

My Project

Generated by Doxygen 1.8.13

Contents

1	File Index	1
1.1	File List	1
2	File Documentation	3
2.1	firefly.cpp File Reference	3
2.2	functions.cpp File Reference	3
2.2.1	Detailed Description	4
2.2.2	Function Documentation	4
2.2.2.1	ackleyOne()	5
2.2.2.2	ackleyTwo()	6
2.2.2.3	alpine()	6
2.2.2.4	dejong()	7
2.2.2.5	eggholder()	7
2.2.2.6	griewank()	7
2.2.2.7	levy()	8
2.2.2.8	mastersCosWave()	8
2.2.2.9	michalewicz()	8
2.2.2.10	pathological()	9
2.2.2.11	quartic()	9
2.2.2.12	rana()	10
2.2.2.13	rastrigin()	10
2.2.2.14	rosenbrok()	10
2.2.2.15	schwefel()	11
2.2.2.16	sinEnvlSinWave()	11

2.2.2.17	step()	11
2.2.2.18	stretchVSinWave()	12
2.2.2.19	w()	12
2.3	harmony.cpp File Reference	13
2.3.1	Detailed Description	13
2.3.2	Function Documentation	13
2.3.2.1	addNewHarmony()	13
2.3.2.2	adjustPitch()	14
2.3.2.3	harmony()	14
2.3.2.4	initializeHS()	14
2.3.2.5	runHarmony()	15
2.3.2.6	updateRecordsFF()	15
2.4	main.cpp File Reference	15
2.4.1	Detailed Description	16
2.4.2	Function Documentation	16
2.4.2.1	main()	16
2.5	Parameters.cpp File Reference	17
2.5.1	Detailed Description	17
2.5.2	Function Documentation	17
2.5.2.1	compareDoubles()	17
2.5.2.2	getParameters()	18
2.6	particleSwarm.cpp File Reference	18
2.6.1	Detailed Description	19
2.6.2	Function Documentation	19
2.6.2.1	initializePSO()	19
2.6.2.2	particleSwarm()	19
2.6.2.3	runParticleSwarm()	20
2.6.2.4	updateFitness()	20
2.6.2.5	updateGlobalBest()	20
2.6.2.6	updateParticles()	21
2.6.2.7	updatePersonalBest()	21
2.6.2.8	updateRecords()	21
2.6.2.9	updateVelocity()	22
2.7	Population.cpp File Reference	22
2.7.1	Detailed Description	22
2.8	RecordKeeper.cpp File Reference	22
2.8.1	Detailed Description	23
2.9	runFuncs.cpp File Reference	23
2.9.1	Detailed Description	23
2.9.2	Function Documentation	24
2.9.2.1	runSolution()	24

Chapter 1

File Index

1.1 File List

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

firefly.cpp	Optimizes a population using the Firefly algorithm	3
functions.cpp	This file contains the various functions being used to test optimization	3
harmony.cpp	13
main.cpp	Project 4 for CS470. This program run a population of solution vectors through Particle Swarm algorithm, Firefly algorithm, and Harmony Search algorithm then print all the results to csv files	15
Parameters.cpp	C++ file that provides utilities. This file contains all of the various utilities that provide extra functions, such as reading in parameters, and setting values in vectors	17
particleSwarm.cpp	Optimizes a population using the Particle Swarm algorithm	18
Population.cpp	Holds optimization information	22
RecordKeeper.cpp	Holds information about multiple optiization experimentations	22
runFuncs.cpp	Passes a solution vector through a function and returns the resulting value	23

Chapter 2

File Documentation

2.1 firefly.cpp File Reference

Optimizes a population using the Firefly algorithm.

```
#include <ctime>
#include <iostream>
#include <random>
#include <thread>
#include "firefly.h"
#include "runFuncs.h"
Include dependency graph for firefly.cpp:
```

2.2 functions.cpp File Reference

This file contains the various functions being used to test optimization.

```
#include <cmath>
#include "functions.h"
Include dependency graph for functions.cpp:
```

Functions

- double [schwefel](#) (double *vec, int n)
Schwefel's Function.
- double [dejong](#) (double *vec, int n)
De Jong's first function.
- double [rosenbrok](#) (double *vec, int n)
Rosenbrok's function.
- double [rastrigin](#) (double *vec, int n)
Rastrigin's function.
- double [griewank](#) (double *vec, int n)
Griewank's function.
- double [sinEnvlSinWave](#) (double *vec, int n)
Sine Envelope Sine Wave function.

