Genetic Algorithm Project

Python in engineering calculations

Genetic Algorithm – history, description

In computer science **genetic algorithm (GA)** is an algorithm used to optimization. It belongs to the class of evolutionary algorithms (EA). They are in general inspired by the process of natural selection – through implementation of biologically inspired operations such as:

- mutation
- crossover
- selection

Side Note

In this project the n-best approach with variable step value and elite picking has been used, however it is compulsory to mention that there are many other possible ways to implement genetic algorithm. Adding mutation or different type of crossover such as tournament is likely to improve results generated by the algorithm but is not required to present the basic algorithm usage. In this document only the chosen strategy will be shortly described. To broaden your knowledge regarding GA's please refer to literature and sources at the end of this document.

General Idea

In genetic algorithm there are few objects (often implemented as classes in object oriented programming):

- *Population* – a set of candidate solutions to given problem
- *Candidate* – (sometimes phenotype) solution to the problem consisting of properties – chromosomes or genotype which can be mutated and altered
- *Gens* – (or genotypes) are properties , e.g. x and y for 2D function (variables in general)

The evolution starts from a random population of randomly generated candidates. Each iteration of population is called *generation.* Then the fitness of every individual is calculated – **Fitness** is most often the value of the objective function.

With n – best approach the candidates array is sorted and n best individuals are picked. They are used to create new population. Elite picking is a way of saving some number of best results – they are overwritten to the next population. It is a safety feature to secure the best result/s in the next population.

Created algorithm is used to optimize a function in a given range, For example:

$$x_1 \in [-5; 5], x_2 \in [-5; 5], x_3 \in [10; 100], \ldots$$

When function consists of 2 variables it is possible to visualize the operation of algorithm on 3D graph, however the algorithm in its original form is capable of scaling to more dimensions without any issues.

In the created code the variable step is calculated using function:

$$Step[i] = Initial\ Step \times \left[1 + e^{\frac{i - \frac{n}{P_1}}{P_2}}\right]^{-1}$$

where,

n – number of generations that are going to be computed

The reason that such approach has been used is to divide the algorithm life cycle into two parts: *Exploration* and Exploitation. The use of variable step enables to control the proportion:

$$\frac{Exploration}{Exploitation}$$

The **exploration** is a process of creating new candidates in a broad range, not allowing to get stuck in local optimum. The step during exploration should be in range 10% - 40% (of variable range).

The **exploitation** is a process of creating new candidates in a small range, allowing to get accurate and complete knowledge of current optimum. The step during exploitation should be in range 5% - 0.001%.

Below on ***Figure 1*** the visualization of step function for parameters: *Initial Step = 3.5, n = 100, P1 = 2.5, P2 = 6* is presented*.*

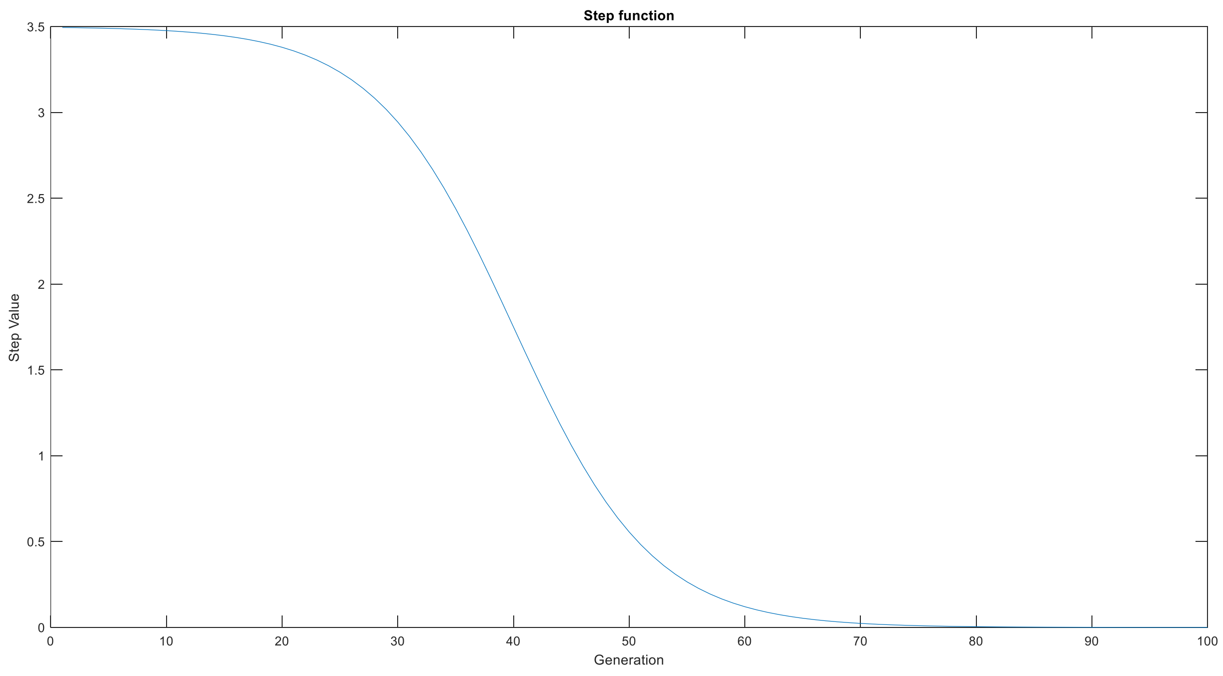


Figure 1 - Step value visualization

So for each candidate chosen for crossover (n – best succession) the step multiplied be the random number in range [-1;1] is added. This creates the new population and the process repeats until the given number of generations has been reached.

