CS471 Project 1

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.

Functions

- double average (const std::vector< double > &v)

  Calculates the average for a vector of doubles.
- double standardDeviation (const std::vector< double > &v)

  Calculates the standard deviation for a vector of doubles.
- double range (const std::vector< double > &v)

  Calculates the range for a vector of doubles.
- double median (const std::vector< double > &v)

  Calculates the median for a vector of doubles.

4.1.1 Function Documentation

4.1.1.1 average()

```
double mdata::average (
            const std::vector< double > & v )
```

Calculates the average for a vector of doubles.

Parameters

v	Vector of double values

Returns

The average value of the vector

Definition at line 23 of file datastats.cpp.

Referenced by proj1::mfuncExperiment::runAllFunc(), and standardDeviation().

```
00024 {
00025     size_t vSize = v.size();
00026     double sum = 0.0;
00027
00028     for (size_t i = 0; i < vSize; i++)
00029         sum += v[i];
00030
00031     return sum / vSize;
00032 }
```

4.1.1.2 median()

```
double mdata::median (
            const std::vector< double > & v )
```

Calculates the median for a vector of doubles.

Parameters

v	Vector of double values

Returns

The median of the vector

Definition at line 85 of file datastats.cpp.

Referenced by proj1::mfuncExperiment::runAllFunc().

```
00086 {
00087     size_t vSize = v.size();
00088     double* vSorted = new double[vSize];
00089     double retVal = 0;
00090
00091     for (int i = 0; i < vSize; i++)
00092         vSorted[i] = v[i];
00093
00094     std::sort(vSorted, vSorted + vSize);
00095
00096     if (vSize % 2 != 0)
00097     {
00098         // Odd number of values
00099         retVal = vSorted[vSize / 2];
00100     }
00101     else
00102     {
00103         // Even number of values
00104         double low = vSorted[(vSize / 2) - 1];
00105         double high = vSorted[vSize / 2];
00106         retVal = (high + low) / 2;
00107     }
00108
00109     delete[] vSorted;
00110     return retVal;
00111 }
```

4.1.1.3 range()

```
double mdata::range (
            const std::vector< double > & v )
```

Calculates the range for a vector of doubles.

Parameters

v	Vector of double values

Returns

The range of the vector

Definition at line 61 of file datastats.cpp.

Referenced by proj1::mfuncExperiment::runAllFunc().

```
00062 {
00063     size_t vSize = v.size();
00064     double min = v[0];
00065     double max = v[0];
00066
00067     for (size_t i = 1; i < vSize; i++)
00068     {
00069         double cur = v[i];
00070
00071         if (cur < min) min = cur;
00072
00073         if (cur > max) max = cur;
00074     }
00075
00076     return max - min;
00077 }
```

4.1.1.4 standardDeviation()

```
double mdata::standardDeviation (
            const std::vector< double > & v )
```

Calculates the standard deviation for a vector of doubles.

Parameters

v	Vector of double values

Returns

The standard deviation of the vector

Definition at line 40 of file datastats.cpp.

References average().

Referenced by proj1::mfuncExperiment::runAllFunc().

```
00041 {
00042     size_t vSize = v.size();
00043     double mean = average(v);
00044     double sum = 0;
00045 
00046     for (size_t i = 0; i < vSize; i++)
00047     {
00048         double subMean = v[i] - mean;
00049         sum += subMean * subMean;
00050     }
00051 
00052     return sqrt(sum / vSize);
00053 }
```

4.2 mfunc Namespace Reference

Functions

- const char * schwefelDesc ()
- double schwefel (double *v, size_t n)

  Function 1. Implementation of Schwefel's mathematical function.
- const char * dejongDesc ()
- double dejong (double *v, size_t n)

  Function 2. Implementation of 1st De Jong's mathematical function.
- const char * rosenbrokDesc ()
- double rosenbrok (double *v, size_t n)

  Function 3. Implementation of the Rosenbrock mathematical function.
- const char * rastriginDesc ()
- double rastrigin (double *v, size_t n)

  Function 4. Implementation of the Rastrigin mathematical function.
- const char * griewangkDesc ()
- double griewangk (double *v, size_t n)

  Function 5. Implementation of the Griewangk mathematical function.
- const char * sineEnvelopeSineWaveDesc ()
- double sineEnvelopeSineWave (double *v, size_t n)

  Function 6. Implementation of the Sine Envelope Sine Wave mathematical function.
- const char * stretchedVSineWaveDesc ()
- double stretchedVSineWave (double *v, size_t n)

  Function 7. Implementation of the Stretched V Sine Wave mathematical function.
- const char * ackleysOneDesc ()
- double ackleysOne (double *v, size_t n)

  Function 8. Implementation of Ackley's One mathematical function.
- const char * ackleysTwoDesc ()
- double ackleysTwo (double *v, size_t n)

  Function 9. Implementation of Ackley's Two mathematical function.
- const char * eggHolderDesc ()
- double eggHolder (double *v, size_t n)

  Function 10. Implementation of the Egg Holder mathematical function.
- const char * ranaDesc ()
- double rana (double *v, size_t n)

  Function 11. Implementation of the Rana mathematical function.
- const char * pathologicalDesc ()
- double pathological (double *v, size_t n)

  Function 12. Implementation of the Pathological mathematical function.
- const char * michalewiczDesc ()

- double michalewicz (double *v, size_t n)

  Function 13. Implementation of the Michalewicz mathematical function.
- const char * mastersCosineWaveDesc ()
- double mastersCosineWave (double *v, size_t n)

  Function 14. Implementation of the Masters Cosine Wave mathematical function.
- const char * quarticDesc ()
- double quartic (double *v, size_t n)

  Function 15. Implementation of the Quartic mathematical function.
- const char * levyDesc ()
- double levy (double *v, size_t n)

  Function 16. Implementation of the Levy mathematical function.
- const char * stepDesc ()
- double step (double *v, size_t n)

  Function 17. Implementation of the Step mathematical function.
- const char * alpineDesc ()
- double alpine (double *v, size_t n)

  Function 18. Implementation of the Alpine mathematical function.
- bool fExec (unsigned int f, double *v, size_t n, double &outResult)

  Executes a specific function Executes the function with the given id and returns true on success. Otherwise returns false if id is invalid.
- const char * fDesc (unsigned int f)

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

Variables

- const unsigned int NUM_FUNCTIONS = 18

4.2.1 Detailed Description

Scope for all math functions

4.2.2 Function Documentation

4.2.2.1 ackleysOne()

```
double mfunc::ackleysOne (
            double * v,
            size_t n )
```

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

Parameters

v	Vector as a double array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 295 of file mfunc.cpp.

Referenced by fExec().

```
00296 {
00297     double f = 0.0;
00298
00299     for (size_t i = 0; i < n - 1; i++)
00300     {
00301         double a = (1.0 / pow(M_E, 0.2)) * sqrt(v[i]*v[i] + v[i+1]*v[i+1]);
00302         double b = 3.0 * (cos(2.0*v[i]) + sin(2.0*v[i+1]));
00303         f += a + b;
00304     }
00305
00306     return f;
00307 }
```

4.2.2.2 ackleysOneDesc()

```
const char * mfunc::ackleysOneDesc ( )
```

Returns a string description of the ackleysOne() function

Returns

C-string description

Definition at line 283 of file mfunc.cpp.

References _ackleysOneDesc.

Referenced by fDesc().

```
00284 {
00285     return _ackleysOneDesc;
00286 }
```

4.2.2.3 ackleysTwo()

```
double mfunc::ackleysTwo (
            double * v,
            size_t n )
```

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

Parameters

v	Vector as a double array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 327 of file mfunc.cpp.

Referenced by fExec().

```
00328 {
00329     double f = 0.0;
00330
00331     for (size_t i = 0; i < n - 1; i++)
00332     {
00333         double a = 20.0 / pow(M_E, 0.2 * sqrt((v[i]*v[i] + v[i+1]*v[i+1]) / 2.0));
00334         double b = pow(M_E, 0.5 * (cos(2.0 * M_PI * v[i]) + cos(2.0 * M_PI * v[i+1])));
00335         f += 20.0 + M_E - a - b;
00336     }
00337
00338     return f;
00339 }
```

4.2.2.4 ackleysTwoDesc()

```
const char * mfunc::ackleysTwoDesc ( )
```

Returns a string description of the ackleysTwo() function

Returns

C-string description

Definition at line 315 of file mfunc.cpp.

References _ackleysTwoDesc.

Referenced by fDesc().

```
00316 {
00317     return _ackleysTwoDesc;
00318 }
```

4.2.2.5 alpine()

```
double mfunc::alpine (
            double * v,
            size_t n )
```

Function 18. Implementation of the Alpine mathematical function.

Parameters

v	Vector as a double array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 627 of file mfunc.cpp.

Referenced by fExec().

```
00628 {
00629     double f = 0.0;
00630
00631     for (size_t i = 0; i < n; i++)
00632     {
00633         f += fabs(v[i] * sin(v[i]) + 0.1*v[i]);
00634     }
00635
00636     return f;
00637 }
```

4.2.2.6 alpineDesc()

```
const char * mfunc::alpineDesc ( )
```

Returns a string description of the alpine() function

Returns

C-string description

Definition at line 615 of file mfunc.cpp.

References _alpineDesc.

Referenced by fDesc().

```
00616 {
00617     return _alpineDesc;
00618 }
```

4.2.2.7 dejong()

```
double mfunc::dejong (
            double * v,
            size_t n )
```

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

Parameters

v	Vector as a double array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 99 of file mfunc.cpp.

Referenced by fExec().

```
00100 {
00101     double f = 0.0;
00102
00103     for (size_t i = 0; i < n; i++)
00104     {
00105         f += v[i] * v[i];
00106     }
00107
00108     return f;
00109 }
```

4.2.2.8 dejongDesc()

```
const char * mfunc::dejongDesc ( )
```

Returns a string description of the dejong() function

Returns

C-string description

Definition at line 87 of file mfunc.cpp.

References _dejongDesc.

Referenced by fDesc().

```
00088 {
00089     return _dejongDesc;
00090 }
```

4.2.2.9 eggHolder()

```
double mfunc::eggHolder (
            double * v,
            size_t n )
```

Function 10. Implementation of the Egg Holder mathematical function.

Parameters

v	Vector as a double array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 359 of file mfunc.cpp.

Referenced by fExec().

```
00360 {
00361     double f = 0.0;
00362
00363     for (size_t i = 0; i < n - 1; i++)
00364     {
00365         double a = -1.0 * v[i] * sin(sqrt(fabs(v[i] - v[i+1] - 47.0)));
00366         double b = (v[i+1] + 47) * sin(sqrt(fabs(v[i+1] + 47.0 + (v[i]/2.0))));
00367         f += a - b;
00368     }
00369
00370     return f;
00371 }
```

4.2.2.10 eggHolderDesc()

```
const char * mfunc::eggHolderDesc ( )
```

Returns a string description of the eggHolder() function

Returns

C-string description

Definition at line 347 of file mfunc.cpp.

References _eggHolderDesc.

Referenced by fDesc().

```
00348 {
00349     return _eggHolderDesc;
00350 }
```

4.2.2.11 fDesc()

```
const char * mfunc::fDesc (
            unsigned int f )
```

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 723 of file mfunc.cpp.

References ackleysOneDesc(), ackleysTwoDesc(), alpineDesc(), dejongDesc(), eggHolderDesc(), griewangk↩Desc(), levyDesc(), mastersCosineWaveDesc(), michalewiczDesc(), pathologicalDesc(), quarticDesc(), ranaDesc(), rastriginDesc(), rosenbrokDesc(), schwefelDesc(), sineEnvelopeSineWaveDesc(), stepDesc(), and stretchedV↩SineWaveDesc().

Referenced by proj1::mfuncExperiment::runAllFunc().

```
00724 {
00725     switch (f)
00726     {
00727         case 1:
00728             return schwefelDesc();
00729         case 2:
00730             return dejongDesc();
00731         case 3:
00732             return rosenbrokDesc();
00733         case 4:
00734             return rastriginDesc();
00735         case 5:
00736             return griewangkDesc();
00737         case 6:
00738             return sineEnvelopeSineWaveDesc();
00739         case 7:
00740             return stretchedVSineWaveDesc();
00741         case 8:
00742             return ackleysOneDesc();
00743         case 9:
00744             return ackleysTwoDesc();
00745         case 10:
00746             return eggHolderDesc();
00747         case 11:
00748             return ranaDesc();
00749         case 12:
00750             return pathologicalDesc();
00751         case 13:
00752             return michalewiczDesc();
00753         case 14:
00754             return mastersCosineWaveDesc();
00755         case 15:
00756             return quarticDesc();
00757         case 16:
00758             return levyDesc();
00759         case 17:
00760             return stepDesc();
00761         case 18:
00762             return alpineDesc();
00763         default:
00764             return NULL;
00765     }
00766 }
```

4.2.2.12 fExec()

```
bool mfunc::fExec (
            unsigned int f,
            double * v,
            size_t n,
            double & outResult )
```

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 double 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 651 of file mfunc.cpp.

References ackleysOne(), ackleysTwo(), alpine(), dejong(), eggHolder(), griewangk(), levy(), mastersCosineWave(), michalewicz(), pathological(), quartic(), rana(), rastrigin(), rosenbrok(), schwefel(), sineEnvelopeSineWave(), step(), and stretchedVSineWave().

Referenced by proj1::mfuncExperiment::runFunc().

```
00652 {
00653     switch (f)
00654     {
00655         case 1:
00656             outResult = schwefel(v, n);
00657             return true;
00658         case 2:
00659             outResult = dejong(v, n);
00660             return true;
00661         case 3:
00662             outResult = rosenbrok(v, n);
00663             return true;
00664         case 4:
00665             outResult = rastrigin(v, n);
00666             return true;
00667         case 5:
00668             outResult = griewangk(v, n);
00669             return true;
00670         case 6:
00671             outResult = sineEnvelopeSineWave(v, n);
00672             return true;
00673         case 7:
00674             outResult = stretchedVSineWave(v, n);
00675             return true;
00676         case 8:
00677             outResult = ackleysOne(v, n);
00678             return true;
00679         case 9:
00680             outResult = ackleysTwo(v, n);
00681             return true;
00682         case 10:
00683             outResult = eggHolder(v, n);
00684             return true;
00685         case 11:
00686             outResult = rana(v, n);
00687             return true;
00688         case 12:
00689             outResult = pathological(v, n);
00690             return true;
00691         case 13:
00692             outResult = michalewicz(v, n);
00693             return true;
00694         case 14:
00695             outResult = mastersCosineWave(v, n);
00696             return true;
00697         case 15:
00698             outResult = quartic(v, n);
00699             return true;
00700         case 16:
00701             outResult = levy(v, n);
00702             return true;
00703         case 17:
00704             outResult = step(v, n);
00705             return true;
00706         case 18:
```

```
00707             outResult = alpine(v, n);
00708             return true;
00709         default:
00710             return false;
00711     }
00712 }
```

4.2.2.13 griewangk()

```
double mfunc::griewangk (
            double * v,
            size_t n )
```

Function 5. Implementation of the Griewangk mathematical function.

