Swarm Optimization

1.1

Generated by Doxygen 1.8.14

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

Class Documentation

2.1 Benchmark Class Reference

Public Types

- typedef double(Benchmark::* **Fitness**) (vector< double >)

Public Member Functions

- Benchmark ()
- Benchmark (int, int, Range)
- ~Benchmark ()
- void readText (const char *)
- void toCSV (const char *)
- void add_results (const vector< string > &)
- void clear_test_data ()
- void setSimulation (int count)
- void addHeader (string)
- vector< vector< double > > test_data ()
- void pseudo_random_matrix (int, int, Range)
- double schwefel (vector< double >)
- double first_de_jong (vector< double >)
- double rosenbrock (vector< double >)
- double rastrigin (vector< double >)
- double griewank (vector< double >)
- double sine_envelope_sine_wave (vector< double >)
- double stretch_v_sine_wave (vector< double >)
- double ackley_one (vector< double >)
- double ackley_two (vector< double >)
- double egg_holder (vector< double >)
- double rana (vector< double >)
- double pathological (vector< double >)
- double michalewicz (vector< double >)
- double masters_cosine_wave (vector< double >)
- double shekel_foxholes (vector< double >)

2.1.1 Detailed Description

Definition at line 13 of file Benchmark.h.

2.1.2 Constructor & Destructor Documentation

2.1.2.1 Benchmark() [1/2]

```
Benchmark::Benchmark ( )
```

This is the default constructor for Benchmark.

Definition at line 14 of file Benchmark.cpp.

2.1.2.2 Benchmark() [2/2]

```
Benchmark::Benchmark (
            int dimension,
            int max,
            Range rng )  [explicit]
```

This is an overloaded constructor for Benchmark that allows for creating simulation test data.

Parameters

dimension	amount of dimensions.
max	amount of simulations.
rng	this is the range (interval the values are in.

Definition at line 32 of file Benchmark.cpp.

2.1.2.3 ~Benchmark()

```
Benchmark::~Benchmark ( )
```

This is the destructor for Benchmark that will do clean up.

Definition at line 47 of file Benchmark.cpp.

2.1.3 Member Function Documentation

2.1.3.1 ackley_one()

```
double Benchmark::ackley_one (
            vector< double > x )
```

Find the benchmark result for Ackley One.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 341 of file Benchmark.cpp.

2.1.3.2 ackley_two()

```
double Benchmark::ackley_two (
            vector< double > x )
```

Find the benchmark result for Ackley Two.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 369 of file Benchmark.cpp.

2.1.3.3 add_results()

```
void Benchmark::add_results (
            const vector< string > & results )
```

This adds the computed results to Benchmark for converting to a CSV file.

Parameters

results	

Definition at line 85 of file Benchmark.cpp.

2.1.3.4 addHeader()

```
void Benchmark::addHeader (
            string hd )
```

This will add Time (sec) to the headers for the CSV file.

Definition at line 146 of file Benchmark.cpp.

2.1.3.5 clear_test_data()

```
void Benchmark::clear_test_data ( )
```

Clears the simulation test data.

Definition at line 94 of file Benchmark.cpp.

2.1.3.6 egg_holder()

```
double Benchmark::egg_holder (
            vector< double > x )
```

Find the benchmark result for Egg Holder.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 397 of file Benchmark.cpp.

2.1.3.7 first_de_jong()

```
double Benchmark::first_de_jong (
            vector< double > x )
```

Find the benchmark result for De Jong 1st.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 188 of file Benchmark.cpp.

2.1.3.8 griewank()

```
double Benchmark::griewank (
            vector< double > x )
```

Find the benchmark result for Griewank.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 261 of file Benchmark.cpp.

2.1.3.9 masters_cosine_wave()

```
double Benchmark::masters_cosine_wave (
            vector< double > x )
```

Find the benchmark result for Master's Cosine Wave.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 501 of file Benchmark.cpp.

