CS471 Project 3

Generated by Doxygen 1.8.13

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 mdata Namespace Reference

Classes

- class DataTable

  The DataTable class is a simple table of values with labeled columns.
- class Population

  Data class for storing a multi-dimensional population of data with the associated fitness.

4.2 mfunc Namespace Reference

Classes

- struct DEParams

  Simple structure that holds various parameters to run the differential evolutionary algorithm.
- class DifferentialEvolution

  The DifferentialEvolution class executes the differential evolution algorithm on a given population using the given parameters passed via a DEParams structure. To start, call the "run" function.
- class Experiment

  Contains classes for running the CS471 project experiment.
- struct FunctionDesc

  get() returns a function's description Returns a C-string description for the given function id if the id is valid. Otherwise returns null
- struct Functions

  Struct containing all static math functions. A function can be called directly by name, or indirectly using Functions::get or Functions::exec.
- struct GAParams

  Simple structure that holds various parameters for the genetic algorithm.
- class GeneticAlgorithm

  The GeneticAlgorithm class executes the genetic algorithm with the specified parameters. To start, execute the run() function.
- struct RandomBounds

  Simple struct for storing the minimum and maximum input vector bounds for a function.

Typedefs

- template<class T >
  using mfuncPtr = T(*)(T *, size_t)

  Function pointer that takes two arguments T and size_t, and returns a T value.*

Enumerations

- enum DEStrategy {
  DEStrategy::Best1Exp = 0, DEStrategy::Rand1Exp = 1, DEStrategy::RandToBest1Exp = 2, DEStrategy::Best2Exp = 3,
  DEStrategy::Rand2Exp = 4, DEStrategy::Best1Bin = 5, DEStrategy::Rand1Bin = 6, DEStrategy::RandToBest1Bin = 7,
  DEStrategy::Best2Bin = 8, DEStrategy::Rand2Bin = 9, DEStrategy::Count = 10 }

  Enum used to specify which differential evolution strategy should be used.
- enum PerturbedVector { PerturbedVector::Best, PerturbedVector::Random, PerturbedVector::RandToBest }

  Enum used to specify what vector should be perturbed during mutation.
- enum NumberDiffVectors { NumberDiffVectors::One, NumberDiffVectors::Two }

  Enum used to specify the number of random difference vectors used during mutation.
- enum CrossoverStrat { CrossoverStrat::Exponential, CrossoverStrat::Binomial }

  Enum used to specify a crossover strategy.
- enum Algorithm { Algorithm::GeneticAlgorithm = 0, Algorithm::DifferentialEvolution = 1, Algorithm::Count = 2 }

  Simple enum that selects one of the evolutionary algorithms.

Variables

- constexpr const unsigned int NUM_FUNCTIONS = _NUM_FUNCTIONS

4.2.1 Detailed Description

Scope for all math functions

4.2.2 Typedef Documentation

4.2.2.1 mfuncPtr

```
template<class T >
using mfunc::mfuncPtr = typedef T (*)(T*, size_t)
```

Function pointer that takes two arguments T* and size_t, and returns a T value.

Template Parameters

T	Data type for vector and return value

Definition at line 28 of file mfuncptr.h.

4.2.3 Enumeration Type Documentation

4.2.3.1 Algorithm

```
enum mfunc::Algorithm  [strong]
```

Simple enum that selects one of the evolutionary algorithms.

Enumerator

GeneticAlgorithm	
DifferentialEvolution	
Count	

Definition at line 43 of file experiment.h.

```
00044     {
00045         GeneticAlgorithm = 0,
00046         DifferentialEvolution = 1,
00047         Count = 2
00048     };
```

4.2.3.2 CrossoverStrat

```
enum mfunc::CrossoverStrat  [strong]
```

Enum used to specify a crossover strategy.

Enumerator

Exponential	
Binomial	

Definition at line 66 of file diffevoalg.h.

```
00067     {
00068         Exponential,
00069         Binomial
00070     };
```

4.2.3.3 DEStrategy

```
enum mfunc::DEStrategy  [strong]
```

Enum used to specify which differential evolution strategy should be used.

Enumerator

Best1Exp	
Rand1Exp	
RandToBest1Exp	
Best2Exp	
Rand2Exp	
Best1Bin	
Rand1Bin	
RandToBest1Bin	
Best2Bin	
Rand2Bin	
Count	

Definition at line 29 of file diffevoalg.h.

```
00030     {
00031         Best1Exp = 0,
00032         Rand1Exp = 1,
00033         RandToBest1Exp = 2,
00034         Best2Exp = 3,
00035         Rand2Exp = 4,
00036         Best1Bin = 5,
00037         Rand1Bin = 6,
00038         RandToBest1Bin = 7,
00039         Best2Bin = 8,
00040         Rand2Bin = 9,
00041         Count = 10
00042     };
```

4.2.3.4 NumberDiffVectors

```
enum mfunc::NumberDiffVectors  [strong]
```

Enum used to specify the number of random difference vectors used during mutation.

Enumerator

One	
Two	

Definition at line 57 of file diffevoalg.h.

```
00058     {
00059         One,
00060         Two
00061     };
```

4.2.3.5 PerturbedVector

```
enum mfunc::PerturbedVector  [strong]
```

Enum used to specify what vector should be perturbed during mutation.

Enumerator

Best	
Random	
RandToBest	

Definition at line 47 of file diffevoalg.h.

```
00048     {
00049         Best,
00050         Random,
00051         RandToBest
00052     };
```

4.2.4 Variable Documentation

4.2.4.1 NUM_FUNCTIONS

```
constexpr const unsigned int mfunc::NUM_FUNCTIONS = _NUM_FUNCTIONS
```

Constant value for the total number of math functions contained in this namespace

Definition at line 67 of file mfunctions.h.

Referenced by mfunc::Experiment< T >::runDEThreaded(), mfunc::Experiment< T >::testAllFunc_DE(), and mfunc::Experiment< T >::testAllFunc_GA().

4.3 util Namespace Reference

Classes

- class IniReader

  *The IniReader class is a simple *.ini file reader and parser.*

Functions

- template<class T = double>
  void initArray (T *a, size_t size, T val)

  Initializes an array with some set value.
- template<class T = double>
  void initMatrix (T **m, size_t rows, size_t cols, T val)

  Initializes a matrix with a set value for each entry.
- template<class T = double>
  bool releaseArray (T *&a)

  Releases an allocated array's memory and sets the pointer to nullptr.
- template<class T = double>
  void releaseMatrix (T **&m, size_t rows)

  Releases an allocated matrix's memory and sets the pointer to nullptr.
- template<class T = double>
  T * allocArray (size_t size)

  Allocates a new array of the given data type.
- template<class T = double>
  T ** allocMatrix (size_t rows, size_t cols)

  Allocates a new matrix of the given data type.
- template<class T = double>
  void copyArray (T *src, T *dest, size_t size)

  Copies the elements from one equal-sized array to another.

4.3.1 Function Documentation

4.3.1.1 allocArray()

```
template<class T = double>
T* util::allocArray (
            size_t size )  [inline]
```

Allocates a new array of the given data type.

Template Parameters

Data	type of the array

Parameters

size	Number of elements in the array

Returns

Returns a pointer to the new array, or nullptr allocation fails

Definition at line 116 of file mem.h.

```
00117     {
00118         return new(std::nothrow) T[size];
00119     }
```

4.3.1.2 allocMatrix()

```
template<class T = double>
T** util::allocMatrix (
            size_t rows,
            size_t cols )  [inline]
```

Allocates a new matrix of the given data type.

Template Parameters

Data	type of the matrix entries

Parameters

rows	The number of rows
cols	The number of columns

Returns

Returns a pointer to the new matrix, or nullptr if allocation fails

Definition at line 130 of file mem.h.

```
00131     {
00132         T** m = (T**)allocArray<T*>(rows);
00133         if (m == nullptr) return nullptr;
00134 
00135         for (size_t i = 0; i < rows; i++)
00136         {
00137             m[i] = allocArray<T>(cols);
00138             if (m[i] == nullptr)
00139             {
00140                 releaseMatrix<T>(m, rows);
00141                 return nullptr;
00142             }
00143         }
00144 
00145         return m;
00146     }
```

4.3.1.3 copyArray()

```
template<class T = double>
void util::copyArray (
            T * src,
            T * dest,
            size_t size )  [inline]
```

Copies the elements from one equal-sized array to another.

Template Parameters

Data	type of the array

Parameters

src	Source array from where the elements will be copied from
dest	Destination array from where the elements will be copied to
size	Number of elements in the array

Definition at line 157 of file mem.h.

```
00158     {
00159         for (size_t i = 0; i < size; i++)
00160             dest[i] = src[i];
00161     }
```

4.3.1.4 initArray()

```
template<class T = double>
void util::initArray (
            T * a,
            size_t size,
            T val )  [inline]
```

Initializes an array with some set value.

Template Parameters

Data	type of array

Parameters

a	Pointer to array
size	Size of the array
val	Value to initialize the array to

Definition at line 29 of file mem.h.

Referenced by initMatrix().

```
00030     {
00031         if (a == nullptr) return;
00032 
00033         for (size_t i = 0; i < size; i++)
00034         {
00035             a[i] = val;
00036         }
00037     }
```

4.3.1.5 initMatrix()

```
template<class T = double>
void util::initMatrix (
            T ** m,
            size_t rows,
            size_t cols,
            T val )  [inline]
```

Initializes a matrix with a set value for each entry.

Template Parameters

Data	type of matrix entries

Parameters

m	Pointer to a matrix
rows	Number of rows in matrix
cols	Number of columns in matrix
val	Value to initialize the matrix to

Definition at line 49 of file mem.h.

References initArray().

```
00050     {
00051         if (m == nullptr) return;
00052
00053         for (size_t i = 0; i < rows; i++)
00054         {
00055             initArray(m[i], cols, val);
00056         }
00057     }
```

4.3.1.6 releaseArray()

```
template<class T = double>
bool util::releaseArray (
            T *& a )
```

Releases an allocated array's memory and sets the pointer to nullptr.

Template Parameters

Data	type of array

Parameters

a	Pointer to array

Definition at line 66 of file mem.h.

```
00067     {
00068         if (a == nullptr) return true;
00069 
00070         try
00071         {
00072             delete[] a;
00073             a = nullptr;
00074             return true;
00075         }
00076         catch(...)
00077         {
00078             return false;
00079         }
00080     }
```

4.3.1.7 releaseMatrix()

```
template<class T = double>
void util::releaseMatrix (
              T **& m,
              size_t rows )
```

Releases an allocated matrix's memory and sets the pointer to nullptr.

Template Parameters

Data	type of the matrix

Parameters

m	Pointer th the matrix
rows	The number of rows in the matrix

Definition at line 90 of file mem.h.

Referenced by mdata::DataTable< T >::~DataTable().

```
00091     {
00092         if (m == nullptr) return;
00093 
00094         for (size_t i = 0; i < rows; i++)
00095         {
00096             if (m[i] != nullptr)
00097             {
00098                 // Release each row
00099                 releaseArray<T>(m[i]);
00100             }
00101         }
00102 
00103         // Release columns
00104         delete[] m;
00105         m = nullptr;
00106     }
```

Chapter 5

Class Documentation

5.1 mdata::DataTable< T > Class Template Reference

The DataTable class is a simple table of values with labeled columns.

`#include <datatable.h>`

Public Member Functions

- DataTable (size_t _rows, size_t _cols)
  Construct a new Data Table object Throws std::length_error and std::bad_alloc.
- ~DataTable ()
  Destroy the Data Table object.
- void clearData ()
- std::string getColLabel (size_t colIndex)
  Gets the string label for the column with the given index.
- void setColLabel (size_t colIndex, std::string newLabel)
  Sets the string label for the column with the given index.
- T getEntry (size_t row, size_t col)
  Returns the value in the table at the given row and column.
- void setEntry (size_t row, size_t col, T val)
  Set the value for the table entry at the given row and column.
- bool exportCSV (const char *filePath)
  Exports the contents of this DataTable to a .csv file.

5.1.1 Detailed Description

template<class T>
class mdata::DataTable< T >

The DataTable class is a simple table of values with labeled columns.

– Initialize a DataTable object with a specified number of rows and columns: DataTable table(rows, columns);

Set a column's label:

table.setColLabel(0, "Column 1");

Set an entry in the table:

table.setEntry(n, m, value);

Where 'n' is the row, 'm' is the column, and 'value' is the value of the entry

Export the table to a ∗.csv file:

bool success = table.exportCSV("my_file.csv");

Definition at line 50 of file datatable.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 DataTable()

```
template<class T>
mdata::DataTable< T >::DataTable (
            size_t _rows,
            size_t _cols )  [inline]
```

Construct a new Data Table object Throws std::length_error and std::bad_alloc.

Parameters

_rows	Number of rows in table
_cols	Number of columns in table

Definition at line 60 of file datatable.h.

```
00060                                           : rows(_rows), cols(_cols), dataMatrix(nullptr)
00061         {
00062             if (rows == 0)
00063                 throw std::length_error("Table rows must be greater than 0.");
00064             else if (cols == 0)
00065                 throw std::length_error("Table columns must be greater than 0.");
00066
00067             dataMatrix = util::allocMatrix<T>(rows, cols);
00068             if (dataMatrix == nullptr)
00069                 throw std::bad_alloc();
00070
00071             colLabels.resize(_cols, std::string());
00072         }
```

5.1.2.2 ~DataTable()

```
template<class T>
mdata::DataTable< T >::~DataTable ( )  [inline]
```

Destroy the Data Table object.

Definition at line 77 of file datatable.h.

References util::releaseMatrix().

```
00078         {
00079             util::releaseMatrix(dataMatrix, rows);
00080         }
```

5.1.3 Member Function Documentation

5.1.3.1 clearData()

```
template<class T>
void mdata::DataTable< T >::clearData ( )  [inline]
```

Definition at line 82 of file datatable.h.

Referenced by mfunc::Experiment< T >::testAllFunc_DE(), and mfunc::Experiment< T >::testAllFunc_GA().

```
00083         {
00084             util::initMatrix<T>(dataMatrix, rows, cols, 0);
00085         }
```

5.1.3.2 exportCSV()

```
template<class T>
bool mdata::DataTable< T >::exportCSV (
            const char * filePath )  [inline]
```

Exports the contents of this DataTable to a .csv file.

Parameters

filePath	Path to the file that will be filled with this table's values

Returns

true If the file was successfully written to
false If there was an error opening the file

Definition at line 160 of file datatable.h.

Referenced by mfunc::Experiment< T >::testAllFunc_DE(), and mfunc::Experiment< T >::testAllFunc_GA().

```
00161         {
00162             if (dataMatrix == nullptr) return false;
00163 
00164             using namespace std;
00165             ofstream outFile;
00166             outFile.open(filePath, ofstream::out | ofstream::trunc);
00167             if (!outFile.good()) return false;
00168 
00169             // Print column labels
00170             for (unsigned int c = 0; c < cols; c++)
00171             {
00172                 outFile << colLabels[c];
00173                 if (c < cols - 1) outFile << ",";
00174             }
00175 
00176             outFile << endl;
00177 
00178             // Print data rows
00179             for (unsigned int r = 0; r < rows; r++)
00180             {
00181                 for (unsigned int c = 0; c < cols; c++)
00182                 {
00183                     outFile << std::setprecision(8) << dataMatrix[r][c];
00184                     if (c < cols - 1) outFile << ",";
00185                 }
00186                 outFile << endl;
00187             }
00188 
00189             outFile.close();
00190             return true;
00191         }
```

5.1.3.3 getColLabel()

```
template<class T>
std::string mdata::DataTable< T >::getColLabel (
            size_t colIndex )  [inline]
```

Gets the string label for the column with the given index.

Parameters

colIndex	Index of the column

Returns

std::string String value of the column label

Definition at line 93 of file datatable.h.

```
00094         {
00095             if (colIndex >= colLabels.size())
00096                 throw std::out_of_range("Column index out of range");
00097 
00098             return colLabels[colIndex];
00099         }
```

5.1.3.4 getEntry()

```
template<class T>
T mdata::DataTable< T >::getEntry (
            size_t row,
            size_t col )  [inline]
```

Returns the value in the table at the given row and column.

Parameters

row	Row index of the table
col	Column index of the table

Returns

T Value of the entry at the given row and column

Definition at line 122 of file datatable.h.

```
00123         {
00124             if (dataMatrix == nullptr)
00125                 throw std::runtime_error("Data matrix not allocated");
00126             if (row >= rows)
00127                 throw std::out_of_range("Table row out of range");
00128             else if (col >= cols)
00129                 throw std::out_of_range("Table column out of range");
00130 
00131             return dataMatrix[row][col];
00132         }
```

5.1.3.5 setColLabel()

```
template<class T>
void mdata::DataTable< T >::setColLabel (
            size_t colIndex,
            std::string newLabel )  [inline]
```

Sets the string label for the column with the given index.

Parameters

colIndex	Index of the column
newLabel	New string label for the column

Definition at line 107 of file datatable.h.

Referenced by mfunc::Experiment< T >::testAllFunc_DE(), and mfunc::Experiment< T >::testAllFunc_GA().

```
00108         {
00109             if (colIndex >= colLabels.size())
```

```
00110                 throw std::out_of_range("Column index out of range");
00111
00112             colLabels[colIndex] = newLabel;
00113         }
```

5.1.3.6 setEntry()

```
template<class T>
void mdata::DataTable< T >::setEntry (
             size_t row,
             size_t col,
             T val )  [inline]
```

Set the value for the table entry at the given row and column.

Parameters

row	Row index of the table
col	Column index of the table
val	New value for the entry

Definition at line 141 of file datatable.h.

Referenced by mfunc::Experiment< T >::runDEThreaded(), and mfunc::Experiment< T >::runGAThreaded().

```
00142         {
00143             if (dataMatrix == nullptr)
00144                 throw std::runtime_error("Data matrix not allocated");
00145             if (row >= rows)
00146                 throw std::out_of_range("Table row out of range");
00147             else if (col >= cols)
00148                 throw std::out_of_range("Table column out of range");
00149
00150             dataMatrix[row][col] = val;
00151         }
```

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

- include/datatable.h

5.2 mfunc::DEParams< T > Struct Template Reference

Simple structure that holds various parameters to run the differential evolutionary algorithm.

```
#include <diffevoalg.h>
```

Public Member Functions

- DEParams ()

Public Attributes

- mdata::DataTable< T > * fitnessTable
- size_t fitTableCol
- mdata::Population< T > * mainPop
- mdata::Population< T > * nextPop
- mfuncPtr< T > fPtr
- T fMinBound
- T fMaxBound
- unsigned int generations
- double crFactor
- double scalingFactor1
- double scalingFactor2
- DEStrategy strategy

5.2.1 Detailed Description

template<class T>
struct mfunc::DEParams< T >

Simple structure that holds various parameters to run the differential evolutionary algorithm.

Template Parameters

T	Datatype used by the algorithm

Definition at line 79 of file diffevoalg.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 DEParams()

```
template<class T>
mfunc::DEParams< T >::DEParams ( )  [inline]
```

Definition at line 94 of file diffevoalg.h.

References mfunc::Best1Exp.

```
00095         {
00096             fitnessTable = nullptr;
00097             fitTableCol = 0;
00098             mainPop = nullptr;
00099             nextPop = nullptr;
00100             fPtr = nullptr;
00101             fMinBound = 0;
00102             fMaxBound = 0;
00103             generations = 0;
00104             crFactor = 0;
00105             scalingFactor1 = 0;
00106             scalingFactor2 = 0;
00107             strategy = DEStrategy::Best1Exp;
00108         }
```

5.2.3 Member Data Documentation

5.2.3.1 crFactor

```
template<class T>
double mfunc::DEParams< T >::crFactor
```

Definition at line 89 of file diffevoalg.h.

Referenced by mfunc::DifferentialEvolution< T >::run(), mfunc::Experiment< T >::runDEThreaded(), and mfunc::Experiment< T >::testAllFunc_DE().

5.2.3.2 fitnessTable

```
template<class T>
mdata::DataTable<T>* mfunc::DEParams< T >::fitnessTable
```

Definition at line 81 of file diffevoalg.h.

Referenced by mfunc::DifferentialEvolution< T >::run(), and mfunc::Experiment< T >::testAllFunc_DE().

5.2.3.3 fitTableCol

```
template<class T>
size_t mfunc::DEParams< T >::fitTableCol
```

Definition at line 82 of file diffevoalg.h.

Referenced by mfunc::DifferentialEvolution< T >::run(), and mfunc::Experiment< T >::testAllFunc_DE().

5.2.3.4 fMaxBound

```
template<class T>
T mfunc::DEParams< T >::fMaxBound
```

Definition at line 87 of file diffevoalg.h.

Referenced by mfunc::DifferentialEvolution< T >::run(), and mfunc::Experiment< T >::testAllFunc_DE().

5.2.3.5 fMinBound

```
template<class T>
T mfunc::DEParams< T >::fMinBound
```

Definition at line 86 of file diffevoalg.h.

Referenced by mfunc::DifferentialEvolution< T >::run(), and mfunc::Experiment< T >::testAllFunc_DE().

5.2.3.6 fPtr

```
template<class T>
mfuncPtr<T> mfunc::DEParams< T >::fPtr
```

Definition at line 85 of file diffevoalg.h.

Referenced by mfunc::DifferentialEvolution< T >::run(), and mfunc::Experiment< T >::testAllFunc_DE().

5.2.3.7 generations

```
template<class T>
unsigned int mfunc::DEParams< T >::generations
```

Definition at line 88 of file diffevoalg.h.

Referenced by mfunc::DifferentialEvolution< T >::run(), mfunc::Experiment< T >::runDEThreaded(), and mfunc::Experiment< T >::testAllFunc_DE().

5.2.3.8 mainPop

```
template<class T>
mdata::Population<T>* mfunc::DEParams< T >::mainPop
```

Definition at line 83 of file diffevoalg.h.

Referenced by mfunc::DifferentialEvolution< T >::run(), mfunc::Experiment< T >::runDEThreaded(), and mfunc::Experiment< T >::testAllFunc_DE().

5.2.3.9 nextPop

```
template<class T>
mdata::Population<T>* mfunc::DEParams< T >::nextPop
```

Definition at line 84 of file diffevoalg.h.

Referenced by mfunc::DifferentialEvolution< T >::run(), mfunc::Experiment< T >::runDEThreaded(), and mfunc::Experiment< T >::testAllFunc_DE().

5.2.3.10 scalingFactor1

```
template<class T>
double mfunc::DEParams< T >::scalingFactor1
```

Definition at line 90 of file diffevoalg.h.

Referenced by mfunc::DifferentialEvolution< T >::run(), mfunc::Experiment< T >::runDEThreaded(), and mfunc::Experiment< T >::testAllFunc_DE().

5.2.3.11 scalingFactor2

```
template<class T>
double mfunc::DEParams< T >::scalingFactor2
```

Definition at line 91 of file diffevoalg.h.

Referenced by mfunc::DifferentialEvolution< T >::run(), mfunc::Experiment< T >::runDEThreaded(), and mfunc::Experiment< T >::testAllFunc_DE().

5.2.3.12 strategy

```
template<class T>
DEStrategy mfunc::DEParams< T >::strategy
```

Definition at line 92 of file diffevoalg.h.

Referenced by mfunc::DifferentialEvolution< T >::run(), mfunc::Experiment< T >::runDEThreaded(), and mfunc::Experiment< T >::testAllFunc_DE().

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

- include/diffevoalg.h

5.3 mfunc::DifferentialEvolution< T > Class Template Reference

The DifferentialEvolution class executes the differential evolution algorithm on a given population using the given parameters passed via a DEParams structure. To start, call the "run" function.

`#include <diffevoalg.h>`

Public Member Functions

- DifferentialEvolution ()

  Construct a DifferentialEvolution object.
- ~DifferentialEvolution ()=default
- int run (DEParams< T > params)

  Runs the algorithm with the given parameters.

5.3.1 Detailed Description

template<class T>
class mfunc::DifferentialEvolution< T >

The DifferentialEvolution class executes the differential evolution algorithm on a given population using the given parameters passed via a DEParams structure. To start, call the "run" function.

Template Parameters

T	

Definition at line 119 of file diffevoalg.h.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 DifferentialEvolution()

```
template<class T >
mfunc::DifferentialEvolution< T >::DifferentialEvolution ( )
```

Construct a DifferentialEvolution object.

Template Parameters

T	Datatype used by the algorithm

Definition at line 143 of file diffevoalg.h.

```
00144     : seed(), engine(seed()), rchance(0, 1)
00145 {
00146 }
```

5.3.2.2 ~DifferentialEvolution()

```
template<class T>
mfunc::DifferentialEvolution< T >::~DifferentialEvolution ( )  [default]
```

5.3.3 Member Function Documentation

5.3.3.1 run()

```
template<class T >
int mfunc::DifferentialEvolution< T >::run (
            DEParams< T > p )
```

Runs the algorithm with the given parameters.

Template Parameters

T	Datatype used by the algorithm

Parameters

p	Algorithm parameters

Returns

int Non-zero error code on error

Definition at line 156 of file diffevoalg.h.

References mfunc::Best, mfunc::Best1Bin, mfunc::Best1Exp, mfunc::Best2Bin, mfunc::Best2Exp, mfunc::Binomial, mdata::Population< T >::calcFitness(), mdata::Population< T >::copyPopulation(), mfunc::DEParams< T >::crFactor, mfunc::Exponential, mfunc::DEParams< T >::fitnessTable, mfunc::DEParams< T >::fitTableCol, mfunc::DEParams< T >::fMaxBound, mfunc::DEParams< T >::fMinBound, mfunc::DEParams< T >::fPtr, mfunc::DEParams< T >::generations, mdata::Population< T >::getDimensionsSize(), mdata::Population< T >::getFitness(), mdata::Population< T >::getPopulationPtr(), mfunc::DEParams< T >::mainPop, MAX_RAND_VECTOR_SELECT, mfunc::DEParams< T >::nextPop, mfunc::One, mfunc::Rand1Bin, mfunc::Rand1Exp, mfunc::Rand2Bin, mfunc::Rand2Exp, mfunc::Random, mfunc::RandToBest, mfunc::RandToBest1Bin, mfunc::RandToBest1Exp, mfunc::DEParams< T >::scalingFactor1, mfunc::DEParams< T >::scalingFactor2, mdata::Population< T >::setFitness(), mfunc::DEParams< T >::strategy, and mfunc::Two.

Referenced by mfunc::Experiment< T >::runDEThreaded().

```
00157 {
00158     if (p.mainPop == nullptr || p.nextPop == nullptr || p.fPtr == nullptr)
00159         return 1;
00160 
00161     if (p.mainPop->getPopulationSize() != p.nextPop->getPopulationSize() ||
00162         p.mainPop->getDimensionsSize() != p.nextPop->getDimensionsSize())
00163         return 2;
00164 
00165     const size_t popSize = p.mainPop->getPopulationSize();
00166     const size_t dimSize = p.mainPop->getDimensionsSize();
00167 
00168     // Parse DE strategy
00169     PerturbedVector pertV;
00170     NumberDiffVectors diffV;
00171     CrossoverStrat crStrat;
00172     parseDEStrat(p.strategy, pertV, diffV, crStrat);
00173 
00174     // Prepare populations
00175     p.mainPop->setFitnessNormalization(false);
00176     p.nextPop->setFitnessNormalization(false);
00177     p.mainPop->generate(p.fMinBound, p.fMaxBound);
00178     p.mainPop->calcAllFitness(p.fPtr);
00179 
00180     // Calc best fitness pop index
00181     size_t bestFitIndex = p.mainPop->getBestFitnessIndex();
00182 
00183     for (unsigned int gen = 0; gen < p.generations; gen++)
00184     {
00185         for (size_t i = 0; i < popSize; i++)
00186         {
00187             // For each population in the next generation, mutate and crossover, then select
00188             mutateAndCrossover(p.mainPop, p.nextPop, i, bestFitIndex, pertV, diffV, crStrat, p.
      scalingFactor1, p.scalingFactor2, p.crFactor);
00189             p.nextPop->boundPopulation(i, p.fMinBound, p.fMaxBound);
00190             select(p.mainPop, p.nextPop, i, p.fPtr);
00191         }
00192 
00193         // Swap the two populations
00194         auto tmp = p.mainPop;
00195         p.mainPop = p.nextPop;
00196         p.nextPop = tmp;
00197 
00198         // Recalculate best fitness index and add result to results table
00199         bestFitIndex = p.mainPop->getBestFitnessIndex();
00200         p.fitnessTable->setEntry(gen, p.fitTableCol, p.mainPop->getFitness(bestFitIndex));
00201     }
00202 
00203     return 0;
00204 }
```

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

- include/diffevoalg.h

5.4 mfunc::Experiment< T > Class Template Reference

Contains classes for running the CS471 project experiment.

`#include <experiment.h>`

Public Member Functions

- Experiment ()

  Construct a new Experiment object.
- ~Experiment ()

  Destroys the Experiment object.
- bool init (const char *paramFile)

Initializes the CS471 project 2 experiment. Opens the given parameter file and extracts test parameters. Allocates memory for function vectors and function bounds. Extracts all function bounds.

- int testAllFunc ()

  Executes all functions as specified in the CS471 project 3 document, records results, and outputs the data as a ∗.csv file.

- int testAllFunc_GA ()

  Tests all 18 functions using the genetic algorithm and outputs results.

- int runGAThreaded (GAParams< T > gaParams, mdata::DataTable< double > ∗tTable, size_t tRow, size_t tCol)

  Executes a single iteration of a test with the given parameters.

- int testAllFunc_DE ()

  Tests all 18 functions using the differential evolution algorithm and outputs results.

- int runDEThreaded (DEParams< T > deParams, mdata::DataTable< double > ∗tTable, size_t tRow, size_t tCol)

  Executes a single iteration of a test with the given parameters.

5.4.1 Detailed Description

template<class T>
class mfunc::Experiment< T >

Contains classes for running the CS471 project experiment.

The Experiment class opens a given parameter .ini file and executes the CS471 project 2 experiment with the specified parameters. runAllFunc() runs all 18 functions defined in mfunctions.h a given number of times with vectors of random values that have a given number of dimensions and collects all results/data. This data is then entered into a DataTable and exported as a ∗.csv file.

Definition at line 61 of file experiment.h.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Experiment()

```
template<class T >
Experiment::Experiment ( )
```

Construct a new Experiment object.

Definition at line 58 of file experiment.cpp.

```
00059     : vBounds(nullptr), tPool(nullptr), resultsFile(""), execTimesFile(""), iterations(0)
00060 {
00061 }
```

5.4.2.2 ~Experiment()

```
template<class T >
Experiment::~Experiment ( )
```

Destroys the Experiment object.

Definition at line 68 of file experiment.cpp.

```
00069 {
00070     releaseThreadPool();
00071     releasePopulationPool();
00072     releaseVBounds();
00073 }
```

5.4.3 Member Function Documentation

5.4.3.1 init()

```
template<class T >
bool Experiment::init (
            const char * paramFile )
```

Initializes the CS471 project 2 experiment. Opens the given parameter file and extracts test parameters. Allocates memory for function vectors and function bounds. Extracts all function bounds.

Parameters

paramFile	File path to the parameter ini file

Returns

Returns true if initialization was successful. Otherwise false.

Definition at line 84 of file experiment.cpp.

References mfunc::Count, util::IniReader::getEntry(), util::IniReader::getEntryAs(), INI_TEST_ALGORITHM, INI_↩TEST_DIMENSIONS, INI_TEST_EXECTIMESFILE, INI_TEST_ITERATIONS, INI_TEST_NUMTHREADS, INI_↩TEST_POPULATION, INI_TEST_RESULTSFILE, INI_TEST_SECTION, and util::IniReader::openFile().

Referenced by runExp().

```
00085 {
00086     try
00087     {
00088         // Open and parse parameters file
00089         if (!iniParams.openFile(paramFile))
00090         {
00091             cerr << "Experiment init failed: Unable to open param file: " << paramFile << endl;
00092             return false;
00093         }
00094
00095         // Extract test parameters from ini file
```

```
00096         long numberSol = iniParams.getEntryAs<long>(INI_TEST_SECTION,
      INI_TEST_POPULATION);
00097         long numberDim = iniParams.getEntryAs<long>(INI_TEST_SECTION,
      INI_TEST_DIMENSIONS);
00098         long numberIter = iniParams.getEntryAs<long>(INI_TEST_SECTION,
      INI_TEST_ITERATIONS);
00099         long numberThreads = iniParams.getEntryAs<long>(
      INI_TEST_SECTION, INI_TEST_NUMTHREADS);
00100         unsigned int selectedAlg = iniParams.getEntryAs<unsigned int>(
      INI_TEST_SECTION, INI_TEST_ALGORITHM);
00101         resultsFile = iniParams.getEntry(INI_TEST_SECTION,
      INI_TEST_RESULTSFILE);
00102         execTimesFile = iniParams.getEntry(INI_TEST_SECTION,
      INI_TEST_EXECTIMESFILE);
00103
00104         // Verify test parameters
00105         if (numberSol <= 0)
00106         {
00107             cerr << "Experiment init failed: Param file [test]->"
00108                 << INI_TEST_POPULATION << " entry missing or out of bounds: " <<
      paramFile << endl;
00109             return false;
00110         }
00111         else if (numberDim <= 0)
00112         {
00113             cerr << "Experiment init failed: Param file [test]->"
00114                 << INI_TEST_DIMENSIONS << " entry missing or out of bounds: " <<
      paramFile << endl;
00115             return false;
00116         }
00117         else if (numberIter <= 0)
00118         {
00119             cerr << "Experiment init failed: Param file [test]->"
00120                 << INI_TEST_ITERATIONS << " entry missing or out of bounds: " <<
      paramFile << endl;
00121             return false;
00122         }
00123         else if (numberThreads <= 0)
00124         {
00125             cerr << "Experiment init failed: Param file [test]->"
00126                 << INI_TEST_NUMTHREADS << " entry missing or out of bounds: " <<
      paramFile << endl;
00127             return false;
00128         }
00129         else if (selectedAlg >= static_cast<unsigned int>(Algorithm::Count))
00130         {
00131             cerr << "Experiment init failed: Param file [test]->"
00132                 << INI_TEST_ALGORITHM << " entry missing or out of bounds: " << paramFile
       << endl;
00133             return false;
00134         }
00135
00136         // Cast iterations and test algorithm to correct types
00137         iterations = (size_t)numberIter;
00138         testAlg = static_cast<Algorithm>(selectedAlg);
00139
00140         // Print test parameters to console
00141         cout << "Population size: " << numberSol << endl;
00142         cout << "Dimensions: " << numberDim << endl;
00143         cout << "Iterations: " << iterations << endl;
00144         // cout << "Algorithm: " << enums::AlgorithmNames::get(testAlg) << endl;
00145
00146         // Allocate memory for all population objects. We need one for each thread to prevent conflicts.
00147         if (!allocatePopulationPool((size_t)numberThreads * 2, (size_t)numberSol, (size_t)numberDim))
00148         {
00149             cerr << "Experiment init failed: Unable to allocate populations." << endl;
00150             return false;
00151         }
00152
00153         // Allocate memory for function vector bounds
00154         if (!allocateVBounds())
00155         {
00156             cerr << "Experiment init failed: Unable to allocate vector bounds array." << endl;
00157             return false;
00158         }
00159
00160         // Fill function bounds array with data parsed from iniParams
00161         if (!parseFuncBounds())
00162         {
00163             cerr << "Experiment init failed: Unable to parse vector bounds array." << endl;
00164             return false;
00165         }
00166
00167         // Allocate thread pool
00168         if (!allocateThreadPool((size_t)numberThreads))
00169         {
00170             cerr << "Experiment init failed: Unable to allocate thread pool." << endl;
```

```
00171             return false;
00172         }
00173
00174         cout << "Started " << numberThreads << " worker threads ..." << endl;
00175
00176         // Ready to run an experiment
00177         return true;
00178     }
00179     catch (const std::exception& ex)
00180     {
00181         cerr << "Exception occurred while initializing experiment: " << ex.what() << endl;
00182         return false;
00183     }
00184     catch (...)
00185     {
00186         cerr << "Unknown Exception occurred while initializing experiment." << endl;
00187         return false;
00188     }
00189 }
```

5.4.3.2 runDEThreaded()

```
template<class T >
int Experiment::runDEThreaded (
            DEParams< T > deParams,
            mdata::DataTable< double > * tTable,
            size_t tRow,
            size_t tCol )
```

Executes a single iteration of a test with the given parameters.

