Minimum Finder

1.0

Generated by Doxygen 1.12.0

Chapter 1

Minimum Finder

Find the minimum of one multi-dimss function

The program provides a menu for the user to select one of the following algorithms:

- Steepest Descent Method
- Conjugate Gradient Method
- Simulated Annealing
- Genetic Algorithm

Users can choose to use default parameters or customize them. The program displays the results including the minimum point, function value, total iterations, and execution time.

Option 5 compares all methods using default parameters, displaying their performance side by side.

1.0.1 Features

- Implementation of four optimization algorithms
- Interactive menu for algorithm selection
- Customizable parameters
- Detailed results with performance metrics
- Comparative analysis of algorithms

1.0.2 Usage

1. Run the program.
2. Select an optimization algorithm or choose to compare all methods.
3. Opt to use default parameters or enter custom parameters.
4. View the results and performance metrics.

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

Class Documentation

4.1 DefaultParameters Struct Reference

`#include <structs.h>`

Public Attributes

- double x0 = 0.0
- double y0 = 0.0
- double alpha_sd = 0.0050
- double tol_sd = 1e-8
- int maxIter_sd = 100000
- double tol_cg = 1e-8
- int maxIter_cg = 100000
- double T0_sa = 2000.0
- double Tmin_sa = 1e-8
- double alpha_sa = 0.99
- int maxIter_sa = 200000
- int populationSize_ga = 100
- int generations_ga = 5000
- double mutationRate_ga = 0.02
- double crossoverRate_ga = 0.8

4.1.1 Detailed Description

Definition at line 29 of file structs.h.

4.1.2 Member Data Documentation

4.1.2.1 alpha_sa

```
double DefaultParameters::alpha_sa = 0.99
```

Definition at line 47 of file structs.h.

4.1.2.2 alpha_sd

```
double DefaultParameters::alpha_sd = 0.0050
```

Definition at line 36 of file structs.h.

4.1.2.3 crossoverRate_ga

```
double DefaultParameters::crossoverRate_ga = 0.8
```

Definition at line 54 of file structs.h.

4.1.2.4 generations_ga

```
int DefaultParameters::generations_ga = 5000
```

Definition at line 52 of file structs.h.

4.1.2.5 maxIter_cg

```
int DefaultParameters::maxIter_cg = 100000
```

Definition at line 42 of file structs.h.

4.1.2.6 maxIter_sa

```
int DefaultParameters::maxIter_sa = 200000
```

Definition at line 48 of file structs.h.

4.1.2.7 maxIter_sd

```
int DefaultParameters::maxIter_sd = 100000
```

Definition at line 38 of file structs.h.

4.1.2.8 mutationRate_ga

```
double DefaultParameters::mutationRate_ga = 0.02
```

Definition at line 53 of file structs.h.

4.1.2.9 populationSize_ga

```
int DefaultParameters::populationSize_ga = 100
```

Definition at line 51 of file structs.h.

4.1.2.10 T0_sa

```
double DefaultParameters::T0_sa = 2000.0
```

Definition at line 45 of file structs.h.

4.1.2.11 Tmin_sa

```
double DefaultParameters::Tmin_sa = 1e-8
```

Definition at line 46 of file structs.h.

4.1.2.12 tol_cg

```
double DefaultParameters::tol_cg = 1e-8
```

Definition at line 41 of file structs.h.

4.1.2.13 tol_sd

```
double DefaultParameters::tol_sd = 1e-8
```

Definition at line 37 of file structs.h.

4.1.2.14 x0

```
double DefaultParameters::x0 = 0.0
```

Definition at line 32 of file structs.h.

4.1.2.15 y0

```
double DefaultParameters::y0 = 0.0
```

Definition at line 33 of file structs.h.

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

- structs.h

4.2 Individual Struct Reference

```
#include <structs.h>
```

Public Member Functions

- Individual (double x_val=0, double y_val=0)

Public Attributes

- double x
- double y
- double fitness

4.2.1 Detailed Description

Definition at line 18 of file structs.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Individual()

```
Individual::Individual (
            double x_val = 0,
            double y_val = 0)  [inline]
```

Definition at line 25 of file structs.h.

```
00025 : x(x_val), y(y_val), fitness(0) {}
```

4.2.3 Member Data Documentation

4.2.3.1 fitness

```
double Individual::fitness
```

Definition at line 22 of file structs.h.

4.2.3.2 x

```
double Individual::x
```

Definition at line 20 of file structs.h.

4.2.3.3 y

```
double Individual::y
```

Definition at line 21 of file structs.h.

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

- structs.h

4.3 Result Struct Reference

`#include <structs.h>`

Public Attributes

- double x
- double y
- double f
- int iterations
- double duration

4.3.1 Detailed Description

Definition at line 58 of file structs.h.

4.3.2 Member Data Documentation

4.3.2.1 duration

`double Result::duration`

Definition at line 64 of file structs.h.

4.3.2.2 f

`double Result::f`

Definition at line 62 of file structs.h.

4.3.2.3 iterations

`int Result::iterations`

Definition at line 63 of file structs.h.

4.3.2.4 x

`double Result::x`

Definition at line 60 of file structs.h.

4.3.2.5 y

`double Result::y`

Definition at line 61 of file structs.h.

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

- structs.h

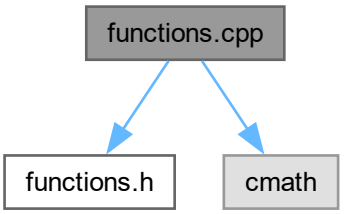
Chapter 5

File Documentation

5.1 functions.cpp File Reference

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

Include dependency graph for functions.cpp:



Functions

- double functionToMinimize (double x, double y)
- void computeGradient (double x, double y, double &dx, double &dy)
- double lineSearchBacktracking (double x, double y, double d_x, double d_y, double alpha_init, double rho, double c)

5.1.1 Function Documentation

5.1.1.1 computeGradient()

```
void computeGradient (
            double x,
            double y,
            double & dx,
            double & dy)
```

Definition at line 19 of file functions.cpp.

```
00020 {
00021     dx = cos(x + y) - sin(x + 2 * y);
00022     dy = cos(x + y) - 2 * sin(x + 2 * y);
00023 }
```

5.1.1.2 functionToMinimize()

```
double functionToMinimize (
            double x,
            double y)
```

Definition at line 13 of file functions.cpp.

```
00014 {
00015     return sin(x + y) + cos(x + 2 * y);
00016 }
```

5.1.1.3 lineSearchBacktracking()

```
double lineSearchBacktracking (
            double x,
            double y,
            double d_x,
            double d_y,
            double alpha_init,
            double rho,
            double c)
```

Definition at line 26 of file functions.cpp.

```
00027 {
00028     double alpha = alpha_init;
00029     double f0 = functionToMinimize(x, y);
00030     double grad_dot_dir = d_x * d_x + d_y * d_y; // Since d is -grad
00031     while (functionToMinimize(x + alpha * d_x, y + alpha * d_y) > f0 + c * alpha * grad_dot_dir)
00032     {
00033         alpha *= rho;
00034         if (alpha < 1e-8)
00035             break;
00036     }
00037     return alpha;
00038 }
```

5.2 functions.cpp

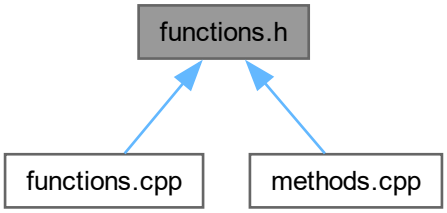
Go to the documentation of this file.

```
00001 /*
00002 @Author: Gilbert Young
00003 @Time: 2024/09/19 08:56
00004 @File_name: functions.cpp
00005 @Description:
00006 Implementation file containing the definitions of the mathematical functions used in the optimization
      algorithms.
00007 */
00008
00009 #include "functions.h"
00010 #include <cmath>
00011
00012 // Function to minimize
00013 double functionToMinimize(double x, double y)
00014 {
00015     return sin(x + y) + cos(x + 2 * y);
00016 }
00017
00018 // Compute gradient of the function
00019 void computeGradient(double x, double y, double &dx, double &dy)
00020 {
00021     dx = cos(x + y) - sin(x + 2 * y);
00022     dy = cos(x + y) - 2 * sin(x + 2 * y);
00023 }
00024
00025 // Backtracking Line Search
00026 double lineSearchBacktracking(double x, double y, double d_x, double d_y, double alpha_init, double
      rho, double c)
00027 {
```

```
00028     double alpha = alpha_init;
00029     double f0 = functionToMinimize(x, y);
00030     double grad_dot_dir = d_x * d_x + d_y * d_y; // Since d is -grad
00031     while (functionToMinimize(x + alpha * d_x, y + alpha * d_y) > f0 + c * alpha * grad_dot_dir)
00032     {
00033         alpha *= rho;
00034         if (alpha < 1e-8)
00035             break;
00036     }
00037     return alpha;
00038 }
```

5.3 functions.h File Reference

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



Functions

- double functionToMinimize (double x, double y)
- void computeGradient (double x, double y, double &dx, double &dy)
- double lineSearchBacktracking (double x, double y, double dx, double dy, double alpha_init=1.0, double rho=0.5, double c=1e-4)

5.3.1 Function Documentation

5.3.1.1 computeGradient()

```
void computeGradient (
            double x,
            double y,
            double & dx,
            double & dy)
```

Definition at line 19 of file functions.cpp.

```
00020 {
00021     dx = cos(x + y) - sin(x + 2 * y);
00022     dy = cos(x + y) - 2 * sin(x + 2 * y);
00023 }
```

5.3.1.2 functionToMinimize()

```
double functionToMinimize (
            double x,
            double y)
```

Definition at line 13 of file functions.cpp.

```
00014 {
00015     return sin(x + y) + cos(x + 2 * y);
00016 }
```

5.3.1.3 lineSearchBacktracking()

```
double lineSearchBacktracking (
            double x,
            double y,
            double dx,
            double dy,
            double alpha_init = 1.0,
            double rho = 0.5,
            double c = 1e-4)
```

Definition at line 26 of file functions.cpp.

```
00027 {
00028     double alpha = alpha_init;
00029     double f0 = functionToMinimize(x, y);
00030     double grad_dot_dir = d_x * d_x + d_y * d_y; // Since d is -grad
00031     while (functionToMinimize(x + alpha * d_x, y + alpha * d_y) > f0 + c * alpha * grad_dot_dir)
00032     {
00033         alpha *= rho;
00034         if (alpha < 1e-8)
00035             break;
00036     }
00037     return alpha;
00038 }
```

5.4 functions.h

Go to the documentation of this file.