Parameters

v	Vector as a double array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 192 of file mfunc.cpp.

Referenced by fExec().

```
00193 {
00194     double sum = 0.0;
00195     double product = 0.0;
00196 
00197     for (size_t i = 0; i < n; i++)
00198     {
00199         sum += (v[i] * v[i]) / 4000.0;
00200     }
00201 
00202     for (size_t i = 0; i < n; i++)
00203     {
00204         product *= cos(v[i] / sqrt(i + 1.0));
00205     }
00206 
00207     return 1.0 + sum - product;
00208 }
```

4.2.2.14 griewangkDesc()

```
const char * mfunc::griewangkDesc ( )
```

Returns a string description of the griewangk() function

Returns

C-string description

Definition at line 180 of file mfunc.cpp.

References _griewangkDesc.

Referenced by fDesc().

```
00181 {
00182     return _griewangkDesc;
00183 }
```

4.2.2.15 levy()

```
double mfunc::levy (
            double * v,
            size_t n )
```

Function 16. Implementation of the Levy mathematical function.

Parameters

v	Vector as a double array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 557 of file mfunc.cpp.

References w().

Referenced by fExec().

```
00558 {
00559     double f = 0.0;
00560
00561     for (size_t i = 0; i < n - 1; i++)
00562     {
00563         double a = w(v[i]) - 1.0;
00564         a *= a;
00565         double b = sin(M_PI * w(v[i]) + 1.0);
00566         b *= b;
00567         double c = w(v[n - 1]) - 1.0;
00568         c *= c;
00569         double d = sin(2.0 * M_PI * w(v[n - 1]));
00570         d *= d;
00571         f += a * (1.0 + 10.0 * b) + c * (1.0 + d);
00572     }
00573
00574     double e = sin(M_PI * w(v[0]));
00575     return e*e + f;
00576 }
```

4.2.2.16 levyDesc()

```
const char * mfunc::levyDesc ( )
```

Returns a string description of the levy() function

Returns

C-string description

Definition at line 545 of file mfunc.cpp.

References _levyDesc.

Referenced by fDesc().

```
00546 {
00547     return _levyDesc;
00548 }
```

4.2.2.17 mastersCosineWave()

```
double mfunc::mastersCosineWave (
            double * v,
            size_t n )
```

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

Parameters

v	Vector as a double array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 487 of file mfunc.cpp.

Referenced by fExec().

```
00488 {
00489     double f = 0.0;
00490
00491     for (size_t i = 0; i < n - 1; i++)
00492     {
00493         double a = pow(M_E, (-1.0/8.0)*(v[i]*v[i] + v[i+1]*v[i+1] + 0.5*v[i+1]*v[i]));
00494         double b = cos(4 * sqrt(v[i]*v[i] + v[i+1]*v[i+1] + 0.5*v[i]*v[i+1]));
00495         f += a * b;
00496     }
00497
00498     return -1.0 * f;
00499 }
```

4.2.2.18 mastersCosineWaveDesc()

```
const char * mfunc::mastersCosineWaveDesc ( )
```

Returns a string description of the mastersCosineWave() function

Returns

C-string description

Definition at line 475 of file mfunc.cpp.

References _mastersCosineWaveDesc.

Referenced by fDesc().

```
00476 {
00477     return _mastersCosineWaveDesc;
00478 }
```

4.2.2.19 michalewicz()

```
double mfunc::michalewicz (
            double * v,
            size_t n )
```

Function 13. Implementation of the Michalewicz mathematical function.

Parameters

v	Vector as a double array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 457 of file mfunc.cpp.

Referenced by fExec().

```
00458 {
00459     double f = 0.0;
00460
00461     for (size_t i = 0; i < n; i++)
00462     {
00463         f += sin(v[i]) * pow(sin(((i+1) * v[i] * v[i]) / M_PI), 20);
00464     }
00465
00466     return -1.0 * f;
00467 }
```

4.2.2.20 michalewiczDesc()

```
const char * mfunc::michalewiczDesc ( )
```

Returns a string description of the michalewicz() function

Returns

C-string description

Definition at line 445 of file mfunc.cpp.

References _michalewiczDesc.

Referenced by fDesc().

```
00446 {
00447     return _michalewiczDesc;
00448 }
```

4.2.2.21 pathological()

```
double mfunc::pathological (
            double * v,
            size_t n )
```

Function 12. Implementation of the Pathological mathematical function.

Parameters

v	Vector as a double array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 423 of file mfunc.cpp.

Referenced by fExec().

```
00424 {
00425     double f = 0.0;
00426
00427     for (size_t i = 0; i < n - 1; i++)
00428     {
00429         double a = sin(sqrt(100.0*v[i]*v[i] + v[i+1]*v[i+1]));
00430         a = (a*a) - 0.5;
00431         double b = (v[i]*v[i] - 2*v[i]*v[i+1] + v[i+1]*v[i+1]);
00432         b = 1.0 + 0.001 * b*b;
00433         f += 0.5 + (a/b);
00434     }
00435
00436     return f;
00437 }
```

4.2.2.22 pathologicalDesc()

```
const char * mfunc::pathologicalDesc ( )
```

Returns a string description of the pathological() function

Returns

C-string description

Definition at line 411 of file mfunc.cpp.

References _pathologicalDesc.

Referenced by fDesc().

```
00412 {
00413     return _pathologicalDesc;
00414 }
```

4.2.2.23 quartic()

```
double mfunc::quartic (
            double * v,
            size_t n )
```

Function 15. Implementation of the Quartic mathematical function.

Parameters

v	Vector as a double array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 519 of file mfunc.cpp.

Referenced by fExec().

```
00520 {
00521     double f = 0.0;
00522
00523     for (size_t i = 0; i < n; i++)
00524     {
00525         f += (i+1) * v[i] * v[i] * v[i] * v[i];
00526     }
00527
00528     return f;
00529 }
```

4.2.2.24 quarticDesc()

```
const char * mfunc::quarticDesc ( )
```

Returns a string description of the quartic() function

Returns

C-string description

Definition at line 507 of file mfunc.cpp.

References _quarticDesc.

Referenced by fDesc().

```
00508 {
00509     return _quarticDesc;
00510 }
```

4.2.2.25 rana()

```
double mfunc::rana (
            double * v,
            size_t n )
```

Function 11. Implementation of the Rana mathematical function.

Parameters

v	Vector as a double array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 391 of file mfunc.cpp.

Referenced by fExec().

```
00392 {
00393     double f = 0.0;
00394
00395     for (size_t i = 0; i < n - 1; i++)
00396     {
00397         double a = v[i] * sin(sqrt(fabs(v[i+1] - v[i] + 1.0))) * cos(sqrt(fabs(v[i+1] + v[i] + 1.0)));
00398         double b = (v[i+1] + 1.0) * cos(sqrt(fabs(v[i+1] - v[i] + 1.0))) * sin(sqrt(fabs(v[i+1] + v[i] + 1.
      0)));
00399         f += a + b;
00400     }
00401
00402     return f;
00403 }
```

4.2.2.26 ranaDesc()

```
const char * mfunc::ranaDesc ( )
```

Returns a string description of the rana() function

Returns

C-string description

Definition at line 379 of file mfunc.cpp.

References _ranaDesc.

Referenced by fDesc().

```
00380 {
00381     return _ranaDesc;
00382 }
```

4.2.2.27 rastrigin()

```
double mfunc::rastrigin (
            double * v,
            size_t n )
```

Function 4. Implementation of the Rastrigin mathematical function.

Parameters

v	Vector as a double array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 162 of file mfunc.cpp.

Referenced by fExec().

```
00163 {
00164     double f = 0.0;
00165
00166     for (size_t i = 0; i < n; i++)
00167     {
00168         f += (v[i] * v[i]) - (10.0 * cos(2.0 * M_PI * v[i]));
00169     }
00170
00171     return 10.0 * n * f;
00172 }
```

4.2.2.28 rastriginDesc()

```
const char * mfunc::rastriginDesc ( )
```

Returns a string description of the rastrigin() function

Returns

C-string description

Definition at line 150 of file mfunc.cpp.

References _rastriginDesc.

Referenced by fDesc().

```
00151 {
00152     return _rastriginDesc;
00153 }
```

4.2.2.29 rosenbrok()

```
double mfunc::rosenbrok (
            double * v,
            size_t n )
```

Function 3. Implementation of the Rosenbrock mathematical function.

Parameters

v	Vector as a double array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 129 of file mfunc.cpp.

Referenced by fExec().

```
00130 {
00131     double f = 0.0;
00132
00133     for (size_t i = 0; i < n - 1; i++)
00134     {
00135         double a = ((v[i] * v[i]) - v[i+1]);
00136         double b = (1.0 - v[i]);
00137         f += 100.0 * a * a;
00138         f += b * b;
00139     }
00140
00141     return f;
00142 }
```

4.2.2.30 rosenbrokDesc()

```
const char * mfunc::rosenbrokDesc ( )
```

Returns a string description of the rosenbrok() function

Returns

C-string description

Definition at line 117 of file mfunc.cpp.

References _rosenbrokDesc.

Referenced by fDesc().

```
00118 {
00119     return _rosenbrokDesc;
00120 }
```

4.2.2.31 schwefel()

```
double mfunc::schwefel (
            double * v,
            size_t n )
```

Function 1. Implementation of Schwefel's mathematical function.

Parameters

v	Vector as a double array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 69 of file mfunc.cpp.

Referenced by fExec().

```
00070 {
00071     double f = 0.0;
00072
00073     for (size_t i = 0; i < n; i++)
00074     {
00075         f += (-1.0 * v[i]) * sin(sqrt(fabs(v[i])));
00076     }
00077
00078     return (418.9829 * n) - f;
00079 }
```

4.2.2.32 schwefelDesc()

```
const char * mfunc::schwefelDesc ( )
```

Returns a string description of the schwefel() function

Returns

C-string description

Definition at line 57 of file mfunc.cpp.

References _schwefelDesc.

Referenced by fDesc().

```
00058 {
00059     return _schwefelDesc;
00060 }
```

4.2.2.33 sineEnvelopeSineWave()

```
double mfunc::sineEnvelopeSineWave (
            double * v,
            size_t n )
```

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

Parameters

v	Vector as a double array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 228 of file mfunc.cpp.

Referenced by fExec().

```
00229 {
00230     double f = 0.0;
00231
00232     for (size_t i = 0; i < n - 1; i++)
00233     {
00234         double a = sin(v[i]*v[i] + v[i+1]*v[i+1] - 0.5);
00235         a *= a;
00236         double b = (1 + 0.001*(v[i]*v[i] + v[i+1]*v[i+1]));
00237         b *= b;
00238         f += 0.5 + (a / b);
00239     }
00240
00241     return -1.0 * f;
00242 }
```

4.2.2.34 sineEnvelopeSineWaveDesc()

```
const char * mfunc::sineEnvelopeSineWaveDesc ( )
```

Returns a string description of the sineEnvelopeSineWave() function

Returns

C-string description

Definition at line 216 of file mfunc.cpp.

References _sineEnvelopeSineWaveDesc.

Referenced by fDesc().

```
00217 {
00218     return _sineEnvelopeSineWaveDesc;
00219 }
```

4.2.2.35 step()

```
double mfunc::step (
            double * v,
            size_t n )
```

Function 17. Implementation of the Step mathematical function.

Parameters

v	Vector as a double array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 596 of file mfunc.cpp.

Referenced by fExec().

```
00597 {
00598     double f = 0.0;
00599
00600     for (size_t i = 0; i < n; i++)
00601     {
00602         double a = fabs(v[i]) + 0.5;
00603         f += a * a;
00604     }
00605
00606     return f;
00607 }
```

4.2.2.36 stepDesc()

```
const char * mfunc::stepDesc ( )
```

Returns a string description of the step() function

Returns

C-string description

Definition at line 584 of file mfunc.cpp.

References _stepDesc.

Referenced by fDesc().

```
00585 {
00586     return _stepDesc;
00587 }
```

4.2.2.37 stretchedVSineWave()

```
double mfunc::stretchedVSineWave (
            double * v,
            size_t n )
```

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

Parameters

v	Vector as a double array
n	Size of the vector 'v'

Returns

The result of the mathematical function

Definition at line 262 of file mfunc.cpp.

References nthroot().

Referenced by fExec().

```
00263 {
00264     double f = 0.0;
00265
00266     for (size_t i = 0; i < n - 1; i++)
00267     {
00268         double a = nthroot(v[i]*v[i] + v[i+1]*v[i+1], 4.0);
00269         double b = sin(50.0 * nthroot(v[i]*v[i] + v[i+1]*v[i+1], 10.0));
00270         b *= b;
00271         f += a * b + 1.0;
00272     }
00273
00274     return f;
00275 }
```

4.2.2.38 stretchedVSineWaveDesc()

```
const char * mfunc::stretchedVSineWaveDesc ( )
```

Returns a string description of the stretchedVSineWave() function

Returns

C-string description

Definition at line 250 of file mfunc.cpp.

References _stretchedVSineWaveDesc.

Referenced by fDesc().

```
00251 {
00252     return _stretchedVSineWaveDesc;
00253 }
```

4.2.3 Variable Documentation

4.2.3.1 NUM_FUNCTIONS

```
const unsigned int mfunc::NUM_FUNCTIONS = 18
```

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

Definition at line 49 of file mfunc.cpp.

Referenced by proj1::mfuncExperiment::runAllFunc(), and proj1::mfuncExperiment::runFunc().

