

CS 471 Project 4 Doxygen

Generated by Doxygen 1.8.15

| | |
|---------------------------------------|----------|
| 1 Data Structure Index | 1 |
| 1.1 Data Structures | 1 |
| 2 File Index | 3 |
| 2.1 File List | 3 |
| 3 Data Structure Documentation | 5 |
| 3.1 _FA1 Struct Reference | 5 |
| 3.1.1 Detailed Description | 6 |
| 3.1.2 Field Documentation | 6 |
| 3.1.2.1 alpha | 6 |
| 3.1.2.2 beta | 6 |
| 3.1.2.3 betaMin | 6 |
| 3.1.2.4 gamma | 6 |
| 3.1.2.5 l0 | 7 |
| 3.1.2.6 index | 7 |
| 3.1.2.7 intensity | 7 |
| 3.1.2.8 newXi | 7 |
| 3.1.2.9 newXiFitness | 7 |
| 3.1.2.10 r | 7 |
| 3.1.2.11 tempFitness | 8 |
| 3.1.2.12 tempPopulation | 8 |
| 3.2 _HS1 Struct Reference | 8 |
| 3.2.1 Detailed Description | 8 |
| 3.2.2 Field Documentation | 9 |
| 3.2.2.1 BW | 9 |
| 3.2.2.2 HMCR | 9 |
| 3.2.2.3 HSfitness | 9 |
| 3.2.2.4 HSVector | 9 |
| 3.2.2.5 PAR | 9 |
| 3.3 _initData1 Struct Reference | 10 |
| 3.3.1 Detailed Description | 10 |
| 3.3.2 Field Documentation | 10 |
| 3.3.2.1 fitness | 10 |
| 3.3.2.2 functionNumber | 10 |
| 3.3.2.3 max | 11 |
| 3.3.2.4 min | 11 |
| 3.3.2.5 population | 11 |
| 3.4 _PSO1 Struct Reference | 11 |
| 3.4.1 Detailed Description | 12 |
| 3.4.2 Field Documentation | 12 |
| 3.4.2.1 c1 | 12 |
| 3.4.2.2 c2 | 12 |

| | |
|--|-----------|
| 3.4.2.3 gBest | 12 |
| 3.4.2.4 k | 12 |
| 3.4.2.5 pBest | 13 |
| 3.4.2.6 velocity | 13 |
| 4 File Documentation | 15 |
| 4.1 C:/Users/emyli/Desktop/Project4_471/ArrayMem.c File Reference | 15 |
| 4.1.1 Function Documentation | 15 |
| 4.1.1.1 createDblArray() | 15 |
| 4.1.1.2 fillIn() | 16 |
| 4.1.1.3 freeMem() | 16 |
| 4.1.1.4 singleArray() | 17 |
| 4.2 C:/Users/emyli/Desktop/Project4_471/ArrayMem.h File Reference | 17 |
| 4.2.1 Function Documentation | 17 |
| 4.2.1.1 createDblArray() | 17 |
| 4.2.1.2 fillIn() | 18 |
| 4.2.1.3 freeMem() | 19 |
| 4.2.1.4 singleArray() | 19 |
| 4.3 C:/Users/emyli/Desktop/Project4_471/FA.c File Reference | 19 |
| 4.3.1 Function Documentation | 20 |
| 4.3.1.1 Eqn2() | 20 |
| 4.3.1.2 Eqn3() | 20 |
| 4.3.1.3 Eqn4() | 21 |
| 4.3.1.4 readInputFA() | 21 |
| 4.3.1.5 startFA() | 23 |
| 4.3.1.6 updateIntensity() | 23 |
| 4.4 C:/Users/emyli/Desktop/Project4_471/FA.h File Reference | 24 |
| 4.4.1 Typedef Documentation | 24 |
| 4.4.1.1 FA | 25 |
| 4.4.2 Function Documentation | 25 |
| 4.4.2.1 Eqn2() | 25 |
| 4.4.2.2 Eqn3() | 25 |
| 4.4.2.3 Eqn4() | 26 |
| 4.4.2.4 readInputFA() | 27 |
| 4.4.2.5 startFA() | 28 |
| 4.4.2.6 updateIntensity() | 28 |
| 4.5 C:/Users/emyli/Desktop/Project4_471/Functions.c File Reference | 29 |
| 4.5.1 Function Documentation | 30 |
| 4.5.1.1 ackley1() | 30 |
| 4.5.1.2 ackley2() | 30 |
| 4.5.1.3 alpine() | 31 |
| 4.5.1.4 deJong() | 31 |

| | |
|--|----|
| 4.5.1.5 eggHolder() | 32 |
| 4.5.1.6 griewangk() | 32 |
| 4.5.1.7 levy() | 33 |
| 4.5.1.8 masters() | 33 |
| 4.5.1.9 michalewicz() | 33 |
| 4.5.1.10 pathological() | 34 |
| 4.5.1.11 quartic() | 34 |
| 4.5.1.12 rana() | 35 |
| 4.5.1.13 rastrigin() | 35 |
| 4.5.1.14 rosenbrock() | 36 |
| 4.5.1.15 schwefel() | 36 |
| 4.5.1.16 sineEnvelope() | 37 |
| 4.5.1.17 sineStretched() | 37 |
| 4.5.1.18 step() | 37 |
| 4.6 C:/Users/emyli/Desktop/Project4_471/Functions.h File Reference | 38 |
| 4.6.1 Function Documentation | 39 |
| 4.6.1.1 ackley1() | 39 |
| 4.6.1.2 ackley2() | 39 |
| 4.6.1.3 alpine() | 40 |
| 4.6.1.4 deJong() | 41 |
| 4.6.1.5 eggHolder() | 41 |
| 4.6.1.6 griewangk() | 42 |
| 4.6.1.7 levy() | 42 |
| 4.6.1.8 masters() | 43 |
| 4.6.1.9 michalewicz() | 44 |
| 4.6.1.10 pathological() | 44 |
| 4.6.1.11 quartic() | 45 |
| 4.6.1.12 rana() | 46 |
| 4.6.1.13 rastrigin() | 46 |
| 4.6.1.14 rosenbrock() | 47 |
| 4.6.1.15 schwefel() | 47 |
| 4.6.1.16 sineEnvelope() | 48 |
| 4.6.1.17 sineStretched() | 49 |
| 4.6.1.18 step() | 50 |
| 4.7 C:/Users/emyli/Desktop/Project4_471/HS.c File Reference | 50 |
| 4.7.1 Function Documentation | 51 |
| 4.7.1.1 checkBounds() | 51 |
| 4.7.1.2 readInputHS() | 51 |
| 4.7.1.3 startHS() | 52 |
| 4.7.1.4 updateHM() | 52 |
| 4.7.1.5 updateWithPAR() | 53 |
| 4.8 C:/Users/emyli/Desktop/Project4_471/HS.h File Reference | 53 |

| | |
|---|----|
| 4.8.1 Typedef Documentation | 53 |
| 4.8.1.1 HS | 54 |
| 4.8.2 Function Documentation | 54 |
| 4.8.2.1 checkBounds() | 54 |
| 4.8.2.2 readInputHS() | 54 |
| 4.8.2.3 startHS() | 55 |
| 4.8.2.4 updateHM() | 56 |
| 4.8.2.5 updateWithPAR() | 56 |
| 4.9 C:/Users/emyli/Desktop/Project4_471/main.c File Reference | 57 |
| 4.9.1 Function Documentation | 57 |
| 4.9.1.1 main() | 57 |
| 4.10 C:/Users/emyli/Desktop/Project4_471/mt19937ar.c File Reference | 57 |
| 4.10.1 Macro Definition Documentation | 58 |
| 4.10.1.1 LOWER_MASK | 58 |
| 4.10.1.2 M | 58 |
| 4.10.1.3 MATRIX_A | 58 |
| 4.10.1.4 N | 59 |
| 4.10.1.5 UPPER_MASK | 59 |
| 4.10.2 Function Documentation | 59 |
| 4.10.2.1 genrand_int31() | 59 |
| 4.10.2.2 genrand_int32() | 59 |
| 4.10.2.3 genrand_real1() | 59 |
| 4.10.2.4 genrand_real2() | 59 |
| 4.10.2.5 genrand_real3() | 60 |
| 4.10.2.6 genrand_res53() | 60 |
| 4.10.2.7 init_by_array() | 60 |
| 4.10.2.8 init_genrand() | 60 |
| 4.10.3 Variable Documentation | 60 |
| 4.10.3.1 mt | 60 |
| 4.10.3.2 mti | 60 |
| 4.11 C:/Users/emyli/Desktop/Project4_471/mt19937ar.h File Reference | 61 |
| 4.11.1 Function Documentation | 61 |
| 4.11.1.1 genrand_int31() | 61 |
| 4.11.1.2 genrand_int32() | 61 |
| 4.11.1.3 genrand_real1() | 61 |
| 4.11.1.4 genrand_real2() | 61 |
| 4.11.1.5 genrand_real3() | 62 |
| 4.11.1.6 genrand_res53() | 62 |
| 4.11.1.7 init_by_array() | 62 |
| 4.11.1.8 init_genrand() | 62 |
| 4.12 C:/Users/emyli/Desktop/Project4_471/PSO.c File Reference | 62 |
| 4.12.1 Function Documentation | 63 |

| | |
|--|----|
| 4.12.1.1 calcPopulation() | 63 |
| 4.12.1.2 calcVelocity() | 63 |
| 4.12.1.3 readInput() | 63 |
| 4.12.1.4 startPSO() | 64 |
| 4.13 C:/Users/emyli/Desktop/Project4_471/PSO.h File Reference | 65 |
| 4.13.1 Typedef Documentation | 65 |
| 4.13.1.1 PSO | 66 |
| 4.13.2 Function Documentation | 66 |
| 4.13.2.1 calcPopulation() | 66 |
| 4.13.2.2 calcVelocity() | 67 |
| 4.13.2.3 readInput() | 67 |
| 4.13.2.4 startPSO() | 68 |
| 4.14 C:/Users/emyli/Desktop/Project4_471/SelectFunction.c File Reference | 69 |
| 4.14.1 Function Documentation | 69 |
| 4.14.1.1 getFun() | 69 |
| 4.14.1.2 getFunSingle() | 70 |
| 4.15 C:/Users/emyli/Desktop/Project4_471/SelectFunction.h File Reference | 70 |
| 4.15.1 Function Documentation | 70 |
| 4.15.1.1 getFun() | 71 |
| 4.15.1.2 getFunSingle() | 71 |
| 4.16 C:/Users/emyli/Desktop/Project4_471/Util.c File Reference | 72 |
| 4.16.1 Function Documentation | 72 |
| 4.16.1.1 copyDbl() | 73 |
| 4.16.1.2 copySingle() | 73 |
| 4.16.1.3 findBest() | 73 |
| 4.16.1.4 printDblDim() | 74 |
| 4.16.1.5 printSingle() | 74 |
| 4.16.1.6 replaceArray() | 75 |
| 4.16.1.7 sortAscendingOrder() | 75 |
| 4.17 C:/Users/emyli/Desktop/Project4_471/Util.h File Reference | 75 |
| 4.17.1 Typedef Documentation | 76 |
| 4.17.1.1 initData | 76 |
| 4.17.2 Function Documentation | 76 |
| 4.17.2.1 copyDbl() | 76 |
| 4.17.2.2 copySingle() | 77 |
| 4.17.2.3 findBest() | 77 |
| 4.17.2.4 printDblDim() | 78 |
| 4.17.2.5 printSingle() | 78 |
| 4.17.2.6 replaceArray() | 78 |
| 4.17.2.7 sortAscendingOrder() | 79 |

