CS 471 Project 4: Particle Swarm, Firefly, Harmonic Search Algorithms

Central Washington University

Taylor Apple

Generated by Doxygen 1.8.15

1 Class Index 1

1.1 Class List . . . 1

2 File Index 3

2.1 File List . . . 3

3 Class Documentation 5

3.1 _EquationInfo Struct Reference . . . 5

3.1.1 Detailed Description . . . 5

3.2 _FireflySwarm Struct Reference . . . 6

3.2.1 Detailed Description . . . 6

3.3 _HarmonicPop Struct Reference . . . 6

3.3.1 Detailed Description . . . 6

3.4 _Info Struct Reference . . . 7

3.4.1 Detailed Description . . . 7

3.5 _Particle Struct Reference . . . 7

3.5.1 Detailed Description . . . 7

4 File Documentation 9

4.1 General/EquationHandlers.h File Reference . . . 9

4.1.1 Detailed Description . . . 10

4.1.2 Function Documentation . . . 10

4.1.2.1 ackleyOneHandler() . . . 10

4.1.2.2 ackleyTwoHandler() . . . 11

4.1.2.3 alpineHandler() . . . 11

4.1.2.4 deJongHandler() . . . 12

4.1.2.5 eggHolderHandler() . . . 13

4.1.2.6 griewangkHandler() . . . 13

4.1.2.7 levyHandler() . . . 14

4.1.2.8 mastersCosineWaveHandler() . . . 15

4.1.2.9 michalewiczHandler() . . . 15

4.1.2.10 pathologicalHandler() . . . 16

4.1.2.11 quarticHandler() . . . 17

4.1.2.12 ranaHandler() . . . 17

4.1.2.13 rastgrinHandler() . . . 18

4.1.2.14 rosenbrockHandler() . . . 19

4.1.2.15 runEquationsAsThreads() . . . 19

4.1.2.16 schwefelHandler() . . . 21

4.1.2.17 sineEnvSineWaveHandler() . . . 22

4.1.2.18 stepHandler() . . . 23

4.1.2.19 stretchVSineWaveHandler() . . . 23

4.2 General/Equations.c File Reference . . . 24

4.2.1 Detailed Description . . . 25

4.2.2 Function Documentation . . . 25
4.2.2.1 ackleyOneHost() . . . 25
4.2.2.2 ackleyTwoHost() . . . 25
4.2.2.3 alpineHost() . . . 26
4.2.2.4 deJongHost() . . . 26
4.2.2.5 eggHolderHost() . . . 26
4.2.2.6 griewangkHost() . . . 27
4.2.2.7 levyHost() . . . 27
4.2.2.8 mastersCosineWaveHost() . . . 27
4.2.2.9 michalewiczHost() . . . 28
4.2.2.10 pathologicalHost() . . . 28
4.2.2.11 quarticHost() . . . 28
4.2.2.12 ranaHost() . . . 29
4.2.2.13 rastgrinHost() . . . 29
4.2.2.14 rosenbrockHost() . . . 29
4.2.2.15 schwefelHost() . . . 30
4.2.2.16 sineEnvSineWaveHost() . . . 30
4.2.2.17 stepHost() . . . 30
4.2.2.18 stretchVSineWaveHost() . . . 31
4.3 General/Equations.h File Reference . . . 31
4.3.1 Detailed Description . . . 31
4.3.2 Function Documentation . . . 32
4.3.2.1 ackleyOneHost() . . . 32
4.3.2.2 ackleyTwoHost() . . . 32
4.3.2.3 alpineHost() . . . 32
4.3.2.4 deJongHost() . . . 33
4.3.2.5 eggHolderHost() . . . 33
4.3.2.6 griewangkHost() . . . 33
4.3.2.7 levyHost() . . . 34
4.3.2.8 mastersCosineWaveHost() . . . 34
4.3.2.9 michalewiczHost() . . . 34
4.3.2.10 pathologicalHost() . . . 35
4.3.2.11 quarticHost() . . . 35
4.3.2.12 ranaHost() . . . 35
4.3.2.13 rastgrinHost() . . . 36
4.3.2.14 rosenbrockHost() . . . 36
4.3.2.15 schwefelHost() . . . 36
4.3.2.16 sineEnvSineWaveHost() . . . 37
4.3.2.17 stepHost() . . . 37
4.3.2.18 stretchVSineWaveHost() . . . 37
4.4 General/FA.h File Reference . . . 37
4.4.1 Detailed Description . . . 38

4.4.2 Function Documentation . . . 38
4.4.2.1 addVector() . . . 38
4.4.2.2 calcAttractedVector() . . . 39
4.4.2.3 calcAttractiveness() . . . 39
4.4.2.4 calcDistanceSquared() . . . 40
4.4.2.5 fireflyAlg() . . . 40
4.4.2.6 fireflyLoop() . . . 41
4.4.2.7 lightIntensity() . . . 42
4.4.2.8 moveFirefliesLoop() . . . 42
4.4.2.9 newBest() . . . 43
4.5 General/Harmonic.c File Reference . . . 43
4.5.1 Detailed Description . . . 43
4.5.2 Function Documentation . . . 44
4.5.2.1 harmonicIteration() . . . 44
4.5.2.2 harmonicTest() . . . 44
4.5.2.3 newVector() . . . 45
4.5.2.4 pitchAdjustment() . . . 45
4.5.2.5 updateBest() . . . 46
4.6 General/Harmonic.h File Reference . . . 46
4.6.1 Detailed Description . . . 46
4.6.2 Function Documentation . . . 47
4.6.2.1 harmonicIteration() . . . 47
4.6.2.2 harmonicTest() . . . 47
4.6.2.3 newVector() . . . 48
4.6.2.4 pitchAdjustment() . . . 48
4.6.2.5 updateBest() . . . 49
4.7 General/HostCalls.h File Reference . . . 49
4.7.1 Detailed Description . . . 49
4.7.2 Variable Documentation . . . 49
4.7.2.1 equationHostCalls . . . 49
4.8 General/Init.c File Reference . . . 50
4.8.1 Detailed Description . . . 51
4.8.2 Enumeration Type Documentation . . . 51
4.8.2.1 inputFlag . . . 51
4.8.3 Function Documentation . . . 51
4.8.3.1 checkTestType() . . . 52
4.8.3.2 init() . . . 52
4.8.3.3 processAlpha() . . . 55
4.8.3.4 processBandwidth() . . . 56
4.8.3.5 processBeta() . . . 56
4.8.3.6 processC1() . . . 57
4.8.3.7 processC2() . . . 57

4.8.3.8 processDampener() . . . 58
4.8.3.9 processDimensions() . . . 58
4.8.3.10 processDimList() . . . 58
4.8.3.11 processEquations() . . . 59
4.8.3.12 processExperiments() . . . 60
4.8.3.13 processGamma() . . . 60
4.8.3.14 processHMCR() . . . 60
4.8.3.15 processIterations() . . . 61
4.8.3.16 processPAR() . . . 61
4.8.3.17 processRanges() . . . 62
4.8.3.18 processVectors() . . . 63
4.9 General/Init.h File Reference . . . 63
4.9.1 Detailed Description . . . 64
4.9.2 Function Documentation . . . 64
4.9.2.1 checkTestType() . . . 64
4.9.2.2 init() . . . 65
4.9.2.3 processAlpha() . . . 68
4.9.2.4 processBandwidth() . . . 68
4.9.2.5 processBeta() . . . 69
4.9.2.6 processC1() . . . 69
4.9.2.7 processC2() . . . 70
4.9.2.8 processDampener() . . . 70
4.9.2.9 processDimensions() . . . 71
4.9.2.10 processDimList() . . . 71
4.9.2.11 processEquations() . . . 72
4.9.2.12 processGamma() . . . 72
4.9.2.13 processHMCR() . . . 73
4.9.2.14 processIterations() . . . 73
4.9.2.15 processPAR() . . . 74
4.9.2.16 processRanges() . . . 74
4.9.2.17 processVectors() . . . 75
4.10 General/MersenneMatrix.c File Reference . . . 76
4.10.1 Detailed Description . . . 76
4.10.2 Function Documentation . . . 76
4.10.2.1 createMatrix() . . . 76
4.10.2.2 createVelocities() . . . 77
4.10.2.3 genDblInRange() . . . 78
4.10.2.4 genNonNegInt() . . . 78
4.11 General/MersenneMatrix.h File Reference . . . 78
4.11.1 Detailed Description . . . 79
4.11.2 Function Documentation . . . 79
4.11.2.1 createMatrix() . . . 79

4.11.2.2 createVelocities() . . . 80
4.11.2.3 genDblInRange() . . . 80
4.11.2.4 genNonNegInt() . . . 81
4.12 General/PSO.c File Reference . . . 81
4.12.1 Detailed Description . . . 82
4.12.2 Function Documentation . . . 82
4.12.2.1 calcGBestModifier() . . . 82
4.12.2.2 calcNewVector() . . . 82
4.12.2.3 calcNewVelocity() . . . 83
4.12.2.4 calcPBestModifier() . . . 83
4.12.2.5 particleLoop() . . . 84
4.12.2.6 particleSwarmAlg() . . . 84
4.13 General/PSO.h File Reference . . . 85
4.13.1 Detailed Description . . . 85
4.13.2 Function Documentation . . . 86
4.13.2.1 calcGBestModifier() . . . 86
4.13.2.2 calcNewVector() . . . 86
4.13.2.3 calcNewVelocity() . . . 86
4.13.2.4 calcPBestModifier() . . . 87
4.13.2.5 particleLoop() . . . 87
4.13.2.6 particleSwarmAlg() . . . 88
4.14 General/Utilities.c File Reference . . . 89
4.14.1 Detailed Description . . . 90
4.14.2 Function Documentation . . . 90
4.14.2.1 allocateEmptyMatrix() . . . 90
4.14.2.2 allocateHPop() . . . 91
4.14.2.3 copyArray() . . . 91
4.14.2.4 copyMatrix() . . . 91
4.14.2.5 createParticles() . . . 92
4.14.2.6 evalNewWorst() . . . 92
4.14.2.7 evaluateFitness() . . . 93
4.14.2.8 evaluatePop() . . . 93
4.14.2.9 freeEquationInfo() . . . 94
4.14.2.10 freeFireflySwarm() . . . 95
4.14.2.11 freeHPop() . . . 95
4.14.2.12 freeInfo() . . . 95
4.14.2.13 freeMatrix() . . . 96
4.14.2.14 freeParticles() . . . 96
4.14.2.15 lock() . . . 96
4.14.2.16 printDArray() . . . 97
4.14.2.17 printIArray() . . . 97
4.14.2.18 printMatrix() . . . 97

4.14.2.19 unlock() . . . 99
4.14.2.20 writePopulationLogToFile() . . . 99
4.14.2.21 writeResultToFile() . . . 100
4.15 General/Utilities.h File Reference . . . 101
4.15.1 Detailed Description . . . 103
4.15.2 Macro Definition Documentation . . . 103
4.15.2.1 DEFAULT_INIT_FILE . . . 103
4.15.2.2 FILE_ARGUMENT . . . 103
4.15.2.3 LINE_LENGTH . . . 103
4.15.2.4 MAX_FILE_NAME_LEN . . . 103
4.15.2.5 MAX_NUM_EQUATIONS . . . 103
4.15.2.6 MINIMUM_ITERATIONS . . . 103
4.15.2.7 MS_PER_SEC . . . 104
4.15.2.8 NS_PER_MS . . . 104
4.15.2.9 RANGE_MAX_POS . . . 104
4.15.2.10 RANGE_MIN_POS . . . 104
4.15.2.11 RANGE_SIZE . . . 104
4.15.3 Typedef Documentation . . . 104
4.15.3.1 EquationInfo . . . 104
4.15.3.2 FireflySwarm . . . 104
4.15.3.3 HPop . . . 105
4.15.3.4 Info . . . 105
4.15.3.5 Particle . . . 105
4.15.4 Enumeration Type Documentation . . . 105
4.15.4.1 EquationPosition . . . 105
4.15.4.2 TestType . . . 105
4.15.5 Function Documentation . . . 105
4.15.5.1 allocateEmptyMatrix() . . . 105
4.15.5.2 allocateHPop() . . . 106
4.15.5.3 copyArray() . . . 106
4.15.5.4 copyMatrix() . . . 106
4.15.5.5 createParticles() . . . 107
4.15.5.6 evalNewWorst() . . . 107
4.15.5.7 evaluateFitness() . . . 108
4.15.5.8 evaluatePop() . . . 108
4.15.5.9 freeEquationInfo() . . . 109
4.15.5.10 freeFireflySwarm() . . . 110
4.15.5.11 freeHPop() . . . 110
4.15.5.12 freeInfo() . . . 110
4.15.5.13 freeMatrix() . . . 112
4.15.5.14 freeParticles() . . . 112
4.15.5.15 genRandIntP() . . . 113

4.15.5.16 genRandRealP() . . . 113
4.15.5.17 printDArray() . . . 113
4.15.5.18 printIArray() . . . 114
4.15.5.19 printMatrix() . . . 114
4.15.5.20 writePopulationLogToFile() . . . 115
4.15.5.21 writeResultToFile() . . . 116
4.15.6 Variable Documentation . . . 116
4.15.6.1 mutex . . . 116
4.16 Pthread/EquationHandlers.c File Reference . . . 117
4.16.1 Detailed Description . . . 118
4.16.2 Function Documentation . . . 118
4.16.2.1 ackleyOneHandler() . . . 118
4.16.2.2 ackleyTwoHandler() . . . 118
4.16.2.3 alpineHandler() . . . 119
4.16.2.4 deJongHandler() . . . 119
4.16.2.5 eggHolderHandler() . . . 120
4.16.2.6 griewangkHandler() . . . 120
4.16.2.7 levyHandler() . . . 121
4.16.2.8 mastersCosineWaveHandler() . . . 121
4.16.2.9 michalewiczHandler() . . . 122
4.16.2.10 pathologicalHandler() . . . 122
4.16.2.11 quarticHandler() . . . 123
4.16.2.12 ranaHandler() . . . 123
4.16.2.13 rastgrinHandler() . . . 124
4.16.2.14 rosenbrockHandler() . . . 124
4.16.2.15 runEquationsAsThreads() . . . 125
4.16.2.16 schwefelHandler() . . . 126
4.16.2.17 sineEnvSineWaveHandler() . . . 127
4.16.2.18 stepHandler() . . . 127
4.16.2.19 stretchVSineWaveHandler() . . . 128
4.16.3 Variable Documentation . . . 128
4.16.3.1 testTypeCalls . . . 129
4.17 Pthread/main.c File Reference . . . 129
4.17.1 Detailed Description . . . 129
4.17.2 Function Documentation . . . 129
4.17.2.1 main() . . . 129
4.17.3 Variable Documentation . . . 130
4.17.3.1 equationHandlers . . . 131
4.18 Pthread/UtilP.c File Reference . . . 131
4.18.1 Detailed Description . . . 131
4.18.2 Function Documentation . . . 131
4.18.2.1 genRandIntP() . . . 132

4.18.2.2 genRandRealP() . . . 132
4.19 Win32/EquationHandlers32.c File Reference . . . 132
4.19.1 Detailed Description . . . 134
4.19.2 Function Documentation . . . 134
4.19.2.1 ackleyOneHandler() . . . 134
4.19.2.2 ackleyTwoHandler() . . . 134
4.19.2.3 alpineHandler() . . . 135
4.19.2.4 deJongHandler() . . . 135
4.19.2.5 eggHolderHandler() . . . 136
4.19.2.6 griewangkHandler() . . . 136
4.19.2.7 levyHandler() . . . 137
4.19.2.8 mastersCosineWaveHandler() . . . 137
4.19.2.9 michalewiczHandler() . . . 138
4.19.2.10 pathologicalHandler() . . . 138
4.19.2.11 quarticHandler() . . . 139
4.19.2.12 ranaHandler() . . . 139
4.19.2.13 rastgrinHandler() . . . 140
4.19.2.14 rosenbrockHandler() . . . 140
4.19.2.15 runEquationsAsThreads() . . . 141
4.19.2.16 schwefelHandler() . . . 142
4.19.2.17 sineEnvSineWaveHandler() . . . 143
4.19.2.18 stepHandler() . . . 143
4.19.2.19 stretchVSineWaveHandler() . . . 144
4.19.3 Variable Documentation . . . 144
4.19.3.1 testTypeCalls . . . 144
4.20 Win32/main32.c File Reference . . . 145
4.20.1 Detailed Description . . . 145
4.20.2 Function Documentation . . . 145
4.20.2.1 main() . . . 145
4.20.3 Variable Documentation . . . 146
4.20.3.1 equationHandlers . . . 146
4.21 Win32/Util32.c File Reference . . . 147
4.21.1 Detailed Description . . . 147
4.21.2 Function Documentation . . . 147
4.21.2.1 genRandInt32() . . . 147
4.21.2.2 genRandReal32() . . . 147

Index **149**

Chapter 1

Class Index

1.1 Class List

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

_EquationInfo 5
_FireflySwarm 6
_HarmonicPop 6
_Info 7
_Particle 7

Chapter 2

File Index

2.1 File List

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

General/EquationHandlers.h
This is where the methods for the threaded calls in PThread/main.c or Win32/main32.c are defined . . . 9
General/Equations.c
This is where the methods for the calls in General/TestTypes.c and defined in General/Equations.h are implemented . . . 24
General/Equations.h
This is where the methods for the calls in General/TestTypes.c are defined . . . 31
General/FA.h
This is where all methods pertaining to the Firefly Algorithm implementation are defined . . . 37
General/Harmonic.c
This is where all methods defined in General/Harmonic.h are implemented . . . 43
General/Harmonic.h
This is where all methods pertaining to the Harmonic Search Algorithm implementation are defined . . . 46
General/HostCalls.h
This is where all the references to the objective function call methods are stored to avoid duplication . . . 49
General/Init.c
Handles processing the input file for use throughout this testing . . . 50
General/Init.h
This is where the methods for the building the program info from the init file are defined. Referenced in PThread/main.c or Win32/main32.c . . . 63
General/**m19937ar-cok.h** . . . **??**
General/MersenneMatrix.c
Implementation of the create matrix method defined in General/MersenneMatrix.h . . . 76
General/MersenneMatrix.h
Defines the method which creates the randomized matrix for use in all tests, for all equations in the program and generating a random double in the range . . . 78
General/PSO.c
This is where all methods defined in General/PSO.h are implemented . . . 81
General/PSO.h
This is where all methods pertaining to the Particle Swarm Optimization Algorithm implementation are defined . . . 85

General/Utilities.c
This is where all general purpose methods are implemented. All methods are defined in General/Utilities.h . . . 89
General/Utilities.h
This is where all general purpose methods, constants, enums, and structs are declared . . . 101
Pthread/EquationHandlers.c
Handles the individual equation threads created in PThread/main.c and provides implementation for functions defined in General/EquationHandlers.h . . . 117
Pthread/main.c
The entry point for Unix/Linux machines when executing this program . . . 129
Pthread/UtilP.c
Implementations for the methods declared in General/Utilities.h which produce random numbers using a mutex in POSIX format . . . 131
Win32/EquationHandlers32.c
Handles the individual equation threads created in Win32/main32.c and provides implementation for functions defined in General/EquationHandlers.h . . . 132
Win32/main32.c
The entry point for Win32 machines when executing this program . . . 145
Win32/Util32.c
Implementations for the methods declared in General/Utilities.h which produce random numbers using a mutex in WIN332 format . . . 147

Chapter 3

Class Documentation

3.1 _EquationInfo Struct Reference

`#include <Utilities.h>`

Public Attributes

- char * **equationName**
- int **currExperiment**
- int **printExperiment**
- int **equationNum**
- int **numVectors**
- double * **range**
- int **dimToTest**
- int **iterations**
- double **bandwidth**
- double **PAR**
- double **HMCR**
- double **beta**
- double **gamma**
- double **alpha**
- double **c1**
- double **c2**
- double **k**

3.1.1 Detailed Description

stores all of the information required by a single equation for a single set of dimensions to be processed in the selected test type. used in Win32/EquationsHandlers32.c and PThread/EquationHandlers.c

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

- General/Utilities.h

3.2 _FireflySwarm Struct Reference

`#include <Utilities.h>`

Public Attributes

- double ** **population**
- double * **fitness**
- int **dimensions**
- double **bestFit**
- int **bestPos**
- double **worstFit**
- int **worstPos**

3.2.1 Detailed Description

Stores all information related to a population necessary for the Firefly Algorithm meta heuristics

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

- General/Utilities.h

3.3 _HarmonicPop Struct Reference

`#include <Utilities.h>`

Public Attributes

- double **bestFit**
- int **bestPos**
- double **worstFit**
- int **worstPos**
- double ** **population**
- double * **fitness**
- double * **newHarmonic**

3.3.1 Detailed Description

Stores all information related to a population necessary for the Harmonic Search meta heuristics

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

- General/Utilities.h

3.4 _Info Struct Reference

`#include <Utilities.h>`

Public Attributes

- int **numExperiments**
- int **numVectors**
- int **numDimensions**
- int **numEquations**
- double ** **ranges**
- int * **dimsToTest**
- int **iterations**
- int **testSelection**
- double **bandwidth**
- double **PAR**
- double **HMCR**
- double **beta**
- double **gamma**
- double **alpha**
- double **c1**
- double **c2**
- double **k**

3.4.1 Detailed Description

stores all of the necessary information for the program to run for all functions and all test types

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

- General/Utilities.h

3.5 _Particle Struct Reference

`#include <Utilities.h>`

Public Attributes

- double * **pBestFit**
- double **gBestFit**
- double **gWorstFit**
- double ** **velocities**
- double ** **personalBest**
- double ** **population**
- double * **fitness**
- int **worstPos**
- int **bestPos**

3.5.1 Detailed Description

Stores all information related to a population necessary for the Particle Swarm meta heuristics

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

- General/Utilities.h

Chapter 4

File Documentation

4.1 General/EquationHandlers.h File Reference

This is where the methods for the threaded calls in PThread/main.c or Win32/main32.c are defined.

```
#include "Utilities.h"
```

Functions

- int runEquationsAsThreads (int, char *, Info *)

  Creates EquationInfo structs from the Info struct passed in and then creates threads for each of the dimensional tests.
- void * schwefelHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * deJongHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * rosenbrockHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * rastgrinHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * griewangkHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * sineEnvSineWaveHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * stretchVSineWaveHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * ackleyOneHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * ackleyTwoHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * eggHolderHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * ranaHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * pathologicalHandler (void *info)

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

- void * michalewiczHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * mastersCosineWaveHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * quarticHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * levyHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * stepHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * alpineHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

4.1.1 Detailed Description

This is where the methods for the threaded calls in PThread/main.c or Win32/main32.c are defined.