4.3 proj1 Namespace Reference

Classes

- class mfuncExperiment

  Contains classes for running the CS471 project 1 experiment.
- struct RandomBounds

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

4.4 util Namespace Reference

Classes

- class IniReader

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

Chapter 5

Class Documentation

5.1 mdata::DataTable Class Reference

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

`#include <datatable.h>`

Public Member Functions

- DataTable (unsigned int cols)

  Constructs a new DataTable object with a specified number of columns.
- ~DataTable ()

  Destroys the DataTable object.
- std::string getColLabel (unsigned int colIndex)

  Returns the label for the column at the specified index. The first column = index 0.
- bool setColLabel (unsigned int colIndex, std::string newLabel)

  Sets the label for the column at the specified index. The first column = index 0.
- unsigned int addRow ()

  Adds a new row to the end of the table.
- unsigned int addRow (const std::vector< std::string > &rowData)

  Adds a new row to the end of the table and fills the row with the data given in the vector of strings rowData.
- std::vector< std::string > & getRow (unsigned int row)

  Returns a reference to the string vector that contains the entries for the given row index.
- void setRow (unsigned int row, const std::vector< std::string > &rowData)

  Sets the data entries for the row at the given index.
- std::string getEntry (unsigned int row, unsigned int col)

  Returns the string value of the entry at the given row and column indices.
- void setEntry (unsigned int row, unsigned int col, std::string val)

  Sets the value of the entry at the given row and column indices.
- void setEntry (unsigned int row, unsigned int col, int val)

  Sets the value of the entry at the given row and column indices.
- void setEntry (unsigned int row, unsigned int col, long val)

  Sets the value of the entry at the given row and column indices.
- void setEntry (unsigned int row, unsigned int col, float val)

  Sets the value of the entry at the given row and column indices.
- void setEntry (unsigned int row, unsigned int col, double val)

  Sets the value of the entry at the given row and column indices.
- bool exportCSV (const char *filePath)

  *Exports the current data table to the given file path in the *.csv format. If the file already exists, it is replaced.*

Protected Attributes

- unsigned int cols
- unsigned int rows
- std::vector< std::string > colLabels
- std::map< unsigned int, std::vector< std::string > > tableData

5.1.1 Detailed Description

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

– Initialize a DataTable object with a specified number of columns: DataTable table(n);

Set a column's label:

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

Add a row to the table: int rowIndex = table.addRow();

or

int rowIndex = table.addRow((std::vector<std::string>)dataVector);

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 55 of file datatable.h.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 DataTable()

```
DataTable::DataTable (
            unsigned int columns )
```

Constructs a new DataTable object with a specified number of columns.

Parameters

columns	The number of columns to be created for the table

Definition at line 24 of file datatable.cpp.

References colLabels, and cols.

```
00024                                           : rows(0), cols(columns),
      colLabels(columns)
00025 {
00026     for (int i = 0; i < cols; i++)
00027     {
00028         colLabels.push_back("(No label)");
00029     }
00030 }
```

5.1.2.2 ~DataTable()

```
DataTable::~DataTable ( )
```

Destroys the DataTable object.

Definition at line 35 of file datatable.cpp.

References colLabels, and tableData.

```
00036 {
00037     colLabels.clear();
00038     tableData.clear();
00039 }
```

5.1.3 Member Function Documentation

5.1.3.1 addRow() [1/2]

```
unsigned int DataTable::addRow ( )
```

Adds a new row to the end of the table.

Returns

The index of the newly added row

Definition at line 77 of file datatable.cpp.

References cols, rows, and tableData.

Referenced by proj1::mfuncExperiment::runAllFunc().

```
00078 {
00079     unsigned int newRowIndex = rows;
00080     rows++;
00081
00082     auto& tableRow = tableData[newRowIndex];
00083     tableRow.clear();
00084
00085     for (int i = 0; i < cols; i++)
00086     {
00087         tableRow.push_back("");
00088     }
00089
00090     return newRowIndex;
00091 }
```

5.1.3.2 addRow() [2/2]

```
unsigned int DataTable::addRow (
            const std::vector< std::string > & rowData )
```

Adds a new row to the end of the table and fills the row with the data given in the vector of strings rowData.

Parameters

rowData	Vector of strings to be entered into the table. rowData[n] = Column[n]

Returns

The index of the newly added row

Definition at line 100 of file datatable.cpp.

References rows, and setRow().

```
00101 {
00102     unsigned int newRowIndex = rows;
00103     rows++;
00104     setRow(newRowIndex, rowData);
00105
00106     return newRowIndex;
00107 }
```

5.1.3.3 exportCSV()

```
bool DataTable::exportCSV (
            const char * filePath )
```

Exports the current data table to the given file path in the ∗.csv format. If the file already exists, it is replaced.

Parameters

filePath	

Returns

true
false

Definition at line 231 of file datatable.cpp.

References colLabels, cols, rows, and tableData.

Referenced by proj1::mfuncExperiment::runAllFunc().

```
00232 {
00233     using namespace std;
00234
00235     ofstream outFile;
00236     outFile.open(filePath, ofstream::out | ofstream::trunc);
00237     if (!outFile.good()) return false;
00238
00239     // Print column labels
00240     for (unsigned int c = 0; c < cols; c++)
00241     {
00242         outFile << colLabels[c];
00243         if (c < cols - 1) outFile << ",";
00244     }
00245
00246     outFile << endl;
00247
00248     // Print data rows
00249     for (unsigned int r = 0; r < rows; r++)
00250     {
00251         for (unsigned int c = 0; c < cols; c++)
00252         {
00253             outFile << tableData[r][c];
00254             if (c < cols - 1) outFile << ",";
00255         }
00256         outFile << endl;
00257     }
00258
00259     outFile.close();
00260     return true;
00261 }
```

5.1.3.4 getColLabel()

```
std::string DataTable::getColLabel (
            unsigned int colIndex )
```

Returns the label for the column at the specified index. The first column = index 0.

Parameters

colIndex	Column index

Returns

A std::string containing the column label

Definition at line 48 of file datatable.cpp.

References colLabels, and cols.

```
00049 {
00050     if (colIndex >= cols) throw "Invalid Column Index";
00051
00052     return colLabels[colIndex];
00053 }
```

5.1.3.5 getEntry()

```
std::string DataTable::getEntry (
            unsigned int row,
            unsigned int col )
```

Returns the string value of the entry at the given row and column indices.

Parameters

row	Index of the row you wish to access
col	Index of the column you wish to access

Returns

The value of the given row and column. Throws a string exception of the row or column is out of bounds.

Definition at line 154 of file datatable.cpp.

References cols, rows, and tableData.

```
00155 {
00156     if (row >= rows || col >= cols) throw "Invalid row/column";
00157
00158     return tableData[row][col];
00159 }
```

5.1.3.6 getRow()

```
std::vector< std::string > & DataTable::getRow (
            unsigned int row )
```

Returns a reference to the string vector that contains the entries for the given row index.

Parameters

row	Index of the row that you wish to access.

Returns

std::vector<std::string>&

Definition at line 116 of file datatable.cpp.

References rows, and tableData.

```
00117 {
00118     if (row >= rows) throw "Invalid row index";
00119
00120     return tableData[row];
00121 }
```

5.1.3.7 setColLabel()

```
bool DataTable::setColLabel (
            unsigned int colIndex,
            std::string newLabel )
```

Sets the label for the column at the specified index. The first column = index 0.

Parameters

colIndex	Column index
newLabel	std::string containing the new column label

Returns

true If the column label was succesfully changed.
false If the column index was invalid.

Definition at line 64 of file datatable.cpp.

References colLabels, and cols.

Referenced by proj1::mfuncExperiment::runAllFunc().

```
00065 {
00066     if (colIndex >= cols) return false;
00067
00068     colLabels[colIndex] = newLabel;
00069     return true;
00070 }
```

5.1.3.8 setEntry() [1/5]

```
void DataTable::setEntry (
            unsigned int row,
            unsigned int col,
            std::string val )
```

Sets the value of the entry at the given row and column indices.

Parameters

row	Index of the row you wish to access
col	Index of the column you wish to access
val	The new value for the entry

Definition at line 168 of file datatable.cpp.

References cols, rows, and tableData.

Referenced by proj1::mfuncExperiment::runAllFunc(), and setEntry().

```
00169 {
00170     if (row >= rows || col >= cols) throw "Invalid row/column";
00171
00172     tableData[row][col] = val;
00173 }
```

5.1.3.9 setEntry() [2/5]

```
void DataTable::setEntry (
            unsigned int row,
            unsigned int col,
            int val )
```

Sets the value of the entry at the given row and column indices.

Parameters

row	Index of the row you wish to access
col	Index of the column you wish to access
val	The new value for the entry

Definition at line 182 of file datatable.cpp.

References setEntry().

```
00183 {
00184     setEntry(row, col, std::to_string(val));
00185 }
```

5.1.3.10 setEntry() [3/5]

```
void DataTable::setEntry (
            unsigned int row,
            unsigned int col,
            long val )
```

Sets the value of the entry at the given row and column indices.

Parameters

row	Index of the row you wish to access
col	Index of the column you wish to access
val	The new value for the entry

Definition at line 194 of file datatable.cpp.

References setEntry().

```
00195 {
00196     setEntry(row, col, std::to_string(val));
00197 }
```

5.1.3.11 setEntry() [4/5]

```
void DataTable::setEntry (
            unsigned int row,
            unsigned int col,
            float val )
```

Sets the value of the entry at the given row and column indices.

Parameters

row	Index of the row you wish to access
col	Index of the column you wish to access
val	The new value for the entry

Definition at line 206 of file datatable.cpp.

References setEntry().

```
00207 {
00208     setEntry(row, col, std::to_string(val));
00209 }
```

5.1.3.12 setEntry() [5/5]

```
void DataTable::setEntry (
            unsigned int row,
            unsigned int col,
            double val )
```

Sets the value of the entry at the given row and column indices.

Parameters

row	Index of the row you wish to access
col	Index of the column you wish to access
val	The new value for the entry

Definition at line 218 of file datatable.cpp.

References setEntry().

```
00219 {
00220     setEntry(row, col, std::to_string(val));
00221 }
```

5.1.3.13 setRow()

```
void DataTable::setRow (
            unsigned int row,
            const std::vector< std::string > & rowData )
```

Sets the data entries for the row at the given index.

Parameters

row	Index of the row that you wish to update.
rowData	Vector of strings that contain the new row data entries.

Definition at line 129 of file datatable.cpp.

References cols, rows, and tableData.

Referenced by addRow().

```
00130 {
00131     if (row >= rows) throw "Invalid row index";
00132
00133     auto& tableRow = tableData[row];
00134     tableRow.clear();
00135
00136     for (unsigned int i = 0; i < cols; i++)
00137     {
00138         if (i < rowData.size())
00139             tableRow.push_back(rowData[i]);
00140         else
00141             tableRow.push_back("(No data)");
00142     }
00143 }
```

5.1.4 Member Data Documentation

5.1.4.1 colLabels

```
std::vector<std::string> mdata::DataTable::colLabels  [protected]
```

Number of rows in the table.

Definition at line 80 of file datatable.h.

Referenced by DataTable(), exportCSV(), getColLabel(), setColLabel(), and ~DataTable().

5.1.4.2 cols

```
unsigned int mdata::DataTable::cols  [protected]
```

Definition at line 78 of file datatable.h.

Referenced by addRow(), DataTable(), exportCSV(), getColLabel(), getEntry(), setColLabel(), setEntry(), and setRow().

5.1.4.3 rows

```
unsigned int mdata::DataTable::rows  [protected]
```

Number of columns in the table.

Definition at line 79 of file datatable.h.

Referenced by addRow(), exportCSV(), getEntry(), getRow(), setEntry(), and setRow().

5.1.4.4 tableData

```
std::map<unsigned int, std::vector<std::string> > mdata::DataTable::tableData  [protected]
```

Vector of column labels. Index n = Col n.

Definition at line 81 of file datatable.h.

Referenced by addRow(), exportCSV(), getEntry(), getRow(), setEntry(), setRow(), and ~DataTable().

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

- include/datatable.h
- src/datatable.cpp

5.2 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.

Protected Member Functions

- bool parseFile ()

  Protected helper function that is called by IniReader::openFile(). Parses the complete ini file and stores all sections and entries in memory.
- void parseEntry (const std::string &sectionName, const std::string &entry)

  Protected helper function that is called by IniReader::parseFile(). Parses a single entry by extracting the key and value.

Protected Attributes

- std::string file
- std::map< std::string, std::map< std::string, std::string > > iniMap

5.2.1 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 45 of file inireader.h.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 IniReader()

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

Construct a new IniReader object.

Definition at line 21 of file inireader.cpp.

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

5.2.2.2 ~IniReader()

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

Destroys the IniReader object.

Definition at line 28 of file inireader.cpp.

References iniMap.

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

5.2.3 Member Function Documentation

5.2.3.1 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.

References iniMap.

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 }
```

5.2.3.2 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(), and iniMap.

Referenced by proj1::mfuncExperiment::init(), and proj1::mfuncExperiment::runFunc().

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

5.2.3.3 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.

References file, and parseFile().

Referenced by proj1::mfuncExperiment::init().

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

5.2.3.4 parseEntry()

```
void IniReader::parseEntry (
            const std::string & sectionName,
            const std::string & entry )  [protected]
```

Protected helper function that is called by IniReader::parseFile(). Parses a single entry by extracting the key and value.

Definition at line 144 of file inireader.cpp.

References iniMap.

Referenced by parseFile().

```
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 }
```

5.2.3.5 parseFile()

```
bool IniReader::parseFile ( )  [protected]
```

Protected helper function that is called by IniReader::openFile(). Parses the complete ini file and stores all sections and entries in memory.

The parsed ini file data.

Returns

Returns true if the file was succesfully opened and parsed.

Definition at line 97 of file inireader.cpp.

References file, iniMap, and parseEntry().

Referenced by openFile().

```
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 }
```

5.2.3.6 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.

References iniMap.

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

5.2.4 Member Data Documentation

5.2.4.1 file

```
std::string util::IniReader::file  [protected]
```

Definition at line 55 of file inireader.h.

Referenced by openFile(), and parseFile().

5.2.4.2 iniMap

```
std::map<std::string, std::map<std::string, std::string> > util::IniReader::iniMap  [protected]
```

The file path for the current ini file data.

Definition at line 56 of file inireader.h.

Referenced by entryExists(), getEntry(), parseEntry(), parseFile(), sectionExists(), and ~IniReader().

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

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

5.3 proj1::mfuncExperiment Class Reference

Contains classes for running the CS471 project 1 experiment.

