N-Dimensional Optimization

Generated by Doxygen 1.9.3

Chapter 1

Optimization

This is project that is used for testing optimization algorithms.

The currently implemented algorithms are:

- `Blind Search`
- `Repeated Local Search`
- `Differential Evolution`
- `Particle Swarm Optimization`

1.1 Running the project

Using the Java Virtual Machine (JVM), run the project from the Main file in this project to generate the desired output files.

1.2 Experiment File Format

```
[algorithm] [DE method] [crossover type] [dimension] [population size] [problem type] [range] [num experiments]
```

- The values for `[algorithm]` are 1 for DE, 2 for PSO, 3 for Blind Search, and 4 for Repeated Local Search.
- The values for `[DE method]` are 1 for DE/best/1, 2 for DE/rand/1, 3 for DE/rand-to-best/1, 4 for DE/best/2, and 5 for DE/rand/2.
- The values for `[crossover type]` are 1 for exponential crossover and 2 for binomial crossover.
- The value for `[dimension]` is the number of elements in each solution vector.
- The value for `[population size]` is the number of solution vectors in the population.
- The value for `[problem type]` is the objective function label: 1 - Schwefel, 2 - De Jong 1, 3 - Rosenbrock, 4 - Rastrigin, 5 - Griewank, 6 - Sine Envelope Sine Wave, 7 - Stretch V Sine Wave, 8 - Ackley One, 9 - Ackley Two, 10 - Egg Holder.
- The value for `[range]` is the range of initial values for each element in the solution vector.
- The value for `[num experiments]` is the number of experiments to run.

The experiments.txt file contains a list of all the experiments to run. The experiments are run in the order they appear in the file. The experiments.txt file should be in the same directory as the Main file to properly run the project.

1.3 Output File Format

Generates a CSV where each row is the resulting fitness values from each experiment. The first entry in each row has information on the algorithm run, how many experiments were run, and the time it took to run the experiments.

The output files are named according to the current system time. The output files are placed in the directory of the Main file.

Chapter 2

Namespace Index

2.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 Package mt

Classes

- class MTRandom

6.2 Package project

Classes

- class Algorithms
- class Particle
- class Population
- class Problem

Chapter 7

Class Documentation

7.1 project.Algorithms Class Reference

Public Member Functions

- Algorithms (int algorithm, Population pop, int problem, int crosstype, int method, int index)
- double[ ] getSolution ()
- double getFitness ()
- double[ ] blindSearch (int iterations, double[ ] bestSol, double fitness)
- double[ ] localSearch (double[ ] initialSol, double[ ] bestSol, boolean tau)
- double[ ] repeatedLocalSearch (double[ ] initialSol, double[ ] bestGlobalSol, double[ ] bestIterSol, boolean tau, int iterations)
- double[ ] DE (int method, int D, int NP, double CR, double F, double lambda, int generations)
- Particle PSO (int iterations, int numParticles, int dimensions, double c1, double c2)

7.1.1 Detailed Description

Definition at line 15 of file Algorithms.java.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 Algorithms()

```
project.Algorithms.Algorithms (
            int algorithm,
            Population pop,
            int problem,
            int crosstype,
            int method,
            int index )
```

Constructor for IAlgs

Parameters

algorithm	- algorithm to run
pop	- population
problem	- problem type
crosstype	- crossover type

Definition at line 51 of file Algorithms.java.

7.1.3 Member Function Documentation

7.1.3.1 blindSearch()

```
double[] project.Algorithms.blindSearch (
            int iterations,
            double[] bestSol,
            double fitness )
```

Returns the best solution found by the blind search algorithm

Parameters

iterations	- number of iterations
bestSol	- best solution found
fitness	- fitness of the best solution

Returns

- best solution found

Definition at line 99 of file Algorithms.java.

7.1.3.2 DE()

```
double[] project.Algorithms.DE (
            int method,
            int D,
            int NP,
            double CR,
            double F,
            double lambda,
            int generations )
```

Runs an experiment for the Differential Evolution algorithm

Parameters

method	- mutation method
D	- Dimensions
NP	- Population size
CR	- Crossover rate
F	- Scaling factor
lambda	- Scaling factor
generations	- Number of generations

Returns

- The best fitness vector

Definition at line 220 of file Algorithms.java.

7.1.3.3 getFitness()

```
double project.Algorithms.getFitness ( )
```

Returns

- the fitness of the solution vector

Definition at line 86 of file Algorithms.java.

7.1.3.4 getSolution()

```
double[ ] project.Algorithms.getSolution ( )
```

Returns

- solution vector of the algorithm

Definition at line 79 of file Algorithms.java.

7.1.3.5 localSearch()

```
double[ ] project.Algorithms.localSearch (
            double[ ] initialSol,
            double[ ] bestSol,
            boolean tau )
```

Returns the best solution found by the local search algorithm

Parameters

initialSol	- initial solution
bestSol	- best solution found
tau	- boolean variable to determine if the algorithm should terminate

Returns

- best solution found

Definition at line 130 of file Algorithms.java.

7.1.3.6 PSO()

```
Particle project.Algorithms.PSO (
            int iterations,
            int numParticles,
            int dimensions,
            double c1,
            double c2 )
```

Runs an experiment for the Particle Swarm Optimization algorithm

Parameters

iterations	- Number of iterations
numParticles	- Number of particles
dimensions	- Number of dimensions
c1	- cognitive factor
c2	- social factor

Returns

- The best fitness particle

Definition at line 345 of file Algorithms.java.

7.1.3.7 repeatedLocalSearch()

```
double[] project.Algorithms.repeatedLocalSearch (
            double[] initialSol,
            double[] bestGlobalSol,
            double[] bestIterSol,
            boolean tau,
            int iterations )
```

Returns the best solution found by repeated local search algorithms

Parameters

initialSol	- initial solution
bestGlobalSol	- best overall solution found
bestIterSol	- best solution found in each iteration
tau	- boolean variable to determine if the algorithm should terminate
iterations	- maximum number of iterations

Returns

- best solution found

Definition at line 177 of file Algorithms.java.

The documentation for this class was generated from the following file:

- project/Algorithms.java

7.2 Main Class Reference

Static Public Member Functions

- static void main (String[ ] args)
- static void fileLoop (String line, BufferedReader br, BufferedWriter bw)
- static BufferedReader readFromFile (String filename)
- static BufferedWriter makeOutFile ()
- static void experiment (int n, Population pop, long[ ] times, int algorithm, int problem, int method, int crosstype)
- static void writeFile (BufferedWriter bw, int problem, int n, int m, int alg, double range, long sum, Population pop, int method, int crosstype)

7.2.1 Detailed Description

Definition at line 7 of file Main.java.