- double [stretchVSinWave](#) (double *vec, int n)
Stretched V Sine Wave function.
- double [ackleyOne](#) (double *vec, int n)
Ackley's first function.
- double [ackleyTwo](#) (double *vec, int n)
Ackley's second function.
- double [eggholder](#) (double *vec, int n)
Eggholder function.
- double [rana](#) (double *vec, int n)
Rana's function.
- double [pathological](#) (double *vec, int n)
Pathological function.
- double [michalewicz](#) (double *vec, int n)
Machalewicz's function.
- double [mastersCosWave](#) (double *vec, int n)
Master's Cosine Wave function.
- double [quartic](#) (double *vec, int n)
Quartic function.
- double [w](#) (double *vec, int n)
W: A helper function for Levy's function.
- double [levy](#) (double *vec, int n)
Levy's function.
- double [step](#) (double *vec, int n)
Step function.
- double [alpine](#) (double *vec, int n)
Alpine function.

2.2.1 Detailed Description

This file contains the various functions being used to test optimization.

Author

Matthew Harker

Version

1.0

Date

2019-05-20

Copyright

Copyright (c) 2019

2.2.2 Function Documentation

2.2.2.1 ackleyOne()

```
double ackleyOne (  
    double * vec,  
    int n )
```

Ackley's first function.

Parameters

<i>vec</i>	The vector of values the function will process.
<i>n</i>	The size of the vector.

Returns

double The result of the function.

2.2.2.2 ackleyTwo()

```
double ackleyTwo (
    double * vec,
    int n )
```

Ackley's second function.

Parameters

<i>vec</i>	The vector of values the function will process.
<i>n</i>	The size of the vector.

Returns

double The result of the function.

2.2.2.3 alpine()

```
double alpine (
    double * vec,
    int n )
```

Alpine function.

Parameters

<i>vec</i>	The vector of values the function will process.
<i>n</i>	The size of the vector.

Returns

double The result of the function.

2.2.2.4 dejong()

```
double dejong (
    double * vec,
    int n )
```

De Jong's first function.

Parameters

<i>vec</i>	The vector of values the function will process.
<i>n</i>	The size of the vector.

Returns

double The result of the function.

2.2.2.5 eggholder()

```
double eggholder (
    double * vec,
    int n )
```

Eggholder function.

Parameters

<i>vec</i>	The vector of values the function will process.
<i>n</i>	The size of the vector.

Returns

double The result of the function.

2.2.2.6 griewank()

```
double griewank (
    double * vec,
    int n )
```

Griewank's function.

Parameters

<i>vec</i>	The vector of values the function will process.
<i>n</i>	The size of the vector.

Returns

double The result of the function.

2.2.2.7 levy()

```
double levy (  
    double * vec,  
    int n )
```

Levy's function.

Parameters

<i>vec</i>	The vector of values the function will process.
<i>n</i>	The size of the vector.

Returns

double The result of the function.

2.2.2.8 mastersCosWave()

```
double mastersCosWave (  
    double * vec,  
    int n )
```

Master's Cosine Wave function.

Parameters

<i>vec</i>	The vector of values the function will process.
<i>n</i>	The size of the vector.

Returns

double The result of the function.

2.2.2.9 michalewicz()

```
double michalewicz (  
    double * vec,  
    int n )
```

Machalewicz's function.

Parameters

<i>vec</i>	The vector of values the function will process.
<i>n</i>	The size of the vector.

Returns

double The result of the function.

2.2.2.10 pathological()

```
double pathological (  
    double * vec,  
    int n )
```

Pathological function.

Parameters

<i>vec</i>	The vector of values the function will process.
<i>n</i>	The size of the vector.

Returns

double The result of the function.

2.2.2.11 quartic()

```
double quartic (  
    double * vec,  
    int n )
```

Quartic function.

Parameters

<i>vec</i>	The vector of values the function will process.
<i>n</i>	The size of the vector.

Returns

double The result of the function.

2.2.2.12 rana()

```
double rana (
    double * vec,
    int n )
```

Rana's function.