2.1.3.10 michalewicz()

```
double Benchmark::michalewicz (
            vector< double > x )
```

Find the benchmark result for Michalewicz.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 476 of file Benchmark.cpp.

2.1.3.11 pathological()

```
double Benchmark::pathological (
            vector< double > x )
```

Find the benchmark result for Pathological.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 449 of file Benchmark.cpp.

2.1.3.12 pseudo_random_matrix()

```
void Benchmark::pseudo_random_matrix (
            int dimension,
            int max,
            Range rng )
```

This will randomly generate the simulation test data for the given parameters.

Parameters

dimension	amount of dimensions.
max	amount of simulations.
rng	this is the range (interval the values are in.

Definition at line 132 of file Benchmark.cpp.

2.1.3.13 rana()

```
double Benchmark::rana (
            vector< double > x )
```

Find the benchmark result for Rana.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 423 of file Benchmark.cpp.

2.1.3.14 rastrigin()

```
double Benchmark::rastrigin (
            vector< double > x )
```

Find the benchmark result for Rastrigin's Saddle.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 238 of file Benchmark.cpp.

2.1.3.15 readText()

```
void Benchmark::readText (
            const char * name )
```

This will read in a text file (*.txt) for Shekel's Foxhole test.

Parameters

name	the name of the file

Definition at line 61 of file Benchmark.cpp.

2.1.3.16 rosenbrock()

```
double Benchmark::rosenbrock (
            vector< double > x )
```

Find the benchmark result for Rosenbrock.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 211 of file Benchmark.cpp.

2.1.3.17 schwefel()

```
double Benchmark::schwefel (
            vector< double > x )
```

Find the benchmark result for Schwefel.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 165 of file Benchmark.cpp.

2.1.3.18 setSimulation()

```
void Benchmark::setSimulation (
            int count )
```

This will set the amount of simulations being performed on the benchmark algorithm

Parameters

count	the amount of simulation performed

Definition at line 106 of file Benchmark.cpp.

2.1.3.19 shekel_foxholes()

```
double Benchmark::shekel_foxholes (
            vector< double > x )
```

Find the benchmark result for Shekel's Foxhole.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 529 of file Benchmark.cpp.

2.1.3.20 sine_envelope_sine_wave()

```
double Benchmark::sine_envelope_sine_wave (
            vector< double > x )
```

Find the benchmark result for Sine Envelope Sine Wave.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 287 of file Benchmark.cpp.

2.1.3.21 stretch_v_sine_wave()

```
double Benchmark::stretch_v_sine_wave (
            vector< double > x )
```

Find the benchmark result for Stretch V Sine Wave.

Parameters

x	vector containing stochastic numbers in given range for all dimensions

Returns

the calculated fitness result.

Definition at line 314 of file Benchmark.cpp.

2.1.3.22 test_data()

```
vector< vector< double > > Benchmark::test_data ( )
```

This is a getter method to retrieve the simulation test data.

Returns

The the test data

Definition at line 118 of file Benchmark.cpp.

2.1.3.23 toCSV()

```
void Benchmark::toCSV (
            const char * name )
```

This will write all the test data to a comma-delimited text file (∗.csv).

Parameters

name	the name of the file

Definition at line 73 of file Benchmark.cpp.

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

- C:/Users/Shane Vance/CLionProjects/Optimization/Benchmark.h
- C:/Users/Shane Vance/CLionProjects/Optimization/Benchmark.cpp

2.2 ConstraintsFile Class Reference

Public Member Functions

- ConstraintsFile ()
- ConstraintsFile (const char *)

Public Attributes

- vector< string > **functionName**
- vector< Range > **range**
- vector< Dimension > **dimension**
- string **extraFile**
- int **pMax**
- int **iMax**
- int **totalSimulations**
- double **c1**
- double **c2**
- double **alpha**
- double **beta**
- double **gamma**

2.2.1 Detailed Description

Definition at line 79 of file ConstraintsFile.h.