Template Parameters

T	The data type used by the test

Parameters

tParams	The parameters used to set up the test

Returns

int An error code if any

Definition at line 491 of file experiment.cpp.

References mfunc::Count, mfunc::DEParams< T >::crFactor, mfunc::GAParams< T >::crProb, mfunc::GAParams< T >::elitismRate, mfunc::GAParams< T >::generations, mfunc::DEParams< T >::generations, util::IniReader::getEntry(), util::IniReader::getEntryAs(), INI_DIFFEVO_CRPROB, INI_DIFFEVO_GENERATIONS, INI_DIFFEVO_SCALEF1, INI_DIFFEVO_SCALEF2, INI_DIFFEVO_SECTION, INI_DIFFEVO_STRATEGY, INI_GENALG_CRPROB, INI_GENALG_ELITISMRATE, INI_GENALG_GENERATIONS, INI_GENALG_MUTPREC, INI_GENALG_MUTPROB, INI_GENALG_MUTRANGE, INI_GENALG_SECTION, mfunc::DEParams< T >::mainPop, mfunc::RandomBounds< T >::max, mfunc::RandomBounds< T >::min, mfunc::GAParams< T >::mutPrec, mfunc::GAParams< T >::mutProb, mfunc::GAParams< T >::mutRange, mfunc::DEParams< T >::nextPop, mfunc::NUM_FUNCTIONS, mfunc::DifferentialEvolution< T >::run(), mfunc::DEParams< T >::scalingFactor1, mfunc::DEParams< T >::scalingFactor2, mdata::DataTable< T >::setEntry(), and mfunc::DEParams< T >::strategy.

```
00492 {
00493     // Retrieve the next two population objects from the population pool
00494     mdata::Population<T>* popMain = popPoolRemove();
00495     mdata::Population<T>* popNext = popPoolRemove();
00496     deParams.mainPop = popMain;
00497     deParams.nextPop = popNext;
00498 
00499     high_resolution_clock::time_point t_start = high_resolution_clock::now();
00500 
00501     // Run 1 iteration of the differential evolution algorithm
00502     DifferentialEvolution<T> deAlg;
00503     int retVal = deAlg.run(deParams);
00504 
00505     high_resolution_clock::time_point t_end = high_resolution_clock::now();
00506     double execTimeMs = static_cast<double>(duration_cast<nanoseconds>(t_end - t_start).count()) / 1000000.
      0;
00507 
00508     // Record execution time
00509     if (tTable != nullptr)
00510         tTable->setEntry(tRow, tCol, execTimeMs);
00511 
00512     popPoolAdd(popNext);
00513     popPoolAdd(popMain);
00514 
00515     return retVal;
00516 }
```

5.4.3.3 runGAThreaded()

```
template<class T >
int Experiment::runGAThreaded (
              GAParams< T > gaParams,
              mdata::DataTable< double > * tTable,
              size_t tRow,
              size_t tCol )
```

Executes a single iteration of a test with the given parameters.

Template Parameters

T	The data type used by the test

Parameters

tParams	The parameters used to set up the test

Returns

int An error code if any

Definition at line 338 of file experiment.cpp.

References mfunc::GAParams< T >::auxPop, mfunc::GAParams< T >::mainPop, mfunc::GeneticAlgorithm< T >::run(), and mdata::DataTable< T >::setEntry().

```
00339 {
00340     // Retrieve the next two population objects from the population pool
00341     mdata::Population<T>* popMain = popPoolRemove();
00342     mdata::Population<T>* popAux = popPoolRemove();
```

```
00343     gaParams.mainPop = popMain;
00344     gaParams.auxPop = popAux;
00345 
00346     high_resolution_clock::time_point t_start = high_resolution_clock::now();
00347 
00348     // Run 1 iteration of the genetic algorithm
00349     GeneticAlgorithm<T> gaAlg;
00350     int retVal = gaAlg.run(gaParams);
00351 
00352     high_resolution_clock::time_point t_end = high_resolution_clock::now();
00353     double execTimeMs = static_cast<double>(duration_cast<nanoseconds>(t_end - t_start).count()) / 1000000.
      0;
00354 
00355     // Record execution time
00356     if (tTable != nullptr)
00357         tTable->setEntry(tRow, tCol, execTimeMs);
00358 
00359     popPoolAdd(popAux);
00360     popPoolAdd(popMain);
00361 
00362     return retVal;
00363 }
```

5.4.3.4 testAllFunc()

```
template<class T >
int Experiment::testAllFunc ( )
```

Executes all functions as specified in the CS471 project 3 document, records results, and outputs the data as a ∗.csv file.

Returns

Returns 0 on success. Returns a non-zero error code on failure.

Definition at line 198 of file experiment.cpp.

References mfunc::DifferentialEvolution, mfunc::GeneticAlgorithm, mfunc::Experiment< T >::testAllFunc_DE(), and mfunc::Experiment< T >::testAllFunc_GA().

Referenced by runExp().

```
00199 {
00200     switch (testAlg)
00201     {
00202         case Algorithm::GeneticAlgorithm:
00203             return testAllFunc_GA();
00204         case Algorithm::DifferentialEvolution:
00205             return testAllFunc_DE();
00206         default:
00207             return 1;
00208     }
00209 }
```

5.4.3.5 testAllFunc_DE()

```
template<class T >
int Experiment::testAllFunc_DE ( )
```

Tests all 18 functions using the differential evolution algorithm and outputs results.

Returns

int Non-zero error code on failure

Definition at line 371 of file experiment.cpp.

References mdata::DataTable< T >::clearData(), mfunc::DEParams< T >::crFactor, ThreadPool::enqueue(), mdata::DataTable< T >::exportCSV(), mfunc::DEParams< T >::fitnessTable, mfunc::DEParams< T >::fitTableCol, mfunc::DEParams< T >::fMaxBound, mfunc::DEParams< T >::fMinBound, mfunc::DEParams< T >::fPtr, mfunc::DEParams< T >::generations, mfunc::Functions< T >::get(), mfunc::DEParams< T >::mainPop, mfunc::DEParams< T >::nextPop, mfunc::NUM_FUNCTIONS, mfunc::DEParams< T >::scalingFactor1, mfunc::DEParams< T >::scalingFactor2, mdata::DataTable< T >::setColLabel(), ThreadPool::stopAndJoinAll(), and mfunc::DEParams< T >::strategy.

Referenced by mfunc::Experiment< T >::testAllFunc().

```
00372 {
00373     if (populationsPool.size() == 0) return 1;
00374 
00375     // Load DE specific parameters from ini file
00376     DEParams<T> _p;
00377     if (!loadDEParams(_p)) return 2;
00378 
00379     // Use those parameters as a template
00380     const DEParams<T>& paramTemplate = _p;
00381 
00382     // Create results and times tables
00383     mdata::DataTable<T> resultsTable(paramTemplate.
      generations, iterations);
00384     mdata::DataTable<double> execTimesTable(
      NUM_FUNCTIONS, iterations);
00385 
00386     // Set table column labels
00387     for (unsigned int c = 0; c < iterations; c++)
00388         resultsTable.setColLabel(c, std::string("Exp_") + std::to_string(c + 1));
00389 
00390     for (unsigned int c = 0; c < iterations; c++)
00391         execTimesTable.setColLabel(c, std::string("Exp_") + std::to_string(c + 1));
00392 
00393     std::vector<std::future<int>> testFutures;
00394 
00395     high_resolution_clock::time_point t_start = high_resolution_clock::now();
00396 
00397     // Run each function a number of iterations
00398     for (unsigned int f = 1; f <= mfunc::NUM_FUNCTIONS; f++)
00399     {
00400         // Reset results table
00401         resultsTable.clearData();
00402 
00403         // Queue up all iterations for current function in thread pool
00404         for (size_t exp = 0; exp < iterations; exp++)
00405         {
00406             // Set up genetic alg parameters
00407             DEParams<T> deParams;
00408             deParams.fitnessTable = &resultsTable;
00409             deParams.fitTableCol = exp;
00410             deParams.mainPop = nullptr;
00411             deParams.nextPop = nullptr;
00412             deParams.fPtr = Functions<T>::get(f);
00413             deParams.fMinBound = vBounds[f-1].min;
00414             deParams.fMaxBound = vBounds[f-1].max;
00415             deParams.generations = paramTemplate.generations;
00416             deParams.crFactor = paramTemplate.crFactor;
00417             deParams.scalingFactor1 = paramTemplate.scalingFactor1;
00418             deParams.scalingFactor2 = paramTemplate.scalingFactor2;
00419             deParams.strategy = paramTemplate.strategy;
```

```
00420
00421             // Add iteration to thread pool
00422             testFutures.emplace_back(
00423                 tPool->enqueue(&Experiment<T>::runDEThreaded, this,
      deParams, &execTimesTable, f - 1, exp)
00424             );
00425         }
00426
00427         // Join all thread futures and get result
00428         for (size_t futIndex = 0; futIndex < testFutures.size(); futIndex++)
00429         {
00430             auto& curFut = testFutures[futIndex];
00431
00432             if (!curFut.valid())
00433             {
00434                 // An error occured with one of the threads
00435                 cerr << "Error: Thread future invalid.";
00436                 tPool->stopAndJoinAll();
00437                 return 1;
00438             }
00439
00440             int errCode = curFut.get();
00441             if (errCode)
00442             {
00443                 // An error occurred while running the task.
00444                 // Bail out of function
00445                 tPool->stopAndJoinAll();
00446                 return errCode;
00447             }
00448         }
00449
00450         // Clear thread futures
00451         testFutures.clear();
00452
00453         // Output results for current function
00454         std::string outFile = resultsFile;
00455         outFile = std::regex_replace(outFile, std::regex("\\%ALG%"), "DE");
00456         outFile = std::regex_replace(outFile, std::regex("\\%FUNC%"), std::to_string(f));
00457
00458         if (!outFile.empty())
00459         {
00460             resultsTable.exportCSV(outFile.c_str());
00461             cout << "Exported function results to: " << outFile << endl << flush;
00462         }
00463     }
00464
00465     // Output total execution times for each function iteration
00466     std::string timesFile = execTimesFile;
00467     timesFile = std::regex_replace(timesFile, std::regex("\\%ALG%"), "DE");
00468
00469     if (!execTimesFile.empty())
00470     {
00471         execTimesTable.exportCSV(timesFile.c_str());
00472         cout << "Exported execution times to: " << timesFile << endl << flush;
00473     }
00474
00475     high_resolution_clock::time_point t_end = high_resolution_clock::now();
00476     long double totalExecTime = static_cast<long double>(duration_cast<nanoseconds>(t_end - t_start).count(
      )) / 1000000000.0L;
00477
00478     cout << endl << "Test finished. Total time: " << std::setprecision(7) << totalExecTime << " seconds." <
      < endl;
00479
00480     return 0;
00481 }
```

5.4.3.6 testAllFunc_GA()

```
template<class T >
int Experiment::testAllFunc_GA ( )
```

Tests all 18 functions using the genetic algorithm and outputs results.

Returns

int Non-zero error code on failure

Definition at line 217 of file experiment.cpp.

References mfunc::GAParams< T >::auxPop, mdata::DataTable< T >::clearData(), mfunc::GAParams< T >::crProb, mfunc::GAParams< T >::elitismRate, ThreadPool::enqueue(), mdata::DataTable< T >::exportCSV(), mfunc::GAParams< T >::fitnessTable, mfunc::GAParams< T >::fitTableCol, mfunc::GAParams< T >::fMaxBound, mfunc::GAParams< T >::fMinBound, mfunc::GAParams< T >::fPtr, mfunc::GAParams< T >::generations, mfunc::Functions< T >::get(), mfunc::GAParams< T >::mainPop, mfunc::GAParams< T >::mutPrec, mfunc::GAParams< T >::mutProb, mfunc::GAParams< T >::mutRange, mfunc::NUM_FUNCTIONS, mdata::DataTable< T >::setColLabel(), and ThreadPool::stopAndJoinAll().

Referenced by mfunc::Experiment< T >::testAllFunc().

```
00218 {
00219     if (populationsPool.size() == 0) return 1;
00220 
00221     // Load genetic algorithm specific parameters from ini file
00222     GAParams<T> _p;
00223     if (!loadGAParams(_p)) return 2;
00224 
00225     // Use those parameters as a template
00226     const GAParams<T>& paramTemplate = _p;
00227 
00228     // Create results and times tables
00229     mdata::DataTable<T> resultsTable(paramTemplate.
      generations, iterations);
00230     mdata::DataTable<double> execTimesTable(
      NUM_FUNCTIONS, iterations);
00231 
00232     // Set table column labels
00233     for (unsigned int c = 0; c < iterations; c++)
00234         resultsTable.setColLabel(c, std::string("Exp_") + std::to_string(c + 1));
00235 
00236     for (unsigned int c = 0; c < iterations; c++)
00237         execTimesTable.setColLabel(c, std::string("Exp_") + std::to_string(c + 1));
00238 
00239     std::vector<std::future<int>> testFutures;
00240 
00241     high_resolution_clock::time_point t_start = high_resolution_clock::now();
00242 
00243     // Run each function a number of iterations
00244     for (unsigned int f = 1; f <= mfunc::NUM_FUNCTIONS; f++)
00245     {
00246         // Reset results table
00247         resultsTable.clearData();
00248 
00249         // Queue up all iterations for current function in thread pool
00250         for (size_t exp = 0; exp < iterations; exp++)
00251         {
00252             // Set up genetic alg parameters
00253             GAParams<T> gaParams;
00254             gaParams.fitnessTable = &resultsTable;
00255             gaParams.fitTableCol = exp;
00256             gaParams.mainPop = nullptr;
00257             gaParams.auxPop = nullptr;
00258             gaParams.fPtr = Functions<T>::get(f);
00259             gaParams.fMinBound = vBounds[f-1].min;
00260             gaParams.fMaxBound = vBounds[f-1].max;
00261             gaParams.generations = paramTemplate.generations;
00262             gaParams.crProb = paramTemplate.crProb;
00263             gaParams.mutProb = paramTemplate.mutProb;
00264             gaParams.mutRange = paramTemplate.mutRange;
00265             gaParams.mutPrec = paramTemplate.mutPrec;
00266             gaParams.elitismRate = paramTemplate.elitismRate;
00267 
00268             // Add iteration to thread pool
00269             testFutures.emplace_back(
00270                 tPool->enqueue(&Experiment<T>::runGAThreaded, this,
      gaParams, &execTimesTable, f - 1, exp)
00271             );
00272         }
00273 
00274         // Join all thread futures and get result
00275         for (size_t futIndex = 0; futIndex < testFutures.size(); futIndex++)
00276         {
00277             auto& curFut = testFutures[futIndex];
```

```
00278
00279             if (!curFut.valid())
00280             {
00281                 // An error occured with one of the threads
00282                 cerr << "Error: Thread future invalid.";
00283                 tPool->stopAndJoinAll();
00284                 return 1;
00285             }
00286
00287             int errCode = curFut.get();
00288             if (errCode)
00289             {
00290                 // An error occurred while running the task.
00291                 // Bail out of function
00292                 tPool->stopAndJoinAll();
00293                 return errCode;
00294             }
00295         }
00296
00297         // Clear thread futures
00298         testFutures.clear();
00299
00300         // Output results for current function
00301         std::string outFile = resultsFile;
00302         outFile = std::regex_replace(outFile, std::regex("\\%ALG%"), "GA");
00303         outFile = std::regex_replace(outFile, std::regex("\\%FUNC%"), std::to_string(f));
00304
00305         if (!outFile.empty())
00306         {
00307             resultsTable.exportCSV(outFile.c_str());
00308             cout << "Exported function results to: " << outFile << endl << flush;
00309         }
00310     }
00311
00312     // Output total execution times for each function iteration
00313     std::string timesFile = execTimesFile;
00314     timesFile = std::regex_replace(timesFile, std::regex("\\%ALG%"), "GA");
00315
00316     if (!execTimesFile.empty())
00317     {
00318         execTimesTable.exportCSV(timesFile.c_str());
00319         cout << "Exported execution times to: " << timesFile << endl << flush;
00320     }
00321
00322     high_resolution_clock::time_point t_end = high_resolution_clock::now();
00323     long double totalExecTime = static_cast<long double>(duration_cast<nanoseconds>(t_end - t_start).count(
    )) / 1000000000.0L;
00324
00325     cout << endl << "Test finished. Total time: " << std::setprecision(7) << totalExecTime << " seconds." <
    < endl;
00326
00327     return 0;
00328 }
```

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

- include/experiment.h
- src/experiment.cpp

5.5 mfunc::FunctionDesc Struct Reference

get() returns a function's description Returns a C-string description for the given function id if the id is valid. Otherwise returns null

`#include <mfunctions.h>`

Static Public Member Functions

- static const char * get (unsigned int f)

5.5.1 Detailed Description

get() returns a function's description Returns a C-string description for the given function id if the id is valid. Otherwise returns null

Parameters

f	Function id to retrieve the description for

Returns

A C-string containing the function description if id is valid, otherwise null.

Definition at line 76 of file mfunctions.h.

5.5.2 Member Function Documentation

5.5.2.1 get()

```
static const char* mfunc::FunctionDesc::get (
            unsigned int f )  [inline], [static]
```

Definition at line 78 of file mfunctions.h.

References _ackleysOneDesc, _ackleysOneId, _ackleysTwoDesc, _ackleysTwoId, _alpineDesc, _alpineId, _dejongDesc, _dejongId, _eggHolderDesc, _eggHolderId, _griewangkDesc, _griewangkId, _levyDesc, _levyId, _mastersCosineWaveDesc, _mastersCosineWaveId, _michalewiczDesc, _michalewiczId, _pathologicalDesc, _pathologicalId, _quarticDesc, _quarticId, _ranaDesc, _ranaId, _rastriginDesc, _rastriginId, _rosenbrokDesc, _rosenbrokId, _schwefelDesc, _schwefelId, _sineEnvelopeSineWaveDesc, _sineEnvelopeSineWaveId, _stepDesc, _stepId, _stretchedVSineWaveDesc, and _stretchedVSineWaveId.

```
00079         {
00080             switch (f)
00081             {
00082                 case _schwefelId:
00083                     return _schwefelDesc;
00084                 case _dejongId:
00085                     return _dejongDesc;
00086                 case _rosenbrokId:
00087                     return _rosenbrokDesc;
00088                 case _rastriginId:
00089                     return _rastriginDesc;
00090                 case _griewangkId:
00091                     return _griewangkDesc;
00092                 case _sineEnvelopeSineWaveId:
00093                     return _sineEnvelopeSineWaveDesc;
00094                 case _stretchedVSineWaveId:
00095                     return _stretchedVSineWaveDesc;
00096                 case _ackleysOneId:
00097                     return _ackleysOneDesc;
00098                 case _ackleysTwoId:
00099                     return _ackleysTwoDesc;
00100                 case _eggHolderId:
00101                     return _eggHolderDesc;
00102                 case _ranaId:
00103                     return _ranaDesc;
00104                 case _pathologicalId:
00105                     return _pathologicalDesc;
00106                 case _michalewiczId:
00107                     return _michalewiczDesc;
00108                 case _mastersCosineWaveId:
00109                     return _mastersCosineWaveDesc;
00110                 case _quarticId:
00111                     return _quarticDesc;
00112                 case _levyId:
00113                     return _levyDesc;
00114                 case _stepId:
00115                     return _stepDesc;
00116                 case _alpineId:
00117                     return _alpineDesc;
00118                 default:
00119                     return NULL;
00120             }
00121         }
```

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

- include/mfunctions.h

5.6 mfunc::Functions< T > Struct Template Reference

Struct containing all static math functions. A function can be called directly by name, or indirectly using Functions::get or Functions::exec.

```
#include <mfunctions.h>
```

Static Public Member Functions

- static T schwefel (T *v, size_t n)

  Function 1. Implementation of Schwefel's mathematical function.
- static T dejong (T *v, size_t n)

  Function 2. Implementation of 1st De Jong's mathematical function.
- static T rosenbrok (T *v, size_t n)

  Function 3. Implementation of the Rosenbrock mathematical function.
- static T rastrigin (T *v, size_t n)

  Function 4. Implementation of the Rastrigin mathematical function.
- static T griewangk (T *v, size_t n)

  Function 5. Implementation of the Griewangk mathematical function.
- static T sineEnvelopeSineWave (T *v, size_t n)

  Function 6. Implementation of the Sine Envelope Sine Wave mathematical function.
- static T stretchedVSineWave (T *v, size_t n)

  Function 7. Implementation of the Stretched V Sine Wave mathematical function.
- static T ackleysOne (T *v, size_t n)

  Function 8. Implementation of Ackley's One mathematical function.
- static T ackleysTwo (T *v, size_t n)

  Function 9. Implementation of Ackley's Two mathematical function.
- static T eggHolder (T *v, size_t n)

  Function 10. Implementation of the Egg Holder mathematical function.
- static T rana (T *v, size_t n)

  Function 11. Implementation of the Rana mathematical function.
- static T pathological (T *v, size_t n)

  Function 12. Implementation of the Pathological mathematical function.
- static T mastersCosineWave (T *v, size_t n)

  Function 14. Implementation of the Masters Cosine Wave mathematical function.
- static T michalewicz (T *v, size_t n)

  Function 13. Implementation of the Michalewicz mathematical function.
- static T quartic (T *v, size_t n)

  Function 15. Implementation of the Quartic mathematical function.
- static T levy (T *v, size_t n)

  Function 16. Implementation of the Levy mathematical function.
- static T step (T *v, size_t n)

  Function 17. Implementation of the Step mathematical function.
- static T alpine (T *v, size_t n)

Function 18. Implementation of the Alpine mathematical function.

- static mfuncPtr< T > get (unsigned int f)

  Returns a function pointer to the math function with the given id.
- static bool exec (unsigned int f, T *v, size_t n, T &outResult)

  Executes a specific function Executes the function with the given id and returns true on success. Otherwise returns false if id is invalid.
- static T nthroot (T x, T n)
- static T w (T x)
- static size_t getCallCounter (unsigned int f)

  Returns the number of times the specified function id has been executed.
- static void resetCallCounters ()

  Resets all function call counters to zero.

5.6.1 Detailed Description

template<class T>
struct mfunc::Functions< T >

Struct containing all static math functions. A function can be called directly by name, or indirectly using Functions::get or Functions::exec.

Template Parameters

T	Data type for function calculations

Definition at line 132 of file mfunctions.h.

5.6.2 Member Function Documentation

5.6.2.1 ackleysOne()

```
template<class T >
T mfunc::Functions< T >::ackleysOne (
            T * v,
            size_t n )  [static]
```

Function 8. Implementation of Ackley's One mathematical function.

Parameters

v	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 385 of file mfunctions.h.

References _ackleysOneId.

```
00386 {
00387     fCounterInc(_ackleysOneId);
00388
00389     T f = 0.0;
00390
00391     for (size_t i = 0; i < n - 1; i++)
00392     {
00393         T a = (static_cast<T>(1.0) / std::pow(static_cast<T>(M_E), static_cast<T>(0.2))) * std::sqrt(v[i]*v
      [i] + v[i+1]*v[i+1]);
00394         T b = static_cast<T>(3.0) * (std::cos(static_cast<T>(2.0) * v[i]) + std::sin(static_cast<T>(2.0) *
      v[i+1]));
00395         f += a + b;
00396     }
00397
00398     return f;
00399 }
```

5.6.2.2 ackleysTwo()

```
template<class T >
T mfunc::Functions< T >::ackleysTwo (
            T * v,
            size_t n )  [static]
```

Function 9. Implementation of Ackley's Two mathematical function.

Parameters

v	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 411 of file mfunctions.h.

References _ackleysTwoId.

```
00412 {
00413     fCounterInc(_ackleysTwoId);
00414
00415     T f = 0.0;
00416
00417     for (size_t i = 0; i < n - 1; i++)
00418     {
00419         T a = static_cast<T>(20.0) / std::pow(static_cast<T>(M_E), static_cast<T>(0.2) * std::sqrt((v[i]*v[
      i] + v[i+1]*v[i+1]) / static_cast<T>(2.0)));
00420         T b = std::pow(static_cast<T>(M_E), static_cast<T>(0.5) *
00421             (std::cos(static_cast<T>(2.0) * static_cast<T>(M_PI) * v[i]) + std::cos(static_cast<T>(2.0) *
      static_cast<T>(M_PI) * v[i+1])));
```

```
00422         f += static_cast<T>(20.0) + static_cast<T>(M_E) - a - b;
00423     }
00424
00425     return f;
00426 }
```

5.6.2.3 alpine()

```
template<class T >
T mfunc::Functions< T >::alpine (
            T * v,
            size_t n )  [static]
```

Function 18. Implementation of the Alpine mathematical function.

Parameters

v	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 659 of file mfunctions.h.

References _alpineId.

```
00660 {
00661     fCounterInc(_alpineId);
00662
00663     T f = 0.0;
00664
00665     for (size_t i = 0; i < n; i++)
00666     {
00667         f += std::abs(v[i] * std::sin(v[i]) + static_cast<T>(0.1)*v[i]);
00668     }
00669
00670     return f;
00671 }
```

5.6.2.4 dejong()

```
template<class T >
T mfunc::Functions< T >::dejong (
            T * v,
            size_t n )  [static]
```

Function 2. Implementation of 1st De Jong's mathematical function.

Parameters

v	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 225 of file mfunctions.h.

References _dejongId.

```
00226 {
00227     fCounterInc(_dejongId);
00228
00229     T f = 0.0;
00230
00231     for (size_t i = 0; i < n; i++)
00232     {
00233         f += v[i] * v[i];
00234     }
00235
00236     return f;
00237 }
```

5.6.2.5 eggHolder()

```
template<class T >
T mfunc::Functions< T >::eggHolder (
            T * v,
            size_t n )  [static]
```

Function 10. Implementation of the Egg Holder mathematical function.

Parameters

v	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 438 of file mfunctions.h.

References _eggHolderId.

```
00439 {
00440     fCounterInc(_eggHolderId);
00441
00442     T f = 0.0;
00443
```

```
00444     for (size_t i = 0; i < n - 1; i++)
00445     {
00446         T a = static_cast<T>(-1.0) * v[i] * std::sin(std::sqrt(std::abs(v[i] - v[i+1] - static_cast<T>(47.0
      ))));
00447         T b = (v[i+1] + static_cast<T>(47)) * std::sin(std::sqrt(std::abs(v[i+1] + static_cast<T>(47.0) + (
      v[i]/static_cast<T>(2.0)))));
00448         f += a - b;
00449     }
00450
00451     return f;
00452 }
```

5.6.2.6 exec()

```
template<class T >
bool mfunc::Functions< T >::exec (
            unsigned int f,
            T * v,
            size_t n,
            T & outResult )  [static]
```

Executes a specific function Executes the function with the given id and returns true on success. Otherwise returns false if id is invalid.

Parameters

f	Function id to execute
v	Vector as a T value array
n	Size of the vector 'v'
outResult	Output reference variable for the result of the mathematical function

Returns

true if 'f' is a valid id and the function was ran. Otherwise false.

Definition at line 743 of file mfunctions.h.

```
00744 {
00745     auto fPtr = get(f);
00746     if (fPtr == nullptr) return false;
00747
00748     outResult = fPtr(v, n);
00749     return true;
00750 }
```

5.6.2.7 get()

```
template<class T >
mfunc::mfuncPtr< T > mfunc::Functions< T >::get (
            unsigned int f )  [static]
```

Returns a function pointer to the math function with the given id.

Template Parameters

T	Data type to be used in the function's calculations

Parameters

f	Id of the function (1-18)

Returns

mfunc::mfuncPtr<T> Function pointer to the associated function, or nullptr if the id is invalid.

Definition at line 685 of file mfunctions.h.

References _ackleysOneId, _ackleysTwoId, _alpineId, _dejongId, _eggHolderId, _griewangkId, _levyId, _mastersCosineWaveId, _michalewiczId, _pathologicalId, _quarticId, _ranaId, _rastriginId, _rosenbrokId, _schwefelId, _sineEnvelopeSineWaveId, _stepId, and _stretchedVSineWaveId.

Referenced by mfunc::Experiment< T >::testAllFunc_DE(), and mfunc::Experiment< T >::testAllFunc_GA().

```
00686 {
00687     switch (f)
00688     {
00689         case _schwefelId:
00690             return Functions<T>::schwefel;
00691         case _dejongId:
00692             return Functions<T>::dejong;
00693         case _rosenbrokId:
00694             return Functions<T>::rosenbrok;
00695         case _rastriginId:
00696             return Functions<T>::rastrigin;
00697         case _griewangkId:
00698             return Functions<T>::griewangk;
00699         case _sineEnvelopeSineWaveId:
00700             return Functions<T>::sineEnvelopeSineWave;
00701         case _stretchedVSineWaveId:
00702             return Functions<T>::stretchedVSineWave;
00703         case _ackleysOneId:
00704             return Functions<T>::ackleysOne;
00705         case _ackleysTwoId:
00706             return Functions<T>::ackleysTwo;
00707         case _eggHolderId:
00708             return Functions<T>::eggHolder;
00709         case _ranaId:
00710             return Functions<T>::rana;
00711         case _pathologicalId:
00712             return Functions<T>::pathological;
00713         case _michalewiczId:
00714             return Functions<T>::michalewicz;
00715         case _mastersCosineWaveId:
00716             return Functions<T>::mastersCosineWave;
00717         case _quarticId:
00718             return Functions<T>::quartic;
00719         case _levyId:
00720             return Functions<T>::levy;
00721         case _stepId:
00722             return Functions<T>::step;
00723         case _alpineId:
00724             return Functions<T>::alpine;
00725         default:
00726             return nullptr;
00727     }
00728 }
```

5.6.2.8 getCallCounter()

```
template<class T >
size_t mfunc::Functions< T >::getCallCounter (
            unsigned int f )  [static]
```

Returns the number of times the specified function id has been executed.

Returns

size_t Number of times the given function id has been executed

Definition at line 758 of file mfunctions.h.

References _NUM_FUNCTIONS.

```
00759 {
00760     if (f == 0 || f > _NUM_FUNCTIONS)
00761         return 0;
00762
00763     return fCallCounters[f - 1];
00764 }
```

5.6.2.9 griewangk()

```
template<class T >
T mfunc::Functions< T >::griewangk (
            T * v,
            size_t n )  [static]
```

Function 5. Implementation of the Griewangk mathematical function.

Parameters

v	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 300 of file mfunctions.h.

References _griewangkId.

```
00301 {
00302     fCounterInc(_griewangkId);
00303
00304     T sum = 0.0;
00305     T product = 0.0;
00306
00307     for (size_t i = 0; i < n; i++)
```

```
00308     {
00309         sum += (v[i] * v[i]) / static_cast<T>(4000.0);
00310     }
00311
00312     for (size_t i = 0; i < n; i++)
00313     {
00314         product *= std::cos(v[i] / std::sqrt(static_cast<T>(i + 1.0)));
00315     }
00316
00317     return static_cast<T>(1.0) + sum - product;
00318 }
```

5.6.2.10 levy()

```
template<class T >
T mfunc::Functions< T >::levy (
            T * v,
            size_t n )  [static]
```

Function 16. Implementation of the Levy mathematical function.

Parameters

v	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 601 of file mfunctions.h.

References _levyId.

```
00602 {
00603     fCounterInc(_levyId);
00604
00605     T f = 0.0;
00606
00607     for (size_t i = 0; i < n - 1; i++)
00608     {
00609         T a = w(v[i]) - static_cast<T>(1.0);
00610         a *= a;
00611         T b = std::sin(static_cast<T>(M_PI) * w(v[i]) + static_cast<T>(1.0));
00612         b *= b;
00613         T c = w(v[n - 1]) - static_cast<T>(1.0);
00614         c *= c;
00615         T d = std::sin(static_cast<T>(2.0) * static_cast<T>(M_PI) * w(v[n - 1]));
00616         d *= d;
00617         f += a * (static_cast<T>(1.0) + static_cast<T>(10.0) * b) + c * (static_cast<T>(1.0) + d);
00618     }
00619
00620     T e = std::sin(static_cast<T>(M_PI) * w(v[0]));
00621     return e*e + f;
00622 }
```

5.6.2.11 mastersCosineWave()

```
template<class T >
T mfunc::Functions< T >::mastersCosineWave (
            T * v,
            size_t n )  [static]
```

Function 14. Implementation of the Masters Cosine Wave mathematical function.