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

| | | |
|----------------------------|---|----|
| _FA1 | Structure for the FA algorithm | 5 |
| _HS1 | Structure for the HS algorithm | 8 |
| _initData1 | Structure for the bounds, population, fitness and function number | 10 |
| _PSO1 | Structure for the PSO algorithm | 11 |

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

| | |
|--|----|
| C:/Users/emyli/Desktop/Project4_471/ArrayMem.c | 15 |
| C:/Users/emyli/Desktop/Project4_471/ArrayMem.h | 17 |
| C:/Users/emyli/Desktop/Project4_471/FA.c | 19 |
| C:/Users/emyli/Desktop/Project4_471/FA.h | 24 |
| C:/Users/emyli/Desktop/Project4_471/Functions.c | 29 |
| C:/Users/emyli/Desktop/Project4_471/Functions.h | 38 |
| C:/Users/emyli/Desktop/Project4_471/HS.c | 50 |
| C:/Users/emyli/Desktop/Project4_471/HS.h | 53 |
| C:/Users/emyli/Desktop/Project4_471/main.c | 57 |
| C:/Users/emyli/Desktop/Project4_471/mt19937ar.c | 57 |
| C:/Users/emyli/Desktop/Project4_471/mt19937ar.h | 61 |
| C:/Users/emyli/Desktop/Project4_471/PSO.c | 62 |
| C:/Users/emyli/Desktop/Project4_471/PSO.h | 65 |
| C:/Users/emyli/Desktop/Project4_471/SelectFunction.c | 69 |
| C:/Users/emyli/Desktop/Project4_471/SelectFunction.h | 70 |
| C:/Users/emyli/Desktop/Project4_471/Util.c | 72 |
| C:/Users/emyli/Desktop/Project4_471/Util.h | 75 |

Chapter 3

Data Structure Documentation

3.1 _FA1 Struct Reference

Structure for the FA algorithm.

```
#include <FA.h>
```

Data Fields

- double [alpha](#)
constant parameter (0.5)
- double [betaMin](#)
constant parameter(0.2)
- double [gamma](#)
light absorption coefficient (.01-100)
- double [beta](#)
attractiveness of a firefly
- double [r](#)
the distance from one firefly to another
- double * [intensity](#)
the intensity from one firefly to another
- double * [newXi](#)
new vector for the population
- int [index](#)
keeps track of the worst fitness' index
- double * [tempFitness](#)
temporary fitness for the copied population
- double [newXiFitness](#)
fitness for the new vector for the population
- double ** [tempPopulation](#)
temporary population copy
- double * [IO](#)

3.1.1 Detailed Description

Structure for the FA algorithm.

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

Firefly Algorithm header file. This class implements the methods for performing the Firefly Algorithm This algorithm models the social behavior of fireflies based on the flashing and attraction characteristics of fireflies. It was developed by Yang in 2010.

3.1.2 Field Documentation

3.1.2.1 alpha

`double alpha`

constant parameter (0.5)

3.1.2.2 beta

`double beta`

attractiveness of a firefly

3.1.2.3 betaMin

`double betaMin`

constant parameter(0.2)

3.1.2.4 gamma

`double gamma`

light absorption coefficient (.01-100)

3.1.2.5 I0

```
double* I0
```

3.1.2.6 index

```
int index
```

keeps track of the worst fitness' index

3.1.2.7 intensity

```
double* intensity
```

the intensity from one firefly to another

3.1.2.8 newXi

```
double* newXi
```

new vector for the population

3.1.2.9 newXiFitness

```
double newXiFitness
```

fitness for the new vector for the population

3.1.2.10 r

```
double r
```

the distance from one firefly to another

3.1.2.11 tempFitness

```
double* tempFitness
```

temporary fitness for the copied population

3.1.2.12 tempPopulation

```
double** tempPopulation
```

temporary population copy

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

- C:/Users/emyli/Desktop/Project4_471/FA.h

3.2 _HS1 Struct Reference

Structure for the HS algorithm.

```
#include <HS.h>
```

Data Fields

- double [HMCR](#)
Harmony Memory Considering Rate; (0.7-0.95)
- double [PAR](#)
Pitch Adjusting Rate; (0.1-0.5)
- double [BW](#)
Bandwidth; (0.2)
- double * [HSVector](#)
a new harmony vector
- double [HSfitness](#)
new harmony fitness vector

3.2.1 Detailed Description

Structure for the HS algorithm.

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

Harmony Search Algorithm header file. This class implements the methods for performing the Harmony Search Algorithm. This algorithm models the music improvisation process where the musicians improvise their instruments' pitch by searching for a perfect state of harmony (Geem et al. 2001)

3.2.2 Field Documentation

3.2.2.1 BW

`double BW`

Bandwidth; (0.2)

3.2.2.2 HMCR

`double HMCR`

Harmony Memory Considering Rate; (0.7-0.95)

3.2.2.3 HSfitness

`double HSfitness`

new harmony fitness vector

3.2.2.4 HSVector

`double* HSVector`

a new harmony vector

3.2.2.5 PAR

`double PAR`

Pitch Adjusting Rate; (0.1-0.5)

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

- C:/Users/emyli/Desktop/Project4_471/[HS.h](#)

3.3 `_initData1` Struct Reference

Structure for the bounds, population, fitness and function number.

```
#include <Util.h>
```

Data Fields

- double `max`
- double `min`
upper bound
- double ** `population`
lower bound
- double * `fitness`
random matrix
- int `functionNumber`
fitness from matrix

3.3.1 Detailed Description

Structure for the bounds, population, fitness and function number.

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

This project implements three meta-heuristic optimization algorithms. Particle Swarm Optimization (PSO), Firefly Algorithm (FA), and Harmony Search Algorithm (HS).

3.3.2 Field Documentation

3.3.2.1 `fitness`

```
double* fitness
```

random matrix

3.3.2.2 `functionNumber`

```
int functionNumber
```

fitness from matrix

3.3.2.3 max

double max

3.3.2.4 min

double min

upper bound

3.3.2.5 population

double** population

lower bound

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

- C:/Users/emyli/Desktop/Project4_471/[Util.h](#)

3.4 _PSO1 Struct Reference

Structure for the PSO algorithm.

```
#include <PSO.h>
```

Data Fields

- double [k](#)
Dampening factor (0.8-1.2)
- double [c1](#)
value go towards pBest (0-2)
- double [c2](#)
value go towards gBest (0-2)
- double [gBest](#)
global best solution of the population
- double * [pBest](#)
personal best solution by a specific particle
- double ** [velocity](#)
velocity of the population

3.4.1 Detailed Description

Structure for the PSO algorithm.

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

Particle Swarm Optimization header file. This class implements the methods for performing Particle Swarm Optimization. This algorithm models flocking and schooling patterns of birds and fishes. It was invented by Russell Eberhart and James Kennedy in 1995.

3.4.2 Field Documentation

3.4.2.1 c1

`double c1`

value go towards pBest (0-2)

3.4.2.2 c2

`double c2`

value go towards gBest (0-2)

3.4.2.3 gBest

`double gBest`

global best solution of the population

3.4.2.4 k

`double k`

Dampening factor (0.8-1.2)

3.4.2.5 pBest

```
double* pBest
```

personal best solution by a specific particle

3.4.2.6 velocity

```
double** velocity
```

velocity of the population

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

- C:/Users/emyli/Desktop/Project4_471/[PSO.h](#)

Chapter 4

File Documentation

4.1 C:/Users/emyli/Desktop/Project4_471/ArrayMem.c File Reference

```
#include <stdlib.h>
#include "ArrayMem.h"
#include "mt19937ar.h"
```

Functions

- double ** [createDblArray](#) (int col, int row)
Generates a double pointer array that has pointer to a single pointer array The arrays are initialized to 0 by using calloc()
- double * [singleArray](#) (int n)
- double ** [fillIn](#) (double **arr, int row, int col, double min, double max)
- void [freeMem](#) (int row, double **matrix)

4.1.1 Function Documentation

4.1.1.1 createDblArray()

```
double** createDblArray (
    int col,
    int row )
```

Generates a double pointer array that has pointer to a single pointer array The arrays are initialized to 0 by using calloc()

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

This project implements three meta-heuristic optimization algorithms. Particle Swarm Optimization (PSO), Firefly Algorithm (FA), and Harmony Search Algorithm (HS). Generates a double pointer array that has pointer to a single pointer array The arrays are initialized to 0 by using calloc()

Parameters

| | |
|------------|--------------------------------------|
| <i>col</i> | the size of the columns of the array |
| <i>row</i> | the size of the rows of the array |

Returns

double pointer array

4.1.1.2 fillIn()

```
double** fillIn (
    double ** arr,
    int row,
    int col,
    double min,
    double max )
```

Fills in a double pointer array with the MersenneTwister random numbers from a specified range

Parameters

| | |
|------------|--|
| <i>arr</i> | double pointer array |
| <i>row</i> | the size of the rows of the array |
| <i>col</i> | the size of the columns of the array |
| <i>min</i> | the minimum size of the random numbers |
| <i>max</i> | the maximum size of the random numbers |

Returns

double pointer array

4.1.1.3 freeMem()

```
void freeMem (
    int row,
    double ** matrix )
```

Free the memory used for a double pointer array

Parameters

| | |
|---------------|-----------------------------------|
| <i>matrix</i> | double pointer array |
| <i>row</i> | the size of the rows of the array |

4.1.1.4 singleArray()

```
double* singleArray (
    int n )
```

Generates a single pointer array The array are initialized to 0 by using calloc()

Parameters

| | |
|----------|-----------------------------------|
| <i>n</i> | the size of the rows of the array |
|----------|-----------------------------------|

Returns

single pointer array

4.2 C:/Users/emyli/Desktop/Project4_471/ArrayMem.h File Reference

Functions

- double ** [createDblArray](#) (int col, int row)
Generates a double pointer array that has pointer to a single pointer array The arrays are initialized to 0 by using calloc()
- double * [singleArray](#) (int n)
- double ** [fillIn](#) (double **arr, int row, int col, double min, double max)
- void [freeMem](#) (int row, double **matrix)

4.2.1 Function Documentation

4.2.1.1 createDblArray()

```
double** createDblArray (
    int col,
    int row )
```

Generates a double pointer array that has pointer to a single pointer array The arrays are initialized to 0 by using calloc()

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

This project implements three meta-heuristic optimization algorithms. Particle Swarm Optimization (PSO), Firefly Algorithm (FA), and Harmony Search Algorithm (HS).