Here we define and describe the methods which are referenced in PThread/main.c or Win32/main32.c and implemented in either PThread/EquationHandlers.c or Win32/EquationHandlers32.c depending on whether you are using a Win32 or Unix/Linux machine. Based on the equation we pass certain information to the runEquationsAsThreads method which then runs each equation threaded by the dimensions required for each test. E.g. if there are 3 dimensions to test 3 threads will be created to run the functions, one for each dimension.

4.1.2 Function Documentation

4.1.2.1 ackleyOneHandler()

```
void * ackleyOneHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.1.2.2 ackleyTwoHandler()

```
void * ackleyTwoHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.1.2.3 alpineHandler()

```
void * alpineHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.1.2.4 deJongHandler()

```
void * deJongHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.1.2.5 eggHolderHandler()

```
void * eggHolderHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.1.2.6 griewangkHandler()

```
void * griewangkHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.1.2.7 levyHandler()

```
void * levyHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.1.2.8 mastersCosineWaveHandler()

```
void * mastersCosineWaveHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.1.2.9 michalewiczHandler()

```
void * michalewiczHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.1.2.10 pathologicalHandler()

```
void * pathologicalHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.1.2.11 quarticHandler()

```
void * quarticHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.1.2.12 ranaHandler()

```
void * ranaHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.1.2.13 rastgrinHandler()

```
void * rastgrinHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.1.2.14 rosenbrockHandler()

```
void * rosenbrockHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.1.2.15 runEquationsAsThreads()

```
int runEquationsAsThreads (
            int equationPos,
            char * eqName,
            Info * data )
```

Creates EquationInfo structs from the Info struct passed in and then creates threads for each of the dimensional tests.

This method takes in some basic information from the handler methods, equationPos and eqName, and then utilizes the Info struct passed to each equation's thread to create a specific EquationInfo struct for each dimensional test. Then it passes these structs to individual threads for each dimensional test.

Parameters

equationPos	- indicates the position within the various arrays of the Info struct for this equation's info
eqName	- Name of the equation being run used for file naming
data	- the Info struct which will be referenced to create the appropriate EquationInfo structs

Returns

0 on success, -1 on failures

< Store the number of different dimensions to be tested locally

< Create an array of EquationInfo structs defined in General/Utilities.h

Iterate through the number of different dimensions to be tested, create an EquationInfo struct to be passed to the equation function for each dimension, and fill it with relevant info for the dimension to be tested. Finally, store it to the array of structs.

< Create a temporary struct to be added to the array defined above

< Set the equationName to the value passed form the handler

< Set the equationNum to one greater than the position in a 0 based array

< Set the number of vectors for the test

< Set the specific dimensions for this test

< Set the number of iterations for this test

< Set the bandwidth for the harmonic test

< Set the pitch adjustment rate for the harmonic test

< Set the Harmony Memory Considering Rate for the harmonic test

< Set the beta value for attractiveness in the firefly algorithm test

< Set the absorption rate value for the firefly algorithm test

< Set the alpha value for the firefly algorithm test

< Set the personalBest term modifier c1 for PSO test

< Set the globalBest term modifier c2 for PSO test

< Set the dampening factor for PSO test

< Allocate space for the range of this equation

< Allocate space for the range of this equation

Loop through the range for this equation and copy the values over.

< Copy the value from the data Info struct to the range array

< Store the temp struct to the array

Create an array of pthread_t which is how threads are referenced in POSIX threads. Iterate for the number of experiments to be completed and within each iteration Iterate from 0 to (numDim - 1) and start a thread for each equation dimension from the constant defined above. Each thread will be passed a reference to the EquationInfo struct for each dimension. If it fails, print the error message, free the struct, and return failure.

< Set the currentExperiment number for this test

Wait for all the threads to finish, and if there was an error, print the last error and return failure.

This equation has finished for all dimensions and experiments, free the threads array, free the array of EquationInfo structs and return success.

< Store the number of different dimensions to be tested locally

< Create an array of EquationInfo structs defined in General/Utilities.h

Iterate through the number of different dimensions to be tested, create an EquationInfo struct to be passed to the equation function for each dimension, and fill it with relevant info for the dimension to be tested. Finally, store it to the array of structs.

< Create a temporary struct to be added to the array defined above

< Set the equationName to the value passed form the handler

< Set the equationNum to done greater than the position in a 0 based array

< Set the number of vectors for the test

< Set the specific dimensions for this test

< Set the number of iterations for the test

< Set the bandwidth for the harmonic test

< Set the pitch adjustment rate for the harmonic test

< Set the Harmony Memory Considering Rate for the harmonic test

< Set the beta value for attractiveness in the firefly algorithm test

< Set the absorption rate value for the firefly algorithm test

< Set the alpha value for the firefly algorithm test

< Set the personalBest term modifier c1 for PSO test

< Set the globalBest term modifier c2 for PSO test

< Set the dampening factor for PSO test

< Allocate space for the range of this equation

Loop through the range for this equation and copy the values over.

< Copy the value from the data Info struct to the range array

< Store the temp struct to the array

Create an array of Handles which is how threads are referenced in Win32 threads. Iterate for the number of experiments to be completed and within each iteration Iterate from 0 to (numDim - 1) and start a thread for each equation dimension from the constant defined above. Each thread will be passed a reference to the EquationInfo struct for each dimension. If it fails, print the error message, free the struct, and return failure.

< Set the currentExperiment number for this test

Wait for all the threads to finish, and if there was an error, print the last error and return failure.

This equation has finished for all dimensions and experiments, free the threads array, free the array of EquationInfo structs and return success.

4.1.2.16 schwefelHandler()

```
void * schwefelHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.1.2.17 sineEnvSineWaveHandler()

```
void * sineEnvSineWaveHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.1.2.18 stepHandler()

```
void * stepHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.1.2.19 stretchVSineWaveHandler()

```
void * stretchVSineWaveHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.2 General/Equations.c File Reference

This is where the methods for the calls in General/TestTypes.c and defined in General/Equations.h are implemented.

```
#include "Utilities.h"
#include "Equations.h"
#include "MersenneMatrix.h"
```

Functions

- double schwefelHost (const double *vector, int numDim)
- double deJongHost (const double *vector, int numDim)
- double rosenbrockHost (const double *vector, int numDim)
- double rastgrinHost (const double *vector, int numDim)
- double griewangkHost (const double *vector, int numDim)
- double sineEnvSineWaveHost (const double *vector, int numDim)
- double stretchVSineWaveHost (const double *vector, int numDim)
- double ackleyOneHost (const double *vector, int numDim)
- double ackleyTwoHost (const double *vector, int numDim)
- double eggHolderHost (const double *vector, int numDim)
- double ranaHost (const double *vector, int numDim)
- double pathologicalHost (const double *vector, int numDim)
- double michalewiczHost (const double *vector, int numDim)
- double mastersCosineWaveHost (const double *vector, int numDim)
- double quarticHost (const double *vector, int numDim)
- double levyHost (const double *vector, int numDim)
- double stepHost (const double *vector, int numDim)
- double alpineHost (const double *vector, int numDim)

4.2.1 Detailed Description

This is where the methods for the calls in General/TestTypes.c and defined in General/Equations.h are implemented.

Here we implement the functions to perform calculations on a single vector and produce a result for that vector based on a set of predetermined equations.

4.2.2 Function Documentation

4.2.2.1 ackleyOneHost()

```
double ackleyOneHost (
            const double * vector,
            int numDim )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Declare the variable to represent the next element

< Initialize the ackley constant to 1/e^0.2

Loop through the vector for testing, calculating the sum of the following for each element ackleyConst * sqrt((currElement^2 + nextElement^2)) + (3 * cos(2 * currElement)) + sin(2 * nextElement)

return the final resulting value

4.2.2.2 ackleyTwoHost()

```
double ackleyTwoHost (
            const double * vector,
            int numDim )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Declare the variable to represent the next element

Loop through the vector for testing, calculating the sum of the following for each element 20 + e - (20 / e^(0.2 * sqrt((currElement^2 + nextElement^2) / 2))) - e^(0.5 * (cos(2 * pi * currElement) + cos(2 * pi * nextElement)))

return the final resulting value

4.2.2.3 alpineHost()

```
double alpineHost (
            const double * vector,
            int numDim )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

Loop through the vector for testing, calculating the sum of the following for each element fabs((element * sin(element)) + (0.1 * element))

return the final resulting value

4.2.2.4 deJongHost()

```
double deJongHost (
            const double * vector,
            int numDim )
```

< Initialize the variable to store the resulting calculation

Loop through the vector for testing, for each element square it, and sum all the squares.

return the final resulting value

4.2.2.5 eggHolderHost()

```
double eggHolderHost (
            const double * vector,
            int numDim )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Declare the variable to represent the next element

Loop through the vector for testing, calculating the sum of the following for each element (-1 * currElement * sin(sqrt(absoluteVal(currElement - nextElement -47)))) - ((nextElement + 47) * sin(sqrt(absoluteVal(nextElement + 47 + (currElement / 2)))))

return the final resulting value

4.2.2.6 griewangkHost()

```
double griewangkHost (
            const double * vector,
            int numDim )
```

< Initializing the variable to store the summation result

< Initializing the variable to store the product result

< Initialize the variable to store the final result

< Declare the variable to represent the current element of the summation

Loop through the vector for testing, for the summation result, add all terms of the current element squared divided by 4000, and for the product result multiply all terms of cos(current element divided by the square root of the current index). Once the summation and product are complete take 1 + the summation result - the product result.

return the final resulting value

4.2.2.7 levyHost()

```
double levyHost (
            const double * vector,
            int numDim )
```

< Initialize the variable to store the resulting calculation

< Initialize the variable for storing W0 in the vector

< Initialize the variable for storing Wn in the vector

< Declares the variable for declaring Wi for the current dimension in the vector

Loop through the vector for testing, calculating the sum of the following for each element first calculating Wi = 1.0 + ((vector[j] - 1.0)/4.0) (pow(wi - 1.0, 2.0) ∗ (1.0 + 10.0 ∗ pow(sin((M_PI ∗ wi) + 1.0), 2.0))) + (pow(Wn - 1.0, 2.0) ∗ (1.0 + pow(sin(2.0 ∗ M_PI ∗ Wn), 2.0))) once this total is acquired adding pow(sin(M_PI ∗ W0), 2.0) for the final result

return the final resulting value

4.2.2.8 mastersCosineWaveHost()

```
double mastersCosineWaveHost (
            const double * vector,
            int numDim )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Declare the variable to represent the next element

Loop through the vector for testing, calculating the sum of the following for each element exp((-1.0 / 8.0) ∗(pow(element, 2.0) + pow(nextElem, 2.0) + (0.5 ∗ nextElem ∗ element))) cos(pow(pow(element, 2.0) + pow(next↩ Elem, 2.0) + (0.5 ∗ nextElem ∗ element), 0.25))

return the final resulting value

4.2.2.9 michalewiczHost()

```
double michalewiczHost (
            const double * vector,
            int numDim )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

Loop through the vector for testing, calculating the sum of the following for each element sin(element) * pow(sin(((j+1) * pow(element, 2.0)) / M_PI), 20.0)

return the final resulting value

4.2.2.10 pathologicalHost()

```
double pathologicalHost (
            const double * vector,
            int numDim )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Declare the variable to represent the next element

Loop through the vector for testing, calculating the sum of the following for each element 0.5 + ((sin(pow(sqrt((100.0 * pow(element, 2.0)) + pow(nextElem, 2.0)),2.0)) - 0.5) /(1.0 + (0.001 * pow(pow(element, 2.0) - (2.0 * element * nextElem) + pow(nextElem, 2.0), 2.0))))

return the final resulting value

4.2.2.11 quarticHost()

```
double quarticHost (
            const double * vector,
            int numDim )
```

< Initialize the variable to store the resulting calculation

Loop through the vector for testing, calculating the sum of the following for each element (j + 1.0) * pow(vector[j], 4.0)

return the final resulting value

4.2.2.12 ranaHost()

```
double ranaHost (
            const double * vector,
            int numDim )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Declare the variable to represent the next element

Loop through the vector for testing, calculating the sum of the following for each element (element * sin(sqrt(fabs(nextElem - element + 1.0))) * cos(sqrt(fabs(nextElem + element + 1.0))))

- ((nextElem + 1.0) * cos(sqrt(fabs(nextElem - element + 1.0))) * sin(sqrt(fabs(nextElem + element + 1.0))))

return the final resulting value

4.2.2.13 rastgrinHost()

```
double rastgrinHost (
            const double * vector,
            int numDim )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Set the Rastgrin constant to multiply the sum by after calculation

Loop through the vector for testing, take the current element squared and subtract 10 times the cos(2 times pi times the current element), sum all of the values transformed in this manner, and multiply the total sum by the constant defined above. (10 times the number of dimensions/elements.)

return the final resulting value

4.2.2.14 rosenbrockHost()

```
double rosenbrockHost (
            const double * vector,
            int numDim )
```

< Implement the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Declare the variable to represent the next element

Loop through the vector for testing, multiply 100 by (current element squared - the next element) squared, then add (1 - the current element) squared, and sum all of these transformed values.

return the final resulting value

4.2.2.15 schwefelHost()

```
double schwefelHost (
            const double * vector,
            int numDim )
```

< Declare the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Schwefel's constant, subtract the summation result from this number for final result

Loop through the vector for testing, multiply the current element by -1 and multiply that by sin(sqrt(|element|)). Once the total sum of all elements transformed in that manner is calculated subtract the sum from the constant defined above.

< Store the current element being operated on

< Calculate the sum

< store to the calculated value: constant - sum

return the final resulting value

4.2.2.16 sineEnvSineWaveHost()

```
double sineEnvSineWaveHost (
            const double * vector,
            int numDim )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Declare the variable to represent the next element

Loop through the vector for testing, adding 0.5 to sin((currElement^2 + nextElement^2 - 0.5)^2)/(1.0 + 0.↩ 001∗(currElement^2 + nextElement^2)^2) once the total value has been calculated multiply it by -1 for the final result.

return the final resulting value

4.2.2.17 stepHost()

```
double stepHost (
            const double * vector,
            int numDim )
```

< Initialize the variable to store the resulting calculation

Loop through the vector for testing, calculating the sum of the following for each element pow(fabs(vector[j]) + 0.5, 2.0)

return the final resulting value

4.2.2.18 stretchVSineWaveHost()

```
double stretchVSineWaveHost (
            const double * vector,
            int numDim )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Declare the variable to represent the next element

Loop through the vector for testing, calculating the sum of the following for each element (currElement^2 + next↩ Element^2)^0.25 * sin((50 * (currElement^2 + nextElement^2)^0.1)^2) + 1

return the final resulting value

4.3 General/Equations.h File Reference

This is where the methods for the calls in General/TestTypes.c are defined.

Functions

- double schwefelHost (const double *, int)
- double deJongHost (const double *, int)
- double rosenbrockHost (const double *, int)
- double rastgrinHost (const double *, int)
- double griewangkHost (const double *, int)
- double sineEnvSineWaveHost (const double *, int)
- double stretchVSineWaveHost (const double *, int)
- double ackleyOneHost (const double *, int)
- double ackleyTwoHost (const double *, int)
- double eggHolderHost (const double *, int)
- double ranaHost (const double *, int)
- double pathologicalHost (const double *, int)
- double michalewiczHost (const double *, int)
- double mastersCosineWaveHost (const double *, int)
- double quarticHost (const double *, int)
- double levyHost (const double *, int)
- double stepHost (const double *, int)
- double alpineHost (const double *, int)

4.3.1 Detailed Description

This is where the methods for the calls in General/TestTypes.c are defined.

Here we define and describe the methods which are referenced in General/TestTypes.c and implemented in General/Equations.c. Based on the equation we process a randomized vector stored in a TestInfo struct matrix, perform different calculations returning the result for the vector.

4.3.2 Function Documentation

4.3.2.1 ackleyOneHost()

```
double ackleyOneHost (
            const double * ,
            int  )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Declare the variable to represent the next element

< Initialize the ackley constant to 1/e^0.2

Loop through the vector for testing, calculating the sum of the following for each element ackleyConst * sqrt((currElement^2 + nextElement^2)) + (3 * cos(2 * currElement)) + sin(2 * nextElement)

return the final resulting value

4.3.2.2 ackleyTwoHost()

```
double ackleyTwoHost (
            const double * ,
            int  )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Declare the variable to represent the next element

Loop through the vector for testing, calculating the sum of the following for each element 20 + e - (20 / e^(0.2 * sqrt((currElement^2 + nextElement^2) / 2))) - e^(0.5 * (cos(2 * pi * currElement) + cos(2 * pi * nextElement)))

return the final resulting value

4.3.2.3 alpineHost()

```
double alpineHost (
            const double * ,
            int  )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

Loop through the vector for testing, calculating the sum of the following for each element fabs((element * sin(element)) + (0.1 * element))

return the final resulting value

4.3.2.4 deJongHost()

```
double deJongHost (
            const double * ,
            int  )
```

< Initialize the variable to store the resulting calculation

Loop through the vector for testing, for each element square it, and sum all the squares.

return the final resulting value

4.3.2.5 eggHolderHost()

```
double eggHolderHost (
            const double * ,
            int  )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Declare the variable to represent the next element

Loop through the vector for testing, calculating the sum of the following for each element (-1 * currElement * sin(sqrt(absoluteVal(currElement - nextElement -47)))) - ((nextElement + 47) * sin(sqrt(absoluteVal(nextElement + 47 + (currElement / 2)))))

return the final resulting value

4.3.2.6 griewangkHost()

```
double griewangkHost (
            const double * ,
            int  )
```

< Initializing the variable to store the summation result

< Initializing the variable to store the product result

< Initialize the variable to store the final result

< Declare the variable to represent the current element of the summation

Loop through the vector for testing, for the summation result, add all terms of the current element squared divided by 4000, and for the product result multiply all terms of cos(current element divided by the square root of the current index). Once the summation and product are complete take 1 + the summation result - the product result.

return the final resulting value

4.3.2.7 levyHost()

```
double levyHost (
            const double * ,
            int  )
```

< Initialize the variable to store the resulting calculation

< Initialize the variable for storing W0 in the vector

< Initialize the variable for storing Wn in the vector

< Declares the variable for declaring Wi for the current dimension in the vector

Loop through the vector for testing, calculating the sum of the following for each element first calculating Wi = 1.0 + ((vector[j] - 1.0)/4.0) (pow(wi - 1.0, 2.0) ∗ (1.0 + 10.0 ∗ pow(sin((M_PI ∗ wi) + 1.0), 2.0))) + (pow(Wn - 1.0, 2.0) ∗ (1.0 + pow(sin(2.0 ∗ M_PI ∗ Wn), 2.0))) once this total is acquired adding pow(sin(M_PI ∗ W0), 2.0) for the final result

return the final resulting value

4.3.2.8 mastersCosineWaveHost()

```
double mastersCosineWaveHost (
            const double * ,
            int  )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Declare the variable to represent the next element

Loop through the vector for testing, calculating the sum of the following for each element exp((-1.0 / 8.0) ∗(pow(element, 2.0) + pow(nextElem, 2.0) + (0.5 ∗ nextElem ∗ element))) cos(pow(pow(element, 2.0) + pow(next↩ Elem, 2.0) + (0.5 ∗ nextElem ∗ element), 0.25))

return the final resulting value

4.3.2.9 michalewiczHost()

```
double michalewiczHost (
            const double * ,
            int  )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

Loop through the vector for testing, calculating the sum of the following for each element sin(element) ∗ pow(sin(((j+1) ∗ pow(element, 2.0)) / M_PI), 20.0)

return the final resulting value

4.3.2.10 pathologicalHost()

```
double pathologicalHost (
            const double * ,
            int  )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Declare the variable to represent the next element

Loop through the vector for testing, calculating the sum of the following for each element 0.5 + ((sin(pow(sqrt((100.0 * pow(element, 2.0)) + pow(nextElem, 2.0)),2.0)) - 0.5) /(1.0 + (0.001 * pow(pow(element, 2.0) - (2.0 * element * nextElem) + pow(nextElem, 2.0), 2.0))))

return the final resulting value

4.3.2.11 quarticHost()

```
double quarticHost (
            const double * ,
            int  )
```

< Initialize the variable to store the resulting calculation

Loop through the vector for testing, calculating the sum of the following for each element (j + 1.0) * pow(vector[j], 4.0)

return the final resulting value

4.3.2.12 ranaHost()

```
double ranaHost (
            const double * ,
            int  )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Declare the variable to represent the next element

Loop through the vector for testing, calculating the sum of the following for each element (element * sin(sqrt(fabs(nextElem - element + 1.0))) * cos(sqrt(fabs(nextElem + element + 1.0))))

- ((nextElem + 1.0) * cos(sqrt(fabs(nextElem - element + 1.0))) * sin(sqrt(fabs(nextElem + element + 1.0))))

return the final resulting value

4.3.2.13 rastgrinHost()

```
double rastgrinHost (
            const double * ,
            int  )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Set the Rastgrin constant to multiply the sum by after calculation

Loop through the vector for testing, take the current element squared and subtract 10 times the cos(2 times pi times the current element), sum all of the values transformed in this manner, and multiply the total sum by the constant defined above. (10 times the number of dimensions/elements.)

return the final resulting value

4.3.2.14 rosenbrockHost()

```
double rosenbrockHost (
            const double * ,
            int  )
```

< Implement the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Declare the variable to represent the next element

Loop through the vector for testing, multiply 100 by (current element squared - the next element) squared, then add (1 - the current element) squared, and sum all of these transformed values.

return the final resulting value

4.3.2.15 schwefelHost()

```
double schwefelHost (
            const double * ,
            int  )
```

< Declare the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Schwefel's constant, subtract the summation result from this number for final result

Loop through the vector for testing, multiply the current element by -1 and multiply that by sin(sqrt(|element|)). Once the total sum of all elements transformed in that manner is calculated subtract the sum from the constant defined above.