2.2.2 Constructor & Destructor Documentation

2.2.2.1 ConstraintsFile() [1/2]

```
ConstraintsFile::ConstraintsFile ( )  [default]
```

This is the default constructor for the constraints class

2.2.2.2 ConstraintsFile() [2/2]

```
ConstraintsFile::ConstraintsFile (
            const char * name )  [explicit]
```

This will read constraints file with the values used for the Benchmark testing

Parameters

name	the name of the file you would like to read

Definition at line 20 of file ConstraintsFile.cpp.

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

- C:/Users/Shane Vance/CLionProjects/Optimization/ConstraintsFile.h
- C:/Users/Shane Vance/CLionProjects/Optimization/ConstraintsFile.cpp

2.3 Dimension Struct Reference

Public Member Functions

- **Dimension** (int lb, int ub)

Public Attributes

- int **LB**
- int **UB**

2.3.1 Detailed Description

Definition at line 27 of file ConstraintsFile.h.

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

- C:/Users/Shane Vance/CLionProjects/Optimization/ConstraintsFile.h

2.4 GeneticAlgorithms Class Reference

Public Member Functions

- GeneticAlgorithms (const Benchmark &)
- double simpleGA (Benchmark::Fitness, int, unsigned int, Range, int, double, Mutation, double)
- double diffEvolution (Benchmark::Fitness, unsigned int, int, int, double, double, Range, int)

2.4.1 Detailed Description

Definition at line 28 of file GeneticAlgorithms.h.

2.4.2 Constructor & Destructor Documentation

2.4.2.1 GeneticAlgorithms()

```
GeneticAlgorithms::GeneticAlgorithms (
            const Benchmark & newBM )  [explicit]
```

This sets up the genetic algorithm class.

Parameters

newBM	passes this for evaluating the functions

Definition at line 16 of file GeneticAlgorithms.cpp.

2.4.3 Member Function Documentation

2.4.3.1 diffEvolution()

```
double GeneticAlgorithms::diffEvolution (
            Benchmark::Fitness fn,
            unsigned int dim,
            int g_max,
            int np,
            double F,
            double cr,
            Range rng,
            int strategy )
```

This will evaluate the best solution of the fitness function using differential evolution and a selected strategy. The strategies vary from 1-10 and the first 5 are exponential and the other 5 are binomial. It goes through and selects random indexes to be evaluated with respect to its selected strategy.

Parameters

fn	the fitness function
dim	the amount of dimensions needed
g_max	the max amount of generations
np	the size of the population
F	the mutation rate
cr	the crossover rate
rng	the range of the fitness function
strategy	the selected strategy (1-10)

Returns

the best solution

Definition at line 99 of file GeneticAlgorithms.cpp.

2.4.3.2 simpleGA()

```
double GeneticAlgorithms::simpleGA (
            Benchmark::Fitness f,
```

```
int ns,
unsigned int dim,
Range rng,
int t_max,
double cr,
Mutation m,
double er )
```

This is a version of the Genetic Algorithms (GA) that is known as Simple GA. This is the simplest algorithms of the GA's. This will find the best (most optimal solution) of the given fitness function with respect to its dimension and population.

Parameters

f	the fitness (cost) evaluator
ns	the size of the population
dim	the amount of chromosomes in the population (the dimension)
rng	the range of values
t_max	the max amount of generations
cr	the crossover rate
m	the mutation parameter
er	the elitism rate

Returns

the best solution

Definition at line 35 of file GeneticAlgorithms.cpp.

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

- C:/Users/Shane Vance/CLionProjects/Optimization/GeneticAlgorithms.h
- C:/Users/Shane Vance/CLionProjects/Optimization/GeneticAlgorithms.cpp

2.5 LocalSearch Class Reference

Public Member Functions

- LocalSearch ()
- LocalSearch (const Benchmark &)
- ~LocalSearch ()
- void setCount (int count)
- int getCount ()
- vector< double > randomWalk (Benchmark::Fitness, vector< double >, double *, int, Range)
- vector< double > localSearch (Benchmark::Fitness, const vector< double > &, double *, double, Range)
- vector< double > iterativeLocalSearch (Benchmark::Fitness, const vector< double > &, double *, double, int, Range)

2.5.1 Detailed Description

Definition at line 14 of file LocalSearch.h.