Parameters

| | |
|------------|--------------------------------------|
| <i>col</i> | the size of the columns of the array |
| <i>row</i> | the size of the rows of the array |

Returns

double pointer array

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

This project implements three meta-heuristic optimization algorithms. Particle Swarm Optimization (PSO), Firefly Algorithm (FA), and Harmony Search Algorithm (HS). Generates a double pointer array that has pointer to a single pointer array The arrays are initialized to 0 by using calloc()

Parameters

| | |
|------------|--------------------------------------|
| <i>col</i> | the size of the columns of the array |
| <i>row</i> | the size of the rows of the array |

Returns

double pointer array

4.2.1.2 fillIn()

```
double** fillIn (
    double ** arr,
    int row,
    int col,
    double min,
    double max )
```

Fills in a double pointer array with the MersenneTwister random numbers from a specified range

Parameters

| | |
|------------|--|
| <i>arr</i> | double pointer array |
| <i>row</i> | the size of the rows of the array |
| <i>col</i> | the size of the columns of the array |
| <i>min</i> | the minimum size of the random numbers |
| <i>max</i> | the maximum size of the random numbers |

Returns

double pointer array

4.2.1.3 freeMem()

```
void freeMem (
    int row,
    double ** matrix )
```

Free the memory used for a double pointer array

Parameters

| | |
|---------------|-----------------------------------|
| <i>matrix</i> | double pointer array |
| <i>row</i> | the size of the rows of the array |

4.2.1.4 singleArray()

```
double* singleArray (
    int n )
```

Generates a single pointer array The array are initialized to 0 by using calloc()

Parameters

| | |
|----------|-----------------------------------|
| <i>n</i> | the size of the rows of the array |
|----------|-----------------------------------|

Returns

single pointer array

4.3 C:/Users/emyli/Desktop/Project4_471/FA.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include <time.h>
#include "Util.h"
#include "FA.h"
#include "ArrayMem.h"
#include "SelectFunction.h"
#include "mt19937ar.h"
```

Functions

- void [readInputFA](#) (FILE *input, FILE *fileOut, FILE *filePop, int iterations, int fitnessCallCounter, [initData](#) *myData, int NS, int DIM)
Function that reads from the FA input file to initialize the structure variables for the FA algorithm.
- void [startFA](#) (FA *myFA, FILE *fileOut, FILE *filePop, const int NS, const int DIM, const int iterations, [initData](#) *myData, int fitnessCallCounter)
Function that runs the FA algorithm.
- void [Eqn3](#) (const double *xi, const double *xj, int DIM, FA *myFA)
Function that finds the distance between two fireflies.
- double [Eqn2](#) (FA *myFA)
Function that updates Beta.
- void [Eqn4](#) (const double *xi, const double *xj, const int DIM, FA *myFA, [initData](#) *myData)
Function that creates a new firefly vector, checks if the vector is better than the worst firefly in the population.
- double [updateIntensity](#) (FA *myFA, int j)
Function that updates the intensity of the fireflies.

4.3.1 Function Documentation

4.3.1.1 Eqn2()

```
double Eqn2 (
    FA * myFA )
```

Function that updates Beta.

Function that updates Beta

Parameters

| | |
|-------------|--------------------------|
| <i>myFA</i> | the FA structure pointer |
|-------------|--------------------------|

Returns

Beta

4.3.1.2 Eqn3()

```
void Eqn3 (
    const double * xi,
    const double * xj,
    int DIM,
    FA * myFA )
```

Function that finds the distance between two fireflies.

function that moves firefly j towards firefly i; EQN. 3

Parameters

| | |
|-------------|--------------------------------------|
| <i>xi</i> | firefly vector at i |
| <i>xj</i> | firefly vector at j |
| <i>DIM</i> | number of dimensions for a data type |
| <i>myFA</i> | the PSO structure pointer |

4.3.1.3 Eqn4()

```
void Eqn4 (
    const double * xi,
    const double * xj,
    const int DIM,
    FA * myFA,
    initData * myData )
```

Function that creates a new firefly vector, checks if the vector is better than the worst firefly in the population.

Evaluate and update the worst firefly in the population; EQN. 4

Parameters

| | |
|---------------|--|
| <i>xi</i> | firefly vector at i |
| <i>xj</i> | firefly vector at j |
| <i>DIM</i> | number of dimensions for a data type |
| <i>myFA</i> | the PSO structure pointer |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |

4.3.1.4 readInputFA()

```
void readInputFA (
    FILE * input,
    FILE * fileOut,
    FILE * filePop,
    int iterations,
    int fitnessCallCounter,
    initData * myData,
    int NS,
    int DIM )
```

Function that reads from the FA input file to initialize the structure variables for the FA algorithm.

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

Firefly Algorithm class file. This class implements the methods for performing the Firefly Algorithm This algorithm models the social behavior of fireflies based on the flashing and attraction characteristics of fireflies. It was developed by Yang in 2010. Function that reads from the FA input file to initialize the structure variables for the FA algorithm

Parameters

| | |
|---------------------------|--|
| <i>input</i> | file that has the values for the myData structure |
| <i>fileOut</i> | file that prints the best solution to a csv |
| <i>filePop</i> | file that prints the population to a csv |
| <i>iterations</i> | number of iterations to run the algorithm for |
| <i>fitnessCallCounter</i> | count how many times the fitness function was called |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>NS</i> | number of solutions for a data type |
| <i>DIM</i> | number of dimensions for a data type |

4.3.1.5 startFA()

```
void startFA (
    FA * myFA,
    FILE * fileOut,
    FILE * filePop,
    const int NS,
    const int DIM,
    const int iterations,
    initData * myData,
    int fitnessCallCounter )
```

Function that runs the FA algorithm.

Function that runs the FA algorithm

Parameters

| | |
|---------------------------|--|
| <i>fileOut</i> | file that prints the best solution to a csv |
| <i>filePop</i> | file that prints the population to a csv |
| <i>NS</i> | number of solutions for a data type |
| <i>DIM</i> | number of dimensions for a data type |
| <i>iterations</i> | number of iterations to run the algorithm for |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>fitnessCallCounter</i> | count how many times the fitness function was called |
| <i>myFA</i> | the FA structure |

4.3.1.6 updateIntensity()

```
double updateIntensity (
    FA * myFA,
    int j )
```

Function that updates the intensity of the fireflies.

Function that updates the intensity of the fireflies

Parameters

| | |
|---------------|--|
| <i>xi</i> | firefly vector at i |
| <i>xj</i> | firefly vector at j |
| <i>DIM</i> | number of dimensions for a data type |
| <i>myFA</i> | the PSO structure pointer |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |

4.4 C:/Users/emyli/Desktop/Project4_471/FA.h File Reference

```
#include "Util.h"
```

Data Structures

- struct [_FA1](#)
Structure for the FA algorithm.

Typedefs

- typedef struct [_FA1](#) [FA](#)
Structure for the FA algorithm.

Functions

- void [readInputFA](#) (FILE *input, FILE *fileOut, FILE *filePop, int iterations, int fitnessCallCounter, [initData](#) *myData, int NS, int DIM)
Function that reads from the FA input file to initialize the structure variables for the FA algorithm.
- void [startFA](#) ([FA](#) *myFA, FILE *fileOut, FILE *filePop, int NS, int DIM, int iterations, [initData](#) *myData, int fitnessCallCounter)
Function that runs the FA algorithm.
- double [Eqn2](#) ([FA](#) *myFA)
Function that updates Beta.
- void [Eqn3](#) (const double *xi, const double *xj, int DIM, [FA](#) *myFA)
Function that finds the distance between two fireflies.
- void [Eqn4](#) (const double *xi, const double *xj, int DIM, [FA](#) *myFA, [initData](#) *myData)
Function that creates a new firefly vector, checks if the vector is better than the worst firefly in the population.
- double [updateIntensity](#) ([FA](#) *myFA, int j)
Function that updates the intensity of the fireflies.

4.4.1 Typedef Documentation

4.4.1.1 FA

```
typedef struct _FA1 FA
```

Structure for the FA algorithm.

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

Firefly Algorithm header file. This class implements the methods for performing the Firefly Algorithm. This algorithm models the social behavior of fireflies based on the flashing and attraction characteristics of fireflies. It was developed by Yang in 2010.

4.4.2 Function Documentation

4.4.2.1 Eqn2()

```
double Eqn2 (
    FA * myFA )
```

Function that updates Beta.

Parameters

| | |
|-------------|--------------------------|
| <i>myFA</i> | the FA structure pointer |
|-------------|--------------------------|

Returns

Beta

Function that updates Beta

Parameters

| | |
|-------------|--------------------------|
| <i>myFA</i> | the FA structure pointer |
|-------------|--------------------------|

Returns

Beta

4.4.2.2 Eqn3()

```
void Eqn3 (
    const double * xi,
```

```

    const double * xj,
    int DIM,
    FA * myFA )

```

Function that finds the distance between two fireflies.

Parameters

| | |
|-------------|--------------------------------------|
| <i>xi</i> | firefly vector at i |
| <i>xj</i> | firefly vector at j |
| <i>DIM</i> | number of dimensions for a data type |
| <i>myFA</i> | the PSO structure pointer |

function that moves firefly j towards firefly i; EQN. 3

Parameters

| | |
|-------------|--------------------------------------|
| <i>xi</i> | firefly vector at i |
| <i>xj</i> | firefly vector at j |
| <i>DIM</i> | number of dimensions for a data type |
| <i>myFA</i> | the PSO structure pointer |

4.4.2.3 Eqn4()

```

void Eqn4 (
    const double * xi,
    const double * xj,
    const int DIM,
    FA * myFA,
    initData * myData )

```

Function that creates a new firefly vector, checks if the vector is better than the worst firefly in the population.

Parameters

| | |
|---------------|--|
| <i>xi</i> | firefly vector at i |
| <i>xj</i> | firefly vector at j |
| <i>DIM</i> | number of dimensions for a data type |
| <i>myFA</i> | the PSO structure pointer |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |

Evaluate and update the worst firefly in the population; EQN. 4

Parameters

| | |
|------------|--------------------------------------|
| <i>xi</i> | firefly vector at i |
| <i>xj</i> | firefly vector at j |
| <i>DIM</i> | number of dimensions for a data type |

Parameters

| | |
|---------------|--|
| <i>myFA</i> | the PSO structure pointer |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |

4.4.2.4 readInputFA()

```
void readInputFA (
    FILE * input,
    FILE * fileOut,
    FILE * filePop,
    int iterations,
    int fitnessCallCounter,
    initData * myData,
    int NS,
    int DIM )
```

Function that reads from the FA input file to initialize the structure variables for the FA algorithm.

Parameters

| | |
|---------------------------|--|
| <i>input</i> | file that has the values for the myData structure |
| <i>fileOut</i> | file that prints the best solution to a csv |
| <i>filePop</i> | file that prints the population to a csv |
| <i>iterations</i> | number of iterations to run the algorithm for |
| <i>fitnessCallCounter</i> | count how many times the fitness function was called |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>NS</i> | number of solutions for a data type |
| <i>DIM</i> | number of dimensions for a data type |

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

Firefly Algorithm class file. This class implements the methods for performing the Firefly Algorithm This algorithm models the social behavior of fireflies based on the flashing and attraction characteristics of fireflies. It was developed by Yang in 2010. Function that reads from the FA input file to initialize the structure variables for the FA algorithm

Parameters

| | |
|---------------------------|--|
| <i>input</i> | file that has the values for the myData structure |
| <i>fileOut</i> | file that prints the best solution to a csv |
| <i>filePop</i> | file that prints the population to a csv |
| <i>iterations</i> | number of iterations to run the algorithm for |
| <i>fitnessCallCounter</i> | count how many times the fitness function was called |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>NS</i> | number of solutions for a data type |
| <i>DIM</i> | number of dimensions for a data type |

4.4.2.5 startFA()

```
void startFA (
    FA * myFA,
    FILE * fileOut,
    FILE * filePop,
    const int NS,
    const int DIM,
    const int iterations,
    initData * myData,
    int fitnessCallCounter )
```

Function that runs the FA algorithm.

Parameters

| | |
|---------------------------|--|
| <i>fileOut</i> | file that prints the best solution to a csv |
| <i>filePop</i> | file that prints the population to a csv |
| <i>NS</i> | number of solutions for a data type |
| <i>DIM</i> | number of dimensions for a data type |
| <i>iterations</i> | number of iterations to run the algorithm for |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>fitnessCallCounter</i> | count how many times the fitness function was called |
| <i>myFA</i> | the FA structure |

Function that runs the FA algorithm

Parameters

| | |
|---------------------------|--|
| <i>fileOut</i> | file that prints the best solution to a csv |
| <i>filePop</i> | file that prints the population to a csv |
| <i>NS</i> | number of solutions for a data type |
| <i>DIM</i> | number of dimensions for a data type |
| <i>iterations</i> | number of iterations to run the algorithm for |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>fitnessCallCounter</i> | count how many times the fitness function was called |
| <i>myFA</i> | the FA structure |

4.4.2.6 updateIntensity()

