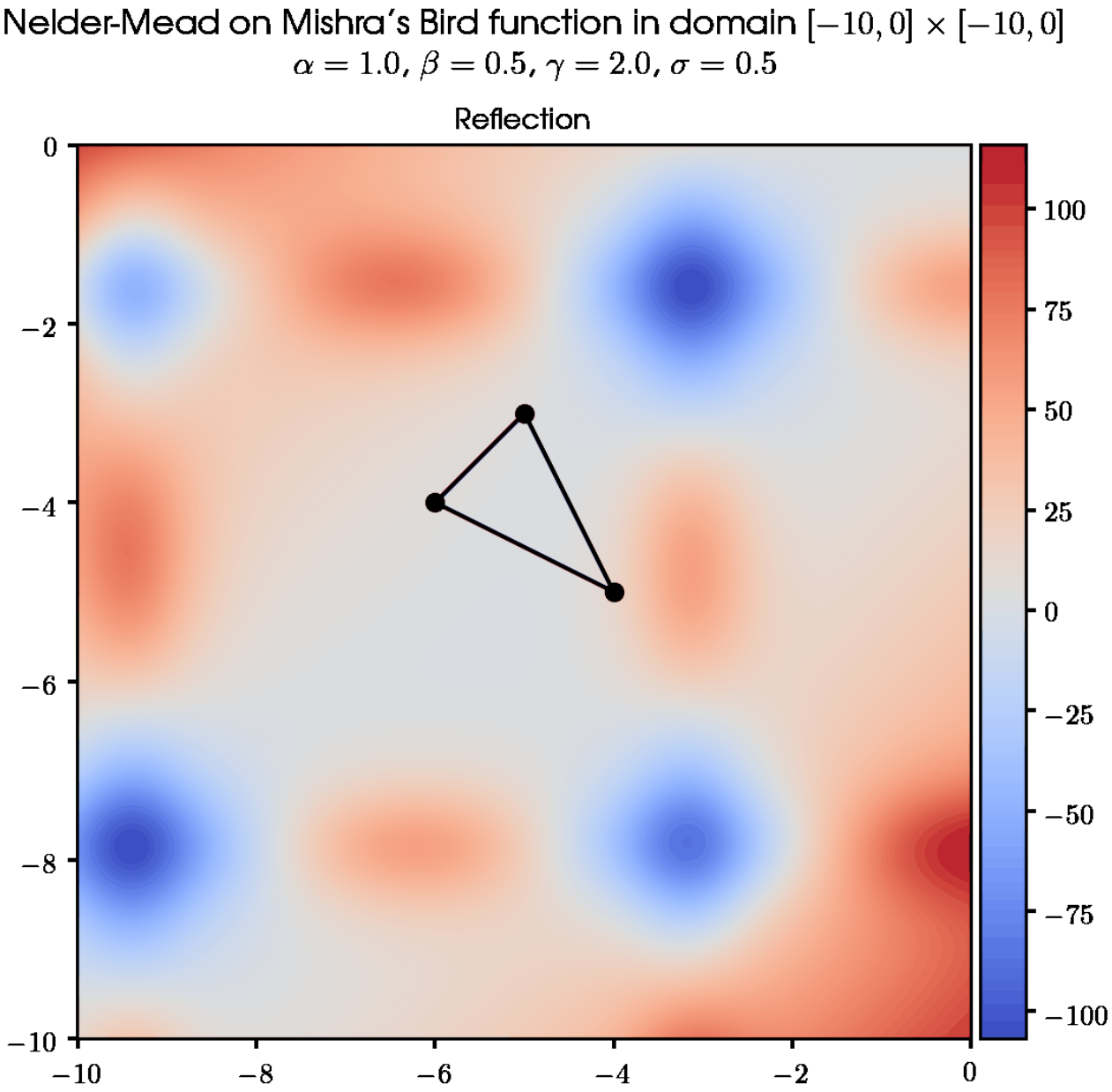
where σ -- shrinkage coefficient, $0 < \sigma < 1$. (If $\sigma \geqslant 1$, shrinked point x_i overlaps the best point x_1, if $\sigma \leqslant 0$, shrinked point x_i becomes extended)

Go to step 1.

