

# ОПТИМІЗАЦІЯ С ГЕНЕТИЧНИ АЛГОРИТМИ

Research described in this tutorial was partially supported by the National Scientific Program "Information and Communication Technologies for a Single Digital Market in Science, Education and Security (ICTinSES)", financed by the Ministry of Education and Science.

Developed by Velbazhd Software LLC

Числената оптимизация в многомерни пространства е едно от предизвикателствата в приложната математика. Когато размерността на пространството е значителна ( $>50$ ), търсеният оптимум е на силно нелинейна функция или функция с прекъсвания, оптимизационната задача става дори по-сложна.



При трудните оптимизационни задачи много често се прибегва до използване на евристични числени методи. Такива методи са генетичните алгоритми, еволюция на разликите, рояк от частици, симулирано закаляване, колония на мравките и други.



При генетичните алгоритми се работи в пространството на решенията, като всяка точка в това пространство представлява вектор от стойности. В термините на генетичните алгоритми всяка точка в многомерното пространство се нарича хромозома.





Хромозомите се подлагат на оценка, като се подават за пресмятане от целевата функция (функцията на която се търси оптимум). Резултатът от пресмятането е едно число, наречено жизнена стойност.



Множество от случайно избрани решения (размерът на множеството се определя експериментално) формира популация в генетичния алгоритъм. На практика, това са решения от които някои по-близо до търсения оптимум, други по-далече.



Популацията се подлага на процес наречен еволюция. При еволюцията старото поколение в популацията се репродуцира и създава ново поколение. Репродукцията в класическите генетични алгоритми се получава след кръстосване и мутация.



По предварително избран метод за селекция (изборът е свързан с това по-жизнените хромозоми да стават родители) две хромозоми (в някои модификации може и повече) стават родители на две нови хромозоми (деца).





Кръстосването най-често е размяна на първата половина от двете хромозоми при случайна точка на срязване. Друг много популярен подход е равномерното кръстосване, където на случаен принцип се решава за всеки алел в родителите към кое дете да премине.



За мутацията има множество различни подходи, но най-често се избира един алел и той бива подложен на мутация с някаква малка стойност. Това се случва след кръстосването.



Ново полученото поколение се подлага на оценка, при което се пресмята целевата функция за всяка нова хромозома и се определя нейната жизнена стойност. Целта е по-жизнените хромозоми да останат в популацията, а по-слабите да отпаднат.



С цел да се запазят най-добрите намерени решения, понякога се прилага правило на елита. При правилото на елита предварително определен процент хромозоми от популацията, които са показали най-добри резултати за целия процес по еволюцията, винаги остават част от популацията и не изпадат от нея.





При стохастичните евристични методи се използват случайни (или псевдослучайни) числа за извършване на много от пресмятанията. Това налага този вид евристични методи да бъдат старателно изследвани, така че да се установи тяхната ефективност. Това изследване се прави с помощта на еталонни функции.



Sphere Function е функция с която е добре да се започне при провеждането на тестовете. Функцията е непрекъснатата, има добре изявен и известен глобален оптимум.



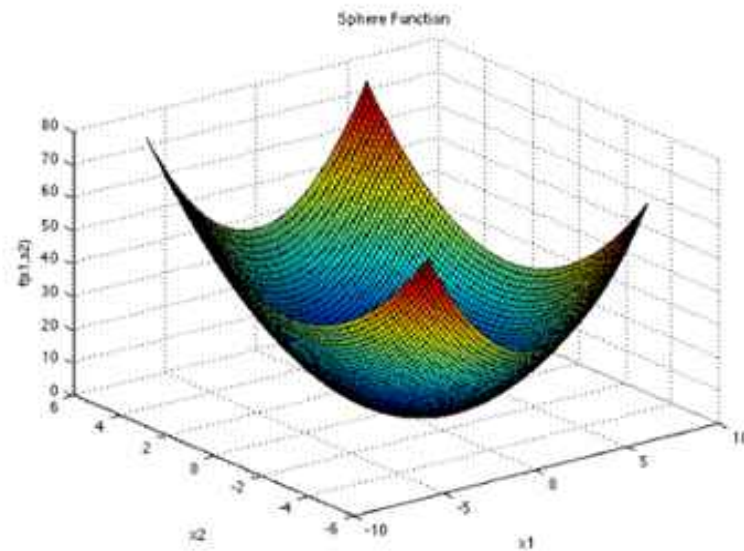
# Virtual Library of Simulation Experiments:

## Test Functions and Datasets

HOME
OPTIMIZATION
EMULATION/ PREDICTION
UNCERTAINTY QUANTIFICATION
MULTI FIDELITY SIMULATION
CALIBRATION/ TUNING
SCREENING
INTEGRATION
FUNCTIONAL DATA
ABOUT
OTHER TEST FUNCTIONS AND CODE

### Optimization Test Problems

#### SPHERE FUNCTION



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

#### Description:

*Dimensions:*  $d$

The Sphere function has  $d$  local minima except for the global one. It is continuous, convex and unimodal. The plot shows its two-dimensional form.

#### Input Domain:

The function is usually evaluated on the hypercube  $x_i \in [-5.12, 5.12]$ , for all  $i = 1, \dots, d$ .

#### Global Minimum:

$$f(\mathbf{x}^*) = 0, \text{ at } \mathbf{x}^* = (0, \dots, 0)$$

Styblinski-Tang Function е следващата по сложност функция, която се характеризира също с добре известен глобален оптимум, но не е чак толкова плавна и има полегатото дъно.





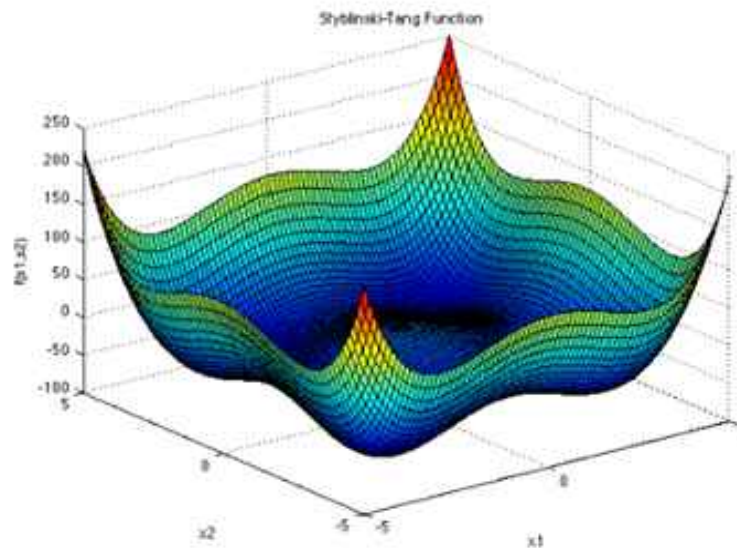
# Virtual Library of Simulation Experiments:

## Test Functions and Datasets

HOME
OPTIMIZATION
EMULATION/ PREDICTION
UNCERTAINTY QUANTIFICATION
MULTI FIDELITY SIMULATION
CALIBRATION/ TUNING
SCREENING
INTEGRATION
FUNCTIONAL DATA
ABOUT
OTHER TEST FUNCTIONS AND CODE

### Optimization Test Problems

#### STYBLINSKI-TANG FUNCTION



$$f(\mathbf{x}) = \frac{1}{2} \sum_{i=1}^d (x_i^4 - 16x_i^2 + 5x_i)$$

#### Description:

*Dimensions: d*

The Styblinski-Tang function is shown here in its two-dimensional form.

