Project 4 - Swarm Optimization

1.0

Generated by Doxygen 1.8.15

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 FA_Analysis Struct Reference

Firefly Algorithm Analysis Firefly Algorithm Analysis Structure, to keep track of the analysis performed on each population in the population list.

```
#include <FireflyAlgorithm.h>
```

Public Attributes

- string header = "Function ID,Average Fitness,Standard Deviation,Range(min),Range(max),Median,Time(ms),Function Calls\n"
- vector< int > functionIDs
- vector< double > avgFunctionFitness
- vector< double > standardDeviation
- vector< vector< double > > ranges
- vector< double > medianFunctionFitness
- vector< double > executionTimes
- vector< int > functionCalls

3.1.1 Detailed Description

Firefly Algorithm Analysis Firefly Algorithm Analysis Structure, to keep track of the analysis performed on each population in the population list.

3.1.2 Member Data Documentation

3.1.2.1 avgFunctionFitness

```
vector<double> FA_Analysis::avgFunctionFitness
```

List of the average fitness from function.

3.1.2.2 executionTimes

```
vector<double> FA_Analysis::executionTimes
```

List of execution times in ms for all functions.

3.1.2.3 functionCalls

```
vector<int> FA_Analysis::functionCalls
```

List of the amount of times a function was called.

3.1.2.4 functionIDs

```
vector<int> FA_Analysis::functionIDs
```

List of function IDs.

3.1.2.5 header

```
string FA_Analysis::header = "Function ID,Average Fitness,Standard Deviation,Range(min),Range(max),Median,Time
Calls\n"
```

Header used when saving the data.

3.1.2.6 medianFunctionFitness

```
vector<double> FA_Analysis::medianFunctionFitness
```

List of the Median fitness for each function.

3.1.2.7 ranges

```
vector<vector<double> > FA_Analysis::ranges
```

List of ranges for each fitness function.

3.1.2.8 standardDeviation

```
vector<double> FA_Analysis::standardDeviation
```

List of standard fitness deviations.

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

- FireflyAlgorithm.h

3.2 FA_Config Struct Reference

Holds all the user defined variables. Firefly Algorithm Configuration Structure, where all user defined variables that are used to configure the Firefly Algorithm are stored.

```
#include <FireflyAlgorithm.h>
```

Public Attributes

- int dimensions
- int popSize
- int iterations
- double alpha
- double betaMin
- double gamma

3.2.1 Detailed Description

Holds all the user defined variables. Firefly Algorithm Configuration Structure, where all user defined variables that are used to configure the Firefly Algorithm are stored.

3.2.2 Member Data Documentation

3.2.2.1 alpha

```
double FA_Config::alpha
```

Alpha scaling factor - range [0,1].

3.2.2.2 betaMin

```
double FA_Config::betaMin
```

Beta scaling factor.

3.2.2.3 dimensions

```
int FA_Config::dimensions
```

Number of dimensions per individual in population.

3.2.2.4 gamma

```
double FA_Config::gamma
```

Gamma scaling factor.

3.2.2.5 iterations

```
int FA_Config::iterations
```

Maximum number of iterations.

3.2.2.6 popSize

```
int FA_Config::popSize
```

Population size.

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

- FireflyAlgorithm.h

3.3 FA_Population Struct Reference

Holds all the population information. Firefly Algorithm Population Structure, holds all the data related to the population of the Firefly Algorithm.

```
#include <FireflyAlgorithm.h>
```

Public Attributes

- int functionID
- vector< double > bounds
- int functionCounter = 0
- vector< vector< double > > pop
- vector< double > fitness
- vector< double > bestGlobFit
- double executionTime = -1.0

3.3.1 Detailed Description

Holds all the population information. Firefly Algorithm Population Structure, holds all the data related to the population of the Firefly Algorithm.

3.3.2 Member Data Documentation

3.3.2.1 bestGlobFit

```
vector<double> FA_Population::bestGlobFit
```

A list of best global fitness values from each iteration.

3.3.2.2 bounds

```
vector<double> FA_Population::bounds
```

Holds the (min,max) bounds of the values for each individual in the population.

3.3.2.3 executionTime

```
double FA_Population::executionTime = -1.0
```

Time(ms) it took to run the Firefly Algorithm on this population.

3.3.2.4 fitness

```
vector<double> FA_Population::fitness
```

The fitness for each vector in the population matrix.

3.3.2.5 functionCounter

```
int FA_Population::functionCounter = 0
```

The function counter keeps track of how many times the benchmark function was called.

3.3.2.6 functionID

```
int FA_Population::functionID
```

The ID determines which benchmark function to call.

3.3.2.7 pop

```
vector<vector<double> > FA_Population::pop
```

The population matrix.

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

- FireflyAlgorithm.h

3.4 FireflyAlgorithm Class Reference

Public Member Functions

- FireflyAlgorithm (int dimensions, int populationSize, int maxIterations, double alpha, double beta, double gamma)

  The Firefly Algorithm constructor.
- double runFireflyAlgorithm (int functionID, double minBound, double maxBound)

  Runs the Firefly Algorithm with set parameters.
- void analyzeFAResults ()

  Analyzes the results of the Firefly Algorithm.
- void printFAResults ()

  Prints the Results of the Firefly Algorithm.
- void printFAAnalysis ()

  Prints the Analysis of the Firefly Algorithm Results.
- void saveFAResults ()

  Saves all Firefly Algorithm Results to file.
- void saveFAAnalysis ()

  Saves the Analysis of the Firefly Algorithm to file.
- void saveEndingPopulation ()

  Saves the ending population solutions to file.

Private Member Functions

- void generateFAPopulation (FA_Population &population, mt19937 &randGenerator)

  Generates the initial population.
- void evaluatePopulation (int functionID, vector< vector< double >> &pop, vector< double > &fitness, int &functionCounter)

  Calculates fitness of all solutions in population.
- void evaluateIndividual (const int &functionID, vector< double > &indiv, double &fitness, int &functionCounter)

  Calculate the fitness of an individual solution of the population.
- double calculateDistanceBetweenFireflies (const vector< double > &firefly1, const vector< double > &firefly2)

  Calculates the distance between two fireflies.
- void calculateLightIntensity (const double &ffFit1, const double &ffFit2, const double &r, const double &gamma, double &lIntensity1, double &lIntensity2)

  Calculates the light intensity of two fireflies.
- double calculateAttractiveness (const double &betaMin, const double &gamma, const double &r)

  Calculates the attractiveness between two fireflies.
- void moveFirefly (const vector< double > &brightFF, vector< double > &lessBrightFF, const double &alpha, const double &betaMin, const double &gamma, const double &r, mt19937 &randGenerator)

  Moves the less brighter firefly towards the brighter firefly.
- void moveLessBrightFireflies (const int &currFFIndex, FA_Population &population, const double &alpha, const double &betaMin, const double &gamma, mt19937 &randGenerator)

  Moves all fireflies that are less brighter than the current firefly, towards the current firefly.
- void iterateFireflies (FA_Population &population, const double &alpha, const double &betaMin, const double &gamma, mt19937 &randGenerator)

  Iterates the firefly population to the next generation.

Private Attributes

- FA_Config **faConfig**
- vector< FA_Population > **popList**
- FA_Analysis **faAnalysis**

3.4.1 Constructor & Destructor Documentation

3.4.1.1 FireflyAlgorithm()

```
FireflyAlgorithm::FireflyAlgorithm (
             int dimensions,
             int populationSize,
             int maxIterations,
             double alpha,
             double beta,
             double gamma )
```

The Firefly Algorithm constructor.

The Firefly Algorithm constructor.

Note

This is the only constructor for the Firefly Algorithm, no default constructor exists.

Parameters

dimensions	The number of elements per individual vector in the population.
populationSize	The size of the population.
maxIterations	The maximum number of iterations.
alpha	The alpha scaling factor.
beta	The minimum beta scaling factor.
gamma	The gamma scaling factor.

3.4.2 Member Function Documentation

3.4.2.1 analyzeFAResults()

```
void FireflyAlgorithm::analyzeFAResults ( )
```

Analyzes the results of the Firefly Algorithm.

Analyzes the results of the Firefly Algorithm.

3.4.2.2 calculateAttractiveness()

```
double FireflyAlgorithm::calculateAttractiveness (
            const double & betaMin,
            const double & gamma,
            const double & r )  [private]
```

Calculates the attractiveness between two fireflies.

Calculates the attractiveness between two fireflies.

Parameters

betaMin	The beta scaling factor.
gamma	The gamma scaling factor.
r	The distance between the two fireflies.

Returns

Returns the attractiveness value between two fireflies.

3.4.2.3 calculateDistanceBetweenFireflies()

```
double FireflyAlgorithm::calculateDistanceBetweenFireflies (
            const vector< double > & firefly1,
            const vector< double > & firefly2 )  [private]
```

Calculates the distance between two fireflies.

Calculates the distance between two fireflies.

Parameters

firefly1	The first firefly.
firefly2	The second firefly.

Returns

Returns the distance between the two fireflies.

3.4.2.4 calculateLightIntensity()

```
void FireflyAlgorithm::calculateLightIntensity (
            const double & ffFit1,
            const double & ffFit2,
            const double & r,
            const double & gamma,
            double & lIntensity1,
            double & lIntensity2 )  [private]
```

Calculates the light intensity of two fireflies.

Calculates the light intensity of two fireflies.

Note

lIntensity1/lIntensity2 can be null, initialized, or uninitialized since they'll be changed in the function.

Parameters

ffFit1	Fitness of the first firefly.
ffFit2	Fitness of the second firefly.
r	The distance between the two fireflies.
gamma	The gamma scaling factor.
lIntensity1	The light intensity of the first firefly.
lIntensity2	The light intensity of the second firefly.