```
#include <proj1.h>
```

Public Member Functions

- mfuncExperiment ()

  Construct a new mfuncExperiment object.
- ~mfuncExperiment ()

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

  Initializes the CS471 project 1 experiment. Opens the given parameter file and extracts test parameters. Allocates memory for function vectors and function bounds. Extracts all function bounds.
- int runAllFunc ()

  *Executes all functions as specified in the CS471 project 1 document, records results, computes statistics, and outputs the data as a *.csv file.*
- int runFunc (unsigned int funcId, std::vector< double > &resultArrOut, double &timeOut)

  Runs the specifed function given by it's function id a certain number of times, records the execution time, and appends all results to the resultArrOut reference vector.

5.3.1 Detailed Description

Contains classes for running the CS471 project 1 experiment.

The mfuncExperiment class opens a given parameter .ini file and executes the CS471 project 1 experiment with the specified parameters. runAllFunc() runs all 18 functions defined in mfunc.cpp 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 45 of file proj1.h.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 mfuncExperiment()

```
mfuncExperiment::mfuncExperiment ( )
```

Construct a new mfuncExperiment object.

Definition at line 26 of file proj1.cpp.

```
00026                                : vMatrix(nullptr), vBounds(nullptr), nbrDim(0), nbrSol(0)
00027 {
00028 }
```

5.3.2.2 ~mfuncExperiment()

```
mfuncExperiment::~mfuncExperiment ( )
```

Destroys the mfuncExperiment object.

Definition at line 34 of file proj1.cpp.

```
00035 {
00036     releaseVMatrix();
00037     releaseVBounds();
00038 }
```

5.3.3 Member Function Documentation

5.3.3.1 init()

```
bool mfuncExperiment::init (
            const char * paramFile )
```

Initializes the CS471 project 1 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 48 of file proj1.cpp.

References util::IniReader::getEntry(), and util::IniReader::openFile().

Referenced by main().

```
00049 {
00050     // Open and parse parameters file
00051     if (!iniParams.openFile(paramFile))
00052     {
00053         cout << "Experiment init failed: Unable to open param file: " << paramFile << endl;
00054         return false;
00055     }
00056
00057     long numberSol;
00058     long numberDim;
00059
00060     // Attempt to parse number of solutions and vector dimensions size
00061     // from iniParams
00062     try
00063     {
00064         std::string entry;
00065
00066         entry = iniParams.getEntry("test", "number");
```

```
00067         if (entry.empty())
00068         {
00069             cout << "Experiment init failed: Param file missing [test]->number entry: " << paramFile <<
      endl;
00070             return false;
00071         }
00072 
00073         numberSol = std::atol(entry.c_str());
00074 
00075         entry = iniParams.getEntry("test", "dimensions");
00076         if (entry.empty())
00077         {
00078             cout << "Experiment init failed: Param file missing [test]->dimensions entry: " << paramFile <<
       endl;
00079             return false;
00080         }
00081 
00082         numberDim = std::atol(entry.c_str());
00083 
00084         if (numberSol <= 0)
00085         {
00086             cout << "Experiment init failed: Param file [test]->number entry out of bounds: " << paramFile
      << endl;
00087             return false;
00088         }
00089 
00090         if (numberDim <= 0)
00091         {
00092             cout << "Experiment init failed: Param file [test]->dimensions entry out of bounds: " <<
      paramFile << endl;
00093             return false;
00094         }
00095     }
00096     catch (const std::exception& ex)
00097     {
00098         cout << "Experiment init failed: Exception while parsing param file: " << paramFile << endl;
00099         return false;
00100     }
00101 
00102     nbrSol = (size_t)numberSol;
00103     nbrDim = (size_t)numberDim;
00104 
00105     // Get csv output file path
00106     resultsFile = iniParams.getEntry("test", "output_file");
00107 
00108     // Allocate memory for vector * solutions matrix
00109     if (!allocateVMatrix())
00110     {
00111         cout << "Experiment init failed: Unable to allocate vector matrix." << endl;
00112         return false;
00113     }
00114 
00115     // Allocate memory for function bounds
00116     if (!allocateVBounds())
00117     {
00118         cout << "Experiment init failed: Unable to allocate vector bounds array." << endl;
00119         return false;
00120     }
00121 
00122     // Fill function bounds array with data parsed from iniParams
00123     if (!parseFuncBounds())
00124     {
00125         cout << "Experiment init failed: Unable to parse vector bounds array." << endl;
00126         return false;
00127     }
00128 
00129     return true;
00130 }
```

5.3.3.2 runAllFunc()

```
int mfuncExperiment::runAllFunc ( )
```

Executes all functions as specified in the CS471 project 1 document, records results, computes statistics, and outputs the data as a $*$.csv file.

Returns

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

Definition at line 139 of file proj1.cpp.

References mdata::DataTable::addRow(), mdata::average(), mdata::DataTable::exportCSV(), mfunc::fDesc(), mdata::median(), mfunc::NUM_FUNCTIONS, mdata::range(), runFunc(), mdata::DataTable::setColLabel(), mdata::DataTable::setEntry(), and mdata::standardDeviation().

Referenced by main().

```
00140 {
00141     if (vMatrix == nullptr || nbrDim == 0 || nbrSol == 0) return 1;
00142 
00143     // function desc. | average | standard dev. | range | median | time
00144     mdata::DataTable resultsTable(8);
00145     resultsTable.setColLabel(0, "Function");
00146     resultsTable.setColLabel(1, "Vector Min");
00147     resultsTable.setColLabel(2, "Vector Max");
00148     resultsTable.setColLabel(3, "Average");
00149     resultsTable.setColLabel(4, "Standard Deviation");
00150     resultsTable.setColLabel(5, "Range");
00151     resultsTable.setColLabel(6, "Median");
00152     resultsTable.setColLabel(7, "Total Time (ms)");
00153 
00154     // Create a vector which is used to store all function results
00155     std::vector<double> fResults;
00156     double fTime = 0.0;
00157 
00158     // Execute all functions
00159     for (unsigned int f = 1; f <= mfunc::NUM_FUNCTIONS; f++)
00160     {
00161         int err = runFunc(f, fResults, fTime);
00162         if (err)
00163             return err;
00164         else
00165         {
00166             // Insert function result and statistics into results table as a new row
00167             unsigned int rowIndex = resultsTable.addRow();
00168             resultsTable.setEntry(rowIndex, 0, mfunc::fDesc(f));
00169             resultsTable.setEntry(rowIndex, 1, to_string(vBounds[f-1].min));
00170             resultsTable.setEntry(rowIndex, 2, to_string(vBounds[f-1].max));
00171             resultsTable.setEntry(rowIndex, 3, mdata::average(fResults));
00172             resultsTable.setEntry(rowIndex, 4, mdata::standardDeviation(fResults));
00173             resultsTable.setEntry(rowIndex, 5, mdata::range(fResults));
00174             resultsTable.setEntry(rowIndex, 6, mdata::median(fResults));
00175             resultsTable.setEntry(rowIndex, 7, fTime);
00176         }
00177     }
00178 
00179     if (!resultsFile.empty())
00180     {
00181         // Export results table to a *.csv file
00182         cout << "Exporting results to: " << resultsFile << endl;
00183         resultsTable.exportCSV(resultsFile.c_str());
00184     }
00185 
00186     return 0;
00187 }
```

5.3.3.3 runFunc()

```
int mfuncExperiment::runFunc (
            unsigned int funcId,
            std::vector< double > & resultArrOut,
            double & timeOut )
```

Runs the specifed function given by it's function id a certain number of times, records the execution time, and appends all results to the resultArrOut reference vector.

Parameters

funcId	The id of the function to run
resultArrOut	Out reference variable that function results are appended to
timeOut	Out reference variable that the execution time in ms is set to.

Returns

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

Definition at line 198 of file proj1.cpp.

References mfunc::fExec(), util::IniReader::getEntry(), proj1::RandomBounds::max, proj1::RandomBounds::min, and mfunc::NUM_FUNCTIONS.

Referenced by runAllFunc().

```
00199 {
00200     if (!genFuncVectors(funcId)) return 1;
00201
00202     resultArrOut.clear();
00203     resultArrOut.reserve(nbrSol);
00204
00205     double fResult = 0;
00206
00207     high_resolution_clock::time_point t_start = high_resolution_clock::now();
00208
00209     for (int i = 0; i < nbrSol; i++)
00210     {
00211         if (!mfunc::fExec(funcId, vMatrix[i], nbrDim, fResult))
00212             return 2;
00213
00214         resultArrOut.push_back(fResult);
00215     }
00216
00217     high_resolution_clock::time_point t_end = high_resolution_clock::now();
00218     timeOut = duration_cast<nanoseconds>(t_end - t_start).count() / 1000000.0;
00219
00220     return 0;
00221 }
```

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

- include/proj1.h
- src/proj1.cpp

5.4 proj1::RandomBounds Struct Reference

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

`#include <proj1.h>`

Public Attributes

- double min = 0.0
- double max = 0.0

5.4.1 Detailed Description

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

Definition at line 29 of file proj1.h.

5.4.2 Member Data Documentation

5.4.2.1 max

```
double proj1::RandomBounds::max = 0.0
```

Definition at line 32 of file proj1.h.

Referenced by proj1::mfuncExperiment::runFunc().

5.4.2.2 min

```
double proj1::RandomBounds::min = 0.0
```

Definition at line 31 of file proj1.h.

Referenced by proj1::mfuncExperiment::runFunc().

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

- include/proj1.h

Chapter 6

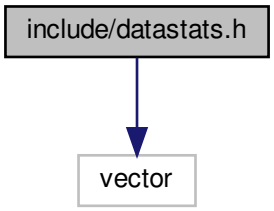
File Documentation

6.1 include/datastats.h File Reference

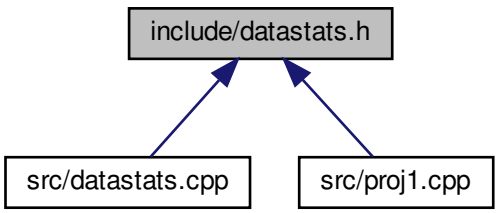
Header file for various data statistic functions.

```
#include <vector>
```

Include dependency graph for datastats.h:



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



Namespaces

- mdata

Functions

- double mdata::average (const std::vector< double > &v)

  Calculates the average for a vector of doubles.
- double mdata::standardDeviation (const std::vector< double > &v)

  Calculates the standard deviation for a vector of doubles.
- double mdata::range (const std::vector< double > &v)

  Calculates the range for a vector of doubles.
- double mdata::median (const std::vector< double > &v)

  Calculates the median for a vector of doubles.

6.1.1 Detailed Description

Header file for various data statistic functions.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-04-01

Copyright

Definition in file datastats.h.

6.2 datastats.h

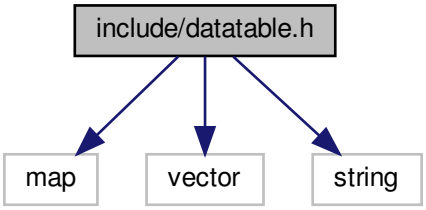
```
00001
00012 #ifndef __DATASTATS_H
00013 #define __DATASTATS_H
00014
00015 #include <vector>
00016
00017 namespace mdata
00018 {
00019     double average(const std::vector<double>& v);
00020     double standardDeviation(const std::vector<double>& v);
00021     double range(const std::vector<double>& v);
00022     double median(const std::vector<double>& v);
00023 }
00024
00025 #endif
00026
00027 // =========================
00028 // End of datastats.h
00029 // =========================
```

6.3 include/datatable.h File Reference

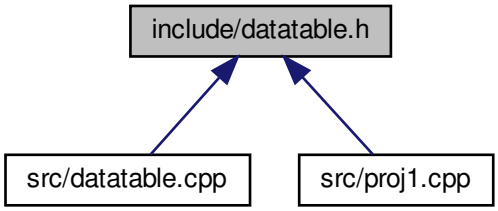
Header file for the DataTable class, which represents a spreadsheet/table of values that can be exported to a ∗.csv file.

```
#include <map>
#include <vector>
#include <string>
```

Include dependency graph for datatable.h:



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



Classes

- class mdata::DataTable

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

Namespaces

- mdata

6.3.1 Detailed Description

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

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-04-01

Copyright

Copyright (c) 2019

Definition in file datatable.h.

6.4 datatable.h

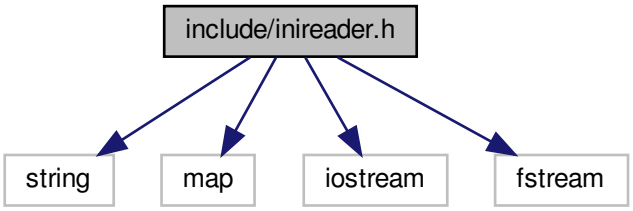
```
00001
00013 #ifndef __DATATABLE_H
00014 #define __DATATABLE_H
00015
00016 #include <map>
00017 #include <vector>
00018 #include <string>
00019
00020 namespace mdata
00021 {
00055     class DataTable
00056     {
00057     public:
00058         DataTable(unsigned int cols);
00059         ~DataTable();
00060
00061         std::string getColLabel(unsigned int colIndex);
00062         bool setColLabel(unsigned int colIndex, std::string newLabel);
00063
00064         unsigned int addRow();
00065         unsigned int addRow(const std::vector<std::string>& rowData);
00066         std::vector<std::string>& getRow(unsigned int row);
00067         void setRow(unsigned int row, const std::vector<std::string>& rowData);
00068
00069         std::string getEntry(unsigned int row, unsigned int col);
00070         void setEntry(unsigned int row, unsigned int col, std::string val);
00071         void setEntry(unsigned int row, unsigned int col, int val);
00072         void setEntry(unsigned int row, unsigned int col, long val);
00073         void setEntry(unsigned int row, unsigned int col, float val);
00074         void setEntry(unsigned int row, unsigned int col, double val);
00075
00076         bool exportCSV(const char* filePath);
00077     protected:
00078         unsigned int cols;
00079         unsigned int rows;
00080         std::vector<std::string> colLabels;
00081         std::map<unsigned int, std::vector<std::string>> tableData;
00082     };
00083 } // mdata
00084
00085 #endif
00086
00087 // =========================
00088 // End of datatable.h
00089 // =========================
```

6.5 include/inireader.h File Reference

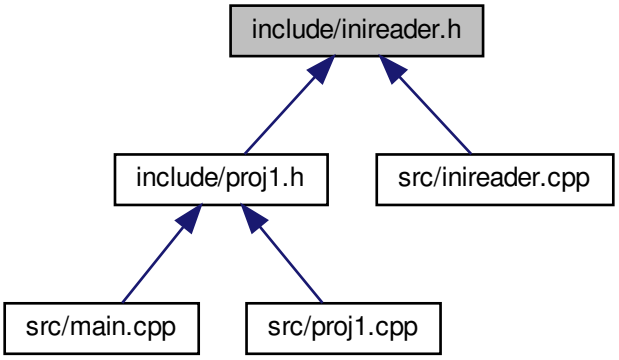
Header file for the IniReader class, which can open and parse simple ∗.ini files.