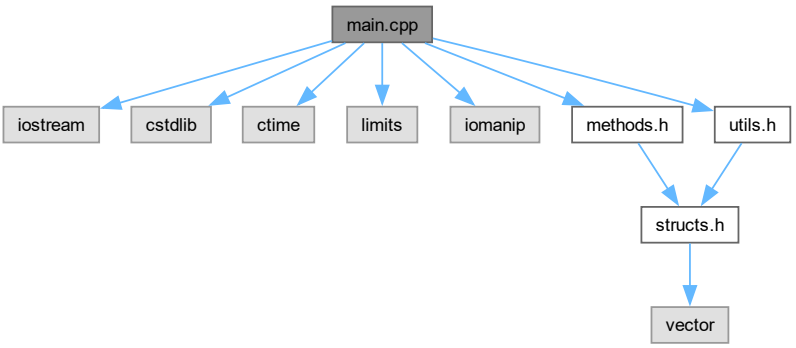
```
00001 /*
00002 @Author: Gilbert Young
00003 @Time: 2024/09/19 08:56
00004 @File_name: functions.h
00005 @Description:
00006 Header file containing function declarations for the mathematical functions used in the optimization
      algorithms:
00007 1. functionToMinimize: the function to be minimized.
00008 2. computeGradient: computes the gradient of the function.
00009 3. lineSearchBacktracking: performs a backtracking line search for the Conjugate Gradient method.
00010 */
00011
00012 #ifndef FUNCTIONS_H
00013 #define FUNCTIONS_H
00014
00015 double functionToMinimize(double x, double y);
00016 void computeGradient(double x, double y, double &dx, double &dy);
00017 double lineSearchBacktracking(double x, double y, double dx, double dy, double alpha_init = 1.0,
      double rho = 0.5, double c = 1e-4);
00018
00019 #endif // FUNCTIONS_H
```

5.5 main.cpp File Reference

Entry point for the Optimization Algorithms Solver project.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <limits>
#include <iomanip>
#include "methods.h"
#include "utils.h"
```

Include dependency graph for main.cpp:



Functions

- int main ()

5.5.1 Detailed Description

Entry point for the Optimization Algorithms Solver project.

Author

Gilbert Young

Date

2024/09/19

Definition in file main.cpp.

5.5.2 Function Documentation

5.5.2.1 main()

```
int main ()
```

Definition at line 49 of file main.cpp.

```
00050 {
00051     srand(static_cast<unsigned int>(time(0)));
00052     char choice;
00053     DefaultParameters params;
00054 
00055     do
00056     {
00057         std::cout « "\nOptimization Algorithms Menu:\n";
00058         std::cout « "1. Steepest Descent Method\n";
00059         std::cout « "2. Conjugate Gradient Method\n";
00060         std::cout « "3. Simulated Annealing\n";
00061         std::cout « "4. Genetic Algorithm\n";
00062         std::cout « "5. Compare All Methods\n";
00063         std::cout « "Enter your choice (1-5): ";
00064         int option;
00065         std::cin » option;
00066 
00067         if (option >= 1 && option <= 4)
00068         {
00069             displayDefaultParameters(params, option);
00070             std::cout « "\n1. Use default parameters\n";
00071             std::cout « "2. Customize parameters\n";
00072             std::cout « "Enter your choice (1-2): ";
00073             int subOption;
00074             std::cin » subOption;
00075 
00076             Result res;
00077             if (subOption == 1)
00078             {
00079                 // Run method with default parameters
00080                 if (option == 1)
00081                     res = steepestDescent(params.x0, params.y0, params.alpha_sd, params.maxIter_sd,
      params.tol_sd);
00082                 else if (option == 2)
00083                     res = conjugateGradient(params.x0, params.y0, params.maxIter_cg, params.tol_cg);
00084                 else if (option == 3)
00085                     res = simulatedAnnealing(params.x0, params.y0, params.T0_sa, params.Tmin_sa,
      params.alpha_sa, params.maxIter_sa);
00086                 else if (option == 4)
00087                     res = geneticAlgorithm(params.populationSize_ga, params.generations_ga,
      params.mutationRate_ga, params.crossoverRate_ga);
00088 
00089                 // Display results
00090                 std::cout « "\nResults:\n";
00091                 std::cout « "Minimum at (" « std::fixed « std::setprecision(4) « res.x « ", " « res.y
      « ")\n";
00092                 std::cout « "Minimum value: " « std::fixed « std::setprecision(4) « res.f « "\n";
00093                 std::cout « "Total iterations: " « res.iterations « "\n";
00094                 std::cout « "Execution Time: " « std::scientific « std::setprecision(3) « res.duration
      « " seconds\n";
00095             }
00096             else if (subOption == 2)
00097             {
00098                 // Customize parameters
00099                 std::cout « "Enter initial x (default " « std::fixed « std::setprecision(4) «
      params.x0 « "): ";
00100                 std::cin » params.x0;
00101                 std::cout « "Enter initial y (default " « std::fixed « std::setprecision(4) «
      params.y0 « "): ";
00102                 std::cin » params.y0;
00103 
00104                 if (option == 1)
00105                 {
00106                     std::cout « "Enter learning rate alpha (default " « std::fixed «
      std::setprecision(4) « params.alpha_sd « "): ";
00107                     std::cin » params.alpha_sd;
00108                     std::cout « "Enter maximum iterations (default " « params.maxIter_sd « "): ";
00109                     std::cin » params.maxIter_sd;
00110                     std::cout « "Enter tolerance (default " « std::scientific « std::setprecision(3) «
      params.tol_sd « "): ";
00111                     std::cin » params.tol_sd;
00112                     res = steepestDescent(params.x0, params.y0, params.alpha_sd, params.maxIter_sd,
      params.tol_sd);
00113                 }
00114                 else if (option == 2)
00115                 {
```

```
00116                     std::cout « "Enter maximum iterations (default " « params.maxIter_cg « "): ";
00117                     std::cin » params.maxIter_cg;
00118                     std::cout « "Enter tolerance (default " « std::scientific « std::setprecision(3) «
      params.tol_cg « "): ";
00119                     std::cin » params.tol_cg;
00120                     res = conjugateGradient(params.x0, params.y0, params.maxIter_cg, params.tol_cg);
00121                 }
00122                 else if (option == 3)
00123                 {
00124                     std::cout « "Enter initial temperature T0 (default " « std::fixed «
      std::setprecision(4) « params.T0_sa « "): ";
00125                     std::cin » params.T0_sa;
00126                     std::cout « "Enter minimum temperature Tmin (default " « std::scientific «
      std::setprecision(6) « params.Tmin_sa « "): ";
00127                     std::cin » params.Tmin_sa;
00128                     std::cout « "Enter cooling rate alpha (default " « std::fixed «
      std::setprecision(4) « params.alpha_sa « "): ";
00129                     std::cin » params.alpha_sa;
00130                     std::cout « "Enter maximum iterations (default " « params.maxIter_sa « "): ";
00131                     std::cin » params.maxIter_sa;
00132                     res = simulatedAnnealing(params.x0, params.y0, params.T0_sa, params.Tmin_sa,
      params.alpha_sa, params.maxIter_sa);
00133                 }
00134                 else if (option == 4)
00135                 {
00136                     std::cout « "Enter population size (default " « params.populationSize_ga « "): ";
00137                     std::cin » params.populationSize_ga;
00138                     std::cout « "Enter number of generations (default " « params.generations_ga « "):
      ";
00139                     std::cin » params.generations_ga;
00140                     std::cout « "Enter mutation rate (default " « std::fixed « std::setprecision(4) «
      params.mutationRate_ga « "): ";
00141                     std::cin » params.mutationRate_ga;
00142                     std::cout « "Enter crossover rate (default " « std::fixed « std::setprecision(4) «
      params.crossoverRate_ga « "): ";
00143                     std::cin » params.crossoverRate_ga;
00144                     res = geneticAlgorithm(params.populationSize_ga, params.generations_ga,
      params.mutationRate_ga, params.crossoverRate_ga);
00145                 }
00146
00147                 // Display results
00148                 std::cout « "\nResults:\n";
00149                 std::cout « "Minimum at (" « std::fixed « std::setprecision(4) « res.x « ", " « res.y
      « ")\n";
00150                 std::cout « "Minimum value: " « std::fixed « std::setprecision(4) « res.f « "\n";
00151                 std::cout « "Total iterations: " « res.iterations « "\n";
00152                 std::cout « "Execution Time: " « std::scientific « std::setprecision(3) « res.duration
      « " seconds\n";
00153             }
00154             else
00155             {
00156                 std::cout « "Invalid sub-option. Please select 1 or 2.\n";
00157             }
00158         }
00159         else if (option == 5)
00160         {
00161             // Compare all methods with default parameters
00162             compareMethods(params);
00163         }
00164         else
00165         {
00166             std::cout « "Invalid option. Please select 1-5.\n";
00167         }
00168
00169         // Ask user if they want to run again
00170         std::cout « "\nDo you want to run the program again? (y/n): ";
00171         std::cin » choice;
00172     } while (choice == 'y' || choice == 'Y');
00173
00174     // Wait for user input before exiting
00175     std::cout « "\nPress Enter to exit...";
00176     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'); // Clear input buffer
00177     std::cin.get();                                                     // Wait for Enter key
00178     return 0;
00179 }
```

5.6 main.cpp

Go to the documentation of this file.