3.4.2.5 evaluateIndividual()

```
void FireflyAlgorithm::evaluateIndividual (
            const int & functionID,
            vector< double > & indiv,
            double & fitness,
            int & functionCounter )  [private]
```

Calculate the fitness of an individual solution of the population.

Calculate the fitness of an individual solution of the population.

Note

Makes function call to SwarmUtilities.h --> calculateFitnessOfVector().

Parameters

functionID	The ID of the benchmark function to use.
indiv	The individual of the population.
fitness	The fitness variable for the individual.
functionCounter	A counter to keep track of how many times fitness function was called.

3.4.2.6 evaluatePopulation()

```
void FireflyAlgorithm::evaluatePopulation (
            int functionID,
            vector< vector< double >> & pop,
            vector< double > & fitness,
            int & functionCounter )  [private]
```

Calculates fitness of all solutions in population.

Calculates fitness of all solutions in population.

Note

Makes function call to SwarmUtilities.h --> calculateFitnessOfVector().

Parameters

functionID	The ID of the benchmark function to use.
pop	The population matrix.
fitness	The fitness vector for each solution from the population.
functionCounter	A counter to keep track of how many times fitness function was called.

3.4.2.7 generateFAPopulation()

```
void FireflyAlgorithm::generateFAPopulation (
            FA_Population & population,
            mt19937 & randGenerator )  [private]
```

Generates the initial population.

Generates the initial population.

Note

Makes function call to SwarmUtilities.h --> createMatrixMT().

Parameters

population	The FA_Population structure that holds the population.
randGenerator	The Mersenne Twister pseudo-random number generator.

3.4.2.8 iterateFireflies()

```
void FireflyAlgorithm::iterateFireflies (
            FA_Population & population,
            const double & alpha,
            const double & betaMin,
            const double & gamma,
            mt19937 & randGenerator )  [private]
```

Iterates the firefly population to the next generation.

Iterates the firefly population to the next generation.

Parameters

population	The population of fireflies.
alpha	The alpha scaling factor.
betaMin	The beta scaling factor.
gamma	The gamma scaling factor.
randGenerator	The Mersenne Twister pseudo-random number generator.

3.4.2.9 moveFirefly()

```
void FireflyAlgorithm::moveFirefly (
             const vector< double > & brightFF,
             vector< double > & lessBrightFF,
             const double & alpha,
             const double & betaMin,
             const double & gamma,
             const double & r,
             mt19937 & randGenerator )  [private]
```

Moves the less brighter firefly towards the brighter firefly.

Moves the less brighter firefly towards the brighter firefly.

Parameters

brightFF	The brighter firefly.
lessBrightFF	The less brighter firefly.
alpha	The alpha scaling factor.
betaMin	The beta scaling factor.
gamma	The gamma scaling factor.
r	The distance between the two fireflies.
randGenerator	The Mersenne Twister pseudo-random number generator.

3.4.2.10 moveLessBrightFireflies()

```
void FireflyAlgorithm::moveLessBrightFireflies (
             const int & currFFIndex,
             FA_Population & population,
             const double & alpha,
             const double & betaMin,
             const double & gamma,
             mt19937 & randGenerator )  [private]
```

Moves all fireflies that are less brighter than the current firefly, towards the current firefly.

Moves all less brighter fireflies towards the current firefly.

Parameters

currFFIndex	Index of the current firefly in the population.
population	The population of fireflies.
alpha	The alpha scaling factor.
betaMin	The beta scaling factor.
gamma	The gamma scaling factor.
randGenerator	The Mersenne Twister pseudo-random number generator.

3.4.2.11 printFAAnalysis()

```
void FireflyAlgorithm::printFAAnalysis ( )
```

Prints the Analysis of the Firefly Algorithm Results.

Prints the Analysis of the Firefly Algorithm Results.

3.4.2.12 printFAResults()

```
void FireflyAlgorithm::printFAResults ( )
```

Prints the Results of the Firefly Algorithm.

Prints the Results of the Firefly Algorithm.

3.4.2.13 runFireflyAlgorithm()

```
double FireflyAlgorithm::runFireflyAlgorithm (
             int functionID,
             double minBound,
             double maxBound )
```

Runs the Firefly Algorithm with set parameters.

Runs the Firefly Algorithm with set parameters.

Parameters

functionID	The ID that references which Benchmark Function to use.
minBound,maxBound	The minimum and maximum bounds of the individuals in the firefly algorithm.

Returns

Returns the best global fitness.

3.4.2.14 saveEndingPopulation()

```
void FireflyAlgorithm::saveEndingPopulation ( )
```

Saves the ending population solutions to file.

Saves the ending population solutions to file.

3.4.2.15 saveFAAnalysis()

```
void FireflyAlgorithm::saveFAAnalysis ( )
```

Saves the Analysis of the Firefly Algorithm to file.

Saves the Analysis of the Firefly Algorithm to file.

3.4.2.16 saveFAResults()

```
void FireflyAlgorithm::saveFAResults ( )
```

Saves all Firefly Algorithm Results to file.

Saves all Firefly Algorithm Results to file.

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

- FireflyAlgorithm.h
- FireflyAlgorithm.cpp

3.5 HarmonySearch Class Reference

Public Member Functions

- HarmonySearch (int dimensions, int populationSize, int maxIterations, double HMCR, double PAR, double bandwidth)

  The Harmony Search constructor.
- double runHarmonySearch (int functionID, double minBound, double maxBound)

  Runs the Harmony Search with set parameters.
- void analyzeHSResults ()

  Analyzes the results of the Harmony Search.
- void analyzeHSWorstResults ()

  Analyses the worst results of the Harmony Search.
- void printHSResults ()

  Prints the Results of the Harmony Search.
- void printHSAnalysis ()

  Prints the Analysis of the Harmony Search Results.
- void printHSWorstAnalysis ()

  Prints the Analysis of the worst Harmony Search Results.
- void saveHSResults ()

  Saves all Harmony Search Results to file.
- void saveHSAnalysis ()

  Saves the Analysis of the Harmony Search to file.
- void saveHSWorstAnalysis ()

  Saves the Analysis of the worst Harmony Search Results to file.

Private Member Functions

- void generateHSPopulation (HS_Population &population, mt19937 &randGenerator)

  Generates the initial population.
- void evaluatePopulation (int functionID, vector< vector< double >> &pop, vector< double > &fitness, int &functionCounter)

  Calculates fitness of all solutions in population.
- void evaluateIndividual (const int &functionID, vector< double > &indiv, double &fitness, int &functionCounter)

  Calculate the fitness of an individual solution of the population.
- double chooseRandomHarmonic (const vector< vector< double >> &pop, mt19937 &randGenerator)

  Choose a random harmonic (dimension) from the population.
- double adjustHarmonicPitch (const vector< vector< double >> &pop, const int &dim, const double &PAR, const double &bandwidth, mt19937 &randGenerator)

  Adjust the pitch of a random solution's harmonic (dimension).
- double generateNewRandHarmonic (const double &minBound, const double &maxBound, mt19937 &rand↩ Generator)

  Generate a new harmonic (dimension) within the bounds.
- vector< double > generateNewRandHarmony (HS_Population &population, const double &HMCR, const double &PAR, const double &bandwidth, mt19937 &randGenerator)

  Generates a new random harmony from existing population.
- void iterateHarmony (HS_Population &population, const double &HMCR, const double &PAR, const double &bandwidth, mt19937 &randGenerator)

  Iterate the harmonic population.

Private Attributes

- HS_Config **hsConfig**
- vector< HS_Population > **popList**
- HS_Analysis **hsAnalysis**
- HS_Analysis_Worst **hsWorstAnalysis**

3.5.1 Constructor & Destructor Documentation

3.5.1.1 HarmonySearch()

```
HarmonySearch::HarmonySearch (
             int dimensions,
             int populationSize,
             int maxIterations,
             double HMCR,
             double PAR,
             double bandwidth )
```

The Harmony Search constructor.

The Harmony Search constructor.

Note

This is the only constructor for the Harmony Search, no default constructor exists.

Parameters

dimensions	The number of elements per individual vector in the population.
populationSize	The size of the population.
maxIterations	The maximum number of iterations.
HMCR	The Harmony Memory Consideration Rate.
PAR	The Pitch Adjustment Rate.
bandwidth	The bandwidth range.

3.5.2 Member Function Documentation

3.5.2.1 adjustHarmonicPitch()

```
double HarmonySearch::adjustHarmonicPitch (
             const vector< vector< double >> & pop,
             const int & dim,
             const double & PAR,
             const double & bandwidth,
             mt19937 & randGenerator )  [private]
```

Adjust the pitch of a random solution's harmonic (dimension).

Adjust the pitch of a random solution's harmonic (dimension).

Parameters

pop	The matrix population.
dim	The harmonic (dimension) where to adjust the pitch.
PAR	The Pitch Adjustment Rate.
bandwidth	The bandwidth range for the adjustment.
randGenerator	The Mersenne Twister pseudo-random number generator.

Returns

Returns the harmonic with the adjusted pitch.

3.5.2.2 analyzeHSResults()

```
void HarmonySearch::analyzeHSResults ( )
```

Analyzes the results of the Harmony Search.

Analyzes the results of the Harmony Search.

3.5.2.3 analyzeHSWorstResults()

```
void HarmonySearch::analyzeHSWorstResults ( )
```

Analyses the worst results of the Harmony Search.

Analyses the worst results of the Harmony Search.