```
#include <string>
#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

6.5.1 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.

6.6 inireader.h

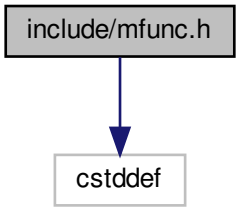
```
00001
00013 #ifndef __INIREADER_H
00014 #define __INIREADER_H
00015
00016 #include <string>
00017 #include <map>
00018 #include <iostream>
00019 #include <fstream>
00020
00021 namespace util
00022 {
00045     class IniReader
00046     {
00047     public:
00048         IniReader();
00049         ~IniReader();
00050         bool openFile(std::string filePath);
00051         bool sectionExists(std::string section);
00052         bool entryExists(std::string section, std::string entry);
00053         std::string getEntry(std::string section, std::string entry);
00054     protected:
00055         std::string file;
00056         std::map<std::string, std::map<std::string, std::string>> iniMap;
00058         bool parseFile();
00059         void parseEntry(const std::string& sectionName, const std::string& entry);
00060     };
00061 }
00062
00063 #endif
00064
00065 // ========================
00066 // End of inireader.h
00067 // ========================
```

6.7 include/mfunc.h File Reference

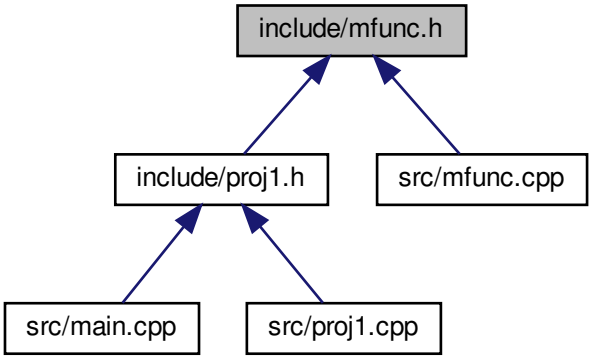
Contains various math function definitions.

```
#include <cstddef>
```

Include dependency graph for mfunc.h:



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



Namespaces

- mfunc

Functions

- const char * mfunc::schwefelDesc ()
- double mfunc::schwefel (double *v, size_t n)

  Function 1. Implementation of Schwefel's mathematical function.
- const char * mfunc::dejongDesc ()

- double mfunc::dejong (double *v, size_t n)

  Function 2. Implementation of 1st De Jong's mathematical function.
- const char * mfunc::rosenbrokDesc ()
- double mfunc::rosenbrok (double *v, size_t n)

  Function 3. Implementation of the Rosenbrock mathematical function.
- const char * mfunc::rastriginDesc ()
- double mfunc::rastrigin (double *v, size_t n)

  Function 4. Implementation of the Rastrigin mathematical function.
- const char * mfunc::griewangkDesc ()
- double mfunc::griewangk (double *v, size_t n)

  Function 5. Implementation of the Griewangk mathematical function.
- const char * mfunc::sineEnvelopeSineWaveDesc ()
- double mfunc::sineEnvelopeSineWave (double *v, size_t n)

  Function 6. Implementation of the Sine Envelope Sine Wave mathematical function.
- const char * mfunc::stretchedVSineWaveDesc ()
- double mfunc::stretchedVSineWave (double *v, size_t n)

  Function 7. Implementation of the Stretched V Sine Wave mathematical function.
- const char * mfunc::ackleysOneDesc ()
- double mfunc::ackleysOne (double *v, size_t n)

  Function 8. Implementation of Ackley's One mathematical function.
- const char * mfunc::ackleysTwoDesc ()
- double mfunc::ackleysTwo (double *v, size_t n)

  Function 9. Implementation of Ackley's Two mathematical function.
- const char * mfunc::eggHolderDesc ()
- double mfunc::eggHolder (double *v, size_t n)

  Function 10. Implementation of the Egg Holder mathematical function.
- const char * mfunc::ranaDesc ()
- double mfunc::rana (double *v, size_t n)

  Function 11. Implementation of the Rana mathematical function.
- const char * mfunc::pathologicalDesc ()
- double mfunc::pathological (double *v, size_t n)

  Function 12. Implementation of the Pathological mathematical function.
- const char * mfunc::michalewiczDesc ()
- double mfunc::michalewicz (double *v, size_t n)

  Function 13. Implementation of the Michalewicz mathematical function.
- const char * mfunc::mastersCosineWaveDesc ()
- double mfunc::mastersCosineWave (double *v, size_t n)

  Function 14. Implementation of the Masters Cosine Wave mathematical function.
- const char * mfunc::quarticDesc ()
- double mfunc::quartic (double *v, size_t n)

  Function 15. Implementation of the Quartic mathematical function.
- const char * mfunc::levyDesc ()
- double mfunc::levy (double *v, size_t n)

  Function 16. Implementation of the Levy mathematical function.
- const char * mfunc::stepDesc ()
- double mfunc::step (double *v, size_t n)

  Function 17. Implementation of the Step mathematical function.
- const char * mfunc::alpineDesc ()
- double mfunc::alpine (double *v, size_t n)

  Function 18. Implementation of the Alpine mathematical function.
- bool mfunc::fExec (unsigned int f, double *v, size_t n, double &outResult)

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

- const char * mfunc::fDesc (unsigned int f)

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

Variables

- const unsigned int mfunc::NUM_FUNCTIONS = 18

6.7.1 Detailed Description

Contains various math function definitions.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-03-29

Copyright

Copyright (c) 2019

Definition in file mfunc.h.

6.8 mfunc.h

```
00001
00012 #ifndef __MFUNC_H
00013 #define __MFUNC_H
00014
00015 #include <cstddef> // size_t definition
00016
00020 namespace mfunc
00021 {
00022     extern const unsigned int NUM_FUNCTIONS;
00023
00024     const char* schwefelDesc();
00025     double schwefel(double* v, size_t n);
00026
00027     const char* dejongDesc();
00028     double dejong(double* v, size_t n);
00029
00030     const char* rosenbrokDesc();
00031     double rosenbrok(double* v, size_t n);
00032
00033     const char* rastriginDesc();
00034     double rastrigin(double* v, size_t n);
00035
00036     const char* griewangkDesc();
```

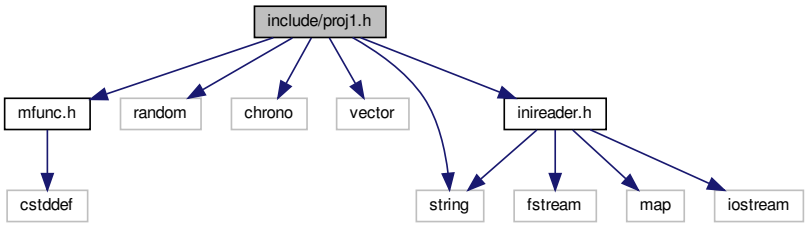
```
00037     double griewangk(double* v, size_t n);
00038
00039     const char* sineEnvelopeSineWaveDesc();
00040     double sineEnvelopeSineWave(double* v, size_t n);
00041
00042     const char* stretchedVSineWaveDesc();
00043     double stretchedVSineWave(double* v, size_t n);
00044
00045     const char* ackleysOneDesc();
00046     double ackleysOne(double* v, size_t n);
00047
00048     const char* ackleysTwoDesc();
00049     double ackleysTwo(double* v, size_t n);
00050
00051     const char* eggHolderDesc();
00052     double eggHolder(double* v, size_t n);
00053
00054     const char* ranaDesc();
00055     double rana(double* v, size_t n);
00056
00057     const char* pathologicalDesc();
00058     double pathological(double* v, size_t n);
00059
00060     const char* michalewiczDesc();
00061     double michalewicz(double* v, size_t n);
00062
00063     const char* mastersCosineWaveDesc();
00064     double mastersCosineWave(double* v, size_t n);
00065
00066     const char* quarticDesc();
00067     double quartic(double* v, size_t n);
00068
00069     const char* levyDesc();
00070     double levy(double* v, size_t n);
00071
00072     const char* stepDesc();
00073     double step(double* v, size_t n);
00074
00075     const char* alpineDesc();
00076     double alpine(double* v, size_t n);
00077
00078     bool fExec(unsigned int f, double* v, size_t n, double& outResult);
00079     const char* fDesc(unsigned int f);
00080 }
00081
00082 #endif
00083
00084 // ========================
00085 // End of mfunc.h
00086 // ========================
```

6.9 include/proj1.h File Reference

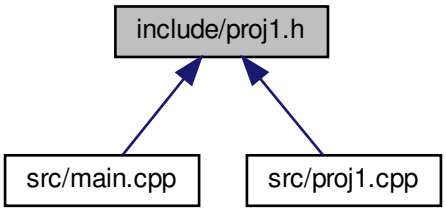
Contains the basic logic and functions to run the cs471 project 1 experiment.

```
#include <string>
#include <random>
#include <chrono>
#include <vector>
#include "mfunc.h"
#include "inireader.h"
```

Include dependency graph for proj1.h:



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



Classes

- struct proj1::RandomBounds

  Simple struct for storing the minimum and maximum input vector bounds for a function.
- class proj1::mfuncExperiment

  Contains classes for running the CS471 project 1 experiment.

Namespaces

- proj1

6.9.1 Detailed Description

Contains the basic logic and functions to run the cs471 project 1 experiment.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-04-01

Copyright

Definition in file proj1.h.

6.10 proj1.h

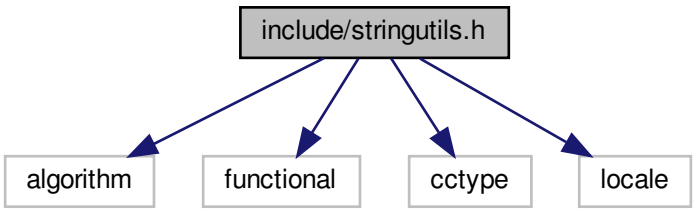
```
00001
00013 #ifndef __PROJ1_H
00014 #define __PROJ1_H
00015
00016 #include <string>
00017 #include <random>
00018 #include <chrono>
00019 #include <vector>
00020 #include "mfunc.h"
00021 #include "inireader.h"
00022
00023 namespace proj1
00024 {
00029     struct RandomBounds
00030     {
00031         double min = 0.0;
00032         double max = 0.0;
00033     };
00034
00045     class mfuncExperiment
00046     {
00047     public:
00048         mfuncExperiment();
00049         ~mfuncExperiment();
00050         bool init(const char* paramFile);
00051         int runAllFunc();
00052         int runFunc(unsigned int funcId, std::vector<double>& resultArrOut, double& timeOut);
00053     private:
00054         util::IniReader iniParams;
00055         std::string resultsFile;
00056         size_t nbrDim;
00057         size_t nbrSol;
00058         double** vMatrix;
00059         RandomBounds* vBounds;
00061         std::random_device rdev;
00062         std::mt19937 rgen;
00064         bool genFuncVectors(unsigned int funcId);
00065
00066         bool parseFuncBounds();
00067
00068         bool allocateVMatrix();
00069         void releaseVMatrix();
00070
00071         bool allocateVBounds();
00072         void releaseVBounds();
00073     };
00074 } // proj1
00075
00076 #endif
00077
00078 // ========================
00079 // End of proj1.h
00080 // ========================
```

6.11 include/stringutils.h File Reference

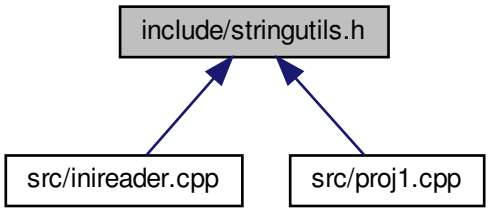
Contains various string manipulation helper functions.

```
#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

6.11.1 Detailed Description

Contains various string manipulation helper functions.

Author

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

Date

2019-04-01

Definition in file stringutils.h.

6.12 stringutils.h

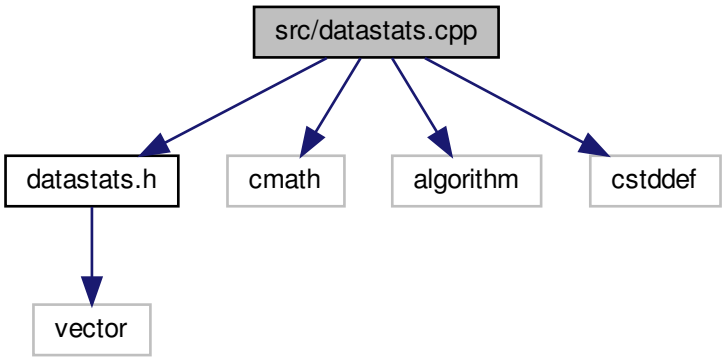
```
00001
00008 #ifndef __STRINGUTILS_H
00009 #define __STRINGUTILS_H
00010
00011 #include <algorithm>
00012 #include <functional>
00013 #include <cctype>
00014 #include <locale>
00015
00016 namespace util
00017 {
00018     // ====================================================
00019     // The string 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 // =========================
```

6.13 src/datastats.cpp File Reference

Implementation file for various data statistic functions.