```
00001
00041 #include <iostream>
```

```
00042 #include <cstdlib>
00043 #include <ctime>
00044 #include <limits>
00045 #include <iomanip>
00046 #include "methods.h"
00047 #include "utils.h"
00048
00049 int main()
00050 {
00051     srand(static_cast<unsigned int>(time(0)));
00052     char choice;
00053     DefaultParameters params;
00054
00055     do
00056     {
00057         std::cout « "\nOptimization Algorithms Menu:\n";
00058         std::cout « "1. Steepest Descent Method\n";
00059         std::cout « "2. Conjugate Gradient Method\n";
00060         std::cout « "3. Simulated Annealing\n";
00061         std::cout « "4. Genetic Algorithm\n";
00062         std::cout « "5. Compare All Methods\n";
00063         std::cout « "Enter your choice (1-5): ";
00064         int option;
00065         std::cin » option;
00066
00067         if (option >= 1 && option <= 4)
00068         {
00069             displayDefaultParameters(params, option);
00070             std::cout « "\n1. Use default parameters\n";
00071             std::cout « "2. Customize parameters\n";
00072             std::cout « "Enter your choice (1-2): ";
00073             int subOption;
00074             std::cin » subOption;
00075
00076             Result res;
00077             if (subOption == 1)
00078             {
00079                 // Run method with default parameters
00080                 if (option == 1)
00081                     res = steepestDescent(params.x0, params.y0, params.alpha_sd, params.maxIter_sd,
      params.tol_sd);
00082                 else if (option == 2)
00083                     res = conjugateGradient(params.x0, params.y0, params.maxIter_cg, params.tol_cg);
00084                 else if (option == 3)
00085                     res = simulatedAnnealing(params.x0, params.y0, params.T0_sa, params.Tmin_sa,
      params.alpha_sa, params.maxIter_sa);
00086                 else if (option == 4)
00087                     res = geneticAlgorithm(params.populationSize_ga, params.generations_ga,
      params.mutationRate_ga, params.crossoverRate_ga);
00088
00089                 // Display results
00090                 std::cout « "\nResults:\n";
00091                 std::cout « "Minimum at (" « std::fixed « std::setprecision(4) « res.x « ", " « res.y
      « ")\n";
00092                 std::cout « "Minimum value: " « std::fixed « std::setprecision(4) « res.f « "\n";
00093                 std::cout « "Total iterations: " « res.iterations « "\n";
00094                 std::cout « "Execution Time: " « std::scientific « std::setprecision(3) « res.duration
      « " seconds\n";
00095             }
00096             else if (subOption == 2)
00097             {
00098                 // Customize parameters
00099                 std::cout « "Enter initial x (default " « std::fixed « std::setprecision(4) «
      params.x0 « "): ";
00100                 std::cin » params.x0;
00101                 std::cout « "Enter initial y (default " « std::fixed « std::setprecision(4) «
      params.y0 « "): ";
00102                 std::cin » params.y0;
00103
00104                 if (option == 1)
00105                 {
00106                     std::cout « "Enter learning rate alpha (default " « std::fixed «
      std::setprecision(4) « params.alpha_sd « "): ";
00107                     std::cin » params.alpha_sd;
00108                     std::cout « "Enter maximum iterations (default " « params.maxIter_sd « "): ";
00109                     std::cin » params.maxIter_sd;
00110                     std::cout « "Enter tolerance (default " « std::scientific « std::setprecision(3) «
      params.tol_sd « "): ";
00111                     std::cin » params.tol_sd;
00112                     res = steepestDescent(params.x0, params.y0, params.alpha_sd, params.maxIter_sd,
      params.tol_sd);
00113                 }
00114                 else if (option == 2)
00115                 {
00116                     std::cout « "Enter maximum iterations (default " « params.maxIter_cg « "): ";
00117                     std::cin » params.maxIter_cg;
00118                     std::cout « "Enter tolerance (default " « std::scientific « std::setprecision(3) «
```

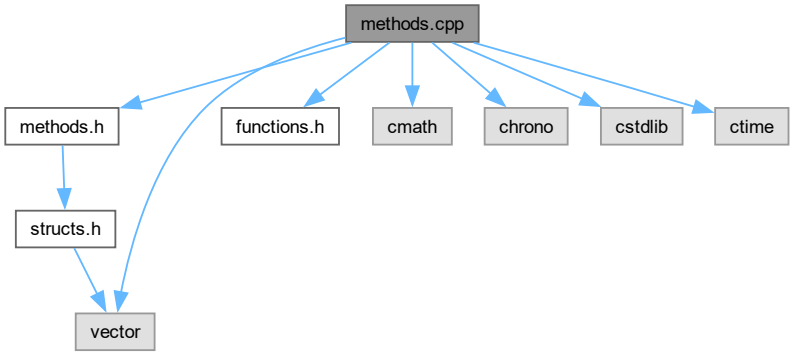
```
      params.tol_cg « "): ";
00119                     std::cin » params.tol_cg;
00120                     res = conjugateGradient(params.x0, params.y0, params.maxIter_cg, params.tol_cg);
00121                 }
00122                 else if (option == 3)
00123                 {
00124                     std::cout « "Enter initial temperature T0 (default " « std::fixed «
      std::setprecision(4) « params.T0_sa « "): ";
00125                     std::cin » params.T0_sa;
00126                     std::cout « "Enter minimum temperature Tmin (default " « std::scientific «
      std::setprecision(6) « params.Tmin_sa « "): ";
00127                     std::cin » params.Tmin_sa;
00128                     std::cout « "Enter cooling rate alpha (default " « std::fixed «
      std::setprecision(4) « params.alpha_sa « "): ";
00129                     std::cin » params.alpha_sa;
00130                     std::cout « "Enter maximum iterations (default " « params.maxIter_sa « "): ";
00131                     std::cin » params.maxIter_sa;
00132                     res = simulatedAnnealing(params.x0, params.y0, params.T0_sa, params.Tmin_sa,
      params.alpha_sa, params.maxIter_sa);
00133                 }
00134                 else if (option == 4)
00135                 {
00136                     std::cout « "Enter population size (default " « params.populationSize_ga « "): ";
00137                     std::cin » params.populationSize_ga;
00138                     std::cout « "Enter number of generations (default " « params.generations_ga « "):
      ";
00139                     std::cin » params.generations_ga;
00140                     std::cout « "Enter mutation rate (default " « std::fixed « std::setprecision(4) «
      params.mutationRate_ga « "): ";
00141                     std::cin » params.mutationRate_ga;
00142                     std::cout « "Enter crossover rate (default " « std::fixed « std::setprecision(4) «
      params.crossoverRate_ga « "): ";
00143                     std::cin » params.crossoverRate_ga;
00144                     res = geneticAlgorithm(params.populationSize_ga, params.generations_ga,
      params.mutationRate_ga, params.crossoverRate_ga);
00145                 }
00146 
00147                 // Display results
00148                 std::cout « "\nResults:\n";
00149                 std::cout « "Minimum at (" « std::fixed « std::setprecision(4) « res.x « ", " « res.y
      « ")\n";
00150                 std::cout « "Minimum value: " « std::fixed « std::setprecision(4) « res.f « "\n";
00151                 std::cout « "Total iterations: " « res.iterations « "\n";
00152                 std::cout « "Execution Time: " « std::scientific « std::setprecision(3) « res.duration
      « " seconds\n";
00153             }
00154             else
00155             {
00156                 std::cout « "Invalid sub-option. Please select 1 or 2.\n";
00157             }
00158         }
00159         else if (option == 5)
00160         {
00161             // Compare all methods with default parameters
00162             compareMethods(params);
00163         }
00164         else
00165         {
00166             std::cout « "Invalid option. Please select 1-5.\n";
00167         }
00168 
00169         // Ask user if they want to run again
00170         std::cout « "\nDo you want to run the program again? (y/n): ";
00171         std::cin » choice;
00172     } while (choice == 'y' || choice == 'Y');
00173 
00174     // Wait for user input before exiting
00175     std::cout « "\nPress Enter to exit...";
00176     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'); // Clear input buffer
00177     std::cin.get();                                                     // Wait for Enter key
00178     return 0;
00179 }
```

5.7 methods.cpp File Reference

```
#include "methods.h"
#include "functions.h"
#include <cmath>
#include <chrono>
```

```
#include <cstdlib>
#include <ctime>
#include <vector>
```

Include dependency graph for methods.cpp:



Functions

- Result steepestDescent (double x0, double y0, double alpha, int maxIter, double tol)
- Result conjugateGradient (double x0, double y0, int maxIter, double tol)
- Result simulatedAnnealing (double x0, double y0, double T0, double Tmin, double alpha, int maxIter)
- Result geneticAlgorithm (int populationSize, int generations, double mutationRate, double crossoverRate)

5.7.1 Function Documentation

5.7.1.1 conjugateGradient()

```
Result conjugateGradient (
            double x0,
            double y0,
            int maxIter,
            double tol)
```

Definition at line 53 of file methods.cpp.

```
00054 {
00055     Result res;
00056     auto start = std::chrono::high_resolution_clock::now();
00057     double x = x0, y = y0, dx, dy;
00058     computeGradient(x, y, dx, dy);
00059     double g0 = dx * dx + dy * dy;
00060     double d_x = -dx;
00061     double d_y = -dy;
00062     res.iterations = 0;
00063 
00064     for (int i = 0; i < maxIter; ++i)
00065     {
00066         // Line search to find optimal alpha
00067         double alpha = lineSearchBacktracking(x, y, d_x, d_y);
00068         // Update positions
00069         x += alpha * d_x;
00070         y += alpha * d_y;
00071         // Compute new gradient
00072         double new_dx, new_dy;
00073         computeGradient(x, y, new_dx, new_dy);
00074         double gk_new = new_dx * new_dx + new_dy * new_dy;
```

```
00075         // Check for convergence
00076         if (sqrt(gk_new) < tol)
00077         {
00078             res.iterations = i + 1;
00079             break;
00080         }
00081         // Compute beta (Fletcher-Reeves)
00082         double beta = gk_new / g0;
00083         // Update directions
00084         d_x = -new_dx + beta * d_x;
00085         d_y = -new_dy + beta * d_y;
00086         // Update gradient magnitude
00087         g0 = gk_new;
00088         res.iterations = i + 1;
00089     }
00090
00091     auto end = std::chrono::high_resolution_clock::now();
00092     res.duration = std::chrono::duration<double>(end - start).count();
00093     res.x = x;
00094     res.y = y;
00095     res.f = functionToMinimize(x, y);
00096     return res;
00097 }
```

5.7.1.2 geneticAlgorithm()

```
Result geneticAlgorithm (
            int populationSize,
            int generations,
            double mutationRate,
            double crossoverRate)
```

Definition at line 138 of file methods.cpp.