< Store the current element being operated on

< Calculate the sum

< store to the calculated value: constant - sum

return the final resulting value

4.3.2.16 sineEnvSineWaveHost()

```
double sineEnvSineWaveHost (
            const double * ,
            int  )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Declare the variable to represent the next element

Loop through the vector for testing, adding 0.5 to sin((currElement^2 + nextElement^2 - 0.5)^2)/(1.0 + 0.↩ 001∗(currElement^2 + nextElement^2)^2) once the total value has been calculated multiply it by -1 for the final result.

return the final resulting value

4.3.2.17 stepHost()

```
double stepHost (
            const double * ,
            int  )
```

< Initialize the variable to store the resulting calculation

Loop through the vector for testing, calculating the sum of the following for each element pow(fabs(vector[j]) + 0.5, 2.0)

return the final resulting value

4.3.2.18 stretchVSineWaveHost()

```
double stretchVSineWaveHost (
            const double * ,
            int  )
```

< Initialize the variable to store the resulting calculation

< Declare the variable to represent the current element of the summation

< Declare the variable to represent the next element

Loop through the vector for testing, calculating the sum of the following for each element (currElement^2 + next↩ Element^2)^0.25 ∗ sin((50 ∗ (currElement^2 + nextElement^2)^0.1)^2) + 1

return the final resulting value

4.4 General/FA.h File Reference

This is where all methods pertaining to the Firefly Algorithm implementation are defined.

```
#include "Utilities.h"
```

Functions

- void * fireflyAlg (void *data)

  Responsible for initializing the algorithm and all data related to the algorithm and calling related functions to start and record results fo the algorithm.
- double lightIntensity (const double *fitness, int iPos, double gamma, double distance)

  Calculates the light intensity of the firefly using a given distance and light absorption rate gamma and the current fitness.
- void fireflyLoop (FireflySwarm *fireflies, double **temp, int popSize, double beta, double gamma, double alpha, int equation, double *range)

  Loops through the population calling the move firefly loop function.
- double calcDistanceSquared (const double *fireflyI, const double *fireflyJ, int dimensions)

  Squares the difference of each dimension between the two vectors and sums them. No square root is needed as it is the distance squared.
- double calcAttractiveness (double **temp, int iPos, int jPos, int currDim, double beta, double gamma, double distance)

  Calculates the term representing the attractiveness and how far the new firefly will move toward the compared firefly.
- double * calcAttractedVector (double **temp, int iPos, int jPos, int dimensions, double beta, double alpha, double gamma, const double *range, double distance)

  Calculates a new vector based on the current value added to the attractiveness, added to some random movement in the range.
- void moveFirefliesLoop (FireflySwarm *fireflies, double **temp, int iPos, double beta, double gamma, double alpha, int equation, int popSize, double *range)

  loops through the entire population and compares the light intensity of each firefly to see if a newly attracted firefly is to be created.
- void newBest (FireflySwarm *pop, double *newVector, double newResult, int popSize)

  updates the metadata regarding the best firefly in the population
- void addVector (FireflySwarm *pop, double *newVector, double newResult, int popSize)

  updates the metadata regarding the worst firefly in the population

4.4.1 Detailed Description

This is where all methods pertaining to the Firefly Algorithm implementation are defined.

Firefly algorithm is the process of taking each firefly/vector and comparing it to every other firefly in the population. If the fitness of the compared vector is better than the current vector then a new vector is created moving dimensions in the direction of the better vector and then evaluating this newly developed vector. This new vector replaces the worst vector and the process continues.

4.4.2 Function Documentation

4.4.2.1 addVector()

```
void addVector (
            FireflySwarm * pop,
            double * newVector,
            double newResult,
            int popSize )
```

updates the metadata regarding the worst firefly in the population

Parameters

pop	- the struct being processed
newVector	- the newly created firefly
newResult	- the fitness of the newly created firefly
popSize-	the size of the firefly population

4.4.2.2 calcAttractedVector()

```
double* calcAttractedVector (
            double ** temp,
            int iPos,
            int jPos,
            int dimensions,
            double beta,
            double alpha,
            double gamma,
            const double * range,
            double distance )
```

Calculates a new vector based on the current value added to the attractiveness, added to some random movement in the range.

Parameters

temp	- the temporary population being considered in the loop
iPos	- the position of the current firefly in the population
jPos	- the position of the compared firefly in the population
dimensions	- the number of dimensions in the firefly
beta	- attractiveness factor
alpha	- the randomness factor for the final term
gamma	- light absorption rate
range	- the range of acceptable values for the equation
distance	- distance between fireflies

Returns

returns a newly developed firefly

4.4.2.3 calcAttractiveness()

```
double calcAttractiveness (
            double ** temp,
            int iPos,
            int jPos,
```

```
            int currDim,
            double beta,
            double gamma,
            double distance )
```

Calculates the term representing the attractiveness and how far the new firefly will move toward the compared firefly.

Parameters

temp	- the temporary population being considered in the loop
iPos	- the position of the current firefly in the population
jPos	- the position of the compared firefly in the population
currDim	- current dimension being processed
beta	- attractiveness factor
gamma	- light absorption rate
distance	- distance between fireflies

Returns

A double value representing the attractiveness of the firefly inverse square proportional to the distance

4.4.2.4 calcDistanceSquared()

```
double calcDistanceSquared (
            const double * fireflyI,
            const double * fireflyJ,
            int dimensions )
```

Squares the difference of each dimension between the two vectors and sums them. No square root is needed as it is the distance squared.

Parameters

fireflyI	- first firefly to be iterated
fireflyJ	- second firefly to be iterated
dimensions	- number of dimensions in the firefly

Returns

A double value representing the distance squared between the two fireflies

4.4.2.5 fireflyAlg()

```
void* fireflyAlg (
            void * data )
```

Responsible for initializing the algorithm and all data related to the algorithm and calling related functions to start and record results fo the algorithm.

Parameters

data	- void pointer to be converted to an EquationInfo struct

Returns

Nothing as it is a threaded function

< Declare the timespec struct storing the start time of the iterations

< Declare the timespec struct storing the end time of the iterations

< Declare the double storing the total runtime of all iterations in milliseconds

< Set the start time using a monotonic clock, meaning it will ignore if the system clock changes

< Store the end time

< Calculate the total runtime by subtracting end time's seconds from the start time's seconds and converting to milliseconds and adding the end time's nanoseconds minus the start time's nanoseconds and converting to milliseconds

write the best and worst to a file

write the new population to a log file

4.4.2.6 fireflyLoop()

```
void fireflyLoop (
            FireflySwarm * fireflies,
            double ** temp,
            int popSize,
            double beta,
            double gamma,
            double alpha,
            int equation,
            double * range )
```

Loops through the population calling the move firefly loop function.

Parameters

fireflies	- struct being processed in the iteration
temp	- the temporary population which will not be updated
popSize	- the size of the population
beta	- the attractiveness factor
gamma	- the light absorption rate
alpha	- the scaling factor on the range
equation	- equation number for the objective function call
range	- range of values acceptable for the population

4.4.2.7 lightIntensity()

```
double lightIntensity (
            const double * fitness,
            int iPos,
            double gamma,
            double distance )
```

Calculates the light intensity of the firefly using a given distance and light absorption rate gamma and the current fitness.

Parameters

fitness	- array of fitness values
iPos	- fitness position of the light intensity being calculated
gamma	- light absorption rate
distance	- distance between fireflies

Returns

The value of the light intensity inverse squarely proportional to the distance

4.4.2.8 moveFirefliesLoop()

```
void moveFirefliesLoop (
            FireflySwarm * fireflies,
            double ** temp,
            int iPos,
            double beta,
            double gamma,
            double alpha,
            int equation,
            int popSize,
            double * range )
```

loops through the entire population and compares the light intensity of each firefly to see if a newly attracted firefly is to be created.

Parameters

fireflies	- the struct being processed
temp	- the temporary population being considered in the loop
iPos	- the position of the current firefly in the population
beta	- attractiveness factor
gamma	- light absorption rate
alpha	- the randomness factor for the final term
equation	- equation number for the objective function call
popSize	- the population size of fireflies
range	- the range of acceptable values for the equation

4.4.2.9 newBest()

```
void newBest (
            FireflySwarm * pop,
            double * newVector,
            double newResult,
            int popSize )
```

updates the metadata regarding the best firefly in the population

Parameters

pop	- the struct being processed
newVector	- the newly created firefly
newResult	- the fitness of the newly created firefly
popSize-	the size of the firefly population

4.5 General/Harmonic.c File Reference

This is where all methods defined in General/Harmonic.h are implemented.

```
#include "Harmonic.h"
#include "MersenneMatrix.h"
#include "Utilities.h"
#include "Equations.h"
```

Functions

- void * harmonicTest (void *data)

  Responsible for initializing the algorithm and all data related to the algorithm and calling related functions to start and record results fo the algorithm.
- void updateBest (HPop *pop, double newResult, EquationInfo info)

  updates the metadata pertaining to the best fitness within the struct
- void newVector (HPop *pop, double newResult, EquationInfo info)

  adds the vector to the population at the worst fitness position
- void pitchAdjustment (double *harmonic, int position, double bandwidth, const double *range)

  adjusts the pitch of the current harmonic based on a random number, the current value and the bandwidth
- void harmonicIteration (HPop *hpop, int NI, int HMS, double HMCR, double PAR, double bandwidth, const double *range)

  This iterates through the number of dimensions and grabs random values from the range or population and occasionally adjusts the value at each dimension.

4.5.1 Detailed Description

This is where all methods defined in General/Harmonic.h are implemented.

4.5.2 Function Documentation

4.5.2.1 harmonicIteration()

```
void harmonicIteration (
            HPop * hpop,
            int NI,
            int HMS,
            double HMCR,
            double PAR,
            double bandwidth,
            const double * range )
```

This iterates through the number of dimensions and grabs random values from the range or population and occasionally adjusts the value at each dimension.

Parameters

hpop	- the struct being processed
NI	- The number of dimensions in the Harmonics
HMS	- Harmonic Size or population size
HMCR	- Harmonic Conssideration Rate
PAR	- Pitch Adjustment Rate
bandwidth	- the bandwidth for the tuning
range	- the range of acceptable values in the search space

4.5.2.2 harmonicTest()

```
void * harmonicTest (
            void * data )
```

Responsible for initializing the algorithm and all data related to the algorithm and calling related functions to start and record results fo the algorithm.

Parameters

data	- void pointer to be converted to an EquationInfo struct

Returns

Nothing as it is a threaded function

< Declare the timespec struct storing the start time of the iterations

< Declare the timespec struct storing the end time of the iterations

< Declare the double storing the total runtime of all iterations in milliseconds

< Set the start time using a monotonic clock, meaning it will ignore if the system clock changes

< Store the end time

< Calculate the total runtime by subtracting end time's seconds from the start time's seconds and converting to milliseconds and adding the end time's nanoseconds minus the start time's nanoseconds and converting to milliseconds

write the best and worst to a file

write the new population to a log file

4.5.2.3 newVector()

```
void newVector (
            HPop * pop,
            double newResult,
            EquationInfo info )
```

adds the vector to the population at the worst fitness position

Parameters

pop	- the struct being processed
newResult	- the newly calculated fitness value
info	- The EquationInfo struct storing equation specific information

4.5.2.4 pitchAdjustment()

```
void pitchAdjustment (
            double * harmonic,
            int position,
            double bandwidth,
            const double * range )
```

adjusts the pitch of the current harmonic based on a random number, the current value and the bandwidth

Parameters

harmonic	- the harmonic to be adjusted
position	- the dimension within the harmonic being adjusted
bandwidth	- the bandwidth for the tuning
range	- the range of acceptable values in the search space

4.5.2.5 updateBest()

```
void updateBest (
            HPop * pop,
            double newResult,
            EquationInfo info )
```

updates the metadata pertaining to the best fitness within the struct

Parameters

pop	- the struct being processed
newResult	- the newly calculated fitness value
info	- The EquationInfo struct storing equation specific information

4.6 General/Harmonic.h File Reference

This is where all methods pertaining to the Harmonic Search Algorithm implementation are defined.

```
#include "Utilities.h"
```

Functions

- void * harmonicTest (void *data)

  Responsible for initializing the algorithm and all data related to the algorithm and calling related functions to start and record results fo the algorithm.
- void pitchAdjustment (double *harmonic, int position, double bandwidth, const double *range)

  adjusts the pitch of the current harmonic based on a random number, the current value and the bandwidth
- void updateBest (HPop *pop, double newResult, EquationInfo info)

  updates the metadata pertaining to the best fitness within the struct
- void newVector (HPop *pop, double newResult, EquationInfo info)

  adds the vector to the population at the worst fitness position
- void harmonicIteration (HPop *hpop, int NI, int HMS, double HMCR, double PAR, double bandwidth, const double *range)

  This iterates through the number of dimensions and grabs random values from the range or population and occasionally adjusts the value at each dimension.

4.6.1 Detailed Description

This is where all methods pertaining to the Harmonic Search Algorithm implementation are defined.

Harmonic Search takes random dimensions from among the population, adjusts the value to test within the neighborhood, and then produces a single vector to be tested per iteration saving it if it is better than the worst vector and reevaluating the population for the new worst vector.

4.6.2 Function Documentation

4.6.2.1 harmonicIteration()

```
void harmonicIteration (
            HPop * hpop,
            int NI,
            int HMS,
            double HMCR,
            double PAR,
            double bandwidth,
            const double * range )
```

This iterates through the number of dimensions and grabs random values from the range or population and occasionally adjusts the value at each dimension.

Parameters

hpop	- the struct being processed
NI	- The number of dimensions in the Harmonics
HMS	- Harmonic Size or population size
HMCR	- Harmonic Conssideration Rate
PAR	- Pitch Adjustment Rate
bandwidth	- the bandwidth for the tuning
range	- the range of acceptable values in the search space

4.6.2.2 harmonicTest()

```
void* harmonicTest (
            void * data )
```

Responsible for initializing the algorithm and all data related to the algorithm and calling related functions to start and record results fo the algorithm.

Parameters

data	- void pointer to be converted to an EquationInfo struct

Returns

Nothing as it is a threaded function

< Declare the timespec struct storing the start time of the iterations

< Declare the timespec struct storing the end time of the iterations

< Declare the double storing the total runtime of all iterations in milliseconds

< Set the start time using a monotonic clock, meaning it will ignore if the system clock changes

< Store the end time

< Calculate the total runtime by subtracting end time's seconds from the start time's seconds and converting to milliseconds and adding the end time's nanoseconds minus the start time's nanoseconds and converting to milliseconds

write the best and worst to a file

write the new population to a log file

4.6.2.3 newVector()

```
void newVector (
            HPop * pop,
            double newResult,
            EquationInfo info )
```

adds the vector to the population at the worst fitness position

Parameters

pop	- the struct being processed
newResult	- the newly calculated fitness value
info	- The EquationInfo struct storing equation specific information

4.6.2.4 pitchAdjustment()

```
void pitchAdjustment (
            double * harmonic,
            int position,
            double bandwidth,
            const double * range )
```

adjusts the pitch of the current harmonic based on a random number, the current value and the bandwidth

Parameters

harmonic	- the harmonic to be adjusted
position	- the dimension within the harmonic being adjusted
bandwidth	- the bandwidth for the tuning
range	- the range of acceptable values in the search space

4.6.2.5 updateBest()

```
void updateBest (
            HPop * pop,
            double newResult,
            EquationInfo info )
```

updates the metadata pertaining to the best fitness within the struct

Parameters

pop	- the struct being processed
newResult	- the newly calculated fitness value
info	- The EquationInfo struct storing equation specific information

4.7 General/HostCalls.h File Reference

This is where all the references to the objective function call methods are stored to avoid duplication.

```
#include "Equations.h"
```

Variables

- const void * equationHostCalls [ ]

  An array of pointers to the equation methods to be used by threads in runEquationsAsThreads()

4.7.1 Detailed Description

This is where all the references to the objective function call methods are stored to avoid duplication.

4.7.2 Variable Documentation

4.7.2.1 equationHostCalls

```
const void* equationHostCalls[]
```

Initial value:

```
={
        &schwefelHost,
        &deJongHost,
        &rosenbrockHost,
        &rastgrinHost,
        &griewangkHost,
        &sineEnvSineWaveHost,
        &stretchVSineWaveHost,
        &ackleyOneHost,
```

```
        &ackleyTwoHost,
        &eggHolderHost,
        &ranaHost,
        &pathologicalHost,
        &michalewiczHost,
        &mastersCosineWaveHost,
        &quarticHost,
        &levyHost,
        &stepHost,
        &alpineHost
}
```

An array of pointers to the equation methods to be used by threads in runEquationsAsThreads()

This is used as an easy way to iterate through the equation methods when creating the threads to start processing each dimension to be tested for each equation in different threads.

4.8 General/Init.c File Reference

Handles processing the input file for use throughout this testing.

```
#include "Utilities.h"
#include "Init.h"
```

Enumerations

- enum inputFlag { NotRead, Reading, Read }

Functions

- int checkTestType ()

  This is the method which takes user input to define the type of tests to run; either Genetic Algorithm or Differential Evolution.
- int processVectors (char *arg, Info *progInfo)

  This is the method which processes the number of vectors from the line containing this information.
- int processExperiments (char *arg, Info *progInfo)
- int processDimensions (char *arg, Info *progInfo)

  This is the method which processes the number of different dimensions from the line containing this information.
- int processDimList (char *arg, Info *progInfo)

  This is the method which processes the array of dimensions to test from the line containing this information.
- int processEquations (char *arg, Info *progInfo)

  This is the method which processes the number of equations to test from the line containing this information.
- int processRanges (char *arg, Info *progInfo)

  This is the method which processes the range values for each equation from the line containing this information.
- int processIterations (char *arg, Info *progInfo)

  This is the method which processes the number of iterations/generations for Genetic Algorithm and Differential Evolution from the line containing this information.
- int processBeta (char *arg, Info *progInfo)

  This is the method which processes the beta/attractiveness factor from the line containing this information.
- int processGamma (char *arg, Info *progInfo)

  This is the method which processes the gamma/light absorption rate from the line containing this information.
- int processAlpha (char *arg, Info *progInfo)

This is the method which processes the alpha/random scaling factor from the line containing this information.

- int processDampener (char *arg, Info *progInfo)

  This is the method which processes the velocity dampener value applied in General/PSO.c from the line containing this information.

- int processC1 (char *arg, Info *progInfo)

  This is the method which processes the personal best modification factor applied in General/PSO.c from the line containing this information.

- int processC2 (char *arg, Info *progInfo)

  This is the method which processes the global best modification factor applied in General/PSO.c from the line containing this information.

- int processHMCR (char *arg, Info *progInfo)

  This is the method which processes the Harmonic Memory Consideration Rate from the line containing this information.

- int processPAR (char *arg, Info *progInfo)

  This is the method which processes the pitch adjustment rate from the line containing this information.

- int processBandwidth (char *arg, Info *progInfo)

  This is the method which processes the bandwidth from the line containing this information.

- int init (char *filename, Info *progInfo)

  This is the method which reads in the init file, and processes the information contained within into the progInfo Info struct using helper methods based on the flags which are set during processing.

4.8.1 Detailed Description

Handles processing the input file for use throughout this testing.

Processes all required lines as designated in the readme file and stores the value to an Info struct pointer and asks for the users input on which test type to run. This info is then used in PThread/main.c or Win32/main32.c and throughout the tests.

4.8.2 Enumeration Type Documentation

4.8.2.1 inputFlag

```
enum inputFlag
```

Enum used for the state of the flags for lines which are being read

Enumerator

NotRead	Signifies that the current line has not been read
Reading	Signifies that the current line is being read and values need to be processed
Read	Signifies that the information for that current line has been processed and stored

4.8.3 Function Documentation

4.8.3.1 checkTestType()

```
int checkTestType ( )
```

This is the method which takes user input to define the type of tests to run; either Genetic Algorithm or Differential Evolution.

Returns

0 for Differential Evolution, 1 for Genetic Algoriothm;

< Signify we are reading the test type input

< declare an array for the user input

Infinitely loop until we get a correct input from the user of 'Random', 'Local', or 'Iterative'

prompt the user for input then listen for input. if the user inputs 'Particle' return the value for PSO defined in the TestType enum within General/Utilities.h, otherwise if the user inputs 'Firefly' return Firefly from this same enum, or if the user inputs 'Harmonic' return Harmonic from this same enum. Otherwise tell the user it is an incorrect response and prompt the user for input again.

4.8.3.2 init()