#### Input Domain:

The function is usually evaluated on the hypercube  $x_i \in [-5, 5]$ , for all  $i = 1, \dots, d$ .

#### Global Minimum:

$$f(\mathbf{x}^*) = -39.16599d, \text{ at } \mathbf{x}^* = (-2.903534, \dots, -2.903534)$$

Rosenbrock Function функцията също има добре известен глобален оптимум, ясно оформена долина и полегати склонове.



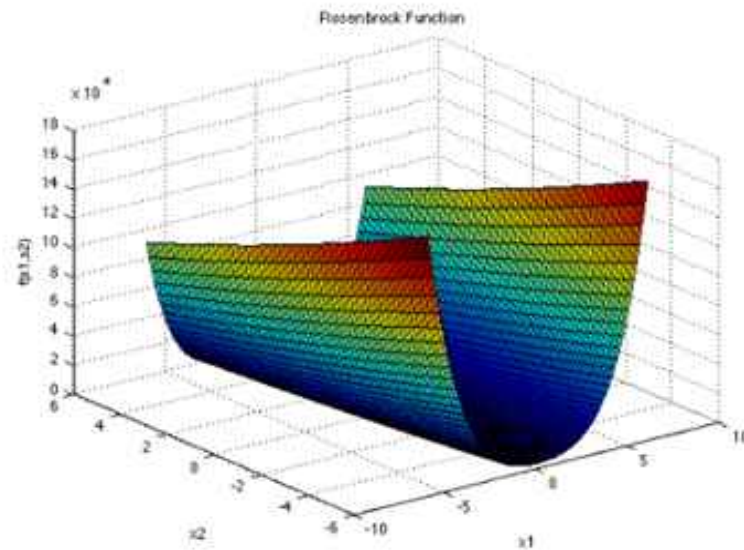
# Virtual Library of Simulation Experiments:

## Test Functions and Datasets

HOME
OPTIMIZATION
EMULATION/ PREDICTION
UNCERTAINTY QUANTIFICATION
MULTI FIDELITY SIMULATION
CALIBRATION/ TUNING
SCREENING
INTEGRATION
FUNCTIONAL DATA
ABOUT
OTHER TEST FUNCTIONS AND CODE

### Optimization Test Problems

## ROSENBROCK FUNCTION



$$f(\mathbf{x}) = \sum_{i=1}^{d-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$$

### Description:

*Dimensions:*  $d$

The Rosenbrock function, also referred to as the Valley or Banana function, is a popular test problem for gradient-based optimization algorithms. It is shown in the plot above in its two-dimensional form.

The function is unimodal, and the global minimum lies in a narrow, parabolic valley. However, even though this valley is easy to find, convergence to the minimum is difficult (Picheny et al., 2012).

### Input Domain:

The function is usually evaluated on the hypercube  $x_i \in [-5, 10]$ , for all  $i = 1, \dots, d$ , although it may be restricted to the hypercube  $x_i \in [-2.048, 2.048]$ , for all  $i = 1, \dots, d$ .

Rastrigin Function функцията представлява най-голямо предизвикателство то всички изброени до момента. Тя има известен глобален оптимум, глобален тренд за схождение, но и множество локални оптимуми.





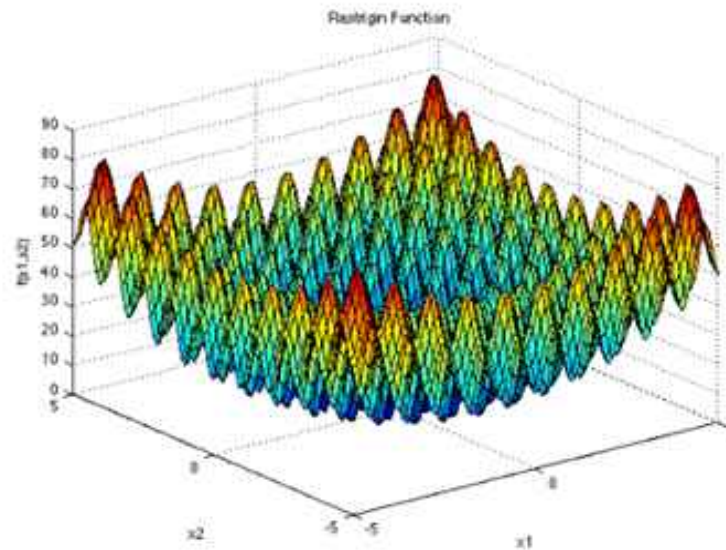
# Virtual Library of Simulation Experiments:

## Test Functions and Datasets

HOME
OPTIMIZATION
EMULATION/ PREDICTION
UNCERTAINTY QUANTIFICATION
MULTI FIDELITY SIMULATION
CALIBRATION/ TUNING
SCREENING
INTEGRATION
FUNCTIONAL DATA
ABOUT
OTHER TEST FUNCTIONS AND CODE

### Optimization Test Problems

#### RASTRIGIN FUNCTION



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

#### Description:

*Dimensions:  $d$*

The Rastrigin function has several local minima. It is highly multimodal, but locations of the minima are regularly distributed. It is shown in the plot above in its two-dimensional form.

#### Input Domain:

The function is usually evaluated on the hypercube  $x_i \in [-5.12, 5.12]$ , for all  $i = 1, \dots, d$ .

#### Global Minimum:

$f(\mathbf{x}^*) = 0$ , at  $\mathbf{x}^* = (0, \dots, 0)$

Библиотеката Genetic Algorithms - Apache Commons ще бъде използвана за да се демонстрира как с генетични алгоритми може да се търсят оптимални или субоптимални решения в многомерни пространства.





## MATH

- [Overview](#)
- [Downloads](#)
- [Latest API docs \(development\)](#)
- [Javadoc \(3.6.1 release\)](#)
- [Javadoc \(3.6 release\)](#)
- [Javadoc \(3.5 release\)](#)
- [Javadoc \(3.4.1 release\)](#)
- [Javadoc \(3.4 release\)](#)
- [Javadoc \(3.3 release\)](#)
- [Javadoc \(3.2 release\)](#)
- [Javadoc \(3.1.1 release\)](#)
- [Javadoc \(3.1 release\)](#)
- [Javadoc \(3.0 release\)](#)
- [Javadoc \(2.2 release\)](#)
- [Issue Tracking](#)
- [Source Repository \(current\)](#)
- [Wiki](#)
- [Developers Guide](#)
- [Proposal](#)

## USER GUIDE

- [Contents](#)
- [Overview](#)
- [Statistics](#)
- [Data Generation](#)
- [Linear Algebra](#)
- [Numerical Analysis](#)
- [Special Functions](#)
- [Utilities](#)
- [Complex Numbers](#)
- [Distributions](#)
- [Fractions](#)
- [Transform Methods](#)
- [Geometry](#)
- [Optimization](#)
- [Curve Fitting](#)
- [Least Squares](#)

# 16 Genetic Algorithms

## 16.1 Overview

The genetics package provides a framework and implementations for genetic algorithms.