```
double updateIntensity (
    FA * myFA,
    int j )
```

Function that updates the intensity of the fireflies.

Parameters

| | |
|---------------|--|
| <i>xi</i> | firefly vector at i |
| <i>xj</i> | firefly vector at j |
| <i>DIM</i> | number of dimensions for a data type |
| <i>myFA</i> | the PSO structure pointer |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |

Function that updates the intensity of the fireflies

Parameters

| | |
|---------------|--|
| <i>xi</i> | firefly vector at i |
| <i>xj</i> | firefly vector at j |
| <i>DIM</i> | number of dimensions for a data type |
| <i>myFA</i> | the PSO structure pointer |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |

4.5 C:/Users/emyli/Desktop/Project4_471/Functions.c File Reference

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

Functions

- double [schwefel](#) (double *array, int n)
calculate the Schwefel optimization function based on random number inputs
- double [deJong](#) (double *array, int n)
calculate the DeJong 1 optimization function based on random number inputs
- double [rosenbrock](#) (double *array, int n)
calculate the Rosenbrock's Saddle optimization function based on random number inputs
- double [rastrigin](#) (double *array, int n)
calculate the Rastrigin optimization function based on random number inputs
- double [griewangk](#) (double *array, int n)
calculate the Griewangk optimization function based on random number inputs
- double [sineEnvelope](#) (double *array, int n)
calculate the Sine Envelope Sine Wave optimization function based on random number inputs
- double [sineStretched](#) (double *array, int n)
calculate the Stretch V Sine Wave optimization function based on random number inputs
- double [ackley1](#) (double *array, int n)
calculate the Ackley One optimization function based on random number inputs
- double [ackley2](#) (double *array, int n)
calculate the Ackley Two optimization function based on random number inputs
- double [eggHolder](#) (double *array, int n)
calculate the Egg Holder optimization function based on random number inputs
- double [rana](#) (double *array, int n)

- calculate the Rana optimization function based on random number inputs*
- double [pathological](#) (double *array, int n)
calculate the Pathological optimization function based on random number inputs
- double [michalewicz](#) (double *array, int n)
calculate the Michalewicz optimization function based on random number inputs
- double [masters](#) (double *array, int n)
calculate the Master's Cosine Wave optimization function based on random number inputs
- double [quartic](#) (double *array, int n)
calculate the Quartic optimization function based on random number inputs
- double [levy](#) (double *array, int n)
calculate the Levy optimization function based on random number inputs
- double [step](#) (double *array, int n)
calculate the Step optimization function based on random number inputs
- double [alpine](#) (double *array, int n)
calculate the Alpine optimization function based on random number inputs

4.5.1 Function Documentation

4.5.1.1 `ackley1()`

```
double ackley1 (
    double * array,
    int n )
```

calculate the Ackley One optimization function based on random number inputs

calculate the Ackley One optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.5.1.2 `ackley2()`

```
double ackley2 (
    double * array,
    int n )
```

calculate the Ackley Two optimization function based on random number inputs

calculate the Ackley Two optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.5.1.3 alpine()

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

calculate the Alpine optimization function based on random number inputs

calculate the Alpine optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.5.1.4 deJong()

```
double deJong (  
    double * array,  
    int n )
```

calculate the DeJong 1 optimization function based on random number inputs

calculate the DeJong 1 optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.5.1.5 eggHolder()

```
double eggHolder (
    double * array,
    int n )
```

calculate the Egg Holder optimization function based on random number inputs

calculate the Egg Holder optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.5.1.6 griewangk()

```
double griewangk (
    double * array,
    int n )
```

calculate the Griewangk optimization function based on random number inputs

calculate the Griewangk optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the product and summation

4.5.1.7 levy()

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

calculate the Levy optimization function based on random number inputs

calculate the Levy optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.5.1.8 masters()

```
double masters (  
    double * array,  
    int n )
```

calculate the Master's Cosine Wave optimization function based on random number inputs

calculate the Master's Cosine Wave optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.5.1.9 michalewicz()

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

calculate the Michalewicz optimization function based on random number inputs

calculate the Michalewicz optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.5.1.10 pathological()

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

calculate the Pathological optimization function based on random number inputs

calculate the Pathological optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.5.1.11 quartic()

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

calculate the Quartic optimization function based on random number inputs

calculate the Quartic optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.5.1.12 rana()

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

calculate the Rana optimization function based on random number inputs

calculate the Rana optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.5.1.13 rastrigin()

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

calculate the Rastrigin optimization function based on random number inputs

calculate the Rastrigin optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.5.1.14 rosenbrock()

```
double rosenbrock (
    double * array,
    int n )
```

calculate the Rosenbrock's Saddle optimization function based on random number inputs

calculate the Rosenbrock's Saddle optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.5.1.15 schwefel()

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

calculate the Schwefel optimization function based on random number inputs

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

This project implements three meta-heuristic optimization algorithms. Particle Swarm Optimization (PSO), Firefly Algorithm (FA), and Harmony Search Algorithm (HS).calculate the Schwefel optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.5.1.16 sineEnvelope()

```
double sineEnvelope (
    double * array,
    int n )
```

calculate the Sine Envelope Sine Wave optimization function based on random number inputs

calculate the Sine Envelope Sine Wave optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.5.1.17 sineStretched()

```
double sineStretched (
    double * array,
    int n )
```

calculate the Stretch V Sine Wave optimization function based on random number inputs

calculate the Stretch V Sine Wave optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.5.1.18 step()

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

calculate the Step optimization function based on random number inputs

calculate the Step optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.6 C:/Users/emyli/Desktop/Project4_471/Functions.h File Reference

Functions

- double [schwefel](#) (double *array, int n)
calculate the Schwefel optimization function based on random number inputs
- double [deJong](#) (double *array, int n)
calculate the DeJong 1 optimization function based on random number inputs
- double [rosenbrock](#) (double *array, int n)
calculate the Rosenbrock's Saddle optimization function based on random number inputs
- double [rastrigin](#) (double *array, int n)
calculate the Rastrigin optimization function based on random number inputs
- double [griewangk](#) (double *array, int n)
calculate the Griewangk optimization function based on random number inputs
- double [sineEnvelope](#) (double *array, int n)
calculate the Sine Envelope Sine Wave optimization function based on random number inputs
- double [sineStretched](#) (double *array, int n)
calculate the Stretch V Sine Wave optimization function based on random number inputs
- double [ackley1](#) (double *array, int n)
calculate the Ackley One optimization function based on random number inputs
- double [ackley2](#) (double *array, int n)
calculate the Ackley Two optimization function based on random number inputs
- double [eggHolder](#) (double *array, int n)
calculate the Egg Holder optimization function based on random number inputs
- double [rana](#) (double *array, int n)
calculate the Rana optimization function based on random number inputs
- double [pathological](#) (double *array, int n)
calculate the Pathological optimization function based on random number inputs
- double [michalewicz](#) (double *array, int n)
calculate the Michalewicz optimization function based on random number inputs
- double [masters](#) (double *array, int n)
calculate the Master's Cosine Wave optimization function based on random number inputs
- double [quartic](#) (double *array, int n)
calculate the Quartic optimization function based on random number inputs
- double [levy](#) (double *array, int n)
calculate the Levy optimization function based on random number inputs
- double [step](#) (double *array, int n)
calculate the Step optimization function based on random number inputs
- double [alpine](#) (double *array, int n)
calculate the Alpine optimization function based on random number inputs

4.6.1 Function Documentation

4.6.1.1 ackley1()

```
double ackley1 (  
    double * array,  
    int n )
```

calculate the Ackley One optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

calculate the Ackley One optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.6.1.2 ackley2()

```
double ackley2 (  
    double * array,  
    int n )
```

calculate the Ackley Two optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

calculate the Ackley Two optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.6.1.3 alpine()

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

calculate the Alpine optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

calculate the Alpine optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.6.1.4 deJong()

```
double deJong (
    double * array,
    int n )
```

calculate the DeJong 1 optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

calculate the DeJong 1 optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.6.1.5 eggHolder()

```
double eggHolder (
    double * array,
    int n )
```

calculate the Egg Holder optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

calculate the Egg Holder optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.6.1.6 griewangk()

```
double griewangk (
    double * array,
    int n )
```

calculate the Griewangk optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the product and summation

calculate the Griewangk optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the product and summation

4.6.1.7 levy()

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

calculate the Levy optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

calculate the Levy optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.6.1.8 masters()

```
double masters (  
    double * array,  
    int n )
```

calculate the Master's Cosine Wave optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

calculate the Master's Cosine Wave optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.6.1.9 michalewicz()

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

calculate the Michalewicz optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

calculate the Michalewicz optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.6.1.10 pathological()

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

calculate the Pathological optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

calculate the Pathological optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.6.1.11 quartic()

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

calculate the Quartic optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

calculate the Quartic optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.6.1.12 rana()

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

calculate the Rana optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

calculate the Rana optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.6.1.13 rastrigin()

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

calculate the Rastrigin optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

calculate the Rastrigin optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.6.1.14 rosenbrock()

```
double rosenbrock (  
    double * array,  
    int n )
```

calculate the Rosenbrock's Saddle optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

calculate the Rosenbrock's Saddle optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.6.1.15 schwefel()

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

calculate the Schwefel optimization function based on random number inputs

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

This project implements three meta-heuristic optimization algorithms. Particle Swarm Optimization (PSO), Firefly Algorithm (FA), and Harmony Search Algorithm (HS).

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

This project implements three meta-heuristic optimization algorithms. Particle Swarm Optimization (PSO), Firefly Algorithm (FA), and Harmony Search Algorithm (HS).calculate the Schwefel optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.6.1.16 sineEnvelope()

```
double sineEnvelope (
    double * array,
    int n )
```

calculate the Sine Envelope Sine Wave optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

calculate the Sine Envelope Sine Wave optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.6.1.17 sineStretched()

```
double sineStretched (  
    double * array,  
    int n )
```

calculate the Stretch V Sine Wave optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

calculate the Stretch V Sine Wave optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.6.1.18 step()

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

calculate the Step optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

calculate the Step optimization function based on random number inputs

Parameters

| | |
|--------------|-----------------------------------|
| <i>array</i> | single array |
| <i>n</i> | the size of the rows of the array |

Returns

final calculated number from the summation

4.7 C:/Users/emyli/Desktop/Project4_471/HS.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
#include <time.h>
#include <math.h>
#include "Util.h"
#include "HS.h"
#include "ArrayMem.h"
#include "SelectFunction.h"
#include "mt19937ar.h"
```

Functions

- void [readInputHS](#) (FILE *input, FILE *fileOut, FILE *filePop, int iterations, int fitnessCallCounter, [initData](#) *myData, int NS, int DIM)
- void [startHS](#) ([HS](#) *myHS, FILE *fileOut, FILE *filePop, const int NS, const int DIM, const int iterations, [initData](#) *myData, int fitnessCallCounter, double *holdb, double *holdw, double *replace)
- void [updateWithPAR](#) ([HS](#) *myHS, [initData](#) *myData, int k)
- void [checkBounds](#) (int k, [initData](#) *myData, [HS](#) *myHS)
- void [updateHM](#) ([HS](#) *myHS, int NS, int DIM, [initData](#) *myData)