3.5.2.4 chooseRandomHarmonic()

```
double HarmonySearch::chooseRandomHarmonic (
            const vector< vector< double >> & pop,
            mt19937 & randGenerator )  [private]
```

Choose a random harmonic (dimension) from the population.

Choose a random harmonic (dimension) from the population.

Parameters

pop	The matrix population.
randGenerator	The Mersenne Twister pseudo-random number generator.

Returns

Returns a random harmonic (dimension) from the population.

3.5.2.5 evaluateIndividual()

```
void HarmonySearch::evaluateIndividual (
            const int & functionID,
            vector< double > & indiv,
            double & fitness,
            int & functionCounter )  [private]
```

Calculate the fitness of an individual solution of the population.

Calculate the fitness of an individual solution of the population.

Note

Makes function call to SwarmUtilities.h --> calculateFitnessOfVector().

Parameters

functionID	The ID of the benchmark function to use.
indiv	The individual of the population.
fitness	The fitness variable for the individual.
functionCounter	A counter to keep track of how many times fitness function was called.

3.5.2.6 evaluatePopulation()

```
void HarmonySearch::evaluatePopulation (
            int functionID,
            vector< vector< double >> & pop,
            vector< double > & fitness,
            int & functionCounter )  [private]
```

Calculates fitness of all solutions in population.

Calculates fitness of all solutions in population.

Note

Makes function call to SwarmUtilities.h --> calculateFitnessOfVector().

Parameters

functionID	The ID of the benchmark function to use.
pop	The population matrix.
fitness	The fitness vector for each solution from the population.
functionCounter	A counter to keep track of how many times fitness function was called.

3.5.2.7 generateHSPopulation()

```
void HarmonySearch::generateHSPopulation (
            HS_Population & population,
            mt19937 & randGenerator )  [private]
```

Generates the initial population.

Generates the initial population.

Note

Makes function call to SwarmUtilities.h --> createMatrixMT().

Parameters

population	The HS_Population structure that holds the population.
randGenerator	The Mersenne Twister pseudo-random number generator.

3.5.2.8 generateNewRandHarmonic()

```
double HarmonySearch::generateNewRandHarmonic (
            const double & minBound,
            const double & maxBound,
            mt19937 & randGenerator )  [private]
```

Generate a new harmonic (dimension) within the bounds.

Generate a new harmonic (dimension) within the bounds.

Parameters

minBound,maxBound	The minimum and maximum bounds of the individual (harmonic).
randGenerator	The Mersenne Twister pseudo-random number generator.

Returns

Returns a newly generated harmonic within the specified bounds.

3.5.2.9 generateNewRandHarmony()

```
vector< double > HarmonySearch::generateNewRandHarmony (
             HS_Population & population,
             const double & HMCR,
             const double & PAR,
             const double & bandwidth,
             mt19937 & randGenerator )  [private]
```

Generates a new random harmony from existing population.

Generates a new random harmony from existing population.

Parameters

population	The HS_Population structure that holds the population.
HMCR	The Harmony Memory Consideration Rate.
PAR	The Pitch Adjustment Rate.
bandwidth	The bandwidth range.
randGenerator	The Mersenne Twister pseudo-random number generator.

Returns

Returns a new random harmony (solution).

3.5.2.10 iterateHarmony()

```
void HarmonySearch::iterateHarmony (
             HS_Population & population,
             const double & HMCR,
             const double & PAR,
             const double & bandwidth,
             mt19937 & randGenerator )  [private]
```

Iterate the harmonic population.

Iterate the harmonic population.

Parameters

population	The HS_Population structure that holds the population.
HMCR	The Harmony Memory Consideration Rate.
PAR	The Pitch Adjustment Rate.
bandwidth	The bandwidth range.
randGenerator	The Mersenne Twister pseudo-random number generator.

3.5.2.11 printHSAnalysis()

```
void HarmonySearch::printHSAnalysis ( )
```

Prints the Analysis of the Harmony Search Results.

Prints the Analysis of the Harmony Search Results.

3.5.2.12 printHSResults()

```
void HarmonySearch::printHSResults ( )
```

Prints the Results of the Harmony Search.

Prints the Results of the Harmony Search.

3.5.2.13 printHSWorstAnalysis()

```
void HarmonySearch::printHSWorstAnalysis ( )
```

Prints the Analysis of the worst Harmony Search Results.

Prints the Analysis of the worst Harmony Search Results.

3.5.2.14 runHarmonySearch()

```
double HarmonySearch::runHarmonySearch (
             int functionID,
             double minBound,
             double maxBound )
```

Runs the Harmony Search with set parameters.

Runs the Harmony Search with set parameters.

Parameters

functionID	The ID that references which Benchmark Function to use.
minBound,maxBound	The minimum and maximum bounds of the individuals in Harmony Search.

Returns

Returns the best global fitness.

3.5.2.15 saveHSAnalysis()

```
void HarmonySearch::saveHSAnalysis ( )
```

Saves the Analysis of the Harmony Search to file.

Saves the Analysis of the Harmony Search to file.

3.5.2.16 saveHSResults()

```
void HarmonySearch::saveHSResults ( )
```

Saves all Harmony Search Results to file.

Saves all Harmony Search Results to file.

3.5.2.17 saveHSWorstAnalysis()

```
void HarmonySearch::saveHSWorstAnalysis ( )
```

Saves the Analysis of the worst Harmony Search Results to file.

Saves the Analysis of the worst Harmony Search Results to file.

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

- HarmonySearch.h
- HarmonySearch.cpp

3.6 HS_Analysis Struct Reference

Harmony Search Analysis Harmony Search Analysis Structure, to keep track of the analysis performed on each population in the population list.

```
#include <HarmonySearch.h>
```

Public Attributes

- string header = "Function ID,Average Fitness,Standard Deviation,Range(min),Range(max),Median,Time(ms),Function Calls\n"
- vector< int > functionIDs
- vector< double > avgFunctionFitness
- vector< double > standardDeviation
- vector< vector< double > > ranges
- vector< double > medianFunctionFitness
- vector< double > executionTimes
- vector< int > functionCalls

3.6.1 Detailed Description

Harmony Search Analysis Harmony Search Analysis Structure, to keep track of the analysis performed on each population in the population list.

3.6.2 Member Data Documentation

3.6.2.1 avgFunctionFitness

`vector<double> HS_Analysis::avgFunctionFitness`

List of the average fitness from function.

3.6.2.2 executionTimes

`vector<double> HS_Analysis::executionTimes`

List of execution times in ms for all functions.

3.6.2.3 functionCalls

`vector<int> HS_Analysis::functionCalls`

List of the amount of times a function was called.

3.6.2.4 functionIDs

`vector<int> HS_Analysis::functionIDs`

List of function IDs.

3.6.2.5 header

```
string HS_Analysis::header = "Function ID,Average Fitness,Standard Deviation,Range(min),Range(max),Median,Time
Calls\n"
```

Header used when saving the data.

3.6.2.6 medianFunctionFitness

`vector<double> HS_Analysis::medianFunctionFitness`

List of the Median fitness for each function.

3.6.2.7 ranges

```
vector<vector<double> > HS_Analysis::ranges
```

List of ranges for each fitness function.

3.6.2.8 standardDeviation

```
vector<double> HS_Analysis::standardDeviation
```

List of standard fitness deviations.

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

- HarmonySearch.h

3.7 HS_Analysis_Worst Struct Reference

Harmony Search Analysis Worst Harmony Search Analysis Worst Structure, to keep track of the analysis performed on the list of worst solutions of each population in the population list.

```
#include <HarmonySearch.h>
```

Public Attributes

- string header = "Function ID,Average Fitness,Standard Deviation,Range(min),Range(max),Median,Time(ms),Function Calls\n"
- vector< int > functionIDs
- vector< double > avgFunctionFitness
- vector< double > standardDeviation
- vector< vector< double > > ranges
- vector< double > medianFunctionFitness
- vector< double > executionTimes
- vector< int > functionCalls

3.7.1 Detailed Description

Harmony Search Analysis Worst Harmony Search Analysis Worst Structure, to keep track of the analysis performed on the list of worst solutions of each population in the population list.

3.7.2 Member Data Documentation

3.7.2.1 avgFunctionFitness

```
vector<double> HS_Analysis_Worst::avgFunctionFitness
```

List of the average fitness from function.

3.7.2.2 executionTimes

```
vector<double> HS_Analysis_Worst::executionTimes
```

List of execution times in ms for all functions.

3.7.2.3 functionCalls

```
vector<int> HS_Analysis_Worst::functionCalls
```

List of the amount of times a function was called.

3.7.2.4 functionIDs

```
vector<int> HS_Analysis_Worst::functionIDs
```

List of function IDs.

3.7.2.5 header

```
string HS_Analysis_Worst::header = "Function ID,Average Fitness,Standard Deviation,Range(min),Range(max),Media
Calls\n"
```

Header used when saving the data.

3.7.2.6 medianFunctionFitness

```
vector<double> HS_Analysis_Worst::medianFunctionFitness
```

List of the Median fitness for each function.

3.7.2.7 ranges

```
vector<vector<double> > HS_Analysis_Worst::ranges
```

List of ranges for each fitness function.

3.7.2.8 standardDeviation

```
vector<double> HS_Analysis_Worst::standardDeviation
```

List of standard fitness deviations.

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

- HarmonySearch.h

3.8 HS_Config Struct Reference

Holds all the user defined variables. Harmony Search Configuration Structure, where all user defined variables that are used to configure the Harmony Search are stored.

```
#include <HarmonySearch.h>
```

Public Attributes

- int dimensions
- int popSize
- int iterations
- double HMCR
- double PAR
- double bandwidth

3.8.1 Detailed Description

Holds all the user defined variables. Harmony Search Configuration Structure, where all user defined variables that are used to configure the Harmony Search are stored.