The Code

The project is divided in two scripts:

- GA.py – The script consisting of genetic algorithm
- Display.py – The script that generates a GUI window allowing to chose, adjust parameters and visualize algorithm on 4 functions used to rate the algorithm

Algorithm File

Arguments:

```
def myParser():
    parser = argparse.ArgumentParser()
    parser.add_argument("Population_size", type=int,
                        help="Number of candidates in one population")
    parser.add_argument(
        "Iteration", type=int, help="Number of generations that are going to
be computed")
    parser.add_argument(
        "Function", type=int, help=f"Type of function to optimize:\n\t0 -
x\u00b2 + y\u00b2\n\t1 - Ackley function\n\t2 - Himmelblau's function\n\t3 -
Hölder table function")
    parser.add_argument(
        "Initial_step", type=float, help="Initial step for new candidates
creating")
    parser.add_argument(
        "Parameter_1", type=float, help="Parameter 1 for the step proportion -
exploration/exploatation \n\tStep[iter] = Initial_Step * (1/(1+e^(iter-
(MaxStep/P1))/P2))")
    parser.add_argument(
        "Parameter_2", type=float, help="Parameter 2 for the step proportion -
exploration/exploatation \n\tStep[iter] = Initial_Step * (1/(1+e^(iter-
(MaxStep/P1))/P2))")
    parser.add_argument(
        "Best_succession", type=int, help="Number of best candidates that are
going to take part in new population generation")

    args = parser.parse_args()
    return args
```

Picking a function for optimizing (More on this in *Appendix 1)*:

```
def Function(x, y, PICK_):
    if(PICK_ == 0):
        return (x**2 + y**2)
    elif(PICK_ == 1):
        return -20.0 * np.exp(-0.2 * np.sqrt(0.5 * (x**2 + y**2))) -
np.exp(0.5 * (np.cos(2 * np.pi * x) + np.cos(2 * np.pi * y))) + np.e + 20
    elif(PICK_ == 2):
        return (x**2 + y - 11)**2 + (x + y**2 - 7)**2
    elif(PICK_ == 3):
        return -np.absolute(np.sin(x) * np.cos(y) * np.exp(np.absolute(1 -
(np.sqrt(x**2 + y**2)/np.pi))))
```

The main part of algorithm

First random population:

```
 # Random first population
    for k in range(pop):
        P[k][0] = InitialRangeX[0] + np.random.rand() * \
            (InitialRangeX[1]-InitialRangeX[0])
        P[k][1] = InitialRangeY[0] + np.random.rand() * \
            (InitialRangeY[1]-InitialRangeY[0])
```