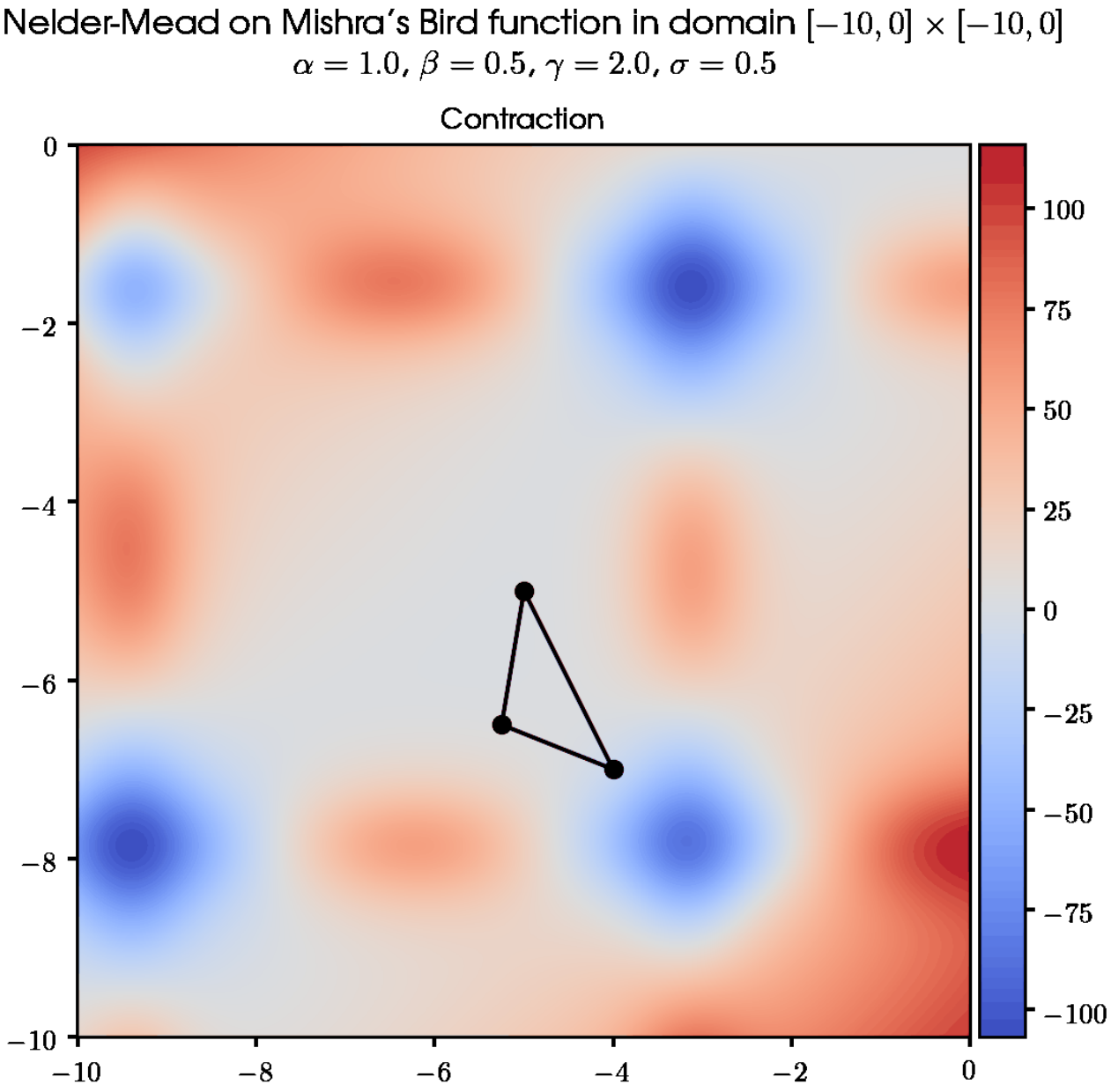
Examples

This algorithm, as any method in global optimization, is highly dependable on the initial coonditions. For instance, if we use different initial simplex or different set of parameters $\{\alpha, \beta, \gamma, \sigma\}$ the resulting optimal point will differ.

