**Методические указания по работе с фреймворком и бенчмарком глобальной оптимизации**

## **Архитектура фреймворка**

Разработанный фреймворк устроен следующим образом: для работы используются конфигурационные файлы, наборы стартовых точек, библиотека методов и библиотека тестовых функций. Всё это суммарно поступает на обработку, в результате чего мы получаем отчет с необходимыми нам данными и первичной обработкой (см. Рисунок 1).

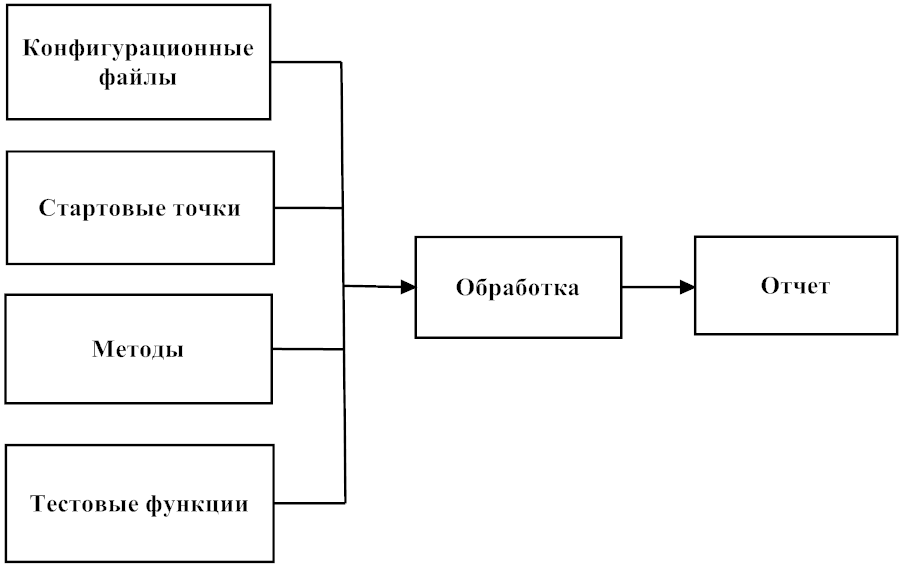


Рисунок 1. Архитектура фреймворка

В конфигурационных файлах прописаны значения параметров. Конфигурационные файлы состоят из общего файла (start.txt) и индивидуальных для каждого метода. В start.txt прописаны значения следующих параметров:

* mcode - порядковый номер метода;
* fcode - порядковый номер функции бенчмарка;
* n - размерность задачи;
* launches - число запусков для усреднения;
* psize - количество точек;
* max\_call\_f - максимальное число вызовов целевой функции.

В папке config находятся конфигурационные файлы с параметрами для каждого метода, которые не вошли в start.txt.

Наборы стартовых точек находятся в каталоге points. Каждый файл предназначен для определенной тестовой функции и размерности. Это отражено в названии файла.

## **Программная реализация фреймворка**

В качестве инструмента разработки использовался язык C++ (стандарт C++11). Среда разработки – Microsoft Visual Studio 2013.

Основные модули разработанного программного комплекса описаны в Таблице 1.

Таблица 1.

Модули программного комплекса

|  |  |
| --- | --- |
| **Модуль и его файловый состав** | **Описание** |
| Базовый модуль (main.cpp) | Модуль отвечает за запуск тестирования любого набора методов из библиотеки на любом наборе тестовых функций. |
| Библиотека методов глобальной оптимизации (sa.h, ea.h, pso.h, eapso.h, difevo.h) | В отдельных заголовочных файлах реализованы следующие алгоритмы: метод имитации отжига, эволюционный алгоритм, метод роя частиц, модификация эволюционного алгоритма EA-PSO, метод дифференциальной эволюции. |
| Библиотеки для разработки многоточечных методов (population.h, person.h, swarm.h, particle.h, popswarm.h, geneparticle.h) | Модуль содержит вспомогательные классы Population, Person, Swarm, Particle, PopSwarm, GeneParticle, упрощающие реализацию многоточечных методов. |
| Библиотека генерации точек (point.h) | Библиотека содержит функции, отвечающие за различные способы генерации точек. |
| Библиотека тестовых функций (testfun.h) | Модуль представляет собой пространство имен testfun, которое содержит данные об используемых тестовых функциях, такие как сам программный код, границы области определения и названия. |
| Модуль генерации отчетов (report.h) | Модуль содержит 2 класса, реализующие генерацию отчета: Report и FullReport. |
| Статистический модуль (stats.h) | Модуль отвечает за расчет основных статистик по запускам: минимальное, максимальное, среднее значения и стандартное отклонение. |
| Утилиты (utils.h) | В этом модуле содержатся различные вспомогательные утилиты. |

## **Тестирование методов**

Для того, чтобы протестировать любой набор методов на любом наборе тестовых функций, необходимо сделать следующее:

1. Прописать в start.txt общие параметры. Перед mcode и fcode нужно указать количество методов и тестовых функций соответственно, а сами значения параметров указать после через пробел. Например, для запуска метода имитации отжига и генетического алгоритма на 2 функциях griewank и ackley файл будет выглядеть следующим образом (см. Рис. 2).