4.7.1 Function Documentation

4.7.1.1 checkBounds()

```
void checkBounds (
    int k,
    initData * myData,
    HS * myHS )
```

Function that checks the values within the vector to see if it within the f(x) bounds

Parameters

| | |
|---------------|--|
| <i>k</i> | position of the HS vector |
| <i>myHS</i> | the HS structure pointer |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |

4.7.1.2 readInputHS()

```
void readInputHS (
    FILE * input,
    FILE * fileOut,
    FILE * filePop,
    int iterations,
    int fitnessCallCounter,
    initData * myData,
    int NS,
    int DIM )
```

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

Harmony Search Algorithm header file. This class implements the methods for performing the Harmony Search Algorithm This algorithm models the music improvisation process where the musicians improvise their instruments' pitch by searching for a perfect state of harmony (Geem et al. 2001)Function that reads from the HS input file to initialize the structure variables for the HS algorithm

Parameters

| | |
|---------------------------|--|
| <i>input</i> | file that has the values for the myData structure |
| <i>fileOut</i> | file that prints the best solution to a csv |
| <i>filePop</i> | file that prints the population to a csv |
| <i>iterations</i> | number of iterations to run the algorithm for |
| <i>fitnessCallCounter</i> | count how many times the fitness function was called |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>NS</i> | number of solutions for a data type |
| <i>DIM</i> | number of dimensions for a data type |

4.7.1.3 startHS()

```
void startHS (
    HS * myHS,
    FILE * fileOut,
    FILE * filePop,
    const int NS,
    const int DIM,
    const int iterations,
    initData * myData,
    int fitnessCallCounter,
    double * holdb,
    double * holdw,
    double * replace )
```

Function that runs the HS algorithm

Parameters

| | |
|---------------------------|--|
| <i>fileOut</i> | file that prints the best solution to a csv |
| <i>filePop</i> | file that prints the population to a csv |
| <i>NS</i> | number of solutions for a data type |
| <i>DIM</i> | number of dimensions for a data type |
| <i>iterations</i> | number of iterations to run the algorithm for |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>fitnessCallCounter</i> | count how many times the fitness function was called |
| <i>holdb, holds</i> | the best solutions |
| <i>holdw</i> | holds the worst solutions |
| <i>replace</i> | variable to hold a new arrays elements |

4.7.1.4 updateHM()

```
void updateHM (
    HS * myHS,
    int NS,
    int DIM,
    initData * myData )
```

Function that updates the population if the vector is better than the worst vector in the population

Parameters

| | |
|---------------|--|
| <i>myHS</i> | the HS structure pointer |
| <i>DIM</i> | number of dimensions |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |

4.7.1.5 updateWithPAR()

```
void updateWithPAR (
    HS * myHS,
    initData * myData,
    int k )
```

Function that updates the values from HS vector

Parameters

| | |
|---------------|--|
| <i>myHS</i> | the HS structure pointer |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>k</i> | position of the HS vector |

4.8 C:/Users/emyli/Desktop/Project4_471/HS.h File Reference

```
#include "Util.h"
```

Data Structures

- struct [_HS1](#)
Structure for the HS algorithm.

Typedefs

- typedef struct [_HS1](#) HS
Structure for the HS algorithm.

Functions

- void [readInputHS](#) (FILE *input, FILE *fileOut, FILE *filePop, int iterations, int fitnessCallCounter, [initData](#) *myData, int NS, int DIM)
- void [startHS](#) (HS *myHS, FILE *fileOut, FILE *filePop, int NS, int DIM, int iterations, [initData](#) *myData, int fitnessCallCounter, double *holdb, double *holdw, double *replace)
- void [updateWithPAR](#) (HS *myHS, [initData](#) *myData, int k)
- void [checkBounds](#) (int k, [initData](#) *myData, HS *myHS)
- void [updateHM](#) (HS *myHS, int NS, int DIM, [initData](#) *myData)

4.8.1 Typedef Documentation

4.8.1.1 HS

```
typedef struct _HS1 HS
```

Structure for the HS algorithm.

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

Harmony Search Algorithm header file. This class implements the methods for performing the Harmony Search Algorithm. This algorithm models the music improvisation process where the musicians improvise their instruments' pitch by searching for a perfect state of harmony (Geem et al. 2001)

4.8.2 Function Documentation

4.8.2.1 checkBounds()

```
void checkBounds (
    int k,
    initData * myData,
    HS * myHS )
```

Function that checks the values within the vector to see if it is within the $f(x)$ bounds

Parameters

| | |
|---------------|--|
| <i>k</i> | position of the HS vector |
| <i>myHS</i> | the HS structure pointer |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |

4.8.2.2 readInputHS()

```
void readInputHS (
    FILE * input,
    FILE * fileOut,
    FILE * filePop,
    int iterations,
    int fitnessCallCounter,
    initData * myData,
    int NS,
    int DIM )
```

Function that reads from the HS input file to initialize the structure variables for the HS algorithm

Parameters

| | |
|---------------------------|--|
| <i>input</i> | file that has the values for the myData structure |
| <i>fileOut</i> | file that prints the best solution to a csv |
| <i>filePop</i> | file that prints the population to a csv |
| <i>iterations</i> | number of iterations to run the algorithm for |
| <i>fitnessCallCounter</i> | count how many times the fitness function was called |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>NS</i> | number of solutions for a data type |
| <i>DIM</i> | number of dimensions for a data type |

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

Harmony Search Algorithm header file. This class implements the methods for performing the Harmony Search Algorithm. This algorithm models the music improvisation process where the musicians improvise their instruments' pitch by searching for a perfect state of harmony (Geem et al. 2001). Function that reads from the HS input file to initialize the structure variables for the HS algorithm.

Parameters

| | |
|---------------------------|--|
| <i>input</i> | file that has the values for the myData structure |
| <i>fileOut</i> | file that prints the best solution to a csv |
| <i>filePop</i> | file that prints the population to a csv |
| <i>iterations</i> | number of iterations to run the algorithm for |
| <i>fitnessCallCounter</i> | count how many times the fitness function was called |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>NS</i> | number of solutions for a data type |
| <i>DIM</i> | number of dimensions for a data type |

4.8.2.3 startHS()

```
void startHS (
    HS * myHS,
    FILE * fileOut,
    FILE * filePop,
    const int NS,
    const int DIM,
    const int iterations,
    initData * myData,
    int fitnessCallCounter,
    double * holdb,
    double * holdw,
    double * replace )
```

Function that runs the HS algorithm

Parameters

| | |
|---------------------------|--|
| <i>fileOut</i> | file that prints the best solution to a csv |
| <i>filePop</i> | file that prints the population to a csv |
| <i>NS</i> | number of solutions for a data type |
| <i>DIM</i> | number of dimensions for a data type |
| <i>iterations</i> | number of iterations to run the algorithm for |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>fitnessCallCounter</i> | count how many times the fitness function was called |
| <i>holdb, holds</i> | the best solutions |
| <i>holdw</i> | holds the worst solutions |
| <i>replace</i> | variable to hold a new arrays elements |

4.8.2.4 updateHM()

```
void updateHM (
    HS * myHS,
    int NS,
    int DIM,
    initData * myData )
```

Function that updates the population if the vector is better than the worst vector in the population

Parameters

| | |
|---------------|--|
| <i>myHS</i> | the HS structure pointer |
| <i>DIM</i> | number of dimensions |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |

4.8.2.5 updateWithPAR()

```
void updateWithPAR (
    HS * myHS,
    initData * myData,
    int k )
```

Function that updates the values from HS vector

Parameters

| | |
|---------------|--|
| <i>myHS</i> | the HS structure pointer |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>k</i> | position of the HS vector |

4.9 C:/Users/emyli/Desktop/Project4_471/main.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include "ArrayMem.h"
#include "SelectFunction.h"
#include "Util.h"
#include "mt19937ar.h"
#include "PSO.h"
#include "FA.h"
#include "HS.h"
```

Functions

- int [main](#) ()

4.9.1 Function Documentation

4.9.1.1 main()

```
int main ( )
```

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

This project implements three meta-heuristic optimization algorithms. Particle Swarm Optimization (PSO), Firefly Algorithm (FA), and Harmony Search Algorithm (HS).main method that performs PSO, FA, and HS

Returns

0

< number of solutions

< number of dimensions

< number of iterations

< function call counter

4.10 C:/Users/emyli/Desktop/Project4_471/mt19937ar.c File Reference

```
#include <stdio.h>
#include "mt19937ar.h"
```

Macros

- `#define N 624`
- `#define M 397`
- `#define MATRIX_A 0x9908b0dfUL /* constant vector a */`
- `#define UPPER_MASK 0x80000000UL /* most significant w-r bits */`
- `#define LOWER_MASK 0x7fffffffUL /* least significant r bits */`

Functions

- void `init_genrand` (unsigned long s)
- void `init_by_array` (unsigned long init_key[], int key_length)
- unsigned long `genrand_int32` (void)
- long `genrand_int31` (void)
- double `genrand_real1` (void)
- double `genrand_real2` (void)
- double `genrand_real3` (void)
- double `genrand_res53` (void)

Variables

- static unsigned long `mt` [N]
- static int `mti` =N+1

4.10.1 Macro Definition Documentation

4.10.1.1 LOWER_MASK

```
#define LOWER_MASK 0x7fffffffUL /* least significant r bits */
```

4.10.1.2 M

```
#define M 397
```

4.10.1.3 MATRIX_A

```
#define MATRIX_A 0x9908b0dfUL /* constant vector a */
```

4.10.1.4 N

```
#define N 624
```

4.10.1.5 UPPER_MASK

```
#define UPPER_MASK 0x80000000UL /* most significant w-r bits */
```

4.10.2 Function Documentation

4.10.2.1 genrand_int31()

```
long genrand_int31 (  
    void )
```

4.10.2.2 genrand_int32()

```
unsigned long genrand_int32 (  
    void )
```

4.10.2.3 genrand_real1()

```
double genrand_real1 (  
    void )
```

4.10.2.4 genrand_real2()

```
double genrand_real2 (  
    void )
```

4.10.2.5 genrand_real3()

```
double genrand_real3 (  
    void )
```

4.10.2.6 genrand_res53()

```
double genrand_res53 (  
    void )
```

4.10.2.7 init_by_array()

```
void init_by_array (  
    unsigned long init_key[],  
    int key_length )
```

4.10.2.8 init_genrand()

```
void init_genrand (  
    unsigned long s )
```

4.10.3 Variable Documentation

4.10.3.1 mt

```
unsigned long mt[N] [static]
```

4.10.3.2 mti

```
int mti =N+1 [static]
```

4.11 C:/Users/emyli/Desktop/Project4_471/mt19937ar.h File Reference

Functions

- void [init_genrand](#) (unsigned long s)
- void [init_by_array](#) (unsigned long init_key[], int key_length)
- unsigned long [genrand_int32](#) (void)
- long [genrand_int31](#) (void)
- double [genrand_real1](#) (void)
- double [genrand_real2](#) (void)
- double [genrand_real3](#) (void)
- double [genrand_res53](#) (void)

4.11.1 Function Documentation

4.11.1.1 [genrand_int31\(\)](#)

```
long genrand_int31 (  
    void )
```

4.11.1.2 [genrand_int32\(\)](#)

```
unsigned long genrand_int32 (  
    void )
```

4.11.1.3 [genrand_real1\(\)](#)

```
double genrand_real1 (  
    void )
```

4.11.1.4 [genrand_real2\(\)](#)

```
double genrand_real2 (  
    void )
```

4.11.1.5 genrand_real3()

```
double genrand_real3 (
    void )
```

4.11.1.6 genrand_res53()

```
double genrand_res53 (
    void )
```

4.11.1.7 init_by_array()

```
void init_by_array (
    unsigned long init_key[],
    int key_length )
```

4.11.1.8 init_genrand()

```
void init_genrand (
    unsigned long s )
```

4.12 C:/Users/emyli/Desktop/Project4_471/PSO.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
#include <time.h>
#include <math.h>
#include "ArrayMem.h"
#include "PSO.h"
#include "Util.h"
#include "SelectFunction.h"
#include "mt19937ar.h"
```

Functions

- void [readInput](#) (FILE *input, FILE *fileOut, FILE *filePop, int iterations, int fitnessCallCounter, [initData](#) *myData, int NS, int DIM)
Function that reads from the PSO input file to initialize the structure variables for the PSO algorithm.
- void [startPSO](#) (FILE *fileOut, FILE *filePop, int NS, int DIM, int iterations, [initData](#) *myData, int fitnessCallCounter, [PSO](#) *myPSO)
Function that runs the PSO algorithm.
- void [calcVelocity](#) ([PSO](#) *myPSO, [initData](#) *myData, int DIM, int j)
Function that updates the velocity for the PSO algorithm.
- void [calcPopulation](#) ([PSO](#) *myPSO, [initData](#) *myData, int DIM, int j)
Function that updates the population of our PSO algorithm.