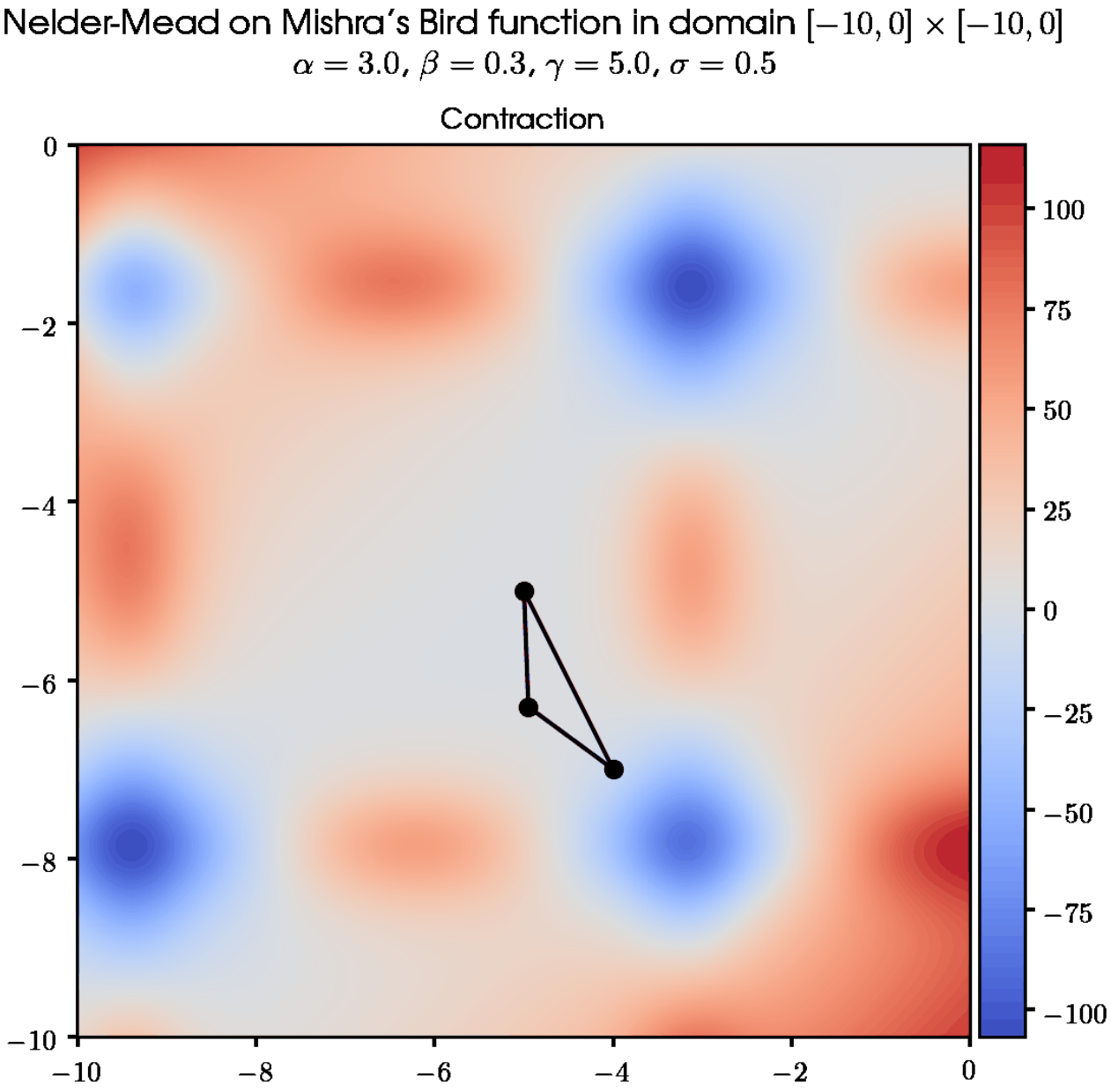
Some random initial simplex and default set of parameters



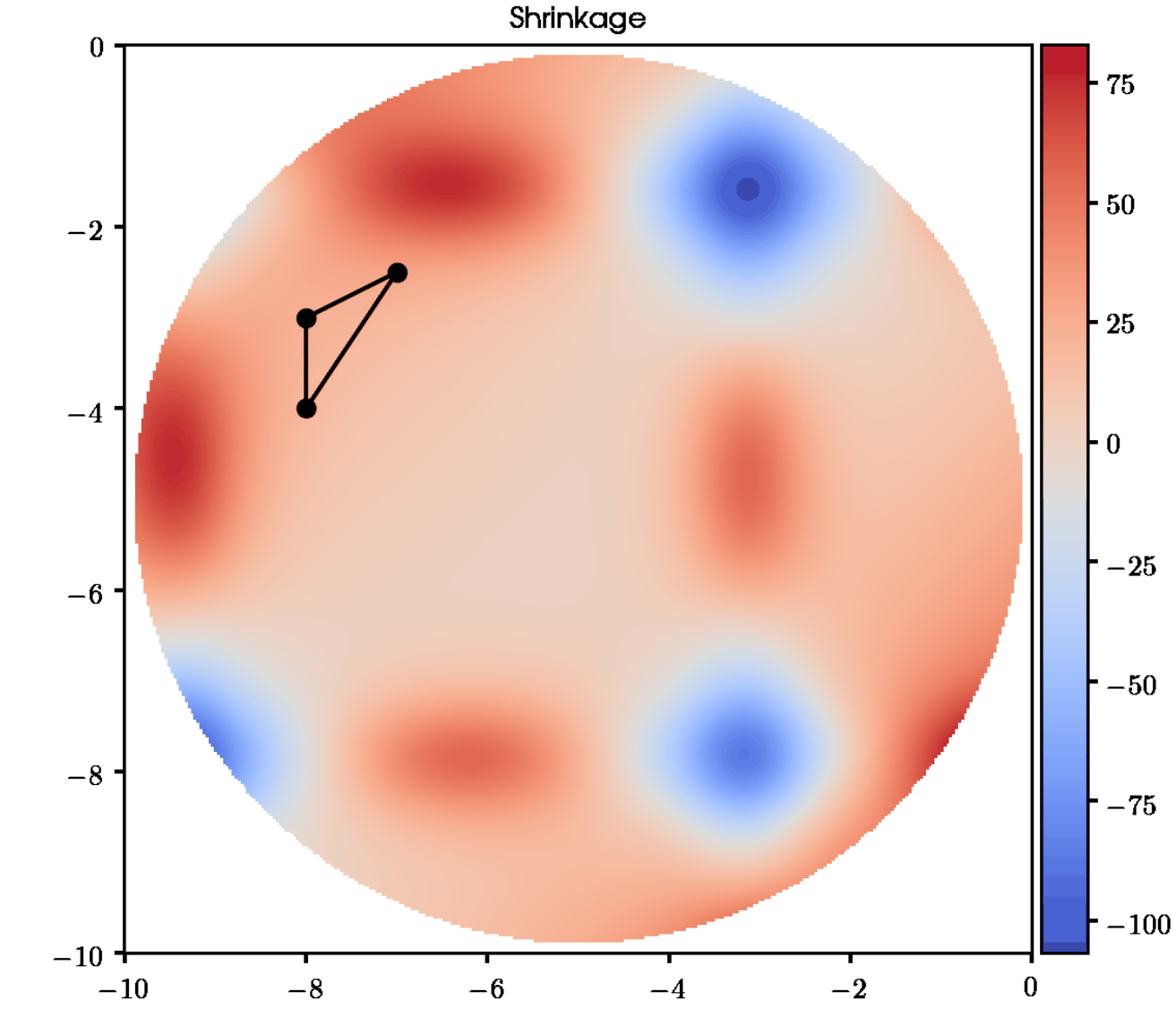
Different initial simplex and same set of parameters