```
#include "datastats.h"
#include <cmath>
#include <algorithm>
#include <cstddef>
```

Include dependency graph for datastats.cpp:



6.13.1 Detailed Description

Implementation file for various data statistic functions.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-04-01

Copyright

Copyright (c) 2019

Definition in file datastats.cpp.

6.14 datastats.cpp

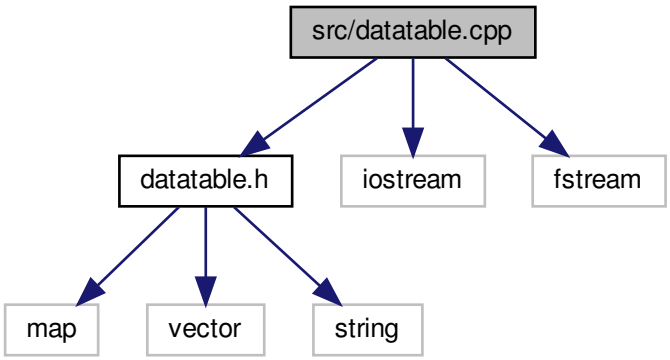
```
00001
00012 #include "datastats.h"
00013 #include <cmath>     // sqrt()
00014 #include <algorithm> // std::sort()
00015 #include <cstddef>   // size_t definition
00016
00023 double mdata::average(const std::vector<double>& v)
00024 {
00025     size_t vSize = v.size();
00026     double sum = 0.0;
00027
00028     for (size_t i = 0; i < vSize; i++)
00029         sum += v[i];
00030
00031     return sum / vSize;
00032 }
00033
00040 double mdata::standardDeviation(const std::vector<double>& v)
00041 {
00042     size_t vSize = v.size();
00043     double mean = average(v);
00044     double sum = 0;
00045
00046     for (size_t i = 0; i < vSize; i++)
00047     {
00048         double subMean = v[i] - mean;
00049         sum += subMean * subMean;
00050     }
00051
00052     return sqrt(sum / vSize);
00053 }
00054
00061 double mdata::range(const std::vector<double>& v)
00062 {
00063     size_t vSize = v.size();
00064     double min = v[0];
00065     double max = v[0];
00066
00067     for (size_t i = 1; i < vSize; i++)
00068     {
00069         double cur = v[i];
00070
00071         if (cur < min) min = cur;
00072
00073         if (cur > max) max = cur;
00074     }
00075
00076     return max - min;
00077 }
00078
00085 double mdata::median(const std::vector<double>& v)
00086 {
00087     size_t vSize = v.size();
00088     double* vSorted = new double[vSize];
00089     double retVal = 0;
00090
00091     for (int i = 0; i < vSize; i++)
00092         vSorted[i] = v[i];
00093
00094     std::sort(vSorted, vSorted + vSize);
00095
00096     if (vSize % 2 != 0)
00097     {
00098         // Odd number of values
00099         retVal = vSorted[vSize / 2];
00100     }
00101     else
00102     {
00103         // Even number of values
00104         double low = vSorted[(vSize / 2) - 1];
00105         double high = vSorted[vSize / 2];
00106         retVal = (high + low) / 2;
00107     }
00108
00109     delete[] vSorted;
00110     return retVal;
00111 }
00112
00113 // ========================
00114 // End of datastats.cpp
00115 // ========================
```

6.15 src/datatable.cpp File Reference

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

```
#include "datatable.h"
#include <iostream>
#include <fstream>
```

Include dependency graph for datatable.cpp:



6.15.1 Detailed Description

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

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-04-01

Copyright

Copyright (c) 2019

Definition in file datatable.cpp.

6.16 datatable.cpp

```
00001
00013 #include "datatable.h"
00014 #include <iostream>
00015 #include <fstream>
00016
00017 using namespace mdata;
00018
00024 DataTable::DataTable(unsigned int columns) : rows(0), cols(columns), colLabels(columns)
00025 {
00026     for (int i = 0; i < cols; i++)
00027     {
00028         colLabels.push_back("(No label)");
00029     }
00030 }
00031
00035 DataTable::~DataTable()
00036 {
00037     colLabels.clear();
00038     tableData.clear();
00039 }
00040
00048 std::string DataTable::getColLabel(unsigned int colIndex)
00049 {
00050     if (colIndex >= cols) throw "Invalid Column Index";
00051
00052     return colLabels[colIndex];
00053 }
00054
00064 bool DataTable::setColLabel(unsigned int colIndex, std::string newLabel)
00065 {
00066     if (colIndex >= cols) return false;
00067
00068     colLabels[colIndex] = newLabel;
00069     return true;
00070 }
00071
00077 unsigned int DataTable::addRow()
00078 {
00079     unsigned int newRowIndex = rows;
00080     rows++;
00081
00082     auto& tableRow = tableData[newRowIndex];
00083     tableRow.clear();
00084
00085     for (int i = 0; i < cols; i++)
00086     {
00087         tableRow.push_back("");
00088     }
00089
00090     return newRowIndex;
00091 }
00092
00100 unsigned int DataTable::addRow(const std::vector<std::string>& rowData)
00101 {
00102     unsigned int newRowIndex = rows;
00103     rows++;
00104     setRow(newRowIndex, rowData);
00105
00106     return newRowIndex;
00107 }
00108
00116 std::vector<std::string>& DataTable::getRow(unsigned int row)
00117 {
00118     if (row >= rows) throw "Invalid row index";
00119
00120     return tableData[row];
00121 }
00122
00129 void DataTable::setRow(unsigned int row, const std::vector<std::string>& rowData)
00130 {
00131     if (row >= rows) throw "Invalid row index";
00132
00133     auto& tableRow = tableData[row];
00134     tableRow.clear();
00135
00136     for (unsigned int i = 0; i < cols; i++)
00137     {
00138         if (i < rowData.size())
00139             tableRow.push_back(rowData[i]);
00140         else
00141             tableRow.push_back("(No data)");
00142     }
00143 }
00144
```

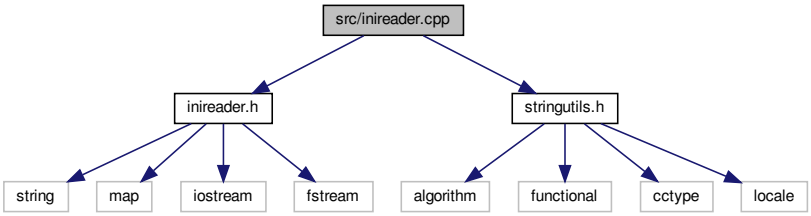
```
00154 std::string DataTable::getEntry(unsigned int row, unsigned int col)
00155 {
00156     if (row >= rows || col >= cols) throw "Invalid row/column";
00157 
00158     return tableData[row][col];
00159 }
00160 
00168 void DataTable::setEntry(unsigned int row, unsigned int col, std::string val)
00169 {
00170     if (row >= rows || col >= cols) throw "Invalid row/column";
00171 
00172     tableData[row][col] = val;
00173 }
00174 
00182 void DataTable::setEntry(unsigned int row, unsigned int col, int val)
00183 {
00184     setEntry(row, col, std::to_string(val));
00185 }
00186 
00194 void DataTable::setEntry(unsigned int row, unsigned int col, long val)
00195 {
00196     setEntry(row, col, std::to_string(val));
00197 }
00198 
00206 void DataTable::setEntry(unsigned int row, unsigned int col, float val)
00207 {
00208     setEntry(row, col, std::to_string(val));
00209 }
00210 
00218 void DataTable::setEntry(unsigned int row, unsigned int col, double val)
00219 {
00220     setEntry(row, col, std::to_string(val));
00221 }
00222 
00231 bool DataTable::exportCSV(const char* filePath)
00232 {
00233     using namespace std;
00234 
00235     ofstream outFile;
00236     outFile.open(filePath, ofstream::out | ofstream::trunc);
00237     if (!outFile.good()) return false;
00238 
00239     // Print column labels
00240     for (unsigned int c = 0; c < cols; c++)
00241     {
00242         outFile << colLabels[c];
00243         if (c < cols - 1) outFile << ",";
00244     }
00245 
00246     outFile << endl;
00247 
00248     // Print data rows
00249     for (unsigned int r = 0; r < rows; r++)
00250     {
00251         for (unsigned int c = 0; c < cols; c++)
00252         {
00253             outFile << tableData[r][c];
00254             if (c < cols - 1) outFile << ",";
00255         }
00256         outFile << endl;
00257     }
00258 
00259     outFile.close();
00260     return true;
00261 }
00262 
00263 // ========================
00264 // End of datatable.cpp
00265 // ========================
```

6.17 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:



6.17.1 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.

6.18 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
```

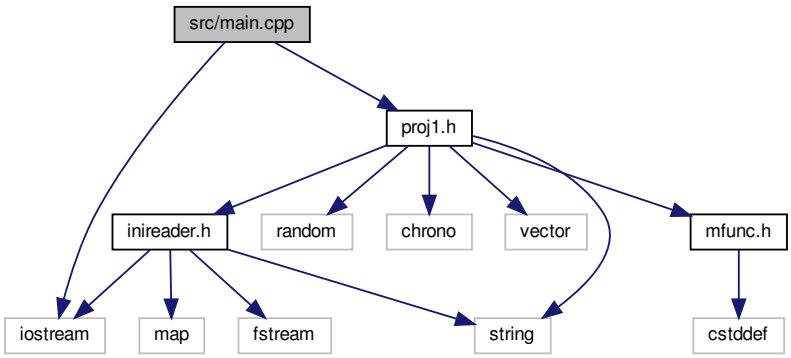
```
00163     // Remove leading and trailing whitespace
00164     s_trim(entryName);
00165     s_trim(entryValue);
00166 
00167     // We cannot have entries with empty keys
00168     if (entryName.empty()) return;
00169 
00170     // Add entry to cache
00171     iniMap[sectionName][entryName] = entryValue;
00172 }
00173 
00174 // ========================
00175 // End of inireader.cpp
00176 // ========================
```

6.19 src/main.cpp File Reference

Program entry point. Creates and runs CS471 project 1 experiment.

```
#include <iostream>
#include "proj1.h"
```

Include dependency graph for main.cpp:



Functions

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

6.19.1 Detailed Description

Program entry point. Creates and runs CS471 project 1 experiment.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-04-01

Copyright

Copyright (c) 2019

Definition in file main.cpp.

6.19.2 Function Documentation

6.19.2.1 main()

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

Definition at line 18 of file main.cpp.

References proj1::mfuncExperiment::init(), and proj1::mfuncExperiment::runAllFunc().

```
00019 {
00020    // Make sure we have enough command line args
00021    if (argc <= 1)
00022    {
00023       cout << "Error: Missing command line parameter." << endl;
00024       cout << "Proper usage: " << argv[0] << " [param file]" << endl;
00025       return EXIT_FAILURE;
00026    }
00027
00028    // Create an instance of the project 1 experiment class
00029    proj1::mfuncExperiment ex;
00030
00031    cout << "Input parameters file: " << argv[1] << endl;
00032    cout << "Initializing experiment ..." << endl;
00033
00034    // If experiment initialization fails, return failure
00035    if (!ex.init(argv[1]))
00036       return EXIT_FAILURE;
00037
00038    // Run experiment and return success code
00039    return ex.runAllFunc();
00040 }
```

6.20 main.cpp

```
00001
00013 #include <iostream>
00014 #include "proj1.h"
00015
00016 using namespace std;
00017
00018 int main(int argc, char** argv)
00019 {
00020    // Make sure we have enough command line args
00021    if (argc <= 1)
00022    {
00023       cout << "Error: Missing command line parameter." << endl;
00024       cout << "Proper usage: " << argv[0] << " [param file]" << endl;
00025       return EXIT_FAILURE;
```

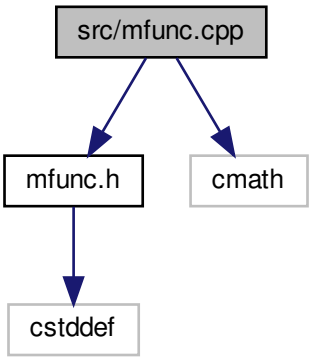
```
00026     }
00027
00028     // Create an instance of the project 1 experiment class
00029     proj1::mfuncExperiment ex;
00030
00031     cout << "Input parameters file: " << argv[1] << endl;
00032     cout << "Initializing experiment ..." << endl;
00033
00034     // If experiment initialization fails, return failure
00035     if (!ex.init(argv[1]))
00036        return EXIT_FAILURE;
00037
00038     // Run experiment and return success code
00039     return ex.runAllFunc();
00040 }
00041
00042 // ========================
00043 // End of main.cpp
00044 // ========================
```

6.21 src/mfunc.cpp File Reference

Implementations for various math functions defined in mfunc.h.

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

Include dependency graph for mfunc.cpp:



Macros

- #define _USE_MATH_DEFINES
- #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"

Functions

- double nthroot (double x, double n)
- double w (double x)

6.21.1 Detailed Description

Implementations for various math functions defined in mfunc.h.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-03-29

Copyright

Copyright (c) 2019

Definition in file mfunc.cpp.

6.21.2 Macro Definition Documentation

6.21.2.1 _ackleysOneDesc

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

Definition at line 23 of file mfunc.cpp.

Referenced by mfunc::ackleysOneDesc().

6.21.2.2 _ackleysTwoDesc

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

Definition at line 24 of file mfunc.cpp.

Referenced by mfunc::ackleysTwoDesc().

6.21.2.3 _alpineDesc

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

Definition at line 33 of file mfunc.cpp.

Referenced by mfunc::alpineDesc().

6.21.2.4 _dejongDesc

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

Definition at line 17 of file mfunc.cpp.

Referenced by mfunc::dejongDesc().

6.21.2.5 _eggHolderDesc

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

Definition at line 25 of file mfunc.cpp.

Referenced by mfunc::eggHolderDesc().

6.21.2.6 _griewangkDesc

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

Definition at line 20 of file mfunc.cpp.

Referenced by mfunc::griewangkDesc().

6.21.2.7 _levyDesc

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

Definition at line 31 of file mfunc.cpp.

Referenced by mfunc::levyDesc().

6.21.2.8 _mastersCosineWaveDesc

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

Definition at line 29 of file mfunc.cpp.

Referenced by mfunc::mastersCosineWaveDesc().

6.21.2.9 _michalewiczDesc

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

Definition at line 28 of file mfunc.cpp.

Referenced by mfunc::michalewiczDesc().

6.21.2.10 _pathologicalDesc

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

Definition at line 27 of file mfunc.cpp.

Referenced by mfunc::pathologicalDesc().

6.21.2.11 _quarticDesc

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

Definition at line 30 of file mfunc.cpp.

Referenced by mfunc::quarticDesc().

6.21.2.12 _ranaDesc

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

Definition at line 26 of file mfunc.cpp.

Referenced by mfunc::ranaDesc().

6.21.2.13 _rastriginDesc

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

Definition at line 19 of file mfunc.cpp.

Referenced by mfunc::rastriginDesc().

6.21.2.14 _rosenbrokDesc

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

Definition at line 18 of file mfunc.cpp.

Referenced by mfunc::rosenbrokDesc().