7.2.2 Member Function Documentation

7.2.2.1 experiment()

```
static void Main.experiment (
            int n,
            Population pop,
            long[ ] times,
            int algorithm,
            int problem,
            int method,
            int crosstype )  [static]
```

Runs [n] experiments of problem type [problem], and stores fitness values in the population

Parameters

n	- number of experiments
pop	- population to store fitness values
times	- array to store time values
algorithm	- algorithm to use
problem	- problem type
method	- method to use for mutation in DE
crosstype	- crossover type for DE

Definition at line 143 of file Main.java.

7.2.2.2 fileLoop()

```
static void Main.fileLoop (
            String line,
            BufferedReader br,
            BufferedWriter bw )  [static]
```

Runs the experiments for the given line

Parameters

line	- line from input file
br	- BufferedReader for input file
bw	- BufferedWriter for output file

Exceptions

IOException	- if there is an error with the input or output files

algorithm - algorithm to use (1 - DE, 2 - PSO) method - method to use (1 - DE/Best/1, 2 - DE/Rand/1, 3 - DE/Rand-To-Best/1, 4 - DE/Best/2, 5 - DE/Rand/2) crosstype - crossover type (1 - exponential, 2 - binomial) m - dimensions n - population size problem - problem type range - range of values numExperiments - number of experiments to run

Definition at line 45 of file Main.java.

7.2.2.3 main()

```
static void Main.main (
            String[] args )  [static]
```

Reading from the input file and creating the output file

Input file format: [algorithm] [DE method] [crossover type] [dimension] [population size] [problem type] [range] [num experiments]

Definition at line 8 of file Main.java.

7.2.2.4 makeOutFile()

```
static BufferedWriter Main.makeOutFile ( )  [static]
```

Creates a CSV file to write the experiments to

Returns

BufferedWriter of that CSV file

Exceptions

IOException	- if there is an error with the output file

Definition at line 121 of file Main.java.

7.2.2.5 readFromFile()

```
static BufferedReader Main.readFromFile (
             String filename )  [static]
```

Reads input file with given filename and returns a buffered reader for the input file

Parameters

filename	- name of the input file

Returns

buffered reader for the input file

Exceptions

IOException	- if there is an error with the input file

Definition at line 104 of file Main.java.

7.2.2.6 writeFile()

```
static void Main.writeFile (
             BufferedWriter bw,
             int problem,
             int n,
             int m,
```

```
int alg,
double range,
long sum,
Population pop,
int method,
int crosstype )  [static]
```

Save experiments to file

Parameters

bw	- BufferedWriter to write to
problem	- Problem type
n	- Number of experiments
m	- Dimensions
range	- Range of values
sum	- Total time for the experiment
pop	- Population
method	- Method to use for mutation in DE
crosstype	- Crossover type for DE

Exceptions

IOException	- if there is an error with the output file

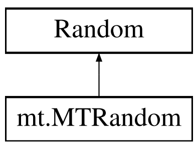
Definition at line 169 of file Main.java.

The documentation for this class was generated from the following file:

- Main.java

7.3 mt.MTRandom Class Reference

Inheritance diagram for mt.MTRandom:



Public Member Functions

- MTRandom ()
- MTRandom (boolean compatible)
- MTRandom (long seed)
- MTRandom (byte[ ] buf)
- MTRandom (int[ ] buf)
- final synchronized void setSeed (long seed)
- final void setSeed (byte[ ] buf)
- final synchronized void setSeed (int[ ] buf)

Static Public Member Functions

- static int[ ] pack (byte[ ] buf)

Protected Member Functions

- final synchronized int next (int bits)

7.3.1 Detailed Description

Version

1.0

Author

David Beaumont, Copyright 2005

A Java implementation of the MT19937 (Mersenne Twister) pseudo random number generator algorithm based upon the original C code by Makoto Matsumoto and Takuji Nishimura (see `http://www.math.sci.↩ hiroshima-u.ac.jp/~m-mat/MT/emt.html` for more information.

As a subclass of java.util.Random this class provides a single canonical method next() for generating bits in the pseudo random number sequence. Anyone using this class should invoke the public inherited methods (next↩ Int(), nextFloat etc.) to obtain values as normal. This class should provide a drop-in replacement for the standard implementation of java.util.Random with the additional advantage of having a far longer period and the ability to use a far larger seed value.

This is **not** a cryptographically strong source of randomness and should **not** be used for cryptographic systems or in any other situation where true random numbers are required.

Definition at line 88 of file MTRandom.java.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 MTRandom() [1/5]

```
mt.MTRandom.MTRandom ( )
```

The default constructor for an instance of MTRandom. This invokes the no-argument constructor for java.util.↩ Random which will result in the class being initialised with a seed value obtained by calling System.currentTime↩ Millis().

Definition at line 127 of file MTRandom.java.

7.3.2.2 MTRandom() [2/5]

```
mt.MTRandom.MTRandom (
            boolean compatible )
```

This version of the constructor can be used to implement identical behaviour to the original C code version of this algorithm including exactly replicating the case where the seed value had not been set prior to calling genrand_int32.

If the compatibility flag is set to true, then the algorithm will be seeded with the same default value as was used in the original C code. Furthermore the setSeed() method, which must take a 64 bit long value, will be limited to using only the lower 32 bits of the seed to facilitate seamless migration of existing C code into Java where identical behaviour is required.

Whilst useful for ensuring backwards compatibility, it is advised that this feature not be used unless specifically required, due to the reduction in strength of the seed value.

Parameters

compatible	Compatibility flag for replicating original behaviour.

Definition at line 150 of file MTRandom.java.

7.3.2.3 MTRandom() [3/5]

```
mt.MTRandom.MTRandom (
            long seed )
```

This version of the constructor simply initialises the class with the given 64 bit seed value. For a better random number sequence this seed value should contain as much entropy as possible.

Parameters

seed	The seed value with which to initialise this class.

Definition at line 163 of file MTRandom.java.

7.3.2.4 MTRandom() [4/5]

```
mt.MTRandom.MTRandom (
            byte[] buf )
```

This version of the constructor initialises the class with the given byte array. All the data will be used to initialise this instance.

Parameters

buf	The non-empty byte array of seed information.

Exceptions

NullPointerException	if the buffer is null.
IllegalArgumentException	if the buffer has zero length.

Definition at line 176 of file MTRandom.java.

7.3.2.5 MTRandom() [5/5]

```
mt.MTRandom.MTRandom (
            int[] buf )
```

This version of the constructor initialises the class with the given integer array. All the data will be used to initialise this instance.

Parameters

buf	The non-empty integer array of seed information.

Exceptions

NullPointerException	if the buffer is null.
IllegalArgumentException	if the buffer has zero length.

Definition at line 190 of file MTRandom.java.

7.3.3 Member Function Documentation

7.3.3.1 next()

```
final synchronized int mt.MTRandom.next (
            int bits )  [protected]
```

This method forms the basis for generating a pseudo random number sequence from this class. If given a value of 32, this method behaves identically to the genrand_int32 function in the original C code and ensures that using the standard nextInt() function (inherited from Random) we are able to replicate behaviour exactly.