Parameters

v	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 542 of file mfunctions.h.

References _mastersCosineWaveId.

```
00543 {
00544     fCounterInc(_mastersCosineWaveId);
00545
00546     T f = 0.0;
00547
00548     for (size_t i = 0; i < n - 1; i++)
00549     {
00550         T a = std::pow(M_E, static_cast<T>(-1.0/8.0)*(v[i]*v[i] + v[i+1]*v[i+1] + static_cast<T>(0.5)*v[i+1
    ]*v[i]));
00551         T b = std::cos(static_cast<T>(4) * std::sqrt(v[i]*v[i] + v[i+1]*v[i+1] + static_cast<T>(0.5)*v[i]*v
    [i+1]));
00552         f += a * b;
00553     }
00554
00555     return static_cast<T>(-1.0) * f;
00556 }
```

5.6.2.12 michalewicz()

```
template<class T >
T mfunc::Functions< T >::michalewicz (
            T * v,
            size_t n )  [static]
```

Function 13. Implementation of the Michalewicz mathematical function.

Parameters

v	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 518 of file mfunctions.h.

References _michalewiczId.

```
00519 {
00520     fCounterInc(_michalewiczId);
00521
00522     T f = 0.0;
00523
00524     for (size_t i = 0; i < n; i++)
00525     {
00526         f += std::sin(v[i]) * std::pow(std::sin(((i+1) * v[i] * v[i]) / static_cast<T>(M_PI)),
      static_cast<T>(20));
00527     }
00528
00529     return -1.0 * f;
00530 }
```

5.6.2.13 nthroot()

```
template<class T >
T mfunc::Functions< T >::nthroot (
            T x,
            T n )  [static]
```

Simple helper function that returns the nth-root

Parameters

x	Value to be taken to the nth power
n	root degree

Returns

The value of the nth-root of x

Definition at line 186 of file mfunctions.h.

```
00187 {
00188     return std::pow(x, static_cast<T>(1.0) / n);
00189 }
```

5.6.2.14 pathological()

```
template<class T >
T mfunc::Functions< T >::pathological (
            T * v,
            size_t n )  [static]
```

Function 12. Implementation of the Pathological mathematical function.

Parameters

v	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 490 of file mfunctions.h.

References _pathologicalId.

```
00491 {
00492     fCounterInc(_pathologicalId);
00493
00494     T f = 0.0;
00495
00496     for (size_t i = 0; i < n - 1; i++)
00497     {
00498         T a = std::sin(std::sqrt(static_cast<T>(100.0)*v[i]*v[i] + v[i+1]*v[i+1]));
00499         a = (a*a) - static_cast<T>(0.5);
00500         T b = (v[i]*v[i] - static_cast<T>(2)*v[i]*v[i+1] + v[i+1]*v[i+1]);
00501         b = static_cast<T>(1.0) + static_cast<T>(0.001) * b*b;
00502         f += static_cast<T>(0.5) + (a/b);
00503     }
00504
00505     return f;
00506 }
```

5.6.2.15 quartic()

```
template<class T >
T mfunc::Functions< T >::quartic (
            T * v,
            size_t n )  [static]
```

Function 15. Implementation of the Quartic mathematical function.

Parameters

v	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 568 of file mfunctions.h.

References _quarticId.

```
00569 {
00570     fCounterInc(_quarticId);
00571
00572     T f = 0.0;
00573
00574     for (size_t i = 0; i < n; i++)
00575     {
00576         f += (i+1) * v[i] * v[i] * v[i] * v[i];
00577     }
00578
00579     return f;
00580 }
```

5.6.2.16 rana()

```
template<class T >
T mfunc::Functions< T >::rana (
            T * v,
            size_t n )  [static]
```

Function 11. Implementation of the Rana mathematical function.

Parameters

v	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 464 of file mfunctions.h.

References _ranaId.

```
00465 {
00466     fCounterInc(_ranaId);
00467
00468     T f = 0.0;
00469
00470     for (size_t i = 0; i < n - 1; i++)
00471     {
00472         T a = v[i] * std::sin(std::sqrt(std::abs(v[i+1] - v[i] + static_cast<T>(1.0)))) * std::cos(
      std::sqrt(std::abs(v[i+1] + v[i] + static_cast<T>(1.0))));
00473         T b = (v[i+1] + static_cast<T>(1.0)) * std::cos(std::sqrt(std::abs(v[i+1] - v[i] + static_cast<T>(1
      .0)))) * std::sin(std::sqrt(std::abs(v[i+1] + v[i] + static_cast<T>(1.0))));
00474         f += a + b;
00475     }
00476
00477     return f;
00478 }
```

5.6.2.17 rastrigin()

```
template<class T >
T mfunc::Functions< T >::rastrigin (
            T * v,
            size_t n )  [static]
```

Function 4. Implementation of the Rastrigin mathematical function.

Parameters

v	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 276 of file mfunctions.h.

References _rastriginId.

```
00277 {
00278     fCounterInc(_rastriginId);
00279
00280     T f = 0.0;
00281
00282     for (size_t i = 0; i < n; i++)
00283     {
00284         f += (v[i] * v[i]) - (static_cast<T>(10.0) * std::cos(static_cast<T>(2.0) * static_cast<T>(M_PI) *
      v[i]));
00285     }
00286
00287     return static_cast<T>(10.0) * static_cast<T>(n) * f;
00288 }
```

5.6.2.18 resetCallCounters()

```
template<class T >
void mfunc::Functions< T >::resetCallCounters ( )  [static]
```

Resets all function call counters to zero.

Definition at line 770 of file mfunctions.h.

References _NUM_FUNCTIONS.

```
00771 {
00772     for (size_t i = 0; i < _NUM_FUNCTIONS; i++)
00773         fCallCounters[i] = 0;
00774 }
```

5.6.2.19 rosenbrok()

```
template<class T >
T mfunc::Functions< T >::rosenbrok (
            T * v,
            size_t n )  [static]
```

Function 3. Implementation of the Rosenbrock mathematical function.

Parameters

v	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 249 of file mfunctions.h.

References _rosenbrokId.

```
00250 {
00251     fCounterInc(_rosenbrokId);
00252
00253     T f = 0.0;
00254
00255     for (size_t i = 0; i < n - 1; i++)
00256     {
00257         T a = ((v[i] * v[i]) - v[i+1]);
00258         T b = (static_cast<T>(1.0) - v[i]);
00259         f += static_cast<T>(100.0) * a * a;
00260         f += b * b;
00261     }
00262
00263     return f;
00264 }
```

5.6.2.20 schwefel()

```
template<class T >
T mfunc::Functions< T >::schwefel (
            T * v,
            size_t n )  [static]
```

Function 1. Implementation of Schwefel's mathematical function.

Parameters

v	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 201 of file mfunctions.h.

References _schwefelId.

```
00202 {
00203     fCounterInc(_schwefelId);
```

```
00204
00205     T f = 0.0;
00206
00207     for (size_t i = 0; i < n; i++)
00208     {
00209         f += (static_cast<T>(-1.0) * v[i]) * std::sin(std::sqrt(std::abs(v[i])));
00210     }
00211
00212     return (static_cast<T>(418.9829) * static_cast<T>(n)) - f;
00213 }
```

5.6.2.21 sineEnvelopeSineWave()

```
template<class T >
T mfunc::Functions< T >::sineEnvelopeSineWave (
            T * v,
            size_t n )  [static]
```

Function 6. Implementation of the Sine Envelope Sine Wave mathematical function.

Parameters

v	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 330 of file mfunctions.h.

References _sineEnvelopeSineWaveId.

```
00331 {
00332     fCounterInc(_sineEnvelopeSineWaveId);
00333
00334     T f = 0.0;
00335
00336     for (size_t i = 0; i < n - 1; i++)
00337     {
00338         T a = std::sin(v[i]*v[i] + v[i+1]*v[i+1] - static_cast<T>(0.5));
00339         a *= a;
00340         T b = (static_cast<T>(1.0) + static_cast<T>(0.001)*(v[i]*v[i] + v[i+1]*v[i+1]));
00341         b *= b;
00342         f += static_cast<T>(0.5) + (a / b);
00343     }
00344
00345     return static_cast<T>(-1.0) * f;
00346 }
```

5.6.2.22 step()

```
template<class T >
T mfunc::Functions< T >::step (
            T * v,
            size_t n )  [static]
```

Function 17. Implementation of the Step mathematical function.

Parameters

v	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 634 of file mfunctions.h.

References _stepId.

```
00635 {
00636     fCounterInc(_stepId);
00637
00638     T f = 0.0;
00639
00640     for (size_t i = 0; i < n; i++)
00641     {
00642         T a = std::abs(v[i]) + static_cast<T>(0.5);
00643         f += a * a;
00644     }
00645
00646     return f;
00647 }
```

5.6.2.23 stretchedVSineWave()

```
template<class T >
T mfunc::Functions< T >::stretchedVSineWave (
            T * v,
            size_t n )  [static]
```

Function 7. Implementation of the Stretched V Sine Wave mathematical function.

Parameters

v	Vector as a T value array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 358 of file mfunctions.h.

References _stretchedVSineWaveId.

```
00359 {
00360     fCounterInc(_stretchedVSineWaveId);
00361
00362     T f = 0.0;
```

```
00363
00364     for (size_t i = 0; i < n - 1; i++)
00365     {
00366         T a = nthroot(v[i]*v[i] + v[i+1]*v[i+1], static_cast<T>(4.0));
00367         T b = std::sin(static_cast<T>(50.0) * nthroot(v[i]*v[i] + v[i+1]*v[i+1], static_cast<T>(10.0
      )));
00368         b *= b;
00369         f += a * b + static_cast<T>(1.0);
00370     }
00371
00372     return f;
00373 }
```

5.6.2.24 w()

```
template<class T >
T mfunc::Functions< T >::w (
            T x )  [static]
```

Helper math function used in levy()

Definition at line 588 of file mfunctions.h.

```
00589 {
00590     return static_cast<T>(1.0) + (x - static_cast<T>(1.0)) / static_cast<T>(4.0);
00591 }
```

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

- include/mfunctions.h

5.7 mfunc::GAParams< T > Struct Template Reference

Simple structure that holds various parameters for the genetic algorithm.

`#include <geneticalg.h>`

Public Member Functions

- GAParams ()

Public Attributes

- mdata::DataTable< T > * fitnessTable
- size_t fitTableCol
- mdata::Population< T > * mainPop
- mdata::Population< T > * auxPop
- mfuncPtr< T > fPtr
- T fMinBound
- T fMaxBound
- unsigned int generations
- double crProb
- double mutProb
- double mutRange
- double mutPrec
- double elitismRate

5.7.1 Detailed Description

template<class T>
struct mfunc::GAParams< T >

Simple structure that holds various parameters for the genetic algorithm.

Template Parameters

T	Datatype used by the algorithm

Definition at line 31 of file geneticalg.h.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 GAParams()

```
template<class T>
mfunc::GAParams< T >::GAParams ( )  [inline]
```

Definition at line 47 of file geneticalg.h.

```
00048         {
00049             fitnessTable = nullptr;
00050             fitTableCol = 0;
00051             mainPop = nullptr;
00052             auxPop = nullptr;
00053             fPtr = nullptr;
00054             fMinBound = 0;
00055             fMaxBound = 0;
00056             generations = 0;
00057             crProb = 0;
00058             mutProb = 0;
00059             mutRange = 0;
00060             mutPrec = 0;
00061             elitismRate = 0;
00062         }
```

5.7.3 Member Data Documentation

5.7.3.1 auxPop

```
template<class T>
mdata::Population<T>* mfunc::GAParams< T >::auxPop
```

Definition at line 36 of file geneticalg.h.

Referenced by mfunc::GeneticAlgorithm< T >::run(), mfunc::Experiment< T >::runGAThreaded(), and mfunc::Experiment< T >::testAllFunc_GA().

5.7.3.2 crProb

```
template<class T>
double mfunc::GAParams< T >::crProb
```

Definition at line 41 of file geneticalg.h.

Referenced by mfunc::GeneticAlgorithm< T >::run(), mfunc::Experiment< T >::runDEThreaded(), and mfunc::Experiment< T >::testAllFunc_GA().

5.7.3.3 elitismRate

```
template<class T>
double mfunc::GAParams< T >::elitismRate
```

Definition at line 45 of file geneticalg.h.

Referenced by mfunc::GeneticAlgorithm< T >::run(), mfunc::Experiment< T >::runDEThreaded(), and mfunc::Experiment< T >::testAllFunc_GA().

5.7.3.4 fitnessTable

```
template<class T>
mdata::DataTable<T>* mfunc::GAParams< T >::fitnessTable
```

Definition at line 33 of file geneticalg.h.

Referenced by mfunc::GeneticAlgorithm< T >::run(), and mfunc::Experiment< T >::testAllFunc_GA().

5.7.3.5 fitTableCol

```
template<class T>
size_t mfunc::GAParams< T >::fitTableCol
```

Definition at line 34 of file geneticalg.h.

Referenced by mfunc::GeneticAlgorithm< T >::run(), and mfunc::Experiment< T >::testAllFunc_GA().

5.7.3.6 fMaxBound

```
template<class T>
T mfunc::GAParams< T >::fMaxBound
```

Definition at line 39 of file geneticalg.h.

Referenced by mfunc::GeneticAlgorithm< T >::run(), and mfunc::Experiment< T >::testAllFunc_GA().

5.7.3.7 fMinBound

```
template<class T>
T mfunc::GAParams< T >::fMinBound
```

Definition at line 38 of file geneticalg.h.

Referenced by mfunc::GeneticAlgorithm< T >::run(), and mfunc::Experiment< T >::testAllFunc_GA().

5.7.3.8 fPtr

```
template<class T>
mfuncPtr<T> mfunc::GAParams< T >::fPtr
```

Definition at line 37 of file geneticalg.h.

Referenced by mfunc::GeneticAlgorithm< T >::run(), and mfunc::Experiment< T >::testAllFunc_GA().

5.7.3.9 generations

```
template<class T>
unsigned int mfunc::GAParams< T >::generations
```

Definition at line 40 of file geneticalg.h.

Referenced by mfunc::GeneticAlgorithm< T >::run(), mfunc::Experiment< T >::runDEThreaded(), and mfunc::↩ Experiment< T >::testAllFunc_GA().

5.7.3.10 mainPop

```
template<class T>
mdata::Population<T>* mfunc::GAParams< T >::mainPop
```

Definition at line 35 of file geneticalg.h.

Referenced by mfunc::GeneticAlgorithm< T >::run(), mfunc::Experiment< T >::runGAThreaded(), and mfunc::↩ Experiment< T >::testAllFunc_GA().

5.7.3.11 mutPrec

```
template<class T>
double mfunc::GAParams< T >::mutPrec
```

Definition at line 44 of file geneticalg.h.

Referenced by mfunc::GeneticAlgorithm< T >::run(), mfunc::Experiment< T >::runDEThreaded(), and mfunc::Experiment< T >::testAllFunc_GA().

5.7.3.12 mutProb

```
template<class T>
double mfunc::GAParams< T >::mutProb
```

Definition at line 42 of file geneticalg.h.

Referenced by mfunc::GeneticAlgorithm< T >::run(), mfunc::Experiment< T >::runDEThreaded(), and mfunc::Experiment< T >::testAllFunc_GA().

5.7.3.13 mutRange

```
template<class T>
double mfunc::GAParams< T >::mutRange
```

Definition at line 43 of file geneticalg.h.

Referenced by mfunc::GeneticAlgorithm< T >::run(), mfunc::Experiment< T >::runDEThreaded(), and mfunc::Experiment< T >::testAllFunc_GA().

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

- include/geneticalg.h

5.8 mfunc::GeneticAlgorithm< T > Class Template Reference

The GeneticAlgorithm class executes the genetic algorithm with the specified parameters. To start, execute the run() function.

```
#include <geneticalg.h>
```

Public Member Functions

- GeneticAlgorithm ()

  Construct a new GeneticAlgorithm object.
- ~GeneticAlgorithm ()=default
- int run (GAParams< T > params)

  Executes the genetic algorithm with the specified parameters.

5.8.1 Detailed Description

template<class T>
class mfunc::GeneticAlgorithm< T >

The GeneticAlgorithm class executes the genetic algorithm with the specified parameters. To start, execute the run() function.

Template Parameters

T	Datatype used by the algorithm

Definition at line 72 of file geneticalg.h.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 GeneticAlgorithm()

```
template<class T >
mfunc::GeneticAlgorithm< T >::GeneticAlgorithm ( )
```

Construct a new GeneticAlgorithm object.

Template Parameters

T	Datatype used by the algorithm

Definition at line 98 of file geneticalg.h.

```
00099     : seed(), engine(seed()), rchance(0, 1)
00100 {
00101 }
```

5.8.2.2 ~GeneticAlgorithm()

```
template<class T>
mfunc::GeneticAlgorithm< T >::~GeneticAlgorithm ( )  [default]
```

5.8.3 Member Function Documentation

5.8.3.1 run()

```
template<class T >
int mfunc::GeneticAlgorithm< T >::run (
            GAParams< T > p )
```

Executes the genetic algorithm with the specified parameters.

Template Parameters

T	Datatype used by the algorithm

Parameters

p	GA parameters

Returns

int Non-zero error code on failure

Definition at line 111 of file geneticalg.h.

References mfunc::GAParams< T >::auxPop, mdata::Population< T >::copyPopulation(), mfunc::GAParams< T >::crProb, mfunc::GAParams< T >::elitismRate, mfunc::GAParams< T >::fitnessTable, mfunc::GAParams< T >::fitTableCol, mfunc::GAParams< T >::fMaxBound, mfunc::GAParams< T >::fMinBound, mfunc::GAParams< T >::fPtr, mfunc::GAParams< T >::generations, mdata::Population< T >::getFitness(), mdata::Population< T >::getPopulationPtr(), mdata::Population< T >::getPopulationSize(), mdata::Population< T >::getTotalFitness(), mfunc::GAParams< T >::mainPop, mfunc::GAParams< T >::mutPrec, mfunc::GAParams< T >::mutProb, mfunc::GAParams< T >::mutRange, and mdata::Population< T >::sortDescendByFitness().

Referenced by mfunc::Experiment< T >::runGAThreaded().

```
00112 {
00113     if (p.mainPop == nullptr || p.auxPop == nullptr || p.fPtr == nullptr)
00114         return 1;
00115
00116     if (p.mainPop->getPopulationSize() != p.auxPop->getPopulationSize() ||
00117         p.mainPop->getDimensionsSize() != p.auxPop->getDimensionsSize())
00118         return 2;
00119
00120     // Get population information
00121     const size_t popSize = p.mainPop->getPopulationSize();
00122     const size_t dimSize = p.mainPop->getDimensionsSize();
00123     const size_t elitism = p.elitismRate * (double)popSize;
00124
00125     // Allocate child buffers
00126     T* childOne = util::allocArray<T>(dimSize);
00127     T* childTwo = util::allocArray<T>(dimSize);
00128
00129     // Prepare populations
00130     p.mainPop->setFitnessNormalization(true);
00131     p.auxPop->setFitnessNormalization(true);
00132     p.mainPop->generate(p.fMinBound, p.fMaxBound);
00133     p.mainPop->calcAllFitness(p.fPtr);
00134
00135     // Loop for a number of generations
00136     for (unsigned int gen = 0; gen < p.generations; gen++)
00137     {
00138         for (size_t s = 0; s < popSize;)
00139         {
00140             size_t p1Index = 0;
00141             size_t p2Index = 0;
00142
00143             // Select parent indices
00144             select(p.mainPop, p1Index, p2Index);
00145
00146             // Produce new children by computing the crossover between parents
00147             crossover(dimSize, p.mainPop->getPopulationPtr(p1Index), p.mainPop->getPopulationPtr(p2Index),
00148                 p.crProb, childOne, childTwo);
00149
00150             // Mutate children
00151             mutate(dimSize, childOne, p.mutProb, p.mutRange, p.mutPrec, p.fMinBound, p.fMaxBound);
00152             mutate(dimSize, childTwo, p.mutProb, p.mutRange, p.mutPrec, p.fMinBound, p.fMaxBound);
00153
00154             // Copy new children into next generation
00155             p.auxPop->copyPopulation(s, childOne);
00156             s++;
00157             if (s < popSize) p.auxPop->copyPopulation(s, childTwo);
```

```
00158             s++;
00159         }
00160
00161         // Recalculate all fitness values for next generation
00162         p.auxPop->calcAllFitness(p.fPtr);
00163
00164         // Select and combine some of the best populations from the previous generation
00165         // with the next generation
00166         reduce(p.mainPop, p.auxPop, elitism);
00167
00168         // Swap the two populations
00169         auto tmp = p.mainPop;
00170         p.mainPop = p.auxPop;
00171         p.auxPop = tmp;
00172
00173         // Record current best population in results table
00174         p.fitnessTable->setEntry(gen, p.fitTableCol, p.mainPop->getMinCost());
00175     }
00176
00177     // Delete children buffers
00178     util::releaseArray<T>(childOne);
00179     util::releaseArray<T>(childTwo);
00180
00181     return 0;
00182 }
```

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

- include/geneticalg.h

5.9 util::IniReader Class Reference

The IniReader class is a simple *.ini file reader and parser.

```
#include <inireader.h>
```

Public Member Functions

- IniReader ()

  Construct a new IniReader object.
- ~IniReader ()

  Destroys the IniReader object.
- bool openFile (std::string filePath)

  Opens the given ini file and parses all sections/entries. The all file data is stored in memory and the file is closed.
- bool sectionExists (std::string section)

  Returns true if the given section exists in the current ini file.
- bool entryExists (std::string section, std::string entry)

  Returns true if the given section and entry key exists in the current ini file.
- std::string getEntry (std::string section, std::string entry)

  Returns the value for the entry that has the given entry key within the given section.
- template<class T >
  T getEntryAs (std::string section, std::string entry)

5.9.1 Detailed Description

The IniReader class is a simple ∗.ini file reader and parser.

– Initialize an IniReader object:

IniReader ini;

Open and parse an ∗.ini file:

ini.openFile("my_ini_file.ini");

Note that the file is immediately closed after parsing, and the file data is retained in memory.

Retrieve an entry from the ini file:

std::string value = ini.getEntry("My Section", "entryKey");

Definition at line 46 of file inireader.h.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 IniReader()

```
IniReader::IniReader ( )
```

Construct a new IniReader object.

Definition at line 21 of file inireader.cpp.

```
00021                     : file(""), iniMap()
00022 {
00023 }
```

5.9.2.2 ∼IniReader()

```
IniReader::~IniReader ( )
```

Destroys the IniReader object.

Definition at line 28 of file inireader.cpp.

```
00029 {
00030     iniMap.clear();
00031 }
```

5.9.3 Member Function Documentation

5.9.3.1 entryExists()

```
bool IniReader::entryExists (
            std::string section,
            std::string entry )
```

Returns true if the given section and entry key exists in the current ini file.

Parameters

section	std::string containing the section name
entry	std::string containing the entry key name

Returns

Returns true if the section and entry key exist in the ini file, otherwise false.

Definition at line 67 of file inireader.cpp.

Referenced by getEntry().

```
00068 {
00069     auto it = iniMap.find(section);
00070     if (it == iniMap.end()) return false;
00071
00072     return it->second.find(entry) != it->second.end();
00073 }
```

5.9.3.2 getEntry()

```
std::string IniReader::getEntry (
            std::string section,
            std::string entry )
```

Returns the value for the entry that has the given entry key within the given section.

Parameters

section	std::string containing the section name
entry	std::string containing the entry key name

Returns

The value of the entry with the given entry key and section. Returns an empty string if the entry does not exist.

Definition at line 84 of file inireader.cpp.

References entryExists().

Referenced by getEntryAs(), mfunc::Experiment< T >::init(), and mfunc::Experiment< T >::runDEThreaded().

```
00085 {
00086     if (!entryExists(section, entry)) return std::string();
00087
00088     return iniMap[section][entry];
00089 }
```

5.9.3.3 getEntryAs()

```
template<class T >
T util::IniReader::getEntryAs (
            std::string section,
            std::string entry )  [inline]
```

Definition at line 57 of file inireader.h.

References getEntry().

Referenced by mfunc::Experiment< T >::init(), and mfunc::Experiment< T >::runDEThreaded().

```
00058         {
00059             std::stringstream ss(getEntry(section, entry));
00060             T retVal;
00061             ss >> retVal;
00062             return retVal;
00063         }
```

5.9.3.4 openFile()

```
bool IniReader::openFile (
            std::string filePath )
```

Opens the given ini file and parses all sections/entries. The all file data is stored in memory and the file is closed.

Parameters

filePath	Path to the ini file you wish to open

Returns

Returns true if the file was succesfully opened and parsed. Otherwise false.

Definition at line 40 of file inireader.cpp.

Referenced by mfunc::Experiment< T >::init().

```
00041 {
00042     file = filePath;
00043     if (!parseFile())
00044         return false;
00045
00046     return true;
00047 }
```

5.9.3.5 sectionExists()

```
bool IniReader::sectionExists (
            std::string section )
```

Returns true if the given section exists in the current ini file.

Parameters

section	std::string containing the section name

Returns

Returns true if the section exists in the ini file, otherwise false.

Definition at line 55 of file inireader.cpp.

```
00056 {
00057     return iniMap.find(section) != iniMap.end();
00058 }
```

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

- include/inireader.h
- src/inireader.cpp

5.10 mdata::Population< T > Class Template Reference

Data class for storing a multi-dimensional population of data with the associated fitness.

`#include <population.h>`

Public Member Functions

- Population (size_t popSize, size_t dimensions)
  Construct a new Population object.
- ~Population ()
  Destroy Population object.
- bool isReady ()
  Returns true if the population instance is allocated and ready to be used.
- size_t getPopulationSize ()
  Returns the size of the population.
- size_t getDimensionsSize ()
  Returns the dimensions of the population.
- T * getPopulationPtr (size_t popIndex)
  Returns an array for the population with the given index.
- void copyPopulation (size_t destIndex, T *srcPop)
  Copies the values from another population vector into this population with the given destination index.
- void copyPopulation (size_t destIndex, const std::vector< T > &srcPop)
  Copies the values from another population vector into this population with the given destination index.
- void boundPopulation (size_t popIndex, T min, T max)
  Ensures that the population with the given index is within the correct value bounds given as parameters.
- void sortDescendByFitness ()
  Sorts the current population in descending order based on the current fitness values using quicksort.
- void setFitnessNormalization (bool useNormalization)

Sets or unsets the flag that determines if fitness values should be normalized after calculation.

- bool generate (T minBound, T maxBound)

  Generates new random values for this population that are within the given bounds. Resets all fitness values to zero.
- bool setFitness (size_t popIndex, T value)

  Sets the fitness value for a specific population vector index.
- bool calcFitness (size_t popIndex, mfunc::mfuncPtr< T > funcPtr)

  Uses the given function pointer to update the fitness value for the population vector at the given index.
- bool calcAllFitness (mfunc::mfuncPtr< T > funcPtr)

  Uses the given function pointer to calculate the fitness values for the entire population matrix.
- T getFitness (size_t popIndex)

  Returns the fitness value for a specific population vector index.
- T * getFitnessPtr (size_t popIndex)

  Returns the fitness value for a specific population vector index.
- std::vector< T > getAllFitness ()

  Returns a std::vector of all current fitness values.
- T * getMaxFitnessPtr ()

  Returns a pointer to the current max fitness value.
- size_t getMaxFitnessIndex ()

  Returns the index of the current max fitness value.
- T * getMinFitnessPtr ()

  Returns a pointer to the current min fitness value.
- size_t getMinFitnessIndex ()

  Returns the index of the current min fitness value.
- T getBestFitness ()

  Returns the value of the current best fitness. Best fitness depends on normalization flag.
- T * getBestFitnessPtr ()

  Returns a pointer to the current best fitness value. The best fitness calculation depends on if normalization is enabled.
- size_t getBestFitnessIndex ()

  Returns the index of the current best fitness value. The best fitness calculation depends on if normalization is enabled.
- T getTotalFitness ()

  Returns the sum of all fitness values.
- T getMinCost ()

  Returns the minimum cost value out of all populations. This value is different than the fitness if normalization is enabled.
- void outputPopulation (std::ostream &outStream, const char *delim, const char *lineBreak)

  Outputs all population data to the given output stream.
- void outputFitness (std::ostream &outStream, const char *delim, const char *lineBreak)

  Outputs all fitness data to the given output stream.
- void debugOutputAll ()

5.10.1 Detailed Description

template<class T>
class mdata::Population< T >

Data class for storing a multi-dimensional population of data with the associated fitness.

Template Parameters

T	Data type of the population.

Definition at line 30 of file population.h.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 Population()

```
template<class T >
Population::Population (
            size_t pSize,
            size_t dimensions )
```

Construct a new Population object.

Template Parameters

T	Data type of the population.

Parameters

pSize	Size of the population.
dimensions	Dimensions of the population.

Definition at line 30 of file population.cpp.

```
00031     : popMatrix(nullptr), popFitness(nullptr), popCost(nullptr), popSize(pSize), popDim(dimensions),
      normFitness(false)
00032 {
00033     if (!allocPopMatrix() || !allocPopFitness())
00034         throw std::bad_alloc();
00035 }
```

5.10.2.2 ~Population()

```
template<class T >
Population::~Population ( )
```

Destroy Population object.

Template Parameters

T	Data type of the population.

Definition at line 43 of file population.cpp.

```
00044 {
```

```
00045     releasePopMatrix();
00046     releasePopFitness();
00047 }
```

5.10.3 Member Function Documentation

5.10.3.1 boundPopulation()

```
template<class T >
void Population::boundPopulation (
             size_t popIndex,
             T min,
             T max )
```

Ensures that the population with the given index is within the correct value bounds given as parameters.

Template Parameters

T	Data type of the population.

Parameters

popIndex	Index of the population to conduct the bounds check
min	Minimum bound
max	Maximum bound

Definition at line 149 of file population.cpp.

References mdata::Population< T >::getPopulationPtr().

```
00150 {
00151     if (popIndex >= popSize) return;
00152
00153     auto v = getPopulationPtr(popIndex);
00154     for (size_t i = 0; i < popDim; i++)
00155     {
00156         if (v[i] < min)
00157             v[i] = min;
00158         else if (v[i] > max)
00159             v[i] = max;
00160     }
00161 }
```

5.10.3.2 calcAllFitness()

```
template<class T >
bool Population::calcAllFitness (
             mfunc::mfuncPtr< T > funcPtr )
```

Uses the given function pointer to calculate the fitness values for the entire population matrix.

Template Parameters

T	Data type of the population.

Parameters

popIndex	Index of the population vector you wish to set the fitness for.
funcPtr	Function pointer to the math function that will be used to calculate the fitness value.

Returns

Returns true on success, otherwise false.

Definition at line 283 of file population.cpp.

References mdata::Population< T >::getMinCost().

```
00284 {
00285     if (popFitness == nullptr) return false;
00286
00287     auto globalMinCost = getMinCost();
00288
00289     for (size_t i = 0; i < popSize; i++)
00290     {
00291         if (normFitness)
00292         {
00293             popCost[i] = funcPtr(popMatrix[i], popDim);
00294             popFitness[i] = normalizeCost(popCost[i], globalMinCost);
00295         }
00296         else
00297         {
00298             popCost[i] = funcPtr(popMatrix[i], popDim);
00299             popFitness[i] = popCost[i];
00300         }
00301     }
00302
00303     return true;
00304 }
```

5.10.3.3 calcFitness()

```
template<class T >
bool Population::calcFitness (
            size_t popIndex,
            mfunc::mfuncPtr< T > funcPtr )
```

Uses the given function pointer to update the fitness value for the population vector at the given index.

Template Parameters

T	Data type of the population.

Parameters

popIndex	Index of the population vector you wish to set the fitness for.

Parameters

funcPtr	Function pointer to the math function that will be used to calculate the fitness value.

Returns

Returns true on success, otherwise false.

Definition at line 254 of file population.cpp.

References mdata::Population< T >::getMinCost().

Referenced by mfunc::DifferentialEvolution< T >::run().

```
00255 {
00256     if (popFitness == nullptr || popIndex >= popSize) return false;
00257
00258     if (normFitness)
00259     {
00260         popCost[popIndex] = funcPtr(popMatrix[popIndex], popDim);
00261         popFitness[popIndex] = normalizeCost(popCost[popIndex], getMinCost());
00262     }
00263     else
00264     {
00265         popCost[popIndex] = funcPtr(popMatrix[popIndex], popDim);
00266         popFitness[popIndex] = popCost[popIndex];
00267     }
00268
00269     return true;
00270 }
```

5.10.3.4 copyPopulation() [1/2]

```
template<class T >
void Population::copyPopulation (
            size_t destIndex,
            T * srcPop )
```

Copies the values from another population vector into this population with the given destination index.

Template Parameters

T	Data type of the population.

Parameters

destIndex	Index of the population vector you wish to overwrite.
srcPop	Pointer to the source population vector that will be copied

Definition at line 110 of file population.cpp.

Referenced by mfunc::GeneticAlgorithm< T >::run(), and mfunc::DifferentialEvolution< T >::run().

```
00111 {
00112     if (popFitness == nullptr || destIndex >= popSize) return;
00113
00114     for (size_t i = 0; i < popDim; i++)
00115     {
00116         popMatrix[destIndex][i] = srcPop[i];
00117     }
00118 }
```

5.10.3.5 copyPopulation() [2/2]

```
template<class T >
void Population::copyPopulation (
             size_t destIndex,
             const std::vector< T > & srcPop )
```

Copies the values from another population vector into this population with the given destination index.

Template Parameters

T	Data type of the population.