```
int init (
            char * filename,
            Info * progInfo )
```

This is the method which reads in the init file, and processes the information contained within into the progInfo Info struct using helper methods based on the flags which are set during processing.

This iterates through the lines of the init file tokenizing the lines on '=' and then sets flags based on what line was just read in. It checks that all of the lines which require an order are processed in the right order and returns -1 on a failure. Failures: Not enough argument lines, Arguments are out of order, couldn't convert a value properly, incorrect number of matching arguments e.g. 3 dimensions expected only 2 provided in the dimension list.

Parameters

progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< Declare a char array of LINE_LENGTH defined in General/Utilities.h to store the current line being processed

< Initialize the flag representing the dampener line having been read to NotRead

< Initialize the flag representing the C1 line having been read to NotRead

< Initialize the flag representing the C2 (Roulette or Tournament) line having been read to NotRead

< Initialize the flag representing the beta/attractiveness factor line having been read to NotRead

< Initialize the flag representing the number of experiments line having been read to NotRead

< Initialize the flag representing the number of vectors line having been read to NotRead

< Initialize the flag representing the number of dimensions line having been read to NotRead

< Initialize the flag representing the different dimensions line having been read to NotRead

< Initialize the flag representing the number of equations line having been read to NotRead

< Initialize the flag representing the alpha line having been read to NotRead

< Initialize the flag representing the ranges line having been read to NotRead

< Initialize the flag representing the number of iterations line having been read to NotRead

< Initialize the flag representing the gamma line having been read to NotRead

< Initialize the flag representing the pitch adjustment line having been read to NotRead

< Initialize the flag representing the HMCR line having been read to NotRead

< Initialize the flag representing the bandwidth line having been read to NotRead

< Initialize a variable tracking the result of comparing to the expected dimensions line flag

< Initialize a variable tracking the result of comparing to the expected ranges line flag

< Stores the tokenized string from the line read in

while we still have lines to read

print the line being read

split the line on '='

while there are more tokens splitting on '='

If we are reading the number of vectors line, processVectors and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the number of dimensions line, processDimensions and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the different dimensions line, processDimList and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the number of equations line, processEquations and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the ranges line, processRanges and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the number of iterations line, processIterations and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the gamma line, processGamma and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the pitch adjustment line, processPAR and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the number of experiments line, processExperiments and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the alpha line, processAlpha and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the dampener line, processDampener and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the C1 line, processC1 and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the C2 line, processC2 and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the beta line, processBeta and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the HMCR line, processHMCR and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the bandwidth line, processBandwidth and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

This point is reached if the program is not currently reading any of the specified lines and is looking to see if it is reading one of the expected lines.

If we haven't read the number of vectors line already and it equals the signifier for the number of vectors line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the number of dimensions line already and it equals the signifier for the number of dimensions line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the different dimensions line already and it equals the signifier for the different dimensions line, and we have read the number of dimensions line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

Otherwise if we have come to the different dimensions line and have not read the number of dimensions in tell the user, close the file and return failure.

If we haven't read the number of Equations line already and it equals the signifier for the number of Equations line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the ranges line already and it equals the signifier for the ranges line, and we have read the number of equations line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

Otherwise if we have come to the ranges line and have not read the number of equations in tell the user, close the file and return failure.

If we haven't read the number of Generations/Iterations line already and it equals the signifier for the number of Generations/Iterations line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the Gamma line already and it equals the signifier for the Gamma line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the pitch adjustment line already and it equals the signifier for the pitch adjustment line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the number of experiments line already and it equals the signifier for the number of experiments line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the alpha line already and it equals the signifier for the alpha line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the dampener line already and it equals the signifier for the dampener line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the C1 line already and it equals the signifier for the C1 line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the C2 line already and it equals the signifier for the C2 line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the beta line already and it equals the signifier for the beta line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the HMCR line already and it equals the signifier for the HMCR line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the bandwidth line already and it equals the signifier for the bandwidth line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

This point is reached if none of the proper tags were found, exiting the arg while loop and effectively moving to the next line.

If one of the expected/required lines was not Read tell the user and close the file then return failure. Otherwise close the file ask the user for the test type and then return success.

4.8.3.3 processAlpha()

```
int processAlpha (
            char * arg,
            Info * progInfo )
```

This is the method which processes the alpha/random scaling factor from the line containing this information.

Failure: A value which does not convert to a positive real number

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

Attempt to convert the value in the given string arg to a double and if it is not a positive double tell the user and return failure. Otherwise return success.

4.8.3.4 processBandwidth()

```
int processBandwidth (
            char * arg,
            Info * progInfo )
```

This is the method which processes the bandwidth from the line containing this information.

Failure: A value which does not convert to a positive real number

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

Attempt to convert the value in the given string arg to a double and if it is not a positive double tell the user and return failure. Otherwise return success.

4.8.3.5 processBeta()

```
int processBeta (
            char * arg,
            Info * progInfo )
```

This is the method which processes the beta/attractiveness factor from the line containing this information.

Failure: A value which does not convert to a positive real number

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

Attempt to convert the value in the given string arg to a double and if it is not a positive double tell the user and return failure. Otherwise return success.

4.8.3.6 processC1()

```
int processC1 (
            char * arg,
            Info * progInfo )
```

This is the method which processes the personal best modification factor applied in General/PSO.c from the line containing this information.

Failure: A value which does not convert to a positive real number between (0, 2]

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

Attempt to convert the value in the given string arg to a double and if it is not a positive double tell the user and return failure. Otherwise return success.

4.8.3.7 processC2()

```
int processC2 (
            char * arg,
            Info * progInfo )
```

This is the method which processes the global best modification factor applied in General/PSO.c from the line containing this information.

Failure: A value which does not convert to a positive real number between (0, 2]

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

Attempt to convert the value in the given string arg to a double and if it is not a positive double tell the user and return failure. Otherwise return success.

4.8.3.8 processDampener()

```
int processDampener (
            char * arg,
            Info * progInfo )
```

This is the method which processes the velocity dampener value applied in General/PSO.c from the line containing this information.

Failure: A value which does not convert to a positive real number

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

Attempt to convert the value in the given string arg to a double and if it is not a positive double tell the user and return failure. Otherwise return success.

4.8.3.9 processDimensions()

```
int processDimensions (
            char * arg,
            Info * progInfo )
```

This is the method which processes the number of different dimensions from the line containing this information.

Failure: can't convert to a positive number

Parameters

arg	- The portion of the line containing the value to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

Attempt to convert the value in the given string arg to an integer and if it is not a positive integer tell the user and return failure. Otherwise allocate space for the array of different dimensions and return success.

4.8.3.10 processDimList()

```
int processDimList (
            char * arg,
            Info * progInfo )
```

This is the method which processes the array of dimensions to test from the line containing this information.

Failure: can't convert to a positive number, not enough or too many dimensions provided

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

Tokenize the current string arg on ',' and initialize variables for processing the line. dim will be the current dimension being processed in the tokenized string, dimCounter counts the number of dimensions which have been processed, not to exceed numDim which is the number of dimensions declared in the input file. NumDimensions is required to be defined before this method is called.

While there are dimensions to process, check that it hasn't exceeded the number of expected dimensions, otherwise tell the user and return failure. Attempt to parse the string to an integer and if it is not a positive integer tell the user and return failure. If successfully parsed set the processed dimension to the corresponding position in the array, increase the counter, and get the next token. Lastly, after all values have been parsed, make sure we have parsed the right number of values, if not, tell the user and return failure, otherwise return success.

If not enough dimensions were provided tell the user and return failure otherwise return success.

4.8.3.11 processEquations()

```
int processEquations (
            char * arg,
            Info * progInfo )
```

This is the method which processes the number of equations to test from the line containing this information.

Failure: can't convert to a positive number or is beyond the maximum number of equations in this testing suite.

Parameters

arg	- The portion of the line containing the value to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

Attempt to convert the value in the given string arg to an integer and if it is not a positive integer tell the user and return failure.

< Store the number of equations locally

if the number of equations defined is larger than the MAX_NUM_EQUATIONS defined in General/Utilities.h tell the user and return failure. Otherwise allocate the arrays to stor optimum values and ranges for each equation and return success.

4.8.3.12 processExperiments()

```
int processExperiments (
            char * arg,
            Info * progInfo )
```

Attempt to convert the value in the given string arg to an integer and if it is not a positive integer tell the user and return failure. Otherwise return success.

4.8.3.13 processGamma()

```
int processGamma (
            char * arg,
            Info * progInfo )
```

This is the method which processes the gamma/light absorption rate from the line containing this information.

Failure: A value which does not convert to a positive real number

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

Attempt to convert the value in the given string arg to a double and if it is not a positive double tell the user and return failure. Otherwise return success.

4.8.3.14 processHMCR()

```
int processHMCR (
            char * arg,
            Info * progInfo )
```

This is the method which processes the Harmonic Memory Consideration Rate from the line containing this information.

Failure: A value which does not convert to a positive real number between [0, 1]

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

Attempt to convert the value in the given string arg to a double and if it is not a positive double tell the user and return failure. Otherwise return success.

4.8.3.15 processIterations()

```
int processIterations (
            char * arg,
            Info * progInfo )
```

This is the method which processes the number of iterations/generations for Genetic Algorithm and Differential Evolution from the line containing this information.

Failure: A value which does not convert to an interger greater than or equal to MINIMUM_ITERATIONS defined in General/Utilities.h

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

Attempt to convert the value in the given string arg to an integer and if it is not a positive integer greater than or equal to MINIMUM_ITERATIONS defined in General/Utilities.h tell the user and return failure. Otherwise return success.

4.8.3.16 processPAR()

```
int processPAR (
            char * arg,
            Info * progInfo )
```

This is the method which processes the pitch adjustment rate from the line containing this information.

Failure: A value which does not convert to a positive real number between [0, 1]

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

Attempt to convert the value in the given string arg to a double and if it is not a positive double tell the user and return failure. Otherwise return success.

4.8.3.17 processRanges()

```
int processRanges (
            char * arg,
            Info * progInfo )
```

This is the method which processes the range values for each equation from the line containing this information.

Failure: not enough or too many range values provided beyond the number of expected equations

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

< Initialize the variable to store the range value

< Initialize the variable to store the range counter of current position

< Store the number of equations locally

< Allocate an array to temporarily store the unparsed ranges for each equation

Tokenize the string on ','

While there are tokens

If our counter equals the number of equations we know we have received too many values, tell the user and return failure. Otherwise, store the unparsed string to the temp array increase the counter and get the next token.

If the counter is less than our number of equations it did not receive enough range values, tell the user and return failure.

Iterate through the temp array from 0 - (numEq - 1), resetting the range counter to 0, and tokenizing each unparsed string on ';'.

While there are tokens

If our counter equals the number of range values in a range we know we have received too many values, tell the user and return failure.

As the range value can be 0 in some cases, it will print a warning if it equals 0 after parsing the string to a double and let the user determine if there was some improper output. Then store the value to the corresponding location, increase the counter, and get the next token.

If the counter is less than the number of range values it did not receive enough range values, tell the user and return failure. Otherwise continue in the loop.

temp is no longer needed as the program parsed all of its values successfully so free temp and return success.

4.8.3.18 processVectors()

```
int processVectors (
            char * arg,
            Info * progInfo )
```

This is the method which processes the number of vectors from the line containing this information.

Failure: can't convert to a positive number

Parameters

arg	- The portion of the line containing the value to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

Attempt to convert the value in the given string arg to an integer and if it is not a positive integer tell the user and return failure. Otherwise return success.

4.9 General/Init.h File Reference

This is where the methods for the building the program info from the init file are defined. Referenced in PThread/main.c or Win32/main32.c.

```
#include "Utilities.h"
```

Functions

- int init (char *, Info *)

  This is the method which reads in the init file, and processes the information contained within into the progInfo Info struct using helper methods based on the flags which are set during processing.
- int processVectors (char *, Info *)

  This is the method which processes the number of vectors from the line containing this information.
- int processDimensions (char *, Info *)

  This is the method which processes the number of different dimensions from the line containing this information.
- int processDimList (char *, Info *)

  This is the method which processes the array of dimensions to test from the line containing this information.
- int processEquations (char *, Info *)

  This is the method which processes the number of equations to test from the line containing this information.
- int processRanges (char *, Info *)

  This is the method which processes the range values for each equation from the line containing this information.
- int processIterations (char *, Info *)

  This is the method which processes the number of iterations/generations for Genetic Algorithm and Differential Evolution from the line containing this information.
- int processBeta (char *arg, Info *progInfo)

This is the method which processes the beta/attractiveness factor from the line containing this information.

- int processGamma (char *arg, Info *progInfo)

  This is the method which processes the gamma/light absorption rate from the line containing this information.
- int processAlpha (char *arg, Info *progInfo)

  This is the method which processes the alpha/random scaling factor from the line containing this information.
- int processDampener (char *arg, Info *progInfo)

  This is the method which processes the velocity dampener value applied in General/PSO.c from the line containing this information.
- int processC1 (char *arg, Info *progInfo)

  This is the method which processes the personal best modification factor applied in General/PSO.c from the line containing this information.
- int processC2 (char *arg, Info *progInfo)

  This is the method which processes the global best modification factor applied in General/PSO.c from the line containing this information.
- int processHMCR (char *arg, Info *progInfo)

  This is the method which processes the Harmonic Memory Consideration Rate from the line containing this information.
- int processPAR (char *arg, Info *progInfo)

  This is the method which processes the pitch adjustment rate from the line containing this information.
- int processBandwidth (char *arg, Info *progInfo)

  This is the method which processes the bandwidth from the line containing this information.
- int checkTestType ()

  This is the method which takes user input to define the type of tests to run; either Genetic Algorithm or Differential Evolution.

4.9.1 Detailed Description

This is where the methods for the building the program info from the init file are defined. Referenced in PThread/main.c or Win32/main32.c.

Here we define and describe the methods which are called in PThread/main.c or Win32/main32.c and implemented in General/Init.c. Based on the input from the init file, we read in all the lines and process the information validating against expected input. All lines for both tests are expected to be present then the user selects which test they would like to run.

4.9.2 Function Documentation

4.9.2.1 checkTestType()

```
int checkTestType ( )
```

This is the method which takes user input to define the type of tests to run; either Genetic Algorithm or Differential Evolution.

Returns

0 for Differential Evolution, 1 for Genetic Algoriothm;

< Signify we are reading the test type input

< declare an array for the user input

Infinitely loop until we get a correct input from the user of 'Random', 'Local', or 'Iterative'

prompt the user for input then listen for input. if the user inputs 'Particle' return the value for PSO defined in the TestType enum within General/Utilities.h, otherwise if the user inputs 'Firefly' return Firefly from this same enum, or if the user inputs 'Harmonic' return Harmonic from this same enum. Otherwise tell the user it is an incorrect response and prompt the user for input again.

4.9.2.2 init()

```
int init (
            char * filename,
            Info * progInfo )
```

This is the method which reads in the init file, and processes the information contained within into the progInfo Info struct using helper methods based on the flags which are set during processing.

This iterates through the lines of the init file tokenizing the lines on '=' and then sets flags based on what line was just read in. It checks that all of the lines which require an order are processed in the right order and returns -1 on a failure. Failures: Not enough argument lines, Arguments are out of order, couldn't convert a value properly, incorrect number of matching arguments e.g. 3 dimensions expected only 2 provided in the dimension list.

Parameters

progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< Declare a char array of LINE_LENGTH defined in General/Utilities.h to store the current line being processed

< Initialize the flag representing the dampener line having been read to NotRead

< Initialize the flag representing the C1 line having been read to NotRead

< Initialize the flag representing the C2 (Roulette or Tournament) line having been read to NotRead

< Initialize the flag representing the beta/attractiveness factor line having been read to NotRead

< Initialize the flag representing the number of experiments line having been read to NotRead

< Initialize the flag representing the number of vectors line having been read to NotRead

< Initialize the flag representing the number of dimensions line having been read to NotRead

< Initialize the flag representing the different dimensions line having been read to NotRead

< Initialize the flag representing the number of equations line having been read to NotRead

< Initialize the flag representing the alpha line having been read to NotRead

< Initialize the flag representing the ranges line having been read to NotRead

< Initialize the flag representing the number of iterations line having been read to NotRead

< Initialize the flag representing the gamma line having been read to NotRead

< Initialize the flag representing the pitch adjustment line having been read to NotRead

< Initialize the flag representing the HMCR line having been read to NotRead

< Initialize the flag representing the bandwidth line having been read to NotRead

< Initialize a variable tracking the result of comparing to the expected dimensions line flag

< Initialize a variable tracking the result of comparing to the expected ranges line flag

< Stores the tokenized string from the line read in

while we still have lines to read

print the line being read

split the line on '='

while there are more tokens splitting on '='

If we are reading the number of vectors line, processVectors and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the number of dimensions line, processDimensions and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the different dimensions line, processDimList and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the number of equations line, processEquations and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the ranges line, processRanges and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the number of iterations line, processIterations and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the gamma line, processGamma and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the pitch adjustment line, processPAR and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the number of experiments line, processExperiments and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the alpha line, processAlpha and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the dampener line, processDampener and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the C1 line, processC1 and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the C2 line, processC2 and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the beta line, processBeta and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the HMCR line, processHMCR and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

If we are reading the bandwidth line, processBandwidth and if it failed close the file and return -1. Otherwise set the flag to Read and break the loop effectively moving to the next line being read.

This point is reached if the program is not currently reading any of the specified lines and is looking to see if it is reading one of the expected lines.

If we haven't read the number of vectors line already and it equals the signifier for the number of vectors line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the number of dimensions line already and it equals the signifier for the number of dimensions line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the different dimensions line already and it equals the signifier for the different dimensions line, and we have read the number of dimensions line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

Otherwise if we have come to the different dimensions line and have not read the number of dimensions in tell the user, close the file and return failure.

If we haven't read the number of Equations line already and it equals the signifier for the number of Equations line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the ranges line already and it equals the signifier for the ranges line, and we have read the number of equations line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

Otherwise if we have come to the ranges line and have not read the number of equations in tell the user, close the file and return failure.

If we haven't read the number of Generations/Iterations line already and it equals the signifier for the number of Generations/Iterations line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the Gamma line already and it equals the signifier for the Gamma line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the pitch adjustment line already and it equals the signifier for the pitch adjustment line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the number of experiments line already and it equals the signifier for the number of experiments line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the alpha line already and it equals the signifier for the alpha line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the dampener line already and it equals the signifier for the dampener line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the C1 line already and it equals the signifier for the C1 line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the C2 line already and it equals the signifier for the C2 line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the beta line already and it equals the signifier for the beta line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the HMCR line already and it equals the signifier for the HMCR line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

If we haven't read the bandwidth line already and it equals the signifier for the bandwidth line, set the flag to Reading, tokenize on '=' to get the value and continue the arg while loop.

This point is reached if none of the proper tags were found, exiting the arg while loop and effectively moving to the next line.

If one of the expected/required lines was not Read tell the user and close the file then return failure. Otherwise close the file ask the user for the test type and then return success.

4.9.2.3 processAlpha()

```
int processAlpha (
            char * arg,
            Info * progInfo )
```

This is the method which processes the alpha/random scaling factor from the line containing this information.

Failure: A value which does not convert to a positive real number

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

Attempt to convert the value in the given string arg to a double and if it is not a positive double tell the user and return failure. Otherwise return success.

4.9.2.4 processBandwidth()

```
int processBandwidth (
            char * arg,
            Info * progInfo )
```

This is the method which processes the bandwidth from the line containing this information.

Failure: A value which does not convert to a positive real number

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

Attempt to convert the value in the given string arg to a double and if it is not a positive double tell the user and return failure. Otherwise return success.

4.9.2.5 processBeta()

```
int processBeta (
            char * arg,
            Info * progInfo )
```

This is the method which processes the beta/attractiveness factor from the line containing this information.

Failure: A value which does not convert to a positive real number

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

Attempt to convert the value in the given string arg to a double and if it is not a positive double tell the user and return failure. Otherwise return success.

4.9.2.6 processC1()

```
int processC1 (
            char * arg,
            Info * progInfo )
```

This is the method which processes the personal best modification factor applied in General/PSO.c from the line containing this information.

Failure: A value which does not convert to a positive real number between (0, 2]

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

Attempt to convert the value in the given string arg to a double and if it is not a positive double tell the user and return failure. Otherwise return success.

4.9.2.7 processC2()

```
int processC2 (
            char * arg,
            Info * progInfo )
```

This is the method which processes the global best modification factor applied in General/PSO.c from the line containing this information.

Failure: A value which does not convert to a positive real number between (0, 2]

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

Attempt to convert the value in the given string arg to a double and if it is not a positive double tell the user and return failure. Otherwise return success.

4.9.2.8 processDampener()

```
int processDampener (
            char * arg,
            Info * progInfo )
```

This is the method which processes the velocity dampener value applied in General/PSO.c from the line containing this information.

Failure: A value which does not convert to a positive real number

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

Attempt to convert the value in the given string arg to a double and if it is not a positive double tell the user and return failure. Otherwise return success.

4.9.2.9 processDimensions()

```
int processDimensions (
            char * arg,
            Info * progInfo )
```

This is the method which processes the number of different dimensions from the line containing this information.

Failure: can't convert to a positive number

Parameters

arg	- The portion of the line containing the value to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

Attempt to convert the value in the given string arg to an integer and if it is not a positive integer tell the user and return failure. Otherwise allocate space for the array of different dimensions and return success.

4.9.2.10 processDimList()

```
int processDimList (
            char * arg,
            Info * progInfo )
```

This is the method which processes the array of dimensions to test from the line containing this information.

Failure: can't convert to a positive number, not enough or too many dimensions provided

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

Tokenize the current string arg on ',' and initialize variables for processing the line. dim will be the current dimension being processed in the tokenized string, dimCounter counts the number of dimensions which have been processed, not to exceed numDim which is the number of dimensions declared in the input file. NumDimensions is required to be defined before this method is called.

While there are dimensions to process, check that it hasn't exceeded the number of expected dimensions, otherwise tell the user and return failure. Attempt to parse the string to an integer and if it is not a positive integer tell the user and return failure. If successfully parsed set the processed dimension to the corresponding position in the array, increase the counter, and get the next token. Lastly, after all values have been parsed, make sure we have parsed the right number of values, if not, tell the user and return failure, otherwise return success.

If not enough dimensions were provided tell the user and return failure otherwise return success.

4.9.2.11 processEquations()