Parameters

<i>vec</i>	The vector of values the function will process.
<i>n</i>	The size of the vector.

Returns

double The result of the function.

2.2.2.13 rastrigin()

```
double rastrigin (
    double * vec,
    int n )
```

Rastrigin's function.

Parameters

<i>vec</i>	The vector of values the function will process.
<i>n</i>	The size of the vector.

Returns

double The result of the function.

2.2.2.14 rosenbrok()

```
double rosenbrok (
    double * vec,
    int n )
```

Rosenbrok's function.

Parameters

<i>vec</i>	The vector of values the function will process.
<i>n</i>	The size of the vector.

Returns

double The result of the function.

2.2.2.15 schwefel()

```
double schwefel (
    double * vec,
    int n )
```

Schwefel's Function.

Parameters

<i>vec</i>	The vector of values the function will process.
<i>n</i>	The size of the vector.

Returns

double The result of the function.

2.2.2.16 sinEnvlSinWave()

```
double sinEnvlSinWave (
    double * vec,
    int n )
```

Sine Envelope Sine Wave function.

Parameters

<i>vec</i>	The vector of values the function will process.
<i>n</i>	The size of the vector.

Returns

double The result of the function.

2.2.2.17 step()

```
double step (
    double * vec,
    int n )
```

Step function.

Parameters

<i>vec</i>	The vector of values the function will process.
<i>n</i>	The size of the vector.

Returns

double The result of the function.

2.2.2.18 stretchVSinWave()

```
double stretchVSinWave (  
    double * vec,  
    int n )
```

Stretched V Sine Wave function.

Parameters

<i>vec</i>	The vector of values the function will process.
<i>n</i>	The size of the vector.

Returns

double The result of the function.

2.2.2.19 w()

```
double w (  
    double * vec,  
    int n )
```

W: A helper function for Levy's function.

Parameters

<i>vec</i>	The vector of values the function will process.
<i>i</i>	The value of the vector to use

Returns

double The result of the helper function.

2.3 harmony.cpp File Reference

```
#include <iostream>
#include <random>
#include <thread>
#include "harmony.h"
#include "runFuncs.h"
Include dependency graph for harmony.cpp:
```

Functions

- void [harmony](#) (Population **pops, RecordKeeper **rks)
Threads the algorithm so each thread runs the population through a specific function.
- void [runHarmony](#) (Population *pop, RecordKeeper *rk)
Runs a population through the Harmony Search algorithm.
- void [initializeHS](#) (Population *pop)
Initializes a population to be optimized.
- double [adjustPitch](#) (double pitch, double bandwidth)
Adjusts the pitches of a harmony.
- void [addNewHarmony](#) (Population *pop, double *newHarm, double newFit)
Inserts a new harmony into a population. The new harmony will be insterted into a sorted position, so the population must already be sorted for this to work. This process will also remove the worst harmony in the population.
- void [updateRecordsFF](#) (Population *pop, RecordKeeper *rk, clock_t timer, const int iter)
Updates the RecordKeeper object with information of the optimization algorithm's progress.

2.3.1 Detailed Description

Author

Matthew Harker

Version

1.0

Date

2019-05-20

Copyright

Copyright (c) 2019

2.3.2 Function Documentation

2.3.2.1 addNewHarmony()

```
void addNewHarmony (
    Population * pop,
    double * newHarm,
    double newFit )
```

Inserts a new harmony into a population. The new harmony will be insterted into a sorted position, so the population must already be sorted for this to work. This process will also remove the worst harmony in the population.

Parameters

<i>pop</i>	The population being optimized
<i>newHarm</i>	The harmony being added into the population
<i>newFit</i>	The fitness of the new harmony being added

2.3.2.2 adjustPitch()

```
double adjustPitch (
    double pitch,
    double bandwidth )
```

Adjusts the pitches of a harmony.

Parameters

<i>pitch</i>	The pitch to be adjusted
<i>bandwidth</i>	A constant that adjusts the pitch

Returns

double The resulting adjustment

2.3.2.3 harmony()

```
void harmony (
    Population ** pops,
    RecordKeeper ** rks )
```

Threads the algorithm so each thread runs the population through a specific function.