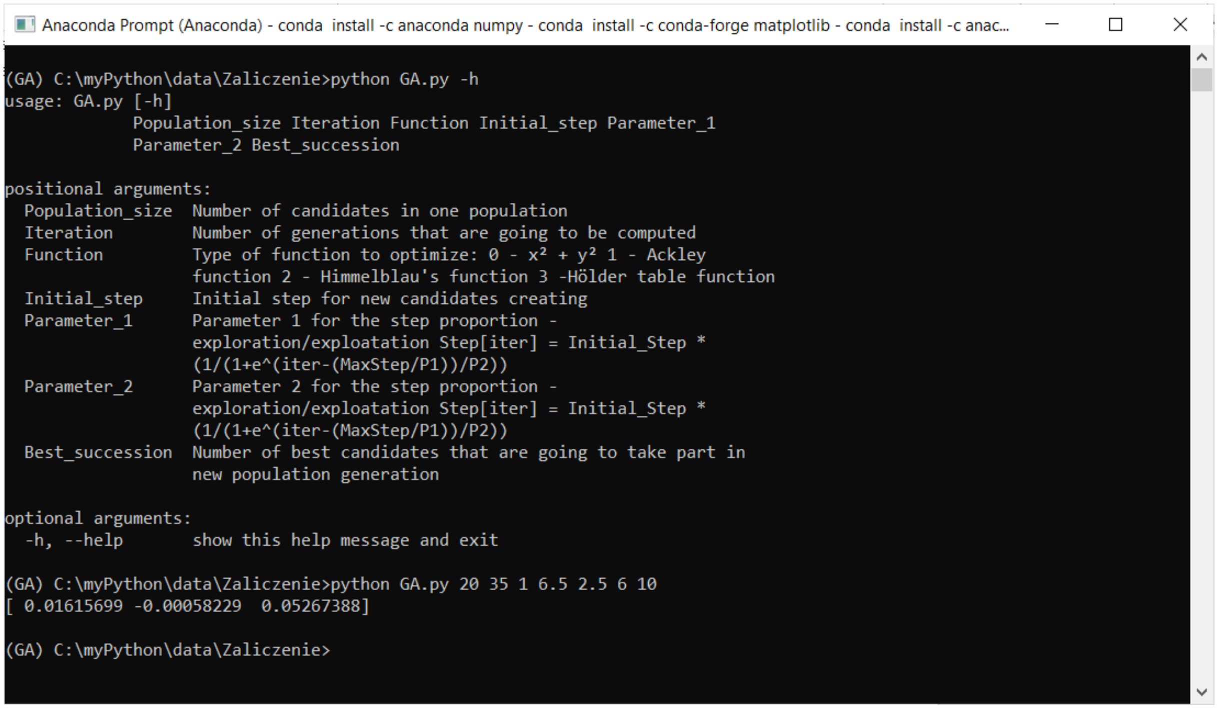
Main loop:

```
 while(EndingCondition == 0):
        Step[iter] = Initial_Step * (1/(1+np.e**(iter-(MaxStep/P1))/P2))

        for k in range(pop):
            P[k][2] = Function(P[k][0], P[k][1], PICK)
            pom.append(P[k][0])
            pom.append(P[k][1])
            pom.append(P[k][2])
            list_of_rows.append(pom.copy())
            pom.clear()
        P_sorted = P[P[:, 2].argsort()]
        P[0] = P_sorted[0]

        for k in range(1, pop):
            index_1 = np.random.randint(0, n)
            index_2 = np.random.randint(0, n)
            P[k][0] = P_sorted[index_1, 0] + \
                Step[iter] * np.random.uniform(-1, 1)
            P[k][1] = P_sorted[index_2, 1] + \
                Step[iter] * np.random.uniform(-1, 1)
            if(P[k][0] < InitialRangeX[0]):
                P[k][0] = InitialRangeX[0]
            else:
                if(P[k][0] > InitialRangeX[1]):
                    P[k][0] = InitialRangeX[1]
            if(P[k][1] < InitialRangeY[0]):
                P[k][1] = InitialRangeY[0]
            else:
                if(P[k][1] > InitialRangeY[1]):
                    P[k][1] = InitialRangeY[1]
        Best = P_sorted[0, :]
        iter += 1
        if(iter == MaxStep):
            EndingCondition = 1
```

The fitness of each candidate is being computed. Then the array of candidates is being sorted by fitness. Next staring form index 1 (the best solutions is not being changed) random candidate from n – best is being picked and modified to create new candidate and in result a new population. After checking if new candidates are inside optimizing range, until the maximum number of generations has been reached the loop continues.

At the end results are being written into csv file.

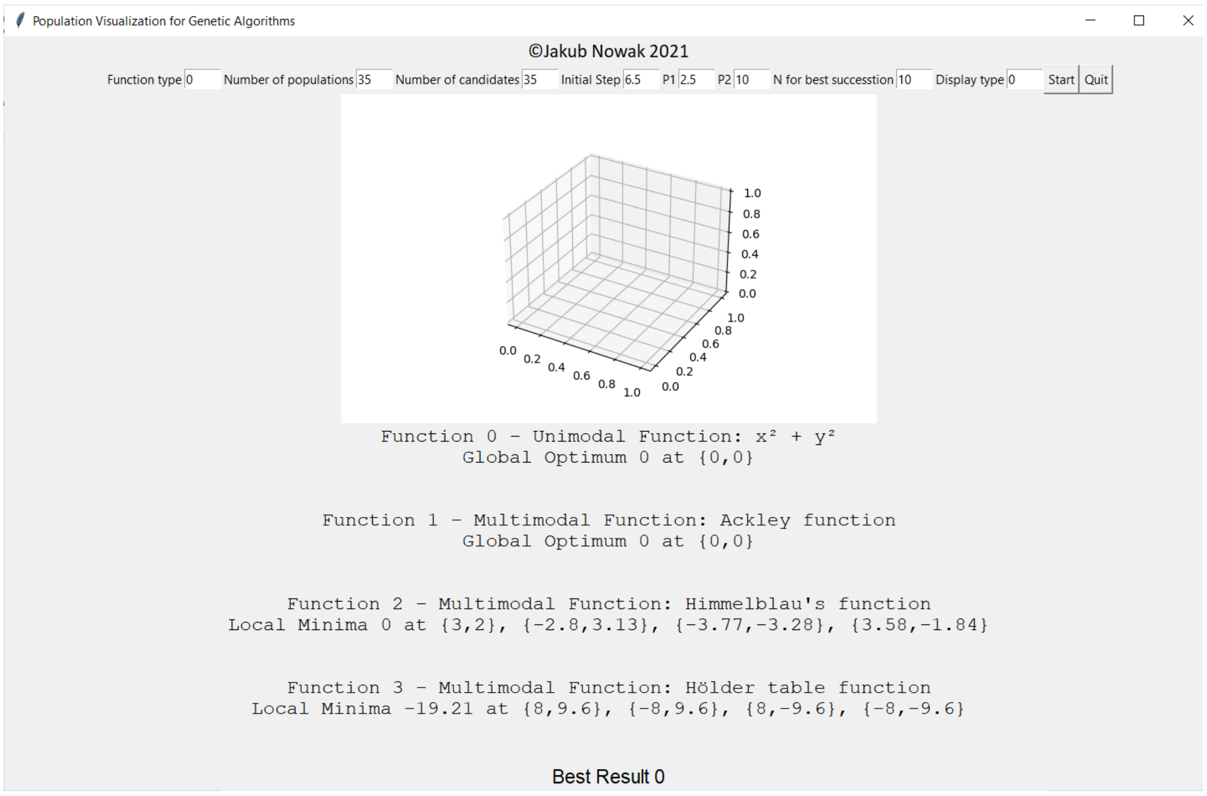
Below is a screenshot form Anaconda Prompt:


Anaconda Prompt (Anaconda) - conda install -c anaconda numpy - conda install -c conda-forge matplotlib - conda install -c anac...

```
(GA) C:\myPython\data\Zaliczenie>python GA.py -h
usage: GA.py [-h]
             Population_size Iteration Function Initial_step Parameter_1
             Parameter_2 Best_succession

positional arguments:
  Population_size  Number of candidates in one population
  Iteration        Number of generations that are going to be computed
  Function         Type of function to optimize: 0 - x² + y² 1 - Ackley
                   function 2 - Himmelblau's function 3 -Hölder table function
  Initial_step     Initial step for new candidates creating
  Parameter_1      Parameter 1 for the step proportion -
                   exploration/exploatation Step[iter] = Initial_Step *
                   (1/(1+e^(iter-(MaxStep/P1))/P2))
  Parameter_2      Parameter 2 for the step proportion -
                   exploration/exploatation Step[iter] = Initial_Step *
                   (1/(1+e^(iter-(MaxStep/P1))/P2))
  Best_succession  Number of best candidates that are going to take part in
                   new population generation

optional arguments:
  -h, --help       show this help message and exit

(GA) C:\myPython\data\Zaliczenie>python GA.py 20 35 1 6.5 2.5 6 10
[ 0.01615699 -0.00058229  0.05267388]

(GA) C:\myPython\data\Zaliczenie>
```

Script used to display the results

The code is included with the report and not presented here since it is not relevant to the topic of optimizing a function. It consist of tkinter and matplotlib objects used to create animation and the window. The result of running the script is below:



©Jakub Nowak 2021

Function type 0 Number of populations 35 Number of candidates 35 Initial Step 6.5 P1 2.5 P2 10 N for best successtion 10 Display type 0

```
Function 0 - Unimodal Function: x² + y²
Global Optimum 0 at {0,0}

Function 1 - Multimodal Function: Ackley function
Global Optimum 0 at {0,0}

Function 2 - Multimodal Function: Himmelblau's function
Local Minima 0 at {3,2}, {-2.8,3.13}, {-3.77,-3.28}, {3.58,-1.84}

Function 3 - Multimodal Function: Hölder table function
Local Minima -19.21 at {8,9.6}, {-8,9.6}, {8,-9.6}, {-8,-9.6}
```

Best Result 0

At the top there are parameters used to choose the function to optimize, algorithm variables and a textbox used to change the display type. Display type 0 results in leaving on plot all solutions computed during algorithm operation. Type 1 displays only the current population.

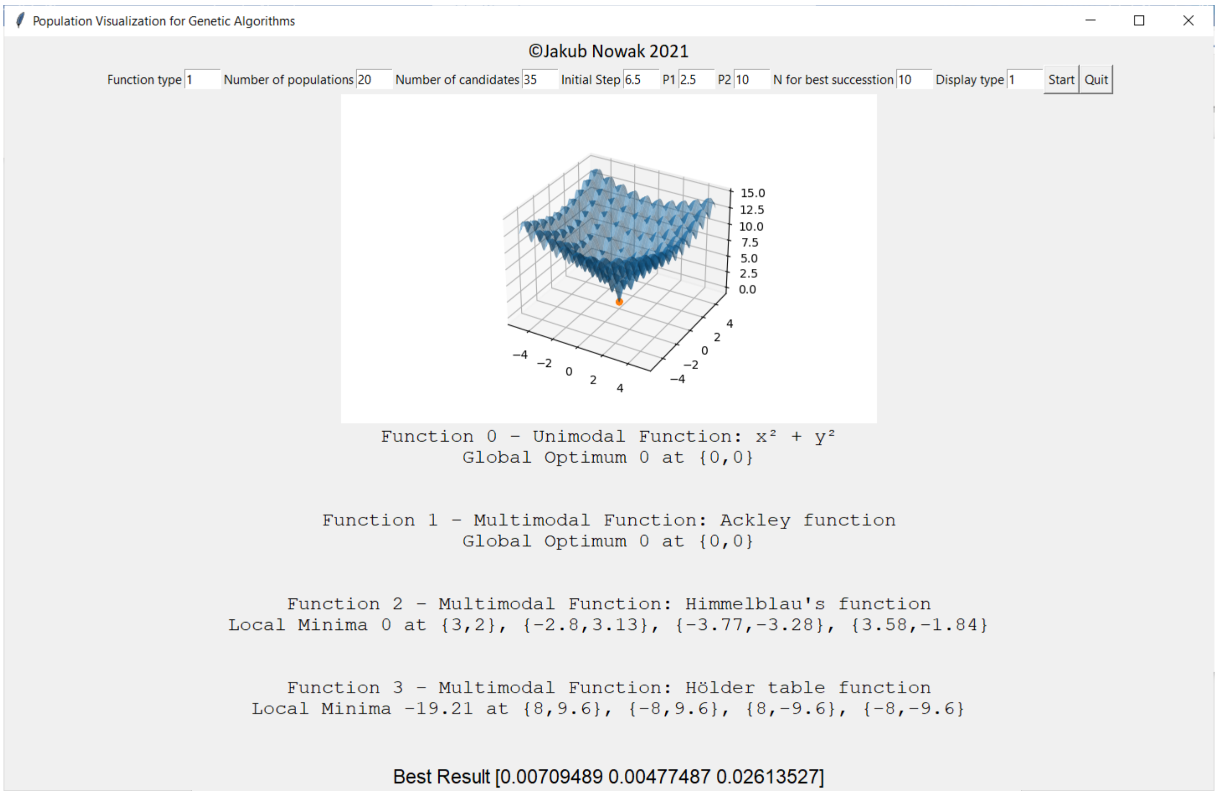
Below the plot is an information used to determine if the generated result is a good approximation of the mathematically computed optimum. Each function type has it optimum and the variables values at which it exits.