```
int processEquations (
            char * arg,
            Info * progInfo )
```

This is the method which processes the number of equations to test from the line containing this information.

Failure: can't convert to a positive number or is beyond the maximum number of equations in this testing suite.

Parameters

arg	- The portion of the line containing the value to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

Attempt to convert the value in the given string arg to an integer and if it is not a positive integer tell the user and return failure.

< Store the number of equations locally

if the number of equations defined is larger than the MAX_NUM_EQUATIONS defined in General/Utilities.h tell the user and return failure. Otherwise allocate the arrays to stor optimum values and ranges for each equation and return success.

4.9.2.12 processGamma()

```
int processGamma (
            char * arg,
            Info * progInfo )
```

This is the method which processes the gamma/light absorption rate from the line containing this information.

Failure: A value which does not convert to a positive real number

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

Attempt to convert the value in the given string arg to a double and if it is not a positive double tell the user and return failure. Otherwise return success.

4.9.2.13 processHMCR()

```
int processHMCR (
            char * arg,
            Info * progInfo )
```

This is the method which processes the Harmonic Memory Consideration Rate from the line containing this information.

Failure: A value which does not convert to a positive real number between [0, 1]

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

Attempt to convert the value in the given string arg to a double and if it is not a positive double tell the user and return failure. Otherwise return success.

4.9.2.14 processIterations()

```
int processIterations (
            char * arg,
            Info * progInfo )
```

This is the method which processes the number of iterations/generations for Genetic Algorithm and Differential Evolution from the line containing this information.

Failure: A value which does not convert to an interger greater than or equal to MINIMUM_ITERATIONS defined in General/Utilities.h

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

Attempt to convert the value in the given string arg to an integer and if it is not a positive integer greater than or equal to MINIMUM_ITERATIONS defined in General/Utilities.h tell the user and return failure. Otherwise return success.

4.9.2.15 processPAR()

```
int processPAR (
            char * arg,
            Info * progInfo )
```

This is the method which processes the pitch adjustment rate from the line containing this information.

Failure: A value which does not convert to a positive real number between [0, 1]

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

Attempt to convert the value in the given string arg to a double and if it is not a positive double tell the user and return failure. Otherwise return success.

4.9.2.16 processRanges()

```
int processRanges (
            char * arg,
            Info * progInfo )
```

This is the method which processes the range values for each equation from the line containing this information.

Failure: not enough or too many range values provided beyond the number of expected equations

Parameters

arg	- The portion of the line containing the values to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

< This is used to house the terminating character of the decimal parsing

< Initialize the variable to store the range value

< Initialize the variable to store the range counter of current position

< Store the number of equations locally

< Allocate an array to temporarily store the unparsed ranges for each equation

Tokenize the string on ','

While there are tokens

If our counter equals the number of equations we know we have received too many values, tell the user and return failure. Otherwise, store the unparsed string to the temp array increase the counter and get the next token.

If the counter is less than our number of equations it did not receive enough range values, tell the user and return failure.

Iterate through the temp array from 0 - (numEq - 1), resetting the range counter to 0, and tokenizing each unparsed string on ';'.

While there are tokens

If our counter equals the number of range values in a range we know we have received too many values, tell the user and return failure.

As the range value can be 0 in some cases, it will print a warning if it equals 0 after parsing the string to a double and let the user determine if there was some improper output. Then store the value to the corresponding location, increase the counter, and get the next token.

If the counter is less than the number of range values it did not receive enough range values, tell the user and return failure. Otherwise continue in the loop.

temp is no longer needed as the program parsed all of its values successfully so free temp and return success.

4.9.2.17 processVectors()

```
int processVectors (
            char * arg,
            Info * progInfo )
```

This is the method which processes the number of vectors from the line containing this information.

Failure: can't convert to a positive number

Parameters

arg	- The portion of the line containing the value to store
progInfo	- The reference to the Info struct for this series of tests defined in General/Utilities.h

Returns

0 for success, -1 on failure;

Attempt to convert the value in the given string arg to an integer and if it is not a positive integer tell the user and return failure. Otherwise return success.

4.10 General/MersenneMatrix.c File Reference

Implementation of the create matrix method defined in General/MersenneMatrix.h.

```
#include "Utilities.h"
#include "MersenneMatrix.h"
#include "m19937ar-cok.h"
```

Functions

- double ** createMatrix (EquationInfo info)

  Using the Mersenne Twister algorithm to develop a set of unique randomized values.
- double ** createVelocities (EquationInfo info)

  Using the Mersenne Twister algorithm to develop a set of unique randomized velocities.
- double genDblInRange (double min, double max)

  Using the Mersenne Twister algorithm to develop a scalar between 0 and 1, a unique randomized value between min and max is produced.
- int genNonNegInt (int max)

  Using the Mersenne Twister algorithm to develop a non-negative integer between 0 and max non-inclusive.

4.10.1 Detailed Description

Implementation of the create matrix method defined in General/MersenneMatrix.h.

In this file the matrix of randomized values with numVectors x dimToTest dimensions is allocated and created using the Mersenne Twister algorithm referenced in General/m19937ar-cok.h.

4.10.2 Function Documentation

4.10.2.1 createMatrix()

```
double ** createMatrix (
            EquationInfo info )
```

Using the Mersenne Twister algorithm to develop a set of unique randomized values.

Parameters

info	- EquationInfo struct

Returns

A matrix (2D-Array) of randomized double values

< Storing the value for number of vectors to test locally

< Storing the value for the number of dimensions in the vector locally

< Allocate the space for the return matrix of random values

< Declare and initialize the value which will be calculated and stored as one of the vector dimensions

< Store the range minimum from the read in file for this equation locally for later calculations. RANGE_MIN_POS is defined in General/Utilities.h

< Store the range maximum from the read in file for this equation locally for later calculations. RANGE_MAX_POS is defined in General/Utilities.h

Iterate through the matrix and allocate the space for the dimensions in each vector

Iterate through every matrix position of matrix[i][j] and generate a random double value in the range [minVal, max↩ Val]. Then finally store that value in matrix[i][j] and repeat till all values are set.

Return the randomly generated matrix.

4.10.2.2 createVelocities()

```
double ** createVelocities (
            EquationInfo info )
```

Using the Mersenne Twister algorithm to develop a set of unique randomized velocities.

Parameters

info	- EquationInfo struct

Returns

A random matrix of initial velocities for use in General/PSO.c

< Storing the value for number of vectors to test locally

< Storing the value for the number of dimensions in the vector locally

< Store the range minimum from the read in file for this equation locally for later calculations. RANGE_MIN_POS is defined in General/Utilities.h

< Store the range maximum from the read in file for this equation locally for later calculations. RANGE_MAX_POS is defined in General/Utilities.h

< calculate the range of possible values as defined in info

< Allocate the space for the return matrix of random values

Iterate through the matrix and allocate the space for the dimensions in each vector

Iterate through every matrix position of matrix[i][j] and generate a random double value in the range [0, .5 ∗ range]. Then finally store that value in matrix[i][j] and repeat till all values are set.

Return the randomly generated matrix.

4.10.2.3 genDblInRange()

```
double genDblInRange (
            double min,
            double max )
```

Using the Mersenne Twister algorithm to develop a scalar between 0 and 1, a unique randomized value between min and max is produced.

Parameters

min	- double representing the minimum value in the range
max	- double representing the maximum value in the range

Returns

A random double between min and max

Posix version of mutex locking and generating random number

4.10.2.4 genNonNegInt()

```
int genNonNegInt (
            int max )
```

Using the Mersenne Twister algorithm to develop a non-negative integer between 0 and max non-inclusive.

Parameters

max	- int representing the maximum value in the range

Returns

A random int between 0 and max

Posix version of mutex locking and generating random number

4.11 General/MersenneMatrix.h File Reference

Defines the method which creates the randomized matrix for use in all tests, for all equations in the program and generating a random double in the range.

```
#include "Utilities.h"
```

Functions

- double ** createMatrix (EquationInfo)

  Using the Mersenne Twister algorithm to develop a set of unique randomized values.
- double ** createVelocities (EquationInfo info)

  Using the Mersenne Twister algorithm to develop a set of unique randomized velocities.
- double genDblInRange (double, double)

  Using the Mersenne Twister algorithm to develop a scalar between 0 and 1, a unique randomized value between min and max is produced.
- int genNonNegInt (int)

  Using the Mersenne Twister algorithm to develop a non-negative integer between 0 and max non-inclusive.

4.11.1 Detailed Description

Defines the method which creates the randomized matrix for use in all tests, for all equations in the program and generating a random double in the range.

4.11.2 Function Documentation

4.11.2.1 createMatrix()

```
double** createMatrix (
            EquationInfo info )
```

Using the Mersenne Twister algorithm to develop a set of unique randomized values.

Parameters

info	- EquationInfo struct

Returns

A matrix (2D-Array) of randomized double values

< Storing the value for number of vectors to test locally

< Storing the value for the number of dimensions in the vector locally

< Allocate the space for the return matrix of random values

< Declare and initialize the value which will be calculated and stored as one of the vector dimensions

< Store the range minimum from the read in file for this equation locally for later calculations. RANGE_MIN_POS is defined in General/Utilities.h

< Store the range maximum from the read in file for this equation locally for later calculations. RANGE_MAX_POS is defined in General/Utilities.h

Iterate through the matrix and allocate the space for the dimensions in each vector

Iterate through every matrix position of matrix[i][j] and generate a random double value in the range [minVal, max↩ Val]. Then finally store that value in matrix[i][j] and repeat till all values are set.

Return the randomly generated matrix.

4.11.2.2 createVelocities()

```
double** createVelocities (
            EquationInfo info )
```

Using the Mersenne Twister algorithm to develop a set of unique randomized velocities.

Parameters

info	- EquationInfo struct

Returns

A random matrix of initial velocities for use in General/PSO.c

< Storing the value for number of vectors to test locally

< Storing the value for the number of dimensions in the vector locally

< Store the range minimum from the read in file for this equation locally for later calculations. RANGE_MIN_POS is defined in General/Utilities.h

< Store the range maximum from the read in file for this equation locally for later calculations. RANGE_MAX_POS is defined in General/Utilities.h

< calculate the range of possible values as defined in info

< Allocate the space for the return matrix of random values

Iterate through the matrix and allocate the space for the dimensions in each vector

Iterate through every matrix position of matrix[i][j] and generate a random double value in the range [0, .5 * range]. Then finally store that value in matrix[i][j] and repeat till all values are set.

Return the randomly generated matrix.

4.11.2.3 genDblInRange()

```
double genDblInRange (
            double min,
            double max )
```

Using the Mersenne Twister algorithm to develop a scalar between 0 and 1, a unique randomized value between min and max is produced.

Parameters

min	- double representing the minimum value in the range
max	- double representing the maximum value in the range

Returns

A random double between min and max

Posix version of mutex locking and generating random number

4.11.2.4 genNonNegInt()

```
int genNonNegInt (
            int max )
```

Using the Mersenne Twister algorithm to develop a non-negative integer between 0 and max non-inclusive.

Parameters

max	- int representing the maximum value in the range

Returns

A random int between 0 and max

Posix version of mutex locking and generating random number

4.12 General/PSO.c File Reference

This is where all methods defined in General/PSO.h are implemented.

```
#include "PSO.h"
#include "Utilities.h"
#include "MersenneMatrix.h"
```

Functions

- void * particleSwarmAlg (void *data)

  Responsible for initializing the algorithm and all data related to the algorithm and calling related functions to start and record results fo the algorithm.
- double calcPBestModifier (Particle *particles, double c1, int vecPos, int dimPos)

  calculates the personal best term when determining a new velocity
- double calcGBestModifier (Particle *particles, double c2, int vecPos, int dimPos)

  calculates the global best term when determining the new velocity
- void calcNewVelocity (Particle *particles, int dimensions, int position, double c1, double c2, double k)

  Calculates the velocity of the current dimension utilizing the old velocity and a term derived from the personal best particle and a term derived from the global best particle.
- void calcNewVector (Particle *particles, int dimensions, int position, const double *range)

  calculates a new vector using the current dimensional value and the velocity calculated in a prior step.
- void particleLoop (Particle *particles, int numParticles, int dimensions, double c1, double c2, double k, double *range, int equation)

  executes the loop which iterates through the population, calculates new velocities for each dimension, creates the new particle and evaluates it

4.12.1 Detailed Description

This is where all methods defined in General/PSO.h are implemented.

4.12.2 Function Documentation

4.12.2.1 calcGBestModifier()

```
double calcGBestModifier (
            Particle * particles,
            double c2,
            int vecPos,
            int dimPos )
```

calculates the global best term when determining the new velocity

Parameters

particles	- the struct being processed
c2	- the globalBest modification factor
vecPos	- the position of the particle in the population
dimPos	- the position of the dimension whose new velocity is being calculated

Returns

returns a double value representing the velocity towards the global best dimension

4.12.2.2 calcNewVector()

```
void calcNewVector (
            Particle * particles,
            int dimensions,
            int position,
            const double * range )
```

calculates a new vector using the current dimensional value and the velocity calculated in a prior step.

Parameters

particles	- the struct being processed
dimensions	- the number of dimensions per particle in the population
position	- the current particle position
range	- the range of acceptable values for the search space

4.12.2.3 calcNewVelocity()

```
void calcNewVelocity (
            Particle * particles,
            int dimensions,
            int position,
            double c1,
            double c2,
            double k )
```

Calculates the velocity of the current dimension utilizing the old velocity and a term derived from the personal best particle and a term derived from the global best particle.

Parameters

particles	- the struct being processed
dimensions	- the number of dimensions per particle in the population
position	- the current particle position
c1	- the personalBest modification factor
c2	- the globalBest modification factor
k	- the velocity dampening factor

4.12.2.4 calcPBestModifier()

```
double calcPBestModifier (
            Particle * particles,
            double c1,
            int vecPos,
            int dimPos )
```

calculates the personal best term when determining a new velocity

Parameters

particles	- the struct being processed
c1	- the personalBest modification factor
vecPos	- the position of the particle in the population
dimPos	- the position of the dimension whose new velocity is being calculated

Returns

returns a double value representing the velocity towards the personal best dimension

4.12.2.5 particleLoop()

```
void particleLoop (
            Particle * particles,
            int numParticles,
            int dimensions,
            double c1,
            double c2,
            double k,
            double * range,
            int equation )
```

executes the loop which iterates through the population, calculates new velocities for each dimension, creates the new particle and evaluates it

Parameters

particles	- The struct to be processed
numParticles	- The number of particles in the population
dimensions	- the number of dimensions per particle
c1	- the personalBest modification factor
c2	- the globalBest modification factor
k	- the velocity dampening factor
range	- the range of accepted values
equation	- the equation number for the objective function call

4.12.2.6 particleSwarmAlg()

```
void * particleSwarmAlg (
            void * data )
```

Responsible for initializing the algorithm and all data related to the algorithm and calling related functions to start and record results fo the algorithm.

Parameters

data	- void pointer to be converted to an EquationInfo struct

Returns

Nothing as it is a threaded function

< Declare the timespec struct storing the start time of the iterations

< Declare the timespec struct storing the end time of the iterations

< Declare the double storing the total runtime of all iterations in milliseconds

< Set the start time using a monotonic clock, meaning it will ignore if the system clock changes

run the particle swarm loop for this iteration

< Store the end time

< Calculate the total runtime by subtracting end time's seconds from the start time's seconds and converting to milliseconds and adding the end time's nanoseconds minus the start time's nanoseconds and converting to milliseconds

write the best and worst to a file

write the new population to a log file

free PSO

4.13 General/PSO.h File Reference

This is where all methods pertaining to the Particle Swarm Optimization Algorithm implementation are defined.

```
#include "Utilities.h"
```

Functions

- void * particleSwarmAlg (void *data)

  Responsible for initializing the algorithm and all data related to the algorithm and calling related functions to start and record results fo the algorithm.
- void particleLoop (Particle *particles, int numParticles, int dimensions, double c1, double c2, double k, double *range, int equation)

  executes the loop which iterates through the population, calculates new velocities for each dimension, creates the new particle and evaluates it
- double calcPBestModifier (Particle *particles, double c1, int vecPos, int dimPos)

  calculates the personal best term when determining a new velocity
- double calcGBestModifier (Particle *particles, double c2, int vecPos, int dimPos)

  calculates the global best term when determining the new velocity
- void calcNewVelocity (Particle *particles, int dimensions, int position, double c1, double c2, double k)

  Calculates the velocity of the current dimension utilizing the old velocity and a term derived from the personal best particle and a term derived from the global best particle.
- void calcNewVector (Particle *particles, int dimensions, int position, const double *range)

  calculates a new vector using the current dimensional value and the velocity calculated in a prior step.

4.13.1 Detailed Description

This is where all methods pertaining to the Particle Swarm Optimization Algorithm implementation are defined.

Genetic Algorithm is the process of taking parent vectors from the population, crossing over values between the two vectors, potentially mutating some or all of the dimensions of these vectors to create new populations. The new populations are then evaluated and the best results from each population are maintained for the next generation. The methods defined herein provide the functionality for this process.

4.13.2 Function Documentation

4.13.2.1 calcGBestModifier()

```
double calcGBestModifier (
            Particle * particles,
            double c2,
            int vecPos,
            int dimPos )
```

calculates the global best term when determining the new velocity

Parameters

particles	- the struct being processed
c2	- the globalBest modification factor
vecPos	- the position of the particle in the population
dimPos	- the position of the dimension whose new velocity is being calculated

Returns

returns a double value representing the velocity towards the global best dimension

4.13.2.2 calcNewVector()

```
void calcNewVector (
            Particle * particles,
            int dimensions,
            int position,
            const double * range )
```

calculates a new vector using the current dimensional value and the velocity calculated in a prior step.

Parameters

particles	- the struct being processed
dimensions	- the number of dimensions per particle in the population
position	- the current particle position
range	- the range of acceptable values for the search space

4.13.2.3 calcNewVelocity()

```
void calcNewVelocity (
```

```
            Particle * particles,
            int dimensions,
            int position,
            double c1,
            double c2,
            double k )
```

Calculates the velocity of the current dimension utilizing the old velocity and a term derived from the personal best particle and a term derived from the global best particle.

Parameters

particles	- the struct being processed
dimensions	- the number of dimensions per particle in the population
position	- the current particle position
c1	- the personalBest modification factor
c2	- the globalBest modification factor
k	- the velocity dampening factor

4.13.2.4 calcPBestModifier()

```
double calcPBestModifier (
            Particle * particles,
            double c1,
            int vecPos,
            int dimPos )
```

calculates the personal best term when determining a new velocity

Parameters

particles	- the struct being processed
c1	- the personalBest modification factor
vecPos	- the position of the particle in the population
dimPos	- the position of the dimension whose new velocity is being calculated

Returns

returns a double value representing the velocity towards the personal best dimension

4.13.2.5 particleLoop()

```
void particleLoop (
            Particle * particles,
            int numParticles,
```

```
            int dimensions,
            double c1,
            double c2,
            double k,
            double * range,
            int equation )
```

executes the loop which iterates through the population, calculates new velocities for each dimension, creates the new particle and evaluates it

Parameters

particles	- The struct to be processed
numParticles	- The number of particles in the population
dimensions	- the number of dimensions per particle
c1	- the personalBest modification factor
c2	- the globalBest modification factor
k	- the velocity dampening factor
range	- the range of accepted values
equation	- the equation number for the objective function call

4.13.2.6 particleSwarmAlg()

```
void* particleSwarmAlg (
            void * data )
```

Responsible for initializing the algorithm and all data related to the algorithm and calling related functions to start and record results fo the algorithm.

Parameters

data	- void pointer to be converted to an EquationInfo struct

Returns

Nothing as it is a threaded function

< Declare the timespec struct storing the start time of the iterations

< Declare the timespec struct storing the end time of the iterations

< Declare the double storing the total runtime of all iterations in milliseconds

< Set the start time using a monotonic clock, meaning it will ignore if the system clock changes

run the particle swarm loop for this iteration

< Store the end time

< Calculate the total runtime by subtracting end time's seconds from the start time's seconds and converting to milliseconds and adding the end time's nanoseconds minus the start time's nanoseconds and converting to milliseconds

write the best and worst to a file

write the new population to a log file

free PSO

4.14 General/Utilities.c File Reference

This is where all general purpose methods are implemented. All methods are defined in General/Utilities.h.