Same initial simplex and different set of parameters

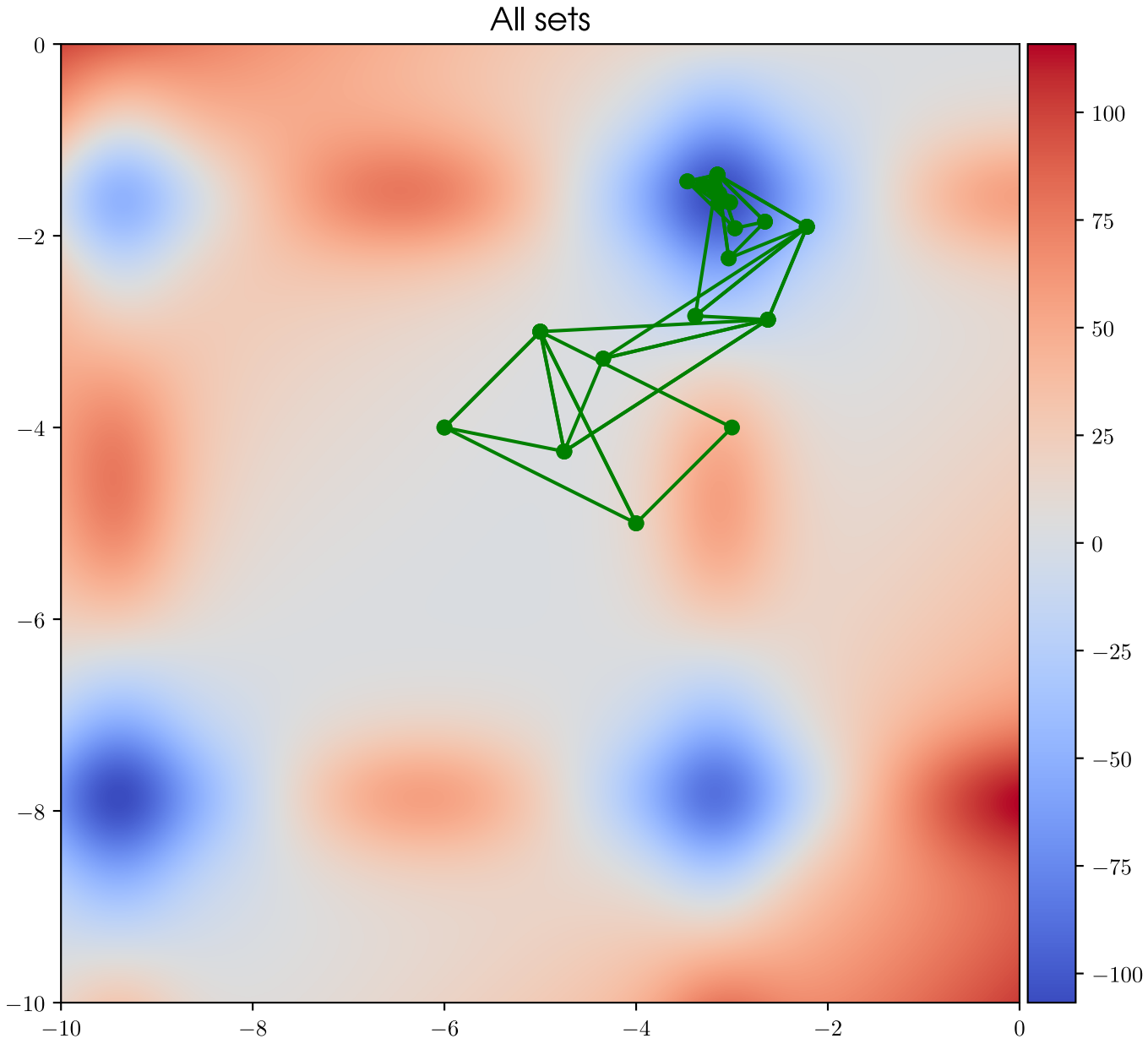


Round domain

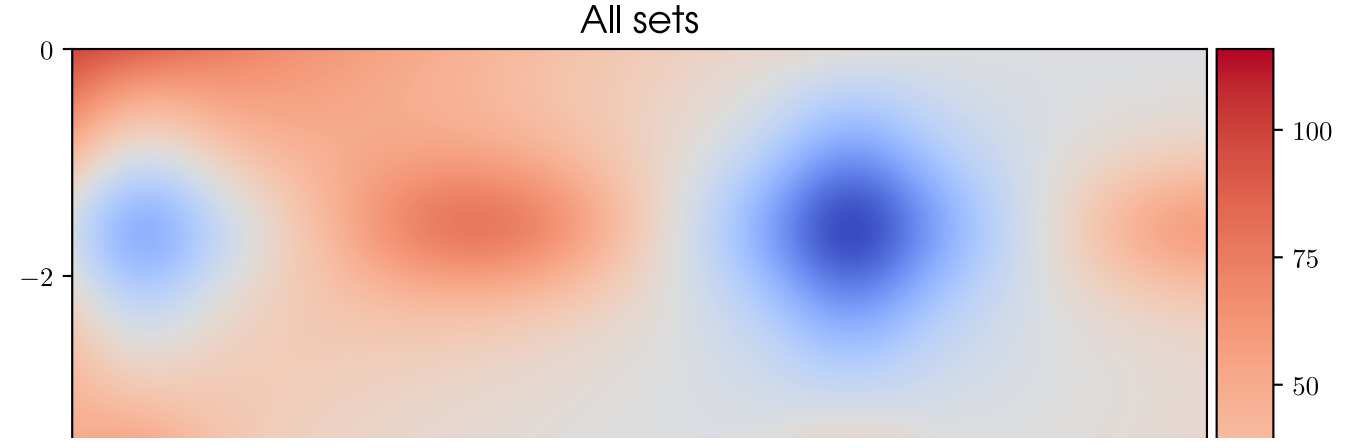


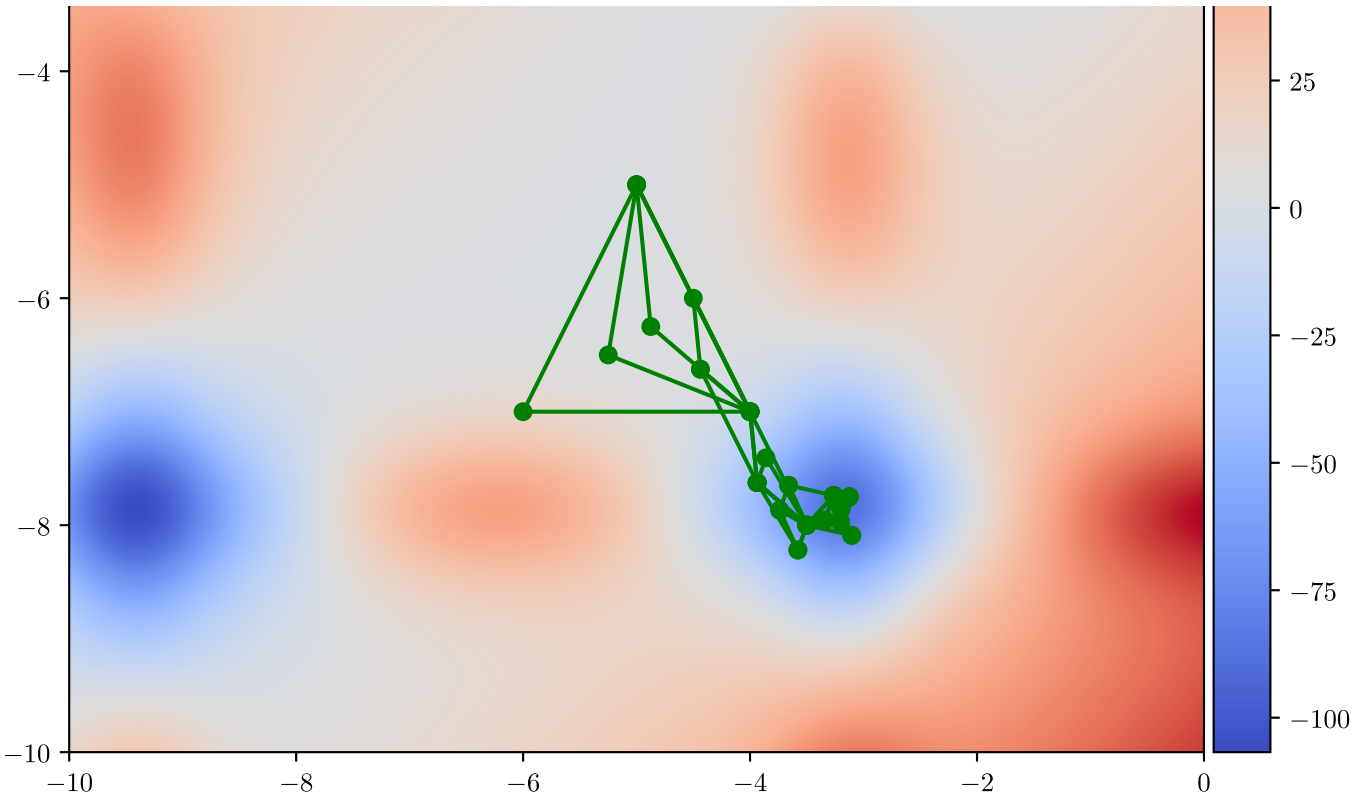
Examples with all sets of simplexes

Nelder-Mead on Mishra's Bird function in domain $[-10, 0] \times [-10, 0]$
$\alpha = 1.0$, $\beta = 0.5$, $\gamma = 2.0$, $\sigma = 0.5$