```
00139 {
00140     Result res;
00141     auto start = std::chrono::high_resolution_clock::now();
00142     std::vector<Individual> population(populationSize);
00143     double xMin = -10.0, xMax = 10.0;
00144     double yMin = -10.0, yMax = 10.0;
00145
00146     // Initialize population
00147     for (auto &ind : population)
00148     {
00149         ind.x = xMin + (xMax - xMin) * ((double)rand() / RAND_MAX);
00150         ind.y = yMin + (yMax - yMin) * ((double)rand() / RAND_MAX);
00151         ind.fitness = functionToMinimize(ind.x, ind.y);
00152     }
00153
00154     res.iterations = generations * populationSize;
00155
00156     for (int gen = 0; gen < generations; ++gen)
00157     {
00158         // Selection (Tournament Selection)
00159         std::vector<Individual> newPopulation;
00160         for (int i = 0; i < populationSize; ++i)
00161         {
00162             int a = rand() % populationSize;
00163             int b = rand() % populationSize;
00164             Individual parent = population[a].fitness < population[b].fitness ? population[a] :
      population[b];
00165             newPopulation.push_back(parent);
00166         }
00167
00168         // Crossover (Single-point)
00169         for (int i = 0; i < populationSize - 1; i += 2)
00170         {
00171             if (((double)rand() / RAND_MAX) < crossoverRate)
00172             {
00173                 double alpha = (double)rand() / RAND_MAX;
00174                 double temp_x1 = alpha * newPopulation[i].x + (1 - alpha) * newPopulation[i + 1].x;
00175                 double temp_y1 = alpha * newPopulation[i].y + (1 - alpha) * newPopulation[i + 1].y;
00176                 double temp_x2 = alpha * newPopulation[i + 1].x + (1 - alpha) * newPopulation[i].x;
00177                 double temp_y2 = alpha * newPopulation[i + 1].y + (1 - alpha) * newPopulation[i].y;
00178                 newPopulation[i].x = temp_x1;
00179                 newPopulation[i].y = temp_y1;
00180                 newPopulation[i + 1].x = temp_x2;
00181                 newPopulation[i + 1].y = temp_y2;
00182             }
00183         }
00184
```

```
00185         // Mutation
00186         for (auto &ind : newPopulation)
00187         {
00188             if (((double)rand() / RAND_MAX) < mutationRate)
00189             {
00190                 ind.x += ((double)rand() / RAND_MAX - 0.5);
00191                 ind.y += ((double)rand() / RAND_MAX - 0.5);
00192                 // Clamp to search space
00193                 if (ind.x < xMin)
00194                     ind.x = xMin;
00195                 if (ind.x > xMax)
00196                     ind.x = xMax;
00197                 if (ind.y < yMin)
00198                     ind.y = yMin;
00199                 if (ind.y > yMax)
00200                     ind.y = yMax;
00201             }
00202             ind.fitness = functionToMinimize(ind.x, ind.y);
00203         }
00204 
00205         population = newPopulation;
00206     }
00207 
00208     // Find best individual
00209     Individual best = population[0];
00210     for (const auto &ind : population)
00211     {
00212         if (ind.fitness < best.fitness)
00213             best = ind;
00214     }
00215 
00216     auto end = std::chrono::high_resolution_clock::now();
00217     res.duration = std::chrono::duration<double>(end - start).count();
00218     res.x = best.x;
00219     res.y = best.y;
00220     res.f = best.fitness;
00221     return res;
00222 }
```

5.7.1.3 simulatedAnnealing()

```
Result simulatedAnnealing (
            double x0,
            double y0,
            double T0,
            double Tmin,
            double alpha,
            int maxIter)
```

Definition at line 100 of file methods.cpp.

```
00101 {
00102     Result res;
00103     auto start = std::chrono::high_resolution_clock::now();
00104     double x = x0, y = y0, f_current = functionToMinimize(x, y);
00105     double T = T0;
00106     res.iterations = 0;
00107 
00108     for (int i = 0; i < maxIter && T > Tmin; ++i)
00109     {
00110         // Generate new candidate solution
00111         double x_new = x + ((double)rand() / RAND_MAX - 0.5);
00112         double y_new = y + ((double)rand() / RAND_MAX - 0.5);
00113         double f_new = functionToMinimize(x_new, y_new);
00114         double delta = f_new - f_current;
00115 
00116         // Accept new solution if better, or with a probability
00117         if (delta < 0 || exp(-delta / T) > ((double)rand() / RAND_MAX))
00118         {
00119             x = x_new;
00120             y = y_new;
00121             f_current = f_new;
00122         }
00123 
00124         // Cool down
00125         T *= alpha;
00126         res.iterations = i + 1;
00127     }
00128 
00129     auto end = std::chrono::high_resolution_clock::now();
```

```
00130     res.duration = std::chrono::duration<double>(end - start).count();
00131     res.x = x;
00132     res.y = y;
00133     res.f = f_current;
00134     return res;
00135 }
```

5.7.1.4 steepestDescent()

```
Result steepestDescent (
            double x0,
            double y0,
            double alpha,
            int maxIter,
            double tol)
```

Definition at line 22 of file methods.cpp.

```
00023 {
00024     Result res;
00025     auto start = std::chrono::high_resolution_clock::now();
00026     double x = x0, y = y0, dx, dy;
00027     res.iterations = 0;
00028 
00029     for (int i = 0; i < maxIter; ++i)
00030     {
00031         computeGradient(x, y, dx, dy);
00032         double norm = sqrt(dx * dx + dy * dy);
00033         if (norm < tol)
00034         {
00035             res.iterations = i;
00036             break;
00037         }
00038         // Update positions
00039         x -= alpha * dx;
00040         y -= alpha * dy;
00041         res.iterations = i + 1;
00042     }
00043 
00044     auto end = std::chrono::high_resolution_clock::now();
00045     res.duration = std::chrono::duration<double>(end - start).count();
00046     res.x = x;
00047     res.y = y;
00048     res.f = functionToMinimize(x, y);
00049     return res;
00050 }
```

5.8 methods.cpp

Go to the documentation of this file.

```
00001 /*
00002 @Author: Gilbert Young
00003 @Time: 2024/09/19 08:56
00004 @File_name: methods.cpp
00005 @Description:
00006 Implementation file containing definitions of the optimization methods:
00007 1. Steepest Descent Method
00008 2. Conjugate Gradient Method
00009 3. Simulated Annealing
00010 4. Genetic Algorithm
00011 */
00012 
00013 #include "methods.h"
00014 #include "functions.h"
00015 #include <cmath>
00016 #include <chrono>
00017 #include <cstdlib>
00018 #include <ctime>
00019 #include <vector>
00020 
00021 // Steepest Descent Method
00022 Result steepestDescent(double x0, double y0, double alpha, int maxIter, double tol)
00023 {
00024     Result res;
```

```
00025     auto start = std::chrono::high_resolution_clock::now();
00026     double x = x0, y = y0, dx, dy;
00027     res.iterations = 0;
00028 
00029     for (int i = 0; i < maxIter; ++i)
00030     {
00031         computeGradient(x, y, dx, dy);
00032         double norm = sqrt(dx * dx + dy * dy);
00033         if (norm < tol)
00034         {
00035             res.iterations = i;
00036             break;
00037         }
00038         // Update positions
00039         x -= alpha * dx;
00040         y -= alpha * dy;
00041         res.iterations = i + 1;
00042     }
00043 
00044     auto end = std::chrono::high_resolution_clock::now();
00045     res.duration = std::chrono::duration<double>(end - start).count();
00046     res.x = x;
00047     res.y = y;
00048     res.f = functionToMinimize(x, y);
00049     return res;
00050 }
00051 
00052 // Nonlinear Conjugate Gradient Method (Fletcher-Reeves)
00053 Result conjugateGradient(double x0, double y0, int maxIter, double tol)
00054 {
00055     Result res;
00056     auto start = std::chrono::high_resolution_clock::now();
00057     double x = x0, y = y0, dx, dy;
00058     computeGradient(x, y, dx, dy);
00059     double g0 = dx * dx + dy * dy;
00060     double d_x = -dx;
00061     double d_y = -dy;
00062     res.iterations = 0;
00063 
00064     for (int i = 0; i < maxIter; ++i)
00065     {
00066         // Line search to find optimal alpha
00067         double alpha = lineSearchBacktracking(x, y, d_x, d_y);
00068         // Update positions
00069         x += alpha * d_x;
00070         y += alpha * d_y;
00071         // Compute new gradient
00072         double new_dx, new_dy;
00073         computeGradient(x, y, new_dx, new_dy);
00074         double gk_new = new_dx * new_dx + new_dy * new_dy;
00075         // Check for convergence
00076         if (sqrt(gk_new) < tol)
00077         {
00078             res.iterations = i + 1;
00079             break;
00080         }
00081         // Compute beta (Fletcher-Reeves)
00082         double beta = gk_new / g0;
00083         // Update directions
00084         d_x = -new_dx + beta * d_x;
00085         d_y = -new_dy + beta * d_y;
00086         // Update gradient magnitude
00087         g0 = gk_new;
00088         res.iterations = i + 1;
00089     }
00090 
00091     auto end = std::chrono::high_resolution_clock::now();
00092     res.duration = std::chrono::duration<double>(end - start).count();
00093     res.x = x;
00094     res.y = y;
00095     res.f = functionToMinimize(x, y);
00096     return res;
00097 }
00098 
00099 // Simulated Annealing
00100 Result simulatedAnnealing(double x0, double y0, double T0, double Tmin, double alpha, int maxIter)
00101 {
00102     Result res;
00103     auto start = std::chrono::high_resolution_clock::now();
00104     double x = x0, y = y0, f_current = functionToMinimize(x, y);
00105     double T = T0;
00106     res.iterations = 0;
00107 
00108     for (int i = 0; i < maxIter && T > Tmin; ++i)
00109     {
00110         // Generate new candidate solution
00111         double x_new = x + ((double)rand() / RAND_MAX - 0.5);
```

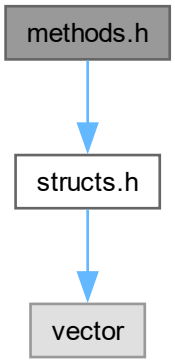
```
00112         double y_new = y + ((double)rand() / RAND_MAX - 0.5);
00113         double f_new = functionToMinimize(x_new, y_new);
00114         double delta = f_new - f_current;
00115 
00116         // Accept new solution if better, or with a probability
00117         if (delta < 0 || exp(-delta / T) > ((double)rand() / RAND_MAX))
00118         {
00119             x = x_new;
00120             y = y_new;
00121             f_current = f_new;
00122         }
00123 
00124         // Cool down
00125         T *= alpha;
00126         res.iterations = i + 1;
00127     }
00128 
00129     auto end = std::chrono::high_resolution_clock::now();
00130     res.duration = std::chrono::duration<double>(end - start).count();
00131     res.x = x;
00132     res.y = y;
00133     res.f = f_current;
00134     return res;
00135 }
00136 
00137 // Genetic Algorithm
00138 Result geneticAlgorithm(int populationSize, int generations, double mutationRate, double
    crossoverRate)
00139 {
00140     Result res;
00141     auto start = std::chrono::high_resolution_clock::now();
00142     std::vector<Individual> population(populationSize);
00143     double xMin = -10.0, xMax = 10.0;
00144     double yMin = -10.0, yMax = 10.0;
00145 
00146     // Initialize population
00147     for (auto &ind : population)
00148     {
00149         ind.x = xMin + (xMax - xMin) * ((double)rand() / RAND_MAX);
00150         ind.y = yMin + (yMax - yMin) * ((double)rand() / RAND_MAX);
00151         ind.fitness = functionToMinimize(ind.x, ind.y);
00152     }
00153 
00154     res.iterations = generations * populationSize;
00155 
00156     for (int gen = 0; gen < generations; ++gen)
00157     {
00158         // Selection (Tournament Selection)
00159         std::vector<Individual> newPopulation;
00160         for (int i = 0; i < populationSize; ++i)
00161         {
00162             int a = rand() % populationSize;
00163             int b = rand() % populationSize;
00164             Individual parent = population[a].fitness < population[b].fitness ? population[a] :
    population[b];
00165             newPopulation.push_back(parent);
00166         }
00167 
00168         // Crossover (Single-point)
00169         for (int i = 0; i < populationSize - 1; i += 2)
00170         {
00171             if (((double)rand() / RAND_MAX) < crossoverRate)
00172             {
00173                 double alpha = (double)rand() / RAND_MAX;
00174                 double temp_x1 = alpha * newPopulation[i].x + (1 - alpha) * newPopulation[i + 1].x;
00175                 double temp_y1 = alpha * newPopulation[i].y + (1 - alpha) * newPopulation[i + 1].y;
00176                 double temp_x2 = alpha * newPopulation[i + 1].x + (1 - alpha) * newPopulation[i].x;
00177                 double temp_y2 = alpha * newPopulation[i + 1].y + (1 - alpha) * newPopulation[i].y;
00178                 newPopulation[i].x = temp_x1;
00179                 newPopulation[i].y = temp_y1;
00180                 newPopulation[i + 1].x = temp_x2;
00181                 newPopulation[i + 1].y = temp_y2;
00182             }
00183         }
00184 
00185         // Mutation
00186         for (auto &ind : newPopulation)
00187         {
00188             if (((double)rand() / RAND_MAX) < mutationRate)
00189             {
00190                 ind.x += ((double)rand() / RAND_MAX - 0.5);
00191                 ind.y += ((double)rand() / RAND_MAX - 0.5);
00192                 // Clamp to search space
00193                 if (ind.x < xMin)
00194                     ind.x = xMin;
00195                 if (ind.x > xMax)
00196                     ind.x = xMax;
```