Note that where the number of bits requested is not equal to 32 then bits will simply be masked out from the top of the returned integer value. That is to say that:

```
mt.setSeed(12345);
int foo = mt.nextInt(16) + (mt.nextInt(16) << 16);
```

will not give the same result as

```
mt.setSeed(12345);
int foo = mt.nextInt(32);
```

Parameters

bits	The number of significant bits desired in the output.

Returns

The next value in the pseudo random sequence with the specified number of bits in the lower part of the integer.

Definition at line 336 of file MTRandom.java.

7.3.3.2 pack()

```
static int[] mt.MTRandom.pack (
            byte[] buf ) [static]
```

This simply utility method can be used in cases where a byte array of seed data is to be used to repeatedly re-seed the random number sequence. By packing the byte array into an integer array first, using this method, and then invoking setSeed() with that; it removes the need to re-pack the byte array each time setSeed() is called.

If the length of the byte array is not a multiple of 4 then it is implicitly padded with zeros as necessary. For example:

```
byte[] { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06 }
```

becomes

```
int[]  { 0x04030201, 0x00000605 }
```

<p<blockquote>

Note that this method will not complain if the given byte array is empty and will produce an empty integer array, but the setSeed() method will throw an exception if the empty integer array is passed to it.

Parameters

buf	The non-null byte array to be packed.

Returns

A non-null integer array of the packed bytes.

Exceptions

NullPointerException	if the given byte array is null.

Definition at line 407 of file MTRandom.java.

7.3.3.3 setSeed() [1/3]

```
final void mt.MTRandom.setSeed (
            byte[] buf )
```

This method resets the state of this instance using the byte array of seed data provided. Note that calling this method is equivalent to calling "setSeed(pack(buf))" and in particular will result in a new integer array being generated during the call. If you wish to retain this seed data to allow the pseudo random sequence to be restarted then it would be more efficient to use the "pack()" method to convert it into an integer array first and then use that to re-seed the instance. The behaviour of the class will be the same in both cases but it will be more efficient.

Parameters

buf	The non-empty byte array of seed information.

Exceptions

NullPointerException	if the buffer is null.
IllegalArgumentException	if the buffer has zero length.

Definition at line 266 of file MTRandom.java.

7.3.3.4 setSeed() [2/3]

```
final synchronized void mt.MTRandom.setSeed (
            int[] buf )
```

This method resets the state of this instance using the integer array of seed data provided. This is the canonical way of resetting the pseudo random number sequence.

Parameters

buf	The non-empty integer array of seed information.

Exceptions

NullPointerException	if the buffer is null.
IllegalArgumentException	if the buffer has zero length.

Definition at line 279 of file MTRandom.java.

7.3.3.5 setSeed() [3/3]

```
final synchronized void mt.MTRandom.setSeed (
            long seed )
```

This method resets the state of this instance using the 64 bits of seed data provided. Note that if the same seed data is passed to two different instances of MTRandom (both of which share the same compatibility state) then the sequence of numbers generated by both instances will be identical.

If this instance was initialised in 'compatibility' mode then this method will only use the lower 32 bits of any seed value passed in and will match the behaviour of the original C code exactly with respect to state initialisation.

Parameters

seed	The 64 bit value used to initialise the random number generator state.

Definition at line 231 of file MTRandom.java.

The documentation for this class was generated from the following file:

- mt/MTRandom.java

7.4 project.Particle Class Reference

Public Member Functions

- Particle (double[ ] solution, double range, int problem)
- void setPBest (Particle pBest)

Public Attributes

- double[ ] solution
- double velocity
- double fitness
- Particle pBest

7.4.1 Detailed Description

Definition at line 14 of file Particle.java.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 Particle()

```
project.Particle.Particle (
            double[ ] solution,
            double range,
            int problem )
```

Constructor for Particle

Parameters

solution	- solution vector
range	- range of values for the particle
problem	- problem type

Definition at line 34 of file Particle.java.

7.4.3 Member Function Documentation

7.4.3.1 setPBest()

```
void project.Particle.setPBest (
            Particle pBest )
```

Sets the pBest particle

Parameters

pBest	- the new pBest particle

Definition at line 49 of file Particle.java.

7.4.4 Member Data Documentation

7.4.4.1 fitness

```
double project.Particle.fitness
```

Definition at line 22 of file Particle.java.

7.4.4.2 pBest

```
Particle project.Particle.pBest
```

Definition at line 25 of file Particle.java.

7.4.4.3 solution

```
double [] project.Particle.solution
```

Definition at line 16 of file Particle.java.

7.4.4.4 velocity

```
double project.Particle.velocity
```

Definition at line 19 of file Particle.java.

The documentation for this class was generated from the following file:

- project/Particle.java

7.5 project.Population Class Reference

Public Member Functions

- Population (int n, int m, double range)
- double[ ] getFitness ()
- void setFitness (int i, double fitness)
- void setSolution (int i, double[ ] sol)
- double[ ][ ] getPopulation ()
- double[ ][ ] getSolution ()
- void setPopulation (double[ ][ ] population)
- double getRange ()
- double[ ][ ] genRandomMatrix (int n, int m)
- double[ ] genRandomArray (int m)
- double[ ][ ] genNeighborhood (int n, int m, double[ ] solution)

7.5.1 Detailed Description

Definition at line 14 of file Population.java.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 Population()

```
project.Population.Population (
            int n,
            int m,
            double range )
```

Constructor for the Population class.

Parameters

n	- number of experiments
m	- number of dimensions
range	- range of the values selected **[-range, range]**

Definition at line 35 of file Population.java.

7.5.3 Member Function Documentation

7.5.3.1 genNeighborhood()

```
double[][] project.Population.genNeighborhood (
            int n,
            int m,
            double[] solution )
```

Returns a neighbor of the given solution vector

Parameters

n	- number of experiments
m	- number of dimensions
solution	- solution vector

Returns

- the neighborhood of the given solution vector

Definition at line 148 of file Population.java.

7.5.3.2 genRandomArray()

```
double[] project.Population.genRandomArray (
            int m )
```

Returns an array of random values within the range

Parameters

m	- dimension of the array

Returns

- array of random values

Definition at line 131 of file Population.java.

7.5.3.3 genRandomMatrix()

```
double[][] project.Population.genRandomMatrix (
        int n,
        int m )
```

Creates an n x m matrix initialized to pseudo-random values

Parameters

n	- number of experiments
m	- number of dimensions

Returns

- n x m matrix

Definition at line 114 of file Population.java.

7.5.3.4 getFitness()

```
double[] project.Population.getFitness ( )
```

Returns the fitness vector

Returns

- fitnesses of the population

Definition at line 47 of file Population.java.

7.5.3.5 getPopulation()

```
double[][] project.Population.getPopulation ( )
```

Returns the population matrix

Returns

- population matrix

Definition at line 76 of file Population.java.