## 16.2 GA Framework

`GeneticAlgorithm` provides an execution framework for Genetic Algorithms (GA). `Populations`, consisting of `Chromosomes` are evolved by the `GeneticAlgorithm` until a `StoppingCondition` is reached. Evolution is determined by `SelectionPolicy`, `MutationPolicy` and `Fitness`.

The GA itself is implemented by the `evolve` method of the `GeneticAlgorithm` class, which looks like this:

```
public Population evolve(Population initial, StoppingCondition condition) {
    Population current = initial;
    while (!condition.isSatisfied(current)) {
        current = nextGeneration(current);
    }
    return current;
}
```

The `nextGeneration` method implements the following algorithm:

1. Get `nextGeneration` population to fill from `current` generation, using its `nextGeneration` method
2. Loop until new generation is filled:
  - Apply configured `SelectionPolicy` to select a pair of parents from `current`
  - With probability = `getCrossoverRate()`, apply configured `CrossoverPolicy` to parents
  - With probability = `getMutationRate()`, apply configured `MutationPolicy` to each of the offspring
  - Add offspring individually to `nextGeneration`, space permitting
3. Return `nextGeneration`

## 16.3 Implementation

Here is an example GA execution:

```
// initialize a new genetic algorithm
GeneticAlgorithm ga = new GeneticAlgorithm(
    new OnePointCrossover<Integer>(),
    1,
    new RandomKeyMutation()
```

За демонстрацията е достатъчно да се подготви Java програма, работеща в команден режим.





GeneticAlgorithmsExample.java

```
1 public class GeneticAlgorithmsExample {  
2     public static void main(String args[]) {  
3  
4     }  
5 }
```

Полиморфизмът в обектно-ориентиран език като Java много добре може да се демонстрира с реализация на функциите, които ще бъдат използвани за оптимизацията. Интерфейс ще послужи за корен на йерархията от наследявания. Всяка функция получава вектор от реални числа и връща едно реално число в резултат на пресмятането.



```
1 import java.util.List;
2
3 interface Function {
4     double value(List<Double> input);
5 }
6
7 public class GeneticAlgorithmsExample {
8     public static void main(String args[]) {
9
10     }
11 }
```

За всяка една от избраните функции се създава клас, който имплементира общия интерфейс. Всеки от наследниците реализира формулата описваща съответната функция.





```
1 import java.util.List;
2
3 interface Function {
4     double value(List<Double> input);
5 }
6
7 class SphereFunction implements Function {
8     @Override
9     public double value(List<Double> input) {
10         double output = 0;
11
12         for (Double value : input) {
13             output += value * value;
14         }
15
16         return output;
17     }
18 }
19
20 public class GeneticAlgorithmsExample {
21     public static void main(String args[]) {
22
23     }
24 }
```

```
GeneticAlgorithmsExample.java
1 import java.util.List;
2
3 interface Function {
4     double value(List<Double> input);
5 }
6
7 class SphereFunction implements Function {
8     @Override
9     public double value(List<Double> input) {
10         double output = 0;
11
12         for (Double value : input) {
13             output += value * value;
14         }
15
16         return output;
17     }
18 }
19
20 class StyblinskiTangFunction implements Function {
21     @Override
22     public double value(List<Double> input) {
23         double output = 0;
24
25         for (Double value : input) {
26             output += value * value * value * value - 16 * value * value
27                 + 5 * value;
28         }
29
30         return output / 20;
31     }
32 }
33
34 public class GeneticAlgorithmsExample {
35     public static void main(String args[]) {
36
37     }
```

```
19
20 class StyblinskiTangFunction implements Function {
21     @Override
22     public double value(List<Double> input) {
23         double output = 0;
24
25         for (Double value : input) {
26             output += value * value * value * value - 16 * value * value
27                 + 5 * value;
28         }
29
30         return output / 20;
31     }
32 }
33
34 class RosenbrockFunction implements Function {
35     @Override
36     public double value(List<Double> input) {
37         double output = 0;
38
39         for (int i = 0; i < input.size() - 1; i++) {
40             double xi0 = input.get(i);
41             double xi1 = input.get(i + 1);
42
43             output += 100 * (xi1 - xi0 * xi0) * (xi1 - xi0 * xi0)
44                 + (1 - xi0) * (1 - xi0);
45         }
46
47         return output;
48     }
49 }
50
51 public class GeneticAlgorithmsExample {
52     public static void main(String args[]) {
53
54     }
55 }
```

```

35=  @Override
36  public double value(List<Double> input) {
37      double output = 0;
38
39      for (int i = 0; i < input.size() - 1; i++) {
40          double xi0 = input.get(i);
41          double xi1 = input.get(i + 1);
42
43          output += 100 * (xi1 - xi0 * xi0) * (xi1 - xi0 * xi0)
44                  + (1 - xi0) * (1 - xi0);
45      }
46
47      return output;
48  }
49 }
50
51 class RastriginFunction implements Function {
52=  @Override
53  public double value(List<Double> input) {
54      double output = 0;
55
56      for (Double value : input) {
57          output += value * value - 10 * Math.cos(2 * Math.PI * value);
58      }
59
60      output += 10 * input.size();
61
62      return output;
63  }
64 }
65
66 public class GeneticAlgorithmsExample {
67=  public static void main(String args[]) {
68
69  }
70 }
71

```



Библиотеката позволява различни възможности за кодиране на хромозомите, но за така описаните функции най-удачно е да се използва хромозома с реални числа.



```

40
41     for (int i = 0; i < input.size() - 1; i++) {
42         double xi0 = input.get(i);
43         double xi1 = input.get(i + 1);
44
45         output += 100 * (xi1 - xi0 * xi0) * (xi1 - xi0 * xi0)
46                 + (1 - xi0) * (1 - xi0);
47     }
48
49     return output;
50 }
51 }
52
53 class RastriginFunction implements Function {
54     @Override
55     public double value(List<Double> input) {
56         double output = 0;
57
58         for (Double value : input) {
59             output += value * value - 10 * Math.cos(2 * Math.PI * value);
60         }
61
62         output += 10 * input.size();
63
64         return output;
65     }
66 }
67
68 class DoubleListChromosome extends AbstractListChromosome<Double> {
69 }
70
71 public class GeneticAlgorithmsExample {
72     public static void main(String args[]) {
73
74     }
75 }
76