Parameters

destIndex	Index of the population vector you wish to overwrite.
srcPop	Reference to a vector containing the source population to copy

Definition at line 129 of file population.cpp.

```
00130 {
00131     if (popFitness == nullptr || destIndex >= popSize) return;
00132
00133     for (size_t i = 0; i < popDim && i < srcPop.size(); i++)
00134     {
00135         popMatrix[destIndex][i] = srcPop[i];
00136     }
00137 }
```

5.10.3.6 debugOutputAll()

```
template<class T >
void Population::debugOutputAll ( )
```

Definition at line 684 of file population.cpp.

```
00685 {
00686     for (size_t i = 0; i < popSize; i++)
00687     {
00688         for (size_t d = 0; d < popDim; d++)
00689         {
00690             std::cout << std::setw(10) << popMatrix[i][d] << " ";
00691         }
00692
00693         std::cout << " | " << std::setw(10) << popCost[i];
00694
00695         std::cout << " | " << std::setw(10) << popFitness[i] << std::endl;
00696     }
00697 }
```

5.10.3.7 generate()

```
template<class T >
bool Population::generate (
            T minBound,
            T maxBound )
```

Generates new random values for this population that are within the given bounds. Resets all fitness values to zero.

Template Parameters

T	Data type of the population.

Parameters

minBound	The minimum bound for a population value.
maxBound	The maximum bound for a population value.

Returns

Returns true of the population was succesfully generated, otherwise false.

Definition at line 199 of file population.cpp.

```
00200 {
00201     if (popMatrix == nullptr) return false;
00202
00203     // Generate a new seed for the mersenne twister engine
00204     rgen = std::mt19937(rdev());
00205
00206     // Set up a uniform distribution for the random number generator with the correct function bounds
00207     std::uniform_real_distribution<T> dist(minBound, maxBound);
00208
00209     // Generate values for all vectors in popMatrix
00210     for (size_t s = 0; s < popSize; s++)
00211     {
00212         for (size_t d = 0; d < popDim; d++)
00213         {
00214             T rand = dist(rgen);
00215             popMatrix[s][d] = rand;
00216         }
00217     }
00218
00219     // Reset popFitness values to 0
00220     initArray<T>(popFitness, popSize, (T)0.0);
00221
00222     return true;
00223 }
```

5.10.3.8 getAllFitness()

```
template<class T >
std::vector< T > Population::getAllFitness ( )
```

Returns a std::vector of all current fitness values.

Template Parameters

T	Data type of the population.

Returns

std::vector<T> std::vector of fitness values

Definition at line 343 of file population.cpp.

```
00344 {
00345     return std::vector<T>(popFitness[0], popFitness[popSize]);
00346 }
```

5.10.3.9 getBestFitness()

```
template<class T >
T Population::getBestFitness ( )
```

Returns the value of the current best fitness. Best fitness depends on normalization flag.

Template Parameters

T	Data type of the population.

Returns

T Value of the current best fitness

Definition at line 420 of file population.cpp.

References mdata::Population< T >::getBestFitnessIndex().

```
00421 {
00422     return popFitness[getBestFitnessIndex()];
00423 }
```

5.10.3.10 getBestFitnessIndex()

```
template<class T >
size_t Population::getBestFitnessIndex ( )
```

Returns the index of the current best fitness value. The best fitness calculation depends on if normalization is enabled.

Template Parameters

T	Data type of the population.

Returns

size_t Index of the best fitness value

Definition at line 448 of file population.cpp.

References mdata::Population< T >::getMaxFitnessIndex(), and mdata::Population< T >::getMinFitnessIndex().

Referenced by mdata::Population< T >::getBestFitness(), and mdata::Population< T >::getBestFitnessPtr().

```
00449 {
00450     if (normFitness)
00451         return getMaxFitnessIndex();
00452     else
00453         return getMinFitnessIndex();
00454 }
```

5.10.3.11 getBestFitnessPtr()

```
template<class T >
T * Population::getBestFitnessPtr ( )
```

Returns a pointer to the current best fitness value. The best fitness calculation depends on if normalization is enabled.

Template Parameters

T	Data type of the population.

Returns

T* Pointer to the best fitness value

Definition at line 434 of file population.cpp.

References mdata::Population< T >::getBestFitnessIndex().

```
00435 {
00436     return &popFitness[getBestFitnessIndex()];
00437 }
```

5.10.3.12 getDimensionsSize()

```
template<class T >
size_t Population::getDimensionsSize ( )
```

Returns the dimensions of the population.

Template Parameters

T	Data type of the population.

Returns

The number of dimensions in the population.

Definition at line 81 of file population.cpp.

Referenced by mfunc::DifferentialEvolution< T >::run().

```
00082 {
00083     return popDim;
00084 }
```

5.10.3.13 getFitness()

```
template<class T >
T Population::getFitness (
            size_t popIndex )
```

Returns the fitness value for a specific population vector index.

Template Parameters

T	Data type of the population.

Parameters

popIndex	Index of the population vector you wish to retrieve the fitness from.

Returns

Returns the fitness value if popIndex is valid. Otherwise zero.

Definition at line 314 of file population.cpp.

Referenced by mfunc::GeneticAlgorithm< T >::run(), and mfunc::DifferentialEvolution< T >::run().

```
00315 {
00316     if (popFitness == nullptr || popIndex >= popSize) return 0;
00317
00318     return popFitness[popIndex];
00319 }
```

5.10.3.14 getFitnessPtr()

```
template<class T >
T * Population::getFitnessPtr (
            size_t popIndex )
```

Returns the fitness value for a specific population vector index.

Template Parameters

T	Data type of the population.

Parameters

popIndex	Index of the population vector you wish to retrieve the fitness from.

Returns

Returns the fitness value if popIndex is valid. Otherwise zero.

Definition at line 329 of file population.cpp.

```
00330 {
00331     if (popFitness == nullptr || popIndex >= popSize) return 0;
00332
00333     return &popFitness[popIndex];
00334 }
```

5.10.3.15 getMaxFitnessIndex()

```
template<class T >
size_t Population::getMaxFitnessIndex ( )
```

Returns the index of the current max fitness value.

Template Parameters

T	Data type of the population.

Returns

size_t Index of the max fitness value

Definition at line 399 of file population.cpp.

Referenced by mdata::Population< T >::getBestFitnessIndex(), and mdata::Population< T >::getMaxFitnessPtr().

```
00400 {
00401     size_t maxIndex = 0;
00402
00403     for (size_t i = 1; i < popSize; i++)
00404     {
00405         if (popFitness[i] > popFitness[maxIndex])
00406             maxIndex = i;
00407     }
00408
00409     return maxIndex;
00410 }
```

5.10.3.16 getMaxFitnessPtr()

```
template<class T >
T * Population::getMaxFitnessPtr ( )
```

Returns a pointer to the current max fitness value.

Template Parameters

T	Data type of the population.

Returns

T* Pointer to the max fitness value

Definition at line 387 of file population.cpp.

References mdata::Population< T >::getMaxFitnessIndex().

```
00388 {
00389     return &popFitness[getMaxFitnessIndex()];
00390 }
```

5.10.3.17 getMinCost()

```
template<class T >
T Population::getMinCost ( )
```

Returns the minimum cost value out of all populations. This value is different than the fitness if normalization is enabled.

Template Parameters

T	Data type of the population.

Returns

T Value of minimum cost

Definition at line 483 of file population.cpp.

Referenced by mdata::Population< T >::calcAllFitness(), and mdata::Population< T >::calcFitness().

```
00484 {
00485     T min = popCost[0];
00486
00487     for (size_t i = 1; i < popSize; i++)
00488     {
00489         if (popCost[i] < min)
00490             min = popCost[i];
00491     }
00492
00493     return min;
00494 }
```

5.10.3.18 getMinFitnessIndex()

```
template<class T >
size_t Population::getMinFitnessIndex ( )
```

Returns the index of the current min fitness value.

Template Parameters

T	Data type of the population.

Returns

size_t Index of the min fitness value

Definition at line 367 of file population.cpp.

Referenced by mdata::Population< T >::getBestFitnessIndex(), and mdata::Population< T >::getMinFitnessPtr().

```
00368 {
00369     size_t minIndex = 0;
00370
00371     for (size_t i = 1; i < popSize; i++)
00372     {
00373         if (popFitness[i] < popFitness[minIndex])
00374             minIndex = i;
00375     }
00376
00377     return minIndex;
00378 }
```

5.10.3.19 getMinFitnessPtr()

```
template<class T >
T * Population::getMinFitnessPtr ( )
```

Returns a pointer to the current min fitness value.

Template Parameters

T	Data type of the population.

Returns

T* Pointer to the min fitness value

Definition at line 355 of file population.cpp.

References mdata::Population< T >::getMinFitnessIndex().

```
00356 {
00357     return &popFitness[getMinFitnessIndex()];
00358 }
```

5.10.3.20 getPopulationPtr()

```
template<class T >
T * Population::getPopulationPtr (
            size_t popIndex )
```

Returns an array for the population with the given index.

Template Parameters

T	Data type of the population.

Parameters

popIndex	Index of the population vector you wish to retrieve.

Returns

Pointer to population vector array at the given index.

Definition at line 94 of file population.cpp.

Referenced by mdata::Population< T >::boundPopulation(), mfunc::GeneticAlgorithm< T >::run(), and mfunc::DifferentialEvolution< T >::run().

```
00095 {
00096     if (popMatrix == nullptr || popIndex >= popSize) return nullptr;
00097
00098     return popMatrix[popIndex];
00099 }
```

5.10.3.21 getPopulationSize()

```
template<class T >
size_t Population::getPopulationSize ( )
```

Returns the size of the population.

Template Parameters

T	Data type of the population.

Returns

The size of the population.

Definition at line 69 of file population.cpp.

Referenced by mfunc::GeneticAlgorithm< T >::run().

```
00070 {
00071     return popSize;
00072 }
```

5.10.3.22 getTotalFitness()

```
template<class T >
T Population::getTotalFitness ( )
```

Returns the sum of all fitness values.

Template Parameters

T	Data type of the population.

Returns

T Sum of all fitness values

Definition at line 463 of file population.cpp.

Referenced by mfunc::GeneticAlgorithm< T >::run().

```
00464 {
00465     T sum = 0;
00466
00467     for (size_t i = 0; i < popSize; i++)
00468     {
00469         sum += popFitness[i];
00470     }
00471
00472     return sum;
00473 }
```

5.10.3.23 isReady()

```
template<class T >
bool Population::isReady ( )
```

Returns true if the population instance is allocated and ready to be used.

Template Parameters

T	Data type of the population.

Returns

Returns true if the population instance is in a valid state.

Definition at line 57 of file population.cpp.

```
00058 {
00059     return popMatrix != nullptr && popFitness != nullptr;
00060 }
```

5.10.3.24 outputFitness()

```
template<class T >
void Population::outputFitness (
            std::ostream & outStream,
            const char * delim,
            const char * lineBreak )
```

Outputs all fitness data to the given output stream.

Template Parameters

T	Data type of the population.

Parameters

outStream	Output stream to write the data to.
delim	Delimiter characters to separate columns.
lineBreak	Delimiter characters to separate rows.

Definition at line 531 of file population.cpp.

```
00532 {
00533     if (popFitness == nullptr) return;
00534
00535     for (size_t j = 0; j < popSize; j++)
00536     {
```

```
00537         outStream << popFitness[j];
00538             if (j < popSize - 1)
00539                 outStream << delim;
00540     }
00541
00542     if (lineBreak != nullptr)
00543         outStream << lineBreak;
00544 }
```

5.10.3.25 outputPopulation()

```
template<class T >
void Population::outputPopulation (
              std::ostream & outStream,
              const char * delim,
              const char * lineBreak )
```

Outputs all population data to the given output stream.

Template Parameters

T	Data type of the population.

Parameters

outStream	Output stream to write the data to.
delim	Delimiter characters to separate columns.
lineBreak	Delimiter characters to separate rows.

Definition at line 505 of file population.cpp.

```
00506 {
00507     if (popMatrix == nullptr) return;
00508
00509     for (size_t j = 0; j < popSize; j++)
00510     {
00511         for (size_t k = 0; k < popDim; k++)
00512         {
00513             outStream << popMatrix[j][k];
00514             if (k < popDim - 1)
00515                 outStream << delim;
00516         }
00517
00518         outStream << lineBreak;
00519     }
00520 }
```

5.10.3.26 setFitness()

```
template<class T >
bool Population::setFitness (
              size_t popIndex,
              T value )
```

Sets the fitness value for a specific population vector index.

Template Parameters

T	Data type of the population.

Parameters

popIndex	Index of the population vector you wish to set the fitness for.
value	The value of the fitness.

Returns

Returns true if the fitness was succesfully set, otherwise false.

Definition at line 234 of file population.cpp.

Referenced by mfunc::DifferentialEvolution< T >::run().

```
00235 {
00236     if (popFitness == nullptr || popIndex >= popSize) return false;
00237
00238     popFitness[popIndex] = value;
00239
00240     return true;
00241 }
```

5.10.3.27 setFitnessNormalization()

```
template<class T >
void Population::setFitnessNormalization (
            bool useNormalization )
```

Sets or unsets the flag that determines if fitness values should be normalized after calculation.

Template Parameters

T	Data type of the population.

Parameters

useNormalization	True if you want to enable fitness normalization

Definition at line 183 of file population.cpp.

```
00184 {
00185     normFitness = useNormalization;
00186 }
```

5.10.3.28 sortDescendByFitness()

```
template<class T >
void Population::sortDescendByFitness ( )
```

Sorts the current population in descending order based on the current fitness values using quicksort.

Template Parameters

T	Data type of the population.

Definition at line 170 of file population.cpp.

Referenced by mfunc::GeneticAlgorithm< T >::run().

```
00171 {
00172     qs_fit_decend(0, popSize - 1);
00173 }
```

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

- include/population.h
- src/population.cpp

5.11 mfunc::RandomBounds< T > Struct Template Reference

Simple struct for storing the minimum and maximum input vector bounds for a function.

```
#include <experiment.h>
```

Public Attributes

- T min = 0.0
- T max = 0.0

5.11.1 Detailed Description

template<class T>
struct mfunc::RandomBounds< T >

Simple struct for storing the minimum and maximum input vector bounds for a function.

Definition at line 34 of file experiment.h.

5.11.2 Member Data Documentation

5.11.2.1 max

```
template<class T>
T mfunc::RandomBounds< T >::max = 0.0
```

Definition at line 37 of file experiment.h.

Referenced by mfunc::Experiment< T >::runDEThreaded().

5.11.2.2 min

```
template<class T>
T mfunc::RandomBounds< T >::min = 0.0
```

Definition at line 36 of file experiment.h.

Referenced by mfunc::Experiment< T >::runDEThreaded().

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

- include/experiment.h

5.12 ThreadPool Class Reference

```
#include <threadpool.h>
```

Public Member Functions

- ThreadPool (size_t)
- template<class F , class... Args>
  auto enqueue (F &&f, Args &&... args) -> std::future< typename std::result_of< F(Args...)>::type >
- ~ThreadPool ()
- void stopAndJoinAll ()

5.12.1 Detailed Description

Copyright (c) 2012 Jakob Progsch, Václav Zeman https://github.com/progschj/ThreadPool

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

This source file has been modified slightly by Andrew Dunn

Definition at line 42 of file threadpool.h.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 ThreadPool()

```
ThreadPool::ThreadPool (
            size_t threads )  [inline]
```

Definition at line 64 of file threadpool.h.

```
00065     :   stop(false)
00066 {
00067     for(size_t i = 0;i<threads;++i)
00068         workers.emplace_back(
00069             [this]
00070             {
00071                 for(;;)
00072                 {
00073                     std::function<void()> task;
00074 
00075                     {
00076                         std::unique_lock<std::mutex> lock(this->queue_mutex);
00077                         this->condition.wait(lock,
00078                             [this]{ return this->stop || !this->tasks.empty(); });
00079                         if(this->stop && this->tasks.empty())
00080                             return;
00081                         task = std::move(this->tasks.front());
00082                         this->tasks.pop();
00083                     }
00084 
00085                     task();
00086                 }
00087             }
00088         );
00089 }
```

5.12.2.2 ~ThreadPool()

```
ThreadPool::~ThreadPool ( )  [inline]
```

Definition at line 117 of file threadpool.h.

References stopAndJoinAll().

```
00118 {
00119     stopAndJoinAll();
00120 }
```

5.12.3 Member Function Documentation

5.12.3.1 enqueue()

```
template<class F , class...  Args>
auto ThreadPool::enqueue (
             F && f,
             Args &&...  args ) -> std::future<typename std::result_of<F(Args...)>::type>
```

Definition at line 93 of file threadpool.h.

Referenced by mfunc::Experiment< T >::testAllFunc_DE(), and mfunc::Experiment< T >::testAllFunc_GA().

```
00095 {
00096     using return_type = typename std::result_of<F(Args...)>::type;
00097
00098     auto task = std::make_shared< std::packaged_task<return_type()> >(
00099             std::bind(std::forward<F>(f), std::forward<Args>(args)...)
00100         );
00101
00102     std::future<return_type> res = task->get_future();
00103     {
00104         std::unique_lock<std::mutex> lock(queue_mutex);
00105
00106         // don't allow enqueueing after stopping the pool
00107         if(stop)
00108             throw std::runtime_error("enqueue on stopped ThreadPool");
00109
00110         tasks.emplace([task](){ (*task)(); });
00111     }
00112     condition.notify_one();
00113     return res;
00114 }
```

5.12.3.2 stopAndJoinAll()

```
void ThreadPool::stopAndJoinAll ( )  [inline]
```

Definition at line 122 of file threadpool.h.

Referenced by mfunc::Experiment< T >::testAllFunc_DE(), mfunc::Experiment< T >::testAllFunc_GA(), and ~ThreadPool().

```
00123 {
00124     {
00125         std::unique_lock<std::mutex> lock(queue_mutex);
00126         stop = true;
00127     }
00128
00129     condition.notify_all();
00130     for(std::thread &worker: workers)
00131         worker.join();
00132 }
```

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

- include/threadpool.h

Chapter 6

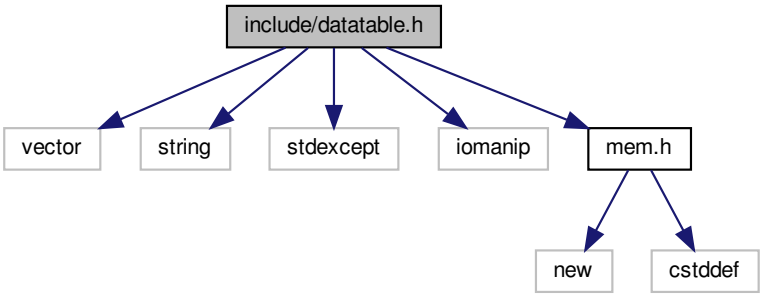
File Documentation

6.1 include/datatable.h File Reference

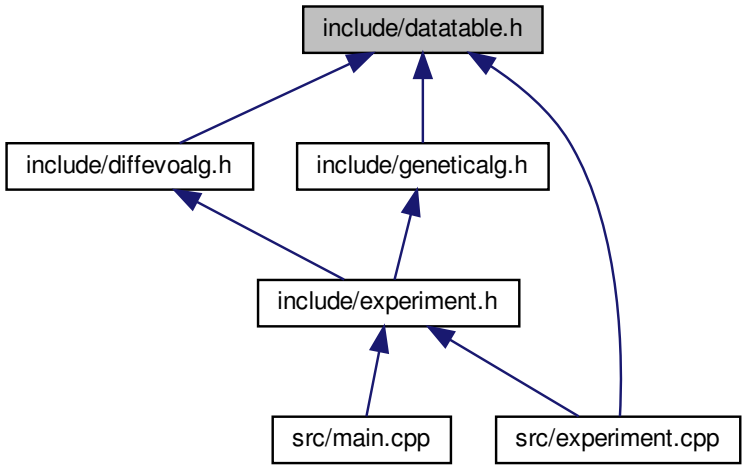
Header file for the DataTable class, which represents a spreadsheet/table of values that can easily be exported to a ∗.csv file.

```
#include <vector>
#include <string>
#include <stdexcept>
#include <iomanip>
#include "mem.h"
```

Include dependency graph for datatable.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mdata::DataTable< T >

  The DataTable class is a simple table of values with labeled columns.

Namespaces

- mdata

6.1.1 Detailed Description

Header file for the DataTable class, which represents a spreadsheet/table of values that can easily be exported to a ∗.csv file.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.2

Date

2019-04-01

Copyright

Copyright (c) 2019

Definition in file datatable.h.

6.2 datatable.h

```
00001
00013 #ifndef __DATATABLE_H
00014 #define __DATATABLE_H
00015
00016 #include <vector>
00017 #include <string>
00018 #include <stdexcept>
00019 #include <iomanip>
00020 #include "mem.h"
00021
00022 namespace mdata
00023 {
00049     template <class T>
00050     class DataTable
00051     {
00052     public:
00060         DataTable(size_t _rows, size_t _cols) : rows(_rows), cols(_cols), dataMatrix(nullptr)
00061         {
00062             if (rows == 0)
00063                 throw std::length_error("Table rows must be greater than 0.");
00064             else if (cols == 0)
00065                 throw std::length_error("Table columns must be greater than 0.");
00066
00067             dataMatrix = util::allocMatrix<T>(rows, cols);
00068             if (dataMatrix == nullptr)
00069                 throw std::bad_alloc();
00070
00071             colLabels.resize(_cols, std::string());
00072         }
00073
00077         ~DataTable()
00078         {
00079             util::releaseMatrix(dataMatrix, rows);
00080         }
00081
00082         void clearData()
00083         {
00084             util::initMatrix<T>(dataMatrix, rows, cols, 0);
00085         }
00086
00093         std::string getColLabel(size_t colIndex)
00094         {
00095             if (colIndex >= colLabels.size())
00096                 throw std::out_of_range("Column index out of range");
00097
00098             return colLabels[colIndex];
00099         }
00100
00107         void setColLabel(size_t colIndex, std::string newLabel)
00108         {
00109              if (colIndex >= colLabels.size())
00110                 throw std::out_of_range("Column index out of range");
00111
00112             colLabels[colIndex] = newLabel;
00113         }
00114
00122         T getEntry(size_t row, size_t col)
00123         {
00124             if (dataMatrix == nullptr)
00125                 throw std::runtime_error("Data matrix not allocated");
00126             if (row >= rows)
00127                 throw std::out_of_range("Table row out of range");
00128             else if (col >= cols)
00129                 throw std::out_of_range("Table column out of range");
00130
00131             return dataMatrix[row][col];
00132         }
00133
00141         void setEntry(size_t row, size_t col, T val)
00142         {
00143             if (dataMatrix == nullptr)
00144                 throw std::runtime_error("Data matrix not allocated");
00145             if (row >= rows)
00146                 throw std::out_of_range("Table row out of range");
00147             else if (col >= cols)
00148                 throw std::out_of_range("Table column out of range");
00149
00150             dataMatrix[row][col] = val;
00151         }
00152
00160         bool exportCSV(const char* filePath)
00161         {
00162             if (dataMatrix == nullptr) return false;
00163
```

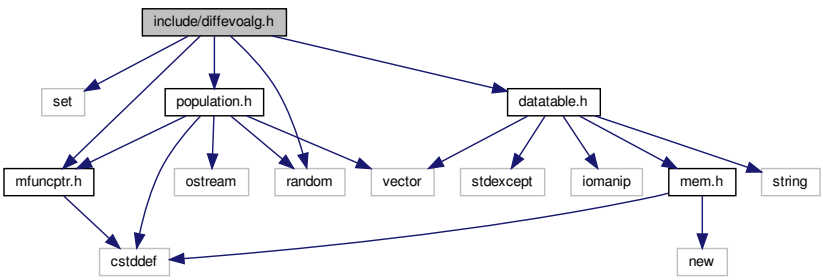
```
00164             using namespace std;
00165             ofstream outFile;
00166             outFile.open(filePath, ofstream::out | ofstream::trunc);
00167             if (!outFile.good()) return false;
00168 
00169             // Print column labels
00170             for (unsigned int c = 0; c < cols; c++)
00171             {
00172                 outFile << colLabels[c];
00173                 if (c < cols - 1) outFile << ",";
00174             }
00175 
00176             outFile << endl;
00177 
00178             // Print data rows
00179             for (unsigned int r = 0; r < rows; r++)
00180             {
00181                 for (unsigned int c = 0; c < cols; c++)
00182                 {
00183                     outFile << std::setprecision(8) << dataMatrix[r][c];
00184                     if (c < cols - 1) outFile << ",";
00185                 }
00186                 outFile << endl;
00187             }
00188 
00189             outFile.close();
00190             return true;
00191         }
00192     private:
00193         size_t rows;
00194         size_t cols;
00195         std::vector<std::string> colLabels;
00196         T** dataMatrix;
00198     };
00199 } // mdata
00200 
00201 #endif
00202 
00203 // ========================
00204 // End of datatable.h
00205 // ========================
```

6.3 include/diffevoalg.h File Reference

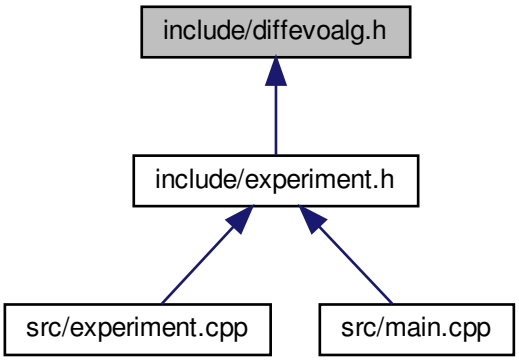
Implementation of the DifferentialEvolution class. Executes the differential evolution algorithm with the specified parameters.

```
#include <set>
#include "population.h"
#include "mfuncptr.h"
#include "datatable.h"
#include "random"
```

Include dependency graph for diffevoalg.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct mfunc::DEParams< T >

  Simple structure that holds various parameters to run the differential evolutionary algorithm.
- class mfunc::DifferentialEvolution< T >

  The DifferentialEvolution class executes the differential evolution algorithm on a given population using the given parameters passed via a DEParams structure. To start, call the "run" function.

Namespaces

- mfunc

Macros

- #define MAX_RAND_VECTOR_SELECT 4

Enumerations

- enum mfunc::DEStrategy { mfunc::DEStrategy::Best1Exp = 0, mfunc::DEStrategy::Rand1Exp = 1, mfunc::DEStrategy::RandToBest1Exp = 2, mfunc::DEStrategy::Best2Exp = 3, mfunc::DEStrategy::Rand2Exp = 4, mfunc::DEStrategy::Best1Bin = 5, mfunc::DEStrategy::Rand1Bin = 6, mfunc::DEStrategy::RandToBest1Bin = 7, mfunc::DEStrategy::Best2Bin = 8, mfunc::DEStrategy::Rand2Bin = 9, mfunc::DEStrategy::Count = 10 }

  Enum used to specify which differential evolution strategy should be used.
- enum mfunc::PerturbedVector { mfunc::PerturbedVector::Best, mfunc::PerturbedVector::Random, mfunc::PerturbedVector::RandToBest }

  Enum used to specify what vector should be perturbed during mutation.
- enum mfunc::NumberDiffVectors { mfunc::NumberDiffVectors::One, mfunc::NumberDiffVectors::Two }

  Enum used to specify the number of random difference vectors used during mutation.
- enum mfunc::CrossoverStrat { mfunc::CrossoverStrat::Exponential, mfunc::CrossoverStrat::Binomial }

  Enum used to specify a crossover strategy.

6.3.1 Detailed Description

Implementation of the DifferentialEvolution class. Executes the differential evolution algorithm with the specified parameters.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-04-29

Copyright

Definition in file diffevoalg.h.

6.3.2 Macro Definition Documentation

6.3.2.1 MAX_RAND_VECTOR_SELECT

```
#define MAX_RAND_VECTOR_SELECT 4
```

Definition at line 21 of file diffevoalg.h.

Referenced by mfunc::DifferentialEvolution< T >::run().