```
#include "Utilities.h"
#include "MersenneMatrix.h"
#include "HostCalls.h"
#include <fcntl.h>
```

Functions

- void lock ()
- void unlock ()
- void writeResultToFile (double bestFit, double worstFit, char *algorithm, int currIter, double time, EquationInfo info)

  This is the method which writes the results of the current Iteration to the designated result file.
- void writePopulationLogToFile (double **population, char *algorithm, int currIter, EquationInfo info)

  This is the method which writes the population log to a file tracking the changes in the population per iteration of an experiment.
- void printDArray (double *list, int size)

  This is the method which prints a double array to the console for debugging.
- void printIArray (int *list, int size)

  This is the method which prints an integer array to the console for debugging.
- void printMatrix (double **matrix, int numRows, int numCols)

  This is the method which prints a 2 dimensional array of doubles to the console for debugging.
- double ** allocateEmptyMatrix (int popSize, int dimensions)

  This is the method which allocates the space for an empty population of a certain size with a certain number of dimensions.
- void copyArray (const double *in, double *out, int size)
- void copyMatrix (double **in, double **out, int height, int width)

  This method is used to copy the values from one matrix into another so the original can be freed.
- void allocateHPop (HPop *pop, int popSize)

  This is the method which allocates the fitness array for the HPop struct.
- void createParticles (Particle *particles, int numVectors, int dimensions, EquationInfo info)

  Initializes a Particle struct for the algorithm implemented in General/PSO.c.
- double evaluateFitness (double *firefly, int dimensions, int equation)

  This method is responsible for executing the appropriate objective function on the given vector.
- void evaluatePop (double **pop, double *fitness, int popSize, int dimensions, int equation, int *objBestPos, int *objWorstPos, double *objBestFit, double *objWorstFit)

Takes an entire population, evaluates the fitness of the entire population and stores the best and worst fitness, and the positions in the population of these fitnesses for aiding the algorithm processing.

- void evalNewWorst (const double *fitness, int popSize, double newResult, int *objWorstPos, double *objWorstFit)

  Evaluates the population to determine the fitness and position of the worst vector in the population.
- void freeEquationInfo (EquationInfo *info)

  frees the EquationInfo struct and related information when a process thread is done
- void freeMatrix (double **matrix, int height)

  frees a matrix of the given size
- void freeInfo (Info *info)

  This frees the Info struct when the program has finished.
- void freeHPop (HPop *pop, int popSize)

  frees the HPop struct
- void freeFireflySwarm (FireflySwarm *pop, int popSize)

  frees the FireflySwarm struct
- void freeParticles (Particle *particles, int popSize)

  frees the Particle struct passed in

4.14.1 Detailed Description

This is where all general purpose methods are implemented. All methods are defined in General/Utilities.h.

4.14.2 Function Documentation

4.14.2.1 allocateEmptyMatrix()

```
double ** allocateEmptyMatrix (
            int popSize,
            int dimensions )
```

This is the method which allocates the space for an empty population of a certain size with a certain number of dimensions.

Parameters

popSize	- The number of rows to be allocated
dimensions	- The number of columns to be allocated per row

Returns

returns an empty matrix

allocate space for population size number of pointers

iterate for the population size and allocate space initializing the values to 0 for each pointer

return the pointer for your population

4.14.2.2 allocateHPop()

```
void allocateHPop (
            HPop * pop,
            int popSize )
```

This is the method which allocates the fitness array for the HPop struct.

Parameters

pop	- The HPop struct being initialized
popSize	- The size of the population being initialized

Returns

No return as it modifies the struct directly

the fitness array storing fitness values for the population is allocated

4.14.2.3 copyArray()

```
void copyArray (
            const double * in,
            double * out,
            int size )
```

iterate through the array being copied, and store the value at the current position to that position in the new array

4.14.2.4 copyMatrix()

```
void copyMatrix (
            double ** in,
            double ** out,
            int height,
            int width )
```

This method is used to copy the values from one matrix into another so the original can be freed.

Parameters

in	- the matrix to be copied
out	- the matrix to story the values being copied
height	- number of rows in the matrix being copied
width	- numbver of columns in the matrix being copied

Returns

No return as it modifies the pointer directly

iterate through the array being copied, and store the value at the current position to that position in the new array

4.14.2.5 createParticles()

```
void createParticles (
            Particle * particles,
            int numVectors,
            int dimensions,
            EquationInfo info )
```

Initializes a Particle struct for the algorithm implemented in General/PSO.c.

Parameters

particles	- the struct to be initialized
numVectors	- the number of particles to be stored in the population
dimensions	- the dimensions of the particles in the population
info	- the EquationInfo struct housing equation specific info

Create a random population matrix

create random initial velocities for the population

allocate an empty matrix to store the personal best particles

copy the initial matrix into the personal best matrix

allocate the fitness array

allocate the array storing personal best fitness

4.14.2.6 evalNewWorst()

```
void evalNewWorst (
            const double * fitness,
            int popSize,
            double newResult,
            int * objWorstPos,
            double * objWorstFit )
```

Evaluates the population to determine the fitness and position of the worst vector in the population.

Parameters

fitness	- the array of fitness values for the population
popSize	- the size of the population to be checked
newResult	- the fitness of the newly developed vector
objWorstPos	- pointer to the structs worst position
objWorstFit	- pointer to the structs worst fitness

< Initialize the worst fitness to the newResult

< Initialize the current fitness to 0

< Initialize the position of the worst fitness to 0

Iterate through the population and for each fitness value if it is greater than the worst fitness then set the worst fitness to this value and the position to the current iteration.

store the worst fitness to the address provided

store the position of the worst fitness to the address provided

4.14.2.7 evaluateFitness()

```
double evaluateFitness (
            double * firefly,
            int dimensions,
            int equation )
```

This method is responsible for executing the appropriate objective function on the given vector.

Parameters

firefly	- The given vector to be sent to the objective function
dimensions	- the number of dimensions in the vector
equation	- the objective function number to be run against

Returns

the fitness of the vector with regards to the objective function

< Declare a function pointer of type double to represent the current equation to be used

< Set the function to the appropriate function reference defined above

return the result of the calculation on the vector

4.14.2.8 evaluatePop()

```
void evaluatePop (
            double ** pop,
            double * fitness,
            int popSize,
            int dimensions,
            int equation,
            int * objBestPos,
            int * objWorstPos,
            double * objBestFit,
            double * objWorstFit )
```

Takes an entire population, evaluates the fitness of the entire population and stores the best and worst fitness, and the positions in the population of these fitnesses for aiding the algorithm processing.

Parameters

pop	- the population matrix for the struct being processed
fitness	- pointer to the fitness array of the processed population
popSize	- The number of rows to be evaluated
dimensions	- The number of columns in the row to be evaluated
equation	- the objective function number to be run against
objBestPos	- pointer to the structs best position
objWorstPos	- pointer to the structs worst position
objBestFit	- pointer to the structs best fitness
objWorstFit	- pointer to the structs worst fitness

< Initialize the best fitness to 0

< Initialize the worst fitness to 0

< Initialize the current fitness to 0

< Initialize the position of the best fitness to 0

< Initialize the position of the worst fitness to 0

Iterate through the population, evaluate the fitness of the current vector, store that fitness, and if its the first iteration store this as the best and worst fitness to be compared against. Otherwise for other iterations if it is less than the best fitness save this value and position as the best otherwise if it is greater than the worst fitness store this value and position as the worst.

store the best fitness to the address provided

store the position of the best fitness to the address provided

store the worst fitness to the address provided

store the position of the worst fitness to the address provided

4.14.2.9 freeEquationInfo()

```
void freeEquationInfo (
            EquationInfo * info )
```

frees the EquationInfo struct and related information when a process thread is done

Parameters

info	- The EquationInfo struct which houses the information for setting up the individual process threads

Returns

No return as it only frees memory

free the pointer storing the range for values to be used in the equation

4.14.2.10 freeFireflySwarm()

```
void freeFireflySwarm (
            FireflySwarm * pop,
            int popSize )
```

frees the FireflySwarm struct

Parameters

pop	- the struct to be freed
popSize	- the size of the population which is being freed

free the population matrix

free the fitness array

4.14.2.11 freeHPop()

```
void freeHPop (
            HPop * pop,
            int popSize )
```

frees the HPop struct

Parameters

pop	- the struct to be freed
popSize	- the size of the population which is being freed

free the population matrix

free the fitness array

4.14.2.12 freeInfo()

```
void freeInfo (
            Info * info )
```

This frees the Info struct when the program has finished.

Parameters

info	- The Info struct which houses the information for the whole program

Returns

No return as it only frees memory

free the pointer storing the dimensions which will be tested as part of the program

if ranges exists free the matrix of ranges

4.14.2.13 freeMatrix()

```
void freeMatrix (
            double ** matrix,
            int height )
```

frees a matrix of the given size

Parameters

matrix	- 2 dimensional array to be freed
height	- number of rows in the matrix

Returns

No return as it only frees memory

iterate through the rows of the matrix and free the pointers for each row

4.14.2.14 freeParticles()

```
void freeParticles (
            Particle * particles,
            int popSize )
```

frees the Particle struct passed in

Parameters

particles	- the struct to be freed
popSize	- the size of the population which is being freed

free the array of personal best fitness values

free the array of fitness values

free the population matrix

free the personal best population matrix

free the velocities matrix

4.14.2.15 lock()

```
void lock ( )
```

lock for writing to file POSIX

4.14.2.16 printDArray()

```
void printDArray (
            double * list,
            int size )
```

This is the method which prints a double array to the console for debugging.

Parameters

list	- The array to be printed
size	- size of the array

Returns

No return as it simply prints to console

add a character indicating the start of an array to the buffer

iterate through the array and add to the buffer the current value separated by commas

finally add a character to signify the end of the array to the buffer and flush the buffer to the console

4.14.2.17 printIArray()

```
void printIArray (
            int * list,
            int size )
```

This is the method which prints an integer array to the console for debugging.

Parameters

list	- The array to be printed
size	- size of the array

Returns

No return as it simply prints to console

add a character indicating the start of an array to the buffer

iterate through the array and add to the buffer the current value separated by commas

finally add a character to signify the end of the array to the buffer and flush the buffer to the console

4.14.2.18 printMatrix()

```
void printMatrix (
            double ** matrix,
```

```
int numRows,
int numCols )
```

This is the method which prints a 2 dimensional array of doubles to the console for debugging.

Parameters

matrix	- The 2 dimensional array to be printed
numRows	- the number of rows in matrix
numCols	- the number of columns per row in matrix

Returns

No return as it simply prints to console

iterate through each row of the matrix

add a character indicating the start of a row to the buffer

iterate through the array and add to the buffer the current value separated by commas

finally add a character to signify the end of the row to the buffer and flush the buffer to the console, then moving to the next row

4.14.2.19 unlock()

```
void unlock ( )
```

done writing to file POSIX

4.14.2.20 writePopulationLogToFile()

```
void writePopulationLogToFile (
            double ** population,
            char * algorithm,
            int currIter,
            EquationInfo info )
```

This is the method which writes the population log to a file tracking the changes in the population per iteration of an experiment.

Parameters

population	- The population resulting from the current iteration
algorithm	- the algorithm name being run, used for file name
currIter	- the current iteration being written to the file
info	- The EquationInfo struct which houses the information for setting up the process

Returns

No return as it simply prints to a file

< Declare the filename as a static char array of MAX_FILE_NAME_LENGTH defined in General/Utilities.h

< Declare the static char array of LINE_LENGTH size defined in General/Utilities.h which will store the values to be printed to the file

set the file name to our expected file

open the file in append mode

if there is an error opening the file tell the user and exit failure

for every row store the current test information to value to be written

write this to the file

Iterate through the dimensions of the row and write the value of the current dimension to the file.

close the file

4.14.2.21 writeResultToFile()

```
void writeResultToFile (
            double bestFit,
            double worstFit,
            char * algorithm,
            int currIter,
            double time,
            EquationInfo info )
```

This is the method which writes the results of the current Iteration to the designated result file.

Parameters

bestFit	- The best fitness of the current iteration
worstFit	- the worst fitness of the current iteration
algorithm	- the algorithm name being run, used for file name
currIter	- the current iteration being written to the file
time	- the time the iteration took to run
info	- The EquationInfo struct which houses the information for setting up the process

Returns

No return as it simply prints to a file

< Declare the filename as a static char array of MAX_FILE_NAME_LENGTH defined in General/Utilities.h

< Declare the static char array of LINE_LENGTH size defined in General/Utilities.h which will store the values to be printed to the file

set the file name to our expected file

open the file in append mode

if there is an error opening the file tell the user and exit failure

store the number of iterations locally

store the current test information to value to be written

write this to the file

close the file

4.15 General/Utilities.h File Reference

This is where all general purpose methods, constants, enums, and structs are declared.

```
#include "Equations.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <pthread.h>
```

Classes

- struct _Info
- struct _EquationInfo
- struct _Particle
- struct _FireflySwarm
- struct _HarmonicPop

Macros

- #define MINIMUM_ITERATIONS 30
- #define RANGE_SIZE 2
- #define RANGE_MIN_POS 0
- #define RANGE_MAX_POS 1
- #define LINE_LENGTH 10000
- #define MAX_NUM_EQUATIONS 18
- #define MAX_FILE_NAME_LEN 255
- #define DEFAULT_INIT_FILE "../General/init.txt"
- #define FILE_ARGUMENT 1
- #define NS_PER_MS 1000000
- #define MS_PER_SEC 1000

Typedefs

- typedef struct _Info Info
- typedef struct _EquationInfo EquationInfo
- typedef struct _Particle Particle
- typedef struct _FireflySwarm FireflySwarm
- typedef struct _HarmonicPop HPop

Enumerations

- enum EquationPosition {
  Schwefel, **DeJong**, **Rosenbrock**, **Rastgrin**,
  Griewangk, **SineEnvelope**, **StretchedWave**, **AckleyOne**,
  AckleyTwo, **EggHolder**, **Rana**, **Pathological**,
  Michalewicz, **MastersCosineWave**, **Quartic**, **Levy**,
  Step, **Alpine** }
- enum TestType { **ParticleSwarm**, **Firefly**, **Harmonic** }

Functions

- int genRandIntP (int modulo)
- double genRandRealP ()
- void writeResultToFile (double bestFit, double worstFit, char *algorithm, int currIter, double time, EquationInfo info)

  This is the method which writes the results of the current Iteration to the designated result file.
- void writePopulationLogToFile (double **population, char *algorithm, int currIter, EquationInfo info)

  This is the method which writes the population log to a file tracking the changes in the population per iteration of an experiment.
- void printDArray (double *list, int size)

  This is the method which prints a double array to the console for debugging.
- void printIArray (int *list, int size)

  This is the method which prints an integer array to the console for debugging.
- void printMatrix (double **matrix, int numRows, int numCols)

  This is the method which prints a 2 dimensional array of doubles to the console for debugging.
- double ** allocateEmptyMatrix (int, int)

  This is the method which allocates the space for an empty population of a certain size with a certain number of dimensions.
- void allocateHPop (HPop *pop, int popSize)

  This is the method which allocates the fitness array for the HPop struct.
- void createParticles (Particle *particles, int numVectors, int dimensions, EquationInfo info)

  Initializes a Particle struct for the algorithm implemented in General/PSO.c.
- void copyArray (const double *, double *, int)
- void copyMatrix (double **, double **, int, int)

  This method is used to copy the values from one matrix into another so the original can be freed.
- double evaluateFitness (double *firefly, int dimensions, int equation)

  This method is responsible for executing the appropriate objective function on the given vector.
- void evaluatePop (double **pop, double *fitness, int popSize, int dimensions, int equation, int *objBestPos, int *objWorstPos, double *objBestFit, double *objWorstFit)

  Takes an entire population, evaluates the fitness of the entire population and stores the best and worst fitness, and the positions in the population of these fitnesses for aiding the algorithm processing.
- void evalNewWorst (const double *fitness, int popSize, double newResult, int *objWorstPos, double *objWorstFit)

  Evaluates the population to determine the fitness and position of the worst vector in the population.
- void freeEquationInfo (EquationInfo *)

  frees the EquationInfo struct and related information when a process thread is done
- void freeInfo (Info *)

  This frees the Info struct when the program has finished.
- void freeMatrix (double **, int)

  frees a matrix of the given size
- void freeHPop (HPop *pop, int popSize)

  frees the HPop struct
- void freeFireflySwarm (FireflySwarm *pop, int popSize)

  frees the FireflySwarm struct
- void freeParticles (Particle *particles, int popSize)

  frees the Particle struct passed in

Variables

- pthread_mutex_t mutex

4.15.1 Detailed Description

This is where all general purpose methods, constants, enums, and structs are declared.

4.15.2 Macro Definition Documentation

4.15.2.1 DEFAULT_INIT_FILE

```
#define DEFAULT_INIT_FILE "../General/init.txt"
```

declare the constant representing the default input file location for the program

4.15.2.2 FILE_ARGUMENT

```
#define FILE_ARGUMENT 1
```

declare the constant for the

4.15.2.3 LINE_LENGTH

```
#define LINE_LENGTH 10000
```

declare a constant for reading in lines of a certain length for the input file

4.15.2.4 MAX_FILE_NAME_LEN

```
#define MAX_FILE_NAME_LEN 255
```

declare the constant for creating the filename string

4.15.2.5 MAX_NUM_EQUATIONS

```
#define MAX_NUM_EQUATIONS 18
```

declare the constant for the maximum number of equations being evaluated

4.15.2.6 MINIMUM_ITERATIONS

```
#define MINIMUM_ITERATIONS 30
```

declare the constant for the minimum number of iterations

4.15.2.7 MS_PER_SEC

```
#define MS_PER_SEC 1000
```

declare the constant to convert milliseconds to seconds

4.15.2.8 NS_PER_MS

```
#define NS_PER_MS 1000000
```

declare the constant for converting nanoseconds to milliseconds

4.15.2.9 RANGE_MAX_POS

```
#define RANGE_MAX_POS 1
```

declare a constant representing the location of the maximum value in the range

4.15.2.10 RANGE_MIN_POS

```
#define RANGE_MIN_POS 0
```

declare a constant representing the location of the minimum value in the range

4.15.2.11 RANGE_SIZE

```
#define RANGE_SIZE 2
```

declare the constant for the size of a range

4.15.3 Typedef Documentation

4.15.3.1 EquationInfo

```
typedef struct _EquationInfo EquationInfo
```

stores all of the information required by a single equation for a single set of dimensions to be processed in the selected test type. used in Win32/EquationsHandlers32.c and PThread/EquationHandlers.c

4.15.3.2 FireflySwarm

```
typedef struct _FireflySwarm FireflySwarm
```

Stores all information related to a population necessary for the Firefly Algorithm meta heuristics

4.15.3.3 HPop

```
typedef struct _HarmonicPop HPop
```

Stores all information related to a population necessary for the Harmonic Search meta heuristics

4.15.3.4 Info

```
typedef struct _Info Info
```

stores all of the necessary information for the program to run for all functions and all test types

4.15.3.5 Particle

```
typedef struct _Particle Particle
```

Stores all information related to a population necessary for the Particle Swarm meta heuristics

4.15.4 Enumeration Type Documentation

4.15.4.1 EquationPosition

```
enum EquationPosition
```

The enum which represents the position of the different functions utilized through the application. Referenced in General/EquationHandlers.c, and General/EquationHandlers32.c

4.15.4.2 TestType

```
enum TestType
```

enum representing the selected test type to be run as selected by the user in General/Init.c

4.15.5 Function Documentation

4.15.5.1 allocateEmptyMatrix()

```
double** allocateEmptyMatrix (
            int popSize,
            int dimensions )
```

This is the method which allocates the space for an empty population of a certain size with a certain number of dimensions.

Parameters

popSize	- The number of rows to be allocated
dimensions	- The number of columns to be allocated per row

Returns

returns an empty matrix

allocate space for population size number of pointers

iterate for the population size and allocate space initializing the values to 0 for each pointer

return the pointer for your population

4.15.5.2 allocateHPop()

```
void allocateHPop (
            HPop * pop,
            int popSize )
```

This is the method which allocates the fitness array for the HPop struct.

Parameters

pop	- The HPop struct being initialized
popSize	- The size of the population being initialized

Returns

No return as it modifies the struct directly

the fitness array storing fitness values for the population is allocated

4.15.5.3 copyArray()

```
void copyArray (
            const double * ,
            double * ,
            int  )
```

iterate through the array being copied, and store the value at the current position to that position in the new array

4.15.5.4 copyMatrix()

```
void copyMatrix (
            double ** in,
            double ** out,
            int height,
            int width )
```

This method is used to copy the values from one matrix into another so the original can be freed.

Parameters

in	- the matrix to be copied
out	- the matrix to story the values being copied
height	- number of rows in the matrix being copied
width	- numbver of columns in the matrix being copied

Returns

No return as it modifies the pointer directly

iterate through the array being copied, and store the value at the current position to that position in the new array

4.15.5.5 createParticles()

```
void createParticles (
            Particle * particles,
            int numVectors,
            int dimensions,
            EquationInfo info )
```

Initializes a Particle struct for the algorithm implemented in General/PSO.c.

Parameters

particles	- the struct to be initialized
numVectors	- the number of particles to be stored in the population
dimensions	- the dimensions of the particles in the population
info	- the EquationInfo struct housing equation specific info

Create a random population matrix

create random initial velocities for the population

allocate an empty matrix to store the personal best particles

copy the initial matrix into the personal best matrix

allocate the fitness array

allocate the array storing personal best fitness

4.15.5.6 evalNewWorst()

```
void evalNewWorst (
            const double * fitness,
            int popSize,
            double newResult,
            int * objWorstPos,
            double * objWorstFit )
```

Evaluates the population to determine the fitness and position of the worst vector in the population.

Parameters

fitness	- the array of fitness values for the population
popSize	- the size of the population to be checked
newResult	- the fitness of the newly developed vector
objWorstPos	- pointer to the structs worst position
objWorstFit	- pointer to the structs worst fitness

< Initialize the worst fitness to the newResult

< Initialize the current fitness to 0

< Initialize the position of the worst fitness to 0

Iterate through the population and for each fitness value if it is greater than the worst fitness then set the worst fitness to this value and the position to the current iteration.

store the worst fitness to the address provided

store the position of the worst fitness to the address provided

4.15.5.7 evaluateFitness()

```
double evaluateFitness (
            double * firefly,
            int dimensions,
            int equation )
```

This method is responsible for executing the appropriate objective function on the given vector.

Parameters

firefly	- The given vector to be sent to the objective function
dimensions	- the number of dimensions in the vector
equation	- the objective function number to be run against

Returns

the fitness of the vector with regards to the objective function

< Declare a function pointer of type double to represent the current equation to be used

< Set the function to the appropriate function reference defined above

return the result of the calculation on the vector

4.15.5.8 evaluatePop()