2.5.2 Constructor & Destructor Documentation

2.5.2.1 LocalSearch() [1/2]

```
LocalSearch::LocalSearch ( )
```

This sets up the LocalSearch algorithm to be used.

Definition at line 13 of file LocalSearch.cpp.

2.5.2.2 LocalSearch() [2/2]

```
LocalSearch::LocalSearch (
            const Benchmark & bm )  [explicit]
```

This is a constructor that passes reference to the benchmark data

Parameters

bm	the Benchmark class that has the test data we need

Definition at line 22 of file LocalSearch.cpp.

2.5.2.3 ~LocalSearch()

```
LocalSearch::~LocalSearch ( )  [default]
```

This is the default destructor for LocalSearch

2.5.3 Member Function Documentation

2.5.3.1 getCount()

```
int LocalSearch::getCount ( )
```

This will get the amount of times it took to find the best solution

Returns

Definition at line 50 of file LocalSearch.cpp.

2.5.3.2 iterativeLocalSearch()

```
vector< double > LocalSearch::iterativeLocalSearch (
            Benchmark::Fitness f,
            const vector< double > & init,
            double * f_best,
            double delta,
            int t_max,
            Range rng )
```

This performs the iterative local search algorithm using the empirical gradient descent to find the best solution of the local optima

Parameters

f	the fitness function we want to use
init	the initial random vector
f_best	the initial best solution
delta	the delta we will use for the neighborhood search
t_max	the maximum amount of iterations we want to test with
rng	the range of values we will be testing with

Returns

Definition at line 156 of file LocalSearch.cpp.

2.5.3.3 localSearch()

```
vector< double > LocalSearch::localSearch (
            Benchmark::Fitness f,
            const vector< double > & init,
            double * f_best,
            double delta,
            Range rng )
```

This performs the local search algorithm using the empirical gradient descent to find the best solution of the local optima

Parameters

f	the fitness function we want to use
init	the initial random vector
f_best	the initial best solution
delta	the delta we will use for the neighborhood search
rng	the range of values we will be testing with

Returns

Definition at line 113 of file LocalSearch.cpp.

2.5.3.4 randomWalk()

```
vector< double > LocalSearch::randomWalk (
            Benchmark::Fitness f_cost,
            vector< double > arg,
            double * fitness_0,
            int itr,
            Range rng )
```

This performs the random walk (a.k.a blind worker) algorithm for finding the best solution for the local optima

Parameters

f_cost	the fitness function we want to use
arg	the vector we want to perform random walk on
fitness_0	the initial best fitness
itr	how many times we want to iterate
dim	the amount of dimensions
rng	the range of the values of lower-bound and upper-bound

Returns

Definition at line 69 of file LocalSearch.cpp.

2.5.3.5 setCount()

```
void LocalSearch::setCount (
            int count )
```

This will set the count for the current iteration

Parameters

count	initialize a value

Definition at line 40 of file LocalSearch.cpp.

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

- C:/Users/Shane Vance/CLionProjects/Optimization/LocalSearch.h
- C:/Users/Shane Vance/CLionProjects/Optimization/LocalSearch.cpp

2.6 Mutation Struct Reference

Public Attributes

- double **rate**
- double **precision**
- double **range**

2.6.1 Detailed Description

Definition at line 41 of file ConstraintsFile.h.

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

- C:/Users/Shane Vance/CLionProjects/Optimization/ConstraintsFile.h

2.7 Parent Struct Reference

Public Attributes

- vector< double > **one**
- vector< double > **two**

2.7.1 Detailed Description

Definition at line 20 of file GeneticAlgorithms.h.