6.21.2.15 _schwefelDesc

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

Definition at line 16 of file mfunc.cpp.

Referenced by mfunc::schwefelDesc().

6.21.2.16 _sineEnvelopeSineWaveDesc

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

Definition at line 21 of file mfunc.cpp.

Referenced by mfunc::sineEnvelopeSineWaveDesc().

6.21.2.17 _stepDesc

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

Definition at line 32 of file mfunc.cpp.

Referenced by mfunc::stepDesc().

6.21.2.18 _stretchedVSineWaveDesc

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

Definition at line 22 of file mfunc.cpp.

Referenced by mfunc::stretchedVSineWaveDesc().

6.21.2.19 _USE_MATH_DEFINES

```
#define _USE_MATH_DEFINES
```

Definition at line 11 of file mfunc.cpp.

6.21.3 Function Documentation

6.21.3.1 nthroot()

```
double nthroot (
            double x,
            double n )  [inline]
```

Simple inline 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 41 of file mfunc.cpp.

Referenced by mfunc::stretchedVSineWave().

```
00042 {
00043     return pow(x, 1.0 / n);
00044 }
```

6.21.3.2 w()

```
double w (
              double x )  [inline]
```

Helper math function used in levy()

Definition at line 536 of file mfunc.cpp.

Referenced by mfunc::levy().

```
00537 {
00538     return 1.0 + (x - 1.0) / 4.0;
00539 }
```

6.22 mfunc.cpp

```
00001
00011 #define _USE_MATH_DEFINES
00012
00013 #include "mfunc.h"
00014 #include <cmath>
00015
00016 #define _schwefelDesc "Schwefel's function"
00017 #define _dejongDesc "1st De Jong's function"
00018 #define _rosenbrokDesc "Rosenbrock"
00019 #define _rastriginDesc "Rastrigin"
00020 #define _griewangkDesc "Griewangk"
00021 #define _sineEnvelopeSineWaveDesc "Sine Envelope Sine Wave"
00022 #define _stretchedVSineWaveDesc "Stretched V Sine Wave"
00023 #define _ackleysOneDesc "Ackley's One"
00024 #define _ackleysTwoDesc "Ackley's Two"
00025 #define _eggHolderDesc "Egg Holder"
00026 #define _ranaDesc "Rana"
00027 #define _pathologicalDesc "Pathological"
00028 #define _michalewiczDesc "Michalewicz"
00029 #define _mastersCosineWaveDesc "Masters Cosine Wave"
00030 #define _quarticDesc "Quartic"
00031 #define _levyDesc "Levy"
00032 #define _stepDesc "Step"
00033 #define _alpineDesc "Alpine"
00034
00041 inline double nthroot(double x, double n)
00042 {
00043     return pow(x, 1.0 / n);
00044 }
00045
00049 const unsigned int mfunc::NUM_FUNCTIONS = 18;
00050
00051 // ================================================
00052
00057 const char* mfunc::schwefelDesc()
00058 {
00059     return _schwefelDesc;
00060 }
00061
00069 double mfunc::schwefel(double* v, size_t n)
00070 {
00071     double f = 0.0;
00072
```

```
00073     for (size_t i = 0; i < n; i++)
00074     {
00075         f += (-1.0 * v[i]) * sin(sqrt(fabs(v[i])));
00076     }
00077 
00078     return (418.9829 * n) - f;
00079 }
00080 
00081 // ================================================
00082 
00087 const char* mfunc::dejongDesc()
00088 {
00089     return _dejongDesc;
00090 }
00091 
00099 double mfunc::dejong(double* v, size_t n)
00100 {
00101     double f = 0.0;
00102 
00103     for (size_t i = 0; i < n; i++)
00104     {
00105         f += v[i] * v[i];
00106     }
00107 
00108     return f;
00109 }
00110 
00111 // ================================================
00112 
00117 const char* mfunc::rosenbrokDesc()
00118 {
00119     return _rosenbrokDesc;
00120 }
00121 
00129 double mfunc::rosenbrok(double* v, size_t n)
00130 {
00131     double f = 0.0;
00132 
00133     for (size_t i = 0; i < n - 1; i++)
00134     {
00135         double a = ((v[i] * v[i]) - v[i+1]);
00136         double b = (1.0 - v[i]);
00137         f += 100.0 * a * a;
00138         f += b * b;
00139     }
00140 
00141     return f;
00142 }
00143 
00144 // ================================================
00145 
00150 const char* mfunc::rastriginDesc()
00151 {
00152     return _rastriginDesc;
00153 }
00154 
00162 double mfunc::rastrigin(double* v, size_t n)
00163 {
00164     double f = 0.0;
00165 
00166     for (size_t i = 0; i < n; i++)
00167     {
00168         f += (v[i] * v[i]) - (10.0 * cos(2.0 * M_PI * v[i]));
00169     }
00170 
00171     return 10.0 * n * f;
00172 }
00173 
00174 // ================================================
00175 
00180 const char* mfunc::griewangkDesc()
00181 {
00182     return _griewangkDesc;
00183 }
00184 
00192 double mfunc::griewangk(double* v, size_t n)
00193 {
00194     double sum = 0.0;
00195     double product = 0.0;
00196 
00197     for (size_t i = 0; i < n; i++)
00198     {
00199         sum += (v[i] * v[i]) / 4000.0;
00200     }
00201 
00202     for (size_t i = 0; i < n; i++)
00203     {
```

```
00204         product *= cos(v[i] / sqrt(i + 1.0));
00205     }
00206
00207     return 1.0 + sum - product;
00208 }
00209
00210 // ================================================
00211
00216 const char* mfunc::sineEnvelopeSineWaveDesc()
00217 {
00218     return _sineEnvelopeSineWaveDesc;
00219 }
00220
00228 double mfunc::sineEnvelopeSineWave(double* v, size_t n)
00229 {
00230     double f = 0.0;
00231
00232     for (size_t i = 0; i < n - 1; i++)
00233     {
00234         double a = sin(v[i]*v[i] + v[i+1]*v[i+1] - 0.5);
00235         a *= a;
00236         double b = (1 + 0.001*(v[i]*v[i] + v[i+1]*v[i+1]));
00237         b *= b;
00238         f += 0.5 + (a / b);
00239     }
00240
00241     return -1.0 * f;
00242 }
00243
00244 // ================================================
00245
00250 const char* mfunc::stretchedVSineWaveDesc()
00251 {
00252     return _stretchedVSineWaveDesc;
00253 }
00254
00262 double mfunc::stretchedVSineWave(double* v, size_t n)
00263 {
00264     double f = 0.0;
00265
00266     for (size_t i = 0; i < n - 1; i++)
00267     {
00268         double a = nthroot(v[i]*v[i] + v[i+1]*v[i+1], 4.0);
00269         double b = sin(50.0 * nthroot(v[i]*v[i] + v[i+1]*v[i+1], 10.0));
00270         b *= b;
00271         f += a * b + 1.0;
00272     }
00273
00274     return f;
00275 }
00276
00277 // ================================================
00278
00283 const char* mfunc::ackleysOneDesc()
00284 {
00285     return _ackleysOneDesc;
00286 }
00287
00295 double mfunc::ackleysOne(double* v, size_t n)
00296 {
00297     double f = 0.0;
00298
00299     for (size_t i = 0; i < n - 1; i++)
00300     {
00301         double a = (1.0 / pow(M_E, 0.2)) * sqrt(v[i]*v[i] + v[i+1]*v[i+1]);
00302         double b = 3.0 * (cos(2.0*v[i]) + sin(2.0*v[i+1]));
00303         f += a + b;
00304     }
00305
00306     return f;
00307 }
00308
00309 // ================================================
00310
00315 const char* mfunc::ackleysTwoDesc()
00316 {
00317     return _ackleysTwoDesc;
00318 }
00319
00327 double mfunc::ackleysTwo(double* v, size_t n)
00328 {
00329     double f = 0.0;
00330
00331     for (size_t i = 0; i < n - 1; i++)
00332     {
00333         double a = 20.0 / pow(M_E, 0.2 * sqrt((v[i]*v[i] + v[i+1]*v[i+1]) / 2.0));
00334         double b = pow(M_E, 0.5 * (cos(2.0 * M_PI * v[i]) + cos(2.0 * M_PI * v[i+1])));
```

```
00335         f += 20.0 + M_E - a - b;
00336     }
00337
00338     return f;
00339 }
00340
00341 // ================================================
00342
00347 const char* mfunc::eggHolderDesc()
00348 {
00349     return _eggHolderDesc;
00350 }
00351
00359 double mfunc::eggHolder(double* v, size_t n)
00360 {
00361     double f = 0.0;
00362
00363     for (size_t i = 0; i < n - 1; i++)
00364     {
00365         double a = -1.0 * v[i] * sin(sqrt(fabs(v[i] - v[i+1] - 47.0)));
00366         double b = (v[i+1] + 47) * sin(sqrt(fabs(v[i+1] + 47.0 + (v[i]/2.0))));
00367         f += a - b;
00368     }
00369
00370     return f;
00371 }
00372
00373 // ================================================
00374
00379 const char* mfunc::ranaDesc()
00380 {
00381     return _ranaDesc;
00382 }
00383
00391 double mfunc::rana(double* v, size_t n)
00392 {
00393     double f = 0.0;
00394
00395     for (size_t i = 0; i < n - 1; i++)
00396     {
00397         double a = v[i] * sin(sqrt(fabs(v[i+1] - v[i] + 1.0))) * cos(sqrt(fabs(v[i+1] + v[i] + 1.0)));
00398         double b = (v[i+1] + 1.0) * cos(sqrt(fabs(v[i+1] - v[i] + 1.0))) * sin(sqrt(fabs(v[i+1] + v[i] + 1.
      0)));
00399         f += a + b;
00400     }
00401
00402     return f;
00403 }
00404
00405 // ================================================
00406
00411 const char* mfunc::pathologicalDesc()
00412 {
00413     return _pathologicalDesc;
00414 }
00415
00423 double mfunc::pathological(double* v, size_t n)
00424 {
00425     double f = 0.0;
00426
00427     for (size_t i = 0; i < n - 1; i++)
00428     {
00429         double a = sin(sqrt(100.0*v[i]*v[i] + v[i+1]*v[i+1]));
00430         a = (a*a) - 0.5;
00431         double b = (v[i]*v[i] - 2*v[i]*v[i+1] + v[i+1]*v[i+1]);
00432         b = 1.0 + 0.001 * b*b;
00433         f += 0.5 + (a/b);
00434     }
00435
00436     return f;
00437 }
00438
00439 // ================================================
00440
00445 const char* mfunc::michalewiczDesc()
00446 {
00447     return _michalewiczDesc;
00448 }
00449
00457 double mfunc::michalewicz(double* v, size_t n)
00458 {
00459     double f = 0.0;
00460
00461     for (size_t i = 0; i < n; i++)
00462     {
00463         f += sin(v[i]) * pow(sin(((i+1) * v[i] * v[i]) / M_PI), 20);
00464     }
```

```
00465
00466     return -1.0 * f;
00467 }
00468
00469 // ================================================
00470
00475 const char* mfunc::mastersCosineWaveDesc()
00476 {
00477     return _mastersCosineWaveDesc;
00478 }
00479
00487 double mfunc::mastersCosineWave(double* v, size_t n)
00488 {
00489     double f = 0.0;
00490
00491     for (size_t i = 0; i < n - 1; i++)
00492     {
00493         double a = pow(M_E, (-1.0/8.0)*(v[i]*v[i] + v[i+1]*v[i+1] + 0.5*v[i+1]*v[i]));
00494         double b = cos(4 * sqrt(v[i]*v[i] + v[i+1]*v[i+1] + 0.5*v[i]*v[i+1]));
00495         f += a * b;
00496     }
00497
00498     return -1.0 * f;
00499 }
00500
00501 // ================================================
00502
00507 const char* mfunc::quarticDesc()
00508 {
00509     return _quarticDesc;
00510 }
00511
00519 double mfunc::quartic(double* v, size_t n)
00520 {
00521     double f = 0.0;
00522
00523     for (size_t i = 0; i < n; i++)
00524     {
00525         f += (i+1) * v[i] * v[i] * v[i] * v[i];
00526     }
00527
00528     return f;
00529 }
00530
00531 // ================================================
00532
00536 inline double w(double x)
00537 {
00538     return 1.0 + (x - 1.0) / 4.0;
00539 }
00540
00545 const char* mfunc::levyDesc()
00546 {
00547     return _levyDesc;
00548 }
00549
00557 double mfunc::levy(double* v, size_t n)
00558 {
00559     double f = 0.0;
00560
00561     for (size_t i = 0; i < n - 1; i++)
00562     {
00563         double a = w(v[i]) - 1.0;
00564         a *= a;
00565         double b = sin(M_PI * w(v[i]) + 1.0);
00566         b *= b;
00567         double c = w(v[n - 1]) - 1.0;
00568         c *= c;
00569         double d = sin(2.0 * M_PI * w(v[n - 1]));
00570         d *= d;
00571         f += a * (1.0 + 10.0 * b) + c * (1.0 + d);
00572     }
00573
00574     double e = sin(M_PI * w(v[0]));
00575     return e*e + f;
00576 }
00577
00578 // ================================================
00579
00584 const char* mfunc::stepDesc()
00585 {
00586     return _stepDesc;
00587 }
00588
00596 double mfunc::step(double* v, size_t n)
00597 {
00598     double f = 0.0;
```

```
00599
00600     for (size_t i = 0; i < n; i++)
00601     {
00602         double a = fabs(v[i]) + 0.5;
00603         f += a * a;
00604     }
00605
00606     return f;
00607 }
00608
00609 // ================================================
00610
00615 const char* mfunc::alpineDesc()
00616 {
00617     return _alpineDesc;
00618 }
00619
00627 double mfunc::alpine(double* v, size_t n)
00628 {
00629     double f = 0.0;
00630
00631     for (size_t i = 0; i < n; i++)
00632     {
00633         f += fabs(v[i] * sin(v[i]) + 0.1*v[i]);
00634     }
00635
00636     return f;
00637 }
00638
00639 // ================================================
00640
00651 bool mfunc::fExec(unsigned int f, double* v, size_t n, double& outResult)
00652 {
00653     switch (f)
00654     {
00655         case 1:
00656             outResult = schwefel(v, n);
00657             return true;
00658         case 2:
00659             outResult = dejong(v, n);
00660             return true;
00661         case 3:
00662             outResult = rosenbrok(v, n);
00663             return true;
00664         case 4:
00665             outResult = rastrigin(v, n);
00666             return true;
00667         case 5:
00668             outResult = griewangk(v, n);
00669             return true;
00670         case 6:
00671             outResult = sineEnvelopeSineWave(v, n);
00672             return true;
00673         case 7:
00674             outResult = stretchedVSineWave(v, n);
00675             return true;
00676         case 8:
00677             outResult = ackleysOne(v, n);
00678             return true;
00679         case 9:
00680             outResult = ackleysTwo(v, n);
00681             return true;
00682         case 10:
00683             outResult = eggHolder(v, n);
00684             return true;
00685         case 11:
00686             outResult = rana(v, n);
00687             return true;
00688         case 12:
00689             outResult = pathological(v, n);
00690             return true;
00691         case 13:
00692             outResult = michalewicz(v, n);
00693             return true;
00694         case 14:
00695             outResult = mastersCosineWave(v, n);
00696             return true;
00697         case 15:
00698             outResult = quartic(v, n);
00699             return true;
00700         case 16:
00701             outResult = levy(v, n);
00702             return true;
00703         case 17:
00704             outResult = step(v, n);
00705             return true;
00706         case 18:
```