4.12.1 Function Documentation

4.12.1.1 calcPopulation()

```
void calcPopulation (
    PSO * myPSO,
    initData * myData,
    int DIM,
    int j )
```

Function that updates the population of our PSO algorithm.

Function that updates the population of our PSO algorithm

Parameters

| | |
|---------------|--|
| <i>myPSO</i> | the PSO structure pointer |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>DIM</i> | number of dimensions for a data type |
| <i>j</i> | position of the vector from the population matrix |

4.12.1.2 calcVelocity()

```
void calcVelocity (
    PSO * myPSO,
    initData * myData,
    int DIM,
    int j )
```

Function that updates the velocity for the PSO algorithm.

Function that updates the velocity for the PSO algorithm

Parameters

| | |
|---------------|--|
| <i>myPSO</i> | the PSO structure pointer |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>DIM</i> | number of dimensions for a data type |
| <i>j</i> | position of the vector from the population matrix |

4.12.1.3 readInput()

```
void readInput (
    FILE * input,
```

```

FILE * fileOut,
FILE * filePop,
int iterations,
int fitnessCallCounter,
initData * myData,
int NS,
int DIM )

```

Function that reads from the PSO input file to initialize the structure variables for the PSO algorithm.

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

Particle Swarm Optimization class file. This class implements the methods for performing Particle Swarm Optimization. This algorithm models flocking and schooling patterns of birds and fishes. It was invented by Russell Eberhart and James Kennedy in 1995. Function that reads from the PSO input file to initialize the structure variables for the PSO algorithm

Parameters

| | |
|---------------------------|--|
| <i>input</i> | file that has the values for the myData structure |
| <i>fileOut</i> | file that prints the best solution to a csv |
| <i>filePop</i> | file that prints the population to a csv |
| <i>iterations</i> | number of iterations to run the algorithm for |
| <i>fitnessCallCounter</i> | count how many times the fitness function was called |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>NS</i> | number of solutions for a data type |
| <i>DIM</i> | number of dimensions for a data type |

4.12.1.4 startPSO()

```

void startPSO (
    FILE * fileOut,
    FILE * filePop,
    int NS,
    int DIM,
    int iterations,
    initData * myData,
    int fitnessCallCounter,
    PSO * myPSO )

```

Function that runs the PSO algorithm.

Function that runs the PSO algorithm

Parameters

| | |
|----------------|---|
| <i>fileOut</i> | output file for the fitness of the population |
| <i>filePop</i> | file that prints the population to a csv |

Parameters

| | |
|---------------------------|--|
| <i>NS</i> | number of solutions for a data type |
| <i>DIM</i> | number of dimensions for a data type |
| <i>iterations</i> | number of iterations to run the algorithm for |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>fitnessCallCounter</i> | count how many times the fitness function was called |
| <i>myPSO</i> | the PSO structure |

4.13 C:/Users/emyli/Desktop/Project4_471/PSO.h File Reference

```
#include "Util.h"
```

Data Structures

- [struct _PSO1](#)
Structure for the PSO algorithm.

Typedefs

- [typedef struct _PSO1 PSO](#)
Structure for the PSO algorithm.

Functions

- void [readInput](#) (FILE *input, FILE *fileOut, FILE *filePop, int iterations, int fitnessCallCounter, [initData](#) *myData, int NS, int DIM)
Function that reads from the PSO input file to initialize the structure variables for the PSO algorithm.
- void [startPSO](#) (FILE *fileFit, FILE *filePop, int NS, int DIM, int iterations, [initData](#) *myData, int fitnessCallCounter, [PSO](#) *myPSO)
Function that runs the PSO algorithm.
- void [calcVelocity](#) ([PSO](#) *myPSO, [initData](#) *myData, int DIM, int j)
Function that updates the velocity for the PSO algorithm.
- void [calcPopulation](#) ([PSO](#) *myPSO, [initData](#) *myData, int DIM, int j)
Function that updates the population of our PSO algorithm.

4.13.1 Typedef Documentation

4.13.1.1 PSO

```
typedef struct _PSO1 PSO
```

Structure for the PSO algorithm.

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

Particle Swarm Optimization header file. This class implements the methods for performing Particle Swarm Optimization. This algorithm models flocking and schooling patterns of birds and fishes. It was invented by Russell Eberhart and James Kennedy in 1995.

4.13.2 Function Documentation

4.13.2.1 calcPopulation()

```
void calcPopulation (
    PSO * myPSO,
    initData * myData,
    int DIM,
    int j )
```

Function that updates the population of our PSO algorithm.

Parameters

| | |
|---------------|--|
| <i>myPSO</i> | the PSO structure pointer |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>DIM</i> | number of dimensions for a data type |
| <i>j</i> | position of the vector from the population matrix |

Function that updates the population of our PSO algorithm

Parameters

| | |
|---------------|--|
| <i>myPSO</i> | the PSO structure pointer |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>DIM</i> | number of dimensions for a data type |
| <i>j</i> | position of the vector from the population matrix |

4.13.2.2 calcVelocity()

```
void calcVelocity (
    PSO * myPSO,
    initData * myData,
    int DIM,
    int j )
```

Function that updates the velocity for the PSO algorithm.

Parameters

| | |
|---------------|--|
| <i>myPSO</i> | the PSO structure pointer |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>DIM</i> | number of dimensions for a data type |
| <i>j</i> | position of the vector from the population matrix |

Function that updates the velocity for the PSO algorithm

Parameters

| | |
|---------------|--|
| <i>myPSO</i> | the PSO structure pointer |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>DIM</i> | number of dimensions for a data type |
| <i>j</i> | position of the vector from the population matrix |

4.13.2.3 readInput()

```
void readInput (
    FILE * input,
    FILE * fileOut,
    FILE * filePop,
    int iterations,
    int fitnessCallCounter,
    initData * myData,
    int NS,
    int DIM )
```

Function that reads from the PSO input file to initialize the structure variables for the PSO algorithm.

Parameters

| | |
|---------------------------|--|
| <i>input</i> | file that has the values for the myData structure |
| <i>fileOut</i> | file that has the values for myPSO structure |
| <i>filePop</i> | file that prints the population to a csv |
| <i>iterations</i> | number of iterations to run the algorithm for |
| <i>fitnessCallCounter</i> | count how many times the fitness function was called |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>NS</i> | number of solutions for a data type |
| <i>DIM</i> | number of dimensions for a data type |

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

Particle Swarm Optimization class file. This class implements the methods for performing Particle Swarm Optimization. This algorithm models flocking and schooling patterns of birds and fishes. It was invented by Russell Eberhart and James Kennedy in 1995. Function that reads from the PSO input file to initialize the structure variables for the PSO algorithm

Parameters

| | |
|---------------------------|--|
| <i>input</i> | file that has the values for the myData structure |
| <i>fileOut</i> | file that prints the best solution to a csv |
| <i>filePop</i> | file that prints the population to a csv |
| <i>iterations</i> | number of iterations to run the algorithm for |
| <i>fitnessCallCounter</i> | count how many times the fitness function was called |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>NS</i> | number of solutions for a data type |
| <i>DIM</i> | number of dimensions for a data type |

4.13.2.4 startPSO()

```
void startPSO (
    FILE * fileOut,
    FILE * filePop,
    int NS,
    int DIM,
    int iterations,
    initData * myData,
    int fitnessCallCounter,
    PSO * myPSO )
```

Function that runs the PSO algorithm.

Parameters

| | |
|---------------------------|--|
| <i>fileOut</i> | output file for the fitness of the population |
| <i>filePop</i> | file that prints the population to a csv |
| <i>NS</i> | number of solutions for a data type |
| <i>DIM</i> | number of dimensions for a data type |
| <i>iterations</i> | number of iterations to run the algorithm for |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>fitnessCallCounter</i> | count how many times the fitness function was called |
| <i>myPSO</i> | the PSO structure |

Function that runs the PSO algorithm

Parameters

| | |
|---------------------------|--|
| <i>fileOut</i> | output file for the fitness of the population |
| <i>filePop</i> | file that prints the population to a csv |
| <i>NS</i> | number of solutions for a data type |
| <i>DIM</i> | number of dimensions for a data type |
| <i>iterations</i> | number of iterations to run the algorithm for |
| <i>myData</i> | Data structure pointer for initializing the bounds and which function to run |
| <i>fitnessCallCounter</i> | count how many times the fitness function was called |
| <i>myPSO</i> | the PSO structure |

4.14 C:/Users/emyli/Desktop/Project4_471/SelectFunction.c File Reference

```
#include "Functions.h"
```

Functions

- double * [getFun](#) (double *results, double **arr, int row, int col, int counter)
Calls an f(x) function based on if the counter is at a certain number The arr is passed to the function and saved into a single pointer array.
- double [getFunSingle](#) (double results, double *arr, int row, int counter)
Calls an f(x) function based on if the counter is at a certain number The arr is passed to the function and saved into a single pointer array.

4.14.1 Function Documentation

4.14.1.1 [getFun\(\)](#)

```
double* getFun (
    double * results,
    double ** arr,
    int row,
    int col,
    int counter )
```

Calls an f(x) function based on if the counter is at a certain number The arr is passed to the function and saved into a single pointer array.

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

This project implements three meta-heuristic optimization algorithms. Particle Swarm Optimization (PSO), Firefly Algorithm (FA), and Harmony Search Algorithm (HS). Calls an f(x) function based on if the counter is at a certain number The arr is passed to the function and saved into a single pointer array

Parameters

| | |
|----------------|--------------------------------------|
| <i>results</i> | single pointer array |
| <i>arr</i> | double pointer array |
| <i>row</i> | the size of the rows of the array |
| <i>col</i> | the size of the columns of the array |
| <i>counter</i> | the case specified |

4.14.1.2 getFunSingle()

```
double getFunSingle (
    double results,
    double * arr,
    int row,
    int counter )
```

Calls an f(x) function based on if the counter is at a certain number The arr is passed to the function and saved into a single pointer array.

Calls an f(x) function based on if the counter is at a certain number The arr is passed to the function and saved into a single pointer array

Parameters

| | |
|----------------|-----------------------------------|
| <i>results</i> | a double |
| <i>arr</i> | single pointer array |
| <i>row</i> | the size of the rows of the array |
| <i>counter</i> | the case specified |

4.15 C:/Users/emyli/Desktop/Project4_471/SelectFunction.h File Reference**Functions**

- double * [getFun](#) (double *results, double **arr, int row, int col, int counter)

Calls an f(x) function based on if the counter is at a certain number The arr is passed to the function and saved into a single pointer array.