```
00197                 if (ind.y < yMin)
00198                     ind.y = yMin;
00199                 if (ind.y > yMax)
00200                     ind.y = yMax;
00201             }
00202             ind.fitness = functionToMinimize(ind.x, ind.y);
00203         }
00204
00205         population = newPopulation;
00206     }
00207
00208     // Find best individual
00209     Individual best = population[0];
00210     for (const auto &ind : population)
00211     {
00212         if (ind.fitness < best.fitness)
00213             best = ind;
00214     }
00215
00216     auto end = std::chrono::high_resolution_clock::now();
00217     res.duration = std::chrono::duration<double>(end - start).count();
00218     res.x = best.x;
00219     res.y = best.y;
00220     res.f = best.fitness;
00221     return res;
00222 }
```

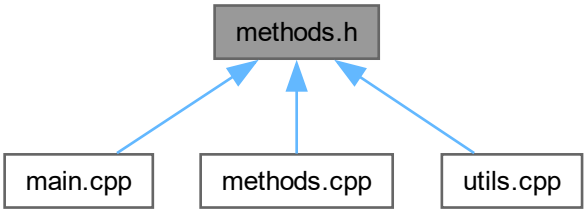
5.9 methods.h File Reference

`#include "structs.h"`

Include dependency graph for methods.h:



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



Functions

- Result steepestDescent (double x0, double y0, double alpha, int maxIter, double tol)
- Result conjugateGradient (double x0, double y0, int maxIter, double tol)
- Result simulatedAnnealing (double x0, double y0, double T0, double Tmin, double alpha, int maxIter)
- Result geneticAlgorithm (int populationSize, int generations, double mutationRate, double crossoverRate)

5.9.1 Function Documentation

5.9.1.1 conjugateGradient()

```
Result conjugateGradient (
            double x0,
            double y0,
            int maxIter,
            double tol)
```

Definition at line 53 of file methods.cpp.

```
00054 {
00055     Result res;
00056     auto start = std::chrono::high_resolution_clock::now();
00057     double x = x0, y = y0, dx, dy;
00058     computeGradient(x, y, dx, dy);
00059     double g0 = dx * dx + dy * dy;
00060     double d_x = -dx;
00061     double d_y = -dy;
00062     res.iterations = 0;
00063 
00064     for (int i = 0; i < maxIter; ++i)
00065     {
00066         // Line search to find optimal alpha
00067         double alpha = lineSearchBacktracking(x, y, d_x, d_y);
00068         // Update positions
00069         x += alpha * d_x;
00070         y += alpha * d_y;
00071         // Compute new gradient
00072         double new_dx, new_dy;
00073         computeGradient(x, y, new_dx, new_dy);
00074         double gk_new = new_dx * new_dx + new_dy * new_dy;
00075         // Check for convergence
00076         if (sqrt(gk_new) < tol)
00077         {
00078             res.iterations = i + 1;
00079             break;
00080         }
00081         // Compute beta (Fletcher-Reeves)
00082         double beta = gk_new / g0;
```

```
00083         // Update directions
00084         d_x = -new_dx + beta * d_x;
00085         d_y = -new_dy + beta * d_y;
00086         // Update gradient magnitude
00087         g0 = gk_new;
00088         res.iterations = i + 1;
00089     }
00090
00091     auto end = std::chrono::high_resolution_clock::now();
00092     res.duration = std::chrono::duration<double>(end - start).count();
00093     res.x = x;
00094     res.y = y;
00095     res.f = functionToMinimize(x, y);
00096     return res;
00097 }
```

5.9.1.2 geneticAlgorithm()

```
Result geneticAlgorithm (
            int populationSize,
            int generations,
            double mutationRate,
            double crossoverRate)
```

Definition at line 138 of file methods.cpp.

```
00139 {
00140     Result res;
00141     auto start = std::chrono::high_resolution_clock::now();
00142     std::vector<Individual> population(populationSize);
00143     double xMin = -10.0, xMax = 10.0;
00144     double yMin = -10.0, yMax = 10.0;
00145
00146     // Initialize population
00147     for (auto &ind : population)
00148     {
00149         ind.x = xMin + (xMax - xMin) * ((double)rand() / RAND_MAX);
00150         ind.y = yMin + (yMax - yMin) * ((double)rand() / RAND_MAX);
00151         ind.fitness = functionToMinimize(ind.x, ind.y);
00152     }
00153
00154     res.iterations = generations * populationSize;
00155
00156     for (int gen = 0; gen < generations; ++gen)
00157     {
00158         // Selection (Tournament Selection)
00159         std::vector<Individual> newPopulation;
00160         for (int i = 0; i < populationSize; ++i)
00161         {
00162             int a = rand() % populationSize;
00163             int b = rand() % populationSize;
00164             Individual parent = population[a].fitness < population[b].fitness ? population[a] :
    population[b];
00165             newPopulation.push_back(parent);
00166         }
00167
00168         // Crossover (Single-point)
00169         for (int i = 0; i < populationSize - 1; i += 2)
00170         {
00171             if (((double)rand() / RAND_MAX) < crossoverRate)
00172             {
00173                 double alpha = (double)rand() / RAND_MAX;
00174                 double temp_x1 = alpha * newPopulation[i].x + (1 - alpha) * newPopulation[i + 1].x;
00175                 double temp_y1 = alpha * newPopulation[i].y + (1 - alpha) * newPopulation[i + 1].y;
00176                 double temp_x2 = alpha * newPopulation[i + 1].x + (1 - alpha) * newPopulation[i].x;
00177                 double temp_y2 = alpha * newPopulation[i + 1].y + (1 - alpha) * newPopulation[i].y;
00178                 newPopulation[i].x = temp_x1;
00179                 newPopulation[i].y = temp_y1;
00180                 newPopulation[i + 1].x = temp_x2;
00181                 newPopulation[i + 1].y = temp_y2;
00182             }
00183         }
00184
00185         // Mutation
00186         for (auto &ind : newPopulation)
00187         {
00188             if (((double)rand() / RAND_MAX) < mutationRate)
00189             {
00190                 ind.x += ((double)rand() / RAND_MAX - 0.5);
00191                 ind.y += ((double)rand() / RAND_MAX - 0.5);
00192                 // Clamp to search space
```

```
00193                 if (ind.x < xMin)
00194                     ind.x = xMin;
00195                 if (ind.x > xMax)
00196                     ind.x = xMax;
00197                 if (ind.y < yMin)
00198                     ind.y = yMin;
00199                 if (ind.y > yMax)
00200                     ind.y = yMax;
00201             }
00202             ind.fitness = functionToMinimize(ind.x, ind.y);
00203         }
00204
00205         population = newPopulation;
00206     }
00207
00208     // Find best individual
00209     Individual best = population[0];
00210     for (const auto &ind : population)
00211     {
00212         if (ind.fitness < best.fitness)
00213             best = ind;
00214     }
00215
00216     auto end = std::chrono::high_resolution_clock::now();
00217     res.duration = std::chrono::duration<double>(end - start).count();
00218     res.x = best.x;
00219     res.y = best.y;
00220     res.f = best.fitness;
00221     return res;
00222 }
```

5.9.1.3 simulatedAnnealing()

```
Result simulatedAnnealing (
            double x0,
            double y0,
            double T0,
            double Tmin,
            double alpha,
            int maxIter)
```

Definition at line 100 of file methods.cpp.