```

Наследяването на базовия клас за хромозоми изисква да се създадат описания на поне четири метода – конструктор, целева функция, метод за валидация на хромозомата (когато това е нужно) и метод за генериране на нова хромозома (в случая с фиксирана дължина).



```
GeneticAlgorithmsExample.java
57     double output = 0;
58
59     for (Double value : input) {
60         output += value * value - 10 * Math.cos(2 * Math.PI * value);
61     }
62
63     output += 10 * input.size();
64
65     return output;
66 }
67 }
68
69 class DoubleListChromosome extends AbstractListChromosome<Double> {
70     public DoubleListChromosome(Double[] representation)
71         throws InvalidRepresentationException {
72         super(representation);
73     }
74
75     @Override
76     public double fitness() {
77         return 0;
78     }
79
80     @Override
81     protected void checkValidity(List<Double> representation)
82         throws InvalidRepresentationException {
83     }
84
85     @Override
86     public AbstractListChromosome<Double> newFixedLengthChromosome(
87         List<Double> representation) {
88         return null;
89     }
90 }
91
92 public class GeneticAlgorithmsExample {
93     public static void main(String args[]) {
```



Тъй като формулите на функциите са изнесени в собствени класове, всяка хромозома е нужно да използва референция към функционален обект (обект само с методи, без член-променливи), с помощта на който да извършва пресмятанията.



```

59     for (Double value : input) {
60         output += value * value - 10 * Math.cos(2 * Math.PI * value);
61     }
62
63     output += 10 * input.size();
64
65     return output;
66 }
67 }
68
69 class DoubleListChromosome extends AbstractListChromosome<Double> {
70     private Function function = null;
71
72     public DoubleListChromosome(Double[] representation, Function function)
73         throws InvalidRepresentationException {
74         super(representation);
75         this.function = function;
76     }
77
78     @Override
79     public double fitness() {
80         return 0;
81     }
82
83     @Override
84     protected void checkValidity(List<Double> representation)
85         throws InvalidRepresentationException {
86     }
87
88     @Override
89     public AbstractListChromosome<Double> newFixedLengthChromosome(
90         List<Double> representation) {
91         return null;
92     }
93 }
94
95 public class GeneticAlgorithmsExample {

```

Всички избрани тестови функции имат глобален минимум. Малко е объркващо, но софтуерната библиотека изисква търсене на глобален максимум и поради тази причина функциите се използват със знак минус за определяне на жизнеността.



```
GeneticAlgorithmsExample.java
64
65     return output;
66 }
67 }
68
69 class DoubleListChromosome extends AbstractListChromosome<Double> {
70     private Function function = null;
71
72     public DoubleListChromosome(Double[] representation, Function function)
73         throws InvalidRepresentationException {
74         super(representation);
75         this.function = function;
76     }
77
78     @Override
79     public double fitness() {
80         return -function.value(getRepresentation());
81     }
82
83     @Override
84     protected void checkValidity(List<Double> representation)
85         throws InvalidRepresentationException {
86     }
87
88     @Override
89     public AbstractListChromosome<Double> newFixedLengthChromosome(
90         List<Double> representation) {
91         return null;
92     }
93 }
94
95 public class GeneticAlgorithmsExample {
96     public static void main(String args[]) {
97
98     }
99 }
100
```



Създаването на нова хромозома може да се изпълни чрез един частен конструктор и извикването му в метода.



```
GeneticAlgorithmsExample.java
64
65     return output;
66 }
67 }
68
69 class DoubleListChromosome extends AbstractListChromosome<Double> {
70     private Function function = null;
71
72     private DoubleListChromosome(List<Double> representation, Function function)
73         throws InvalidRepresentationException {
74         super(representation);
75         this.function = function;
76     }
77
78     public DoubleListChromosome(Double[] representation, Function function)
79         throws InvalidRepresentationException {
80         super(representation);
81         this.function = function;
82     }
83
84     @Override
85     public double fitness() {
86         return -function.value(getRepresentation());
87     }
88
89     @Override
90     protected void checkValidity(List<Double> representation)
91         throws InvalidRepresentationException {
92     }
93
94     @Override
95     public AbstractListChromosome<Double> newFixedLengthChromosome(
96         List<Double> representation) {
97         return new DoubleListChromosome(representation, function);
98     }
99 }
100
```

Равномерното кръстосване се реализира  
чрез наследяване на политика за  
кръстосване.



```

74         throws InvalidRepresentationException {
75         super(representation);
76         this.function = function;
77     }
78
79     public DoubleListChromosome(Double[] representation, Function function)
80         throws InvalidRepresentationException {
81         super(representation);
82         this.function = function;
83     }
84
85     @Override
86     public double fitness() {
87         return -function.value(getRepresentation());
88     }
89
90     @Override
91     protected void checkValidity(List<Double> representation)
92         throws InvalidRepresentationException {
93     }
94
95     @Override
96     public AbstractListChromosome<Double> newFixedLengthChromosome(
97         List<Double> representation) {
98         return new DoubleListChromosome(representation, function);
99     }
100 }
101
102 class UniformCrossover implements CrossoverPolicy {
103 }
104
105 public class GeneticAlgorithmsExample {
106     public static void main(String args[]) {
107
108     }
109 }
110

```



Кръстосването е процес при който два родители трябва да разменят гените си и да се получат две деца.



```

GeneticAlgorithmsExample.java
82= public DoubleListChromosome(Double[] representation, Function function)
83     throws InvalidRepresentationException {
84     super(representation);
85     this.function = function;
86 }
87
88= @Override
89 public double fitness() {
90     return -function.value(getRepresentation());
91 }
92
93= @Override
94 protected void checkValidity(List<Double> representation)
95     throws InvalidRepresentationException {
96 }
97
98= @Override
99 public AbstractListChromosome<Double> newFixedLengthChromosome(
100     List<Double> representation) {
101     return new DoubleListChromosome(representation, function);
102 }
103 }
104
105 class UniformCrossover implements CrossoverPolicy {
106= @Override
107 public ChromosomePair crossover(Chromosome first, Chromosome second)
108     throws MathIllegalArgumentException {
109     return null;
110 }
111 }
112
113 public class GeneticAlgorithmsExample {
114= public static void main(String args[]) {
115
116 }
117 }
118

```

Помощна функция с пакетен обseg на видимост позволява да се вземат числените стойности от родителите и те да бъдат използвани за съставянето на децата.



```

GeneticAlgorithmsExample.java
89     public double fitness() {
90         return -function.value(getRepresentation());
91     }
92
93     @Override
94     protected void checkValidity(List<Double> representation)
95         throws InvalidRepresentationException {
96     }
97
98     @Override
99     public AbstractListChromosome<Double> newFixedLengthChromosome(
100         List<Double> representation) {
101         return new DoubleListChromosome(representation, function);
102     }
103
104     List<Double> getValues() {
105         return super.getRepresentation();
106     }
107 }
108
109 class UniformCrossover implements CrossoverPolicy {
110     @Override
111     public ChromosomePair crossover(Chromosome first, Chromosome second)
112         throws MathIllegalArgumentException {
113         final List<Double> parent1 = ((DoubleListChromosome) first).getValues();
114         final List<Double> parent2 = ((DoubleListChromosome) second).getValues();
115
116         return null;
117     }
118 }
119
120 public class GeneticAlgorithmsExample {
121     public static void main(String args[]) {
122
123     }
124 }
125