7.5.3.6 getRange()

```
double project.Population.getRange ( )
```

Returns the range of the initial population values

Returns

- range

Definition at line 103 of file Population.java.

7.5.3.7 getSolution()

```
double[][] project.Population.getSolution ( )
```

Returns the solutions matrix

Returns

- solutions matrix

Definition at line 85 of file Population.java.

7.5.3.8 setFitness()

```
void project.Population.setFitness (
            int i,
            double fitness )
```

Sets the value of the fitness at the index

Parameters

i	- index
fitness	- fitness value

Definition at line 57 of file Population.java.

7.5.3.9 setPopulation()

```
void project.Population.setPopulation (
            double population[][] )
```

Sets the population matrix to the given matrix

Parameters

population	- given population matrix

Definition at line 94 of file Population.java.

7.5.3.10 setSolution()

```
void project.Population.setSolution (
            int i,
            double[ ] sol )
```

Sets the values of the solution vector at the index

Parameters

i	- index
sol	- solution vector

Definition at line 67 of file Population.java.

The documentation for this class was generated from the following file:

- project/Population.java

7.6 project.Problem Class Reference

Public Member Functions

- Problem (double[ ] vector, int probNum)
- double getFitness ()
- double nthRoot (int root, double value)
- double square (double value)
- double schwefel ()
- double de_jong_1 ()
- double rosenbrock ()
- double rastrigin ()
- double griewank ()
- double sine_envelope ()
- double sine_V ()
- double ackley_one ()
- double ackley_two ()
- double egg_holder ()

7.6.1 Detailed Description

Definition at line 12 of file Problem.java.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 Problem()

```
project.Problem.Problem (
            double[] vector,
            int probNum )
```

Constructor for Problem

Parameters

vector	- vector of values
probNum	- problem type

Definition at line 25 of file Problem.java.

7.6.3 Member Function Documentation

7.6.3.1 ackley_one()

```
double project.Problem.ackley_one ( )
```

Returns the value of the Ackley One function using the values in the vector

Returns

- returns the fitness value of the Ackley One function

Definition at line 194 of file Problem.java.

7.6.3.2 ackley_two()

```
double project.Problem.ackley_two ( )
```

Returns the value of the Ackley Two function using the values in the vector

Returns

- returns the fitness value of the Ackley Two function

Definition at line 209 of file Problem.java.

7.6.3.3 de_jong_1()

`double project.Problem.de_jong_1 ( )`

Returns the value of the De Jong 1 function using the values in the vector

Returns

- returns the fitness value of the De Jong 1 function

Definition at line 109 of file Problem.java.

7.6.3.4 egg_holder()

`double project.Problem.egg_holder ( )`

Returns the value of the Egg Holder function using the values in the vector

Returns

- returns the fitness value of the Egg Holder function

Definition at line 226 of file Problem.java.

7.6.3.5 getFitness()

`double project.Problem.getFitness ( )`

Returns the fitness value of the problem

Returns

- returns the fitness value of the problem

Definition at line 66 of file Problem.java.

7.6.3.6 griewank()

`double project.Problem.griewank ( )`

Returns the value of the Griewank function using the values in the vector

Returns

- returns the fitness value of the Griewank function

Definition at line 148 of file Problem.java.

7.6.3.7 nthRoot()

```
double project.Problem.nthRoot (
            int root,
            double value )
```

Returns the nth root of the input value

Parameters

root	- the root to use
value	- the value to use

Returns

- returns decimal form of the nth root of the input value

Definition at line 77 of file Problem.java.

7.6.3.8 rastrigin()

`double project.Problem.rastrigin ( )`

Returns the value of the Rastrigin function using the values in the vector

Returns

- returns the fitness value of the Rastrigin function

Definition at line 135 of file Problem.java.

7.6.3.9 rosenbrock()

`double project.Problem.rosenbrock ( )`

Returns the value of the Rosenbrock function using the values in the vector

Returns

- returns the fitness value of the Rosenbrock function

Definition at line 122 of file Problem.java.

7.6.3.10 schwefel()

`double project.Problem.schwefel ( )`

Returns the value of the Schwefel function using the values in the vector

Returns

- returns the fitness value of the Schwefel function

Definition at line 96 of file Problem.java.

7.6.3.11 sine_envelope()

```
double project.Problem.sine_envelope ( )
```

Returns the value of the Sine Envelope function using the values in the vector

Returns

- returns the fitness value of the Sine Envelope function

Definition at line 164 of file Problem.java.

7.6.3.12 sine_V()

```
double project.Problem.sine_V ( )
```

Returns the value of the Sine V function using the values in the vector

Returns

- returns the fitness value of the Sine V function

Definition at line 179 of file Problem.java.

7.6.3.13 square()

```
double project.Problem.square (
              double value )
```

Returns the square of the input value

Parameters

value	- the value to use

Returns

- returns the decimal form of the square of the input value

Definition at line 87 of file Problem.java.

The documentation for this class was generated from the following file:

- project/Problem.java

Chapter 8

File Documentation

8.1 Main.java File Reference

Classes

- class Main

8.2 Main.java

Go to the documentation of this file.

```java
import project.Population;
import project.Algorithms;

import java.io.*;
import java.util.stream.LongStream;

public class Main {
    public static void main(String[] args) {
        try {
            BufferedReader br = readFromFile("experiments.txt");
            BufferedWriter bw = makeOutFile();

            // Reads line from input file
            String line = br.readLine();

            // Main loop for conducting experiments
            fileLoop(line, br, bw);

            // Close input and output files when done
            bw.close();
            br.close();

        } catch (IOException e) {

            // If there is an error, print it
            e.printStackTrace();
        }
    }

    public static void fileLoop(String line, BufferedReader br, BufferedWriter bw) {
        try {
            // Loop until end of input file
            while (line != null) {

                // Get experiment parameters
                String[] parts = line.split(" ");

                int algorithm = Integer.parseInt(parts[0]);
                int method = Integer.parseInt(parts[1]);
                int crosstype = Integer.parseInt(parts[2]);
                int m = Integer.parseInt(parts[3]);
```

```
                int n = Integer.parseInt(parts[4]);
                int problem = Integer.parseInt(parts[5]);
                double range = Double.parseDouble(parts[6]);
                int numExperiments = Integer.parseInt(parts[7]);

                // Initialize population
                Population pop = new Population(n, m, range);
                long[] times = new long[n];

                // Run the experiments
                experiment(numExperiments, pop, times, algorithm, problem, method, crosstype);

                // Sum time array for total time for the experiment
                long sum = LongStream.of(times).sum();

                // Save experiment in a CSV file
                writeFile(bw, problem, numExperiments, m, algorithm, range, sum, pop, method, crosstype);

                // Read next line from input file
                line = br.readLine();
            }
        } catch (IOException e) {

            // If there is an error, print it
            e.printStackTrace();
        }
    }

    public static BufferedReader readFromFile(String filename) {
        BufferedReader br = null;
        try {
            File experiments = new File(filename);
            br = new BufferedReader(new FileReader(experiments));
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        return br;
    }

    public static BufferedWriter makeOutFile() {
        BufferedWriter bw = null;
        try {
            bw = new BufferedWriter(new FileWriter("experiments" + System.currentTimeMillis() + ".csv"));
        } catch (IOException e) {
            e.printStackTrace();
        }
        return bw;
    }

    public static void experiment(int n, Population pop, long[] times, int algorithm, int problem, int method,
            int crosstype) {
        for (int i = 0; i < n; i++) {
            long start = System.nanoTime();
            Algorithms alg = new Algorithms(algorithm, pop, problem, crosstype, method, i);
            System.out.println(i);
            times[i] = System.nanoTime() - start;
            pop.setFitness(i, alg.getFitness());
            pop.setSolution(i, alg.getSolution());
        }
    }

    public static void writeFile(BufferedWriter bw, int problem, int n, int m, int alg, double range, long sum,
            Population pop, int method, int crosstype) {
        try {
            // Gets the algorithm type
            String algType = (alg == 1) ? "" : "Partical Swarm Optimization";
            String meth = "";
            // Gets the method type for DE
            switch (method) {
                case 1:
                    meth = "DE/Best/1";
                    break;
                case 2:
                    meth = "DE/Rand/1";
                    break;
                case 3:
                    meth = "DE/Rand-To-Best/1";
                    break;
                case 4:
                    meth = "DE/Best/2";
                    break;
                case 5:
                    meth = "DE/Rand/2";
```

```java
                    break;
                default:
                    break;
            }
            // Gets the crossover type for DE
            String cross = (crosstype == 1) ? "exp" : (crosstype == 2) ? "bin" : "";

            // Writes the summary of the experiments
            bw.write("Problem " + problem + " with " + n + " experiments of dimension " + m + " in range [-" + range
                    + " : " + range + "]"
                    + " using the " + algType + meth + cross + " algorithm that took " + +(double) sum / 1000000
                    + "milliseconds to run" + ",");

            // Writes the fitness values of the population
            for (int i = 0; i < n; i++) {
                bw.write(pop.getFitness()[i] + ",");
            }
            bw.write("\n");

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

8.3 mt/MTRandom.java File Reference

Classes

- class mt.MTRandom

Packages

- package mt

8.4 MTRandom.java

Go to the documentation of this file.

```java
/*
 * MTRandom : A Java implementation of the MT19937 (Mersenne Twister)
 *            pseudo random number generator algorithm based upon the
 *            original C code by Makoto Matsumoto and Takuji Nishimura.
 * Author   : David Beaumont
 * Email    : mersenne-at-www.goui.net
 *
 * For the original C code, see:
 *     http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
 *
 * This version, Copyright (C) 2005, David Beaumont.
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
 */
package mt;

import java.util.Random;
```

```
public class MTRandom extends Random {

    private static final long serialVersionUID = -515082678588212038L;

    // Constants used in the original C implementation
    private final static int UPPER_MASK = 0x80000000;
    private final static int LOWER_MASK = 0x7fffffff;

    private final static int N = 624;
    private final static int M = 397;
    private final static int MAGIC[] = { 0x0, 0x9908b0df };
    private final static int MAGIC_FACTOR1 = 1812433253;
    private final static int MAGIC_FACTOR2 = 1664525;
    private final static int MAGIC_FACTOR3 = 1566083941;
    private final static int MAGIC_MASK1 = 0x9d2c5680;
    private final static int MAGIC_MASK2 = 0xefc60000;
    private final static int MAGIC_SEED = 19650218;
    private final static long DEFAULT_SEED = 5489L;

    // Internal state
    private transient int[] mt;
    private transient int mti;
    private transient boolean compat = false;

    // Temporary buffer used during setSeed(long)
    private transient int[] ibuf;

    public MTRandom() {
    }

    public MTRandom(boolean compatible) {
        super(0L);
        compat = compatible;
        setSeed(compat ? DEFAULT_SEED : System.currentTimeMillis());
    }

    public MTRandom(long seed) {
        super(seed);
    }

    public MTRandom(byte[] buf) {
        super(0L);
        setSeed(buf);
    }

    public MTRandom(int[] buf) {
        super(0L);
        setSeed(buf);
    }

    // Initializes mt[N] with a simple integer seed. This method is
    // required as part of the Mersenne Twister algorithm but need
    // not be made public.
    private final void setSeed(int seed) {

        // Annoying runtime check for initialisation of internal data
        // caused by java.util.Random invoking setSeed() during init.
        // This is unavoidable because no fields in our instance will
        // have been initialised at this point, not even if the code
        // were placed at the declaration of the member variable.
        if (mt == null)
            mt = new int[N];

        // ---- Begin Mersenne Twister Algorithm ----
        mt[0] = seed;
        for (mti = 1; mti < N; mti++) {
            mt[mti] = (MAGIC_FACTOR1 * (mt[mti - 1] ^ (mt[mti - 1] »> 30)) + mti);
        }
        // ---- End Mersenne Twister Algorithm ----
    }

    public final synchronized void setSeed(long seed) {
        if (compat) {
            setSeed((int) seed);
        } else {

            // Annoying runtime check for initialisation of internal data
            // caused by java.util.Random invoking setSeed() during init.
            // This is unavoidable because no fields in our instance will
            // have been initialised at this point, not even if the code
            // were placed at the declaration of the member variable.
            if (ibuf == null)
                ibuf = new int[2];

            ibuf[0] = (int) seed;
            ibuf[1] = (int) (seed »> 32);
```

```java
            setSeed(ibuf);
        }
    }

    public final void setSeed(byte[] buf) {
        setSeed(pack(buf));
    }

    public final synchronized void setSeed(int[] buf) {
        int length = buf.length;
        if (length == 0)
            throw new IllegalArgumentException("Seed buffer may not be empty");
        // ---- Begin Mersenne Twister Algorithm ----
        int i = 1, j = 0, k = (N > length ? N : length);
        setSeed(MAGIC_SEED);
        for (; k > 0; k--) {
            mt[i] = (mt[i] ^ ((mt[i - 1] ^ (mt[i - 1] »> 30)) * MAGIC_FACTOR2)) + buf[j] + j;
            i++;
            j++;
            if (i >= N) {
                mt[0] = mt[N - 1];
                i = 1;
            }
            if (j >= length)
                j = 0;
        }
        for (k = N - 1; k > 0; k--) {
            mt[i] = (mt[i] ^ ((mt[i - 1] ^ (mt[i - 1] »> 30)) * MAGIC_FACTOR3)) - i;
            i++;
            if (i >= N) {
                mt[0] = mt[N - 1];
                i = 1;
            }
        }
        mt[0] = UPPER_MASK; // MSB is 1; assuring non-zero initial array
        // ---- End Mersenne Twister Algorithm ----
    }

    protected final synchronized int next(int bits) {
        // ---- Begin Mersenne Twister Algorithm ----
        int y, kk;
        if (mti >= N) { // generate N words at one time