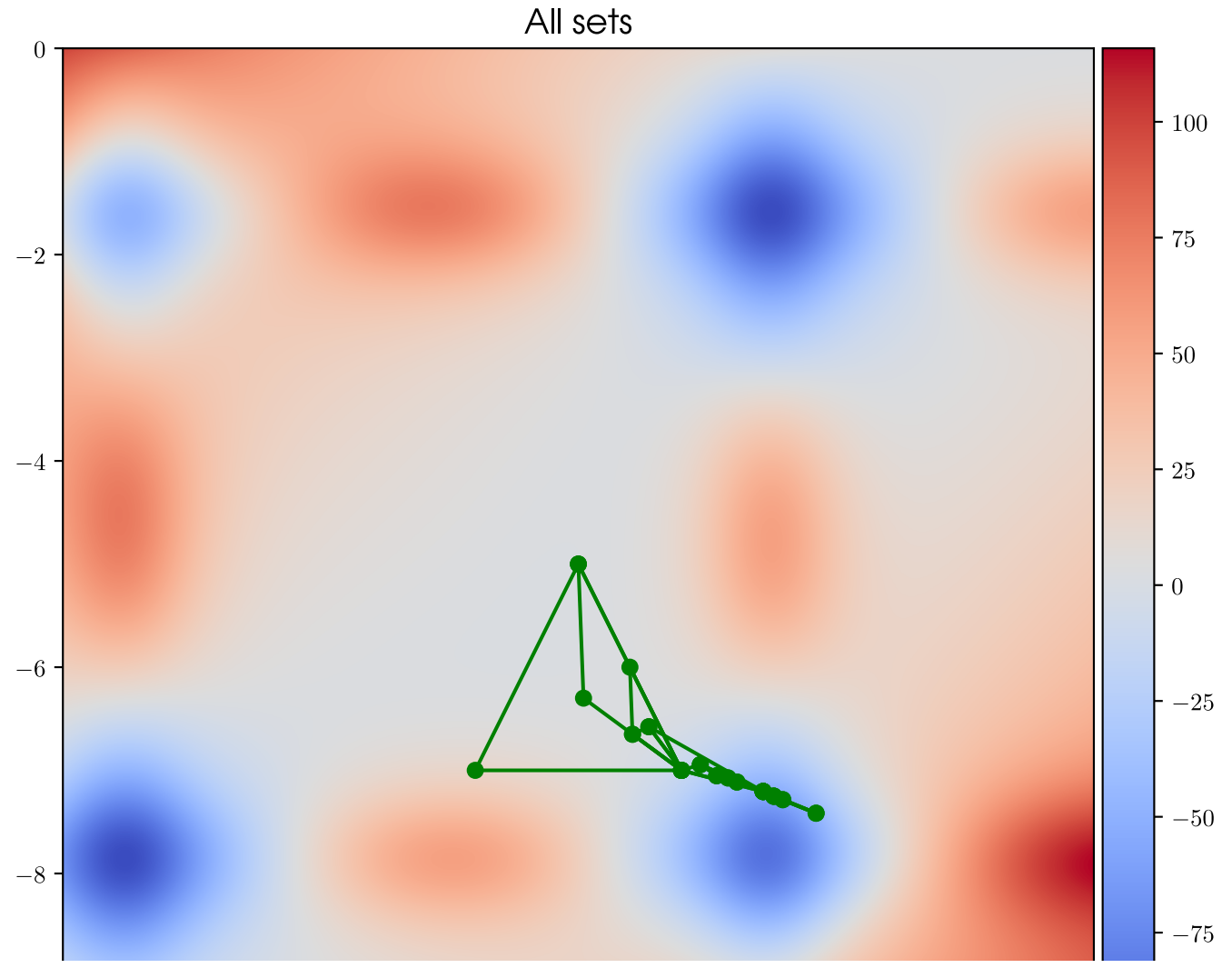


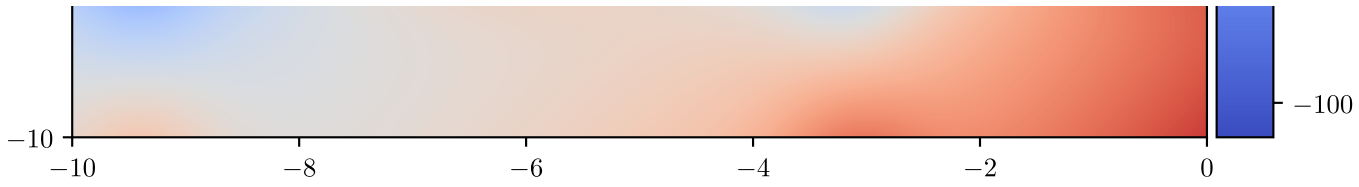
Nelder-Mead on Mishra's Bird function in domain $[-10, 0] \times [-10, 0]$
$\alpha = 1.0$, $\beta = 0.5$, $\gamma = 2.0$, $\sigma = 0.5$