```



Всяко от децата представлява вектор от реални числа. При тази оптимизация дължината на векторите е еднаква, но в други оптимизационни проблеми е възможно векторите да имат различна дължина.



```

90     public double fitness() {
91         return -function.value(getRepresentation());
92     }
93
94     @Override
95     protected void checkValidity(List<Double> representation)
96         throws InvalidRepresentationException {
97     }
98
99     @Override
100    public AbstractListChromosome<Double> newFixedLengthChromosome(
101        List<Double> representation) {
102        return new DoubleListChromosome(representation, function);
103    }
104
105    List<Double> getValues() {
106        return super.getRepresentation();
107    }
108 }
109
110 class UniformCrossover implements CrossoverPolicy {
111     @Override
112     public ChromosomePair crossover(Chromosome first, Chromosome second)
113         throws MathIllegalArgumentException {
114         final List<Double> parent1 = ((DoubleListChromosome) first).getValues();
115         final List<Double> parent2 = ((DoubleListChromosome) second).getValues();
116
117         final List<Double> child1 = new ArrayList<Double>();
118         final List<Double> child2 = new ArrayList<Double>();
119
120         return null;
121     }
122 }
123
124 public class GeneticAlgorithmsExample {
125     public static void main(String args[]) {
126

```

Алелите в двата родителя се обхождат последователно и на случаен принцип се взема решение кой алел в кое дете да попадне.





```

101 public AbstractListChromosome<Double> newFixedLengthChromosome(
102     List<Double> representation) {
103     return new DoubleListChromosome(representation, function);
104 }
105
106 List<Double> getValues() {
107     return super.getRepresentation();
108 }
109 }
110
111 class UniformCrossover implements CrossoverPolicy {
112     private static Random PRNG = new Random();
113
114     @Override
115     public ChromosomePair crossover(Chromosome first, Chromosome second)
116         throws MathIllegalArgumentException {
117         final List<Double> parent1 = ((DoubleListChromosome) first).getValues();
118         final List<Double> parent2 = ((DoubleListChromosome) second).getValues();
119
120         final List<Double> child1 = new ArrayList<Double>();
121         final List<Double> child2 = new ArrayList<Double>();
122
123         for (int i = 0; i < parent1.size() && i < parent2.size(); i++) {
124             if (PRNG.nextBoolean() == true) {
125                 } else {
126                 }
127         }
128
129         return null;
130     }
131 }
132
133 public class GeneticAlgorithmsExample {
134     public static void main(String args[]) {
135
136     }
137 }

```



Размяната се прави алел по алел, а не както е при класическото кръстосване с една точка на срязване.



```

GeneticAlgorithmsExample.java
101 public AbstractListChromosome<Double> newFixedLengthChromosome(
102     List<Double> representation) {
103     return new DoubleListChromosome(representation, function);
104 }
105
106 List<Double> getValues() {
107     return super.getRepresentation();
108 }
109 }
110
111 class UniformCrossover implements CrossoverPolicy {
112     private static Random PRNG = new Random();
113
114     @Override
115     public ChromosomePair crossover(Chromosome first, Chromosome second)
116         throws MathIllegalArgumentException {
117         final List<Double> parent1 = ((DoubleListChromosome) first).getValues();
118         final List<Double> parent2 = ((DoubleListChromosome) second).getValues();
119
120         final List<Double> child1 = new ArrayList<Double>();
121         final List<Double> child2 = new ArrayList<Double>();
122
123         for (int i = 0; i < parent1.size() && i < parent2.size(); i++) {
124             if (PRNG.nextBoolean() == true) {
125                 child1.add(parent1.get(i));
126                 child2.add(parent2.get(i));
127             } else {
128                 child1.add(parent2.get(i));
129                 child2.add(parent1.get(i));
130             }
131         }
132
133         return null;
134     }
135 }
136
137 public class GeneticAlgorithmsExample {

```

Двете деца формират двойка новосъздадени  
хромозоми.





```

GeneticAlgorithmsExample.java
106=    List<Double> getValues() {
107        return super.getRepresentation();
108    }
109 }
110
111 class UniformCrossover implements CrossoverPolicy {
112     private static Random PRNG = new Random();
113
114     @Override
115     public ChromosomePair crossover(Chromosome first, Chromosome second)
116         throws MathIllegalArgumentException {
117         final List<Double> parent1 = ((DoubleListChromosome) first).getValues();
118         final List<Double> parent2 = ((DoubleListChromosome) second).getValues();
119
120         final List<Double> child1 = new ArrayList<Double>();
121         final List<Double> child2 = new ArrayList<Double>();
122
123         for (int i = 0; i < parent1.size() && i < parent2.size(); i++) {
124             if (PRNG.nextBoolean() == true) {
125                 child1.add(parent1.get(i));
126                 child2.add(parent2.get(i));
127             } else {
128                 child1.add(parent2.get(i));
129                 child2.add(parent1.get(i));
130             }
131         }
132
133         return new ChromosomePair(
134             ((DoubleListChromosome) first).newFixedLengthChromosome(child1),
135             ((DoubleListChromosome) second).newFixedLengthChromosome(child2));
136     }
137 }
138
139 public class GeneticAlgorithmsExample {
140=     public static void main(String args[]) {
141
142     }

```



По аналогичен начин на кръстосването се реализира и мутацията, чрез наследяване на политика за мутация.



```

111 class UniformCrossover implements CrossoverPolicy {
112     private static Random PRNG = new Random();
113
114     @Override
115     public ChromosomePair crossover(Chromosome first, Chromosome second)
116         throws MathIllegalArgumentException {
117         final List<Double> parent1 = ((DoubleListChromosome) first).getValues();
118         final List<Double> parent2 = ((DoubleListChromosome) second).getValues();
119
120         final List<Double> child1 = new ArrayList<Double>();
121         final List<Double> child2 = new ArrayList<Double>();
122
123         for (int i = 0; i < parent1.size() && i < parent2.size(); i++) {
124             if (PRNG.nextBoolean() == true) {
125                 child1.add(parent1.get(i));
126                 child2.add(parent2.get(i));
127             } else {
128                 child1.add(parent2.get(i));
129                 child2.add(parent1.get(i));
130             }
131         }
132
133         return new ChromosomePair(
134             ((DoubleListChromosome) first).newFixedLengthChromosome(child1),
135             ((DoubleListChromosome) second).newFixedLengthChromosome(child2));
136     }
137 }
138
139 class RandomDoubleMutation implements MutationPolicy {
140 }
141
142 public class GeneticAlgorithmsExample {
143     public static void main(String args[]) {
144
145     }
146 }
147