```
00101 {
00102     Result res;
00103     auto start = std::chrono::high_resolution_clock::now();
00104     double x = x0, y = y0, f_current = functionToMinimize(x, y);
00105     double T = T0;
00106     res.iterations = 0;
00107
00108     for (int i = 0; i < maxIter && T > Tmin; ++i)
00109     {
00110         // Generate new candidate solution
00111         double x_new = x + ((double)rand() / RAND_MAX - 0.5);
00112         double y_new = y + ((double)rand() / RAND_MAX - 0.5);
00113         double f_new = functionToMinimize(x_new, y_new);
00114         double delta = f_new - f_current;
00115
00116         // Accept new solution if better, or with a probability
00117         if (delta < 0 || exp(-delta / T) > ((double)rand() / RAND_MAX))
00118         {
00119             x = x_new;
00120             y = y_new;
00121             f_current = f_new;
00122         }
00123
00124         // Cool down
00125         T *= alpha;
00126         res.iterations = i + 1;
00127     }
00128
00129     auto end = std::chrono::high_resolution_clock::now();
00130     res.duration = std::chrono::duration<double>(end - start).count();
00131     res.x = x;
00132     res.y = y;
00133     res.f = f_current;
00134     return res;
00135 }
```

5.9.1.4 steepestDescent()

```
Result steepestDescent (
            double x0,
            double y0,
            double alpha,
            int maxIter,
            double tol)
```

Definition at line 22 of file methods.cpp.

```
00023 {
00024     Result res;
00025     auto start = std::chrono::high_resolution_clock::now();
00026     double x = x0, y = y0, dx, dy;
00027     res.iterations = 0;
00028
00029     for (int i = 0; i < maxIter; ++i)
00030     {
00031         computeGradient(x, y, dx, dy);
00032         double norm = sqrt(dx * dx + dy * dy);
00033         if (norm < tol)
00034         {
00035             res.iterations = i;
00036             break;
00037         }
00038         // Update positions
00039         x -= alpha * dx;
00040         y -= alpha * dy;
00041         res.iterations = i + 1;
00042     }
00043
00044     auto end = std::chrono::high_resolution_clock::now();
00045     res.duration = std::chrono::duration<double>(end - start).count();
00046     res.x = x;
00047     res.y = y;
00048     res.f = functionToMinimize(x, y);
00049     return res;
00050 }
```

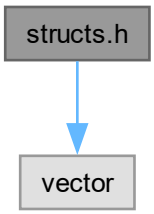
5.10 methods.h

Go to the documentation of this file.

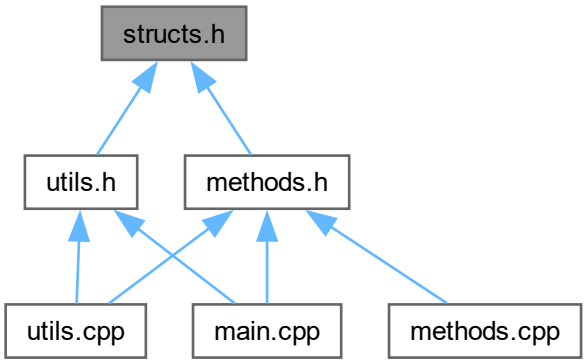
```
00001 /*
00002 @Author: Gilbert Young
00003 @Time: 2024/09/19 08:56
00004 @File_name: methods.h
00005 @Description:
00006 Header file containing declarations of the optimization methods:
00007 1. steepestDescent
00008 2. conjugateGradient
00009 3. simulatedAnnealing
00010 4. geneticAlgorithm
00011 */
00012
00013 #ifndef METHODS_H
00014 #define METHODS_H
00015
00016 #include "structs.h"
00017
00018 // Function prototypes for optimization methods
00019 Result steepestDescent(double x0, double y0, double alpha, int maxIter, double tol);
00020 Result conjugateGradient(double x0, double y0, int maxIter, double tol);
00021 Result simulatedAnnealing(double x0, double y0, double T0, double Tmin, double alpha, int maxIter);
00022 Result geneticAlgorithm(int populationSize, int generations, double mutationRate, double
      crossoverRate);
00023
00024 #endif // METHODS_H
```

5.11 structs.h File Reference

`#include <vector>`
Include dependency graph for structs.h:



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



Classes

- struct Individual
- struct DefaultParameters
- struct Result

5.12 structs.h

Go to the documentation of this file.

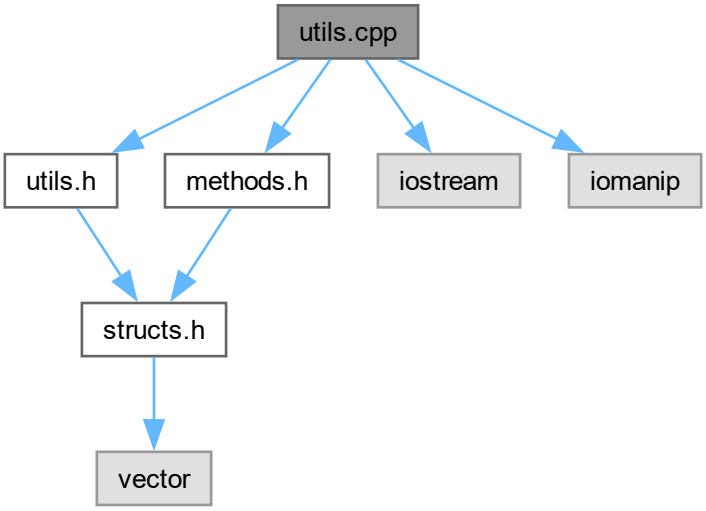
```
00001 /*
00002 @Author: Gilbert Young
00003 @Time: 2024/09/19 08:56
```

```
00004 @File_name: structs.h
00005 @Description:
00006 Header file containing the definitions of data structures used in the optimization algorithms:
00007 1. Individual: structure to store coordinates and fitness for the Genetic Algorithm.
00008 2. DefaultParameters: structure for default parameters for all algorithms.
00009 3. Result: structure to store the optimization results.
00010 */
00011
00012 #ifndef STRUCTS_H
00013 #define STRUCTS_H
00014
00015 #include <vector>
00016
00017 // Structure to store coordinates and fitness for Genetic Algorithm
00018 struct Individual
00019 {
00020     double x;
00021     double y;
00022     double fitness;
00023
00024     // Constructor for easier initialization
00025     Individual(double x_val = 0, double y_val = 0) : x(x_val), y(y_val), fitness(0) {}
00026 };
00027
00028 // Structure for default parameters
00029 struct DefaultParameters
00030 {
00031     // Initial points
00032     double x0 = 0.0;
00033     double y0 = 0.0;
00034
00035     // Steepest Descent parameters
00036     double alpha_sd = 0.0050;
00037     double tol_sd = 1e-8;
00038     int maxIter_sd = 100000;
00039
00040     // Conjugate Gradient parameters
00041     double tol_cg = 1e-8;
00042     int maxIter_cg = 100000;
00043
00044     // Simulated Annealing parameters
00045     double T0_sa = 2000.0;
00046     double Tmin_sa = 1e-8;
00047     double alpha_sa = 0.99;
00048     int maxIter_sa = 200000;
00049
00050     // Genetic Algorithm parameters
00051     int populationSize_ga = 100;
00052     int generations_ga = 5000;
00053     double mutationRate_ga = 0.02;
00054     double crossoverRate_ga = 0.8;
00055 };
00056
00057 // Structure to store optimization results
00058 struct Result
00059 {
00060     double x;
00061     double y;
00062     double f;
00063     int iterations;
00064     double duration; // in seconds
00065 };
00066
00067 #endif // STRUCTS_H
```

5.13 utils.cpp File Reference

```
#include "utils.h"
#include "methods.h"
#include <iostream>
#include <iomanip>
```

Include dependency graph for utils.cpp:



Functions

- void displayDefaultParameters (const DefaultParameters &params, int option)
- void compareMethods (const DefaultParameters &params)

5.13.1 Function Documentation

5.13.1.1 compareMethods()

```
void compareMethods (
            const DefaultParameters & params)
```

Definition at line 56 of file utils.cpp.

```
00057 {
00058     std::cout « "\nComparing All Methods with Default Parameters...\n";
00059
00060     // Display default parameters
00061     std::cout « "\nDefault Parameters:\n";
00062     std::cout « "Initial x: " « std::fixed « std::setprecision(4) « params.x0
00063               « ", Initial y: " « params.y0 « "\n";
00064     std::cout « "Steepest Descent alpha: " « params.alpha_sd
00065               « ", maxIter: " « params.maxIter_sd
00066               « ", tol: " « std::scientific « std::setprecision(3) « params.tol_sd « std::fixed «
      "\n";
00067     std::cout « "Conjugate Gradient maxIter: " « params.maxIter_cg
00068               « ", tol: " « std::scientific « std::setprecision(3) « params.tol_cg « std::fixed «
      "\n";
00069     std::cout « "Simulated Annealing T0: " « params.T0_sa
00070               « ", Tmin: " « std::scientific « std::setprecision(3) « params.Tmin_sa
00071               « std::fixed « ", alpha: " « params.alpha_sa
00072               « ", maxIter: " « params.maxIter_sa « "\n";
00073     std::cout « "Genetic Algorithm populationSize: " « params.populationSize_ga
00074               « ", generations: " « params.generations_ga
00075               « ", mutationRate: " « std::fixed « std::setprecision(4) « params.mutationRate_ga
00076               « ", crossoverRate: " « params.crossoverRate_ga « "\n";
00077
```