At the bottom highlighted is the **Best Result** that was achieved during algorithm operation. By compering it to the information above it is possible to determine if the parameters are picked correctly.

In normal engineering calculation the algorithm is tested on 2D functions an then scaled up to solved more complex problems by running it a lot of times with different parameters to determine the optimum solution for problem.

At the end are also presented functions that scale up to more dimensions and enable to rate the algorithm on its default target.

Example of solution



©Jakub Nowak 2021

Function type 1 Number of populations 20 Number of candidates 35 Initial Step 6.5 P1 2.5 P2 10 N for best successtion 10 Display type 1

```
Function 0 – Unimodal Function: x² + y²
Global Optimum 0 at {0,0}


Function 1 – Multimodal Function: Ackley function
Global Optimum 0 at {0,0}


Function 2 – Multimodal Function: Himmelblau's function
Local Minima 0 at {3,2}, {-2.8,3.13}, {-3.77,-3.28}, {3.58,-1.84}


Function 3 – Multimodal Function: Hölder table function
Local Minima -19.21 at {8,9.6}, {-8,9.6}, {8,-9.6}, {-8,-9.6}
```

Best Result [0.00709489 0.00477487 0.02613527]

More examples will be presented during live presentation.

Real life usage of genetic algorithm

The author thought it is important to present at the end the historical application of GA to show the importance of it in the engineering environment. On March 22, 2006 the ST5 mission was successfully launched into space using the evolved antenna ST5-33-142-7 as one of its antennas. This evolved antenna is the first computer-evolved antenna to be deployed for any application and is the first computer-evolved hardware in space

Source: https://ti.arc.nasa.gov/m/pub-archive/1244h/1244%20%28Hornby%29.pdf

To meet the initial design requirements it was decided to constrain our evolutionary design to a monopole wire antenna with four identical arms, with each arm rotated 90◦ from its neighbors. To produce this type of antenna, the EA evolves a description of a single arm and evaluates these individuals by building a complete antenna using four copies of the evolved arm.

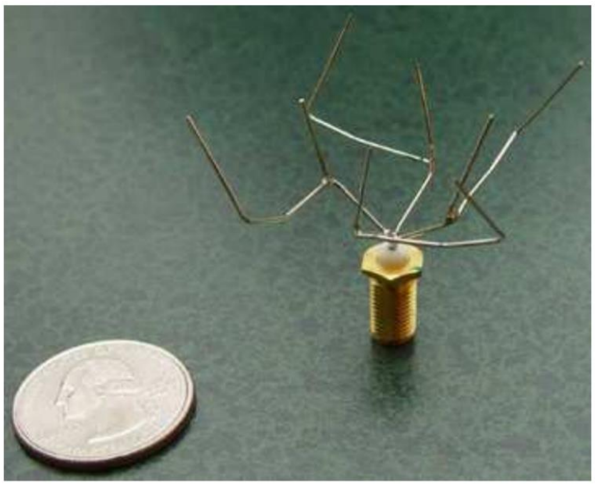
Fitness function:

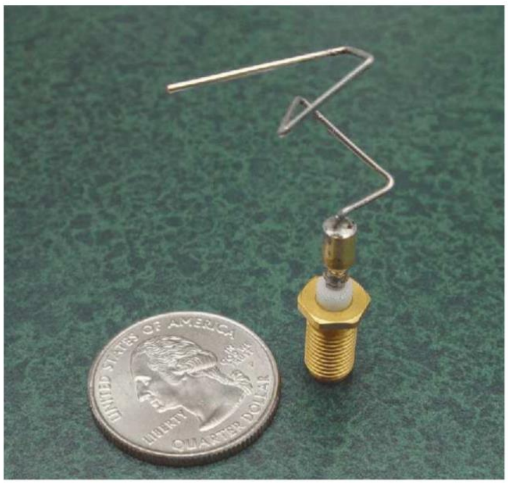
$$F = vsrw \times gain_{error} \times gain_{outlier}$$

vswr – standing wave ratio

The objective of the EA is to produce antenna designs that minimize F.