```

На входа на метода за мутация постъпва хромозома, а на изхода се подава нова хромозома, отразяваща избраната операция за мутация.





```

117         throws MathIllegalArgumentException {
118         final List<Double> parent1 = ((DoubleListChromosome) first).getValues();
119         final List<Double> parent2 = ((DoubleListChromosome) second).getValues();
120
121         final List<Double> child1 = new ArrayList<Double>();
122         final List<Double> child2 = new ArrayList<Double>();
123
124         for (int i = 0; i < parent1.size() && i < parent2.size(); i++) {
125             if (PRNG.nextBoolean() == true) {
126                 child1.add(parent1.get(i));
127                 child2.add(parent2.get(i));
128             } else {
129                 child1.add(parent2.get(i));
130                 child2.add(parent1.get(i));
131             }
132         }
133
134         return new ChromosomePair(
135             ((DoubleListChromosome) first).newFixedLengthChromosome(child1),
136             ((DoubleListChromosome) second).newFixedLengthChromosome(child2));
137     }
138 }
139
140 class RandomDoubleMutation implements MutationPolicy {
141     @Override
142     public Chromosome mutate(Chromosome arg0)
143         throws MathIllegalArgumentException {
144         return null;
145     }
146 }
147
148 public class GeneticAlgorithmsExample {
149     public static void main(String args[]) {
150
151     }
152 }
153

```



Както при кръстосването, така и при мутацията резултатната хромозома има същата дължина както и родителската.



```

GeneticAlgorithmsExample.java
117     throws MathIllegalArgumentException {
118     final List<Double> parent1 = ((DoubleListChromosome) first).getValues();
119     final List<Double> parent2 = ((DoubleListChromosome) second).getValues();
120
121     final List<Double> child1 = new ArrayList<Double>();
122     final List<Double> child2 = new ArrayList<Double>();
123
124     for (int i = 0; i < parent1.size() && i < parent2.size(); i++) {
125         if (PRNG.nextBoolean() == true) {
126             child1.add(parent1.get(i));
127             child2.add(parent2.get(i));
128         } else {
129             child1.add(parent2.get(i));
130             child2.add(parent1.get(i));
131         }
132     }
133
134     return new ChromosomePair(
135         ((DoubleListChromosome) first).newFixedLengthChromosome(child1),
136         ((DoubleListChromosome) second).newFixedLengthChromosome(child2));
137 }
138 }
139
140 class RandomDoubleMutation implements MutationPolicy {
141     @Override
142     public Chromosome mutate(Chromosome original)
143         throws MathIllegalArgumentException {
144         List<Double> parent = ((DoubleListChromosome) original).getValues();
145         Double values[] = new Double[parent.size()];
146
147         return null;
148     }
149 }
150
151 public class GeneticAlgorithmsExample {
152     public static void main(String args[]) {
153

```

Възможни са различни начини за изпълнение на мутацията, но един от вариантите е добавяне/изваждане на малка стойност. В случая това върши работа, тъй като се работи с реални числа и няма ограничения за стойностите в отделните елементи на вектора.





```

126         child1.add(parent1.get(i));
127         child2.add(parent2.get(i));
128     } else {
129         child1.add(parent2.get(i));
130         child2.add(parent1.get(i));
131     }
132 }
133
134     return new ChromosomePair(
135         ((DoubleListChromosome) first).newFixedLengthChromosome(child1),
136         ((DoubleListChromosome) second).newFixedLengthChromosome(child2));
137 }
138 }
139
140 class RandomDoubleMutation implements MutationPolicy {
141     private static Random PRNG = new Random();
142
143     @Override
144     public Chromosome mutate(Chromosome original)
145         throws MathIllegalArgumentException {
146         List<Double> parent = ((DoubleListChromosome) original).getValues();
147         Double values[] = new Double[parent.size()];
148
149         for (int i = 0; i < values.length; i++) {
150             values[i] = parent.get(i) + PRNG.nextDouble() - 0.5D;
151         }
152
153         return null;
154     }
155 }
156
157 public class GeneticAlgorithmsExample {
158     public static void main(String args[]) {
159
160     }
161 }
162

```



Логично е новосъздадената хромозома да бъде използвана със същата функция с която се е използвала и родителската хромозома. Поради тази причина помощна функция връща референция към функционалния обект за пресмятането на жизнеността.



```

131         child2.add(parent2.get(i));
132     } else {
133         child1.add(parent2.get(i));
134         child2.add(parent1.get(i));
135     }
136 }
137
138     return new ChromosomePair(
139         ((DoubleListChromosome) first).newFixedLengthChromosome(child1),
140         ((DoubleListChromosome) second).newFixedLengthChromosome(child2));
141 }
142 }
143
144 class RandomDoubleMutation implements MutationPolicy {
145     private static Random PRNG = new Random();
146
147     @Override
148     public Chromosome mutate(Chromosome original)
149         throws MathIllegalArgumentException {
150         List<Double> parent = ((DoubleListChromosome) original).getValues();
151         Double values[] = new Double[parent.size()];
152
153         for (int i = 0; i < values.length; i++) {
154             values[i] = parent.get(i) + PRNG.nextDouble() - 0.5D;
155         }
156
157         return new DoubleListChromosome(values,
158             ((DoubleListChromosome) original).getFunction());
159     }
160 }
161
162 public class GeneticAlgorithmsExample {
163     public static void main(String args[]) {
164
165     }
166 }
167

```

Така подготвените класове влизат в употреба чрез обект за оптимизация с генетичен алгоритъм. На този обект се задава политиките за кръстосване и мутация. Задава се също процентът кръстосвания (в случая 95% и 1%). Избира се състезателна селекция с четност 2.





```

140     return new ChromosomePair(
141         ((DoubleListChromosome) first).newFixedLengthChromosome(child1),
142         ((DoubleListChromosome) second).newFixedLengthChromosome(child2));
143     }
144 }
145
146 class RandomDoubleMutation implements MutationPolicy {
147     private static Random PRNG = new Random();
148
149     @Override
150     public Chromosome mutate(Chromosome original)
151         throws MathIllegalArgumentException {
152         List<Double> parent = ((DoubleListChromosome) original).getValues();
153         Double values[] = new Double[parent.size()];
154
155         for (int i = 0; i < values.length; i++) {
156             values[i] = parent.get(i) + PRNG.nextDouble() - 0.5D;
157         }
158
159         return new DoubleListChromosome(values,
160             ((DoubleListChromosome) original).getFunction());
161     }
162 }
163
164 public class GeneticAlgorithmsExample {
165     private static double CROSSOVER_RATE = 0.9;
166     private static double MUTATION_RATE = 0.01;
167     private static int TOURNAMENT_AIRITY = 2;
168
169     public static void main(String args[]) {
170         GeneticAlgorithm algorithm = new GeneticAlgorithm(
171             new UniformCrossover(), CROSSOVER_RATE,
172             new RandomDoubleMutation(), MUTATION_RATE,
173             new TournamentSelection(TOURNAMENT_AIRITY));
174     }
175 }
176

