My Project

Generated by Doxygen 1.8.13

Contents

Chapter 1

File Index

1.1 File List

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

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

2.2.2 Function Documentation

2.2.2.1 ackleyOne()

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

Ackley's first function.

Parameters

vec	The vector of values the function will process.
n	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

vec	The vector of values the function will process.
n	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

vec	The vector of values the function will process.
n	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

vec	The vector of values the function will process.
n	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

vec	The vector of values the function will process.
n	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

vec	The vector of values the function will process.
n	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

vec	The vector of values the function will process.
n	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

vec	The vector of values the function will process.
n	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

vec	The vector of values the function will process.
n	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

vec	The vector of values the function will process.
n	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

vec	The vector of values the function will process.
n	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

vec	The vector of values the function will process.
n	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

vec	The vector of values the function will process.
n	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

vec	The vector of values the function will process.
n	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

vec	The vector of values the function will process.
n	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

vec	The vector of values the function will process.
n	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

vec	The vector of values the function will process.
n	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

vec	The vector of values the function will process.
n	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

vec	The vector of values the function will process.
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

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

pop	The population being optimized
newHarm	The harmony being added into the population
newFit	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

pitch	The pitch to be adjusted
bandwidth	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

pops	The array of Population objects
rks	The array of RecordKeeper objects

2.3.2.4 initializeHS()

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

Initializes a population to be optimized.

Parameters

pop	The population to initialize

2.3.2.5 runHarmony()

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

Runs a population through the Harmony Search algorithm.

Parameters

pop	The population being optimized
rk	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

pop	The populaion being optimized
rk	The RecordKeeper object storing the information
timer	Records the time an iteration took to complete
iter	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 algorthim, 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 algorthim, 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

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

n	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 the 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 that the global best, updat 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

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

pop	The population to initialize

2.6.2.2 particleSwarm()

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

Optimizes population objects using Particle Swarm optimization.

Parameters

pops	

2.6.2.3 runParticleSwarm()

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

Runs each thread of PSO.

Parameters

pop	The population to optimize

2.6.2.4 updateFitness()

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

Updates the fitnesses of the particles in the population.

Parameters

pop	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 that the global best, updat the global best particle and its fitness.

Parameters

pop	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

pop	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 the the personal best, update the personal best.

Parameters

pop	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

pop	The population object to get info from
rk	The RecordKeeper object to write info to
timer	Records the legnth of each iteration
iter	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

pop	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

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

solVec	The solution vector
size	The size of the vector
func	Which function to run the vector through

Returns

double The resulting value

Index