Results:





Conclusion:

The evolutionary algorithms we used were not limited to variations of previously developed antenna shapes but generated and tested thousands of completely new types of designs, many of which have unusual structures that expert antenna designers would not be likely to produce. By exploring such a wide range of designs EAs may be able to produce designs of previously unachievable performance.

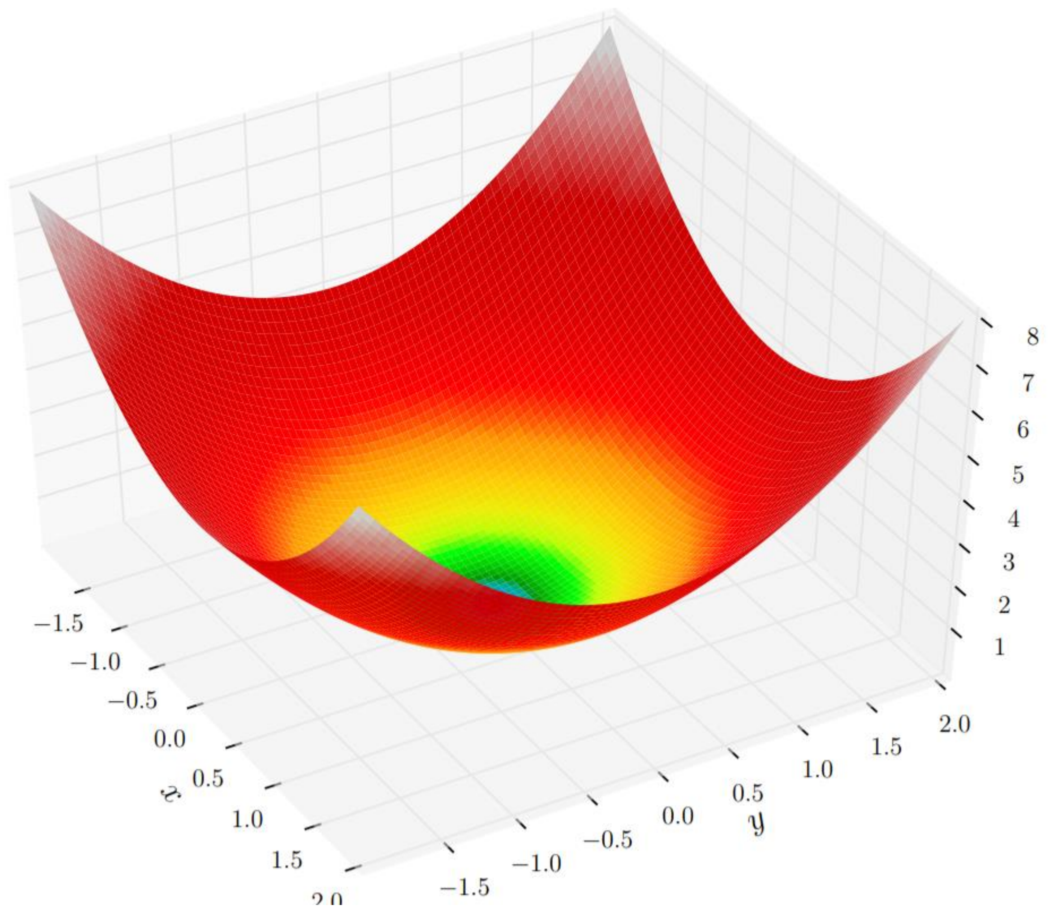
Authors Conclusion:

Python is an extremely powerful tool that used in combination with different packages allows user to create unlimited number of algorithms, models etc. Using different formats, such as csv, json, html etc. allows scripts to work together and allows for clean workflow and teamwork.