```



Първият експеримент е с най-лесната от четирите функции, за която трябва да се създаде първоначална популация.



```

145     }
146 }
147
148 class RandomDoubleMutation implements MutationPolicy {
149     private static Random PRNG = new Random();
150
151     @Override
152     public Chromosome mutate(Chromosome original)
153         throws MathIllegalArgumentException {
154         List<Double> parent = ((DoubleListChromosome) original).getValues();
155         Double values[] = new Double[parent.size()];
156
157         for (int i = 0; i < values.length; i++) {
158             values[i] = parent.get(i) + PRNG.nextDouble() - 0.5D;
159         }
160
161         return new DoubleListChromosome(values,
162             ((DoubleListChromosome) original).getFunction());
163     }
164 }
165
166 public class GeneticAlgorithmsExample {
167     private static double CROSSOVER_RATE = 0.9;
168     private static double MUTATION_RATE = 0.01;
169     private static int TOURNAMENT_AIRITY = 2;
170
171     public static void main(String args[]) {
172         GeneticAlgorithm algorithm = new GeneticAlgorithm(
173             new UniformCrossover(), CROSSOVER_RATE,
174             new RandomDoubleMutation(), MUTATION_RATE,
175             new TournamentSelection(TOURNAMENT_AIRITY));
176
177         Function function = new SphereFunction();
178         List<Chromosome> list = new ArrayList<Chromosome>();
179     }
180 }
181

```

Размерът на популацията е параметър, който се определя експериментално и силно зависи от задачата, която се решава. Избраните тестови функции са многомерни, но за ефективно тестване е избрано 10 мерно пространство. Всяко начално решение е случайно избрана точка в многомерното пространство, близко до центъра на координатната система.





```

158         values[i] = parent.get(i) + PRNG.nextDouble() - 0.5D;
159     }
160
161     return new DoubleListChromosome(values,
162         ((DoubleListChromosome) original).getFunction());
163 }
164 }
165
166 public class GeneticAlgorithmsExample {
167     private static Random PRNG = new Random();
168
169     private static double CROSSOVER_RATE = 0.9;
170     private static double MUTATION_RATE = 0.01;
171     private static int TOURNAMENT_AIRITY = 2;
172
173     private static int POPULATION_SIZE = 37;
174     private static int CHROMOSOME_SIZE = 10;
175
176     public static void main(String args[]) {
177         GeneticAlgorithm algorithm = new GeneticAlgorithm(
178             new UniformCrossover(), CROSSOVER_RATE,
179             new RandomDoubleMutation(), MUTATION_RATE,
180             new TournamentSelection(TOURNAMENT_AIRITY));
181
182         Function function = new SphereFunction();
183         List<Chromosome> list = new ArrayList<Chromosome>();
184
185         for (int i = 0; i < POPULATION_SIZE; i++) {
186             Double values[] = new Double[CHROMOSOME_SIZE];
187             for (int j = 0; j < values.length; j++) {
188                 values[j] = new Double(0.5D - PRNG.nextDouble());
189             }
190             list.add(new DoubleListChromosome(values, function));
191         }
192     }
193 }
194

```



Библиотеката работи на принципа на начална популация и оптимизирана популация. В случая е използвана еластична популация при която е избрано 10% от популацията да служи като елит. Този параметър също се определя експериментално и силно зависи от решавания проблем.



```

165     }
166 }
167
168 public class GeneticAlgorithmsExample {
169     private static Random PRNG = new Random();
170
171     private static double CROSSOVER_RATE = 0.9;
172     private static double MUTATION_RATE = 0.01;
173     private static int TOURNAMENT_AIRITY = 2;
174
175     private static int POPULATION_SIZE = 37;
176     private static int CHROMOSOME_SIZE = 10;
177
178     private static double ELITISM_RATE = 0.1;
179
180     public static void main(String args[]) {
181         GeneticAlgorithm algorithm = new GeneticAlgorithm(
182             new UniformCrossover(), CROSSOVER_RATE,
183             new RandomDoubleMutation(), MUTATION_RATE,
184             new TournamentSelection(TOURNAMENT_AIRITY));
185
186         Function function = new SphereFunction();
187         List<Chromosome> list = new ArrayList<Chromosome>();
188
189         for (int i = 0; i < POPULATION_SIZE; i++) {
190             Double values[] = new Double[CHROMOSOME_SIZE];
191             for (int j = 0; j < values.length; j++) {
192                 values[j] = new Double(0.5D - PRNG.nextDouble());
193             }
194             list.add(new DoubleListChromosome(values, function));
195         }
196
197         Population initial, optimized;
198         initial = optimized = new ElitisticListPopulation(list, list.size(), ELITISM_RATE);
199     }
200 }
201

```

Разпечатват се стойностите на най-добрата хромозома преди оптимизацията и след оптимизацията. Като критерии за спиране на оптимизационния процес се използва фиксиран интервал от време (в случая 60 секунди).





```
172 private static double CROSSOVER_RATE = 0.9;
173 private static double MUTATION_RATE = 0.01;
174 private static int TOURNAMENT_AIRITY = 2;
175
176 private static int POPULATION_SIZE = 37;
177 private static int CHROMOSOME_SIZE = 10;
178
179 private static double ELITISM_RATE = 0.1;
180
181 private static int SINGLE_OPTIMIZATION_SECONDS = 60;
182
183 public static void main(String args[]) {
184     GeneticAlgorithm algorithm = new GeneticAlgorithm(
185         new UniformCrossover(), CROSSOVER_RATE,
186         new RandomDoubleMutation(), MUTATION_RATE,
187         new TournamentSelection(TOURNAMENT_AIRITY));
188
189     Function function = new SphereFunction();
190     List<Chromosome> list = new ArrayList<Chromosome>();
191
192     for (int i = 0; i < POPULATION_SIZE; i++) {
193         Double values[] = new Double[CHROMOSOME_SIZE];
194         for (int j = 0; j < values.length; j++) {
195             values[j] = new Double(0.5D - PRNG.nextDouble());
196         }
197         list.add(new DoubleListChromosome(values, function));
198     }
199
200     Population initial, optimized;
201     initial = optimized = new ElitisticListPopulation(list, list.size(),
202         ELITISM_RATE);
203
204     System.out.println(optimized.getFittestChromosome());
205     optimized = algorithm.evolve(initial, new FixedElapsedTime(SINGLE_OPTIMIZATION_SECONDS));
206     System.out.println(optimized.getFittestChromosome());
207 }
208 }
```



При първата функция оптимумът е нула и се получава при нулев вектор.



```
155 public Chromosome
156     throws MutationException
157     List<Double> values
158     Double value
159
160     for (int i =
161         values[i]
162     }
163
164     return new Chromosome(
165         ((Double[]) values)
166     )
167 }
168
169 public class GeneticAlgorithm
170     private static final int POPULATION_SIZE = 100;
171
172     private static final double CROSSOVER_RATE = 0.8;
173     private static final double MUTATION_RATE = 0.01;
174     private static final int TOURNAMENT_SIZE = 3;
175
176     private static final double MAX_ITERATIONS = 1000;
177     private static final double MIN_FITNESS = 0.0;
178
179     private static final double MAX_FITNESS = 1.0;
180
181     private static final double MIN_CROSSOVER_RATE = 0.5;
182
183     public static void main(String args[]) {
184         GeneticAlgorithm algorithm = new GeneticAlgorithm(
185             new UniformCrossover(), CROSSOVER_RATE,
186             new RandomDoubleMutation(), MUTATION_RATE,
187             new TournamentSelection(TOURNAMENT_SIZE));
188
189         Function function = new SphereFunction();
190         List<Chromosome> list = new ArrayList<Chromosome>();
191     }
```