            // In the original C implementation, mti is checked here
            // to determine if initialisation has occurred; if not
            // it initialises this instance with DEFAULT_SEED (5489).
            // This is no longer necessary as initialisation of the
            // Java instance must result in initialisation occurring
            // Use the constructor MTRandom(true) to enable backwards
            // compatible behaviour.

            for (kk = 0; kk < N - M; kk++) {
                y = (mt[kk] & UPPER_MASK) | (mt[kk + 1] & LOWER_MASK);
                mt[kk] = mt[kk + M] ^ (y »> 1) ^ MAGIC[y & 0x1];
            }
            for (; kk < N - 1; kk++) {
                y = (mt[kk] & UPPER_MASK) | (mt[kk + 1] & LOWER_MASK);
                mt[kk] = mt[kk + (M - N)] ^ (y »> 1) ^ MAGIC[y & 0x1];
            }
            y = (mt[N - 1] & UPPER_MASK) | (mt[0] & LOWER_MASK);
            mt[N - 1] = mt[M - 1] ^ (y »> 1) ^ MAGIC[y & 0x1];

            mti = 0;
        }

        y = mt[mti++];

        // Tempering
        y ^= (y »> 11);
        y ^= (y « 7) & MAGIC_MASK1;
        y ^= (y « 15) & MAGIC_MASK2;
        y ^= (y »> 18);
        // ---- End Mersenne Twister Algorithm ----
        return (y »> (32 - bits));
    }

    // This is a fairly obscure little code section to pack a
    // byte[] into an int[] in little endian ordering.

    public static int[] pack(byte[] buf) {
        int k, blen = buf.length, ilen = ((buf.length + 3) »> 2);
        int[] ibuf = new int[ilen];
        for (int n = 0; n < ilen; n++) {
            int m = (n + 1) « 2;
            if (m > blen)
                m = blen;
            for (k = buf[--m] & 0xff; (m & 0x3) != 0; k = (k « 8) | buf[--m] & 0xff)
```

```
                ;
            ibuf[n] = k;
        }
        return ibuf;
    }
}
```

8.5 project/Algorithms.java File Reference

Classes

- class project.Algorithms

Packages

- package project

8.6 Algorithms.java

Go to the documentation of this file.

```
/*
 * Algorithms : A Java implementation of optimization algorithms: Blind Search,
 *              Iterated Local Search, Differential Evolution, Particle Swarm
 *              Optimization.
 * Author     : Ethan Krug
 * Email      : ethan.c.krug@gmail.com
 * Date       : May, 2022
 *
 * Copyright (C) 2022 Ethan Krug
 */
package project;

import mt.MTRandom;

public class Algorithms {
    // The population of the algorithm
    private Population population;

    // The number of dimensions
    private int m;

    // The number of experiments/iterations
    private int n;

    // The solution vectors in the population
    private double[][] popMatrix;

    // The problem type of the algorithm
    private int problem;

    // The crossover method for the algorithm
    private int crosstype;

    // The solution vector of the algorithm
    private double[] solution;

    // Intialization of the fitness for the current algorithm
    private double fitness = Double.MAX_VALUE;

    // Random number generator
    private MTRandom r;

    public Algorithms(int algorithm, Population pop, int problem, int crosstype, int method, int
  index) {
        this.population = pop;
        this.m = pop.getPopulation()[0].length;
        this.n = pop.getPopulation().length;
        popMatrix = pop.getPopulation();
        this.problem = problem;
        this.crosstype = crosstype;
```

```
        this.r = new MTRandom(false);
        switch (algorithm) {
            case 1:
                this.solution = DE(method, popMatrix[0].length, popMatrix.length, 0.6, 0.9, 0.8, 100);
                break;
            case 2:
                Particle p = PSO(100, popMatrix.length, popMatrix[0].length, 0.8, 1.2);
                this.solution = p.solution;
                break;
            case 3:
                this.solution = blindSearch(n, pop.getPopulation()[index], fitness);
                break;
            case 4:
                this.solution = repeatedLocalSearch(pop.getPopulation()[index], null, null, false, n);
                break;
        }
    }

    public double[] getSolution() {
        return solution;
    }

    public double getFitness() {
        Problem p = new Problem(solution, problem);
        return p.getFitness();
    }

    public double[] blindSearch(int iterations, double[] bestSol, double fitness) {
        for (int i = 0; i < iterations; i++) {
            // Gets a random solution
            double[] arg = population.genRandomArray(m);

            // Evaluate the fitness of the random solution
            Problem prob = new Problem(arg, problem);

            // Get the fitness of the solution
            double fitnessNew = prob.getFitness();

            // If the fitness is better than the current best solution,
            // update the best solution
            if (fitnessNew < fitness) {
                fitness = fitnessNew;
                bestSol = arg;
            }
        }
        this.fitness = fitness;
        return bestSol;
    }

    public double[] localSearch(double[] initialSol, double[] bestSol, boolean tau) {
        // Run algorithm until the solution doesn't improve
        while (tau) {
            tau = false;

            // Generate the neighborhood of the current solution
            double[][] neighborhood = population.genNeighborhood(n, m, initialSol);

            // Initialize best neighborhood fitness and solution
            double bestFitnessInNeighborhood = Double.MAX_VALUE;
            double[] bestSolutionInNeighborhood = new double[m];

            // Evaluate the fitness of the neighborhood
            for (int i = 0; i < neighborhood.length; i++) {
                Problem prob = new Problem(neighborhood[i], problem);
                double fitnessNew = prob.getFitness();

                // If the fitness is better than the current best solution in neighborhood,
                // update the best solution in neighborhood
                if (fitnessNew <= this.fitness) {
                    bestFitnessInNeighborhood = fitnessNew;
                    bestSolutionInNeighborhood = neighborhood[i];
                }
            }

            // If the best solution in the neighborhood is better than the current best
            // solution, update the best solution
            if (bestFitnessInNeighborhood < this.fitness) {
                this.fitness = bestFitnessInNeighborhood;
                bestSol = bestSolutionInNeighborhood;
                tau = true;
            }
        }
        return bestSol;
    }

    public double[] repeatedLocalSearch(double[] initialSol, double[] bestGlobalSol, double[]
  bestIterSol, boolean tau,
```

```java
            int iterations) {

        // Initialize best solutions
        bestGlobalSol = initialSol;
        bestIterSol = initialSol;
        tau = true;

        // Iteration counter
        int t = 1;

        // Run the algorithm until all iterations are complete
        while (t <= iterations) {
            // Get the local search solution for current iteration
            bestIterSol = localSearch(bestGlobalSol, bestIterSol, tau);