```
00078     // Run all methods
00079     Result res_sd = steepestDescent(params.x0, params.y0, params.alpha_sd, params.maxIter_sd,
      params.tol_sd);
00080     Result res_cg = conjugateGradient(params.x0, params.y0, params.maxIter_cg, params.tol_cg);
00081     Result res_sa = simulatedAnnealing(params.x0, params.y0, params.T0_sa, params.Tmin_sa,
      params.alpha_sa, params.maxIter_sa);
00082     Result res_ga = geneticAlgorithm(params.populationSize_ga, params.generations_ga,
      params.mutationRate_ga, params.crossoverRate_ga);
00083
00084     // Display results
00085     std::cout « "\nResults:\n";
00086
00087     // Steepest Descent
00088     std::cout « "Steepest Descent Method:\n";
00089     std::cout « "Minimum at (" « std::fixed « std::setprecision(5) « res_sd.x « ", " « res_sd.y «
      ")\n";
00090     std::cout « "Minimum value: " « std::fixed « std::setprecision(5) « res_sd.f « "\n";
00091     std::cout « "Total iterations: " « res_sd.iterations « "\n";
00092     std::cout « "Execution Time: " « std::scientific « std::setprecision(3) « res_sd.duration « "
      seconds\n\n";
00093
00094     // Conjugate Gradient
00095     std::cout « "Conjugate Gradient Method:\n";
00096     std::cout « "Minimum at (" « std::fixed « std::setprecision(5) « res_cg.x « ", " « res_cg.y «
      ")\n";
00097     std::cout « "Minimum value: " « std::fixed « std::setprecision(5) « res_cg.f « "\n";
00098     std::cout « "Total iterations: " « res_cg.iterations « "\n";
00099     std::cout « "Execution Time: " « std::scientific « std::setprecision(3) « res_cg.duration « "
      seconds\n\n";
00100
00101     // Simulated Annealing
00102     std::cout « "Simulated Annealing:\n";
00103     std::cout « "Minimum at (" « std::fixed « std::setprecision(5) « res_sa.x « ", " « res_sa.y «
      ")\n";
00104     std::cout « "Minimum value: " « std::fixed « std::setprecision(5) « res_sa.f « "\n";
00105     std::cout « "Total iterations: " « res_sa.iterations « "\n";
00106     std::cout « "Execution Time: " « std::scientific « std::setprecision(3) « res_sa.duration « "
      seconds\n\n";
00107
00108     // Genetic Algorithm
00109     std::cout « "Genetic Algorithm:\n";
00110     std::cout « "Minimum at (" « std::fixed « std::setprecision(5) « res_ga.x « ", " « res_ga.y «
      ")\n";
00111     std::cout « "Minimum value: " « std::fixed « std::setprecision(5) « res_ga.f « "\n";
00112     std::cout « "Total iterations: " « res_ga.iterations « "\n";
00113     std::cout « "Execution Time: " « std::scientific « std::setprecision(3) « res_ga.duration « "
      seconds\n";
00114 }
```

5.13.1.2 displayDefaultParameters()

```
void displayDefaultParameters (
            const DefaultParameters & params,
            int option)
```

Definition at line 16 of file utils.cpp.

```
00017 {
00018     if (option == 1)
00019     {
00020         std::cout « "\nCurrent Default Parameters for Steepest Descent:\n";
00021         std::cout « "Initial x: " « std::fixed « std::setprecision(4) « params.x0
00022                   « ", Initial y: " « params.y0 « "\n";
00023         std::cout « "Learning rate alpha: " « params.alpha_sd
00024                   « "\nMaximum iterations: " « params.maxIter_sd
00025                   « "\nTolerance: " « std::scientific « std::setprecision(3) « params.tol_sd «
      std::fixed « "\n";
00026     }
00027     else if (option == 2)
00028     {
00029         std::cout « "\nCurrent Default Parameters for Conjugate Gradient:\n";
00030         std::cout « "Initial x: " « std::fixed « std::setprecision(4) « params.x0
00031                   « ", Initial y: " « params.y0 « "\n";
00032         std::cout « "Maximum iterations: " « params.maxIter_cg
00033                   « "\nTolerance: " « std::scientific « std::setprecision(3) « params.tol_cg «
      std::fixed « "\n";
00034     }
00035     else if (option == 3)
00036     {
00037         std::cout « "\nCurrent Default Parameters for Simulated Annealing:\n";
00038         std::cout « "Initial x: " « std::fixed « std::setprecision(4) « params.x0
00039                   « ", Initial y: " « params.y0 « "\n";
```

```
00040         std::cout « "Initial temperature T0: " « params.T0_sa
00041                   « "\nMinimum temperature Tmin: " « std::scientific « std::setprecision(3) «
      params.Tmin_sa « std::fixed
00042                   « "\nCooling rate alpha: " « params.alpha_sa
00043                   « "\nMaximum iterations: " « params.maxIter_sa « "\n";
00044     }
00045     else if (option == 4)
00046     {
00047         std::cout « "\nCurrent Default Parameters for Genetic Algorithm:\n";
00048         std::cout « "Population size: " « params.populationSize_ga
00049                   « "\nGenerations: " « params.generations_ga
00050                   « "\nMutation rate: " « std::fixed « std::setprecision(4) « params.mutationRate_ga
00051                   « "\nCrossover rate: " « params.crossoverRate_ga « "\n";
00052     }
00053 }
```

5.14 utils.cpp

Go to the documentation of this file.

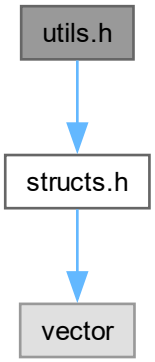
```
00001 /*
00002 @Author: Gilbert Young
00003 @Time: 2024/09/19 08:56
00004 @File_name: utils.cpp
00005 @Description:
00006 Implementation file containing utility functions:
00007 1. displayDefaultParameters: displays the default parameters for the selected algorithm.
00008 2. compareMethods: compares all optimization methods using default parameters.
00009 */
00010
00011 #include "utils.h"
00012 #include "methods.h"
00013 #include <iostream>
00014 #include <iomanip>
00015
00016 void displayDefaultParameters(const DefaultParameters &params, int option)
00017 {
00018     if (option == 1)
00019     {
00020         std::cout « "\nCurrent Default Parameters for Steepest Descent:\n";
00021         std::cout « "Initial x: " « std::fixed « std::setprecision(4) « params.x0
00022                   « ", Initial y: " « params.y0 « "\n";
00023         std::cout « "Learning rate alpha: " « params.alpha_sd
00024                   « "\nMaximum iterations: " « params.maxIter_sd
00025                   « "\nTolerance: " « std::scientific « std::setprecision(3) « params.tol_sd «
      std::fixed « "\n";
00026     }
00027     else if (option == 2)
00028     {
00029         std::cout « "\nCurrent Default Parameters for Conjugate Gradient:\n";
00030         std::cout « "Initial x: " « std::fixed « std::setprecision(4) « params.x0
00031                   « ", Initial y: " « params.y0 « "\n";
00032         std::cout « "Maximum iterations: " « params.maxIter_cg
00033                   « "\nTolerance: " « std::scientific « std::setprecision(3) « params.tol_cg «
      std::fixed « "\n";
00034     }
00035     else if (option == 3)
00036     {
00037         std::cout « "\nCurrent Default Parameters for Simulated Annealing:\n";
00038         std::cout « "Initial x: " « std::fixed « std::setprecision(4) « params.x0
00039                   « ", Initial y: " « params.y0 « "\n";
00040         std::cout « "Initial temperature T0: " « params.T0_sa
00041                   « "\nMinimum temperature Tmin: " « std::scientific « std::setprecision(3) «
      params.Tmin_sa « std::fixed
00042                   « "\nCooling rate alpha: " « params.alpha_sa
00043                   « "\nMaximum iterations: " « params.maxIter_sa « "\n";
00044     }
00045     else if (option == 4)
00046     {
00047         std::cout « "\nCurrent Default Parameters for Genetic Algorithm:\n";
00048         std::cout « "Population size: " « params.populationSize_ga
00049                   « "\nGenerations: " « params.generations_ga
00050                   « "\nMutation rate: " « std::fixed « std::setprecision(4) « params.mutationRate_ga
00051                   « "\nCrossover rate: " « params.crossoverRate_ga « "\n";
00052     }
00053 }
00054
00055 // Compare all methods using default parameters
00056 void compareMethods(const DefaultParameters &params)
00057 {
00058     std::cout « "\nComparing All Methods with Default Parameters...\n";
00059
```