6.4 diffevoalg.h

```
00001
00012 #ifndef __DIFFEVOALG_H
00013 #define __DIFFEVOALG_H
00014
00015 #include <set>
00016 #include "population.h"
00017 #include "mfuncptr.h"
00018 #include "datatable.h"
00019 #include "random"
00020
00021 #define MAX_RAND_VECTOR_SELECT 4
00022
00023 namespace mfunc
00024 {
00025
00029     enum class DEStrategy
00030     {
00031         Best1Exp = 0,
00032         Rand1Exp = 1,
00033         RandToBest1Exp = 2,
00034         Best2Exp = 3,
00035         Rand2Exp = 4,
00036         Best1Bin = 5,
00037         Rand1Bin = 6,
00038         RandToBest1Bin = 7,
00039         Best2Bin = 8,
00040         Rand2Bin = 9,
00041         Count = 10
00042     };
00043
00047     enum class PerturbedVector
00048     {
00049         Best,
00050         Random,
00051         RandToBest
00052     };
00053
00057     enum class NumberDiffVectors
00058     {
00059         One,
00060         Two
00061     };
00062
00066     enum class CrossoverStrat
00067     {
00068         Exponential,
00069         Binomial
00070     };
00071
00078     template <class T>
00079     struct DEParams
00080     {
00081         mdata::DataTable<T>* fitnessTable;
00082         size_t fitTableCol;
00083         mdata::Population<T>* mainPop;
00084         mdata::Population<T>* nextPop;
00085         mfuncPtr<T> fPtr;
00086         T fMinBound;
00087         T fMaxBound;
00088         unsigned int generations;
00089         double crFactor;
00090         double scalingFactor1;
00091         double scalingFactor2;
00092         DEStrategy strategy;
00093
00094         DEParams()
00095         {
00096             fitnessTable = nullptr;
00097             fitTableCol = 0;
00098             mainPop = nullptr;
00099             nextPop = nullptr;
00100             fPtr = nullptr;
00101             fMinBound = 0;
00102             fMaxBound = 0;
00103             generations = 0;
00104             crFactor = 0;
00105             scalingFactor1 = 0;
00106             scalingFactor2 = 0;
00107             strategy = DEStrategy::Best1Exp;
00108         }
00109     };
00110
00118     template <class T>
00119     class DifferentialEvolution
```

```
00120     {
00121     public:
00122         DifferentialEvolution();
00123         ~DifferentialEvolution() = default;
00124         int run(DEParams<T> params);
00125     private:
00126         std::random_device seed;
00127         std::mt19937 engine;
00128         std::uniform_real_distribution<double> rchance;
00129 
00130         void mutateAndCrossover(mdata::Population<T>* mainPop,
      mdata::Population<T>* nextPop, const size_t vIndex, const size_t bestIndex,
00131             const PerturbedVector pertV, const NumberDiffVectors diffV,
      const CrossoverStrat crossStrat, const double sf1, const double sf2, const double crFactor);
00132         void select(mdata::Population<T>* mainPop,
      mdata::Population<T>* nextPop, size_t vIndex,
      mfunc::mfuncPtr<T> fPtr);
00133         void parseDEStrat(DEStrategy mainStrat, PerturbedVector& oPertv,
      NumberDiffVectors& oDiffv, CrossoverStrat& oCross);
00134     };
00135 }
00136 
00142 template <class T>
00143 mfunc::DifferentialEvolution<T>::DifferentialEvolution
      ()
00144     : seed(), engine(seed()), rchance(0, 1)
00145 {
00146 }
00147 
00155 template <class T>
00156 int mfunc::DifferentialEvolution<T>::run(
      DEParams<T> p)
00157 {
00158     if (p.mainPop == nullptr || p.nextPop == nullptr || p.fPtr == nullptr)
00159         return 1;
00160 
00161     if (p.mainPop->getPopulationSize() != p.nextPop->getPopulationSize() ||
00162         p.mainPop->getDimensionsSize() != p.nextPop->getDimensionsSize())
00163         return 2;
00164 
00165     const size_t popSize = p.mainPop->getPopulationSize();
00166     const size_t dimSize = p.mainPop->getDimensionsSize();
00167 
00168     // Parse DE strategy
00169     PerturbedVector pertV;
00170     NumberDiffVectors diffV;
00171     CrossoverStrat crStrat;
00172     parseDEStrat(p.strategy, pertV, diffV, crStrat);
00173 
00174     // Prepare populations
00175     p.mainPop->setFitnessNormalization(false);
00176     p.nextPop->setFitnessNormalization(false);
00177     p.mainPop->generate(p.fMinBound, p.fMaxBound);
00178     p.mainPop->calcAllFitness(p.fPtr);
00179 
00180     // Calc best fitness pop index
00181     size_t bestFitIndex = p.mainPop->getBestFitnessIndex();
00182 
00183     for (unsigned int gen = 0; gen < p.generations; gen++)
00184     {
00185         for (size_t i = 0; i < popSize; i++)
00186         {
00187             // For each population in the next generation, mutate and crossover, then select
00188             mutateAndCrossover(p.mainPop, p.nextPop, i, bestFitIndex, pertV, diffV, crStrat,
      p.scalingFactor1, p.scalingFactor2, p.crFactor);
00189             p.nextPop->boundPopulation(i, p.fMinBound, p.
      fMaxBound);
00190             select(p.mainPop, p.nextPop, i, p.fPtr);
00191         }
00192 
00193         // Swap the two populations
00194         auto tmp = p.mainPop;
00195         p.mainPop = p.nextPop;
00196         p.nextPop = tmp;
00197 
00198         // Recalculate best fitness index and add result to results table
00199         bestFitIndex = p.mainPop->getBestFitnessIndex();
00200         p.fitnessTable->setEntry(gen, p.fitTableCol, p.
      mainPop->getFitness(bestFitIndex));
00201     }
00202 
00203     return 0;
00204 }
00205 
00222 template <class T>
00223 void mfunc::DifferentialEvolution<T>::mutateAndCrossover
      (mdata::Population<T>* mainPop, mdata::Population<T>* nextPop, const
```

```
      size_t vIndex,
00224     const size_t bestIndex, const PerturbedVector pertV, const
      NumberDiffVectors diffV, const CrossoverStrat crossStrat, const double sf1,
      const double sf2, const double crFactor)
00225 {
00226     const size_t dim = mainPop->getDimensionsSize();
00227     std::uniform_int_distribution<long> rdim(0, dim - 1);
00228     size_t pertIndex = 0;
00229 
00230     // Get the index of the perturbed vector
00231     switch (pertV)
00232     {
00233         case PerturbedVector::Best:
00234             pertIndex = bestIndex;
00235             break;
00236         case PerturbedVector::Random:
00237             do
00238             {
00239                 pertIndex = rdim(engine);
00240             } while (pertIndex == vIndex);
00241             break;
00242         case PerturbedVector::RandToBest:
00243             pertIndex = vIndex;
00244             break;
00245     }
00246 
00247     // Calculate unique random vector indices
00248     std::set<size_t> randVectorIndices;
00249 
00250     unsigned int numRandVectors = 2;
00251     if (diffV == NumberDiffVectors::Two)
00252         numRandVectors = 4;
00253 
00254     while (randVectorIndices.size() < numRandVectors)
00255     {
00256         auto rIndex = rdim(engine);
00257         if (rIndex != pertIndex && rIndex != bestIndex)
00258             randVectorIndices.insert(rIndex);
00259     }
00260 
00261     T* randV[MAX_RAND_VECTOR_SELECT] = { };
00262     int rvIndex = 0;
00263 
00264     // Convert random vector indices to population vector pointers
00265     for (auto it = randVectorIndices.begin(); it != randVectorIndices.end(); it++)
00266     {
00267         randV[rvIndex] = mainPop->getPopulationPtr(*it);
00268         rvIndex++;
00269     }
00270 
00271     // Get population vector points for parent vector, purturbed vector, etc
00272     T* curVParent = mainPop->getPopulationPtr(vIndex);
00273     T* curVPert = mainPop->getPopulationPtr(pertIndex);
00274 
00275     nextPop->copyPopulation(vIndex, curVParent);
00276 
00277     T* curVNext = nextPop->getPopulationPtr(vIndex);
00278     T* curVBest = mainPop->getPopulationPtr(bestIndex);
00279 
00280     size_t count = 0;
00281 
00282     if (crossStrat == CrossoverStrat::Exponential)
00283     {
00284         // Mutate with exponential crossover
00285         // Select random starting dimension
00286         size_t d = rdim(engine);
00287 
00288         do
00289         {
00290             if (pertV == PerturbedVector::RandToBest)
00291             {
00292                 curVNext[d] = curVParent[d] + (sf2 * (curVBest[d] - curVParent[d])) + (sf1 * (randV[0][d] -
       randV[1][d]));
00293             }
00294             else
00295             {
00296                 if (numRandVectors == 2)
00297                     curVNext[d] = curVPert[d] + (sf1 * (randV[0][d] - randV[1][d]));
00298                 else
00299                     curVNext[d] = curVPert[d] + (sf1 * (randV[0][d] + randV[1][d] - randV[2][d] - randV[3][
      d]));
00300             }
00301             d = (d + 1) % dim;
00302         } while (rchance(engine) < crFactor);
00303     }
00304     else
00305     {
```

```
00306         // Mutate with binomial crossover
00307
00308         for (size_t d = 0; d < dim; d++)
00309         {
00310             if (rchance(engine) > crFactor)
00311             continue;
00312
00313             if (pertV == PerturbedVector::RandToBest)
00314             {
00315                 curVNext[d] = curVParent[d] + (sf2 * (curVBest[d] - curVParent[d])) + (sf1 * (randV[0][d] -
      randV[1][d]));
00316             }
00317             else
00318             {
00319                 if (numRandVectors == 2)
00320                     curVNext[d] = curVPert[d] + (sf1 * (randV[0][d] - randV[1][d]));
00321                 else
00322                     curVNext[d] = curVPert[d] + (sf1 * (randV[0][d] + randV[1][d] - randV[2][d] - randV[3][
      d]));
00323             }
00324         }
00325     }
00326 }
00327
00338 template <class T>
00339 void mfunc::DifferentialEvolution<T>::select(
      mdata::Population<T>* mainPop, mdata::Population<T>* nextPop,
      size_t vIndex, mfunc::mfuncPtr<T> fPtr)
00340 {
00341     nextPop->calcFitness(vIndex, fPtr);
00342     auto newFit = nextPop->getFitness(vIndex);
00343     auto oldFit = mainPop->getFitness(vIndex);
00344
00345     if (newFit > oldFit)
00346     {
00347         // Reset (Discard) new pop vector back to last generation
00348         nextPop->copyPopulation(vIndex, mainPop->getPopulationPtr(vIndex));
00349         nextPop->setFitness(vIndex, oldFit);
00350     }
00351 }
00352
00362 template <class T>
00363 void mfunc::DifferentialEvolution<T>::parseDEStrat(
      DEStrategy mainStrat, PerturbedVector& oPertv,
      NumberDiffVectors& oDiffv, CrossoverStrat& oCross)
00364 {
00365     switch (mainStrat)
00366     {
00367         case DEStrategy::Best1Exp:
00368             oPertv = PerturbedVector::Best;
00369             oDiffv = NumberDiffVectors::One;
00370             oCross = CrossoverStrat::Exponential;
00371             return;
00372         case DEStrategy::Rand1Exp:
00373             oPertv = PerturbedVector::Random;
00374             oDiffv = NumberDiffVectors::One;
00375             oCross = CrossoverStrat::Exponential;
00376             return;
00377         case DEStrategy::RandToBest1Exp:
00378             oPertv = PerturbedVector::RandToBest;
00379             oDiffv = NumberDiffVectors::One;
00380             oCross = CrossoverStrat::Exponential;
00381             return;
00382         case DEStrategy::Best2Exp:
00383             oPertv = PerturbedVector::Best;
00384             oDiffv = NumberDiffVectors::Two;
00385             oCross = CrossoverStrat::Exponential;
00386             return;
00387         case DEStrategy::Rand2Exp:
00388             oPertv = PerturbedVector::Random;
00389             oDiffv = NumberDiffVectors::Two;
00390             oCross = CrossoverStrat::Exponential;
00391             return;
00392         case DEStrategy::Best1Bin:
00393             oPertv = PerturbedVector::Best;
00394             oDiffv = NumberDiffVectors::One;
00395             oCross = CrossoverStrat::Binomial;
00396             return;
00397         case DEStrategy::Rand1Bin:
00398             oPertv = PerturbedVector::Random;
00399             oDiffv = NumberDiffVectors::One;
00400             oCross = CrossoverStrat::Binomial;
00401             return;
00402         case DEStrategy::RandToBest1Bin:
00403             oPertv = PerturbedVector::RandToBest;
00404             oDiffv = NumberDiffVectors::One;
00405             oCross = CrossoverStrat::Binomial;
```

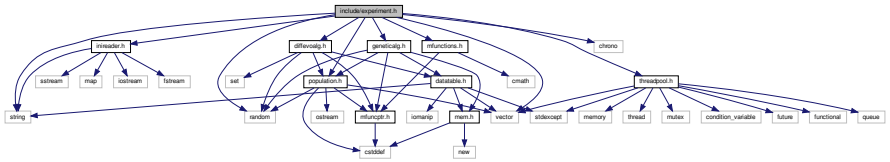
```
00406             return;
00407         case DEStrategy::Best2Bin:
00408             oPertv = PerturbedVector::Best;
00409             oDiffv = NumberDiffVectors::Two;
00410             oCross = CrossoverStrat::Binomial;
00411             return;
00412         case DEStrategy::Rand2Bin:
00413             oPertv = PerturbedVector::Random;
00414             oDiffv = NumberDiffVectors::Two;
00415             oCross = CrossoverStrat::Binomial;
00416             return;
00417         default:
00418             oPertv = PerturbedVector::Best;
00419             oDiffv = NumberDiffVectors::One;
00420             oCross = CrossoverStrat::Exponential;
00421             return;
00422     }
00423 }
00424
00425 #endif
```

6.5 include/experiment.h File Reference

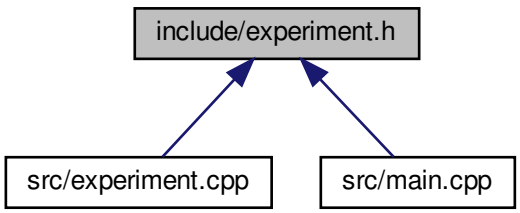
Header file for the Experiment class. Contains the basic logic and functions to run the cs471 project experiment.

```
#include <string>
#include <random>
#include <chrono>
#include <vector>
#include "mfunctions.h"
#include "inireader.h"
#include "population.h"
#include "threadpool.h"
#include "geneticalg.h"
#include "diffevoalg.h"
```

Include dependency graph for experiment.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct mfunc::RandomBounds< T >

  Simple struct for storing the minimum and maximum input vector bounds for a function.
- class mfunc::Experiment< T >

  Contains classes for running the CS471 project experiment.

Namespaces

- mfunc

Enumerations

- enum mfunc::Algorithm { mfunc::Algorithm::GeneticAlgorithm = 0, mfunc::Algorithm::DifferentialEvolution = 1, mfunc::Algorithm::Count = 2 }

  Simple enum that selects one of the evolutionary algorithms.

6.5.1 Detailed Description

Header file for the Experiment class. Contains the basic logic and functions to run the cs471 project experiment.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.3

Date

2019-04-01

Copyright

Definition in file experiment.h.

6.6 experiment.h

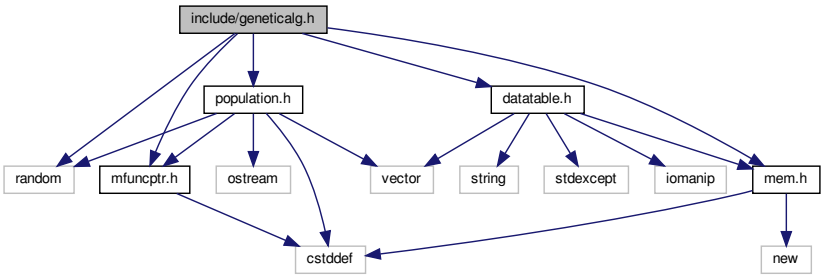
```
00001
00013 #ifndef __EXPERIMENT_H
00014 #define __EXPERIMENT_H
00015
00016 #include <string>
00017 #include <random>
00018 #include <chrono>
00019 #include <vector>
00020 #include "mfunctions.h"
00021 #include "inireader.h"
00022 #include "population.h"
00023 #include "threadpool.h"
00024 #include "geneticalg.h"
00025 #include "diffevoalg.h"
00026
00027 namespace mfunc
00028 {
00033     template<class T>
00034     struct RandomBounds
00035     {
00036         T min = 0.0;
00037         T max = 0.0;
00038     };
00039
00043     enum class Algorithm
00044     {
00045         GeneticAlgorithm = 0,
00046         DifferentialEvolution = 1,
00047         Count = 2
00048     };
00049
00060     template<class T>
00061     class Experiment
00062     {
00063     public:
00064         Experiment();
00065         ~Experiment();
00066         bool init(const char* paramFile);
00067         int testAllFunc();
00068
00069         int testAllFunc_GA();
00070         int runGAThreaded(GAParams<T> gaParams,
      mdata::DataTable<double>* tTable, size_t tRow, size_t tCol);
00071
00072         int testAllFunc_DE();
00073         int runDEThreaded(DEParams<T> deParams,
      mdata::DataTable<double>* tTable, size_t tRow, size_t tCol);
00074     private:
00075         std::mutex popPoolMutex;
00076         util::IniReader iniParams;
00077         std::vector<mdata::Population<T>*> populationsPool;
00078         std::string resultsFile;
00079         std::string execTimesFile;
00080         RandomBounds<T>* vBounds;
00081         ThreadPool* tPool;
00082         size_t iterations;
00083         Algorithm testAlg;
00084
00085         bool loadGAParams(GAParams<T>& refParams);
00086         bool loadDEParams(DEParams<T>& refParams);
00087
00088         mdata::Population<T>* popPoolRemove();
00089         void popPoolAdd(mdata::Population<T>* popPtr);
00090
00091         bool parseFuncBounds();
00092
00093         bool allocatePopulationPool(size_t count, size_t popSize, size_t dimensions);
00094         void releasePopulationPool();
00095
00096         bool allocateVBounds();
00097         void releaseVBounds();
00098
00099         bool allocateThreadPool(size_t numThreads);
00100         void releaseThreadPool();
00101     };
00102 } // mfunc
00103
00104 #endif
00105
00106 // ========================
00107 // End of experiment.h
00108 // ========================
```

6.7 include/geneticalg.h File Reference

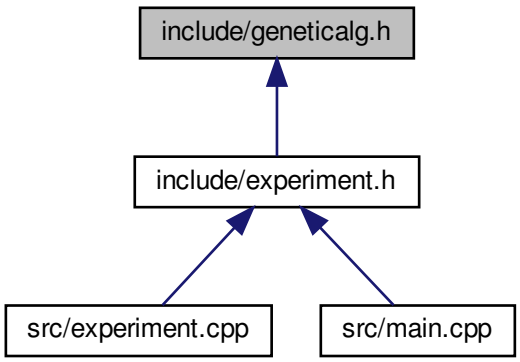
Implementation of the GeneticAlgorithm class. Executes the genetic algorithm with the specified parameters.

```
#include "population.h"
#include "mfuncptr.h"
#include "datatable.h"
#include "random"
#include "mem.h"
```

Include dependency graph for geneticalg.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct mfunc::GAParams< T >

  Simple structure that holds various parameters for the genetic algorithm.
- class mfunc::GeneticAlgorithm< T >

  The GeneticAlgorithm class executes the genetic algorithm with the specified parameters. To start, execute the run() function.

Namespaces

- mfunc

6.7.1 Detailed Description

Implementation of the GeneticAlgorithm class. Executes the genetic algorithm with the specified parameters.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-04-27

Copyright

Copyright (c) 2019

Definition in file geneticalg.h.

6.8 geneticalg.h

```
00001
00013 #ifndef __GENETICALG_H
00014 #define __GENETICALG_H
00015
00016 #include "population.h"
00017 #include "mfuncptr.h"
00018 #include "datatable.h"
00019 #include "random"
00020 #include "mem.h"
00021
00022 namespace mfunc
00023 {
00030     template <class T>
00031     struct GAParams
00032     {
00033         mdata::DataTable<T>* fitnessTable;
00034         size_t fitTableCol;
00035         mdata::Population<T>* mainPop;
00036         mdata::Population<T>* auxPop;
00037         mfuncPtr<T> fPtr;
00038         T fMinBound;
00039         T fMaxBound;
00040         unsigned int generations;
00041         double crProb;
00042         double mutProb;
00043         double mutRange;
00044         double mutPrec;
00045         double elitismRate;
00046
00047         GAParams()
00048         {
00049             fitnessTable = nullptr;
00050             fitTableCol = 0;
00051             mainPop = nullptr;
```

```
00052             auxPop = nullptr;
00053             fPtr = nullptr;
00054             fMinBound = 0;
00055             fMaxBound = 0;
00056             generations = 0;
00057             crProb = 0;
00058             mutProb = 0;
00059             mutRange = 0;
00060             mutPrec = 0;
00061             elitismRate = 0;
00062         }
00063     };
00064
00071     template <class T>
00072     class GeneticAlgorithm
00073     {
00074     public:
00075         GeneticAlgorithm();
00076         ~GeneticAlgorithm() = default;
00077         int run(GAParams<T> params);
00078
00079     private:
00080         std::random_device seed;
00081         std::mt19937 engine;
00082         std::uniform_real_distribution<double> rchance;
00083
00084         void select(mdata::Population<T>* pop, size_t& outP1, size_t& outP2);
00085         size_t selRW(mdata::Population<T>* pop);
00086         void crossover(size_t dim, T* p1, T* p2, double cr, T* outCh1, T* outCh2);
00087         void mutate(size_t dim, T* s, double mutProb, double mutRange, double
      mutPrec, T fMin, T fMax);
00088         void reduce(mdata::Population<T>* oldPop,
      mdata::Population<T>* newPop, size_t elitism);
00089     };
00090 }
00091
00097 template <class T>
00098 mfunc::GeneticAlgorithm<T>::GeneticAlgorithm()
00099     : seed(), engine(seed()), rchance(0, 1)
00100 {
00101 }
00102
00110 template <class T>
00111 int mfunc::GeneticAlgorithm<T>::run(GAParams<T> p)
00112 {
00113     if (p.mainPop == nullptr || p.auxPop == nullptr || p.fPtr == nullptr)
00114         return 1;
00115
00116     if (p.mainPop->getPopulationSize() != p.auxPop->getPopulationSize() ||
00117         p.mainPop->getDimensionsSize() != p.auxPop->getDimensionsSize())
00118         return 2;
00119
00120     // Get population information
00121     const size_t popSize = p.mainPop->getPopulationSize();
00122     const size_t dimSize = p.mainPop->getDimensionsSize();
00123     const size_t elitism = p.elitismRate * (double)popSize;
00124
00125     // Allocate child buffers
00126     T* childOne = util::allocArray<T>(dimSize);
00127     T* childTwo = util::allocArray<T>(dimSize);
00128
00129     // Prepare populations
00130     p.mainPop->setFitnessNormalization(true);
00131     p.auxPop->setFitnessNormalization(true);
00132     p.mainPop->generate(p.fMinBound, p.fMaxBound);
00133     p.mainPop->calcAllFitness(p.fPtr);
00134
00135     // Loop for a number of generations
00136     for (unsigned int gen = 0; gen < p.generations; gen++)
00137     {
00138         for (size_t s = 0; s < popSize;)
00139         {
00140             size_t p1Index = 0;
00141             size_t p2Index = 0;
00142
00143             // Select parent indices
00144             select(p.mainPop, p1Index, p2Index);
00145
00146             // Produce new children by computing the crossover between parents
00147             crossover(dimSize, p.mainPop->getPopulationPtr(p1Index), p.
      mainPop->getPopulationPtr(p2Index),
00148                 p.crProb, childOne, childTwo);
00149
00150             // Mutate children
00151             mutate(dimSize, childOne, p.mutProb, p.mutRange, p.
      mutPrec, p.fMinBound, p.fMaxBound);
00152             mutate(dimSize, childTwo, p.mutProb, p.mutRange, p.
```

```
    mutPrec, p.fMinBound, p.fMaxBound);
00153
00154             // Copy new children into next generation
00155             p.auxPop->copyPopulation(s, childOne);
00156             s++;
00157             if (s < popSize) p.auxPop->copyPopulation(s, childTwo);
00158             s++;
00159         }
00160
00161         // Recalculate all fitness values for next generation
00162         p.auxPop->calcAllFitness(p.fPtr);
00163
00164         // Select and combine some of the best populations from the previous generation
00165         // with the next generation
00166         reduce(p.mainPop, p.auxPop, elitism);
00167
00168         // Swap the two populations
00169         auto tmp = p.mainPop;
00170         p.mainPop = p.auxPop;
00171         p.auxPop = tmp;
00172
00173         // Record current best population in results table
00174         p.fitnessTable->setEntry(gen, p.fitTableCol, p.
    mainPop->getMinCost());
00175     }
00176
00177     // Delete children buffers
00178     util::releaseArray<T>(childOne);
00179     util::releaseArray<T>(childTwo);
00180
00181     return 0;
00182 }
00183
00195 template <class T>
00196 void mfunc::GeneticAlgorithm<T>::select(
    mdata::Population<T>* pop, size_t& outP1, size_t& outP2)
00197 {
00198     outP1 = selRW(pop);
00199     outP2 = selRW(pop);
00200 }
00201
00209 template <class T>
00210 size_t mfunc::GeneticAlgorithm<T>::selRW(
    mdata::Population<T>* pop)
00211 {
00212     std::uniform_real_distribution<T> dist(0, pop->getTotalFitness());
00213     const size_t pSize = pop->getPopulationSize();
00214     T r = dist(engine);
00215     size_t s = 0;
00216
00217     while (s < pSize - 1 && r > 0)
00218     {
00219         r -= pop->getFitness(s);
00220         s += 1;
00221     }
00222
00223     return s;
00224 }
00225
00237 template <class T>
00238 void mfunc::GeneticAlgorithm<T>::crossover(size_t dim, T* p1, T* p2,
    double cr, T* outCh1, T* outCh2)
00239 {
00240     std::uniform_int_distribution<long> rdim(0, dim - 1);
00241
00242     if (rchance(engine) < cr)
00243     {
00244         auto crIndex = rdim(engine);
00245         for (size_t i = 0; i < dim; i++)
00246         {
00247             if (i < crIndex)
00248             {
00249                 outCh1[i] = p1[i];
00250                 outCh2[i] = p2[i];
00251             }
00252             else
00253             {
00254                 outCh1[i] = p2[i];
00255                 outCh2[i] = p1[i];
00256             }
00257         }
00258     }
00259     else
00260     {
00261         for (size_t i = 0; i < dim; i++)
00262         {
00263             outCh1[i] = p1[i];
```

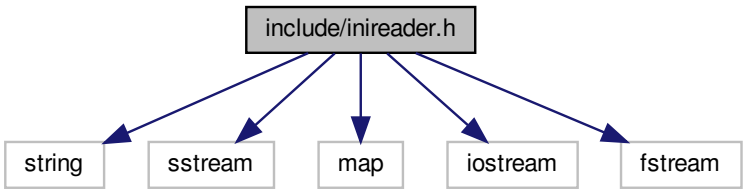
```
00264             outCh2[i] = p2[i];
00265         }
00266     }
00267 }
00268
00281 template <class T>
00282 void mfunc::GeneticAlgorithm<T>::mutate(size_t dim, T* s, double mutProb,
       double mutRange, double mutPrec, T fMin, T fMax)
00283 {
00284     std::uniform_real_distribution<double> rflip(-1, 1);
00285
00286     for (size_t i = 0; i < dim; i++)
00287     {
00288         if (rchance(engine) < mutProb)
00289         {
00290             s[i] += rflip(engine) * (fMax - fMin) * mutRange * std::pow(2, (-1 * rchance(engine) * mutPrec)
      );
00291
00292             if (s[i] < fMin) s[i] = fMin;
00293             else if (s[i] > fMax) s[i] = fMax;
00294         }
00295     }
00296 }
00297
00306 template <class T>
00307 void mfunc::GeneticAlgorithm<T>::reduce(
      mdata::Population<T>* oldPop, mdata::Population<T>* newPop, size_t
      elitism)
00308 {
00309     oldPop->sortDescendByFitness();
00310     newPop->sortDescendByFitness();
00311     auto const lastIndex = newPop->getPopulationSize() - 1;
00312
00313     for (size_t i = 0; i < elitism; i++)
00314     {
00315         newPop->copyPopulation(lastIndex - i, oldPop->
      getPopulationPtr(i));
00316     }
00317 }
00318
00319 #endif
```

6.9 include/inireader.h File Reference

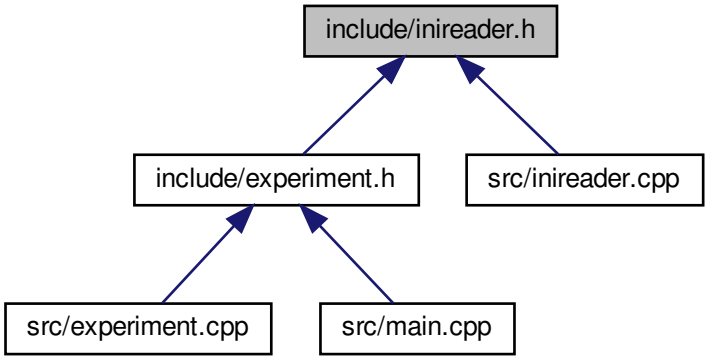
Header file for the IniReader class, which can open and parse simple ∗.ini files.

```
#include <string>
#include <sstream>
#include <map>
#include <iostream>
#include <fstream>
```

Include dependency graph for inireader.h:



This graph shows which files directly or indirectly include this file:



Classes

- class util::IniReader

  The IniReader class is a simple ∗.ini file reader and parser.

Namespaces

- util

6.9.1 Detailed Description

Header file for the IniReader class, which can open and parse simple ∗.ini files.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-04-01

Copyright

Copyright (c) 2019

Definition in file inireader.h.

6.10 inireader.h

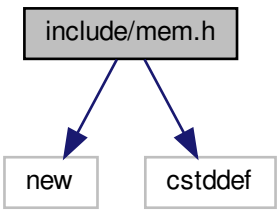
```
00001
00013 #ifndef __INIREADER_H
00014 #define __INIREADER_H
00015
00016 #include <string>
00017 #include <sstream>
00018 #include <map>
00019 #include <iostream>
00020 #include <fstream>
00021
00022 namespace util
00023 {
00046     class IniReader
00047     {
00048     public:
00049         IniReader();
00050         ~IniReader();
00051         bool openFile(std::string filePath);
00052         bool sectionExists(std::string section);
00053         bool entryExists(std::string section, std::string entry);
00054         std::string getEntry(std::string section, std::string entry);
00055
00056         template <class T>
00057         T getEntryAs(std::string section, std::string entry)
00058         {
00059             std::stringstream ss(getEntry(section, entry));
00060             T retVal;
00061             ss >> retVal;
00062             return retVal;
00063         }
00064     private:
00065         std::string file;
00066         std::map<std::string, std::map<std::string, std::string>> iniMap;
00068         bool parseFile();
00069         void parseEntry(const std::string& sectionName, const std::string& entry);
00070     };
00071 }
00072
00073 #endif
00074
00075 // ========================
00076 // End of inireader.h
00077 // ========================
```

6.11 include/mem.h File Reference

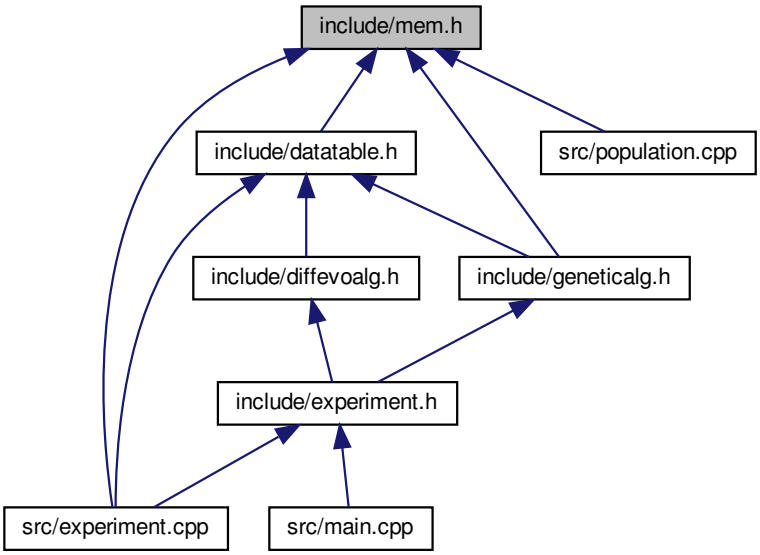
Header file for various memory utility functions.

```
#include <new>
#include <cstddef>
```

Include dependency graph for mem.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- util

Functions

- template<class T = double>
  void util::initArray (T *a, size_t size, T val)

  Initializes an array with some set value.
- template<class T = double>
  void util::initMatrix (T **m, size_t rows, size_t cols, T val)

  Initializes a matrix with a set value for each entry.
- template<class T = double>
  bool util::releaseArray (T *&a)

  Releases an allocated array's memory and sets the pointer to nullptr.
- template<class T = double>
  void util::releaseMatrix (T **&m, size_t rows)

  Releases an allocated matrix's memory and sets the pointer to nullptr.
- template<class T = double>
  T * util::allocArray (size_t size)

  Allocates a new array of the given data type.
- template<class T = double>
  T ** util::allocMatrix (size_t rows, size_t cols)

  Allocates a new matrix of the given data type.
- template<class T = double>
  void util::copyArray (T *src, T *dest, size_t size)

  Copies the elements from one equal-sized array to another.

6.11.1 Detailed Description

Header file for various memory utility functions.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.2

Date

2019-04-02

Copyright

Definition in file mem.h.

6.12 mem.h

```
00001
00012 #ifndef __MEM_H
00013 #define __MEM_H
00014
00015 #include <new> // std::nothrow
00016 #include <cstddef> // size_t definition
00017
00018 namespace util
00019 {
00028     template <class T = double>
00029     inline void initArray(T* a, size_t size, T val)
00030     {
00031         if (a == nullptr) return;
00032
00033         for (size_t i = 0; i < size; i++)
00034         {
00035             a[i] = val;
00036         }
00037     }
00038
00048     template <class T = double>
00049     inline void initMatrix(T** m, size_t rows, size_t cols, T val)
00050     {
00051         if (m == nullptr) return;
00052
00053         for (size_t i = 0; i < rows; i++)
00054         {
00055             initArray(m[i], cols, val);
00056         }
00057     }
00058
00065     template <class T = double>
00066     bool releaseArray(T*& a)
00067     {
00068         if (a == nullptr) return true;
00069
00070         try
00071         {
00072             delete[] a;
00073             a = nullptr;
00074             return true;
00075         }
```

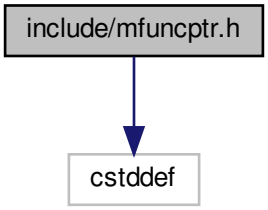
```
00076         catch(...)
00077         {
00078             return false;
00079         }
00080     }
00081
00089     template <class T = double>
00090     void releaseMatrix(T**& m, size_t rows)
00091     {
00092         if (m == nullptr) return;
00093
00094         for (size_t i = 0; i < rows; i++)
00095         {
00096             if (m[i] != nullptr)
00097             {
00098                 // Release each row
00099                 releaseArray<T>(m[i]);
00100             }
00101         }
00102
00103         // Release columns
00104         delete[] m;
00105         m = nullptr;
00106     }
00107
00115     template <class T = double>
00116     inline T* allocArray(size_t size)
00117     {
00118         return new(std::nothrow) T[size];
00119     }
00120
00129     template <class T = double>
00130     inline T** allocMatrix(size_t rows, size_t cols)
00131     {
00132         T** m = (T**)allocArray<T*>(rows);
00133         if (m == nullptr) return nullptr;
00134
00135         for (size_t i = 0; i < rows; i++)
00136         {
00137             m[i] = allocArray<T>(cols);
00138             if (m[i] == nullptr)
00139             {
00140                 releaseMatrix<T>(m, rows);
00141                 return nullptr;
00142             }
00143         }
00144
00145         return m;
00146     }
00147
00156     template <class T = double>
00157     inline void copyArray(T* src, T* dest, size_t size)
00158     {
00159         for (size_t i = 0; i < size; i++)
00160             dest[i] = src[i];
00161     }
00162 }
00163
00164 #endif
00165
00166 // ==========================
00167 // End of mem.h
00168 // ==========================
```

6.13 include/mfuncptr.h File Reference

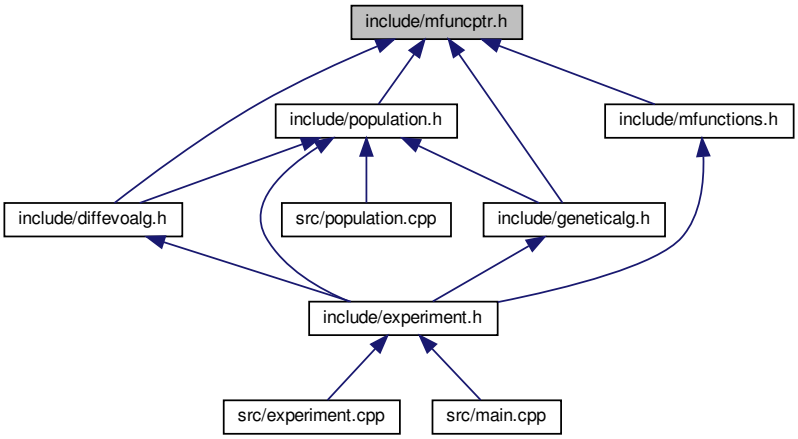
Contains the type definition for mfuncPtr, a templated function pointer to one of the math functions in mfunctions.h.

```
#include <cstddef>
```

Include dependency graph for mfuncptr.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- mfunc

Typedefs

- template<class T >
  using mfunc::mfuncPtr = T(∗)(T ∗, size_t)

  Function pointer that takes two arguments T∗ and size_t, and returns a T value.

6.13.1 Detailed Description

Contains the type definition for mfuncPtr, a templated function pointer to one of the math functions in mfunctions.h.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-04-19

Copyright

Copyright (c) 2019

Definition in file mfuncptr.h.

6.14 mfuncptr.h

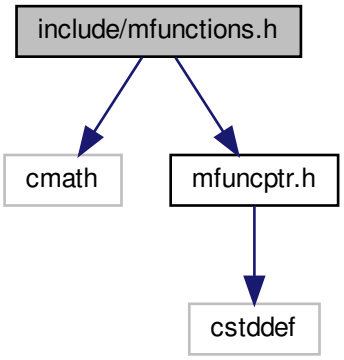
```
00001
00014 #ifndef __MFUNCPTR_H
00015 #define __MFUNCPTR_H
00016
00017 #include <cstddef> // size_t definition
00018
00019 namespace mfunc
00020 {
00027     template <class T>
00028     using mfuncPtr = T (*)(T*, size_t);
00029 }
00030
00031 #endif
00032
00033 // ========================
00034 // End of mfuncptr.h
00035 // ========================
```

6.15 include/mfunctions.h File Reference

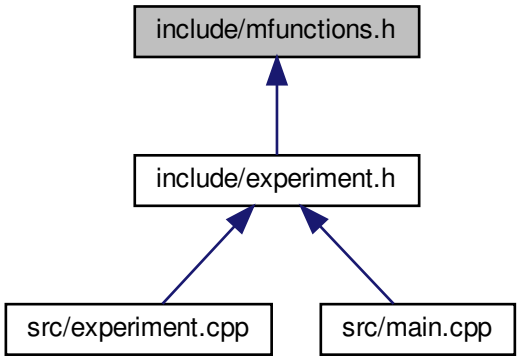
Contains various math function definitions.

```
#include <cmath>
#include "mfuncptr.h"
```

Include dependency graph for mfunctions.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct mfunc::FunctionDesc

  get() returns a function's description Returns a C-string description for the given function id if the id is valid. Otherwise returns null

- struct mfunc::Functions< T >

  Struct containing all static math functions. A function can be called directly by name, or indirectly using Functions::get or Functions::exec.