```
void evaluatePop (
            double ** pop,
```

```
            double * fitness,
            int popSize,
            int dimensions,
            int equation,
            int * objBestPos,
            int * objWorstPos,
            double * objBestFit,
            double * objWorstFit )
```

Takes an entire population, evaluates the fitness of the entire population and stores the best and worst fitness, and the positions in the population of these fitnesses for aiding the algorithm processing.

Parameters

pop	- the population matrix for the struct being processed
fitness	- pointer to the fitness array of the processed population
popSize	- The number of rows to be evaluated
dimensions	- The number of columns in the row to be evaluated
equation	- the objective function number to be run against
objBestPos	- pointer to the structs best position
objWorstPos	- pointer to the structs worst position
objBestFit	- pointer to the structs best fitness
objWorstFit	- pointer to the structs worst fitness

< Initialize the best fitness to 0

< Initialize the worst fitness to 0

< Initialize the current fitness to 0

< Initialize the position of the best fitness to 0

< Initialize the position of the worst fitness to 0

Iterate through the population, evaluate the fitness of the current vector, store that fitness, and if its the first iteration store this as the best and worst fitness to be compared against. Otherwise for other iterations if it is less than the best fitness save this value and position as the best otherwise if it is greater than the worst fitness store this value and position as the worst.

store the best fitness to the address provided

store the position of the best fitness to the address provided

store the worst fitness to the address provided

store the position of the worst fitness to the address provided

4.15.5.9 freeEquationInfo()

```
void freeEquationInfo (
            EquationInfo * info )
```

frees the EquationInfo struct and related information when a process thread is done

Parameters

info	- The EquationInfo struct which houses the information for setting up the individual process threads

Returns

No return as it only frees memory

free the pointer storing the range for values to be used in the equation

4.15.5.10 freeFireflySwarm()

```
void freeFireflySwarm (
            FireflySwarm * pop,
            int popSize )
```

frees the FireflySwarm struct

Parameters

pop	- the struct to be freed
popSize	- the size of the population which is being freed

free the population matrix

free the fitness array

4.15.5.11 freeHPop()

```
void freeHPop (
            HPop * pop,
            int popSize )
```

frees the HPop struct

Parameters

pop	- the struct to be freed
popSize	- the size of the population which is being freed

free the population matrix

free the fitness array

4.15.5.12 freeInfo()

```
void freeInfo (
            Info * info )
```

This frees the Info struct when the program has finished.

Parameters

info	- The Info struct which houses the information for the whole program

Returns

No return as it only frees memory

free the pointer storing the dimensions which will be tested as part of the program

if ranges exists free the matrix of ranges

4.15.5.13 freeMatrix()

```
void freeMatrix (
            double ** matrix,
            int height )
```

frees a matrix of the given size

Parameters

matrix	- 2 dimensional array to be freed
height	- number of rows in the matrix

Returns

No return as it only frees memory

iterate through the rows of the matrix and free the pointers for each row

4.15.5.14 freeParticles()

```
void freeParticles (
            Particle * particles,
            int popSize )
```

frees the Particle struct passed in

Parameters

particles	- the struct to be freed
popSize	- the size of the population which is being freed

free the array of personal best fitness values

free the array of fitness values

free the population matrix

free the personal best population matrix

free the velocities matrix

4.15.5.15 genRandIntP()

```
int genRandIntP (
            int modulo )
```

declare the POSIX method for threaded random integers < Initialize the result

accessing the random number generator

generate a random 32-bit integer number with the method defined in ../General/m19937ar-cok.h

done accessing the random number generator

return the result

4.15.5.16 genRandRealP()

```
double genRandRealP ( )
```

declare the POSIX method for threaded random real numbers < Initialize the result

accessing the random number generator

generate a random real number with the method defined in ../General/m19937ar-cok.h

done accessing the random number generator

return the result

4.15.5.17 printDArray()

```
void printDArray (
            double * list,
            int size )
```

This is the method which prints a double array to the console for debugging.

Parameters

list	- The array to be printed
size	- size of the array

Returns

No return as it simply prints to console

add a character indicating the start of an array to the buffer

iterate through the array and add to the buffer the current value separated by commas

finally add a character to signify the end of the array to the buffer and flush the buffer to the console

4.15.5.18 printIArray()

```
void printIArray (
            int * list,
            int size )
```

This is the method which prints an integer array to the console for debugging.

Parameters

list	- The array to be printed
size	- size of the array

Returns

No return as it simply prints to console

add a character indicating the start of an array to the buffer

iterate through the array and add to the buffer the current value separated by commas

finally add a character to signify the end of the array to the buffer and flush the buffer to the console

4.15.5.19 printMatrix()

```
void printMatrix (
            double ** matrix,
            int numRows,
            int numCols )
```

This is the method which prints a 2 dimensional array of doubles to the console for debugging.

Parameters

matrix	- The 2 dimensional array to be printed
numRows	- the number of rows in matrix
numCols	- the number of columns per row in matrix

Returns

No return as it simply prints to console

iterate through each row of the matrix

add a character indicating the start of a row to the buffer

iterate through the array and add to the buffer the current value separated by commas

finally add a character to signify the end of the row to the buffer and flush the buffer to the console, then moving to the next row

4.15.5.20 writePopulationLogToFile()

```
void writePopulationLogToFile (
            double ** population,
            char * algorithm,
            int currIter,
            EquationInfo info )
```

This is the method which writes the population log to a file tracking the changes in the population per iteration of an experiment.

Parameters

population	- The population resulting from the current iteration
algorithm	- the algorithm name being run, used for file name
currIter	- the current iteration being written to the file
info	- The EquationInfo struct which houses the information for setting up the process

Returns

No return as it simply prints to a file

< Declare the filename as a static char array of MAX_FILE_NAME_LENGTH defined in General/Utilities.h

< Declare the static char array of LINE_LENGTH size defined in General/Utilities.h which will store the values to be printed to the file

set the file name to our expected file

open the file in append mode

if there is an error opening the file tell the user and exit failure

for every row store the current test information to value to be written

write this to the file

Iterate through the dimensions of the row and write the value of the current dimension to the file.

close the file

4.15.5.21 writeResultToFile()

```
void writeResultToFile (
            double bestFit,
            double worstFit,
            char * algorithm,
            int currIter,
            double time,
            EquationInfo info )
```

This is the method which writes the results of the current Iteration to the designated result file.

Parameters

bestFit	- The best fitness of the current iteration
worstFit	- the worst fitness of the current iteration
algorithm	- the algorithm name being run, used for file name
currIter	- the current iteration being written to the file
time	- the time the iteration took to run
info	- The EquationInfo struct which houses the information for setting up the process

Returns

No return as it simply prints to a file

< Declare the filename as a static char array of MAX_FILE_NAME_LENGTH defined in General/Utilities.h

< Declare the static char array of LINE_LENGTH size defined in General/Utilities.h which will store the values to be printed to the file

set the file name to our expected file

open the file in append mode

if there is an error opening the file tell the user and exit failure

store the number of iterations locally

store the current test information to value to be written

write this to the file

close the file

4.15.6 Variable Documentation

4.15.6.1 mutex

```
pthread_mutex_t mutex
```

declare the POSIX mutex to be used when generating random numbers

4.16 Pthread/EquationHandlers.c File Reference

Handles the individual equation threads created in PThread/main.c and provides implementation for functions defined in General/EquationHandlers.h.

```
#include "../General/Utilities.h"
#include "../General/FA.h"
#include "../General/Harmonic.h"
#include "../General/PSO.h"
#include "../General/EquationHandlers.h"
#include <pthread.h>
```

Functions

- int runEquationsAsThreads (int equationPos, char *eqName, Info *data)

  Creates EquationInfo structs from the Info struct passed in and then creates threads for each of the dimensional tests.
- void * schwefelHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * deJongHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * rosenbrockHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * rastgrinHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * griewangkHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * sineEnvSineWaveHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * stretchVSineWaveHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * ackleyOneHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * ackleyTwoHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * eggHolderHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * ranaHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * pathologicalHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * michalewiczHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * mastersCosineWaveHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * quarticHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * levyHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * stepHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * alpineHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Variables

- const void * testTypeCalls [ ]

  An array of pointers to the TestType methods to be used by threads in runEquationsAsThreadss()

4.16.1 Detailed Description

Handles the individual equation threads created in PThread/main.c and provides implementation for functions defined in General/EquationHandlers.h.

Contains the implementations for the functions defined in General/EquationHandlers.h. Processes the Info struct provided by the threads in main.c and creates EquationInfo structs defined in General/Utilities.h to pass to the actual testType methods defined in General/TestTypes.h. Will create a number of threads for each dimension to be tested.

4.16.2 Function Documentation

4.16.2.1 ackleyOneHandler()

```
void* ackleyOneHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.16.2.2 ackleyTwoHandler()

```
void* ackleyTwoHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.16.2.3 alpineHandler()

```
void* alpineHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.16.2.4 deJongHandler()

```
void* deJongHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.16.2.5 eggHolderHandler()

```
void* eggHolderHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.16.2.6 griewangkHandler()

```
void* griewangkHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.16.2.7 levyHandler()

```
void* levyHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.16.2.8 mastersCosineWaveHandler()

```
void* mastersCosineWaveHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.16.2.9 michalewiczHandler()

```
void* michalewiczHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.16.2.10 pathologicalHandler()

```
void* pathologicalHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.16.2.11 quarticHandler()

```
void* quarticHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.16.2.12 ranaHandler()

```
void* ranaHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.16.2.13 rastgrinHandler()

```
void* rastgrinHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.16.2.14 rosenbrockHandler()

```
void* rosenbrockHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.16.2.15 runEquationsAsThreads()

```
int runEquationsAsThreads (
            int equationPos,
            char * eqName,
            Info * data )
```

Creates EquationInfo structs from the Info struct passed in and then creates threads for each of the dimensional tests.

This method takes in some basic information from the handler methods, equationPos and eqName, and then utilizes the Info struct passed to each equation's thread to create a specific EquationInfo struct for each dimensional test. Then it passes these structs to individual threads for each dimensional test.

Parameters

equationPos	- indicates the position within the various arrays of the Info struct for this equation's info
eqName	- Name of the equation being run used for file naming
data	- the Info struct which will be referenced to create the appropriate EquationInfo structs

Returns

0 on success, -1 on failures

< Store the number of different dimensions to be tested locally

< Create an array of EquationInfo structs defined in General/Utilities.h

Iterate through the number of different dimensions to be tested, create an EquationInfo struct to be passed to the equation function for each dimension, and fill it with relevant info for the dimension to be tested. Finally, store it to the array of structs.

< Create a temporary struct to be added to the array defined above

< Set the equationName to the value passed form the handler

< Set the equationNum to one greater than the position in a 0 based array

< Set the number of vectors for the test

< Set the specific dimensions for this test

< Set the number of iterations for this test

< Set the bandwidth for the harmonic test

< Set the pitch adjustment rate for the harmonic test

< Set the Harmony Memory Considering Rate for the harmonic test

< Set the beta value for attractiveness in the firefly algorithm test

< Set the absorption rate value for the firefly algorithm test

< Set the alpha value for the firefly algorithm test

< Set the personalBest term modifier c1 for PSO test

< Set the globalBest term modifier c2 for PSO test

< Set the dampening factor for PSO test

< Allocate space for the range of this equation

< Allocate space for the range of this equation

Loop through the range for this equation and copy the values over.

< Copy the value from the data Info struct to the range array

< Store the temp struct to the array

Create an array of pthread_t which is how threads are referenced in POSIX threads. Iterate for the number of experiments to be completed and within each iteration Iterate from 0 to (numDim - 1) and start a thread for each equation dimension from the constant defined above. Each thread will be passed a reference to the EquationInfo struct for each dimension. If it fails, print the error message, free the struct, and return failure.

< Set the currentExperiment number for this test

Wait for all the threads to finish, and if there was an error, print the last error and return failure.

This equation has finished for all dimensions and experiments, free the threads array, free the array of EquationInfo structs and return success.

4.16.2.16 schwefelHandler()

```
void* schwefelHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.16.2.17 sineEnvSineWaveHandler()

```
void* sineEnvSineWaveHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.16.2.18 stepHandler()

```
void* stepHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.16.2.19 stretchVSineWaveHandler()

```
void* stretchVSineWaveHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.16.3 Variable Documentation

4.16.3.1 testTypeCalls

```
const void* testTypeCalls[]
```

Initial value:

```
= {
        &particleSwarmAlg,
        &fireflyAlg,
        &harmonicTest
}
```

An array of pointers to the TestType methods to be used by threads in runEquationsAsThreadss()

This is used as an easy way to select the correct test type from the user input when creating the threads to start processing each equation by dimension in different threads.

4.17 Pthread/main.c File Reference

The entry point for Unix/Linux machines when executing this program.

```
#include "../General/Utilities.h"
#include "../General/Init.h"
#include "../General/EquationHandlers.h"
#include "../General/m19937ar-cok.h"
#include <pthread.h>
```

Functions

- int main (int argc, char ∗argv[ ])

  Entry point for the program on Unix/Linux machines. Failures: Too many arguments, reading the init file failed, creating threads failed, and error returned from the threads.

Variables

- const void ∗ equationHandlers [ ]

  An array of pointers to the equation handler methods to be used by threads in main()

4.17.1 Detailed Description

The entry point for Unix/Linux machines when executing this program.

Here, the arguments passed to the program are processed to determine the init file name, the init file is processed to determine the parameters for the program to utilize,

4.17.2 Function Documentation

4.17.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Entry point for the program on Unix/Linux machines. Failures: Too many arguments, reading the init file failed, creating threads failed, and error returned from the threads.

Entry point for the program on Win32 machines. Failures: Too many arguments, reading the init file failed, creating threads failed, and error returned from the threads.

Parameters

argc	- This represents the number of arguments provided to the program.
argv	- This is the array of arguments provided to the program. argv[0] = program name, argv[1] = init filepath

Returns

0 on success, -1 on failures.

< Filepath for the init file required for this program

< Info struct that houses the data read in from the init file. Struct defined in General/Utilities.h

Check the arguments passed to the program to determine what init file to use. If no arguments were provided, argc = 1 and use the DEFAULT_INIT_FILE constant. Else if one argument was provided, treat the argument at FILE_ARGUMENT as the filename which, if incorrect, will be handled later. Otherwise too many arguments were provided and the program will print an error message and return failure.

< DEFAULT_INIT_FILE is defined in General/Utilities.h

< FILE_ARGUMENT is defined in General/Utilities.h

Create the mutex to be used when generating random numberss and writing to files

< generating a value to seed the Mersenne Twister algorithm

Seed the Mersenne Twister algorithm defined in General/m19937ar-cok.h

Once the filename is set, process the init file, passing in the Info struct and the file name to the init function of General/Init.h. The init method will print the subsequent errors and if failed return a value less than 0. Free progInfo and return failure.

< creating a local variable for the number of equations to avoid accessing the struct multiple times

Provide user output to verify that init file info was properly read in. Print the list of ranges, dimensions which will be tested, and the optimum values.

Create an array of pthread_t which is how threads are referenced in POSIX threads. Iterate from 0 to (numExp - 1) and set the current experiment number to our iteration. Iterate from 0 to (numEq - 1) and start a thread for each equation handler from the constant defined above. Each thread will be passed a reference to the progInfo struct. If it fails, print the error message, free progInfo and threads, and return failure.

Iterate through the threads array and join all of the threads waiting for them to finish before finalising the program. If it fails, print the error message, free progInfo and threads, and return failure.

The program has finished. Free progInfo and threads, then destroy the mutex then return success.

4.17.3 Variable Documentation

4.17.3.1 equationHandlers

```
const void* equationHandlers[]
```

Initial value:

```
= {
        &schwefelHandler,
        &deJongHandler,
        &rosenbrockHandler,
        &rastgrinHandler,
        &griewangkHandler,
        &sineEnvSineWaveHandler,
        &stretchVSineWaveHandler,
        &ackleyOneHandler,
        &ackleyTwoHandler,
        &eggHolderHandler,
        &ranaHandler,
        &pathologicalHandler,
        &michalewiczHandler,
        &mastersCosineWaveHandler,
        &quarticHandler,
        &levyHandler,
        &stepHandler,
        &alpineHandler
}
```

An array of pointers to the equation handler methods to be used by threads in main()

This is used as an easy way to iterate through the equation handlers when creating the threads to start processing each equation in different threads.

4.18 Pthread/UtilP.c File Reference

contains the implementations for the methods declared in General/Utilities.h which produce random numbers using a mutex in POSIX format.

```
#include "../General/Utilities.h"
#include "../General/m19937ar-cok.h"
#include <pthread.h>
```

Functions

- int genRandIntP (int modulo)
- double genRandRealP ()

4.18.1 Detailed Description

contains the implementations for the methods declared in General/Utilities.h which produce random numbers using a mutex in POSIX format.

4.18.2 Function Documentation

4.18.2.1 genRandIntP()

```
int genRandIntP (
            int modulo )
```

declare the POSIX method for threaded random integers < Initialize the result

accessing the random number generator

generate a random 32-bit integer number with the method defined in ../General/m19937ar-cok.h

done accessing the random number generator

return the result

4.18.2.2 genRandRealP()

```
double genRandRealP ( )
```

declare the POSIX method for threaded random real numbers < Initialize the result

accessing the random number generator

generate a random real number with the method defined in ../General/m19937ar-cok.h

done accessing the random number generator

return the result

4.19 Win32/EquationHandlers32.c File Reference

Handles the individual equation threads created in Win32/main32.c and provides implementation for functions defined in General/EquationHandlers.h.

```
#include "../General/Utilities.h"
#include "../General/FA.h"
#include "../General/Harmonic.h"
#include "../General/PSO.h"
#include "../General/EquationHandlers.h"
#include <process.h>
#include <windows.h>
```

Functions

- int runEquationsAsThreads (int equationPos, char *eqName, Info *data)

  Creates EquationInfo structs from the Info struct passed in and then creates threads for each of the dimensional tests.
- void * schwefelHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * deJongHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * rosenbrockHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * rastgrinHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * griewangkHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * sineEnvSineWaveHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * stretchVSineWaveHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * ackleyOneHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * ackleyTwoHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * eggHolderHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * ranaHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * pathologicalHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * michalewiczHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * mastersCosineWaveHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * quarticHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * levyHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * stepHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()
- void * alpineHandler (void *info)

  This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Variables

- const void * testTypeCalls [ ]

  An array of pointers to the TestType methods to be used by threads in runEquationsAsThreadss()

4.19.1 Detailed Description

Handles the individual equation threads created in Win32/main32.c and provides implementation for functions defined in General/EquationHandlers.h.

Contains the implementations for the functions defined in General/EquationHandlers.h. Processes the Info struct provided by the threads in main32.c and creates EquationInfo structs defined in General/Utilities.h to pass to the actual testType methods defined in General/TestTypes.h. Will create a number of threads for each dimension to be tested.

4.19.2 Function Documentation

4.19.2.1 ackleyOneHandler()

```
void* ackleyOneHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.19.2.2 ackleyTwoHandler()

```
void* ackleyTwoHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.19.2.3 alpineHandler()

```
void* alpineHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.19.2.4 deJongHandler()

```
void* deJongHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.19.2.5 eggHolderHandler()

```
void* eggHolderHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.19.2.6 griewangkHandler()

```
void* griewangkHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.19.2.7 levyHandler()

```
void* levyHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.19.2.8 mastersCosineWaveHandler()

```
void* mastersCosineWaveHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.19.2.9 michalewiczHandler()

```
void* michalewiczHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.19.2.10 pathologicalHandler()

```
void* pathologicalHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.19.2.11 quarticHandler()

```
void* quarticHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.19.2.12 ranaHandler()

```
void* ranaHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.19.2.13 rastgrinHandler()

```
void* rastgrinHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.19.2.14 rosenbrockHandler()

```
void* rosenbrockHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.19.2.15 runEquationsAsThreads()

```
int runEquationsAsThreads (
            int equationPos,
            char * eqName,
            Info * data )
```

Creates EquationInfo structs from the Info struct passed in and then creates threads for each of the dimensional tests.

This method takes in some basic information from the handler methods, equationPos and eqName, and then utilizes the Info struct passed to each equation's thread to create a specific EquationInfo struct for each dimensional test. Then it passes these structs to individual threads for each dimensional test.

Parameters

equationPos	- indicates the position within the various arrays of the Info struct for this equation's info
eqName	- Name of the equation being run used for file naming
data	- the Info struct which will be referenced to create the appropriate EquationInfo structs

Returns

0 on success, -1 on failures

< Store the number of different dimensions to be tested locally

< Create an array of EquationInfo structs defined in General/Utilities.h

Iterate through the number of different dimensions to be tested, create an EquationInfo struct to be passed to the equation function for each dimension, and fill it with relevant info for the dimension to be tested. Finally, store it to the array of structs.

< Create a temporary struct to be added to the array defined above

< Set the equationName to the value passed form the handler

< Set the equationNum to done greater than the position in a 0 based array

< Set the number of vectors for the test

< Set the specific dimensions for this test

< Set the number of iterations for this test

< Set the bandwidth for the harmonic test

< Set the pitch adjustment rate for the harmonic test

< Set the Harmony Memory Considering Rate for the harmonic test

< Set the beta value for attractiveness in the firefly algorithm test

< Set the absorption rate value for the firefly algorithm test

< Set the alpha value for the firefly algorithm test

< Set the personalBest term modifier c1 for PSO test

< Set the globalBest term modifier c2 for PSO test

< Set the dampening factor for PSO test

< Allocate space for the range of this equation

Loop through the range for this equation and copy the values over.

< Copy the value from the data Info struct to the range array

< Store the temp struct to the array

Create an array of Handles which is how threads are referenced in Win32 threads. Iterate for the number of experiments to be completed and within each iteration Iterate from 0 to (numDim - 1) and start a thread for each equation dimension from the constant defined above. Each thread will be passed a reference to the EquationInfo struct for each dimension. If it fails, print the error message, free the struct, and return failure.

< Set the currentExperiment number for this test

Wait for all the threads to finish, and if there was an error, print the last error and return failure.

This equation has finished for all dimensions and experiments, free the threads array, free the array of EquationInfo structs and return success.

4.19.2.16 schwefelHandler()

```
void* schwefelHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.19.2.17 sineEnvSineWaveHandler()

```
void* sineEnvSineWaveHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.19.2.18 stepHandler()

```
void* stepHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.19.2.19 stretchVSineWaveHandler()

```
void* stretchVSineWaveHandler (
            void * info )
```

This is the method which sets up some basic information before passing that info to runEquationsAsThreads()

Parameters

info	- The reference to the Info struct for this series of tests

Returns

Threaded function so no return.