Parameters

<i>pops</i>	The array of Population objects
<i>rks</i>	The array of RecordKeeper objects

2.3.2.4 initializeHS()

```
void initializeHS (
    Population * pop )
```

Initializes a population to be optimized.

Parameters

<i>pop</i>	The population to initialize
------------	------------------------------

2.3.2.5 runHarmony()

```
void runHarmony (
    Population * pop,
    RecordKeeper * rk )
```

Runs a population through the Harmony Search algorithm.

Parameters

<i>pop</i>	The population being optimized
<i>rk</i>	Records the information as the optimization executes

2.3.2.6 updateRecordsFF()

```
void updateRecordsFF (
    Population * pop,
    RecordKeeper * rk,
    clock_t timer,
    const int iter )
```

Updates the RecordKeeper object with information of the optimization algorithm's progress.

Parameters

<i>pop</i>	The populaion being optimized
<i>rk</i>	The RecordKeeper object storing the information
<i>timer</i>	Records the time an iteration took to complete
<i>iter</i>	Which iteration the population has just finished

2.4 main.cpp File Reference

Project 4 for CS470. This program run a population of solution vectors through Particle Swarm algorithm, Firefly algorithm, and Harmony Search algorithm then print all the results to csv files.

```
#include <iostream>
#include "unistd.h"
#include "csv.h"
```

```
#include "firefly.h"
#include "harmony.h"
#include "Parameters.h"
#include "particleSwarm.h"
#include "Population.h"
#include "RecordKeeper.h"
Include dependency graph for main.cpp:
```

Functions

- int [main](#) ()

The Main function. This will run the Patricle Swarm,.

2.4.1 Detailed Description

Project 4 for CS470. This program run a population of solution vectors through Particle Swarm algorithm, Firefly algorithm, and Harmony Search algorithm then print all the results to csv files.

Author

Matthew Harker

Version

4.0

Date

2019-05-20

Copyright

Copyright (c) 2019

2.4.2 Function Documentation

2.4.2.1 main()

```
int main ( )
```

The Main function. This will run the Patricle Swarm,.

Returns

int Indicates status of how the program ended.

2.5 Parameters.cpp File Reference

C++ file that provides utilities. This file contains all of the various utilities that provide extra functions, such as reading in parameters, and setting values in vectors.

```
#include <cmath>
#include <fstream>
#include <iostream>
#include <random>
#include <sstream>
#include <string>
#include "functions.h"
#include "Parameters.h"
```

Include dependency graph for Parameters.cpp:

Functions

- bool [compareDoubles](#) (const double x, const double y)
Checks if a double is sufficiently close to M_PI .
- Parameters [getParameters](#) ()
Reads in parameters from a file and returns an object filled with the values.

2.5.1 Detailed Description

C++ file that provides utilities. This file contains all of the various utilities that provide extra functions, such as reading in parameters, and setting values in vectors.

Author

Matthew Harker

Version

4.0

Date

2019-05-20

Copyright

Copyright (c) 2019

2.5.2 Function Documentation

2.5.2.1 [compareDoubles\(\)](#)

```
bool compareDoubles (
    const double x,
    const double y )
```

Checks if a double is sufficiently close to M_PI .

Parameters

<i>n</i>	The value being checked.
----------	--------------------------

Returns

true The value is PI.
false The value is not PI

2.5.2.2 getParameters()

```
Parameters getParameters ( )
```

Reads in parameters from a file and returns an object filled with the values.

Returns

Parameters The object filled with the read in parameters

2.6 particleSwarm.cpp File Reference

Optimizes a population using the Particle Swarm algorithm.

```
#include <cfloat>
#include <iostream>
#include <random>
#include <thread>
#include "particleSwarm.h"
```

Include dependency graph for particleSwarm.cpp:

Functions

- void [particleSwarm](#) (Population **pops, RecordKeeper **rks)
Optimizes population objects using Particle Swarm optimization.
- void [runParticleSwarm](#) (Population *pop, RecordKeeper *rk)
Runs each thread of PSO.
- void [updateVelocity](#) (Population *pop)
Updates all velocities based on the particle's pBest and the population's gBest.
- void [updateParticles](#) (Population *pop)
Updates the position of every particle. The position of the particle is updated by taking the current position and adding the velocity to it. If the velocity takes the particle out of bounds the new value is set to the relevant bound's value.
- void [updateFitness](#) (Population *pop)
Updates the fitnesses of the particles in the population.
- void [updatePersonalBest](#) (Population *pop)
Updates each particle's personal best. If the particle's new fitness is more optimal than the personal best, update the personal best.
- void [updateGlobalBest](#) (Population *pop)
Updates the global best particle and fitness. If the population contains a particle that is more optimal than the global best, update the global best particle and its fitness.
- void [initializePSO](#) (Population *pop)
Initializes a population. The particles and velocities in the population are set to random values. The personal best arrays and values are set and the global best array and value is set.
- void [updateRecords](#) (Population *pop, RecordKeeper *rk, const clock_t timer, const int iter)
Updates a RecordKeeper object based on the most recent iteration.

2.6.1 Detailed Description

Optimizes a population using the Particle Swarm algorithm.

Author

Matthew Harker

Version

1.0

Date

2019-05-20

Copyright

Copyright (c) 2019

2.6.2 Function Documentation

2.6.2.1 initializePSO()

```
void initializePSO (
    Population * pop )
```

Initializes a population. The particles and velocities in the population are set to random values. The personal best arrays and values are set and the global best array and value is set.

Parameters

<i>pop</i>	The population to initialize
------------	------------------------------

2.6.2.2 particleSwarm()

```
void particleSwarm (
    Population ** pops,
    RecordKeeper ** rks )
```

Optimizes population objects using Particle Swarm optimization.

Parameters

<i>pops</i>	
-------------	--

2.6.2.3 runParticleSwarm()

```
void runParticleSwarm (
    Population * pop,
    RecordKeeper * rk )
```

Runs each thread of PSO.

Parameters

<i>pop</i>	The population to optimize
------------	----------------------------

2.6.2.4 updateFitness()

```
void updateFitness (
    Population * pop )
```

Updates the fitnesses of the particles in the population.

Parameters

<i>pop</i>	The population to update
------------	--------------------------

2.6.2.5 updateGlobalBest()

```
void updateGlobalBest (
    Population * pop )
```

Updates the global best particle and fitness. If the population contains a particle that is more optimal than the global best, update the global best particle and its fitness.

Parameters

<i>pop</i>	The population to update
------------	--------------------------

2.6.2.6 updateParticles()

```
void updateParticles (
    Population * pop )
```

Updates the position of every particle. The position of the particle is updated by taking the current position and adding the velocity to it. If the velocity takes the particle out of bounds the new value is set to the relevant bound's value.

Parameters

<i>pop</i>	The population to update
------------	--------------------------

2.6.2.7 updatePersonalBest()

```
void updatePersonalBest (
    Population * pop )
```

Updates each particle's personal best. If the particle's new fitness is more optimal than the personal best, update the personal best.

Parameters

<i>pop</i>	The population to update
------------	--------------------------

2.6.2.8 updateRecords()

```
void updateRecords (
    Population * pop,
    RecordKeeper * rk,
    const clock_t timer,
    const int iter )
```

Updates a RecordKeeper object based on the most recent iteration.

Parameters

<i>pop</i>	The population object to get info from
<i>rk</i>	The RecordKeeper object to write info to
<i>timer</i>	Records the length of each iteration
<i>iter</i>	Which iteration is being recorded

2.6.2.9 updateVelocity()

```
void updateVelocity (
    Population * pop )
```

Updates all velocities based on the particle's pBest and the population's gBest.

Parameters

<i>pop</i>	The population to update
------------	--------------------------

2.7 Population.cpp File Reference

Holds optimization information.

```
#include <cfloat>
#include <chrono>
#include <climits>
#include <iostream>
#include <random>
#include "Population.h"
#include "runFuncs.h"
Include dependency graph for Population.cpp:
```

2.7.1 Detailed Description

Holds optimization information.

Author

Matthew Harker

Version

3.0

Date

2019-05-20

Copyright

Copyright (c) 2019

2.8 RecordKeeper.cpp File Reference

Holds information about multiple optiization experimentations.

```
#include <iostream>
#include "RecordKeeper.h"
Include dependency graph for RecordKeeper.cpp:
```

2.8.1 Detailed Description

Holds information about multiple optiization experimentations.