Namespaces

- mfunc

Macros

- #define _USE_MATH_DEFINES
- #define _NUM_FUNCTIONS 18
- #define _schwefelDesc "Schwefel's function"
- #define _dejongDesc "1st De Jong's function"
- #define _rosenbrokDesc "Rosenbrock"
- #define _rastriginDesc "Rastrigin"
- #define _griewangkDesc "Griewangk"
- #define _sineEnvelopeSineWaveDesc "Sine Envelope Sine Wave"
- #define _stretchedVSineWaveDesc "Stretched V Sine Wave"
- #define _ackleysOneDesc "Ackley's One"
- #define _ackleysTwoDesc "Ackley's Two"
- #define _eggHolderDesc "Egg Holder"
- #define _ranaDesc "Rana"
- #define _pathologicalDesc "Pathological"
- #define _michalewiczDesc "Michalewicz"
- #define _mastersCosineWaveDesc "Masters Cosine Wave"
- #define _quarticDesc "Quartic"
- #define _levyDesc "Levy"
- #define _stepDesc "Step"
- #define _alpineDesc "Alpine"
- #define _schwefelId 1
- #define _dejongId 2
- #define _rosenbrokId 3
- #define _rastriginId 4
- #define _griewangkId 5
- #define _sineEnvelopeSineWaveId 6
- #define _stretchedVSineWaveId 7
- #define _ackleysOneId 8
- #define _ackleysTwoId 9
- #define _eggHolderId 10
- #define _ranaId 11
- #define _pathologicalId 12
- #define _michalewiczId 13
- #define _mastersCosineWaveId 14
- #define _quarticId 15
- #define _levyId 16
- #define _stepId 17
- #define _alpineId 18

Variables

- constexpr const unsigned int mfunc::NUM_FUNCTIONS = _NUM_FUNCTIONS

6.15.1 Detailed Description

Contains various math function definitions.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-03-29

Copyright

Definition in file mfunctions.h.

6.15.2 Macro Definition Documentation

6.15.2.1 _ackleysOneDesc

```
#define _ackleysOneDesc "Ackley's One"
```

Definition at line 29 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get().

6.15.2.2 _ackleysOneId

```
#define _ackleysOneId 8
```

Definition at line 48 of file mfunctions.h.

Referenced by mfunc::Functions< T >::ackleysOne(), mfunc::FunctionDesc::get(), and mfunc::Functions< T >::get().

6.15.2.3 _ackleysTwoDesc

```
#define _ackleysTwoDesc "Ackley's Two"
```

Definition at line 30 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get().

6.15.2.4 _ackleysTwoId

```
#define _ackleysTwoId 9
```

Definition at line 49 of file mfunctions.h.

Referenced by mfunc::Functions< T >::ackleysTwo(), mfunc::FunctionDesc::get(), and mfunc::Functions< T >::get().

6.15.2.5 _alpineDesc

```
#define _alpineDesc "Alpine"
```

Definition at line 39 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get().

6.15.2.6 _alpineId

```
#define _alpineId 18
```

Definition at line 58 of file mfunctions.h.

Referenced by mfunc::Functions< T >::alpine(), mfunc::FunctionDesc::get(), and mfunc::Functions< T >::get().

6.15.2.7 _dejongDesc

```
#define _dejongDesc "1st De Jong's function"
```

Definition at line 23 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get().

6.15.2.8 _dejongId

```
#define _dejongId 2
```

Definition at line 42 of file mfunctions.h.

Referenced by mfunc::Functions< T >::dejong(), mfunc::FunctionDesc::get(), and mfunc::Functions< T >::get().

6.15.2.9 _eggHolderDesc

```
#define _eggHolderDesc "Egg Holder"
```

Definition at line 31 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get().

6.15.2.10 _eggHolderId

```
#define _eggHolderId 10
```

Definition at line 50 of file mfunctions.h.

Referenced by mfunc::Functions< T >::eggHolder(), mfunc::FunctionDesc::get(), and mfunc::Functions< T >::get().

6.15.2.11 _griewangkDesc

```
#define _griewangkDesc "Griewangk"
```

Definition at line 26 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get().

6.15.2.12 _griewangkId

```
#define _griewangkId 5
```

Definition at line 45 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get(), mfunc::Functions< T >::get(), and mfunc::Functions< T >::griewangk().

6.15.2.13 _levyDesc

```
#define _levyDesc "Levy"
```

Definition at line 37 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get().

6.15.2.14 _levyId

```
#define _levyId 16
```

Definition at line 56 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get(), mfunc::Functions< T >::get(), and mfunc::Functions< T >::levy().

6.15.2.15 _mastersCosineWaveDesc

```
#define _mastersCosineWaveDesc "Masters Cosine Wave"
```

Definition at line 35 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get().

6.15.2.16 _mastersCosineWaveId

```
#define _mastersCosineWaveId 14
```

Definition at line 54 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get(), mfunc::Functions< T >::get(), and mfunc::Functions< T >::masters↩ CosineWave().

6.15.2.17 _michalewiczDesc

```
#define _michalewiczDesc "Michalewicz"
```

Definition at line 34 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get().

6.15.2.18 _michalewiczId

```
#define _michalewiczId 13
```

Definition at line 53 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get(), mfunc::Functions< T >::get(), and mfunc::Functions< T >::michalewicz().

6.15.2.19 _NUM_FUNCTIONS

```
#define _NUM_FUNCTIONS 18
```

Definition at line 20 of file mfunctions.h.

Referenced by mfunc::Functions< T >::getCallCounter(), and mfunc::Functions< T >::resetCallCounters().

6.15.2.20 _pathologicalDesc

```
#define _pathologicalDesc "Pathological"
```

Definition at line 33 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get().

6.15.2.21 _pathologicalId

```
#define _pathologicalId 12
```

Definition at line 52 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get(), mfunc::Functions< T >::get(), and mfunc::Functions< T >::pathological().

6.15.2.22 _quarticDesc

```
#define _quarticDesc "Quartic"
```

Definition at line 36 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get().

6.15.2.23 _quarticId

```
#define _quarticId 15
```

Definition at line 55 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get(), mfunc::Functions< T >::get(), and mfunc::Functions< T >::quartic().

6.15.2.24 _ranaDesc

```
#define _ranaDesc "Rana"
```

Definition at line 32 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get().

6.15.2.25 _ranaId

```
#define _ranaId 11
```

Definition at line 51 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get(), mfunc::Functions< T >::get(), and mfunc::Functions< T >::rana().

6.15.2.26 _rastriginDesc

```
#define _rastriginDesc "Rastrigin"
```

Definition at line 25 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get().

6.15.2.27 _rastriginId

```
#define _rastriginId 4
```

Definition at line 44 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get(), mfunc::Functions< T >::get(), and mfunc::Functions< T >::rastrigin().

6.15.2.28 _rosenbrokDesc

```
#define _rosenbrokDesc "Rosenbrock"
```

Definition at line 24 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get().

6.15.2.29 _rosenbrokId

```
#define _rosenbrokId 3
```

Definition at line 43 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get(), mfunc::Functions< T >::get(), and mfunc::Functions< T >::rosenbrok().

6.15.2.30 _schwefelDesc

```
#define _schwefelDesc "Schwefel's function"
```

Definition at line 22 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get().

6.15.2.31 _schwefelId

```
#define _schwefelId 1
```

Definition at line 41 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get(), mfunc::Functions< T >::get(), and mfunc::Functions< T >::schwefel().

6.15.2.32 _sineEnvelopeSineWaveDesc

```
#define _sineEnvelopeSineWaveDesc "Sine Envelope Sine Wave"
```

Definition at line 27 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get().

6.15.2.33 _sineEnvelopeSineWaveId

```
#define _sineEnvelopeSineWaveId 6
```

Definition at line 46 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get(), mfunc::Functions< T >::get(), and mfunc::Functions< T >::sineEnvelopeSineWave().

6.15.2.34 _stepDesc

```
#define _stepDesc "Step"
```

Definition at line 38 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get().

6.15.2.35 _stepId

```
#define _stepId 17
```

Definition at line 57 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get(), mfunc::Functions< T >::get(), and mfunc::Functions< T >::step().

6.15.2.36 _stretchedVSineWaveDesc

```
#define _stretchedVSineWaveDesc "Stretched V Sine Wave"
```

Definition at line 28 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get().

6.15.2.37 _stretchedVSineWaveId

```
#define _stretchedVSineWaveId 7
```

Definition at line 47 of file mfunctions.h.

Referenced by mfunc::FunctionDesc::get(), mfunc::Functions< T >::get(), and mfunc::Functions< T >::stretchedVSineWave().

6.15.2.38 _USE_MATH_DEFINES

```
#define _USE_MATH_DEFINES
```

Definition at line 15 of file mfunctions.h.

6.16 mfunctions.h

```
00001
00012 #ifndef __MFUNCIONS_H
00013 #define __MFUNCIONS_H
00014
00015 #define _USE_MATH_DEFINES
00016
00017 #include <cmath>
00018 #include "mfuncptr.h"
00019
00020 #define _NUM_FUNCTIONS 18
00021
00022 #define _schwefelDesc "Schwefel's function"
00023 #define _dejongDesc "1st De Jong's function"
00024 #define _rosenbrokDesc "Rosenbrock"
00025 #define _rastriginDesc "Rastrigin"
00026 #define _griewangkDesc "Griewangk"
00027 #define _sineEnvelopeSineWaveDesc "Sine Envelope Sine Wave"
00028 #define _stretchedVSineWaveDesc "Stretched V Sine Wave"
00029 #define _ackleysOneDesc "Ackley's One"
00030 #define _ackleysTwoDesc "Ackley's Two"
00031 #define _eggHolderDesc "Egg Holder"
00032 #define _ranaDesc "Rana"
00033 #define _pathologicalDesc "Pathological"
00034 #define _michalewiczDesc "Michalewicz"
00035 #define _mastersCosineWaveDesc "Masters Cosine Wave"
00036 #define _quarticDesc "Quartic"
00037 #define _levyDesc "Levy"
00038 #define _stepDesc "Step"
00039 #define _alpineDesc "Alpine"
00040
00041 #define _schwefelId 1
00042 #define _dejongId 2
00043 #define _rosenbrokId 3
00044 #define _rastriginId 4
00045 #define _griewangkId 5
00046 #define _sineEnvelopeSineWaveId 6
00047 #define _stretchedVSineWaveId 7
00048 #define _ackleysOneId 8
00049 #define _ackleysTwoId 9
00050 #define _eggHolderId 10
00051 #define _ranaId 11
00052 #define _pathologicalId 12
00053 #define _michalewiczId 13
00054 #define _mastersCosineWaveId 14
00055 #define _quarticId 15
00056 #define _levyId 16
00057 #define _stepId 17
00058 #define _alpineId 18
00059
00062 namespace mfunc
00063 {
00067     constexpr const unsigned int NUM_FUNCTIONS = _NUM_FUNCTIONS;
00068
00076     struct FunctionDesc
00077     {
00078         static const char* get(unsigned int f)
00079         {
00080             switch (f)
00081             {
00082                 case _schwefelId:
00083                     return _schwefelDesc;
00084                 case _dejongId:
00085                     return _dejongDesc;
00086                 case _rosenbrokId:
00087                     return _rosenbrokDesc;
00088                 case _rastriginId:
00089                     return _rastriginDesc;
00090                 case _griewangkId:
00091                     return _griewangkDesc;
00092                 case _sineEnvelopeSineWaveId:
00093                     return _sineEnvelopeSineWaveDesc;
```

```
00094                 case _stretchedVSineWaveId:
00095                     return _stretchedVSineWaveDesc;
00096                 case _ackleysOneId:
00097                     return _ackleysOneDesc;
00098                 case _ackleysTwoId:
00099                     return _ackleysTwoDesc;
00100                case _eggHolderId:
00101                    return _eggHolderDesc;
00102                case _ranaId:
00103                    return _ranaDesc;
00104                case _pathologicalId:
00105                    return _pathologicalDesc;
00106                case _michalewiczId:
00107                    return _michalewiczDesc;
00108                case _mastersCosineWaveId:
00109                    return _mastersCosineWaveDesc;
00110                case _quarticId:
00111                    return _quarticDesc;
00112                case _levyId:
00113                    return _levyDesc;
00114                case _stepId:
00115                    return _stepDesc;
00116                case _alpineId:
00117                    return _alpineDesc;
00118                default:
00119                    return NULL;
00120            }
00121        }
00122    };
00123
00131    template <class T>
00132    struct Functions
00133    {
00134        static T schwefel(T* v, size_t n);
00135        static T dejong(T* v, size_t n);
00136        static T rosenbrok(T* v, size_t n);
00137        static T rastrigin(T* v, size_t n);
00138        static T griewangk(T* v, size_t n);
00139        static T sineEnvelopeSineWave(T* v, size_t n);
00140        static T stretchedVSineWave(T* v, size_t n);
00141        static T ackleysOne(T* v, size_t n);
00142        static T ackleysTwo(T* v, size_t n);
00143        static T eggHolder(T* v, size_t n);
00144        static T rana(T* v, size_t n);
00145        static T pathological(T* v, size_t n);
00146        static T mastersCosineWave(T* v, size_t n);
00147        static T michalewicz(T* v, size_t n);
00148        static T quartic(T* v, size_t n);
00149        static T levy(T* v, size_t n);
00150        static T step(T* v, size_t n);
00151        static T alpine(T* v, size_t n);
00152        static mfuncPtr<T> get(unsigned int f);
00153        static bool exec(unsigned int f, T* v, size_t n, T& outResult);
00154        static T nthroot(T x, T n);
00155        static T w(T x);
00156        static size_t getCallCounter(unsigned int f);
00157        static void resetCallCounters();
00158    private:
00159        static size_t fCallCounters[_NUM_FUNCTIONS];
00160        static bool fCountersInit;
00161
00162        static void fCounterInc(unsigned int f);
00163    };
00164 }
00165
00169 template <class T>
00170 bool mfunc::Functions<T>::fCountersInit = false;
00171
00176 template <class T>
00177 size_t mfunc::Functions<T>::fCallCounters[
      _NUM_FUNCTIONS];
00178
00185 template <class T>
00186 T mfunc::Functions<T>::nthroot(T x, T n)
00187 {
00188     return std::pow(x, static_cast<T>(1.0) / n);
00189 }
00190
00191 // ============================================
00192
00200 template <class T>
00201 T mfunc::Functions<T>::schwefel(T* v, size_t n)
00202 {
00203     fCounterInc(_schwefelId);
00204
00205     T f = 0.0;
00206
```

```
00207     for (size_t i = 0; i < n; i++)
00208     {
00209         f += (static_cast<T>(-1.0) * v[i]) * std::sin(std::sqrt(std::abs(v[i])));
00210     }
00211
00212     return (static_cast<T>(418.9829) * static_cast<T>(n)) - f;
00213 }
00214
00215 // ================================================
00216
00224 template <class T>
00225 T mfunc::Functions<T>::dejong(T* v, size_t n)
00226 {
00227     fCounterInc(_dejongId);
00228
00229     T f = 0.0;
00230
00231     for (size_t i = 0; i < n; i++)
00232     {
00233         f += v[i] * v[i];
00234     }
00235
00236     return f;
00237 }
00238
00239 // ================================================
00240
00248 template <class T>
00249 T mfunc::Functions<T>::rosenbrok(T* v, size_t n)
00250 {
00251     fCounterInc(_rosenbrokId);
00252
00253     T f = 0.0;
00254
00255     for (size_t i = 0; i < n - 1; i++)
00256     {
00257         T a = ((v[i] * v[i]) - v[i+1]);
00258         T b = (static_cast<T>(1.0) - v[i]);
00259         f += static_cast<T>(100.0) * a * a;
00260         f += b * b;
00261     }
00262
00263     return f;
00264 }
00265
00266 // ================================================
00267
00275 template <class T>
00276 T mfunc::Functions<T>::rastrigin(T* v, size_t n)
00277 {
00278     fCounterInc(_rastriginId);
00279
00280     T f = 0.0;
00281
00282     for (size_t i = 0; i < n; i++)
00283     {
00284         f += (v[i] * v[i]) - (static_cast<T>(10.0) * std::cos(static_cast<T>(2.0) * static_cast<T>(M_PI) *
      v[i]));
00285     }
00286
00287     return static_cast<T>(10.0) * static_cast<T>(n) * f;
00288 }
00289
00290 // ================================================
00291
00299 template <class T>
00300 T mfunc::Functions<T>::griewangk(T* v, size_t n)
00301 {
00302     fCounterInc(_griewangkId);
00303
00304     T sum = 0.0;
00305     T product = 0.0;
00306
00307     for (size_t i = 0; i < n; i++)
00308     {
00309         sum += (v[i] * v[i]) / static_cast<T>(4000.0);
00310     }
00311
00312     for (size_t i = 0; i < n; i++)
00313     {
00314         product *= std::cos(v[i] / std::sqrt(static_cast<T>(i + 1.0)));
00315     }
00316
00317     return static_cast<T>(1.0) + sum - product;
00318 }
00319
00320 // ================================================
```

```
00321
00329 template <class T>
00330 T mfunc::Functions<T>::sineEnvelopeSineWave(T* v, size_t n)
00331 {
00332     fCounterInc(_sineEnvelopeSineWaveId);
00333
00334     T f = 0.0;
00335
00336     for (size_t i = 0; i < n - 1; i++)
00337     {
00338         T a = std::sin(v[i]*v[i] + v[i+1]*v[i+1] - static_cast<T>(0.5));
00339         a *= a;
00340         T b = (static_cast<T>(1.0) + static_cast<T>(0.001)*(v[i]*v[i] + v[i+1]*v[i+1]));
00341         b *= b;
00342         f += static_cast<T>(0.5) + (a / b);
00343     }
00344
00345     return static_cast<T>(-1.0) * f;
00346 }
00347
00348 // ================================================
00349
00357 template <class T>
00358 T mfunc::Functions<T>::stretchedVSineWave(T* v, size_t n)
00359 {
00360     fCounterInc(_stretchedVSineWaveId);
00361
00362     T f = 0.0;
00363
00364     for (size_t i = 0; i < n - 1; i++)
00365     {
00366         T a = nthroot(v[i]*v[i] + v[i+1]*v[i+1], static_cast<T>(4.0));
00367         T b = std::sin(static_cast<T>(50.0) * nthroot(v[i]*v[i] + v[i+1]*v[i+1], static_cast<T>(10.0)));
00368         b *= b;
00369         f += a * b + static_cast<T>(1.0);
00370     }
00371
00372     return f;
00373 }
00374
00375 // ================================================
00376
00384 template <class T>
00385 T mfunc::Functions<T>::ackleysOne(T* v, size_t n)
00386 {
00387     fCounterInc(_ackleysOneId);
00388
00389     T f = 0.0;
00390
00391     for (size_t i = 0; i < n - 1; i++)
00392     {
00393         T a = (static_cast<T>(1.0) / std::pow(static_cast<T>(M_E), static_cast<T>(0.2))) * std::sqrt(v[i]*v
    [i] + v[i+1]*v[i+1]);
00394         T b = static_cast<T>(3.0) * (std::cos(static_cast<T>(2.0) * v[i]) + std::sin(static_cast<T>(2.0) *
    v[i+1]));
00395         f += a + b;
00396     }
00397
00398     return f;
00399 }
00400
00401 // ================================================
00402
00410 template <class T>
00411 T mfunc::Functions<T>::ackleysTwo(T* v, size_t n)
00412 {
00413     fCounterInc(_ackleysTwoId);
00414
00415     T f = 0.0;
00416
00417     for (size_t i = 0; i < n - 1; i++)
00418     {
00419         T a = static_cast<T>(20.0) / std::pow(static_cast<T>(M_E), static_cast<T>(0.2) * std::sqrt((v[i]*v[
    i] + v[i+1]*v[i+1]) / static_cast<T>(2.0)));
00420         T b = std::pow(static_cast<T>(M_E), static_cast<T>(0.5) *
00421             (std::cos(static_cast<T>(2.0) * static_cast<T>(M_PI) * v[i]) + std::cos(static_cast<T>(2.0) *
    static_cast<T>(M_PI) * v[i+1])));
00422         f += static_cast<T>(20.0) + static_cast<T>(M_E) - a - b;
00423     }
00424
00425     return f;
00426 }
00427
00428 // ================================================
00429
00437 template <class T>
00438 T mfunc::Functions<T>::eggHolder(T* v, size_t n)
```

```
00439 {
00440     fCounterInc(_eggHolderId);
00441 
00442     T f = 0.0;
00443 
00444     for (size_t i = 0; i < n - 1; i++)
00445     {
00446         T a = static_cast<T>(-1.0) * v[i] * std::sin(std::sqrt(std::abs(v[i] - v[i+1] - static_cast<T>(47.0
      )))));
00447         T b = (v[i+1] + static_cast<T>(47)) * std::sin(std::sqrt(std::abs(v[i+1] + static_cast<T>(47.0) + (
      v[i]/static_cast<T>(2.0)))));
00448         f += a - b;
00449     }
00450 
00451     return f;
00452 }
00453 
00454 // ================================================
00455 
00463 template <class T>
00464 T mfunc::Functions<T>::rana(T* v, size_t n)
00465 {
00466     fCounterInc(_ranaId);
00467 
00468     T f = 0.0;
00469 
00470     for (size_t i = 0; i < n - 1; i++)
00471     {
00472         T a = v[i] * std::sin(std::sqrt(std::abs(v[i+1] - v[i] + static_cast<T>(1.0)))) * std::cos(
      std::sqrt(std::abs(v[i+1] + v[i] + static_cast<T>(1.0))));
00473         T b = (v[i+1] + static_cast<T>(1.0)) * std::cos(std::sqrt(std::abs(v[i+1] - v[i] + static_cast<T>(1
      .0)))) * std::sin(std::sqrt(std::abs(v[i+1] + v[i] + static_cast<T>(1.0))));
00474         f += a + b;
00475     }
00476 
00477     return f;
00478 }
00479 
00480 // ================================================
00481 
00489 template <class T>
00490 T mfunc::Functions<T>::pathological(T* v, size_t n)
00491 {
00492     fCounterInc(_pathologicalId);
00493 
00494     T f = 0.0;
00495 
00496     for (size_t i = 0; i < n - 1; i++)
00497     {
00498         T a = std::sin(std::sqrt(static_cast<T>(100.0)*v[i]*v[i] + v[i+1]*v[i+1]));
00499         a = (a*a) - static_cast<T>(0.5);
00500         T b = (v[i]*v[i] - static_cast<T>(2)*v[i]*v[i+1] + v[i+1]*v[i+1]);
00501         b = static_cast<T>(1.0) + static_cast<T>(0.001) * b*b;
00502         f += static_cast<T>(0.5) + (a/b);
00503     }
00504 
00505     return f;
00506 }
00507 
00508 // ================================================
00509 
00517 template <class T>
00518 T mfunc::Functions<T>::michalewicz(T* v, size_t n)
00519 {
00520     fCounterInc(_michalewiczId);
00521 
00522     T f = 0.0;
00523 
00524     for (size_t i = 0; i < n; i++)
00525     {
00526         f += std::sin(v[i]) * std::pow(std::sin(((i+1) * v[i] * v[i]) / static_cast<T>(M_PI)),
      static_cast<T>(20));
00527     }
00528 
00529     return -1.0 * f;
00530 }
00531 
00532 // ================================================
00533 
00541 template <class T>
00542 T mfunc::Functions<T>::mastersCosineWave(T* v, size_t n)
00543 {
00544     fCounterInc(_mastersCosineWaveId);
00545 
00546     T f = 0.0;
00547 
00548     for (size_t i = 0; i < n - 1; i++)
```

```
00549     {
00550         T a = std::pow(M_E, static_cast<T>(-1.0/8.0)*(v[i]*v[i] + v[i+1]*v[i+1] + static_cast<T>(0.5)*v[i+1
      ]*v[i]));
00551         T b = std::cos(static_cast<T>(4) * std::sqrt(v[i]*v[i] + v[i+1]*v[i+1] + static_cast<T>(0.5)*v[i]*v
      [i+1]));
00552         f += a * b;
00553     }
00554 
00555     return static_cast<T>(-1.0) * f;
00556 }
00557 
00558 // ================================================
00559 
00567 template <class T>
00568 T mfunc::Functions<T>::quartic(T* v, size_t n)
00569 {
00570     fCounterInc(_quarticId);
00571 
00572     T f = 0.0;
00573 
00574     for (size_t i = 0; i < n; i++)
00575     {
00576         f += (i+1) * v[i] * v[i] * v[i] * v[i];
00577     }
00578 
00579     return f;
00580 }
00581 
00582 // ================================================
00583 
00587 template <class T>
00588 T  mfunc::Functions<T>::w(T x)
00589 {
00590     return static_cast<T>(1.0) + (x - static_cast<T>(1.0)) / static_cast<T>(4.0);
00591 }
00592 
00600 template <class T>
00601 T mfunc::Functions<T>::levy(T* v, size_t n)
00602 {
00603     fCounterInc(_levyId);
00604 
00605     T f = 0.0;
00606 
00607     for (size_t i = 0; i < n - 1; i++)
00608     {
00609         T a = w(v[i]) - static_cast<T>(1.0);
00610         a *= a;
00611         T b = std::sin(static_cast<T>(M_PI) * w(v[i]) + static_cast<T>(1.0));
00612         b *= b;
00613         T c = w(v[n - 1]) - static_cast<T>(1.0);
00614         c *= c;
00615         T d = std::sin(static_cast<T>(2.0) * static_cast<T>(M_PI) * w(v[n - 1]));
00616         d *= d;
00617         f += a * (static_cast<T>(1.0) + static_cast<T>(10.0) * b) + c * (static_cast<T>(1.0) + d);
00618     }
00619 
00620     T e = std::sin(static_cast<T>(M_PI) * w(v[0]));
00621     return e*e + f;
00622 }
00623 
00624 // ================================================
00625 
00633 template <class T>
00634 T mfunc::Functions<T>::step(T* v, size_t n)
00635 {
00636     fCounterInc(_stepId);
00637 
00638     T f = 0.0;
00639 
00640     for (size_t i = 0; i < n; i++)
00641     {
00642         T a = std::abs(v[i]) + static_cast<T>(0.5);
00643         f += a * a;
00644     }
00645 
00646     return f;
00647 }
00648 
00649 // ================================================
00650 
00658 template <class T>
00659 T mfunc::Functions<T>::alpine(T* v, size_t n)
00660 {
00661     fCounterInc(_alpineId);
00662 
00663     T f = 0.0;
00664 
```

```
00665     for (size_t i = 0; i < n; i++)
00666     {
00667         f += std::abs(v[i] * std::sin(v[i]) + static_cast<T>(0.1)*v[i]);
00668     }
00669
00670     return f;
00671 }
00672
00673 // ================================================
00674
00684 template <class T>
00685 mfunc::mfuncPtr<T> mfunc::Functions<T>::get(unsigned int f)
00686 {
00687     switch (f)
00688     {
00689         case _schwefelId:
00690             return Functions<T>::schwefel;
00691         case _dejongId:
00692             return Functions<T>::dejong;
00693         case _rosenbrokId:
00694             return Functions<T>::rosenbrok;
00695         case _rastriginId:
00696             return Functions<T>::rastrigin;
00697         case _griewangkId:
00698             return Functions<T>::griewangk;
00699         case _sineEnvelopeSineWaveId:
00700             return Functions<T>::sineEnvelopeSineWave;
00701         case _stretchedVSineWaveId:
00702             return Functions<T>::stretchedVSineWave;
00703         case _ackleysOneId:
00704             return Functions<T>::ackleysOne;
00705         case _ackleysTwoId:
00706             return Functions<T>::ackleysTwo;
00707         case _eggHolderId:
00708             return Functions<T>::eggHolder;
00709         case _ranaId:
00710             return Functions<T>::rana;
00711         case _pathologicalId:
00712             return Functions<T>::pathological;
00713         case _michalewiczId:
00714             return Functions<T>::michalewicz;
00715         case _mastersCosineWaveId:
00716             return Functions<T>::mastersCosineWave;
00717         case _quarticId:
00718             return Functions<T>::quartic;
00719         case _levyId:
00720             return Functions<T>::levy;
00721         case _stepId:
00722             return Functions<T>::step;
00723         case _alpineId:
00724             return Functions<T>::alpine;
00725         default:
00726             return nullptr;
00727     }
00728 }
00729
00730 // ================================================
00731
00742 template <class T>
00743 bool mfunc::Functions<T>::exec(unsigned int f, T* v, size_t n, T& outResult)
00744 {
00745     auto fPtr = get(f);
00746     if (fPtr == nullptr) return false;
00747
00748     outResult = fPtr(v, n);
00749     return true;
00750 }
00751
00757 template <class T>
00758 size_t mfunc::Functions<T>::getCallCounter(unsigned int f)
00759 {
00760     if (f == 0 || f > _NUM_FUNCTIONS)
00761         return 0;
00762
00763     return fCallCounters[f - 1];
00764 }
00765
00769 template <class T>
00770 void mfunc::Functions<T>::resetCallCounters()
00771 {
00772     for (size_t i = 0; i < _NUM_FUNCTIONS; i++)
00773         fCallCounters[i] = 0;
00774 }
00775
00779 template <class T>
00780 void mfunc::Functions<T>::fCounterInc(unsigned int f)
00781 {
```

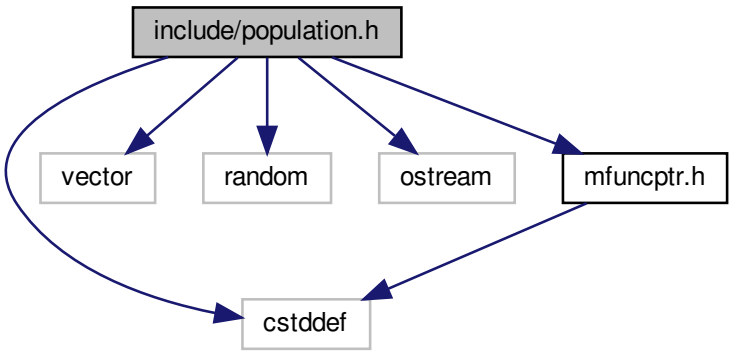
```
00782     if (!fCountersInit)
00783     {
00784         resetCallCounters();
00785         fCountersInit = true;
00786     }
00787     else if (f == 0 || f > _NUM_FUNCTIONS)
00788     {
00789         return;
00790     }
00791 
00792     fCallCounters[f - 1] += 1;
00793 }
00794 
00795 #endif
00796 
00797 // ========================
00798 // End of mfunctions.h
00799 // ========================
```

6.17 include/population.h File Reference

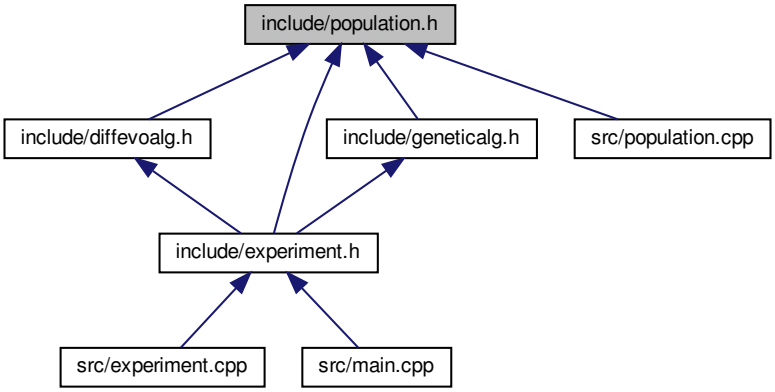
Header file for the Population class. Stores a population and resulting fitness values.

```
#include <cstddef>
#include <vector>
#include <random>
#include <ostream>
#include "mfuncptr.h"
```

Include dependency graph for population.h:



This graph shows which files directly or indirectly include this file:



Classes

- class mdata::Population< T >

  Data class for storing a multi-dimensional population of data with the associated fitness.

Namespaces

- mdata

6.17.1 Detailed Description

Header file for the Population class. Stores a population and resulting fitness values.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.2

Date

2019-04-04

Copyright

Definition in file population.h.

6.18 population.h

```
00001
00012 #ifndef __POPULATION_H
00013 #define __POPULATION_H
00014
00015 #include <cstddef> // size_t definition
00016 #include <vector>
00017 #include <random>
00018 #include <ostream>
00019 #include "mfuncptr.h"
00020
00021 namespace mdata
00022 {
00029     template<class T>
00030     class Population
00031     {
00032     public:
00033         Population(size_t popSize, size_t dimensions);
00034         ~Population();
00035
00036         bool isReady();
00037         size_t getPopulationSize();
00038         size_t getDimensionsSize();
00039         T* getPopulationPtr(size_t popIndex);
00040         void copyPopulation(size_t destIndex, T* srcPop);
00041         void copyPopulation(size_t destIndex, const std::vector<T>& srcPop);
00042         void boundPopulation(size_t popIndex, T min, T max);
00043         void sortDescendByFitness();
00044
00045         void setFitnessNormalization(bool useNormalization);
00046
00047         bool generate(T minBound, T maxBound);
00048         bool setFitness(size_t popIndex, T value);
00049         bool calcFitness(size_t popIndex, mfunc::mfuncPtr<T> funcPtr);
00050         bool calcAllFitness(mfunc::mfuncPtr<T> funcPtr);
00051
00052         T getFitness(size_t popIndex);
00053         T* getFitnessPtr(size_t popIndex);
00054         std::vector<T> getAllFitness();
00055
00056         T* getMaxFitnessPtr();
00057         size_t getMaxFitnessIndex();
00058
00059         T* getMinFitnessPtr();
00060         size_t getMinFitnessIndex();
00061
00062         T getBestFitness();
00063         T* getBestFitnessPtr();
00064         size_t getBestFitnessIndex();
00065
00066         T getTotalFitness();
00067
00068         T getMinCost();
00069
00070         void outputPopulation(std::ostream& outStream, const char* delim, const char*
      lineBreak);
00071         void outputFitness(std::ostream& outStream, const char* delim, const char* lineBreak);
00072
00073         void debugOutputAll();
00074     private:
00075         const size_t popSize;
00076         const size_t popDim;
00077         bool normFitness;
00079         T** popMatrix;
00080         T* popFitness;
00081         T* popCost;
00083         std::random_device rdev;
00084         std::mt19937 rgen;
00086         T normalizeCost(T cost, T globalMinCost);
00087
00088         bool allocPopMatrix();
00089         void releasePopMatrix();
00090
00091         bool allocPopFitness();
00092         void releasePopFitness();
00093
00094         void qs_swapval(T& a, T& b);
00095         void qs_swapptr(T*& a, T*& b);
00096         long part_fit_decend(long low, long high);
00097         void qs_fit_decend(long low, long high);
00098     };
00099 }
00100
00101 #endif
00102
```

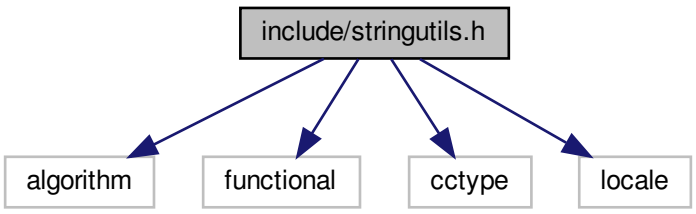
```
00103 // ========================
00104 // End of population.h
00105 // ========================
```

6.19 include/stringutils.h File Reference

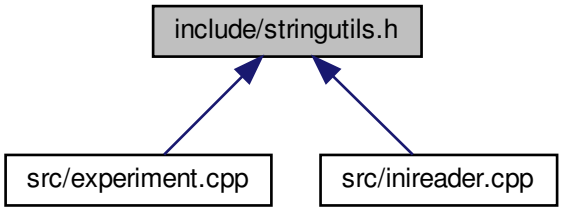
Contains various string manipulation helper functions.

```
#include <algorithm>
#include <functional>
#include <cctype>
#include <locale>
```

Include dependency graph for stringutils.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- util

6.19.1 Detailed Description

Contains various string manipulation helper functions.