Appendix 1 – Test functions for optimization used in the project

Sphere function

$$f(x) = \sum_{i=1}^{n} x_i^2$$

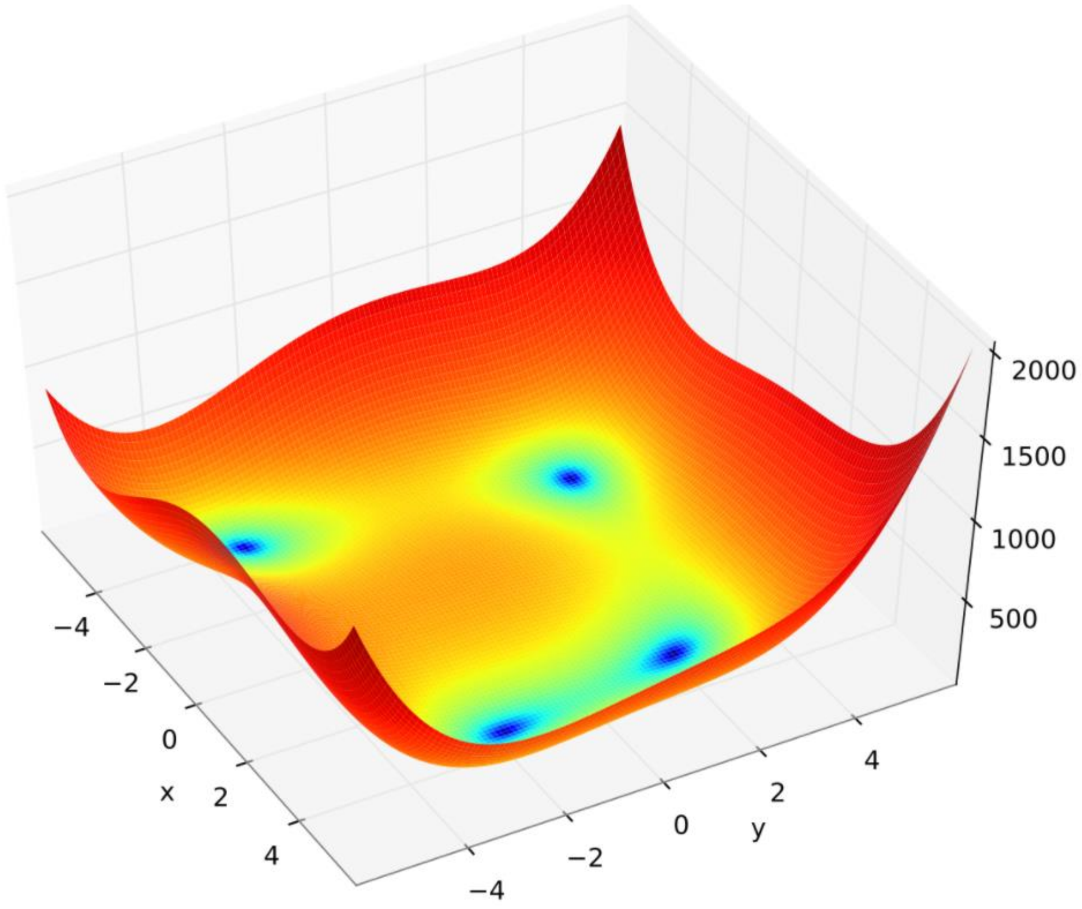


Global Minimum

$$f(x_1, x_2, \dots, x_n) = f(0,0,\dots,0) = 0$$

Himmelblau's function

$f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$

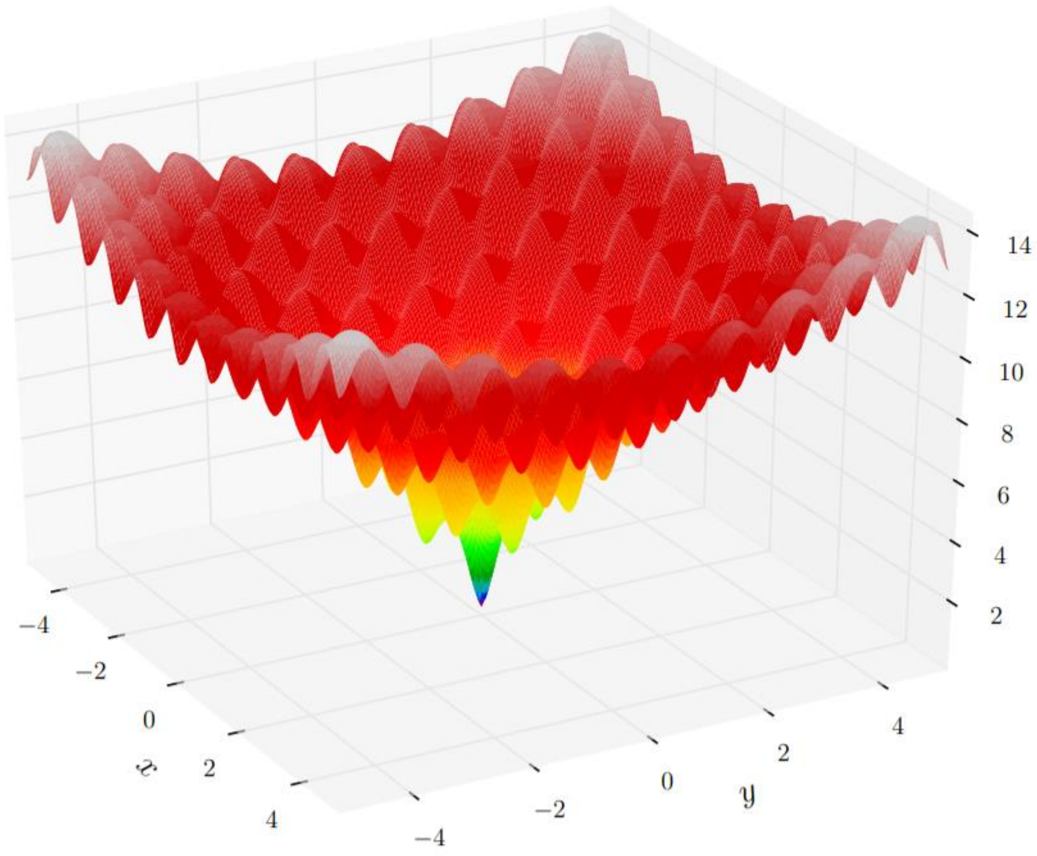


Global minimum:

$$\begin{cases} f(3,2) = 0 \\ f(-2.8,3.13) = 0 \\ f(-3.77,-3.28) = 0 \\ f(3.58,-1{,}84) = 0 \end{cases}$$