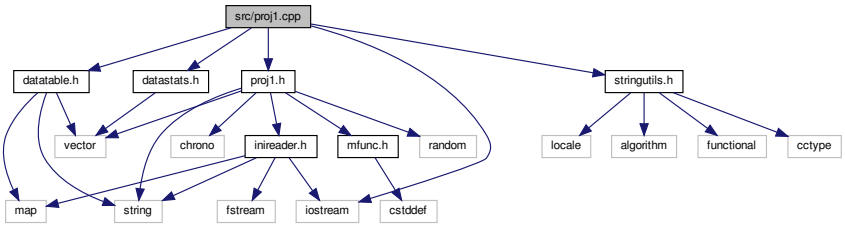
```
00707             outResult = alpine(v, n);
00708             return true;
00709         default:
00710             return false;
00711     }
00712 }
00713
00714 // ================================================
00715
00723 const char* mfunc::fDesc(unsigned int f)
00724 {
00725     switch (f)
00726     {
00727         case 1:
00728             return schwefelDesc();
00729         case 2:
00730             return dejongDesc();
00731         case 3:
00732             return rosenbrokDesc();
00733         case 4:
00734             return rastriginDesc();
00735         case 5:
00736             return griewangkDesc();
00737         case 6:
00738             return sineEnvelopeSineWaveDesc();
00739         case 7:
00740             return stretchedVSineWaveDesc();
00741         case 8:
00742             return ackleysOneDesc();
00743         case 9:
00744             return ackleysTwoDesc();
00745         case 10:
00746             return eggHolderDesc();
00747         case 11:
00748             return ranaDesc();
00749         case 12:
00750             return pathologicalDesc();
00751         case 13:
00752             return michalewiczDesc();
00753         case 14:
00754             return mastersCosineWaveDesc();
00755         case 15:
00756             return quarticDesc();
00757         case 16:
00758             return levyDesc();
00759         case 17:
00760             return stepDesc();
00761         case 18:
00762             return alpineDesc();
00763         default:
00764             return NULL;
00765     }
00766 }
00767
00768 // ========================
00769 // End of mfunc.cpp
00770 // ========================
```

6.23 src/proj1.cpp File Reference

Contains the basic logic and functions to run the cs471 project 1 experiment.

```
#include "proj1.h"
#include "datatable.h"
#include "datastats.h"
#include "stringutils.h"
#include <iostream>
```

Include dependency graph for proj1.cpp:



6.23.1 Detailed Description

Contains the basic logic and functions to run the cs471 project 1 experiment.

Author

Andrew Dunn (Andrew.Dunn@cwu.edu)

Version

0.1

Date

2019-04-01

Copyright

Copyright (c) 2019

Definition in file proj1.cpp.

6.24 proj1.cpp

```
00001
00013 #include "proj1.h"
00014 #include "datatable.h"
00015 #include "datastats.h"
00016 #include "stringutils.h"
00017 #include <iostream>
00018
00019 using namespace std;
00020 using namespace std::chrono;
00021 using namespace proj1;
00022
00026 mfuncExperiment::mfuncExperiment() : vMatrix(nullptr), vBounds(nullptr), nbrDim(0), nbrSol(0)
00027 {
00028 }
00029
00034 mfuncExperiment::~mfuncExperiment()
00035 {
00036     releaseVMatrix();
00037     releaseVBounds();
00038 }
```

```
00039
00048 bool mfuncExperiment::init(const char* paramFile)
00049 {
00050     // Open and parse parameters file
00051     if (!iniParams.openFile(paramFile))
00052     {
00053         cout << "Experiment init failed: Unable to open param file: " << paramFile << endl;
00054         return false;
00055     }
00056
00057     long numberSol;
00058     long numberDim;
00059
00060     // Attempt to parse number of solutions and vector dimensions size
00061     // from iniParams
00062     try
00063     {
00064         std::string entry;
00065
00066         entry = iniParams.getEntry("test", "number");
00067         if (entry.empty())
00068         {
00069             cout << "Experiment init failed: Param file missing [test]->number entry: " << paramFile <<
    endl;
00070             return false;
00071         }
00072
00073         numberSol = std::atol(entry.c_str());
00074
00075         entry = iniParams.getEntry("test", "dimensions");
00076         if (entry.empty())
00077         {
00078             cout << "Experiment init failed: Param file missing [test]->dimensions entry: " << paramFile <<
     endl;
00079             return false;
00080         }
00081
00082         numberDim = std::atol(entry.c_str());
00083
00084         if (numberSol <= 0)
00085         {
00086             cout << "Experiment init failed: Param file [test]->number entry out of bounds: " << paramFile
    << endl;
00087             return false;
00088         }
00089
00090         if (numberDim <= 0)
00091         {
00092             cout << "Experiment init failed: Param file [test]->dimensions entry out of bounds: " <<
    paramFile << endl;
00093             return false;
00094         }
00095     }
00096     catch (const std::exception& ex)
00097     {
00098         cout << "Experiment init failed: Exception while parsing param file: " << paramFile << endl;
00099         return false;
00100     }
00101
00102     nbrSol = (size_t)numberSol;
00103     nbrDim = (size_t)numberDim;
00104
00105     // Get csv output file path
00106     resultsFile = iniParams.getEntry("test", "output_file");
00107
00108     // Allocate memory for vector * solutions matrix
00109     if (!allocateVMatrix())
00110     {
00111         cout << "Experiment init failed: Unable to allocate vector matrix." << endl;
00112         return false;
00113     }
00114
00115     // Allocate memory for function bounds
00116     if (!allocateVBounds())
00117     {
00118         cout << "Experiment init failed: Unable to allocate vector bounds array." << endl;
00119         return false;
00120     }
00121
00122     // Fill function bounds array with data parsed from iniParams
00123     if (!parseFuncBounds())
00124     {
00125         cout << "Experiment init failed: Unable to parse vector bounds array." << endl;
00126         return false;
00127     }
00128
00129     return true;
```

```
00130 }
00131
00139 int mfuncExperiment::runAllFunc()
00140 {
00141     if (vMatrix == nullptr || nbrDim == 0 || nbrSol == 0) return 1;
00142
00143     // function desc. | average | standard dev. | range | median | time
00144     mdata::DataTable resultsTable(8);
00145     resultsTable.setColLabel(0, "Function");
00146     resultsTable.setColLabel(1, "Vector Min");
00147     resultsTable.setColLabel(2, "Vector Max");
00148     resultsTable.setColLabel(3, "Average");
00149     resultsTable.setColLabel(4, "Standard Deviation");
00150     resultsTable.setColLabel(5, "Range");
00151     resultsTable.setColLabel(6, "Median");
00152     resultsTable.setColLabel(7, "Total Time (ms)");
00153
00154     // Create a vector which is used to store all function results
00155     std::vector<double> fResults;
00156     double fTime = 0.0;
00157
00158     // Execute all functions
00159     for (unsigned int f = 1; f <= mfunc::NUM_FUNCTIONS; f++)
00160     {
00161         int err = runFunc(f, fResults, fTime);
00162         if (err)
00163             return err;
00164         else
00165         {
00166             // Insert function result and statistics into results table as a new row
00167             unsigned int rowIndex = resultsTable.addRow();
00168             resultsTable.setEntry(rowIndex, 0, mfunc::fDesc(f));
00169             resultsTable.setEntry(rowIndex, 1, to_string(vBounds[f-1].min));
00170             resultsTable.setEntry(rowIndex, 2, to_string(vBounds[f-1].max));
00171             resultsTable.setEntry(rowIndex, 3, mdata::average(fResults));
00172             resultsTable.setEntry(rowIndex, 4, mdata::standardDeviation(
      fResults));
00173             resultsTable.setEntry(rowIndex, 5, mdata::range(fResults));
00174             resultsTable.setEntry(rowIndex, 6, mdata::median(fResults));
00175             resultsTable.setEntry(rowIndex, 7, fTime);
00176         }
00177     }
00178
00179     if (!resultsFile.empty())
00180     {
00181         // Export results table to a *.csv file
00182         cout << "Exporting results to: " << resultsFile << endl;
00183         resultsTable.exportCSV(resultsFile.c_str());
00184     }
00185
00186     return 0;
00187 }
00188
00198 int mfuncExperiment::runFunc(unsigned int funcId, std::vector<double>& resultArrOut
      , double& timeOut)
00199 {
00200     if (!genFuncVectors(funcId)) return 1;
00201
00202     resultArrOut.clear();
00203     resultArrOut.reserve(nbrSol);
00204
00205     double fResult = 0;
00206
00207     high_resolution_clock::time_point t_start = high_resolution_clock::now();
00208
00209     for (int i = 0; i < nbrSol; i++)
00210     {
00211         if (!mfunc::fExec(funcId, vMatrix[i], nbrDim, fResult))
00212             return 2;
00213
00214         resultArrOut.push_back(fResult);
00215     }
00216
00217     high_resolution_clock::time_point t_end = high_resolution_clock::now();
00218     timeOut = duration_cast<nanoseconds>(t_end - t_start).count() / 1000000.0;
00219
00220     return 0;
00221 }
00222
00232 bool mfuncExperiment::genFuncVectors(unsigned int funcId)
00233 {
00234     if (vMatrix == nullptr || vBounds == nullptr || funcId == 0 || funcId >
      mfunc::NUM_FUNCTIONS) return false;
00235
00236     // Generate a new seed for the mersenne twister engine
00237     rgen = std::mt19937(rdev());
00238
```

```
00239     // Set up a uniform distribution for the random number generator with the correct function bounds
00240     std::uniform_real_distribution<> dist(vBounds[funcId - 1].min, vBounds[funcId - 1].max);
00241
00242     // Generate values for all vectors in vMatrix
00243     for (size_t s = 0; s < nbrSol; s++)
00244     {
00245         for (size_t d = 0; d < nbrDim; d++)
00246         {
00247             vMatrix[s][d] = dist(rgen);
00248         }
00249     }
00250
00251     return true;
00252 }
00253
00260 bool mfuncExperiment::parseFuncBounds()
00261 {
00262     if (vBounds == nullptr) return false;
00263
00264     const string delim = ",";
00265     const string section = "function_range";
00266     string s_min;
00267     string s_max;
00268
00269     // Extract the bounds for each function
00270     for (unsigned int i = 1; i <= mfunc::NUM_FUNCTIONS; i++)
00271     {
00272         // Get bounds entry from ini file for current function
00273         string entry = iniParams.getEntry(section, to_string(i));
00274         if (entry.empty())
00275         {
00276             cout << "Error parsing bounds for function: " << i << endl;
00277             return false;
00278         }
00279
00280         // Find index of ',' delimeter in entry string
00281         auto delimPos = entry.find(delim);
00282         if (delimPos == string::npos || delimPos >= entry.length() - 1)
00283         {
00284             cout << "Error parsing bounds for function: " << i << endl;
00285             return false;
00286         }
00287
00288         // Split string and extract min/max strings
00289         s_min = entry.substr((size_t)0, delimPos);
00290         s_max = entry.substr(delimPos + 1, entry.length());
00291         util::s_trim(s_min);
00292         util::s_trim(s_max);
00293
00294         // Attempt to parse min and max strings into double values
00295         try
00296         {
00297             RandomBounds& b = vBounds[i - 1];
00298             b.min = atof(s_min.c_str());
00299             b.max = atof(s_max.c_str());
00300         }
00301         catch(const std::exception& e)
00302         {
00303             cout << "Error parsing bounds for function: " << i << endl;
00304             std::cerr << e.what() << '\n';
00305             return false;
00306         }
00307     }
00308
00309     return true;
00310 }
00311
00318 bool mfuncExperiment::allocateVMatrix()
00319 {
00320     if (nbrSol == 0) return false;
00321
00322     try
00323     {
00324         releaseVMatrix();
00325         vMatrix = new double*[nbrSol];
00326
00327         for (size_t i = 0; i < nbrSol; i++)
00328         {
00329             vMatrix[i] = new double[nbrDim];
00330         }
00331
00332         return true;
00333     }
00334     catch(const std::exception& e)
00335     {
00336         return false;
00337     }
```

```
00338 }
00339
00343 void mfuncExperiment::releaseVMatrix()
00344 {
00345     if (vMatrix == nullptr) return;
00346
00347     for (size_t i = 0; i < nbrSol; i++)
00348     {
00349         if (vMatrix[i] != NULL)
00350         {
00351             delete[] vMatrix[i];
00352             vMatrix[i] = nullptr;
00353         }
00354     }
00355
00356     delete[] vMatrix;
00357     vMatrix = nullptr;
00358 }
00359
00367 bool mfuncExperiment::allocateVBounds()
00368 {
00369     if (nbrSol == 0) return false;
00370
00371     try
00372     {
00373         releaseVBounds();
00374
00375         vBounds = new RandomBounds[nbrSol];
00376
00377         return true;
00378     }
00379     catch(const std::exception& e)
00380     {
00381         return false;
00382     }
00383 }
00384
00388 void mfuncExperiment::releaseVBounds()
00389 {
00390     if (vBounds == nullptr) return;
00391
00392     delete[] vBounds;
00393     vBounds = nullptr;
00394 }
00395
00396 // ========================
00397 // End of proj1.cpp
00398 // ========================
```

Index