3.8.2 Member Data Documentation

3.8.2.1 bandwidth

```
double HS_Config::bandwidth
```

The Bandwidth.

3.8.2.2 dimensions

```
int HS_Config::dimensions
```

Number of dimensions per individual in population.

3.8.2.3 HMCR

```
double HS_Config::HMCR
```

Harmony Memory Consideration Rate - range [0,1].

3.8.2.4 iterations

```
int HS_Config::iterations
```

Maximum number of iterations.

3.8.2.5 PAR

```
double HS_Config::PAR
```

Pitch Adjustment Rate - range [0,1].

3.8.2.6 popSize

```
int HS_Config::popSize
```

Population size.

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

- HarmonySearch.h

3.9 HS_Population Struct Reference

Holds all the population information. Harmony Search Population Structure, holds all the data related to the population of the Harmony Search.

```
#include <HarmonySearch.h>
```

Public Attributes

- int functionID
- vector< double > bounds
- int functionCounter = 0
- vector< vector< double > > pop
- vector< double > fitness
- vector< double > bestGlobFit
- vector< vector< double > > worstSol
- vector< double > worstFitness
- double executionTime = -1.0

3.9.1 Detailed Description

Holds all the population information. Harmony Search Population Structure, holds all the data related to the population of the Harmony Search.

3.9.2 Member Data Documentation

3.9.2.1 bestGlobFit

```
vector<double> HS_Population::bestGlobFit
```

A list of best global fitness values from each iteration.

3.9.2.2 bounds

```
vector<double> HS_Population::bounds
```

Holds the (min,max) bounds of the values for each individual in the population.

3.9.2.3 executionTime

```
double HS_Population::executionTime = -1.0
```

Time(ms) it took to run the Harmony Search on this population.

3.9.2.4 fitness

```
vector<double> HS_Population::fitness
```

The fitness for each vector in the population matrix.

3.9.2.5 functionCounter

```
int HS_Population::functionCounter = 0
```

The function counter keeps track of how many times the benchmark function was called.

3.9.2.6 functionID

```
int HS_Population::functionID
```

The ID determines which benchmark function to call.

3.9.2.7 pop

```
vector<vector<double> > HS_Population::pop
```

The population matrix.

3.9.2.8 worstFitness

```
vector<double> HS_Population::worstFitness
```

A list of worst fitness values from the worstSol population.

3.9.2.9 worstSol

```
vector<vector<double> > HS_Population::worstSol
```

Population of the worst solutions from the actual population.

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

- HarmonySearch.h

3.10 ParticleSwarm Class Reference

Public Member Functions

- ParticleSwarm (int dimensions, int populationSize, int maxIterations, double kDampeningFactor, double c1, double c2)

  The Particle Swarm constructor.
- double runParticleSwarm (int functionID, double minBound, double maxBound)

  Runs the Particle Swarm with set parameters.
- void analyzePSResults ()

  Analyzes the results of the Particle Swarm.
- void printPSResults ()

  Prints the Results of the Particle Swarm.
- void printPSAnalysis ()

  Prints the Analysis of the Particle Swarm Results.
- void savePSResults ()

  Saves all Particle Swarm Results to file.
- void savePSAnalysis ()

  Saves the Analysis of the Particle Swarm to file.
- void saveEndingPopulation ()

  Saves the ending population solutions to file.

Private Member Functions

- void generatePSPopulation (PS_Population &population, mt19937 &randGenerator)

  Generates the initial population for Particle Swarm.
- void calculateParticleVelocity (vector< double > &pVelocity, vector< double > particle, vector< double > pBest, vector< double > gBest, const double &k, const double &c1, const double &c2, mt19937 &rand↩ Generator)

  Calculates the velocity of a particle in the Particle Swarm.
- void evaluatePopulation (int functionID, vector< vector< double >> &pop, vector< double > &fitness, int &functionCounter)

  Calculates fitness of all solutions in population.
- void evaluateIndividual (const int &functionID, vector< double > &indiv, double &fitness, int &functionCounter)

  Calculate the fitness of an individual solution of the population.
- void updateParticle (PS_Population &population, int particleIndex, mt19937 &randGenerator)

  Updates a single particle in the population.
- void iteratePopulation (PS_Population &population, mt19937 &randGenerator)

  Iterates the Particle Swarm population to the next generation.

Private Attributes

- PS_Config **psConfig**
- vector< PS_Population > **popList**
- PS_Analysis **psAnalysis**

3.10.1 Constructor & Destructor Documentation

3.10.1.1 ParticleSwarm()

```
ParticleSwarm::ParticleSwarm (
            int dimensions,
            int populationSize,
            int maxIterations,
            double kDampeningFactor,
            double c1,
            double c2 )
```

The Particle Swarm constructor.

The Particle Swarm constructor.

Note

This is the only constructor for the Particle Swarm, no default constructor exists.

Parameters

dimensions	The number of elements per individual vector in the population.
populationSize	The size of the population.
maxIterations	The maximum number of iterations.
kDampeningFactor	The dampening factor, to dampen the velocities of particles.
c1	The scaling factor to bring velocity closer to personal best.
c2	The scaling factor to bring velocity closer to global best.

3.10.2 Member Function Documentation

3.10.2.1 analyzePSResults()

```
void ParticleSwarm::analyzePSResults ( )
```

Analyzes the results of the Particle Swarm.

Analyzes the results of the Particle Swarm.

3.10.2.2 calculateParticleVelocity()

```
void ParticleSwarm::calculateParticleVelocity (
            vector< double > & pVelocity,
            vector< double > particle,
            vector< double > pBest,
            vector< double > gBest,
            const double & k,
            const double & c1,
            const double & c2,
            mt19937 & randGenerator )  [private]
```

Calculates the velocity of a particle in the Particle Swarm.

Calculates the velocity of a particle in the Particle Swarm.

Note

If c1 $>$ c2 then velocity is changed towards the personal best.
If c1 $<$ c2 then velocity is changed towards the global best.

Parameters

pVelocity	The velocity vector of the particle.
particle	The particle (individual) of the population.
pBest	The personal best solution of the particle.
gBest	The global best particle in the population.
k	The velocity dampening factor.
c1	The scaling factor to bring velocity closer to personal best.
c2	The scaling factor to bring velocity closer to global best.
randGenerator	The Mersenne Twister pseudo-random number generator.

3.10.2.3 evaluateIndividual()

```
void ParticleSwarm::evaluateIndividual (
            const int & functionID,
            vector< double > & indiv,
            double & fitness,
            int & functionCounter )  [private]
```

Calculate the fitness of an individual solution of the population.

Calculate the fitness of an individual solution of the population.

Note

Makes function call to SwarmUtilities.h --> calculateFitnessOfVector().

Parameters

functionID	The ID of the benchmark function to use.
indiv	The individual of the population.
fitness	The fitness variable for the individual.
functionCounter	A counter to keep track of how many times fitness function was called.

3.10.2.4 evaluatePopulation()

```
void ParticleSwarm::evaluatePopulation (
            int functionID,
            vector< vector< double >> & pop,
            vector< double > & fitness,
            int & functionCounter )  [private]
```

Calculates fitness of all solutions in population.

Calculates fitness of all solutions in population.

Note

Makes function call to SwarmUtilities.h --> calculateFitnessOfVector().

Parameters

functionID	The ID of the benchmark function to use.
pop	The population matrix.
fitness	The fitness vector for each solution from the population.
functionCounter	A counter to keep track of how many times fitness function was called.

3.10.2.5 generatePSPopulation()

```
void ParticleSwarm::generatePSPopulation (
            PS_Population & population,
            mt19937 & randGenerator )  [private]
```

Generates the initial population for Particle Swarm.

Generates the initial population for Particle Swarm.

Note

Makes function call to SwarmUtilities.h --> createMatrixMT().

Parameters

population	The PS_Population structure that holds the particle swarm population.
randGenerator	The Mersenne Twister pseudo-random number generator.

3.10.2.6 iteratePopulation()

```
void ParticleSwarm::iteratePopulation (
            PS_Population & population,
            mt19937 & randGenerator )  [private]
```

Iterates the Particle Swarm population to the next generation.

Iterates the Particle Swarm population to the next generation.

Parameters

population	The PS_Population structure that holds the particle swarm population.
randGenerator	The Mersenne Twister pseudo-random number generator.

3.10.2.7 printPSAnalysis()

```
void ParticleSwarm::printPSAnalysis ( )
```

Prints the Analysis of the Particle Swarm Results.

Prints the Analysis of the Particle Swarm Results.

3.10.2.8 printPSResults()

```
void ParticleSwarm::printPSResults ( )
```

Prints the Results of the Particle Swarm.

Prints the Results of the Particle Swarm.

3.10.2.9 runParticleSwarm()

```
double ParticleSwarm::runParticleSwarm (
             int functionID,
             double minBound,
             double maxBound )
```

Runs the Particle Swarm with set parameters.

Runs the Particle Swarm with set parameters.

Parameters

functionID	The ID that references which Benchmark Function to use.
minBound,maxBound	The minimum and maximum bounds of the individuals in the particle swarm.

Returns

The best global fitness of the Particle Swarm.

3.10.2.10 saveEndingPopulation()

```
void ParticleSwarm::saveEndingPopulation ( )
```

Saves the ending population solutions to file.

Saves the ending population solutions to file.

3.10.2.11 savePSAnalysis()

```
void ParticleSwarm::savePSAnalysis ( )
```

Saves the Analysis of the Particle Swarm to file.