- double [getFunSingle](#) (double results, double *arr, int row, int counter)

Calls an f(x) function based on if the counter is at a certain number The arr is passed to the function and saved into a single pointer array.

4.15.1 Function Documentation

4.15.1.1 getFun()

```
double* getFun (
    double * results,
    double ** arr,
    int row,
    int col,
    int counter )
```

Calls an f(x) function based on if the counter is at a certain number The arr is passed to the function and saved into a single pointer array.

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

This project implements three meta-heuristic optimization algorithms. Particle Swarm Optimization (PSO), Firefly Algorithm (FA), and Harmony Search Algorithm (HS).

Parameters

| | |
|----------------|--------------------------------------|
| <i>results</i> | single pointer array |
| <i>arr</i> | double pointer array |
| <i>row</i> | the size of the rows of the array |
| <i>col</i> | the size of the columns of the array |
| <i>counter</i> | the case specified |

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

This project implements three meta-heuristic optimization algorithms. Particle Swarm Optimization (PSO), Firefly Algorithm (FA), and Harmony Search Algorithm (HS). Calls an f(x) function based on if the counter is at a certain number The arr is passed to the function and saved into a single pointer array

Parameters

| | |
|----------------|--------------------------------------|
| <i>results</i> | single pointer array |
| <i>arr</i> | double pointer array |
| <i>row</i> | the size of the rows of the array |
| <i>col</i> | the size of the columns of the array |
| <i>counter</i> | the case specified |

4.15.1.2 getFunSingle()

```
double getFunSingle (
    double results,
```

```
double * arr,
int row,
int counter )
```

Calls an f(x) function based on if the counter is at a certain number The arr is passed to the function and saved into a single pointer array.

Parameters

| | |
|----------------|-----------------------------------|
| <i>results</i> | a double |
| <i>arr</i> | single pointer array |
| <i>row</i> | the size of the rows of the array |
| <i>counter</i> | the case specified |

Calls an f(x) function based on if the counter is at a certain number The arr is passed to the function and saved into a single pointer array

Parameters

| | |
|----------------|-----------------------------------|
| <i>results</i> | a double |
| <i>arr</i> | single pointer array |
| <i>row</i> | the size of the rows of the array |
| <i>counter</i> | the case specified |

4.16 C:/Users/emyli/Desktop/Project4_471/Util.c File Reference

```
#include <float.h>
#include <stdio.h>
#include "Util.h"
```

Functions

- void [sortAscendingOrder](#) (initData *myData, const int NS, const int DIM, double *b)
- double * [replaceArray](#) (double *a, const double *b, const int NS)
- double [findBest](#) (const double *a, int NS)
- double ** [copyDbl](#) (double **a, double **b, int NS, int DIM)
- double * [copySingle](#) (double *a, const double *b, int DIM)
- void [printDblDim](#) (FILE *file, double **a, int DIM, int NS)
- void [printSingle](#) (FILE *file, double *a, int DIM)

4.16.1 Function Documentation

4.16.1.1 copyDbl()

```
double** copyDbl (
    double ** a,
    double ** b,
    int NS,
    int DIM )
```

copy a arrays contents to another

Parameters

| | |
|------------|---|
| <i>a</i> | a double array |
| <i>b</i> | a double array |
| <i>DIM</i> | the number of dimensions/size of a vector |
| <i>NS</i> | the number of solutions/size of a vector |

Returns

a double array with double array b's contents placed into

4.16.1.2 copySingle()

```
double* copySingle (
    double * a,
    const double * b,
    int DIM )
```

copy a arrays contents to another

Parameters

| | |
|------------|---|
| <i>a</i> | an array |
| <i>b</i> | an array |
| <i>DIM</i> | the number of dimensions/size of a vector |

Returns

a array with array b's contents placed into

4.16.1.3 findBest()

```
double findBest (
    const double * a,
    int NS )
```

find the minimal value in a vector

Parameters

| | |
|-----------|--|
| <i>a</i> | an array |
| <i>NS</i> | the number of solutions/size of a vector |

Returns

a array with array b's elements implemented

4.16.1.4 printDbldim()

```
void printDbldim (
    FILE * file,
    double ** a,
    int DIM,
    int NS )
```

print out a matrix with dimensions printed out horizontally to a file

Parameters

| | |
|-------------|---|
| <i>file</i> | file to print to |
| <i>a</i> | an array |
| <i>DIM</i> | the number of dimensions/size of a vector |
| <i>NS</i> | the number of solutions/size of a vector |

4.16.1.5 printSingle()

```
void printSingle (
    FILE * file,
    double * a,
    int DIM )
```

print out a vector to a file

Parameters

| | |
|-------------|---|
| <i>file</i> | a file to print to |
| <i>a</i> | an array |
| <i>DIM</i> | the number of dimensions/size of a vector |

4.16.1.6 replaceArray()

```
double* replaceArray (
    double * a,
    const double * b,
    const int NS )
```

replace the elements of one array into another

Parameters

| | |
|-----------|--|
| <i>a</i> | an array |
| <i>b</i> | an array |
| <i>NS</i> | the number of solutions/size of a vector |

Returns

a array with array b's elements implemented

4.16.1.7 sortAscendingOrder()

```
void sortAscendingOrder (
    initData * myData,
    const int NS,
    const int DIM,
    double * b )
```

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

This project implements three meta-heuristic optimization algorithms. Particle Swarm Optimization (PSO), Firefly Algorithm (FA), and Harmony Search Algorithm (HS).sort the population in ascending order

Parameters

| | |
|---------------|---|
| <i>myData</i> | the data structure |
| <i>NS</i> | the number of solutions/size of a vector |
| <i>DIM</i> | the size of the dimensions/size of a vector |
| <i>b</i> | temporary vector to hold elements |

4.17 C:/Users/emyli/Desktop/Project4_471/Util.h File Reference

Data Structures

- struct [_initData1](#)

Structure for the bounds, population, fitness and function number.

Typedefs

- typedef struct `_initData1` `initData`
Structure for the bounds, population, fitness and function number.

Functions

- void `sortAscendingOrder` (`initData` *myData, int NS, int DIM, double *b)
- double * `replaceArray` (double *a, const double *b, int NS)
- double `findBest` (const double *a, int NS)
- double ** `copyDbl` (double **a, double **b, int NS, int DIM)
- double * `copySingle` (double *a, const double *b, int DIM)
- void `printDblDim` (FILE *file, double **a, int DIM, int NS)
- void `printSingle` (FILE *file, double *a, int DIM)

4.17.1 Typedef Documentation

4.17.1.1 initData

```
typedef struct _initData1 initData
```

Structure for the bounds, population, fitness and function number.

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

This project implements three meta-heuristic optimization algorithms. Particle Swarm Optimization (PSO), Firefly Algorithm (FA), and Harmony Search Algorithm (HS).

4.17.2 Function Documentation

4.17.2.1 copyDbl()

```
double** copyDbl (
    double ** a,
    double ** b,
    int NS,
    int DIM )
```

copy a arrays contents to another

Parameters

| | |
|------------|---|
| <i>a</i> | a double array |
| <i>b</i> | a double array |
| <i>DIM</i> | the number of dimensions/size of a vector |
| <i>NS</i> | the number of solutions/size of a vector |

Returns

a double array with double array b's contents placed into

4.17.2.2 copySingle()

```
double* copySingle (
    double * a,
    const double * b,
    int DIM )
```

copy a arrays contents to another

Parameters

| | |
|------------|---|
| <i>a</i> | an array |
| <i>b</i> | an array |
| <i>DIM</i> | the number of dimensions/size of a vector |

Returns

a array with array b's contents placed into

4.17.2.3 findBest()

```
double findBest (
    const double * a,
    int NS )
```

find the minimal value in a vector

Parameters

| | |
|-----------|--|
| <i>a</i> | an array |
| <i>NS</i> | the number of solutions/size of a vector |

Returns

a array with array b's elements implemented

4.17.2.4 printDbldim()

```
void printDbldim (
    FILE * file,
    double ** a,
    int DIM,
    int NS )
```

print out a matrix with dimensions printed out horizontally to a file

Parameters

| | |
|-------------|---|
| <i>file</i> | file to print to |
| <i>a</i> | an array |
| <i>DIM</i> | the number of dimensions/size of a vector |
| <i>NS</i> | the number of solutions/size of a vector |

4.17.2.5 printSingle()

```
void printSingle (
    FILE * file,
    double * a,
    int DIM )
```

print out a vector to a file

Parameters

| | |
|-------------|---|
| <i>file</i> | a file to print to |
| <i>a</i> | an array |
| <i>DIM</i> | the number of dimensions/size of a vector |

4.17.2.6 replaceArray()

```
double* replaceArray (
    double * a,
    const double * b,
    const int NS )
```

replace the elements of one array into another

Parameters

| | |
|-----------|--|
| <i>a</i> | an array |
| <i>b</i> | an array |
| <i>NS</i> | the number of solutions/size of a vector |

Returns

a array with array b's elements implemented

4.17.2.7 sortAscendingOrder()

```
void sortAscendingOrder (
    initData * myData,
    const int NS,
    const int DIM,
    double * b )
```

sort the population in ascending order

Parameters

| | |
|---------------|---|
| <i>myData</i> | the data structure |
| <i>NS</i> | the number of solutions/size of a vector |
| <i>DIM</i> | the size of the dimensions/size of a vector |
| <i>b</i> | temporary vector to hold elements |

Author

Emily Bodenhamer CWU ID 41119306 CS 471 Optimization Project 4 Date 5/10/2019

This project implements three meta-heuristic optimization algorithms. Particle Swarm Optimization (PSO), Firefly Algorithm (FA), and Harmony Search Algorithm (HS).sort the population in ascending order

Parameters

| | |
|---------------|---|
| <i>myData</i> | the data structure |
| <i>NS</i> | the number of solutions/size of a vector |
| <i>DIM</i> | the size of the dimensions/size of a vector |
| <i>b</i> | temporary vector to hold elements |