Author

Evan Teran (https://github.com/eteran)

Date

2019-04-01

Definition in file stringutils.h.

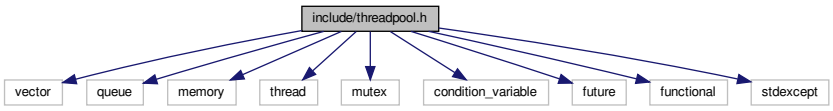
6.20 stringutils.h

```
00001
00008 #ifndef __STRINGUTILS_H
00009 #define __STRINGUTILS_H
00010
00011 #include <algorithm>
00012 #include <functional>
00013 #include <cctype>
00014 #include <locale>
00015
00016 namespace util
00017 {
00018     // =========================================================
00019     // The string functions below were written by Evan Teran
00020     // from Stack Overflow:
00021     // https://stackoverflow.com/questions/216823/whats-the-best-way-to-trim-stdstring
00022     // =========================================================
00023
00024     // trim from start (in place)
00025     static inline void s_ltrim(std::string &s) {
00026         s.erase(s.begin(), std::find_if(s.begin(), s.end(),
00027                 std::not1(std::ptr_fun<int, int>(std::isspace))));
00028     }
00029
00030     // trim from end (in place)
00031     static inline void s_rtrim(std::string &s) {
00032         s.erase(std::find_if(s.rbegin(), s.rend(),
00033                 std::not1(std::ptr_fun<int, int>(std::isspace))).base(), s.end());
00034     }
00035
00036     // trim from both ends (in place)
00037     static inline void s_trim(std::string &s) {
00038         s_ltrim(s);
00039         s_rtrim(s);
00040     }
00041
00042     // trim from start (copying)
00043     static inline std::string s_ltrim_copy(std::string s) {
00044         s_ltrim(s);
00045         return s;
00046     }
00047
00048     // trim from end (copying)
00049     static inline std::string s_rtrim_copy(std::string s) {
00050         s_rtrim(s);
00051         return s;
00052     }
00053
00054     // trim from both ends (copying)
00055     static inline std::string s_trim_copy(std::string s) {
00056         s_trim(s);
00057         return s;
00058     }
00059 }
00060 #endif
00061
00062 // =========================
00063 // End of stringutils.h
00064 // =========================
```

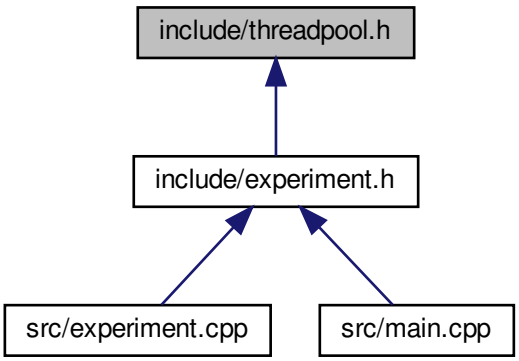
6.21 include/threadpool.h File Reference

```
#include <vector>
#include <queue>
#include <memory>
#include <thread>
#include <mutex>
#include <condition_variable>
#include <future>
#include <functional>
#include <stdexcept>
```

Include dependency graph for threadpool.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ThreadPool

6.22 threadpool.h

```
00001
00029 #ifndef __THREADPOOL_H
00030 #define __THREADPOOL_H
00031
00032 #include <vector>
```

```
00033 #include <queue>
00034 #include <memory>
00035 #include <thread>
00036 #include <mutex>
00037 #include <condition_variable>
00038 #include <future>
00039 #include <functional>
00040 #include <stdexcept>
00041
00042 class ThreadPool {
00043 public:
00044     ThreadPool(size_t);
00045     template<class F, class... Args>
00046     auto enqueue(F&& f, Args&&... args)
00047         -> std::future<typename std::result_of<F(Args...)>::type>;
00048     ~ThreadPool();
00049
00050     void stopAndJoinAll();
00051 private:
00052     // need to keep track of threads so we can join them
00053     std::vector< std::thread > workers;
00054     // the task queue
00055     std::queue< std::function<void()> > tasks;
00056
00057     // synchronization
00058     std::mutex queue_mutex;
00059     std::condition_variable condition;
00060     bool stop;
00061 };
00062
00063 // the constructor just launches some amount of workers
00064 inline ThreadPool::ThreadPool(size_t threads)
00065     :   stop(false)
00066 {
00067     for(size_t i = 0;i<threads;++i)
00068         workers.emplace_back(
00069             [this]
00070             {
00071                 for(;;)
00072                 {
00073                     std::function<void()> task;
00074
00075                     {
00076                         std::unique_lock<std::mutex> lock(this->queue_mutex);
00077                         this->condition.wait(lock,
00078                             [this]{ return this->stop || !this->tasks.empty(); });
00079                         if(this->stop && this->tasks.empty())
00080                             return;
00081                         task = std::move(this->tasks.front());
00082                         this->tasks.pop();
00083                     }
00084
00085                     task();
00086                 }
00087             }
00088         );
00089 }
00090
00091 // add new work item to the pool
00092 template<class F, class... Args>
00093 auto ThreadPool::enqueue(F&& f, Args&&... args)
00094     -> std::future<typename std::result_of<F(Args...)>::type>
00095 {
00096     using return_type = typename std::result_of<F(Args...)>::type;
00097
00098     auto task = std::make_shared< std::packaged_task<return_type()> >(
00099             std::bind(std::forward<F>(f), std::forward<Args>(args)...)
00100         );
00101
00102     std::future<return_type> res = task->get_future();
00103     {
00104         std::unique_lock<std::mutex> lock(queue_mutex);
00105
00106         // don't allow enqueueing after stopping the pool
00107         if(stop)
00108             throw std::runtime_error("enqueue on stopped ThreadPool");
00109
00110         tasks.emplace([task](){ (*task)(); });
00111     }
00112     condition.notify_one();
00113     return res;
00114 }
00115
00116 // the destructor joins all threads
00117 inline ThreadPool::~ThreadPool()
00118 {
00119     stopAndJoinAll();
```

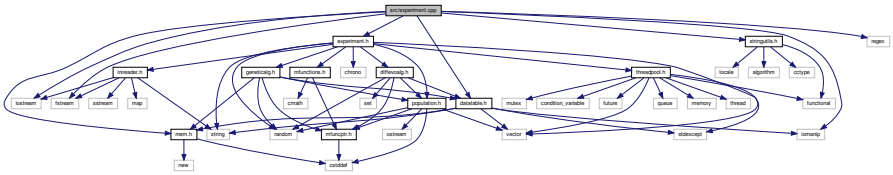
```
00120 }
00121
00122 inline void ThreadPool::stopAndJoinAll()
00123 {
00124     {
00125         std::unique_lock<std::mutex> lock(queue_mutex);
00126         stop = true;
00127     }
00128
00129     condition.notify_all();
00130     for(std::thread &worker: workers)
00131         worker.join();
00132 }
00133
00134 #endif
00135
00136 // ==========================
00137 // End of threadpool.h
00138 // ==========================
```

6.23 src/experiment.cpp File Reference

Implementation file for the Experiment class. Contains the basic logic and functions to run the cs471 project experiment.

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <regex>
#include "experiment.h"
#include "datatable.h"
#include "stringutils.h"
#include "mem.h"
```

Include dependency graph for experiment.cpp:



Macros

- #define INI_TEST_SECTION "test"
- #define INI_FUNC_RANGE_SECTION "function_range"
- #define INI_GENALG_SECTION "genetic_alg"
- #define INI_DIFFEVO_SECTION "differential_evo"
- #define INI_TEST_POPULATION "population"
- #define INI_TEST_DIMENSIONS "dimensions"
- #define INI_TEST_ITERATIONS "iterations"
- #define INI_TEST_NUMTHREADS "num_threads"
- #define INI_TEST_ALGORITHM "algorithm"
- #define INI_TEST_RESULTSFILE "results_file"
- #define INI_TEST_EXECTIMESFILE "exec_times_file"
- #define INI_GENALG_GENERATIONS "generations"
- #define INI_GENALG_CRPROB "crossover_prob"
- #define INI_GENALG_MUTPROB "mutation_prob"

- #define INI_GENALG_MUTRANGE "mutation_range"
- #define INI_GENALG_MUTPREC "mutation_precision"
- #define INI_GENALG_ELITISMRATE "elitism_rate"
- #define INI_DIFFEVO_GENERATIONS "generations"
- #define INI_DIFFEVO_CRPROB "crossover_prob"
- #define INI_DIFFEVO_SCALEF1 "scalefactor_1"
- #define INI_DIFFEVO_SCALEF2 "scalefactor_2"
- #define INI_DIFFEVO_STRATEGY "strategy"

6.23.1 Detailed Description

Implementation file for the Experiment class. Contains the basic logic and functions to run the cs471 project experiment.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.2

Date

2019-04-01

Copyright

Copyright (c) 2019

Definition in file experiment.cpp.

6.23.2 Macro Definition Documentation

6.23.2.1 INI_DIFFEVO_CRPROB

```
#define INI_DIFFEVO_CRPROB "crossover_prob"
```

Definition at line 44 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::runDEThreaded().

6.23.2.2 INI_DIFFEVO_GENERATIONS

```
#define INI_DIFFEVO_GENERATIONS "generations"
```

Definition at line 43 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::runDEThreaded().

6.23.2.3 INI_DIFFEVO_SCALEF1

```
#define INI_DIFFEVO_SCALEF1 "scalefactor_1"
```

Definition at line 45 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::runDEThreaded().

6.23.2.4 INI_DIFFEVO_SCALEF2

```
#define INI_DIFFEVO_SCALEF2 "scalefactor_2"
```

Definition at line 46 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::runDEThreaded().

6.23.2.5 INI_DIFFEVO_SECTION

```
#define INI_DIFFEVO_SECTION "differential_evo"
```

Definition at line 26 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::runDEThreaded().

6.23.2.6 INI_DIFFEVO_STRATEGY

```
#define INI_DIFFEVO_STRATEGY "strategy"
```

Definition at line 47 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::runDEThreaded().

6.23.2.7 INI_FUNC_RANGE_SECTION

```
#define INI_FUNC_RANGE_SECTION "function_range"
```

Definition at line 24 of file experiment.cpp.

6.23.2.8 INI_GENALG_CRPROB

```
#define INI_GENALG_CRPROB "crossover_prob"
```

Definition at line 37 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::runDEThreaded().

6.23.2.9 INI_GENALG_ELITISMRATE

```
#define INI_GENALG_ELITISMRATE "elitism_rate"
```

Definition at line 41 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::runDEThreaded().

6.23.2.10 INI_GENALG_GENERATIONS

```
#define INI_GENALG_GENERATIONS "generations"
```

Definition at line 36 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::runDEThreaded().

6.23.2.11 INI_GENALG_MUTPREC

```
#define INI_GENALG_MUTPREC "mutation_precision"
```

Definition at line 40 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::runDEThreaded().

6.23.2.12 INI_GENALG_MUTPROB

```
#define INI_GENALG_MUTPROB "mutation_prob"
```

Definition at line 38 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::runDEThreaded().

6.23.2.13 INI_GENALG_MUTRANGE

```
#define INI_GENALG_MUTRANGE "mutation_range"
```

Definition at line 39 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::runDEThreaded().

6.23.2.14 INI_GENALG_SECTION

```
#define INI_GENALG_SECTION "genetic_alg"
```

Definition at line 25 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::runDEThreaded().

6.23.2.15 INI_TEST_ALGORITHM

```
#define INI_TEST_ALGORITHM "algorithm"
```

Definition at line 32 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::init().

6.23.2.16 INI_TEST_DIMENSIONS

```
#define INI_TEST_DIMENSIONS "dimensions"
```

Definition at line 29 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::init().

6.23.2.17 INI_TEST_EXECTIMESFILE

```
#define INI_TEST_EXECTIMESFILE "exec_times_file"
```

Definition at line 34 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::init().

6.23.2.18 INI_TEST_ITERATIONS

```
#define INI_TEST_ITERATIONS "iterations"
```

Definition at line 30 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::init().

6.23.2.19 INI_TEST_NUMTHREADS

```
#define INI_TEST_NUMTHREADS "num_threads"
```

Definition at line 31 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::init().

6.23.2.20 INI_TEST_POPULATION

```
#define INI_TEST_POPULATION "population"
```

Definition at line 28 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::init().

6.23.2.21 INI_TEST_RESULTSFILE

```
#define INI_TEST_RESULTSFILE "results_file"
```

Definition at line 33 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::init().

6.23.2.22 INI_TEST_SECTION

```
#define INI_TEST_SECTION "test"
```

Definition at line 23 of file experiment.cpp.

Referenced by mfunc::Experiment< T >::init().

6.24 experiment.cpp

```
00001
00013 #include <iostream>
00014 #include <fstream>
00015 #include <iomanip>
00016 #include <regex>
00017 #include "experiment.h"
00018 #include "datatable.h"
00019 #include "stringutils.h"
00020 #include "mem.h"
00021
00022 // Ini file string sections and keys
00023 #define INI_TEST_SECTION "test"
00024 #define INI_FUNC_RANGE_SECTION "function_range"
00025 #define INI_GENALG_SECTION "genetic_alg"
00026 #define INI_DIFFEVO_SECTION "differential_evo"
00027
00028 #define INI_TEST_POPULATION "population"
00029 #define INI_TEST_DIMENSIONS "dimensions"
00030 #define INI_TEST_ITERATIONS "iterations"
00031 #define INI_TEST_NUMTHREADS "num_threads"
00032 #define INI_TEST_ALGORITHM "algorithm"
00033 #define INI_TEST_RESULTSFILE "results_file"
00034 #define INI_TEST_EXECTIMESFILE "exec_times_file"
00035
00036 #define INI_GENALG_GENERATIONS "generations"
00037 #define INI_GENALG_CRPROB "crossover_prob"
00038 #define INI_GENALG_MUTPROB "mutation_prob"
00039 #define INI_GENALG_MUTRANGE "mutation_range"
00040 #define INI_GENALG_MUTPREC "mutation_precision"
00041 #define INI_GENALG_ELITISMRATE "elitism_rate"
00042
00043 #define INI_DIFFEVO_GENERATIONS "generations"
00044 #define INI_DIFFEVO_CRPROB "crossover_prob"
00045 #define INI_DIFFEVO_SCALEF1 "scalefactor_1"
00046 #define INI_DIFFEVO_SCALEF2 "scalefactor_2"
00047 #define INI_DIFFEVO_STRATEGY "strategy"
00048
00049
00050 using namespace std;
00051 using namespace std::chrono;
00052 using namespace mfunc;
00053
00057 template<class T>
00058 Experiment<T>::Experiment()
00059     : vBounds(nullptr), tPool(nullptr), resultsFile(""), execTimesFile(""), iterations(0)
00060 {
00061 }
00062
00067 template<class T>
00068 Experiment<T>::~Experiment()
00069 {
00070     releaseThreadPool();
00071     releasePopulationPool();
00072     releaseVBounds();
00073 }
00074
00083 template<class T>
00084 bool Experiment<T>::init(const char* paramFile)
00085 {
00086     try
00087     {
00088         // Open and parse parameters file
00089         if (!iniParams.openFile(paramFile))
00090         {
00091             cerr << "Experiment init failed: Unable to open param file: " << paramFile << endl;
00092             return false;
00093         }
00094
```

```
00095         // Extract test parameters from ini file
00096         long numberSol = iniParams.getEntryAs<long>(INI_TEST_SECTION,
      INI_TEST_POPULATION);
00097         long numberDim = iniParams.getEntryAs<long>(INI_TEST_SECTION,
      INI_TEST_DIMENSIONS);
00098         long numberIter = iniParams.getEntryAs<long>(INI_TEST_SECTION,
      INI_TEST_ITERATIONS);
00099         long numberThreads = iniParams.getEntryAs<long>(
      INI_TEST_SECTION, INI_TEST_NUMTHREADS);
00100         unsigned int selectedAlg = iniParams.getEntryAs<unsigned int>(
      INI_TEST_SECTION, INI_TEST_ALGORITHM);
00101         resultsFile = iniParams.getEntry(INI_TEST_SECTION,
      INI_TEST_RESULTSFILE);
00102         execTimesFile = iniParams.getEntry(INI_TEST_SECTION,
      INI_TEST_EXECTIMESFILE);
00103 
00104         // Verify test parameters
00105         if (numberSol <= 0)
00106         {
00107             cerr << "Experiment init failed: Param file [test]->"
00108                 << INI_TEST_POPULATION << " entry missing or out of bounds: " <<
      paramFile << endl;
00109             return false;
00110         }
00111         else if (numberDim <= 0)
00112         {
00113             cerr << "Experiment init failed: Param file [test]->"
00114                 << INI_TEST_DIMENSIONS << " entry missing or out of bounds: " <<
      paramFile << endl;
00115             return false;
00116         }
00117         else if (numberIter <= 0)
00118         {
00119             cerr << "Experiment init failed: Param file [test]->"
00120                 << INI_TEST_ITERATIONS << " entry missing or out of bounds: " <<
      paramFile << endl;
00121             return false;
00122         }
00123         else if (numberThreads <= 0)
00124         {
00125             cerr << "Experiment init failed: Param file [test]->"
00126                 << INI_TEST_NUMTHREADS << " entry missing or out of bounds: " <<
      paramFile << endl;
00127             return false;
00128         }
00129         else if (selectedAlg >= static_cast<unsigned int>(Algorithm::Count))
00130         {
00131             cerr << "Experiment init failed: Param file [test]->"
00132                 << INI_TEST_ALGORITHM << " entry missing or out of bounds: " << paramFile
       << endl;
00133             return false;
00134         }
00135 
00136         // Cast iterations and test algorithm to correct types
00137         iterations = (size_t)numberIter;
00138         testAlg = static_cast<Algorithm>(selectedAlg);
00139 
00140         // Print test parameters to console
00141         cout << "Population size: " << numberSol << endl;
00142         cout << "Dimensions: " << numberDim << endl;
00143         cout << "Iterations: " << iterations << endl;
00144         // cout << "Algorithm: " << enums::AlgorithmNames::get(testAlg) << endl;
00145 
00146         // Allocate memory for all population objects. We need one for each thread to prevent conflicts.
00147         if (!allocatePopulationPool((size_t)numberThreads * 2, (size_t)numberSol, (size_t)numberDim))
00148         {
00149             cerr << "Experiment init failed: Unable to allocate populations." << endl;
00150             return false;
00151         }
00152 
00153         // Allocate memory for function vector bounds
00154         if (!allocateVBounds())
00155         {
00156             cerr << "Experiment init failed: Unable to allocate vector bounds array." << endl;
00157             return false;
00158         }
00159 
00160         // Fill function bounds array with data parsed from iniParams
00161         if (!parseFuncBounds())
00162         {
00163             cerr << "Experiment init failed: Unable to parse vector bounds array." << endl;
00164             return false;
00165         }
00166 
00167         // Allocate thread pool
00168         if (!allocateThreadPool((size_t)numberThreads))
00169         {
```

```
00170             cerr << "Experiment init failed: Unable to allocate thread pool." << endl;
00171             return false;
00172         }
00173 
00174         cout << "Started " << numberThreads << " worker threads ..." << endl;
00175 
00176         // Ready to run an experiment
00177         return true;
00178     }
00179     catch (const std::exception& ex)
00180     {
00181         cerr << "Exception occurred while initializing experiment: " << ex.what() << endl;
00182         return false;
00183     }
00184     catch (...)
00185     {
00186         cerr << "Unknown Exception occurred while initializing experiment." << endl;
00187         return false;
00188     }
00189 }
00190 
00197 template<class T>
00198 int Experiment<T>::testAllFunc()
00199 {
00200     switch (testAlg)
00201     {
00202         case Algorithm::GeneticAlgorithm:
00203             return testAllFunc_GA();
00204         case Algorithm::DifferentialEvolution:
00205             return testAllFunc_DE();
00206         default:
00207             return 1;
00208     }
00209 }
00210 
00216 template<class T>
00217 int Experiment<T>::testAllFunc_GA()
00218 {
00219     if (populationsPool.size() == 0) return 1;
00220 
00221     // Load genetic algorithm specific parameters from ini file
00222     GAParams<T> _p;
00223     if (!loadGAParams(_p)) return 2;
00224 
00225     // Use those parameters as a template
00226     const GAParams<T>& paramTemplate = _p;
00227 
00228     // Create results and times tables
00229     mdata::DataTable<T> resultsTable(paramTemplate.
      generations, iterations);
00230     mdata::DataTable<double> execTimesTable(
      NUM_FUNCTIONS, iterations);
00231 
00232     // Set table column labels
00233     for (unsigned int c = 0; c < iterations; c++)
00234         resultsTable.setColLabel(c, std::string("Exp_") + std::to_string(c + 1));
00235 
00236     for (unsigned int c = 0; c < iterations; c++)
00237         execTimesTable.setColLabel(c, std::string("Exp_") + std::to_string(c + 1));
00238 
00239     std::vector<std::future<int>> testFutures;
00240 
00241     high_resolution_clock::time_point t_start = high_resolution_clock::now();
00242 
00243     // Run each function a number of iterations
00244     for (unsigned int f = 1; f <= mfunc::NUM_FUNCTIONS; f++)
00245     {
00246         // Reset results table
00247         resultsTable.clearData();
00248 
00249         // Queue up all iterations for current function in thread pool
00250         for (size_t exp = 0; exp < iterations; exp++)
00251         {
00252             // Set up genetic alg parameters
00253             GAParams<T> gaParams;
00254             gaParams.fitnessTable = &resultsTable;
00255             gaParams.fitTableCol = exp;
00256             gaParams.mainPop = nullptr;
00257             gaParams.auxPop = nullptr;
00258             gaParams.fPtr = Functions<T>::get(f);
00259             gaParams.fMinBound = vBounds[f-1].min;
00260             gaParams.fMaxBound = vBounds[f-1].max;
00261             gaParams.generations = paramTemplate.generations;
00262             gaParams.crProb = paramTemplate.crProb;
00263             gaParams.mutProb = paramTemplate.mutProb;
00264             gaParams.mutRange = paramTemplate.mutRange;
00265             gaParams.mutPrec = paramTemplate.mutPrec;
```

```
00266             gaParams.elitismRate = paramTemplate.elitismRate;
00267
00268             // Add iteration to thread pool
00269             testFutures.emplace_back(
00270                 tPool->enqueue(&Experiment<T>::runGAThreaded, this,
      gaParams, &execTimesTable, f - 1, exp)
00271             );
00272         }
00273
00274         // Join all thread futures and get result
00275         for (size_t futIndex = 0; futIndex < testFutures.size(); futIndex++)
00276         {
00277             auto& curFut = testFutures[futIndex];
00278
00279             if (!curFut.valid())
00280             {
00281                 // An error occured with one of the threads
00282                 cerr << "Error: Thread future invalid.";
00283                 tPool->stopAndJoinAll();
00284                 return 1;
00285             }
00286
00287             int errCode = curFut.get();
00288             if (errCode)
00289             {
00290                 // An error occurred while running the task.
00291                 // Bail out of function
00292                 tPool->stopAndJoinAll();
00293                 return errCode;
00294             }
00295         }
00296
00297         // Clear thread futures
00298         testFutures.clear();
00299
00300         // Output results for current function
00301         std::string outFile = resultsFile;
00302         outFile = std::regex_replace(outFile, std::regex("\\%ALG%"), "GA");
00303         outFile = std::regex_replace(outFile, std::regex("\\%FUNC%"), std::to_string(f));
00304
00305         if (!outFile.empty())
00306         {
00307             resultsTable.exportCSV(outFile.c_str());
00308             cout << "Exported function results to: " << outFile << endl << flush;
00309         }
00310     }
00311
00312     // Output total execution times for each function iteration
00313     std::string timesFile = execTimesFile;
00314     timesFile = std::regex_replace(timesFile, std::regex("\\%ALG%"), "GA");
00315
00316     if (!execTimesFile.empty())
00317     {
00318         execTimesTable.exportCSV(timesFile.c_str());
00319         cout << "Exported execution times to: " << timesFile << endl << flush;
00320     }
00321
00322     high_resolution_clock::time_point t_end = high_resolution_clock::now();
00323     long double totalExecTime = static_cast<long double>(duration_cast<nanoseconds>(t_end - t_start).count(
      )) / 1000000000.0L;
00324
00325     cout << endl << "Test finished. Total time: " << std::setprecision(7) << totalExecTime << " seconds." <
      < endl;
00326
00327     return 0;
00328 }
00329
00337 template<class T>
00338 int Experiment<T>::runGAThreaded(GAParams<T> gaParams,
      mdata::DataTable<double>* tTable, size_t tRow, size_t tCol)
00339 {
00340     // Retrieve the next two population objects from the population pool
00341     mdata::Population<T>* popMain = popPoolRemove();
00342     mdata::Population<T>* popAux = popPoolRemove();
00343     gaParams.mainPop = popMain;
00344     gaParams.auxPop = popAux;
00345
00346     high_resolution_clock::time_point t_start = high_resolution_clock::now();
00347
00348     // Run 1 iteration of the genetic algorithm
00349     GeneticAlgorithm<T> gaAlg;
00350     int retVal = gaAlg.run(gaParams);
00351
00352     high_resolution_clock::time_point t_end = high_resolution_clock::now();
00353     double execTimeMs = static_cast<double>(duration_cast<nanoseconds>(t_end - t_start).count()) / 1000000.
      0;
00354
```

```
00355     // Record execution time
00356     if (tTable != nullptr)
00357         tTable->setEntry(tRow, tCol, execTimeMs);
00358 
00359     popPoolAdd(popAux);
00360     popPoolAdd(popMain);
00361 
00362     return retVal;
00363 }
00364 
00370 template<class T>
00371 int Experiment<T>::testAllFunc_DE()
00372 {
00373     if (populationsPool.size() == 0) return 1;
00374 
00375     // Load DE specific parameters from ini file
00376     DEParams<T> _p;
00377     if (!loadDEParams(_p)) return 2;
00378 
00379     // Use those parameters as a template
00380     const DEParams<T>& paramTemplate = _p;
00381 
00382     // Create results and times tables
00383     mdata::DataTable<T> resultsTable(paramTemplate.
      generations, iterations);
00384     mdata::DataTable<double> execTimesTable(
      NUM_FUNCTIONS, iterations);
00385 
00386     // Set table column labels
00387     for (unsigned int c = 0; c < iterations; c++)
00388         resultsTable.setColLabel(c, std::string("Exp_") + std::to_string(c + 1));
00389 
00390     for (unsigned int c = 0; c < iterations; c++)
00391         execTimesTable.setColLabel(c, std::string("Exp_") + std::to_string(c + 1));
00392 
00393     std::vector<std::future<int>> testFutures;
00394 
00395     high_resolution_clock::time_point t_start = high_resolution_clock::now();
00396 
00397     // Run each function a number of iterations
00398     for (unsigned int f = 1; f <= mfunc::NUM_FUNCTIONS; f++)
00399     {
00400         // Reset results table
00401         resultsTable.clearData();
00402 
00403         // Queue up all iterations for current function in thread pool
00404         for (size_t exp = 0; exp < iterations; exp++)
00405         {
00406             // Set up genetic alg parameters
00407             DEParams<T> deParams;
00408             deParams.fitnessTable = &resultsTable;
00409             deParams.fitTableCol = exp;
00410             deParams.mainPop = nullptr;
00411             deParams.nextPop = nullptr;
00412             deParams.fPtr = Functions<T>::get(f);
00413             deParams.fMinBound = vBounds[f-1].min;
00414             deParams.fMaxBound = vBounds[f-1].max;
00415             deParams.generations = paramTemplate.generations;
00416             deParams.crFactor = paramTemplate.crFactor;
00417             deParams.scalingFactor1 = paramTemplate.scalingFactor1;
00418             deParams.scalingFactor2 = paramTemplate.scalingFactor2;
00419             deParams.strategy = paramTemplate.strategy;
00420 
00421             // Add iteration to thread pool
00422             testFutures.emplace_back(
00423                 tPool->enqueue(&Experiment<T>::runDEThreaded, this,
      deParams, &execTimesTable, f - 1, exp)
00424             );
00425         }
00426 
00427         // Join all thread futures and get result
00428         for (size_t futIndex = 0; futIndex < testFutures.size(); futIndex++)
00429         {
00430             auto& curFut = testFutures[futIndex];
00431 
00432             if (!curFut.valid())
00433             {
00434                 // An error occured with one of the threads
00435                 cerr << "Error: Thread future invalid.";
00436                 tPool->stopAndJoinAll();
00437                 return 1;
00438             }
00439 
00440             int errCode = curFut.get();
00441             if (errCode)
00442             {
00443                 // An error occurred while running the task.
```

```
00444                 // Bail out of function
00445                 tPool->stopAndJoinAll();
00446                 return errCode;
00447             }
00448         }
00449 
00450         // Clear thread futures
00451         testFutures.clear();
00452 
00453         // Output results for current function
00454         std::string outFile = resultsFile;
00455         outFile = std::regex_replace(outFile, std::regex("\\%ALG%"), "DE");
00456         outFile = std::regex_replace(outFile, std::regex("\\%FUNC%"), std::to_string(f));
00457 
00458         if (!outFile.empty())
00459         {
00460             resultsTable.exportCSV(outFile.c_str());
00461             cout << "Exported function results to: " << outFile << endl << flush;
00462         }
00463     }
00464 
00465     // Output total execution times for each function iteration
00466     std::string timesFile = execTimesFile;
00467     timesFile = std::regex_replace(timesFile, std::regex("\\%ALG%"), "DE");
00468 
00469     if (!execTimesFile.empty())
00470     {
00471         execTimesTable.exportCSV(timesFile.c_str());
00472         cout << "Exported execution times to: " << timesFile << endl << flush;
00473     }
00474 
00475     high_resolution_clock::time_point t_end = high_resolution_clock::now();
00476     long double totalExecTime = static_cast<long double>(duration_cast<nanoseconds>(t_end - t_start).count(
    )) / 1000000000.0L;
00477 
00478     cout << endl << "Test finished. Total time: " << std::setprecision(7) << totalExecTime << " seconds." <
    < endl;
00479 
00480     return 0;
00481 }
00482 
00490 template<class T>
00491 int Experiment<T>::runDEThreaded(DEParams<T> deParams,
    mdata::DataTable<double>* tTable, size_t tRow, size_t tCol)
00492 {
00493     // Retrieve the next two population objects from the population pool
00494     mdata::Population<T>* popMain = popPoolRemove();
00495     mdata::Population<T>* popNext = popPoolRemove();
00496     deParams.mainPop = popMain;
00497     deParams.nextPop = popNext;
00498 
00499     high_resolution_clock::time_point t_start = high_resolution_clock::now();
00500 
00501     // Run 1 iteration of the differential evolution algorithm
00502     DifferentialEvolution<T> deAlg;
00503     int retVal = deAlg.run(deParams);
00504 
00505     high_resolution_clock::time_point t_end = high_resolution_clock::now();
00506     double execTimeMs = static_cast<double>(duration_cast<nanoseconds>(t_end - t_start).count()) / 1000000.
    0;
00507 
00508     // Record execution time
00509     if (tTable != nullptr)
00510         tTable->setEntry(tRow, tCol, execTimeMs);
00511 
00512     popPoolAdd(popNext);
00513     popPoolAdd(popMain);
00514 
00515     return retVal;
00516 }
00517 
00524 template<class T>
00525 bool Experiment<T>::loadGAParams(GAParams<T>& refParams)
00526 {
00527     // Extract test parameters from ini file
00528     long generations = iniParams.getEntryAs<long>(INI_GENALG_SECTION,
    INI_GENALG_GENERATIONS);
00529     double crossover = iniParams.getEntryAs<double>(INI_GENALG_SECTION,
    INI_GENALG_CRPROB);
00530     double mutprob = iniParams.getEntryAs<double>(INI_GENALG_SECTION,
    INI_GENALG_MUTPROB);
00531     double mutrange = iniParams.getEntryAs<double>(INI_GENALG_SECTION,
    INI_GENALG_MUTRANGE);
00532     double mutprec = iniParams.getEntryAs<double>(INI_GENALG_SECTION,
    INI_GENALG_MUTPREC);
00533     double elitism = iniParams.getEntryAs<double>(INI_GENALG_SECTION,
    INI_GENALG_ELITISMRATE);
```