Saves the Analysis of the Particle Swarm to file.

3.10.2.12 savePSResults()

```
void ParticleSwarm::savePSResults ( )
```

Saves all Particle Swarm Results to file.

Saves all Particle Swarm Results to file.

3.10.2.13 updateParticle()

```
void ParticleSwarm::updateParticle (
             PS_Population & population,
             int particleIndex,
             mt19937 & randGenerator ) [private]
```

Updates a single particle in the population.

Updates a single particle in the population.

Parameters

population	The PS_Population structure that holds the particle swarm population.
particleIndex	The index of the particle in the population to update.
randGenerator	The Mersenne Twister pseudo-random number generator.

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

- ParticleSwarm.h
- ParticleSwarm.cpp

3.11 PS_Analysis Struct Reference

Particle Swarm Analysis Particle Swarm Analysis Structure, to keep track of the analysis performed on each population in the population list.

```
#include <ParticleSwarm.h>
```

Public Attributes

- string header = "Function ID,Average Fitness,Standard Deviation,Range(min),Range(max),Median,Time(ms),Function Calls\n"
- vector< int > functionIDs
- vector< double > avgFunctionFitness
- vector< double > standardDeviation
- vector< vector< double > > ranges
- vector< double > medianFunctionFitness
- vector< double > executionTimes
- vector< int > functionCalls

3.11.1 Detailed Description

Particle Swarm Analysis Particle Swarm Analysis Structure, to keep track of the analysis performed on each population in the population list.

3.11.2 Member Data Documentation

3.11.2.1 avgFunctionFitness

```
vector<double> PS_Analysis::avgFunctionFitness
```

List of the average fitness from function.

3.11.2.2 executionTimes

```
vector<double> PS_Analysis::executionTimes
```

List of execution times in ms for all functions.

3.11.2.3 functionCalls

```
vector<int> PS_Analysis::functionCalls
```

List of the amount of times a function was called.

3.11.2.4 functionIDs

```
vector<int> PS_Analysis::functionIDs
```

List of function IDs.

3.11.2.5 header

```
string PS_Analysis::header = "Function ID,Average Fitness,Standard Deviation,Range(min),Range(max),Median,Time
Calls\n"
```

Header used when saving the data.

3.11.2.6 medianFunctionFitness

```
vector<double> PS_Analysis::medianFunctionFitness
```

List of the Median fitness for each function.

3.11.2.7 ranges

```
vector<vector<double> > PS_Analysis::ranges
```

List of ranges for each fitness function.

3.11.2.8 standardDeviation

```
vector<double> PS_Analysis::standardDeviation
```

List of standard fitness deviations.

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

- ParticleSwarm.h

3.12 PS_Config Struct Reference

Holds all the user defined variables. Particle Swarm Configuration Structure, where all user defined variables that are used to configure the Particle Swarm are stored.

```
#include <ParticleSwarm.h>
```

Public Attributes

- int dimensions
- int popSize
- int iterations
- double k
- double c1
- double c2

3.12.1 Detailed Description

Holds all the user defined variables. Particle Swarm Configuration Structure, where all user defined variables that are used to configure the Particle Swarm are stored.

3.12.2 Member Data Documentation

3.12.2.1 c1

```
double PS_Config::c1
```

Scaling factor to bring velocity closer to personal best.

3.12.2.2 c2

```
double PS_Config::c2
```

Scaling factor to bring velocity closer to global best.

3.12.2.3 dimensions

```
int PS_Config::dimensions
```

Number of dimensions per individual in population.

3.12.2.4 iterations

```
int PS_Config::iterations
```

Maximum number of iterations.

3.12.2.5 k

```
double PS_Config::k
```

Dampening factor for the velocity.

3.12.2.6 popSize

```
int PS_Config::popSize
```

Population size.

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

- ParticleSwarm.h

3.13 PS_Population Struct Reference

Holds all the population information. Particle Swarm Population Structure, holds all the data related to the population of the Particle Swarm.

```
#include <ParticleSwarm.h>
```

Public Attributes

- int functionID
- vector< double > bounds
- int functionCounter = 0
- vector< vector< double > > pop
- vector< vector< double > > velocity
- vector< double > fitness
- vector< vector< double > > pBestInd
- vector< double > pBestFitness
- vector< double > gBestIndividual
- double gBestFitness
- vector< double > gBestFitnessList
- double executionTime = -1.0

3.13.1 Detailed Description

Holds all the population information. Particle Swarm Population Structure, holds all the data related to the population of the Particle Swarm.

3.13.2 Member Data Documentation

3.13.2.1 bounds

```
vector<double> PS_Population::bounds
```

Holds the (min,max) bounds of the values for each individual in the population.

3.13.2.2 executionTime

```
double PS_Population::executionTime = -1.0
```

Time(ms) it took to run the Particle Swarm on this population.

3.13.2.3 fitness

```
vector<double> PS_Population::fitness
```

The fitness for each vector in the population matrix.

3.13.2.4 functionCounter

```
int PS_Population::functionCounter = 0
```

The function counter keeps track of how many times the benchmark function was called.

3.13.2.5 functionID

```
int PS_Population::functionID
```

The ID determines which benchmark function to call.

3.13.2.6 gBestFitness

```
double PS_Population::gBestFitness
```

The global best fitness (of gBestIndividual) of the population.

3.13.2.7 gBestFitnessList

```
vector<double> PS_Population::gBestFitnessList
```

List of the best global fitness values from each iteration.

3.13.2.8 gBestIndividual

```
vector<double> PS_Population::gBestIndividual
```

The global best solution of the population.

3.13.2.9 pBestFitness

```
vector<double> PS_Population::pBestFitness
```

The personal best fitness for each individual in the population.

3.13.2.10 pBestInd

```
vector<vector<double> > PS_Population::pBestInd
```

The personal best solution for each individual in the population.

3.13.2.11 pop

```
vector<vector<double> > PS_Population::pop
```

The population matrix.

3.13.2.12 velocity

```
vector<vector<double> > PS_Population::velocity
```

Velocity matrix holds velocity vectors of each individual in the population.

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

- ParticleSwarm.h

Chapter 4

File Documentation

4.1 BenchmarkFunctions.cpp File Reference

A library of benchmark functions.

```
#include "BenchmarkFunctions.h"
```

Functions

- double schefelsFunc (vector< double > &vect, int size)

    Performs the Schefel's Function on a vector of elements.
- double deJongsFunc (vector< double > &vect, int size)

    Performs the 1st De Jong's Function on a vector of elements.
- double rosenbrockFunc (vector< double > &vect, int size)

    Performs the Rosenbrock Function on a vector of elements.
- double rastriginFunc (vector< double > &vect, int size)

    Performs the Rastrigin Function on a vector of elements.
- double griewangkFunc (vector< double > &vect, int size)

    Performs the Griewangk Function on a vector of elements.
- double sineEnvelopeSineWaveFunc (vector< double > &vect, int size)

    Performs the Sine Envelope Sine Wave Function on a vector of elements.
- double stretchedVSineWaveFunc (vector< double > &vect, int size)

    Performs the Stretched V Sine Wave Function on a vector of elements.
- double ackleysOneFunc (vector< double > &vect, int size)

    Performs the Ackley's One Function on a vector of elements.
- double ackleysTwoFunc (vector< double > &vect, int size)

    Performs the Ackley's Two Function on a vector of elements.
- double eggHolderFunc (vector< double > &vect, int size)

    Performs the Egg Holder Function on a vector of elements.
- double ranaFunc (vector< double > &vect, int size)

    Performs the Rana Function on a vector of elements.
- double pathologicalFunc (vector< double > &vect, int size)

    Performs the Pathological Function on a vector of elements.
- double michalewiczFunc (vector< double > &vect, int size)

Performs the Michalewicz Function on a vector of elements.
- double mastersCosWaveFunc (vector< double > &vect, int size)

  Performs the Masters Cosine Wave Function on a vector of elements.
- double quarticFunc (vector< double > &vect, int size)

  Performs the Quartic Function on a vector of elements.
- double levyFunc (vector< double > &vect, int size)

  Performs the Levy Function on a vector of elements.
- double stepFunc (vector< double > &vect, int size)

  Performs the Step Function on a vector of elements.
- double alpineFunc (vector< double > &vect, int size)

  Performs the Alpine Function on a vector of elements.

4.1.1 Detailed Description

A library of benchmark functions.