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

- C:/Users/Shane Vance/CLionProjects/Optimization/GeneticAlgorithms.h

2.8 Population Struct Reference

Public Member Functions

- Population (vector< double > genes)
- bool **operator**< (const Population &population) const

Public Attributes

- vector< double > **particle**
- vector< double > **velocity**
- double **cost** {}

2.8.1 Detailed Description

Definition at line 55 of file ConstraintsFile.h.

2.8.2 Constructor & Destructor Documentation

2.8.2.1 Population()

```
Population::Population (
            vector< double > genes )  [inline], [explicit]
```

This setup up the population structure

Parameters

genes	

Definition at line 66 of file ConstraintsFile.h.

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

- C:/Users/Shane Vance/CLionProjects/Optimization/ConstraintsFile.h

2.9 Range Struct Reference

Public Member Functions

- **Range** (double lb, double ub)

Public Attributes

- double **LB**
- double **UB**

2.9.1 Detailed Description

Definition at line 16 of file ConstraintsFile.h.

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

- C:/Users/Shane Vance/CLionProjects/Optimization/ConstraintsFile.h

2.10 SwarmIntelligence Class Reference

Public Member Functions

- SwarmIntelligence ()
- SwarmIntelligence (Benchmark)
- ~SwarmIntelligence ()
- double pso (Benchmark::Fitness, int, double, double, int, int, Range)
- double firefly (Benchmark::Fitness, int, int, int, double, double, double, Range)

2.10.1 Detailed Description

Definition at line 12 of file SwarmIntelligence.h.

2.10.2 Constructor & Destructor Documentation

2.10.2.1 SwarmIntelligence() [1/2]

```
SwarmIntelligence::SwarmIntelligence ( )  [default]
```

This is the default constructor for the Swarm Intelligence class.

2.10.2.2 SwarmIntelligence() [2/2]

```
SwarmIntelligence::SwarmIntelligence (
            Benchmark newBM )  [explicit]
```

This is the default constructor that passes in the BenchmarkClass. This way we can use the functions from there.

Parameters

newBM	Contains information with regard to the benchmark class

Definition at line 22 of file SwarmIntelligence.cpp.

2.10.2.3 ~SwarmIntelligence()

```
SwarmIntelligence::~SwarmIntelligence ( )  [default]
```

This is the default destructor for the SwarmIntelligence class.

2.10.3 Member Function Documentation

2.10.3.1 firefly()

```
double SwarmIntelligence::firefly (
            Benchmark::Fitness fn,
            int dim,
            int iMax,
            int fMax,
            double alpha,
            double betamin,
            double gamma,
            Range rng )
```

This is the Particle Swarm Optimization (PSO) algorithm that will evaluate the objective (fitness) function. It is optimized to find the minimum value of the given objective function.

Parameters

fn	the fitness function
dim	the dimension count
iMax	max iterations
fMax	the max amount of fireflies
alpha	the alpha value
betamin	the minimum value of betamin
gamma	the gamma to be evaluated
rng	the range of fitness function

Returns

the best solution

Definition at line 135 of file SwarmIntelligence.cpp.

2.10.3.2 pso()

```
double SwarmIntelligence::pso (
            Benchmark::Fitness fn,
            int dim,
            double c1,
            double c2,
            int iMax,
            int pMax,
            Range rng )
```

This is the Particle Swarm Optimization (PSO) algorithm that will evaluate the objective (fitness) function. It is optimized to find the minimum value of the given objective function.

Parameters

fn	the fitness function
dim	the amount of dimensions
c1	the step size of first value
c2	the step size of second value
iMax	max iterations
pMax	max population size
rng	the range of values from fitness function

Returns

the best solution

Definition at line 47 of file SwarmIntelligence.cpp.

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

- C:/Users/Shane Vance/CLionProjects/Optimization/SwarmIntelligence.h
- C:/Users/Shane Vance/CLionProjects/Optimization/SwarmIntelligence.cpp