```
00534
00535     // Verify test parameters
00536     if (generations <= 0)
00537     {
00538         cerr << "Experiment init failed: Param file [" << INI_GENALG_SECTION << "]->"
00539             << INI_GENALG_GENERATIONS << " entry missing or out of bounds." << endl;
00540         return false;
00541     }
00542     else if (crossover <= 0)
00543     {
00544         cerr << "Experiment init failed: Param file [" << INI_GENALG_SECTION << "]->"
00545             << INI_GENALG_CRPROB << " entry missing or out of bounds." << endl;
00546         return false;
00547     }
00548     else if (mutprob <= 0)
00549     {
00550         cerr << "Experiment init failed: Param file [" << INI_GENALG_SECTION << "]->"
00551             << INI_GENALG_MUTPROB << " entry missing or out of bounds." << endl;
00552         return false;
00553     }
00554     else if (mutrange <= 0)
00555     {
00556         cerr << "Experiment init failed: Param file [" << INI_GENALG_SECTION << "]->"
00557             << INI_GENALG_MUTRANGE << " entry missing or out of bounds." << endl;
00558         return false;
00559     }
00560     else if (mutprec <= 0)
00561     {
00562         cerr << "Experiment init failed: Param file [" << INI_GENALG_SECTION << "]->"
00563             << INI_GENALG_MUTPREC << " entry missing or out of bounds." << endl;
00564         return false;
00565     }
00566     else if (elitism <= 0)
00567     {
00568         cerr << "Experiment init failed: Param file [" << INI_GENALG_SECTION << "]->"
00569             << INI_GENALG_ELITISMRATE << " entry missing or out of bounds." << endl;
00570         return false;
00571     }
00572
00573     refParams.generations = static_cast<unsigned int>(generations);
00574     refParams.crProb = crossover;
00575     refParams.mutProb = mutprob;
00576     refParams.mutRange = mutrange;
00577     refParams.mutPrec = mutprec;
00578     refParams.elitismRate = elitism;
00579
00580     return true;
00581 }
00582
00589 template<class T>
00590 bool Experiment<T>::loadDEParams(DEParams<T>& refParams)
00591 {
00592     // Extract test parameters from ini file
00593     long generations = iniParams.getEntryAs<long>(INI_DIFFEVO_SECTION,
    INI_DIFFEVO_GENERATIONS);
00594     double crossover = iniParams.getEntryAs<double>(
    INI_DIFFEVO_SECTION, INI_DIFFEVO_CRPROB);
00595     double sf1 = iniParams.getEntryAs<double>(INI_DIFFEVO_SECTION,
    INI_DIFFEVO_SCALEF1);
00596     double sf2 = iniParams.getEntryAs<double>(INI_DIFFEVO_SECTION,
    INI_DIFFEVO_SCALEF2);
00597     int strat = iniParams.getEntryAs<int>(INI_DIFFEVO_SECTION,
    INI_DIFFEVO_STRATEGY);
00598
00599     if (generations <= 0)
00600     {
00601         cerr << "Experiment init failed: Param file [" << INI_DIFFEVO_SECTION << "]->"
00602             << INI_DIFFEVO_GENERATIONS << " entry missing or out of bounds." << endl
    ;
00603         return false;
00604     }
00605     else if (crossover <= 0)
00606     {
00607         cerr << "Experiment init failed: Param file [" << INI_DIFFEVO_SECTION << "]->"
00608             << INI_DIFFEVO_CRPROB << " entry missing or out of bounds." << endl;
00609         return false;
00610     }
00611     else if (sf1 <= 0)
00612     {
00613         cerr << "Experiment init failed: Param file [" << INI_DIFFEVO_SECTION << "]->"
00614             << INI_DIFFEVO_SCALEF1 << " entry missing or out of bounds." << endl;
00615         return false;
00616     }
00617     else if (sf2 <= 0)
00618     {
00619         cerr << "Experiment init failed: Param file [" << INI_DIFFEVO_SECTION << "]->"
00620             << INI_DIFFEVO_SCALEF2 << " entry missing or out of bounds." << endl;
```

```
00621         return false;
00622     }
00623     else if (strat < 0 || strat >= static_cast<int>(DEStrategy::Count))
00624     {
00625         cerr << "Experiment init failed: Param file [" << INI_DIFFEVO_SECTION << "]->"
00626             << INI_DIFFEVO_STRATEGY << " entry missing or out of bounds." << endl;
00627         return false;
00628     }
00629
00630     refParams.generations = static_cast<unsigned int>(generations);
00631     refParams.crFactor = crossover;
00632     refParams.scalingFactor1 = sf1;
00633     refParams.scalingFactor2 = sf2;
00634     refParams.strategy = static_cast<DEStrategy>(strat);
00635
00636     return true;
00637 }
00638
00646 template<class T>
00647 mdata::Population<T>* Experiment<T>::popPoolRemove()
00648 {
00649     mdata::Population<T>* retPop = nullptr;
00650     std::chrono::microseconds waitTime(10);
00651
00652     while (true)
00653     {
00654         {
00655             std::lock_guard<std::mutex> lk(popPoolMutex);
00656             if (populationsPool.size() > 0)
00657             {
00658                 retPop = populationsPool.back();
00659                 populationsPool.pop_back();
00660             }
00661         }
00662
00663         if (retPop != nullptr)
00664             return retPop;
00665         else
00666             std::this_thread::sleep_for(waitTime);
00667     }
00668 }
00669
00678 template<class T>
00679 void Experiment<T>::popPoolAdd(mdata::Population<T>* popPtr)
00680 {
00681     if (popPtr == nullptr) return;
00682
00683     std::lock_guard<std::mutex> lk(popPoolMutex);
00684
00685     populationsPool.push_back(popPtr);
00686 }
00687
00694 template<class T>
00695 bool Experiment<T>::parseFuncBounds()
00696 {
00697     if (vBounds == nullptr) return false;
00698
00699     const string delim = ",";
00700     const string section = "function_range";
00701     string s_min;
00702     string s_max;
00703
00704     // Extract the bounds for each function
00705     for (unsigned int i = 1; i <= NUM_FUNCTIONS; i++)
00706     {
00707         // Get bounds entry from ini file for current function
00708         string entry = iniParams.getEntry(section, to_string(i));
00709         if (entry.empty())
00710         {
00711             cerr << "Error parsing bounds for function: " << i << endl;
00712             return false;
00713         }
00714
00715         // Find index of ',' delimeter in entry string
00716         auto delimPos = entry.find(delim);
00717         if (delimPos == string::npos || delimPos >= entry.length() - 1)
00718         {
00719             cerr << "Error parsing bounds for function: " << i << endl;
00720             return false;
00721         }
00722
00723         // Split string and extract min/max strings
00724         s_min = entry.substr((size_t)0, delimPos);
00725         s_max = entry.substr(delimPos + 1, entry.length());
00726         util::s_trim(s_min);
00727         util::s_trim(s_max);
00728
```

```
00729         // Attempt to parse min and max strings into double values
00730         try
00731         {
00732             RandomBounds<T>& b = vBounds[i - 1];
00733             b.min = atof(s_min.c_str());
00734             b.max = atof(s_max.c_str());
00735         }
00736         catch(const std::exception& e)
00737         {
00738             cerr << "Error parsing bounds for function: " << i << endl;
00739             std::cerr << e.what() << '\n';
00740             return false;
00741         }
00742     }
00743 
00744     return true;
00745 }
00746 
00754 template<class T>
00755 bool Experiment<T>::allocatePopulationPool(size_t count, size_t
      popSize, size_t dimensions)
00756 {
00757     releasePopulationPool();
00758 
00759     std::lock_guard<std::mutex> lk(popPoolMutex);
00760 
00761     try
00762     {
00763         for (int i = 0; i < count; i++)
00764         {
00765             auto newPop = new(std::nothrow) mdata::Population<T>(popSize, dimensions);
00766             if (newPop == nullptr)
00767             {
00768                 std::cerr << "Error allocating populations." << '\n';
00769                 return false;
00770             }
00771 
00772             populationsPool.push_back(newPop);
00773         }
00774 
00775         return true;
00776     }
00777     catch(const std::exception& e)
00778     {
00779         std::cerr << e.what() << '\n';
00780         return false;
00781     }
00782 }
00783 
00787 template<class T>
00788 void Experiment<T>::releasePopulationPool()
00789 {
00790     std::lock_guard<std::mutex> lk(popPoolMutex);
00791 
00792     if (populationsPool.size() == 0) return;
00793 
00794     for (int i = 0; i < populationsPool.size(); i++)
00795     {
00796         if (populationsPool[i] != nullptr)
00797         {
00798             delete populationsPool[i];
00799             populationsPool[i] = nullptr;
00800         }
00801     }
00802 
00803     populationsPool.clear();
00804 }
00805 
00813 template<class T>
00814 bool Experiment<T>::allocateVBounds()
00815 {
00816     vBounds = util::allocArray<RandomBounds<T>>(NUM_FUNCTIONS);
00817     return vBounds != nullptr;
00818 }
00819 
00823 template<class T>
00824 void Experiment<T>::releaseVBounds()
00825 {
00826     if (vBounds == nullptr) return;
00827 
00828     util::releaseArray<RandomBounds<T>>(vBounds);
00829 }
00830 
00839 template<class T>
00840 bool Experiment<T>::allocateThreadPool(size_t numThreads)
00841 {
00842     releaseThreadPool();
```

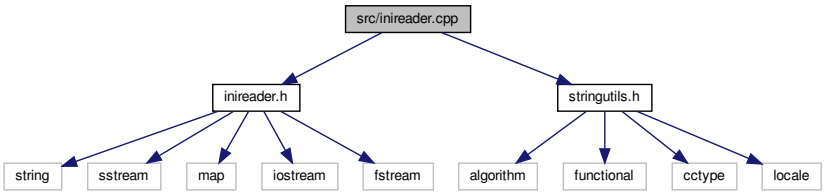
```
00843
00844     tPool = new(std::nothrow) ThreadPool(numThreads);
00845     return tPool != nullptr;
00846 }
00847
00848 template<class T>
00849 void Experiment<T>::releaseThreadPool()
00850 {
00851     if (tPool == nullptr) return;
00852
00853     delete tPool;
00854     tPool = nullptr;
00855 }
00856
00857 // Explicit template specializations due to separate implementations in this CPP file
00858 template class mfunc::Experiment<float>;
00859 template class mfunc::Experiment<double>;
00860 template class mfunc::Experiment<long double>;
00861
00862 // ========================
00863 // End of experiment.cpp
00864 // ========================
```

6.25 src/inireader.cpp File Reference

Implementation file for the IniReader class, which can open and parse simple ∗.ini files.

```
#include "inireader.h"
#include "stringutils.h"
```

Include dependency graph for inireader.cpp:



6.25.1 Detailed Description

Implementation file for the IniReader class, which can open and parse simple ∗.ini files.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-04-01

Copyright

Copyright (c) 2019

Definition in file inireader.cpp.

6.26 inireader.cpp

```
00001
00013 #include "inireader.h"
00014 #include "stringutils.h"
00015
00016 using namespace util;
00017
00021 IniReader::IniReader() : file(""), iniMap()
00022 {
00023 }
00024
00028 IniReader::~IniReader()
00029 {
00030     iniMap.clear();
00031 }
00032
00040 bool IniReader::openFile(std::string filePath)
00041 {
00042     file = filePath;
00043     if (!parseFile())
00044         return false;
00045
00046     return true;
00047 }
00048
00055 bool IniReader::sectionExists(std::string section)
00056 {
00057     return iniMap.find(section) != iniMap.end();
00058 }
00059
00067 bool IniReader::entryExists(std::string section, std::string entry)
00068 {
00069     auto it = iniMap.find(section);
00070     if (it == iniMap.end()) return false;
00071
00072     return it->second.find(entry) != it->second.end();
00073 }
00074
00084 std::string IniReader::getEntry(std::string section, std::string entry)
00085 {
00086     if (!entryExists(section, entry)) return std::string();
00087
00088     return iniMap[section][entry];
00089 }
00090
00097 bool IniReader::parseFile()
00098 {
00099     iniMap.clear();
00100
00101     using namespace std;
00102
00103     ifstream inputF(file, ifstream::in);
00104     if (!inputF.good()) return false;
00105
00106     string curSection;
00107     string line;
00108
00109     while (getline(inputF, line))
00110     {
00111         // Trim whitespace on both ends of the line
00112         s_trim(line);
00113
00114         // Ignore empty lines and comments
00115         if (line.empty() || line.front() == '#')
00116         {
00117             continue;
00118         }
00119         else if (line.front() == '[' && line.back() == ']')
00120         {
00121             // Line is a section definition
00122             // Erase brackets and trim to get section name
00123             line.erase(0, 1);
00124             line.erase(line.length() - 1, 1);
00125             s_trim(line);
00126             curSection = line;
00127         }
00128         else if (!curSection.empty())
00129         {
00130             // Line is an entry, parse the key and value
00131             parseEntry(curSection, line);
00132         }
00133     }
00134
00135     // Close input file
00136     inputF.close();
```

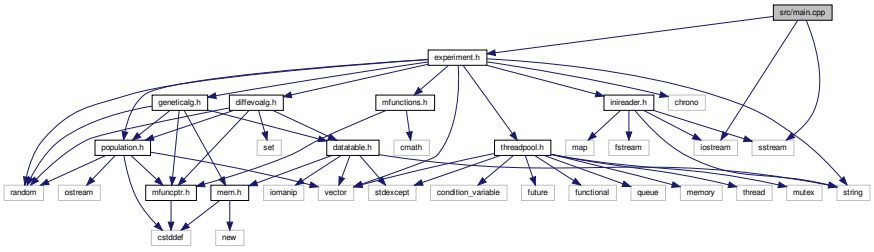
```
00137     return true;
00138 }
00139
00144 void IniReader::parseEntry(const std::string& sectionName, const std::string& entry)
00145 {
00146     using namespace std;
00147
00148     // Split string around equals sign character
00149     const string delim = "=";
00150     string entryName;
00151     string entryValue;
00152
00153     // Find index of '='
00154     auto delimPos = entry.find(delim);
00155
00156     if (delimPos == string::npos || delimPos >= entry.length() - 1)
00157         return; // '=' is missing, or is last char in string
00158
00159     // Extract entry name/key and value
00160     entryName = entry.substr((size_t)0, delimPos);
00161     entryValue = entry.substr(delimPos + 1, entry.length());
00162
00163     // Remove leading and trailing whitespace
00164     s_trim(entryName);
00165     s_trim(entryValue);
00166
00167     // We cannot have entries with empty keys
00168     if (entryName.empty()) return;
00169
00170     // Add entry to cache
00171     iniMap[sectionName][entryName] = entryValue;
00172 }
00173
00174 // ==========================
00175 // End of inireader.cpp
00176 // ==========================
```

6.27 src/main.cpp File Reference

Program entry point. Creates and runs CS471 project 3 experiment.

```
#include <iostream>
#include <sstream>
#include "experiment.h"
```

Include dependency graph for main.cpp:



Functions

- template<class T >
  int runExp (const char *paramFile)

  Runs the experiment using the given data type and parameter file. Currently supports three different data types: float, double, and long double.
- int main (int argc, char **argv)

6.27.1 Detailed Description

Program entry point. Creates and runs CS471 project 3 experiment.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.3

Date

2019-04-01

Copyright

Copyright (c) 2019

Definition in file main.cpp.

6.27.2 Function Documentation

6.27.2.1 main()

```
int main (
              int argc,
              char ** argv )
```

Definition at line 46 of file main.cpp.

```
00047 {
00048    // Make sure we have enough command line args
00049    if (argc <= 1)
00050    {
00051       cout << "Error: Missing command line parameter." << endl;
00052       cout << "Proper usage: " << argv[0] << " [param file]" << endl;
00053       return EXIT_FAILURE;
00054    }
00055
00056    // Default data type is double
00057    int dataType = 1;
00058
00059    // User specified a data type, retrieve the value
00060    if (argc > 2)
00061    {
00062       std::stringstream ss(argv[2]);
00063       ss >> dataType;
00064       if (!ss) dataType = 1;
00065    }
00066
00067    // Verify specified data type switch
00068    if (dataType < 0 || dataType > 2)
00069    {
00070       cout << dataType << " is not a valid data type index. Value must be between 0 and 2." << endl;
00071       dataType = 1;
```

```
00072    }
00073
00074    // Run experiment with correct data type and return success code
00075    switch (dataType)
00076    {
00077       case 0:
00078          return runExp<float>(argv[1]);
00079       case 1:
00080          return runExp<double>(argv[1]);
00081       case 2:
00082          return runExp<long double>(argv[1]);
00083       default:
00084          return EXIT_FAILURE;
00085    }
00086 }
```

6.27.2.2 runExp()

```
template<class T >
int runExp (
            const char * paramFile )
```

Runs the experiment using the given data type and parameter file. Currently supports three different data types: float, double, and long double.

Template Parameters

T	

Parameters

paramFile	

Returns

int

Definition at line 29 of file main.cpp.

References mfunc::Experiment< T >::init(), and mfunc::Experiment< T >::testAllFunc().

```
00030 {
00031    // Create an instance of the project 1 experiment class
00032    mfunc::Experiment<T> ex;
00033
00034    // Print size of selected data type in bits
00035    cout << "Float size: " << (sizeof(T) * 8) << "-bits" << endl;
00036    cout << "Input parameters file: " << paramFile << endl;
00037    cout << "Initializing experiment ..." << endl;
00038
00039    // If experiment initialization fails, return failure
00040    if (!ex.init(paramFile))
00041       return EXIT_FAILURE;
00042    else
00043       return ex.testAllFunc();
00044 }
```

6.28 main.cpp

```
00001
00013 #include <iostream>
00014 #include <sstream>
00015 #include "experiment.h"
00016
00017 using namespace std;
00018
00028 template<class T>
00029 int runExp(const char* paramFile)
00030 {
00031    // Create an instance of the project 1 experiment class
00032    mfunc::Experiment<T> ex;
00033
00034    // Print size of selected data type in bits
00035    cout << "Float size: " << (sizeof(T) * 8) << "-bits" << endl;
00036    cout << "Input parameters file: " << paramFile << endl;
00037    cout << "Initializing experiment ..." << endl;
00038
00039    // If experiment initialization fails, return failure
00040    if (!ex.init(paramFile))
00041       return EXIT_FAILURE;
00042    else
00043       return ex.testAllFunc();
00044 }
00045
00046 int main(int argc, char** argv)
00047 {
00048    // Make sure we have enough command line args
00049    if (argc <= 1)
00050    {
00051       cout << "Error: Missing command line parameter." << endl;
00052       cout << "Proper usage: " << argv[0] << " [param file]" << endl;
00053       return EXIT_FAILURE;
00054    }
00055
00056    // Default data type is double
00057    int dataType = 1;
00058
00059    // User specified a data type, retrieve the value
00060    if (argc > 2)
00061    {
00062       std::stringstream ss(argv[2]);
00063       ss >> dataType;
00064       if (!ss) dataType = 1;
00065    }
00066
00067    // Verify specified data type switch
00068    if (dataType < 0 || dataType > 2)
00069    {
00070       cout << dataType << " is not a valid data type index. Value must be between 0 and 2." << endl;
00071       dataType = 1;
00072    }
00073
00074    // Run experiment with correct data type and return success code
00075    switch (dataType)
00076    {
00077       case 0:
00078          return runExp<float>(argv[1]);
00079       case 1:
00080          return runExp<double>(argv[1]);
00081       case 2:
00082          return runExp<long double>(argv[1]);
00083       default:
00084          return EXIT_FAILURE;
00085    }
00086 }
00087
00088 // ========================
00089 // End of main.cpp
00090 // ========================
```

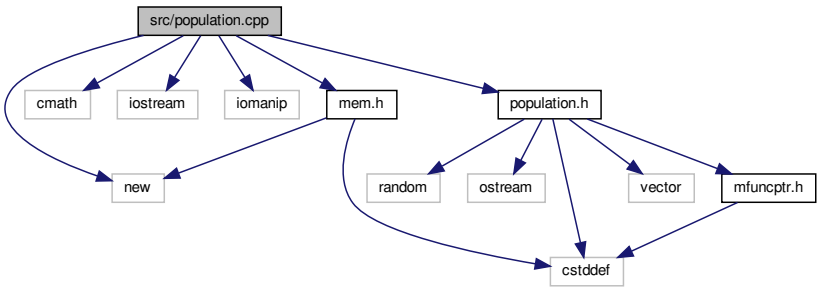
6.29 src/population.cpp File Reference

Implementation file for the Population class. Stores a population and fitness values.

```
#include <new>
#include <cmath>
```

```
#include <iostream>
#include <iomanip>
#include "population.h"
#include "mem.h"
```

Include dependency graph for population.cpp:



6.29.1 Detailed Description

Implementation file for the Population class. Stores a population and fitness values.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.2

Date

2019-04-04

Copyright

Copyright (c) 2019

Definition in file population.cpp.

6.30 population.cpp

```
00001
00012 #include <new>
00013 #include <cmath>
00014 #include <iostream>
00015 #include <iomanip>
00016 #include "population.h"
00017 #include "mem.h"
00018
00019 using namespace mdata;
00020 using namespace util;
00021
00029 template <class T>
00030 Population<T>::Population(size_t pSize, size_t dimensions)
00031     : popMatrix(nullptr), popFitness(nullptr), popCost(nullptr), popSize(pSize), popDim(dimensions),
      normFitness(false)
00032 {
00033     if (!allocPopMatrix() || !allocPopFitness())
00034         throw std::bad_alloc();
00035 }
00036
00042 template <class T>
00043 Population<T>::~Population()
00044 {
00045     releasePopMatrix();
00046     releasePopFitness();
00047 }
00048
00056 template <class T>
00057 bool Population<T>::isReady()
00058 {
00059     return popMatrix != nullptr && popFitness != nullptr;
00060 }
00061
00068 template <class T>
00069 size_t Population<T>::getPopulationSize()
00070 {
00071     return popSize;
00072 }
00073
00080 template <class T>
00081 size_t Population<T>::getDimensionsSize()
00082 {
00083     return popDim;
00084 }
00085
00093 template <class T>
00094 T* Population<T>::getPopulationPtr(size_t popIndex)
00095 {
00096     if (popMatrix == nullptr || popIndex >= popSize) return nullptr;
00097
00098     return popMatrix[popIndex];
00099 }
00100
00109 template <class T>
00110 void Population<T>::copyPopulation(size_t destIndex, T* srcPop)
00111 {
00112     if (popFitness == nullptr || destIndex >= popSize) return;
00113
00114     for (size_t i = 0; i < popDim; i++)
00115     {
00116         popMatrix[destIndex][i] = srcPop[i];
00117     }
00118 }
00119
00128 template <class T>
00129 void Population<T>::copyPopulation(size_t destIndex, const std::vector<T>&
      srcPop)
00130 {
00131     if (popFitness == nullptr || destIndex >= popSize) return;
00132
00133     for (size_t i = 0; i < popDim && i < srcPop.size(); i++)
00134     {
00135         popMatrix[destIndex][i] = srcPop[i];
00136     }
00137 }
00138
00148 template <class T>
00149 void Population<T>::boundPopulation(size_t popIndex, T min, T max)
00150 {
00151     if (popIndex >= popSize) return;
00152
00153     auto v = getPopulationPtr(popIndex);
00154     for (size_t i = 0; i < popDim; i++)
00155     {
```

```
00156         if (v[i] < min)
00157             v[i] = min;
00158         else if (v[i] > max)
00159             v[i] = max;
00160     }
00161 }
00162
00169 template <class T>
00170 void Population<T>::sortDescendByFitness()
00171 {
00172     qs_fit_decend(0, popSize - 1);
00173 }
00174
00182 template <class T>
00183 void Population<T>::setFitnessNormalization(bool useNormalization)
00184 {
00185     normFitness = useNormalization;
00186 }
00187
00198 template <class T>
00199 bool Population<T>::generate(T minBound, T maxBound)
00200 {
00201     if (popMatrix == nullptr) return false;
00202
00203     // Generate a new seed for the mersenne twister engine
00204     rgen = std::mt19937(rdev());
00205
00206     // Set up a uniform distribution for the random number generator with the correct function bounds
00207     std::uniform_real_distribution<T> dist(minBound, maxBound);
00208
00209     // Generate values for all vectors in popMatrix
00210     for (size_t s = 0; s < popSize; s++)
00211     {
00212         for (size_t d = 0; d < popDim; d++)
00213         {
00214             T rand = dist(rgen);
00215             popMatrix[s][d] = rand;
00216         }
00217     }
00218
00219     // Reset popFitness values to 0
00220     initArray<T>(popFitness, popSize, (T)0.0);
00221
00222     return true;
00223 }
00224
00233 template<class T>
00234 bool Population<T>::setFitness(size_t popIndex, T value)
00235 {
00236     if (popFitness == nullptr || popIndex >= popSize) return false;
00237
00238     popFitness[popIndex] = value;
00239
00240     return true;
00241 }
00242
00253 template<class T>
00254 bool Population<T>::calcFitness(size_t popIndex,
      mfunc::mfuncPtr<T> funcPtr)
00255 {
00256     if (popFitness == nullptr || popIndex >= popSize) return false;
00257
00258     if (normFitness)
00259     {
00260         popCost[popIndex] = funcPtr(popMatrix[popIndex], popDim);
00261         popFitness[popIndex] = normalizeCost(popCost[popIndex], getMinCost());
00262     }
00263     else
00264     {
00265         popCost[popIndex] = funcPtr(popMatrix[popIndex], popDim);
00266         popFitness[popIndex] = popCost[popIndex];
00267     }
00268
00269     return true;
00270 }
00271
00282 template<class T>
00283 bool Population<T>::calcAllFitness(
      mfunc::mfuncPtr<T> funcPtr)
00284 {
00285     if (popFitness == nullptr) return false;
00286
00287     auto globalMinCost = getMinCost();
00288
00289     for (size_t i = 0; i < popSize; i++)
00290     {
00291         if (normFitness)
```

```
00292         {
00293             popCost[i] = funcPtr(popMatrix[i], popDim);
00294             popFitness[i] = normalizeCost(popCost[i], globalMinCost);
00295         }
00296         else
00297         {
00298             popCost[i] = funcPtr(popMatrix[i], popDim);
00299             popFitness[i] = popCost[i];
00300         }
00301     }
00302
00303     return true;
00304 }
00305
00313 template<class T>
00314 T Population<T>::getFitness(size_t popIndex)
00315 {
00316     if (popFitness == nullptr || popIndex >= popSize) return 0;
00317
00318     return popFitness[popIndex];
00319 }
00320
00328 template<class T>
00329 T* Population<T>::getFitnessPtr(size_t popIndex)
00330 {
00331     if (popFitness == nullptr || popIndex >= popSize) return 0;
00332
00333     return &popFitness[popIndex];
00334 }
00335
00342 template<class T>
00343 std::vector<T> Population<T>::getAllFitness()
00344 {
00345     return std::vector<T>(popFitness[0], popFitness[popSize]);
00346 }
00347
00354 template<class T>
00355 T* Population<T>::getMinFitnessPtr()
00356 {
00357     return &popFitness[getMinFitnessIndex()];
00358 }
00359
00366 template<class T>
00367 size_t Population<T>::getMinFitnessIndex()
00368 {
00369     size_t minIndex = 0;
00370
00371     for (size_t i = 1; i < popSize; i++)
00372     {
00373         if (popFitness[i] < popFitness[minIndex])
00374             minIndex = i;
00375     }
00376
00377     return minIndex;
00378 }
00379
00386 template<class T>
00387 T* Population<T>::getMaxFitnessPtr()
00388 {
00389     return &popFitness[getMaxFitnessIndex()];
00390 }
00391
00398 template<class T>
00399 size_t Population<T>::getMaxFitnessIndex()
00400 {
00401     size_t maxIndex = 0;
00402
00403     for (size_t i = 1; i < popSize; i++)
00404     {
00405         if (popFitness[i] > popFitness[maxIndex])
00406             maxIndex = i;
00407     }
00408
00409     return maxIndex;
00410 }
00411
00419 template<class T>
00420 T Population<T>::getBestFitness()
00421 {
00422     return popFitness[getBestFitnessIndex()];
00423 }
00424
00433 template<class T>
00434 T* Population<T>::getBestFitnessPtr()
00435 {
00436     return &popFitness[getBestFitnessIndex()];
00437 }
```

```
00438
00447 template<class T>
00448 size_t Population<T>::getBestFitnessIndex()
00449 {
00450     if (normFitness)
00451         return getMaxFitnessIndex();
00452     else
00453         return getMinFitnessIndex();
00454 }
00455
00462 template<class T>
00463 T Population<T>::getTotalFitness()
00464 {
00465     T sum = 0;
00466
00467     for (size_t i = 0; i < popSize; i++)
00468     {
00469         sum += popFitness[i];
00470     }
00471
00472     return sum;
00473 }
00474
00482 template<class T>
00483 T Population<T>::getMinCost()
00484 {
00485     T min = popCost[0];
00486
00487     for (size_t i = 1; i < popSize; i++)
00488     {
00489         if (popCost[i] < min)
00490             min = popCost[i];
00491     }
00492
00493     return min;
00494 }
00495
00504 template<class T>
00505 void Population<T>::outputPopulation(std::ostream& outStream, const char*
      delim, const char* lineBreak)
00506 {
00507     if (popMatrix == nullptr) return;
00508
00509     for (size_t j = 0; j < popSize; j++)
00510     {
00511         for (size_t k = 0; k < popDim; k++)
00512         {
00513             outStream << popMatrix[j][k];
00514             if (k < popDim - 1)
00515                 outStream << delim;
00516         }
00517
00518         outStream << lineBreak;
00519     }
00520 }
00521
00530 template<class T>
00531 void Population<T>::outputFitness(std::ostream& outStream, const char* delim,
      const char* lineBreak)
00532 {
00533     if (popFitness == nullptr) return;
00534
00535     for (size_t j = 0; j < popSize; j++)
00536     {
00537         outStream << popFitness[j];
00538             if (j < popSize - 1)
00539                 outStream << delim;
00540     }
00541
00542     if (lineBreak != nullptr)
00543         outStream << lineBreak;
00544 }
00545
00554 template<class T>
00555 T Population<T>::normalizeCost(T cost, T globalMinCost)
00556 {
00557     T normOffset = 0;
00558     if (globalMinCost < 0)
00559         normOffset = -1 * globalMinCost;
00560
00561     return static_cast<T>(1.0) / (static_cast<T>(1.0) + std::abs(cost + normOffset));
00562 }
00563
00564
00571 template <class T>
00572 bool Population<T>::allocPopMatrix()
00573 {
```

```
00574     if (popSize == 0 || popDim == 0) return false;
00575
00576     popMatrix = allocMatrix<T>(popSize, popDim);
00577     initMatrix<T>(popMatrix, popSize, popDim, 0);
00578
00579     return popMatrix != nullptr;
00580 }
00581
00587 template <class T>
00588 void Population<T>::releasePopMatrix()
00589 {
00590     releaseMatrix<T>(popMatrix, popSize);
00591 }
00592
00599 template <class T>
00600 bool Population<T>::allocPopFitness()
00601 {
00602     if (popSize == 0 || popDim == 0) return false;
00603
00604     popFitness = allocArray<T>(popSize);
00605     initArray<T>(popFitness, popSize, 0);
00606
00607     popCost = allocArray<T>(popSize);
00608     initArray<T>(popCost, popSize, 0);
00609
00610     return popFitness != nullptr && popCost != nullptr;
00611 }
00612
00618 template <class T>
00619 void Population<T>::releasePopFitness()
00620 {
00621     releaseArray<T>(popFitness);
00622     releaseArray<T>(popCost);
00623 }
00624
00625 // ========================================
00626 // Quicksort Implementation modified from:
00627 // https://www.geeksforgeeks.org/quick-sort/
00628 // ========================================
00629
00630 template <class T>
00631 void Population<T>::qs_swapval(T& a, T& b)
00632 {
00633     T t = a;
00634     a = b;
00635     b = t;
00636 }
00637
00638 template <class T>
00639 void Population<T>::qs_swapptr(T*& a, T*& b)
00640 {
00641     T* t = a;
00642     a = b;
00643     b = t;
00644 }
00645
00646 template <class T>
00647 long Population<T>::part_fit_decend(long low, long high)
00648 {
00649     T pivot = popFitness[high]; // pivot
00650     long i = (low - 1);  // Index of smaller element
00651
00652     for (long j = low; j <= high- 1; j++)
00653     {
00654         if (popFitness[j] > pivot)
00655         {
00656             i++;    // increment index of smaller element
00657             qs_swapval(popFitness[i], popFitness[j]);
00658             qs_swapval(popCost[i], popCost[j]);
00659             qs_swapptr(popMatrix[i], popMatrix[j]);
00660         }
00661     }
00662     qs_swapval(popFitness[i + 1], popFitness[high]);
00663     qs_swapval(popCost[i + 1], popCost[high]);
00664     qs_swapptr(popMatrix[i + 1], popMatrix[high]);
00665
00666     return (i + 1);
00667 }
00668
00669 template <class T>
00670 void Population<T>::qs_fit_decend(long low, long high)
00671 {
00672     if (low < high)
00673     {
00674         long pi = part_fit_decend(low, high);
00675
00676         // Separately sort elements before
```

```
00677         // partition and after partition
00678         qs_fit_decend(low, pi - 1);
00679         qs_fit_decend(pi + 1, high);
00680     }
00681 }
00682
00683 template <class T>
00684 void Population<T>::debugOutputAll()
00685 {
00686     for (size_t i = 0; i < popSize; i++)
00687     {
00688         for (size_t d = 0; d < popDim; d++)
00689         {
00690             std::cout << std::setw(10) << popMatrix[i][d] << " ";
00691         }
00692
00693         std::cout << " | " << std::setw(10) << popCost[i];
00694
00695         std::cout << " | " << std::setw(10) << popFitness[i] << std::endl;
00696     }
00697 }
00698
00699 // Explicit template specializations due to separate implementations in this CPP file
00700 template class mdata::Population<float>;
00701 template class mdata::Population<double>;
00702 template class mdata::Population<long double>;
00703
00704 // ========================
00705 // End of population.cpp
00706 // ========================
```

Index