Author

Al Timofeyev

Date

April 17, 2019

4.1.2 Function Documentation

4.1.2.1 ackleysOneFunc()

```
double ackleysOneFunc (
            vector< double > & vect,
            int size )
```

Performs the Ackley's One Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.2 ackleysTwoFunc()

```
double ackleysTwoFunc (
             vector< double > & vect,
             int size )
```

Performs the Ackley's Two Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.3 alpineFunc()

```
double alpineFunc (
             vector< double > & vect,
             int size )
```

Performs the Alpine Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.4 deJongsFunc()

```
double deJongsFunc (
             vector< double > & vect,
             int size )
```

Performs the 1st De Jong's Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.5 eggHolderFunc()

```
double eggHolderFunc (
            vector< double > & vect,
            int size )
```

Performs the Egg Holder Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.6 griewangkFunc()

```
double griewangkFunc (
            vector< double > & vect,
            int size )
```

Performs the Griewangk Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.7 levyFunc()

```
double levyFunc (
            vector< double > & vect,
            int size )
```

Performs the Levy Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.8 mastersCosWaveFunc()

```
double mastersCosWaveFunc (
            vector< double > & vect,
            int size )
```

Performs the Masters Cosine Wave Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.9 michalewiczFunc()

```
double michalewiczFunc (
            vector< double > & vect,
            int size )
```

Performs the Michalewicz Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.10 pathologicalFunc()

```
double pathologicalFunc (
            vector< double > & vect,
            int size )
```

Performs the Pathological Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.11 quarticFunc()

```
double quarticFunc (
            vector< double > & vect,
            int size )
```

Performs the Quartic Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.12 ranaFunc()

```
double ranaFunc (
            vector< double > & vect,
            int size )
```

Performs the Rana Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.13 rastriginFunc()

```
double rastriginFunc (
            vector< double > & vect,
            int size )
```

Performs the Rastrigin Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.14 rosenbrockFunc()

```
double rosenbrockFunc (
            vector< double > & vect,
            int size )
```

Performs the Rosenbrock Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.15 schefelsFunc()

```
double schefelsFunc (
            vector< double > & vect,
            int size )
```

Performs the Schefel's Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.16 sineEnvelopeSineWaveFunc()

```
double sineEnvelopeSineWaveFunc (
            vector< double > & vect,
            int size )
```

Performs the Sine Envelope Sine Wave Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.17 stepFunc()

```
double stepFunc (
            vector< double > & vect,
            int size )
```

Performs the Step Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.1.2.18 stretchedVSineWaveFunc()

```
double stretchedVSineWaveFunc (
            vector< double > & vect,
            int size )
```

Performs the Stretched V Sine Wave Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2 BenchmarkFunctions.h File Reference

A library of benchmark functions.

```
#include <vector>
#include <math.h>
#include <cmath>
```

Functions

- double schefelsFunc (vector< double > &vect, int size)

  Performs the Schefel's Function on a vector of elements.
- double deJongsFunc (vector< double > &vect, int size)

  Performs the 1st De Jong's Function on a vector of elements.
- double rosenbrockFunc (vector< double > &vect, int size)

  Performs the Rosenbrock Function on a vector of elements.
- double rastriginFunc (vector< double > &vect, int size)

  Performs the Rastrigin Function on a vector of elements.
- double griewangkFunc (vector< double > &vect, int size)

  Performs the Griewangk Function on a vector of elements.
- double sineEnvelopeSineWaveFunc (vector< double > &vect, int size)

  Performs the Sine Envelope Sine Wave Function on a vector of elements.
- double stretchedVSineWaveFunc (vector< double > &vect, int size)

  Performs the Stretched V Sine Wave Function on a vector of elements.
- double ackleysOneFunc (vector< double > &vect, int size)

  Performs the Ackley's One Function on a vector of elements.
- double ackleysTwoFunc (vector< double > &vect, int size)

  Performs the Ackley's Two Function on a vector of elements.
- double eggHolderFunc (vector< double > &vect, int size)

  Performs the Egg Holder Function on a vector of elements.
- double ranaFunc (vector< double > &vect, int size)

Performs the Rana Function on a vector of elements.

- double pathologicalFunc (vector< double > &vect, int size)

  Performs the Pathological Function on a vector of elements.
- double michalewiczFunc (vector< double > &vect, int size)

  Performs the Michalewicz Function on a vector of elements.
- double mastersCosWaveFunc (vector< double > &vect, int size)

  Performs the Masters Cosine Wave Function on a vector of elements.
- double quarticFunc (vector< double > &vect, int size)

  Performs the Quartic Function on a vector of elements.
- double levyFunc (vector< double > &vect, int size)

  Performs the Levy Function on a vector of elements.
- double stepFunc (vector< double > &vect, int size)

  Performs the Step Function on a vector of elements.
- double alpineFunc (vector< double > &vect, int size)

  Performs the Alpine Function on a vector of elements.

4.2.1 Detailed Description

A library of benchmark functions.

Author

Al Timofeyev

Date

April 17, 2019

4.2.2 Function Documentation

4.2.2.1 ackleysOneFunc()

```
double ackleysOneFunc (
            vector< double > & vect,
            int size )
```

Performs the Ackley's One Function on a vector of elements.

Performs the Ackley's One Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.2 ackleysTwoFunc()

```
double ackleysTwoFunc (
            vector< double > & vect,
            int size )
```

Performs the Ackley's Two Function on a vector of elements.

Performs the Ackley's Two Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.3 alpineFunc()

```
double alpineFunc (
            vector< double > & vect,
            int size )
```

Performs the Alpine Function on a vector of elements.

Performs the Alpine Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.4 deJongsFunc()

```
double deJongsFunc (
             vector< double > & vect,
             int size )
```

Performs the 1st De Jong's Function on a vector of elements.

Performs the 1st De Jong's Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.5 eggHolderFunc()

```
double eggHolderFunc (
             vector< double > & vect,
             int size )
```

Performs the Egg Holder Function on a vector of elements.

Performs the Egg Holder Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.6 griewangkFunc()

```
double griewangkFunc (
             vector< double > & vect,
             int size )
```

Performs the Griewangk Function on a vector of elements.

Performs the Griewangk Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.7 levyFunc()

```
double levyFunc (
            vector< double > & vect,
            int size )
```

Performs the Levy Function on a vector of elements.

Performs the Levy Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.8 mastersCosWaveFunc()

```
double mastersCosWaveFunc (
            vector< double > & vect,
            int size )
```

Performs the Masters Cosine Wave Function on a vector of elements.

Performs the Masters Cosine Wave Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.9 michalewiczFunc()

```
double michalewiczFunc (
             vector< double > & vect,
             int size )
```

Performs the Michalewicz Function on a vector of elements.

Performs the Michalewicz Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.10 pathologicalFunc()

```
double pathologicalFunc (
             vector< double > & vect,
             int size )
```

Performs the Pathological Function on a vector of elements.

Performs the Pathological Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.11 quarticFunc()

```
double quarticFunc (
             vector< double > & vect,
             int size )
```

Performs the Quartic Function on a vector of elements.

Performs the Quartic Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.12 ranaFunc()

```
double ranaFunc (
             vector< double > & vect,
             int size )
```

Performs the Rana Function on a vector of elements.

Performs the Rana Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.13 rastriginFunc()

```
double rastriginFunc (
             vector< double > & vect,
             int size )
```

Performs the Rastrigin Function on a vector of elements.

Performs the Rastrigin Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.14 rosenbrockFunc()

```
double rosenbrockFunc (
            vector< double > & vect,
            int size )
```

Performs the Rosenbrock Function on a vector of elements.

Performs the Rosenbrock Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.15 schefelsFunc()

```
double schefelsFunc (
            vector< double > & vect,
            int size )
```

Performs the Schefel's Function on a vector of elements.

Performs the Schefel's Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.16 sineEnvelopeSineWaveFunc()

```
double sineEnvelopeSineWaveFunc (
            vector< double > & vect,
            int size )
```

Performs the Sine Envelope Sine Wave Function on a vector of elements.

Performs the Sine Envelope Sine Wave Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.17 stepFunc()

```
double stepFunc (
            vector< double > & vect,
            int size )
```

Performs the Step Function on a vector of elements.

Performs the Step Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.2.2.18 stretchedVSineWaveFunc()

```
double stretchedVSineWaveFunc (
            vector< double > & vect,
            int size )
```

Performs the Stretched V Sine Wave Function on a vector of elements.

Performs the Stretched V Sine Wave Function on a vector of elements.

Parameters

vect	The vector of elements on which to perform calculations.
size	The number of elements in vector.

Returns

The results of the calculations (fitness).

4.3 FireflyAlgorithm.cpp File Reference

This is an implementation of a Firefly Algorithm.

```
#include "FireflyAlgorithm.h"
```

4.3.1 Detailed Description

This is an implementation of a Firefly Algorithm.

Author

Al Timofeyev

Date

May 14, 2019

4.4 FireflyAlgorithm.h File Reference

This is an implementation of a Firefly Algorithm.

```
#include <fstream>
#include <chrono>
#include "SwarmUtilities.h"
```

Classes

- struct FA_Config

  Holds all the user defined variables. Firefly Algorithm Configuration Structure, where all user defined variables that are used to configure the Firefly Algorithm are stored.
- struct FA_Population

  Holds all the population information. Firefly Algorithm Population Structure, holds all the data related to the population of the Firefly Algorithm.
- struct FA_Analysis

  Firefly Algorithm Analysis Firefly Algorithm Analysis Structure, to keep track of the analysis performed on each population in the population list.
- class FireflyAlgorithm

4.4.1 Detailed Description

This is an implementation of a Firefly Algorithm.

Author

Al Timofeyev

Date

May 14, 2019

4.5 HarmonySearch.h File Reference

This is an implementation of a Harmony Search.

```
#include <fstream>
#include <chrono>
#include "SwarmUtilities.h"
```

Classes

- struct HS_Config

  Holds all the user defined variables. Harmony Search Configuration Structure, where all user defined variables that are used to configure the Harmony Search are stored.
- struct HS_Population

  Holds all the population information. Harmony Search Population Structure, holds all the data related to the population of the Harmony Search.
- struct HS_Analysis

  Harmony Search Analysis Harmony Search Analysis Structure, to keep track of the analysis performed on each population in the population list.
- struct HS_Analysis_Worst

  Harmony Search Analysis Worst Harmony Search Analysis Worst Structure, to keep track of the analysis performed on the list of worst solutions of each population in the population list.
- class HarmonySearch

4.5.1 Detailed Description

This is an implementation of a Harmony Search.

Author

Al Timofeyev

Date

May 16, 2019

4.6 ParticleSwarm.cpp File Reference

This is an implementation of a Particle Swarm.

```
#include "ParticleSwarm.h"
```

4.6.1 Detailed Description

This is an implementation of a Particle Swarm.