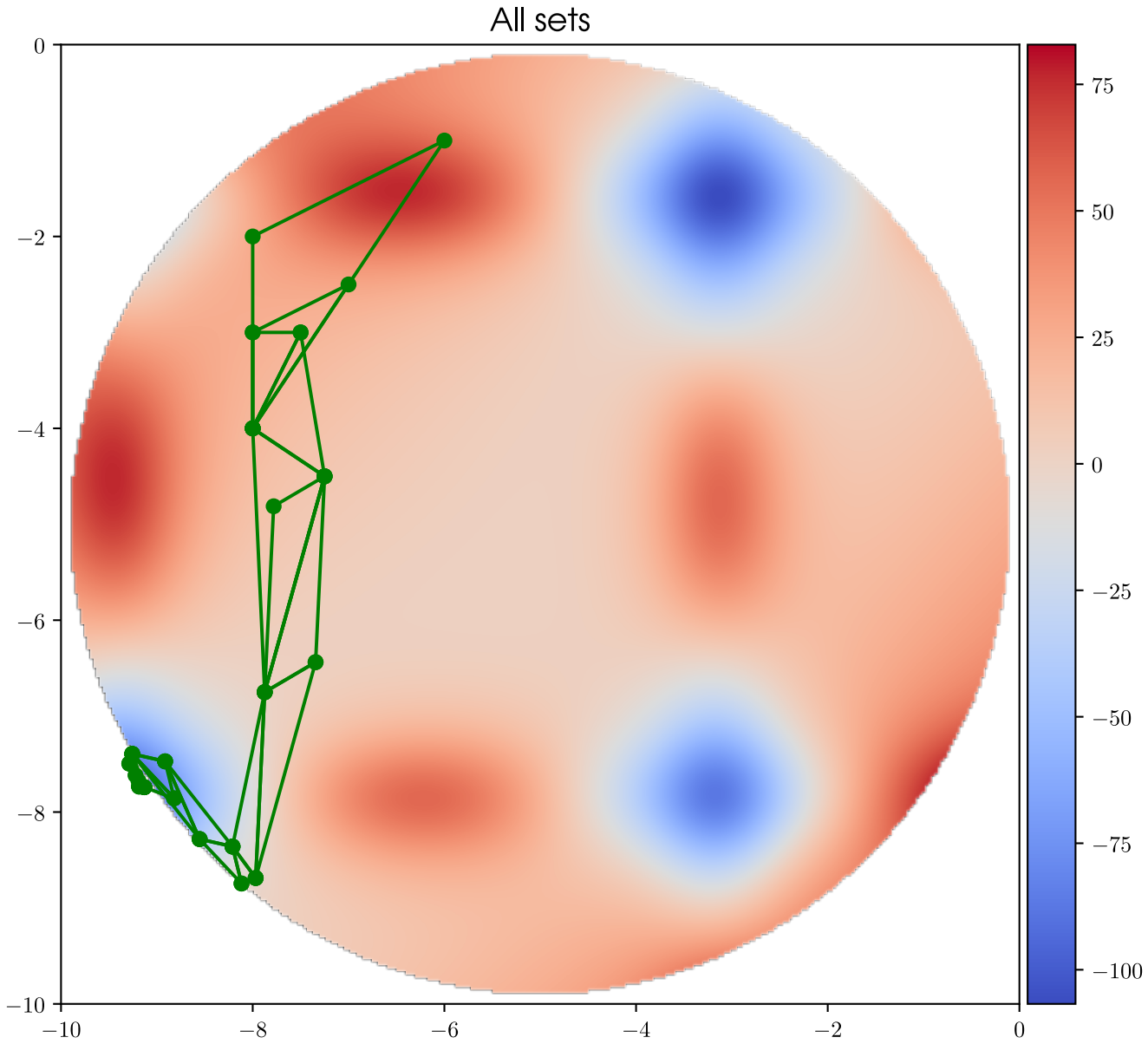


Nelder-Mead on Mishra's Bird function in domain $[-10, 0] \times [-10, 0]$
$\alpha = 3.0$, $\beta = 0.3$, $\gamma = 5.0$, $\sigma = 0.5$





Nelder-Mead on Mishra's Bird function in domain $(x+5)^2 + (y+5)^2 < 25$
$\alpha = 1.0, \beta = 0.5, \gamma = 2.0, \sigma = 0.5$