Index

- [_FA1](#), [5](#)
 - [alpha](#), [6](#)
 - [beta](#), [6](#)
 - [betaMin](#), [6](#)
 - [gamma](#), [6](#)
 - [l0](#), [6](#)
 - [index](#), [7](#)
 - [intensity](#), [7](#)
 - [newXi](#), [7](#)
 - [newXiFitness](#), [7](#)
 - [r](#), [7](#)
 - [tempFitness](#), [7](#)
 - [tempPopulation](#), [8](#)
 - [_HS1](#), [8](#)
 - [BW](#), [9](#)
 - [HMCR](#), [9](#)
 - [HSfitness](#), [9](#)
 - [HSVector](#), [9](#)
 - [PAR](#), [9](#)
 - [_PSO1](#), [11](#)
 - [c1](#), [12](#)
 - [c2](#), [12](#)
 - [gBest](#), [12](#)
 - [k](#), [12](#)
 - [pBest](#), [12](#)
 - [velocity](#), [13](#)
 - [_initData1](#), [10](#)
 - [fitness](#), [10](#)
 - [functionNumber](#), [10](#)
 - [max](#), [10](#)
 - [min](#), [11](#)
 - [population](#), [11](#)
- [ackley1](#)
 - [Functions.c](#), [30](#)
 - [Functions.h](#), [39](#)
- [ackley2](#)
 - [Functions.c](#), [30](#)
 - [Functions.h](#), [39](#)
- [alpha](#)
 - [_FA1](#), [6](#)
- [alpine](#)
 - [Functions.c](#), [31](#)
 - [Functions.h](#), [40](#)
- [ArrayMem.c](#)
 - [createDbIArray](#), [15](#)
 - [fillIn](#), [16](#)
 - [freeMem](#), [16](#)
 - [singleArray](#), [17](#)
- [ArrayMem.h](#)
 - [createDbIArray](#), [17](#)
 - [fillIn](#), [18](#)
 - [freeMem](#), [19](#)
 - [singleArray](#), [19](#)
- [beta](#)
 - [_FA1](#), [6](#)
- [betaMin](#)
 - [_FA1](#), [6](#)
- [BW](#)
 - [_HS1](#), [9](#)
- [c1](#)
 - [_PSO1](#), [12](#)
- [c2](#)
 - [_PSO1](#), [12](#)
- [C:/Users/emyli/Desktop/Project4_471/ArrayMem.c](#), [15](#)
- [C:/Users/emyli/Desktop/Project4_471/ArrayMem.h](#), [17](#)
- [C:/Users/emyli/Desktop/Project4_471/FA.c](#), [19](#)
- [C:/Users/emyli/Desktop/Project4_471/FA.h](#), [24](#)
- [C:/Users/emyli/Desktop/Project4_471/Functions.c](#), [29](#)
- [C:/Users/emyli/Desktop/Project4_471/Functions.h](#), [38](#)
- [C:/Users/emyli/Desktop/Project4_471/HS.c](#), [50](#)
- [C:/Users/emyli/Desktop/Project4_471/HS.h](#), [53](#)
- [C:/Users/emyli/Desktop/Project4_471/main.c](#), [57](#)
- [C:/Users/emyli/Desktop/Project4_471/mt19937ar.c](#), [57](#)
- [C:/Users/emyli/Desktop/Project4_471/mt19937ar.h](#), [61](#)
- [C:/Users/emyli/Desktop/Project4_471/PSO.c](#), [62](#)
- [C:/Users/emyli/Desktop/Project4_471/PSO.h](#), [65](#)
- [C:/Users/emyli/Desktop/Project4_471/SelectFunction.c](#), [69](#)
- [C:/Users/emyli/Desktop/Project4_471/SelectFunction.h](#), [70](#)
- [C:/Users/emyli/Desktop/Project4_471/Util.c](#), [72](#)
- [C:/Users/emyli/Desktop/Project4_471/Util.h](#), [75](#)
- [calcPopulation](#)
 - [PSO.c](#), [63](#)
 - [PSO.h](#), [66](#)
- [calcVelocity](#)
 - [PSO.c](#), [63](#)
 - [PSO.h](#), [66](#)
- [checkBounds](#)
 - [HS.c](#), [51](#)
 - [HS.h](#), [54](#)
- [copyDbI](#)
 - [Util.c](#), [72](#)
 - [Util.h](#), [76](#)
- [copySingle](#)
 - [Util.c](#), [73](#)
 - [Util.h](#), [77](#)

- createDblArray
 - ArrayMem.c, 15
 - ArrayMem.h, 17
- deJong
 - Functions.c, 31
 - Functions.h, 40
- eggHolder
 - Functions.c, 32
 - Functions.h, 41
- Eqn2
 - FA.c, 20
 - FA.h, 25
- Eqn3
 - FA.c, 20
 - FA.h, 25
- Eqn4
 - FA.c, 21
 - FA.h, 26
- FA
 - FA.h, 24
- FA.c
 - Eqn2, 20
 - Eqn3, 20
 - Eqn4, 21
 - readInputFA, 21
 - startFA, 23
 - updateIntensity, 23
- FA.h
 - Eqn2, 25
 - Eqn3, 25
 - Eqn4, 26
 - FA, 24
 - readInputFA, 27
 - startFA, 28
 - updateIntensity, 28
- fillIn
 - ArrayMem.c, 16
 - ArrayMem.h, 18
- findBest
 - Util.c, 73
 - Util.h, 77
- fitness
 - _initData1, 10
- freeMem
 - ArrayMem.c, 16
 - ArrayMem.h, 19
- functionNumber
 - _initData1, 10
- Functions.c
 - ackley1, 30
 - ackley2, 30
 - alpine, 31
 - deJong, 31
 - eggHolder, 32
 - griewangk, 32
 - levy, 32
 - masters, 33
 - Michalewicz, 33
 - pathological, 34
 - quartic, 34
 - rana, 35
 - rastrigin, 35
 - rosenbrock, 35
 - schwefel, 36
 - sineEnvelope, 36
 - sineStretched, 37
 - step, 37
- Functions.h
 - ackley1, 39
 - ackley2, 39
 - alpine, 40
 - deJong, 40
 - eggHolder, 41
 - griewangk, 42
 - levy, 42
 - masters, 43
 - Michalewicz, 44
 - pathological, 44
 - quartic, 45
 - rana, 45
 - rastrigin, 46
 - rosenbrock, 47
 - schwefel, 47
 - sineEnvelope, 48
 - sineStretched, 49
 - step, 49
- gamma
 - _FA1, 6
- gBest
 - _PSO1, 12
- genrand_int31
 - mt19937ar.c, 59
 - mt19937ar.h, 61
- genrand_int32
 - mt19937ar.c, 59
 - mt19937ar.h, 61
- genrand_real1
 - mt19937ar.c, 59
 - mt19937ar.h, 61
- genrand_real2
 - mt19937ar.c, 59
 - mt19937ar.h, 61
- genrand_real3
 - mt19937ar.c, 59
 - mt19937ar.h, 61
- genrand_res53
 - mt19937ar.c, 60
 - mt19937ar.h, 62
- getFun
 - SelectFunction.c, 69
 - SelectFunction.h, 70
- getFunSingle
 - SelectFunction.c, 70
 - SelectFunction.h, 71

griewangk
 Functions.c, [32](#)
 Functions.h, [42](#)

HMCR
 _HS1, [9](#)

HS
 HS.h, [53](#)

HS.c
 checkBounds, [51](#)
 readInputHS, [51](#)
 startHS, [52](#)
 updateHM, [52](#)
 updateWithPAR, [53](#)

HS.h
 checkBounds, [54](#)
 HS, [53](#)
 readInputHS, [54](#)
 startHS, [55](#)
 updateHM, [56](#)
 updateWithPAR, [56](#)

HSfitness
 _HS1, [9](#)

HSVector
 _HS1, [9](#)

l0
 _FA1, [6](#)

index
 _FA1, [7](#)

init_by_array
 mt19937ar.c, [60](#)
 mt19937ar.h, [62](#)

init_genrand
 mt19937ar.c, [60](#)
 mt19937ar.h, [62](#)

initData
 Util.h, [76](#)

intensity
 _FA1, [7](#)

k
 _PSO1, [12](#)

levy
 Functions.c, [32](#)
 Functions.h, [42](#)

LOWER_MASK
 mt19937ar.c, [58](#)

M
 mt19937ar.c, [58](#)

main
 main.c, [57](#)

main.c
 main, [57](#)

masters
 Functions.c, [33](#)
 Functions.h, [43](#)

MATRIX_A
 mt19937ar.c, [58](#)

max
 _initData1, [10](#)

michalewicz
 Functions.c, [33](#)
 Functions.h, [44](#)

min
 _initData1, [11](#)

mt
 mt19937ar.c, [60](#)

mt19937ar.c
 genrand_int31, [59](#)
 genrand_int32, [59](#)
 genrand_real1, [59](#)
 genrand_real2, [59](#)
 genrand_real3, [59](#)
 genrand_res53, [60](#)
 init_by_array, [60](#)
 init_genrand, [60](#)
 LOWER_MASK, [58](#)
 M, [58](#)
 MATRIX_A, [58](#)
 mt, [60](#)
 mti, [60](#)
 N, [58](#)
 UPPER_MASK, [59](#)

mt19937ar.h
 genrand_int31, [61](#)
 genrand_int32, [61](#)
 genrand_real1, [61](#)
 genrand_real2, [61](#)
 genrand_real3, [61](#)
 genrand_res53, [62](#)
 init_by_array, [62](#)
 init_genrand, [62](#)

mti
 mt19937ar.c, [60](#)

N
 mt19937ar.c, [58](#)

newXi
 _FA1, [7](#)

newXiFitness
 _FA1, [7](#)

PAR
 _HS1, [9](#)

pathological
 Functions.c, [34](#)
 Functions.h, [44](#)

pBest
 _PSO1, [12](#)

population
 _initData1, [11](#)

printDblDim
 Util.c, [74](#)
 Util.h, [78](#)

printSingle

- Util.c, [74](#)
- Util.h, [78](#)
- PSO
 - PSO.h, [65](#)
- PSO.c
 - calcPopulation, [63](#)
 - calcVelocity, [63](#)
 - readInput, [63](#)
 - startPSO, [64](#)
- PSO.h
 - calcPopulation, [66](#)
 - calcVelocity, [66](#)
 - PSO, [65](#)
 - readInput, [67](#)
 - startPSO, [68](#)
- quartic
 - Functions.c, [34](#)
 - Functions.h, [45](#)
- r
 - _FA1, [7](#)
- rana
 - Functions.c, [35](#)
 - Functions.h, [45](#)
- rastrigin
 - Functions.c, [35](#)
 - Functions.h, [46](#)
- readInput
 - PSO.c, [63](#)
 - PSO.h, [67](#)
- readInputFA
 - FA.c, [21](#)
 - FA.h, [27](#)
- readInputHS
 - HS.c, [51](#)
 - HS.h, [54](#)
- replaceArray
 - Util.c, [74](#)
 - Util.h, [78](#)
- rosenbrock
 - Functions.c, [35](#)
 - Functions.h, [47](#)
- schwefel
 - Functions.c, [36](#)
 - Functions.h, [47](#)
- SelectFunction.c
 - getFun, [69](#)
 - getFunSingle, [70](#)
- SelectFunction.h
 - getFun, [70](#)
 - getFunSingle, [71](#)
- sineEnvelope
 - Functions.c, [36](#)
 - Functions.h, [48](#)
- sineStretched
 - Functions.c, [37](#)
 - Functions.h, [49](#)
- singleArray
 - ArrayMem.c, [17](#)
 - ArrayMem.h, [19](#)
- sortAscendingOrder
 - Util.c, [75](#)
 - Util.h, [79](#)
- startFA
 - FA.c, [23](#)
 - FA.h, [28](#)
- startHS
 - HS.c, [52](#)
 - HS.h, [55](#)
- startPSO
 - PSO.c, [64](#)
 - PSO.h, [68](#)
- step
 - Functions.c, [37](#)
 - Functions.h, [49](#)
- tempFitness
 - _FA1, [7](#)
- tempPopulation
 - _FA1, [8](#)
- updateHM
 - HS.c, [52](#)
 - HS.h, [56](#)
- updateIntensity
 - FA.c, [23](#)
 - FA.h, [28](#)
- updateWithPAR
 - HS.c, [53](#)
 - HS.h, [56](#)
- UPPER_MASK
 - mt19937ar.c, [59](#)
- Util.c
 - copyDbl, [72](#)
 - copySingle, [73](#)
 - findBest, [73](#)
 - printDblDim, [74](#)
 - printSingle, [74](#)
 - replaceArray, [74](#)
 - sortAscendingOrder, [75](#)
- Util.h
 - copyDbl, [76](#)
 - copySingle, [77](#)
 - findBest, [77](#)
 - initData, [76](#)
 - printDblDim, [78](#)
 - printSingle, [78](#)
 - replaceArray, [78](#)
 - sortAscendingOrder, [79](#)
- velocity
 - _PSO1, [13](#)