<terminated> GeneticAlgorithmsExample [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_91.jdk/Contents/Home/bin/java (Mar 31, 2020, 5:43:35 PM)

(f=-0.41233844265470676 [-0.3772590489997033, 0.26305113723577167, -0.058537854953111  
(f=-4.928464685493138E-5 [-1.665405949008436E-4, -0.0011850534838405702, -0.001377836

Втората функция е по-трудна, но векторът на решението доближава известния глобален оптимум.



```
155 public Chromosome
156     throws MutationException
157     List<Double> values
158     Double value
159
160     for (int i =
161         values[i]
162     }
163
164     return new Chromosome(
165         ((Double[]) values).clone()
166     )
167 }
168
169 public class GeneticAlgorithm
170     private static final int MAX_ITERATIONS = 1000;
171
172     private static Chromosome bestChromosome;
173     private static double bestFitness;
174     private static int iterations;
175
176     private static Chromosome randomChromosome() {
177         return new Chromosome(
178             new ArrayList<Double>()
179         );
180     }
181     private static Chromosome tournamentSelection() {
182
183     public static void main(String args[]) {
184         GeneticAlgorithm algorithm = new GeneticAlgorithm(
185             new UniformCrossover(), CROSSOVER_RATE,
186             new RandomDoubleMutation(), MUTATION_RATE,
187             new TournamentSelection(TOURNAMENT_AIRITY));
188
189         Function function = new StyblinskiTangFunction();
190         List<Chromosome> list = new ArrayList<Chromosome>();
191     }
```

<terminated> GeneticAlgorithmsExample [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_91.jdk/Contents/Home/bin/java (Mar 31, 2020, 5:46:45 PM)

(f=11.179418109948186 [-0.08196621105339408, -0.4581074082643607, 0.02217101827830592  
(f=391.6614573004389 [-2.904849200869883, -2.9023347641904538, -2.903930085748275, -2



При третата функция резултатите не са толкова добри. Известно е, че минимумът се постига при нула и единичен вектор.



```

155 public Chromosome
156     throws MutationException
157     List<Double> values
158     Double value
159
160     for (int i =
161         values[i]
162     }
163
164     return new Chromosome(
165         ((Double) value)
166     }
167 }

```

```

168
169 public class GeneticAlgorithm
170     private static final
171
172     private static final
173     private static final
174     private static final
175
176     private static final
177     private static final
178
179     private static final
180
181     private static final
182

```

```

183 public static void main(String args[]) {
184     GeneticAlgorithm algorithm = new GeneticAlgorithm(
185         new UniformCrossover(), CROSSOVER_RATE,
186         new RandomDoubleMutation(), MUTATION_RATE,
187         new TournamentSelection(TOURNAMENT_AIRITY));
188
189     Function function = new RosenbrockFunction();
190     List<Chromosome> list = new ArrayList<Chromosome>();
191

```

<terminated> GeneticAlgorithmsExample [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_91.jdk/Contents/Home/bin/java (Mar 31, 2020, 5:50:06 PM)

```

(f=-43.24915517301 [-0.2196698730286616, -0.2135423758217626, 0.17579739948764395, 0.
(f=-6.716927161335633 [0.699844091722202, 0.49674994910880754, 0.25587520120608287, 0

```

Макар и четвъртата функция да е най-нагъната при нея резултатите също са добри. Глобалният оптимум отново е в нулата и се постига при нулев вектор.



```
155 public Chromosome
156     throws MutationException
157     List<Double> values
158     Double value
159
160     for (int i =
161         values[i]
162     }
163
164     return new Chromosome(
165         ((Double) value)
166     )
167 }
168
169 public class GeneticAlgorithm
170     private static final int MAX_ITERATIONS = 1000;
171
172     private static Chromosome bestChromosome;
173     private static double bestFitness;
174     private static int iterations;
175
176     private static int populationSize;
177     private static int tournamentSize;
178
179     private static Chromosome[] population;
180
181     private static int tournamentSelection(Chromosome[] population) {
182
183     public static void main(String args[]) {
184         GeneticAlgorithm algorithm = new GeneticAlgorithm(
185             new UniformCrossover(), CROSSOVER_RATE,
186             new RandomDoubleMutation(), MUTATION_RATE,
187             new TournamentSelection(TOURNAMENT_AIRITY));
188
189         Function function = new RastriginFunction();
190         List<Chromosome> list = new ArrayList<Chromosome>();
191
```

<terminated> GeneticAlgorithmsExample [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0\_91.jdk/Contents/Home/bin/java (Mar 31, 2020, 5:53:03 PM)

(f=-53.14252956269649 [0.24093435075163594, -0.11332181232454741, 0.2512879414089403, -0.012022562565306316 [-0.003123670639210152, 4.998555981909902E-4, -1.49900087682



Кодът на проекта е достъпен в облачната услуга GitHub.



[Pull requests](#)[Issues](#)[Marketplace](#)[Explore](#)[TodorBalabanov /](#)[Watch](#) 0[Star](#) 0[Fork](#) 0

## Apache-Commons-Genetic-Algorithms-for-Simple-Functions

[Code](#)[Issues](#) 0[Pull requests](#) 0[Actions](#)[Projects](#) 0[Wiki](#)[Security](#)[Insights](#)[Settings](#)

### Apache Commons Genetic Algorithms for Simple Functions

[Edit](#)[genetic-algorithm](#)[optimization](#)[multidimensional](#)[Manage topics](#)[3 commits](#)[1 branch](#)[0 packages](#)[0 releases](#)[1 contributor](#)[GPL-3.0](#)Branch: [master](#)[New pull request](#)[Create new file](#)[Upload files](#)[Find file](#)[Clone or download](#)**TodorBalabanov** Few more benchmark functions were added.Latest commit [f097ad6](#) yesterday

<a href="#">gradle/wrapper</a>	Initial version was created.	yesterday
<a href="#">src/main/java</a>	Few more benchmark functions were added.	yesterday
<a href="#">.gitignore</a>	Initial version was created.	yesterday
<a href="#">LICENSE</a>	Initial commit	yesterday
<a href="#">README.md</a>	Initial commit	yesterday
<a href="#">build.gradle</a>	Initial version was created.	yesterday
<a href="#">gradlew</a>	Initial version was created.	yesterday
<a href="#">gradlew.bat</a>	Initial version was created.	yesterday
<a href="#">settings.gradle</a>	Initial version was created.	yesterday

[README.md](#)

Макар и да е изключително полезно, когато се използва готова софтуерна библиотека, такава употреба има и своите недостатъци. Един от основните проблеми с Genetic Algorithms - Apache Commons е трудността да се проследяват изчисленията, тъй като те се случват във вътрешността на библиотеката до която програмистът няма пряк достъп. Използването на голямо количество случайни числа също много възпрепятства процеса по надеждност.