Ackley function

$$f(x,y) = -20 \times e^{-0.2\sqrt{0.5\times(x^2+y^2)}} - e^{0.5\times(\cos(2\pi x)+\cos(2\pi y))} + e + 20$$

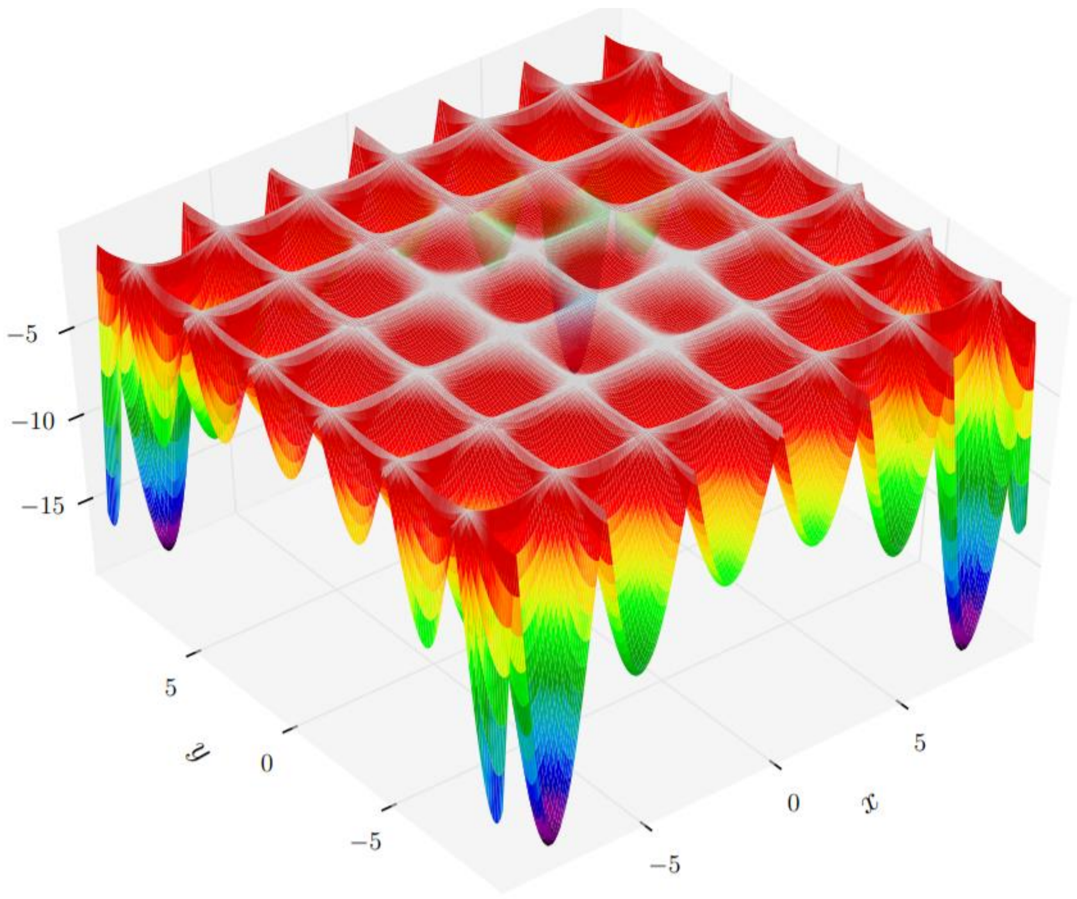


Global Minimum

$$f(0,0) = 0$$

Hölder table function

$$f(x,y) = -\left|\sin x \cos y \times e^{\left|1-\frac{\sqrt{x^2+y^2}}{\pi}\right|}\right|$$



Global minimum

$$\begin{cases} f(8.06,9.66) = -19.2085 \\ f(-8.06,9.66) = -19.2085 \\ f(8.06,-9.66) = -19.2085 \\ f(-8.06,-9.66) = -19.2085 \end{cases}$$

Appendix 2 – Sources

GA technical background

https://link.springer.com/article/10.1007/s11042-020-10139-6

https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

https://en.wikipedia.org/wiki/Genetic_algorithm

https://en.wikipedia.org/wiki/Evolutionary_algorithm

Test functions

https://en.wikipedia.org/wiki/Test_functions_for_optimization

https://towardsdatascience.com/optimization-eye-pleasure-78-benchmark-test-functions-for-single-objective-optimization-92e7ed1d1f12 (very broad range of test functions)

Real life implementation

https://ti.arc.nasa.gov/m/pub-archive/1244h/1244%20%28Hornby%29.pdf