Author

Al Timofeyev

Date

May 10, 2019

4.7 ParticleSwarm.h File Reference

This is an implementation of a Particle Swarm.

```
#include <fstream>
#include <chrono>
#include "SwarmUtilities.h"
```

Classes

- struct PS_Config

  Holds all the user defined variables. Particle Swarm Configuration Structure, where all user defined variables that are used to configure the Particle Swarm are stored.
- struct PS_Population

  Holds all the population information. Particle Swarm Population Structure, holds all the data related to the population of the Particle Swarm.
- struct PS_Analysis

  Particle Swarm Analysis Particle Swarm Analysis Structure, to keep track of the analysis performed on each population in the population list.
- class ParticleSwarm

4.7.1 Detailed Description

This is an implementation of a Particle Swarm.

Author

Al Timofeyev

Date

May 10, 2019

4.8 SwarmUtilities.cpp File Reference

Utilities library for swarm algorithms.

```
#include "SwarmUtilities.h"
```

Functions

- void printAllFunctionIDs ()

  Prints all the possible Function IDs to the screen.
- vector< vector< double > > createMatrix (int rows, int columns, double minBound, double maxBound)

  Creates a matrix of doubles using Mersenne Twister.
- vector< vector< double > > createMatrixMT (int rows, int columns, double minBound, double maxBound, mt19937 &randGenerator)

  Creates a matrix of doubles using Mersenne Twister.
- double calculateFitnessOfVector (vector< double > &vect, int functionID)

  Calculates the fitness of a vector.
- vector< double > calculateFitnessOfMatrix (vector< vector< double >> matrix, int functionID)

  Calculates the fitness of all vectors of a matrix.
- double calculateAverage (vector< double > vect)

  Calculates the average value of a vector of doubles.
- double calculateStandardDeviation (vector< double > vect)

  Calculates the standard deviation value of a vector of doubles.
- void quicksort (vector< double > &fitnessList, vector< vector< double >> &matrix, int L, int R)

  Sorts a matrix and its fitness vector based on the fitness.
- void swap (vector< double > &fitnessList, vector< vector< double >> &matrix, int x, int y)

  Swaps the fitness' and their corresponding vectors in the matrix.
- void quicksort (vector< double > &vec, int L, int R)

  A normal Quicksort implementation for vector arrays of doubles.
- void swap (vector< double > &v, int x, int y)

  Swaps two values of a vector array of doubles.

4.8.1 Detailed Description

Utilities library for swarm algorithms.

Author

Al Timofeyev

Date

May 10, 2019

4.8.2 Function Documentation

4.8.2.1 calculateAverage()

```
double calculateAverage (
            vector< double > vect )
```

Calculates the average value of a vector of doubles.

Parameters

vect	The vector of doubles.

Returns

The average value of the vector.

4.8.2.2 calculateFitnessOfMatrix()

```
vector<double> calculateFitnessOfMatrix (
            vector< vector< double >> matrix,
            int functionID )
```

Calculates the fitness of all vectors of a matrix.

Calculates the fitness of all the vectors of the matrix stored All the fitness results are stored in the fitness vector variable.

Parameters

matrix	The matrix that holds all the vectors for calculating the fitness.
functionID	The ID of the function to use for calculating the fitness.

Returns

A vector of fitness values.

4.8.2.3 calculateFitnessOfVector()

```
double calculateFitnessOfVector (
            vector< double > & vect,
            int functionID )
```

Calculates the fitness of a vector.

The fitness of a vector is calculated by the Benchmark Function referenced by the functionID.

Note

This function makes a call to BenchmarkFunctions.h.

Parameters

vect	The vector of elements on which the Benchmark Functions operate.
functionID	The ID that references which Benchmark Function to use.

Returns

The fitness of the vector.

4.8.2.4 calculateStandardDeviation()

```
double calculateStandardDeviation (
            vector< double > vect )
```

Calculates the standard deviation value of a vector of doubles.

Parameters

vect	The vector of doubles.

Returns

The standard deviation value of the vector.

4.8.2.5 createMatrix()

```
vector<vector<double> > createMatrix (
            int rows,
            int columns,
            double minBound,
            double maxBound )
```

Creates a matrix of doubles using Mersenne Twister.

A matrix is constructed using the Mersenne Twister in the <random> library with the user-specified min/max boundaries.

Parameters

rows	The number of vectors in the matrix.
columns	The number of elements in each vector of the matrix.
minBound,maxBound	The max/min boundaries are the range in which to generate numbers.

Returns

The fully constructed matrix of doubles.

4.8.2.6 createMatrixMT()

```
vector<vector<double> > createMatrixMT (
            int rows,
            int columns,
            double minBound,
            double maxBound,
            mt19937 & randGenerator )
```

Creates a matrix of doubles using Mersenne Twister.

A matrix is constructed using the Mersenne Twister in the <random> library with the user-specified min/max boundaries.

Parameters

rows	The number of vectors in the matrix.
columns	The number of elements in each vector of the matrix.
minBound,maxBound	The max/min boundaries are the range in which to generate numbers.
randGenerator	The Mersenne Twister pseudo-random number generator.

Returns

The fully constructed matrix of doubles.

4.8.2.7 printAllFunctionIDs()

```
void printAllFunctionIDs ( )
```

Prints all the possible Function IDs to the screen.

Prints all possible Function ID, as well as the functions they reference, to the screen.

4.8.2.8 quicksort() [1/2]

```
void quicksort (
            vector< double > & fitnessList,
            vector< vector< double >> & matrix,
            int L,
            int R )
```

Sorts a matrix and its fitness vector based on the fitness.

Note

Sorted in Ascending Order.
Smallest (minimum) fitness gets moved to index 0, along with its vector from matrix.
Largest (maximum) fitness gets moved to the last index, along with its vector from matrix.

Parameters

fitnessList	The list of fitness values that correspond to each row of the matrix.
matrix	A matrix of double values.
L	The starting index for the quicksort (inclusive).
R	The ending index for the quicksort (inclusive).

4.8.2.9 quicksort() [2/2]

```
void quicksort (
            vector< double > & vec,
            int L,
            int R )
```

A normal Quicksort implementation for vector arrays of doubles.

Note

Sorted in Ascending Order.
Smallest value gets moved to index 0.
Largest value gets moved to the last index.

Parameters

vec	Vector array of doubles.
L	The starting index for the quicksort (inclusive).
R	The ending index for the quicksort (inclusive).

4.8.2.10 swap() [1/2]

```
void swap (
             vector< double > & fitnessList,
             vector< vector< double >> & matrix,
             int x,
             int y )
```

Swaps the fitness' and their corresponding vectors in the matrix.

Parameters

fitnessList	The list of fitness values that correspond to each row of the matrix.
matrix	A matrix of double values.
x	The 1st index of the fitness/vector for the swap.
y	The 2nd index of the fitness/vector for the swap.

4.8.2.11 swap() [2/2]

```
void swap (
             vector< double > & v,
             int x,
             int y )
```

Swaps two values of a vector array of doubles.

Parameters

v	The vector in which values are swapped.
x	The 1st index of the fitness/vector for the swap.
y	The 2nd index of the fitness/vector for the swap.

4.9 SwarmUtilities.h File Reference

Utilities library for swarm algorithms.

```
#include <iostream>
#include <vector>
#include <cmath>
#include <random>
#include "BenchmarkFunctions.h"
```

Functions

- void printAllFunctionIDs ()

    Prints all the possible Function IDs to the screen.
- vector< vector< double > > createMatrix (int rows, int columns, double minBound, double maxBound)

    Creates a matrix of doubles using Mersenne Twister.
- vector< vector< double > > createMatrixMT (int rows, int columns, double minBound, double maxBound, mt19937 &randGenerator)

    Creates a matrix of doubles using Mersenne Twister.
- double calculateFitnessOfVector (vector< double > &vect, int functionID)

    Calculates the fitness of a vector.
- vector< double > calculateFitnessOfMatrix (vector< vector< double >> matrix, int functionID)

    Calculates the fitness of all vectors of a matrix.
- double calculateAverage (vector< double > vect)

    Calculates the average value of a vector of doubles.
- double calculateStandardDeviation (vector< double > vect)

    Calculates the standard deviation value of a vector of doubles.
- void quicksort (vector< double > &fitnessList, vector< vector< double >> &matrix, int L, int R)

    Sorts a matrix and its fitness vector based on the fitness.
- void swap (vector< double > &fitnessList, vector< vector< double >> &matrix, int x, int y)

    Swaps the fitness' and their corresponding vectors in the matrix.
- void quicksort (vector< double > &vec, int L, int R)

    A normal Quicksort implementation for vector arrays of doubles.
- void swap (vector< double > &v, int x, int y)

    Swaps two values of a vector array of doubles.

4.9.1 Detailed Description

Utilities library for swarm algorithms.

Author

Al Timofeyev

Date

May 10, 2019

4.9.2 Function Documentation

4.9.2.1 calculateAverage()

```
double calculateAverage (
            vector< double > vect )
```

Calculates the average value of a vector of doubles.

Calculates the average value of a vector of doubles.

Parameters

vect	The vector of doubles.

Returns

The average value of the vector.

4.9.2.2 calculateFitnessOfMatrix()

```
vector<double> calculateFitnessOfMatrix (
             vector< vector< double >> matrix,
             int functionID )
```

Calculates the fitness of all vectors of a matrix.

Calculates the fitness of all vectors in matrix.

Calculates the fitness of all the vectors of the matrix stored All the fitness results are stored in the fitness vector variable.

Parameters

matrix	The matrix that holds all the vectors for calculating the fitness.
functionID	The ID of the function to use for calculating the fitness.

Returns

A vector of fitness values.