Code

Open in Colab

Example 1

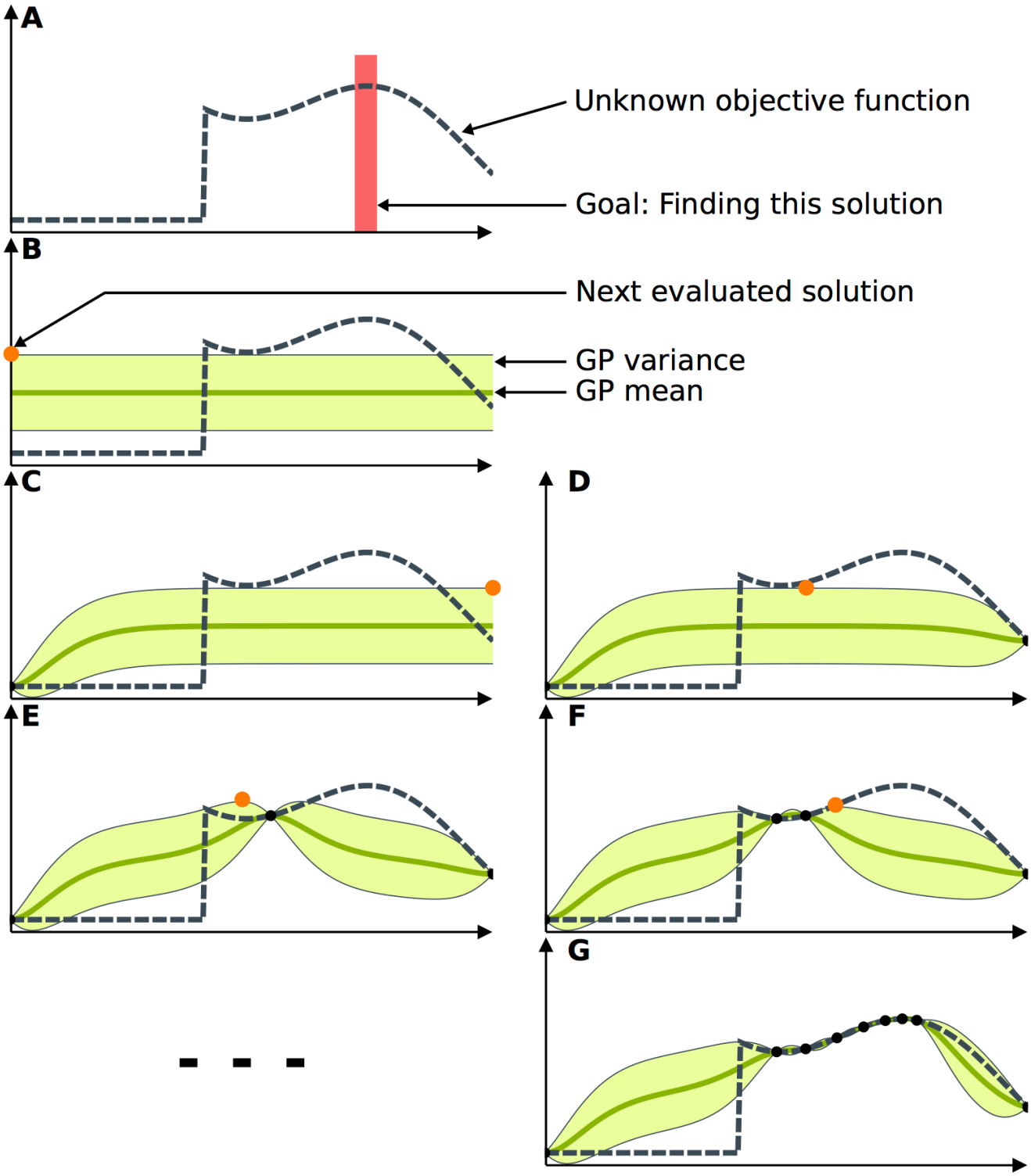
Implement Rastrigin function $f : \mathbb{R}^d \to \mathbb{R}$ for d = 10. link

$$f(\mathbf{x}) = 10d + \sum_{i=1}^{d} \left[x_i^2 - 10\cos(2\pi x_i) \right]$$

- Consider global optimization from here.
- Try to plot 4 graphs for different d from {10, 100, 1000, 10000}. On each graph you are to plot f from N_{fev} for 5 methods: `basinhopping`, `brute`, `differential_evolution`, `shgo`, `dual_annealing` from scipy, where N_{fev} - the number of function evaluations. This information is usually available from `specific_optimizer.nfev`. If you will need bounds for the optimizer, use $x_i \in [-5, 5]$.

Example 2

Machine learning models often have hyperparameters. To choose optimal one between them one can use GridSearch or RandomSearch. But these algorithms computationally uneffective and don't use any sort of information about type of optimized function. To overcome this problem one can use bayesian optimization. Using this method we optimize our model by sequentially chosing points based on prior information about function.



.

In this task you will use optuna package for hyperparameter optimization RandomForestClassifier. Your task is to find best Random Forest model varying at least 3 hyperparameters on iris dataset. Examples can be find here or here

```bash
!pip install optuna
```

```python
import sklearn.datasets
import sklearn.ensemble
import sklearn.model_selection
import sklearn.svm

import optuna

iris = sklearn.datasets.load_iris()
x, y = iris.data, iris.target
```

Example 3

Try to perform hyperparameter optimization in context of any metric for imbalanced classification problem with optuna and keras.