            // Get fitness of current iteration and global iteration
            Problem prob = new Problem(bestIterSol, problem);
            Problem probGlobal = new Problem(bestGlobalSol, problem);

            // If the fitness of the current iteration is better than the global
            // solution, update the global solution
            if (prob.getFitness() < probGlobal.getFitness()) {
                bestGlobalSol = bestIterSol;
            }
            t++;
            bestIterSol = population.genRandomArray(m);
        }
        return bestGlobalSol;
    }

    public double[] DE(int method, int D, int NP, double CR, double F, double lambda, int generations) {
        int generation = 0;
        while (generation < generations) {
            // Iterate over every solution in the population
            for (int i = 0; i < NP; i++) {

                // Getting the randomly selected vector indexes
                int r1 = i, r2 = i, r3 = i, r4 = i, r5 = i, jrand = r.nextInt(D);
                while (same(i, r1, r2, r3, r4, r5)) {
                    r1 = r.nextInt(NP);
                    r2 = r.nextInt(NP);
                    r3 = r.nextInt(NP);
                    r4 = r.nextInt(NP);
                    r5 = r.nextInt(NP);
                }

                // noisy vector
                double[] u = new double[D];

                // Mutation of noisy vector
                boolean crossed = false;
                while (!crossed) {
                    for (int k = 0; k < D; k++) {
                        if (r.nextDouble() < CR || k == jrand) {
                            u[k] = method(method, lambda, F, r1, r2, r3, r4, r5, k, i);
                            crossed = true;
                        } else {
                            u[k] = popMatrix[i][k];
                        }
                    }
                    if (crosstype == 2) {
                        crossed = true;
                    }
                }

                // Selection
                Problem p = new Problem(u, problem);
                Problem x = new Problem(popMatrix[i], problem);
                if (Math.abs(p.getFitness()) <= Math.abs(x.getFitness())) {
                    popMatrix[i] = u;
                }
            }
            generation++;
        }
        return popMatrix[bestSol()];
    }

    private boolean same(int i, int r1, int r2, int r3, int r4, int r5) {
        return i == r1 || i == r2 || i == r3 || i == r4 || i == r5 || r1 == r2 || r1 == r3 || r1 == r4
                || r1 == r5 || r2 == r3 || r2 == r4 || r2 == r5 || r3 == r4 || r3 == r5 || r4 == r5;
    }

    private double method(int method, double lambda, double F, int r1, int r2, int r3, int r4, int r5, int k, int i) {
        switch (method) {
            case 1: // DE/best/1
```

```
                return popMatrix[bestSol()][k] + F * (popMatrix[r1][k] - popMatrix[r2][k]);
            case 2: // DE/rand/1
                return popMatrix[r1][k] + F * (popMatrix[r2][k] - popMatrix[r3][k]);
            case 3: // DE/rand-to-best/1
                return popMatrix[i][k] + lambda * (popMatrix[bestSol()][k] - popMatrix[i][k])
                        + F * (popMatrix[r1][k] - popMatrix[r2][k]);
            case 4: // DE/best/2
                return popMatrix[bestSol()][k]
                        + F * (popMatrix[r1][k] + popMatrix[r2][k] - popMatrix[r3][k] - popMatrix[r4][k]);
            case 5: // DE/rand/2
                return popMatrix[r5][k]
                        + F * (popMatrix[r1][k] + popMatrix[r2][k] - popMatrix[r3][k] - popMatrix[r4][k]);
            default: // Invalid method
                return Double.NaN;
        }
    }

    private int bestSol() {
        int best = 0;
        for (int i = 0; i < popMatrix.length; i++) {
            Problem p = new Problem(popMatrix[i], problem);
            Problem b = new Problem(popMatrix[best], problem);
            if (Math.abs(p.getFitness()) <= Math.abs(b.getFitness())) {
                best = i;
            }
        }
        return best;
    }

    public Particle PSO(int iterations, int numParticles, int dimensions, double c1, double c2) {
        // Get range of values
        double range = population.getRange();

        // Initialize the particles
        Particle[] particles = new Particle[numParticles];
        for (int i = 0; i < numParticles; i++) {
            particles[i] = new Particle(popMatrix[i], range, problem);
            particles[i].setPBest(particles[i]);
        }

        // Initialize the best particle
        Particle gBest = particles[0];
        for (int i = 1; i < numParticles; i++) {
            if (Math.abs(particles[i].fitness) < gBest.fitness) {
                gBest = particles[i];
            }
        }

        // Runs the main part of the PSO algorithm
        for (int t = 0; t < iterations; t++) {
            // Update the states of every particle in the swarm
            for (int j = 0; j < numParticles; j++) {
                // Update the velocity and position of the particles
                for (int k = 0; k < dimensions; k++) {
                    // Calculate the new velocity
                    double addVel = particles[j].velocity
                            + c1 * r.nextDouble() * (particles[j].pBest.solution[k] - particles[j].solution[k])
                            + c2 * r.nextDouble() * (gBest.solution[k] - particles[j].solution[k]);
                    // Add the new velocity to the current position
                    particles[j].solution[k] += addVel;
                }

                // Update the pBest and fitness of the particle
                Problem p = new Problem(particles[j].solution, problem);
                if (Math.abs(p.getFitness()) < Math.abs(particles[j].fitness)) {
                    particles[j].pBest = particles[j];
                }
                particles[j].fitness = p.getFitness();

                // Update the gBest
                if (Math.abs(particles[j].fitness) < gBest.fitness) {
                    gBest = particles[j];
                }
            }
        }
        return gBest;
    }
}
```

8.7 project/Particle.java File Reference

Classes

- class project.Particle

Packages

- package project

8.8 Particle.java

Go to the documentation of this file.

```java
/*
 * Particle : A particle object implemented for the Particle Swarm Optimization
 *             algorithm.
 * Author    : Ethan Krug
 * Email     : ethan.c.krug@gmail.com
 * Date      : May, 2022
 *
 * Copyright (C) 2022 Ethan Krug
 */
package project;

import mt.MTRandom;

public class Particle {
    // The current position of the particle
    public double[] solution;

    // The current velocity of the particle
    public double velocity;

    // The fitness of the particle
    public double fitness;

    // The best position the particle has ever achieved
    public Particle pBest;

    public Particle(double[] solution, double range, int problem) {
        this.solution = solution;
        this.fitness = new Problem(solution, problem).getFitness();
        this.pBest = null;
        MTRandom r = new MTRandom();
        // Upper bound is [range] and lower bound is [-range], so 50% of
        // [U - L] = [range]
        this.velocity = r.nextDouble() * range;
    }

    public void setPBest(Particle pBest) {
        this.pBest = pBest;
    }
}
```

8.9 project/Population.java File Reference

Classes

- class project.Population

Packages

- package project

8.10 Population.java

Go to the documentation of this file.