< Set the position where the equation specific information can be found in the Info struct. References Enum EquationsPosition defined in General/Utilities.h

< Cast the passed in struct back to an Info struct for processing

< Set the name of the equation to be run for file output in writeResultsToFile defined in General/Utilities.h

pass in the variables defined above to the runEquationsAsThreads method and if it returns less than 0 it failed and exit failure

4.19.3 Variable Documentation

4.19.3.1 testTypeCalls

```
const void* testTypeCalls[]
```

Initial value:

```
= {
        &particleSwarmAlg,
        &fireflyAlg,
        &harmonicTest
}
```

An array of pointers to the TestType methods to be used by threads in runEquationsAsThreadss()

This is used as an easy way to select the correct test type from the user input when creating the threads to start processing each equation by dimension in different threads.

4.20 Win32/main32.c File Reference

The entry point for Win32 machines when executing this program.

```
#include "../General/Utilities.h"
#include "../General/Init.h"
#include "../General/EquationHandlers.h"
#include <process.h>
#include <windows.h>
```

Functions

- int main (int argc, char *argv[])

Variables

- const void * equationHandlers [ ]

  An array of pointers to the equation handler methods to be used by threads in main()

4.20.1 Detailed Description

The entry point for Win32 machines when executing this program.

Here, the arguments passed to the program are processed to determine the init file name, the init file is processed to determine the parameters for the program to utilize,

4.20.2 Function Documentation

4.20.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

< Filepath for the init file required for this program

< Info struct that houses the data read in from the init file. Struct defined in General/Utilities.h

Check the arguments passed to the program to determine what init file to use. If no arguments were provided, argc = 1 and use the DEFAULT_INIT_FILE constant. Else if one argument was provided, treat the argument at FILE_ARGUMENT as the filename which, if incorrect, will be handled later. Otherwise too many arguments were provided and the program will print an error message and return failure.

< DEFAULT_INIT_FILE is defined in General/Utilities.h

< FILE_ARGUMENT is defined in General/Utilities.h

Create the mutex to be used when generating random numberss and writing to files

default security attributes

initially not owned

unnamed

< generating a value to seed the Mersenne Twister algorithm

Seed the Mersenne Twister algorithm defined in General/m19937ar-cok.h

Once the filename is set, process the init file, passing in the Info struct and the file name to the init function of General/Init.h. The init method will print the subsequent errors and if failed return a value less than 0. Free progInfo and return failure.

< creating a local variable for the number of experiments to avoid accessing the struct multiple times

< creating a local variable for the number of equations to avoid accessing the struct multiple times

Provide user output to verify that init file info was properly read in. Print the list of ranges, dimensions which will be tested, and the optimum values.

Create an array of Handles which is how threads are referenced in Win32 threads. Iterate from 0 to (numExp - 1) and set the current experiment number to our iteration. Iterate from 0 to (numEq - 1) and start a thread for each equation handler from the constant defined above. Each thread will be passed a reference to the progInfo struct. If it fails, print the error message, free progInfo, and return failure.

Wait for all the threads to finish, and if there was an error, print the last error and return failure.

The program has finished. Free progInfo and threads, and close the mutex handle then return success.

4.20.3 Variable Documentation

4.20.3.1 equationHandlers

```
const void* equationHandlers[]
```

Initial value:

```
={
        &schwefelHandler,
        &deJongHandler,
        &rosenbrockHandler,
        &rastgrinHandler,
        &griewangkHandler,
        &sineEnvSineWaveHandler,
        &stretchVSineWaveHandler,
        &ackleyOneHandler,
        &ackleyTwoHandler,
        &eggHolderHandler,
        &ranaHandler,
        &pathologicalHandler,
        &michalewiczHandler,
        &mastersCosineWaveHandler,
        &quarticHandler,
        &levyHandler,
        &stepHandler,
        &alpineHandler
}
```

An array of pointers to the equation handler methods to be used by threads in main()

This is used as an easy way to iterate through the equation handlers when creating the threads to start processing each equation in different threads.

4.21 Win32/Util32.c File Reference

contains the implementations for the methods declared in General/Utilities.h which produce random numbers using a mutex in WIN332 format.

```
#include "../General/Utilities.h"
#include "../General/m19937ar-cok.h"
```

Functions

- int genRandInt32 (int modulo)
- double genRandReal32 ()

4.21.1 Detailed Description

contains the implementations for the methods declared in General/Utilities.h which produce random numbers using a mutex in WIN332 format.

4.21.2 Function Documentation

4.21.2.1 genRandInt32()

```
int genRandInt32 (
            int modulo )
```

< Initialize the result

accessing the random number generator

generate a random 32-bit integer number with the method defined in ../General/m19937ar-cok.h

done accessing the random number generator

return the result

4.21.2.2 genRandReal32()

```
double genRandReal32 ( )
```

< Initialize the result

accessing the random number generator

generate a random real number with the method defined in ../General/m19937ar-cok.h

done accessing the random number generator

return the result

Index

_EquationInfo, 5
_FireflySwarm, 6
_HarmonicPop, 6
_Info, 7
_Particle, 7

ackleyOneHandler
    EquationHandlers.c, 118
    EquationHandlers.h, 10
    EquationHandlers32.c, 134
ackleyOneHost
    Equations.c, 25
    Equations.h, 32
ackleyTwoHandler
    EquationHandlers.c, 118
    EquationHandlers.h, 11
    EquationHandlers32.c, 134
ackleyTwoHost
    Equations.c, 25
    Equations.h, 32
addVector
    FA.h, 38
allocateEmptyMatrix
    Utilities.c, 90
    Utilities.h, 105
allocateHPop
    Utilities.c, 90
    Utilities.h, 106
alpineHandler
    EquationHandlers.c, 119
    EquationHandlers.h, 11
    EquationHandlers32.c, 135
alpineHost
    Equations.c, 25
    Equations.h, 32

calcAttractedVector
    FA.h, 39
calcAttractiveness
    FA.h, 39
calcDistanceSquared
    FA.h, 40
calcGBestModifier
    PSO.c, 82
    PSO.h, 86
calcNewVector
    PSO.c, 82
    PSO.h, 86
calcNewVelocity
    PSO.c, 83
    PSO.h, 86
calcPBestModifier
    PSO.c, 83
    PSO.h, 87
checkTestType
    Init.c, 51
    Init.h, 64
copyArray
    Utilities.c, 91
    Utilities.h, 106
copyMatrix
    Utilities.c, 91
    Utilities.h, 106
createMatrix
    MersenneMatrix.c, 76
    MersenneMatrix.h, 79
createParticles
    Utilities.c, 91
    Utilities.h, 107
createVelocities
    MersenneMatrix.c, 77
    MersenneMatrix.h, 79

DEFAULT_INIT_FILE
    Utilities.h, 103
deJongHandler
    EquationHandlers.c, 119
    EquationHandlers.h, 12
    EquationHandlers32.c, 135
deJongHost
    Equations.c, 26
    Equations.h, 32

eggHolderHandler
    EquationHandlers.c, 120
    EquationHandlers.h, 13
    EquationHandlers32.c, 136
eggHolderHost
    Equations.c, 26
    Equations.h, 33
equationHandlers
    main.c, 130
    main32.c, 146
EquationHandlers.c
    ackleyOneHandler, 118
    ackleyTwoHandler, 118
    alpineHandler, 119
    deJongHandler, 119
    eggHolderHandler, 120
    griewangkHandler, 120

levyHandler, 121
mastersCosineWaveHandler, 121
michalewiczHandler, 122
pathologicalHandler, 122
quarticHandler, 123
ranaHandler, 123
rastgrinHandler, 124
rosenbrockHandler, 124
runEquationsAsThreads, 125
schwefelHandler, 126
sineEnvSineWaveHandler, 127
stepHandler, 127
stretchVSineWaveHandler, 128
testTypeCalls, 128
EquationHandlers.h
ackleyOneHandler, 10
ackleyTwoHandler, 11
alpineHandler, 11
deJongHandler, 12
eggHolderHandler, 13
griewangkHandler, 13
levyHandler, 14
mastersCosineWaveHandler, 15
michalewiczHandler, 15
pathologicalHandler, 16
quarticHandler, 17
ranaHandler, 17
rastgrinHandler, 18
rosenbrockHandler, 19
runEquationsAsThreads, 19
schwefelHandler, 21
sineEnvSineWaveHandler, 22
stepHandler, 23
stretchVSineWaveHandler, 23
EquationHandlers32.c
ackleyOneHandler, 134
ackleyTwoHandler, 134
alpineHandler, 135
deJongHandler, 135
eggHolderHandler, 136
griewangkHandler, 136
levyHandler, 137
mastersCosineWaveHandler, 137
michalewiczHandler, 138
pathologicalHandler, 138
quarticHandler, 139
ranaHandler, 139
rastgrinHandler, 140
rosenbrockHandler, 140
runEquationsAsThreads, 141
schwefelHandler, 142
sineEnvSineWaveHandler, 143
stepHandler, 143
stretchVSineWaveHandler, 144
testTypeCalls, 144
equationHostCalls
HostCalls.h, 49
EquationInfo
Utilities.h, 104
EquationPosition
Utilities.h, 105
Equations.c
ackleyOneHost, 25
ackleyTwoHost, 25
alpineHost, 25
deJongHost, 26
eggHolderHost, 26
griewangkHost, 26
levyHost, 27
mastersCosineWaveHost, 27
michalewiczHost, 27
pathologicalHost, 28
quarticHost, 28
ranaHost, 28
rastgrinHost, 29
rosenbrockHost, 29
schwefelHost, 29
sineEnvSineWaveHost, 30
stepHost, 30
stretchVSineWaveHost, 30
Equations.h
ackleyOneHost, 32
ackleyTwoHost, 32
alpineHost, 32
deJongHost, 32
eggHolderHost, 33
griewangkHost, 33
levyHost, 33
mastersCosineWaveHost, 34
michalewiczHost, 34
pathologicalHost, 34
quarticHost, 35
ranaHost, 35
rastgrinHost, 35
rosenbrockHost, 36
schwefelHost, 36
sineEnvSineWaveHost, 36
stepHost, 37
stretchVSineWaveHost, 37
evalNewWorst
Utilities.c, 92
Utilities.h, 107
evaluateFitness
Utilities.c, 93
Utilities.h, 108
evaluatePop
Utilities.c, 93
Utilities.h, 108

FA.h
addVector, 38
calcAttractedVector, 39
calcAttractiveness, 39
calcDistanceSquared, 40
fireflyAlg, 40
fireflyLoop, 41
lightIntensity, 42

moveFirefliesLoop, 42
newBest, 43
FILE_ARGUMENT
Utilities.h, 103
fireflyAlg
FA.h, 40
fireflyLoop
FA.h, 41
FireflySwarm
Utilities.h, 104
freeEquationInfo
Utilities.c, 94
Utilities.h, 109
freeFireflySwarm
Utilities.c, 94
Utilities.h, 110
freeHPop
Utilities.c, 95
Utilities.h, 110
freeInfo
Utilities.c, 95
Utilities.h, 110
freeMatrix
Utilities.c, 96
Utilities.h, 112
freeParticles
Utilities.c, 96
Utilities.h, 112

genDblInRange
MersenneMatrix.c, 77
MersenneMatrix.h, 80
General/EquationHandlers.h, 9
General/Equations.c, 24
General/Equations.h, 31
General/FA.h, 37
General/Harmonic.c, 43
General/Harmonic.h, 46
General/HostCalls.h, 49
General/Init.c, 50
General/Init.h, 63
General/MersenneMatrix.c, 76
General/MersenneMatrix.h, 78
General/PSO.c, 81
General/PSO.h, 85
General/Utilities.c, 89
General/Utilities.h, 101
genNonNegInt
MersenneMatrix.c, 78
MersenneMatrix.h, 81
genRandInt32
Util32.c, 147
genRandIntP
Utilities.h, 113
UtilP.c, 131
genRandReal32
Util32.c, 147
genRandRealP
Utilities.h, 113
UtilP.c, 132
griewangkHandler
EquationHandlers.c, 120
EquationHandlers.h, 13
EquationHandlers32.c, 136
griewangkHost
Equations.c, 26
Equations.h, 33

Harmonic.c
harmonicIteration, 44
harmonicTest, 44
newVector, 45
pitchAdjustment, 45
updateBest, 45
Harmonic.h
harmonicIteration, 47
harmonicTest, 47
newVector, 48
pitchAdjustment, 48
updateBest, 48
harmonicIteration
Harmonic.c, 44
Harmonic.h, 47
harmonicTest
Harmonic.c, 44
Harmonic.h, 47
HostCalls.h
equationHostCalls, 49
HPop
Utilities.h, 104

Info
Utilities.h, 105
init
Init.c, 52
Init.h, 64
Init.c
checkTestType, 51
init, 52
inputFlag, 51
NotRead, 51
processAlpha, 55
processBandwidth, 55
processBeta, 56
processC1, 56
processC2, 57
processDampener, 57
processDimensions, 58
processDimList, 58
processEquations, 59
processExperiments, 59
processGamma, 60
processHMCR, 60
processIterations, 61
processPAR, 61
processRanges, 62
processVectors, 62
Read, 51

Reading, 51
Init.h
checkTestType, 64
init, 64
processAlpha, 67
processBandwidth, 68
processBeta, 68
processC1, 69
processC2, 69
processDampener, 70
processDimensions, 70
processDimList, 71
processEquations, 71
processGamma, 72
processHMCR, 72
processIterations, 73
processPAR, 73
processRanges, 74
processVectors, 75
inputFlag
Init.c, 51

levyHandler
EquationHandlers.c, 121
EquationHandlers.h, 14
EquationHandlers32.c, 137
levyHost
Equations.c, 27
Equations.h, 33
lightIntensity
FA.h, 42
LINE_LENGTH
Utilities.h, 103
lock
Utilities.c, 96

main
main.c, 129
main32.c, 145
main.c
equationHandlers, 130
main, 129
main32.c
equationHandlers, 146
main, 145
mastersCosineWaveHandler
EquationHandlers.c, 121
EquationHandlers.h, 15
EquationHandlers32.c, 137
mastersCosineWaveHost
Equations.c, 27
Equations.h, 34
MAX_FILE_NAME_LEN
Utilities.h, 103
MAX_NUM_EQUATIONS
Utilities.h, 103
MersenneMatrix.c
createMatrix, 76
createVelocities, 77
genDblInRange, 77
genNonNegInt, 78
MersenneMatrix.h
createMatrix, 79
createVelocities, 79
genDblInRange, 80
genNonNegInt, 81
michalewiczHandler
EquationHandlers.c, 122
EquationHandlers.h, 15
EquationHandlers32.c, 138
michalewiczHost
Equations.c, 27
Equations.h, 34
MINIMUM_ITERATIONS
Utilities.h, 103
moveFirefliesLoop
FA.h, 42
MS_PER_SEC
Utilities.h, 103
mutex
Utilities.h, 116

newBest
FA.h, 43
newVector
Harmonic.c, 45
Harmonic.h, 48
NotRead
Init.c, 51
NS_PER_MS
Utilities.h, 104

Particle
Utilities.h, 105
particleLoop
PSO.c, 83
PSO.h, 87
particleSwarmAlg
PSO.c, 84
PSO.h, 88
pathologicalHandler
EquationHandlers.c, 122
EquationHandlers.h, 16
EquationHandlers32.c, 138
pathologicalHost
Equations.c, 28
Equations.h, 34
pitchAdjustment
Harmonic.c, 45
Harmonic.h, 48
printDArray
Utilities.c, 96
Utilities.h, 113
printIArray
Utilities.c, 97
Utilities.h, 114
printMatrix
Utilities.c, 97

Utilities.h, 114
processAlpha
Init.c, 55
Init.h, 67
processBandwidth
Init.c, 55
Init.h, 68
processBeta
Init.c, 56
Init.h, 68
processC1
Init.c, 56
Init.h, 69
processC2
Init.c, 57
Init.h, 69
processDampener
Init.c, 57
Init.h, 70
processDimensions
Init.c, 58
Init.h, 70
processDimList
Init.c, 58
Init.h, 71
processEquations
Init.c, 59
Init.h, 71
processExperiments
Init.c, 59
processGamma
Init.c, 60
Init.h, 72
processHMCR
Init.c, 60
Init.h, 72
processIterations
Init.c, 61
Init.h, 73
processPAR
Init.c, 61
Init.h, 73
processRanges
Init.c, 62
Init.h, 74
processVectors
Init.c, 62
Init.h, 75
PSO.c
calcGBestModifier, 82
calcNewVector, 82
calcNewVelocity, 83
calcPBestModifier, 83
particleLoop, 83
particleSwarmAlg, 84
PSO.h
calcGBestModifier, 86
calcNewVector, 86
calcNewVelocity, 86
calcPBestModifier, 87
particleLoop, 87
particleSwarmAlg, 88
Pthread/EquationHandlers.c, 117
Pthread/main.c, 129
Pthread/UtilP.c, 131

quarticHandler
EquationHandlers.c, 123
EquationHandlers.h, 17
EquationHandlers32.c, 139
quarticHost
Equations.c, 28
Equations.h, 35

ranaHandler
EquationHandlers.c, 123
EquationHandlers.h, 17
EquationHandlers32.c, 139
ranaHost
Equations.c, 28
Equations.h, 35
RANGE_MAX_POS
Utilities.h, 104
RANGE_MIN_POS
Utilities.h, 104
RANGE_SIZE
Utilities.h, 104
rastgrinHandler
EquationHandlers.c, 124
EquationHandlers.h, 18
EquationHandlers32.c, 140
rastgrinHost
Equations.c, 29
Equations.h, 35
Read
Init.c, 51
Reading
Init.c, 51
rosenbrockHandler
EquationHandlers.c, 124
EquationHandlers.h, 19
EquationHandlers32.c, 140
rosenbrockHost
Equations.c, 29
Equations.h, 36
runEquationsAsThreads
EquationHandlers.c, 125
EquationHandlers.h, 19
EquationHandlers32.c, 141

schwefelHandler
EquationHandlers.c, 126
EquationHandlers.h, 21
EquationHandlers32.c, 142
schwefelHost
Equations.c, 29
Equations.h, 36

sineEnvSineWaveHandler
EquationHandlers.c, 127
EquationHandlers.h, 22
EquationHandlers32.c, 143
sineEnvSineWaveHost
Equations.c, 30
Equations.h, 36
stepHandler
EquationHandlers.c, 127
EquationHandlers.h, 23
EquationHandlers32.c, 143
stepHost
Equations.c, 30
Equations.h, 37
stretchVSineWaveHandler
EquationHandlers.c, 128
EquationHandlers.h, 23
EquationHandlers32.c, 144
stretchVSineWaveHost
Equations.c, 30
Equations.h, 37

TestType
Utilities.h, 105
testTypeCalls
EquationHandlers.c, 128
EquationHandlers32.c, 144

unlock
Utilities.c, 99
updateBest
Harmonic.c, 45
Harmonic.h, 48
Util32.c
genRandInt32, 147
genRandReal32, 147
Utilities.c
allocateEmptyMatrix, 90
allocateHPop, 90
copyArray, 91
copyMatrix, 91
createParticles, 91
evalNewWorst, 92
evaluateFitness, 93
evaluatePop, 93
freeEquationInfo, 94
freeFireflySwarm, 94
freeHPop, 95
freeInfo, 95
freeMatrix, 96
freeParticles, 96
lock, 96
printDArray, 96
printIArray, 97
printMatrix, 97
unlock, 99
writePopulationLogToFile, 99
writeResultToFile, 100
Utilities.h
allocateEmptyMatrix, 105
allocateHPop, 106
copyArray, 106
copyMatrix, 106
createParticles, 107
DEFAULT_INIT_FILE, 103
EquationInfo, 104
EquationPosition, 105
evalNewWorst, 107
evaluateFitness, 108
evaluatePop, 108
FILE_ARGUMENT, 103
FireflySwarm, 104
freeEquationInfo, 109
freeFireflySwarm, 110
freeHPop, 110
freeInfo, 110
freeMatrix, 112
freeParticles, 112
genRandIntP, 113
genRandRealP, 113
HPop, 104
Info, 105
LINE_LENGTH, 103
MAX_FILE_NAME_LEN, 103
MAX_NUM_EQUATIONS, 103
MINIMUM_ITERATIONS, 103
MS_PER_SEC, 103
mutex, 116
NS_PER_MS, 104
Particle, 105
printDArray, 113
printIArray, 114
printMatrix, 114
RANGE_MAX_POS, 104
RANGE_MIN_POS, 104
RANGE_SIZE, 104
TestType, 105
writePopulationLogToFile, 115
writeResultToFile, 115
UtilP.c
genRandIntP, 131
genRandRealP, 132

Win32/EquationHandlers32.c, 132
Win32/main32.c, 145
Win32/Util32.c, 147
writePopulationLogToFile
Utilities.c, 99
Utilities.h, 115
writeResultToFile
Utilities.c, 100
Utilities.h, 115