4.9.2.3 calculateFitnessOfVector()

```
double calculateFitnessOfVector (
             vector< double > & vect,
             int functionID )
```

Calculates the fitness of a vector.

Calculates the fitness of a single vector.

The fitness of a vector is calculated by the Benchmark Function referenced by the functionID.

Note

This function makes a call to BenchmarkFunctions.h.

Parameters

vect	The vector of elements on which the Benchmark Functions operate.
functionID	The ID that references which Benchmark Function to use.

Returns

The fitness of the vector.

4.9.2.4 calculateStandardDeviation()

```
double calculateStandardDeviation (
            vector< double > vect )
```

Calculates the standard deviation value of a vector of doubles.

Calculates the standard deviation value of a vector of doubles.

Parameters

vect	The vector of doubles.

Returns

The standard deviation value of the vector.

4.9.2.5 createMatrix()

```
vector<vector<double> > createMatrix (
            int rows,
            int columns,
            double minBound,
            double maxBound )
```

Creates a matrix of doubles using Mersenne Twister.

Creates a matrix with the given min/max bound for the given number of rows/columns.

A matrix is constructed using the Mersenne Twister in the <random> library with the user-specified min/max boundaries.

Parameters

rows	The number of vectors in the matrix.
columns	The number of elements in each vector of the matrix.
minBound,maxBound	The max/min boundaries are the range in which to generate numbers.

Returns

The fully constructed matrix of doubles.

4.9.2.6 createMatrixMT()

```
vector<vector<double> > createMatrixMT (
            int rows,
            int columns,
            double minBound,
            double maxBound,
            mt19937 & randGenerator )
```

Creates a matrix of doubles using Mersenne Twister.

Creates a matrix with the given min/max bound for the given number of rows/columns.

A matrix is constructed using the Mersenne Twister in the <random> library with the user-specified min/max boundaries.

Parameters

rows	The number of vectors in the matrix.
columns	The number of elements in each vector of the matrix.
minBound,maxBound	The max/min boundaries are the range in which to generate numbers.
randGenerator	The Mersenne Twister pseudo-random number generator.

Returns

The fully constructed matrix of doubles.

4.9.2.7 printAllFunctionIDs()

```
void printAllFunctionIDs ( )
```

Prints all the possible Function IDs to the screen.

Prints all the possible Function IDs to the screen.

Prints all possible Function ID, as well as the functions they reference, to the screen.

4.9.2.8 quicksort() [1/2]

```
void quicksort (
            vector< double > & fitnessList,
            vector< vector< double >> & matrix,
            int L,
            int R )
```

Sorts a matrix and its fitness vector based on the fitness.

Special Quicksort implementation for fitness/matrices.

Note

Sorted in Ascending Order.
Smallest (minimum) fitness gets moved to index 0, along with its vector from matrix.
Largest (maximum) fitness gets moved to the last index, along with its vector from matrix.

Parameters

fitnessList	The list of fitness values that correspond to each row of the matrix.
matrix	A matrix of double values.
L	The starting index for the quicksort (inclusive).
R	The ending index for the quicksort (inclusive).

4.9.2.9 quicksort() [2/2]

```
void quicksort (
            vector< double > & vec,
            int L,
            int R )
```

A normal Quicksort implementation for vector arrays of doubles.

Normal Quicksort implementation for vector arrays.

Note

Sorted in Ascending Order.
Smallest value gets moved to index 0.
Largest value gets moved to the last index.

Parameters

vec	Vector array of doubles.
L	The starting index for the quicksort (inclusive).
R	The ending index for the quicksort (inclusive).

4.9.2.10 swap() [1/2]

```
void swap (
             vector< double > & fitnessList,
             vector< vector< double >> & matrix,
             int x,
             int y )
```

Swaps the fitness' and their corresponding vectors in the matrix.

Swap function for the Quicksort.

Parameters

fitnessList	The list of fitness values that correspond to each row of the matrix.
matrix	A matrix of double values.
x	The 1st index of the fitness/vector for the swap.
y	The 2nd index of the fitness/vector for the swap.

4.9.2.11 swap() [2/2]

```
void swap (
             vector< double > & v,
             int x,
             int y )
```

Swaps two values of a vector array of doubles.

Parameters

v	The vector in which values are swapped.
x	The 1st index of the fitness/vector for the swap.
y	The 2nd index of the fitness/vector for the swap.

4.10 utilities.cpp File Reference

This utilities file is used as a helper file for ProcessFunctions.h and SearchAlgorithms.h, and to create matricies using the Mersenne Twister.

```
#include "utilities.h"
```

Functions

- vector< double > parseStringDbl (string str, string delimiter)

  Parses a string of numbers into a vector of doubles.
- vector< int > parseStringInt (string str, string delimiter)

  Parses a string of numbers into a vector of integers.
- vector< string > parseStringStr (string str, string delimiter)

  Parses a string of elements into a vector of strings.
- void prepForFunctionMatrix (vector< double > &setup)

  Resizes the vector to size 3.

4.10.1 Detailed Description

This utilities file is used as a helper file for ProcessFunctions.h and SearchAlgorithms.h, and to create matricies using the Mersenne Twister.

Author

Al Timofeyev

Date

April 15, 2019

4.10.2 Function Documentation

4.10.2.1 parseStringDbl()

```
vector<double> parseStringDbl (
            string str,
            string delimiter )
```

Parses a string of numbers into a vector of doubles.

Constructs and returns a vector of doubles, given a string list of numbers and a delimiter.

Note

The input string str MUST be a list of doubles!

Parameters

str	A string list of numbers.
delimiter	A string of character(s) used to separate the numbers in the string list.

Returns

Returns a vector filled with doubles that were extracted from the string list.

4.10.2.2 parseStringInt()

```
vector<int> parseStringInt (
            string str,
            string delimiter )
```

Parses a string of numbers into a vector of integers.

Constructs and returns a vector of integers, given a string list of numbers and a delimiter.

Note

The input string list MUST be a list of integers!

Parameters

str	A string list of numbers.
delimiter	A string of character(s) used to separate the numbers in the string list.

Returns

Returns a vector filled with integers that were extracted from the string list.

4.10.2.3 parseStringStr()

```
vector<string> parseStringStr (
            string str,
            string delimiter )
```

Parses a string of elements into a vector of strings.

Constructs and returns a vector of strings, given a string list of elements and a delimiter.

Parameters

str	A string list of characters.
delimiter	A string of character(s) used to separate the numbers in the string list.

Returns

Returns a vector filled with integers that were extracted from the string list.

4.10.2.4 prepForFunctionMatrix()

```
void prepForFunctionMatrix (
            vector< double > & setup )
```

Resizes the vector to size 3.

Resizes the given vector to size three in order to prep it for the matrix of a function. Because to generate a matrix, you only need 3 values: function ID, minimum bound, maximum bound.

Parameters

setup	The vector that's going to be resized for the matrix setup.

4.11 utilities.h File Reference

This utilities file is used as a helper file for ProcessFunctions.h and SearchAlgorithms.h, and to create matricies using the Mersenne Twister.

```
#include <iostream>
#include <string>
#include <string.h>
#include <vector>
#include <cmath>
```

Functions

- vector< double > parseStringDbl (string str, string delimiter)

    Parses a string of numbers into a vector of doubles.
- vector< int > parseStringInt (string str, string delimiter)

    Parses a string of numbers into a vector of integers.
- vector< string > parseStringStr (string str, string delimiter)

    Parses a string of elements into a vector of strings.
- void prepForFunctionMatrix (vector< double > &setup)

    Resizes the vector to size 3.

4.11.1 Detailed Description

This utilities file is used as a helper file for ProcessFunctions.h and SearchAlgorithms.h, and to create matricies using the Mersenne Twister.

Author

Al Timofeyev

Date

April 15, 2019

4.11.2 Function Documentation

4.11.2.1 parseStringDbl()

```
vector<double> parseStringDbl (
            string str,
            string delimiter )
```

Parses a string of numbers into a vector of doubles.

Parses a string of numbers into a vector of doubles.

Constructs and returns a vector of doubles, given a string list of numbers and a delimiter.

Note

The input string str MUST be a list of doubles!

Parameters

str	A string list of numbers.
delimiter	A string of character(s) used to separate the numbers in the string list.

Returns

Returns a vector filled with doubles that were extracted from the string list.

4.11.2.2 parseStringInt()

```
vector<int> parseStringInt (
            string str,
            string delimiter )
```

Parses a string of numbers into a vector of integers.

Parses a string of numbers into a vector of integers.

Constructs and returns a vector of integers, given a string list of numbers and a delimiter.

Note

The input string list MUST be a list of integers!

Parameters

str	A string list of numbers.
delimiter	A string of character(s) used to separate the numbers in the string list.

Returns

Returns a vector filled with integers that were extracted from the string list.

4.11.2.3 parseStringStr()

```
vector<string> parseStringStr (
             string str,
             string delimiter )
```

Parses a string of elements into a vector of strings.

Parses a string of characters into a vector of strings.

Constructs and returns a vector of strings, given a string list of elements and a delimiter.

Parameters

str	A string list of characters.
delimiter	A string of character(s) used to separate the numbers in the string list.

Returns

Returns a vector filled with integers that were extracted from the string list.

4.11.2.4 prepForFunctionMatrix()

```
void prepForFunctionMatrix (
             vector< double > & setup )
```

Resizes the vector to size 3.

Preps the setup vector for the matrix of a function by resizing to size 3.

Resizes the given vector to size three in order to prep it for the matrix of a function. Because to generate a matrix, you only need 3 values: function ID, minimum bound, maximum bound.

Parameters

setup	The vector that's going to be resized for the matrix setup.

Index