```java
/*
 * Population : Population object implemented as a control structure to assist
 *              in running the optimization algorithms.
 * Author   : Ethan Krug
 * Email    : ethan.c.krug@gmail.com
 * Date     : May, 2022
 *
 * Copyright (C) 2022 Ethan Krug
 */
package project;

import mt.MTRandom;

public class Population {

    // Holds the generated values for each experiment
    private double[][] population;

    // Holds the fitness values of each experiment
    private double[] fitness;

    // Holds the solution vectors
    private double[][] solutions;

    // Range of possible values
    private double range;

    public Population(int n, int m, double range) {
        this.range = range;
        fitness = new double[n];
        solutions = new double[n][m];
        population = genRandomMatrix(n, m);
    }

    public double[] getFitness() {
        return fitness;
    }

    public void setFitness(int i, double fitness) {
        this.fitness[i] = fitness;
    }

    public void setSolution(int i, double[] sol) {
        solutions[i] = sol;
    }

    public double[][] getPopulation() {
        return population;
    }

    public double[][] getSolution() {
        return solutions;
    }

    public void setPopulation(double[][] population) {
        this.population = population;
    }

    public double getRange() {
        return range;
    }

    public double[][] genRandomMatrix(int n, int m) {
        double[][] matrix = new double[n][m];
        MTRandom r = new MTRandom(false);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                matrix[i][j] = r.nextDouble() * (range - (-range)) + (-range);
            }
        }
        return matrix;
    }

    public double[] genRandomArray(int m) {
        double[] array = new double[m];
        MTRandom r = new MTRandom(false);
        for (int i = 0; i < m; i++) {
            array[i] = r.nextDouble() * (range - (-range)) + (-range);
        }
        return array;
    }

```

```
    public double[][] genNeighborhood(int n, int m, double[] solution) {
        double[][] neighborhood = new double[n][m];
        MTRandom r = new MTRandom(false);

        // Fill a neighborhood with pseudo-random values based off of the solution
        // vector within a given range
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                double value = solution[j] + r.nextDouble() * (range - (-range)) + (-range);
                if (value > range) {
                    value = range;
                } else if (value < -range) {
                    value = -range;
                }

                neighborhood[i][j] = value;
            }
        }
        return neighborhood;
    }
}
```

8.11 project/Problem.java File Reference

Classes

- class project.Problem

Packages

- package project

8.12 Problem.java

Go to the documentation of this file.

```
/*
 * Problem : A Java implementation of ten mathematical functions typically
 *           used in testing optimization algorithms.
 * Author   : Ethan Krug
 * Email    : ethan.c.krug@gmail.com
 * Date     : May, 2022
 *
 * Copyright (C) 2022 Ethan Krug
 */
package project;

public class Problem {
    // Holds the generated values passed to the constructor
    private double[] values;

    // Holds the fitness value of values based on the problem type
    private double fitness;

    public Problem(double[] vector, int probNum) {
        values = vector;
        switch (probNum) {
            case 1:
                fitness = schwefel();
                break;
            case 2:
                fitness = de_jong_1();
                break;
            case 3:
                fitness = rosenbrock();
                break;
            case 4:
                fitness = rastrigin();
                break;
            case 5:
                fitness = griewank();
```

```
                break;
            case 6:
                fitness = sine_envelope();
                break;
            case 7:
                fitness = sine_V();
                break;
            case 8:
                fitness = ackley_one();
                break;
            case 9:
                fitness = ackley_two();
                break;
            case 10:
                fitness = egg_holder();
                break;
        }
    }

    public double getFitness() {
        return fitness;
    }

    public double nthRoot(int root, double value) {
        return Math.pow(value, 1.0 / root);
    }

    public double square(double value) {
        return value * value;
    }

    public double schwefel() {
        double sum = 0;
        for (int i = 0; i < values.length; i++) {
            sum += -values[i] * Math.sin(Math.sqrt(Math.abs(values[i])));
        }
        return 418.9829 * values.length - sum;
    }

    public double de_jong_1() {
        double sum = 0;
        for (int i = 0; i < values.length; i++) {
            sum += square(values[i]);
        }
        return sum;
    }

    public double rosenbrock() {
        double sum = 0;
        for (int i = 0; i < values.length - 1; i++) {
            sum += 100 * square((square(values[i]) - values[i + 1])) + square((1 - values[i]));
        }
        return sum;
    }

    public double rastrigin() {
        double sum = 0;
        for (int i = 0; i < values.length; i++) {
            sum += square(values[i]) - 10 * Math.cos(2 * Math.PI * values[i]);
        }
        return 10 * values.length + sum;
    }

    public double griewank() {
        double sum = 0;
        double prod = 1;
        for (int i = 0; i < values.length; i++) {
            sum += square(values[i]);
            prod *= Math.cos(values[i] / Math.sqrt(i + 1));
        }
        return sum / 4000 - prod + 1;
    }

    public double sine_envelope() {
        double sum = 0;
        for (int i = 0; i < values.length - 1; i++) {
            double top = square(Math.sin(square(values[i]) + square(values[i + 1]) - 0.5));
            double bottom = square(1 + 0.001 * (square(values[i]) + square(values[i + 1])));
            sum += 0.5 + top / bottom;
        }
        return -sum;
    }

    public double sine_V() {
        double sum = 0;
        for (int i = 0; i < values.length - 1; i++) {
            double first = nthRoot(4, square(values[i]) + square(values[i + 1]));
```

```
            double second = square(Math.sin(50 * nthRoot(10, square(values[i]) + square(values[i +
    1])))));
            sum += first * second + 1;
        }
        return sum;
    }

    public double ackley_one() {
        double sum = 0;
        for (int i = 0; i < values.length - 1; i++) {
            double first = Math.pow(Math.E, -0.2) * Math.sqrt(square(values[i]) + square(values[i +
    1]));
            double second = 3 * (Math.cos(2 * values[i]) + Math.sin(2 * values[i + 1]));
            sum += first + second;
        }
        return sum;
    }

    public double ackley_two() {
        double sum = 0;
        for (int i = 0; i < values.length - 1; i++) {
            double first = Math.pow(Math.E,
                    0.2 * Math.sqrt((square(values[i]) + square(values[i + 1])) / 2));
            double second = Math.pow(Math.E,
                    0.5 * (Math.cos(2 * Math.PI * values[i]) + Math.cos(2 * Math.PI * values[i +
    1])));
            sum += 20 + Math.E - (20 / first) - second;
        }
        return sum;
    }

    public double egg_holder() {
        double sum = 0;
        for (int i = 0; i < values.length - 1; i++) {
            double first = -values[i] * Math.sin(Math.sqrt(Math.abs(values[i] - values[i + 1] - 47)));
            double second = (values[i + 1] + 47) * Math.sin(Math.sqrt(Math.abs(values[i + 1] + 47 +
    values[i] / 2)));
            sum += first - second;
        }
        return sum;
    }

}
```

8.13 README.md File Reference

Index