```
00060     // Display default parameters
00061     std::cout « "\nDefault Parameters:\n";
00062     std::cout « "Initial x: " « std::fixed « std::setprecision(4) « params.x0
00063               « ", Initial y: " « params.y0 « "\n";
00064     std::cout « "Steepest Descent alpha: " « params.alpha_sd
00065               « ", maxIter: " « params.maxIter_sd
00066               « ", tol: " « std::scientific « std::setprecision(3) « params.tol_sd « std::fixed «
      "\n";
00067     std::cout « "Conjugate Gradient maxIter: " « params.maxIter_cg
00068               « ", tol: " « std::scientific « std::setprecision(3) « params.tol_cg « std::fixed «
      "\n";
00069     std::cout « "Simulated Annealing T0: " « params.T0_sa
00070               « ", Tmin: " « std::scientific « std::setprecision(3) « params.Tmin_sa
00071               « std::fixed « ", alpha: " « params.alpha_sa
00072               « ", maxIter: " « params.maxIter_sa « "\n";
00073     std::cout « "Genetic Algorithm populationSize: " « params.populationSize_ga
00074               « ", generations: " « params.generations_ga
00075               « ", mutationRate: " « std::fixed « std::setprecision(4) « params.mutationRate_ga
00076               « ", crossoverRate: " « params.crossoverRate_ga « "\n";
00077
00078     // Run all methods
00079     Result res_sd = steepestDescent(params.x0, params.y0, params.alpha_sd, params.maxIter_sd,
      params.tol_sd);
00080     Result res_cg = conjugateGradient(params.x0, params.y0, params.maxIter_cg, params.tol_cg);
00081     Result res_sa = simulatedAnnealing(params.x0, params.y0, params.T0_sa, params.Tmin_sa,
      params.alpha_sa, params.maxIter_sa);
00082     Result res_ga = geneticAlgorithm(params.populationSize_ga, params.generations_ga,
      params.mutationRate_ga, params.crossoverRate_ga);
00083
00084     // Display results
00085     std::cout « "\nResults:\n";
00086
00087     // Steepest Descent
00088     std::cout « "Steepest Descent Method:\n";
00089     std::cout « "Minimum at (" « std::fixed « std::setprecision(5) « res_sd.x « ", " « res_sd.y «
      ")\n";
00090     std::cout « "Minimum value: " « std::fixed « std::setprecision(5) « res_sd.f « "\n";
00091     std::cout « "Total iterations: " « res_sd.iterations « "\n";
00092     std::cout « "Execution Time: " « std::scientific « std::setprecision(3) « res_sd.duration « "
      seconds\n\n";
00093
00094     // Conjugate Gradient
00095     std::cout « "Conjugate Gradient Method:\n";
00096     std::cout « "Minimum at (" « std::fixed « std::setprecision(5) « res_cg.x « ", " « res_cg.y «
      ")\n";
00097     std::cout « "Minimum value: " « std::fixed « std::setprecision(5) « res_cg.f « "\n";
00098     std::cout « "Total iterations: " « res_cg.iterations « "\n";
00099     std::cout « "Execution Time: " « std::scientific « std::setprecision(3) « res_cg.duration « "
      seconds\n\n";
00100
00101     // Simulated Annealing
00102     std::cout « "Simulated Annealing:\n";
00103     std::cout « "Minimum at (" « std::fixed « std::setprecision(5) « res_sa.x « ", " « res_sa.y «
      ")\n";
00104     std::cout « "Minimum value: " « std::fixed « std::setprecision(5) « res_sa.f « "\n";
00105     std::cout « "Total iterations: " « res_sa.iterations « "\n";
00106     std::cout « "Execution Time: " « std::scientific « std::setprecision(3) « res_sa.duration « "
      seconds\n\n";
00107
00108     // Genetic Algorithm
00109     std::cout « "Genetic Algorithm:\n";
00110     std::cout « "Minimum at (" « std::fixed « std::setprecision(5) « res_ga.x « ", " « res_ga.y «
      ")\n";
00111     std::cout « "Minimum value: " « std::fixed « std::setprecision(5) « res_ga.f « "\n";
00112     std::cout « "Total iterations: " « res_ga.iterations « "\n";
00113     std::cout « "Execution Time: " « std::scientific « std::setprecision(3) « res_ga.duration « "
      seconds\n";
00114 }
```

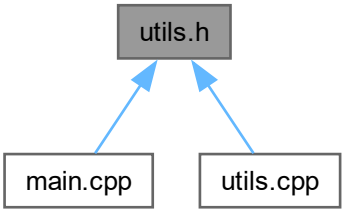
5.15 utils.h File Reference

```
#include "structs.h"
```

Include dependency graph for utils.h:



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



Functions

- void displayDefaultParameters (const DefaultParameters &params, int option)
- void compareMethods (const DefaultParameters &params)

5.15.1 Function Documentation

5.15.1.1 compareMethods()

```
void compareMethods (
            const DefaultParameters & params)
```

Definition at line 56 of file utils.cpp.

```
00057 {
00058     std::cout « "\nComparing All Methods with Default Parameters...\n";
00059
00060     // Display default parameters
00061     std::cout « "\nDefault Parameters:\n";
00062     std::cout « "Initial x: " « std::fixed « std::setprecision(4) « params.x0
00063               « ", Initial y: " « params.y0 « "\n";
00064     std::cout « "Steepest Descent alpha: " « params.alpha_sd
00065               « ", maxIter: " « params.maxIter_sd
00066               « ", tol: " « std::scientific « std::setprecision(3) « params.tol_sd « std::fixed «
      "\n";
00067     std::cout « "Conjugate Gradient maxIter: " « params.maxIter_cg
00068               « ", tol: " « std::scientific « std::setprecision(3) « params.tol_cg « std::fixed «
      "\n";
00069     std::cout « "Simulated Annealing T0: " « params.T0_sa
00070               « ", Tmin: " « std::scientific « std::setprecision(3) « params.Tmin_sa
00071               « std::fixed « ", alpha: " « params.alpha_sa
00072               « ", maxIter: " « params.maxIter_sa « "\n";
00073     std::cout « "Genetic Algorithm populationSize: " « params.populationSize_ga
00074               « ", generations: " « params.generations_ga
00075               « ", mutationRate: " « std::fixed « std::setprecision(4) « params.mutationRate_ga
00076               « ", crossoverRate: " « params.crossoverRate_ga « "\n";
00077
00078     // Run all methods
00079     Result res_sd = steepestDescent(params.x0, params.y0, params.alpha_sd, params.maxIter_sd,
      params.tol_sd);
00080     Result res_cg = conjugateGradient(params.x0, params.y0, params.maxIter_cg, params.tol_cg);
00081     Result res_sa = simulatedAnnealing(params.x0, params.y0, params.T0_sa, params.Tmin_sa,
      params.alpha_sa, params.maxIter_sa);
00082     Result res_ga = geneticAlgorithm(params.populationSize_ga, params.generations_ga,
      params.mutationRate_ga, params.crossoverRate_ga);
00083
00084     // Display results
00085     std::cout « "\nResults:\n";
00086
00087     // Steepest Descent
00088     std::cout « "Steepest Descent Method:\n";
00089     std::cout « "Minimum at (" « std::fixed « std::setprecision(5) « res_sd.x « ", " « res_sd.y «
      ")\n";
00090     std::cout « "Minimum value: " « std::fixed « std::setprecision(5) « res_sd.f « "\n";
00091     std::cout « "Total iterations: " « res_sd.iterations « "\n";
00092     std::cout « "Execution Time: " « std::scientific « std::setprecision(3) « res_sd.duration « "
      seconds\n\n";
00093
00094     // Conjugate Gradient
00095     std::cout « "Conjugate Gradient Method:\n";
00096     std::cout « "Minimum at (" « std::fixed « std::setprecision(5) « res_cg.x « ", " « res_cg.y «
      ")\n";
00097     std::cout « "Minimum value: " « std::fixed « std::setprecision(5) « res_cg.f « "\n";
00098     std::cout « "Total iterations: " « res_cg.iterations « "\n";
00099     std::cout « "Execution Time: " « std::scientific « std::setprecision(3) « res_cg.duration « "
      seconds\n\n";
00100
00101     // Simulated Annealing
00102     std::cout « "Simulated Annealing:\n";
00103     std::cout « "Minimum at (" « std::fixed « std::setprecision(5) « res_sa.x « ", " « res_sa.y «
      ")\n";
00104     std::cout « "Minimum value: " « std::fixed « std::setprecision(5) « res_sa.f « "\n";
00105     std::cout « "Total iterations: " « res_sa.iterations « "\n";
00106     std::cout « "Execution Time: " « std::scientific « std::setprecision(3) « res_sa.duration « "
      seconds\n\n";
00107
00108     // Genetic Algorithm
00109     std::cout « "Genetic Algorithm:\n";
00110     std::cout « "Minimum at (" « std::fixed « std::setprecision(5) « res_ga.x « ", " « res_ga.y «
      ")\n";
00111     std::cout « "Minimum value: " « std::fixed « std::setprecision(5) « res_ga.f « "\n";
00112     std::cout « "Total iterations: " « res_ga.iterations « "\n";
00113     std::cout « "Execution Time: " « std::scientific « std::setprecision(3) « res_ga.duration « "
      seconds\n";
00114 }
```

5.15.1.2 displayDefaultParameters()

```
void displayDefaultParameters (
            const DefaultParameters & params,
            int option)
```

Definition at line 16 of file utils.cpp.

```
00017 {
```

```
00018     if (option == 1)
00019     {
00020         std::cout « "\nCurrent Default Parameters for Steepest Descent:\n";
00021         std::cout « "Initial x: " « std::fixed « std::setprecision(4) « params.x0
00022                   « ", Initial y: " « params.y0 « "\n";
00023         std::cout « "Learning rate alpha: " « params.alpha_sd
00024                   « "\nMaximum iterations: " « params.maxIter_sd
00025                   « "\nTolerance: " « std::scientific « std::setprecision(3) « params.tol_sd «
      std::fixed « "\n";
00026     }
00027     else if (option == 2)
00028     {
00029         std::cout « "\nCurrent Default Parameters for Conjugate Gradient:\n";
00030         std::cout « "Initial x: " « std::fixed « std::setprecision(4) « params.x0
00031                   « ", Initial y: " « params.y0 « "\n";
00032         std::cout « "Maximum iterations: " « params.maxIter_cg
00033                   « "\nTolerance: " « std::scientific « std::setprecision(3) « params.tol_cg «
      std::fixed « "\n";
00034     }
00035     else if (option == 3)
00036     {
00037         std::cout « "\nCurrent Default Parameters for Simulated Annealing:\n";
00038         std::cout « "Initial x: " « std::fixed « std::setprecision(4) « params.x0
00039                   « ", Initial y: " « params.y0 « "\n";
00040         std::cout « "Initial temperature T0: " « params.T0_sa
00041                   « "\nMinimum temperature Tmin: " « std::scientific « std::setprecision(3) «
      params.Tmin_sa « std::fixed
00042                   « "\nCooling rate alpha: " « params.alpha_sa
00043                   « "\nMaximum iterations: " « params.maxIter_sa « "\n";
00044     }
00045     else if (option == 4)
00046     {
00047         std::cout « "\nCurrent Default Parameters for Genetic Algorithm:\n";
00048         std::cout « "Population size: " « params.populationSize_ga
00049                   « "\nGenerations: " « params.generations_ga
00050                   « "\nMutation rate: " « std::fixed « std::setprecision(4) « params.mutationRate_ga
00051                   « "\nCrossover rate: " « params.crossoverRate_ga « "\n";
00052     }
00053 }
```

5.16 utils.h

Go to the documentation of this file.

```
00001 /*
00002 @Author: Gilbert Young
00003 @Time: 2024/09/19 08:56
00004 @File_name: utils.h
00005 @Description:
00006 Header file containing utility functions:
00007 1. displayDefaultParameters: displays the default parameters for the selected algorithm.
00008 2. compareMethods: compares all optimization methods using default parameters.
00009 */
00010
00011 #ifndef UTILS_H
00012 #define UTILS_H
00013
00014 #include "structs.h"
00015
00016 void displayDefaultParameters(const DefaultParameters &params, int option);
00017 void compareMethods(const DefaultParameters &params);
00018
00019 #endif // UTILS_H
```

Index