Author

Matthew

Version

4.0

Date

2019-05-20

Copyright

Copyright (c) 2019

2.9 runFuncs.cpp File Reference

Passes a solution vector through a function and returns the resulting value.

```
#include <iostream>
#include "functions.h"
#include "runFuncs.h"
Include dependency graph for runFuncs.cpp:
```

Functions

- double [runSolution](#) (double *solVec, const int size, const int func)
Runs one solution vector through a function.

2.9.1 Detailed Description

Passes a solution vector through a function and returns the resulting value.

Author

Matthew Harker

Version

4.0

Date

2019-05-20

Copyright

Copyright (c) 2019

2.9.2 Function Documentation

2.9.2.1 runSolution()

```
double runSolution (
    double * solVec,
    const int size,
    const int func )
```

Runs one solution vector through a function.

Parameters

<i>solVec</i>	The solution vector
<i>size</i>	The size of the vector
<i>func</i>	Which function to run the vector through

Returns

double The resulting value

Index

- ackleyOne
 - functions.cpp, 4
- ackleyTwo
 - functions.cpp, 6
- addNewHarmony
 - harmony.cpp, 13
- adjustPitch
 - harmony.cpp, 14
- alpine
 - functions.cpp, 6
- compareDoubles
 - Parameters.cpp, 17
- dejong
 - functions.cpp, 6
- eggholder
 - functions.cpp, 7
- firefly.cpp, 3
- functions.cpp, 3
 - ackleyOne, 4
 - ackleyTwo, 6
 - alpine, 6
 - dejong, 6
 - eggholder, 7
 - griewank, 7
 - levy, 8
 - mastersCosWave, 8
 - Michalewicz, 8
 - pathological, 9
 - quartic, 9
 - rana, 9
 - rastrigin, 10
 - rosenbrock, 10
 - Schwefel, 11
 - sinEnvlSinWave, 11
 - step, 11
 - stretchVSinWave, 12
 - w, 12
- getParameters
 - Parameters.cpp, 18
- griewank
 - functions.cpp, 7
- harmony
 - harmony.cpp, 14
- harmony.cpp, 13
 - addNewHarmony, 13
 - adjustPitch, 14
 - harmony, 14
 - initializeHS, 14
 - runHarmony, 15
 - updateRecordsFF, 15
- initializeHS
 - harmony.cpp, 14
- initializePSO
 - particleSwarm.cpp, 19
- levy
 - functions.cpp, 8
- main
 - main.cpp, 16
- main.cpp, 15
 - main, 16
- mastersCosWave
 - functions.cpp, 8
- Michalewicz
 - functions.cpp, 8
- Parameters.cpp, 17
 - compareDoubles, 17
 - getParameters, 18
- particleSwarm
 - particleSwarm.cpp, 19
- particleSwarm.cpp, 18
 - initializePSO, 19
 - particleSwarm, 19
 - runParticleSwarm, 20
 - updateFitness, 20
 - updateGlobalBest, 20
 - updateParticles, 20
 - updatePersonalBest, 21
 - updateRecords, 21
 - updateVelocity, 21
- pathological
 - functions.cpp, 9
- Population.cpp, 22
- quartic
 - functions.cpp, 9
- rana
 - functions.cpp, 9
- rastrigin
 - functions.cpp, 10
- RecordKeeper.cpp, 22
- rosenbrock

- functions.cpp, [10](#)
- runFuncs.cpp, [23](#)
 - runSolution, [24](#)
- runHarmony
 - harmony.cpp, [15](#)
- runParticleSwarm
 - particleSwarm.cpp, [20](#)
- runSolution
 - runFuncs.cpp, [24](#)
- schwefel
 - functions.cpp, [11](#)
- sinEnvlSinWave
 - functions.cpp, [11](#)
- step
 - functions.cpp, [11](#)
- stretchVSinWave
 - functions.cpp, [12](#)
- updateFitness
 - particleSwarm.cpp, [20](#)
- updateGlobalBest
 - particleSwarm.cpp, [20](#)
- updateParticles
 - particleSwarm.cpp, [20](#)
- updatePersonalBest
 - particleSwarm.cpp, [21](#)
- updateRecords
 - particleSwarm.cpp, [21](#)
- updateRecordsFF
 - harmony.cpp, [15](#)
- updateVelocity
 - particleSwarm.cpp, [21](#)
- w
 - functions.cpp, [12](#)