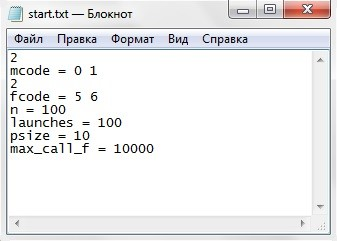


Рисунок 2. Файл с общими параметрами

Для того чтобы узнать порядковый номер метода можно посмотреть главную функцию main() в файле main.cpp (см. Листинг 1), а номер функции - функцию вызова f в testfun.h (см. Листинг 2).

Листинг 1. Функция main() в файле main.cpp

|  |
| --- |
| int main()  {  string input\_fname = "start.txt";  FILE \*input\_file = freopen(input\_fname.c\_str(), "r", stdin);  string s;  char c;  int nm, nf, n, launches, psize, max\_call\_f;  cin >> nm >> s >> c;  vector<int> mcodes(nm);  for (int i = 0; i < nm; i++)  cin >> mcodes[i];  cin >> nf >> s >> c;  vector<int> fcodes(nf);  for (int i = 0; i < nf; i++)  cin >> fcodes[i];  cin >> s >> c >> n;  cin >> s >> c >> launches;  cin >> s >> c >> psize;  cin >> s >> c >> max\_call\_f;  fclose(input\_file);  for (int i = 0; i < nm; i++)  for (int j = 0; j < nf; j++)  {  **switch (mcodes[i])**  **{**  **case 0: {start\_sa(fcodes[j], n, launches, max\_call\_f); break; } //метод имитации отжига**  **case 1: {start\_ga(fcodes[j], n, launches, psize, max\_call\_f); break; } //генетический алгоритм**  **case 2: {start\_difevo(fcodes[j], n, launches, psize, max\_call\_f); break; } //метод дифференциальной эволюции**  **case 3: {start\_pso(fcodes[j], n, launches, psize, max\_call\_f); break; } //метод роя частиц**  **case 4: {start\_gapso(fcodes[j], n, launches, psize, max\_call\_f); break; }**  **case 5: {start\_ais(fcodes[j], n, launches, psize, max\_call\_f); break; } //метод искусcтвенных имунных систем**  **}**  }  return 0;  } |

Листинг 2. Код функции f() в файле testfun.h

|  |
| --- |
| double f(const int &n, const point &x, const int &fcode)  {  ++call\_f;  switch (fcode)  {  case 0: return f\_sphere(n, x);  case 1: return f\_rosenbrock(n, x);  case 2: return f\_gauss(n, x);  case 3: return f\_rastrigin(n, x);  case 4: return f\_schwefel(n, x);  case 5: return f\_griewank(n, x);  case 6: return f\_ackley(n, x);  case 7: return f\_rastrigin\_novgorod(n, x);  case 8: return f\_step(n, x);  case 9: return f\_salomon(n, x);  case 10: return f\_alpine1(n, x);  case 11: return f\_brown(n, x);  case 12: return f\_deb1(n, x);  case 13: return f\_egg\_holder(n, x);  case 14: return f\_pinter(n, x);  case 15: return f\_shubert3(n, x);  case 16: return f\_streched\_v\_sine\_wave(n, x);  case 17: return f\_trigonometric2(n, x);  case 18: return f\_wavy(n, x);  case 19: return f\_whitley(n, x);  default: return 0;  }  } |

2. Изменить данные в соответствующих файлах параметров методов. В первой строке файла указать количество наборов параметров, далее прописать параметры через пробел (см. Рис. 3).

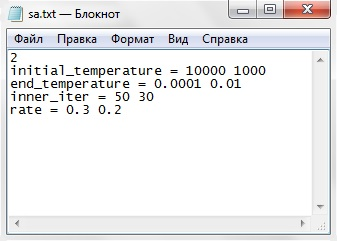


Рисунок 3. Пример файла параметров метода

1. Запустить программу и ждать окончания работы.   
   В случае отсутствия необходимого набора стартовых точек для нужной тестовой функции и/или размерности задачи он будет сгенерирован автоматически и сохранен.
2. В папку results добавятся 2 файла формата csv с результатами тестирования. (см. Рис. 4). Имя первого файла состоит из названия метода, функции, размерности и параметров через «\_». Первая строка содержит количество вызовов тестовой функции на каждой итерации. Каждая следующая строка – каждый отдельный запуск, в ячейке которого записано значение функции на соответствующей итерации. Последняя строка – усредненные по столбцам значения.

Имя второго файла начинается со «Stats\_». В этом файле отображаются основные статистики по запускам: минимальное, максимальное, среднее значения и стандартное отклонение достигнутого минимума (см. Рис. 5).

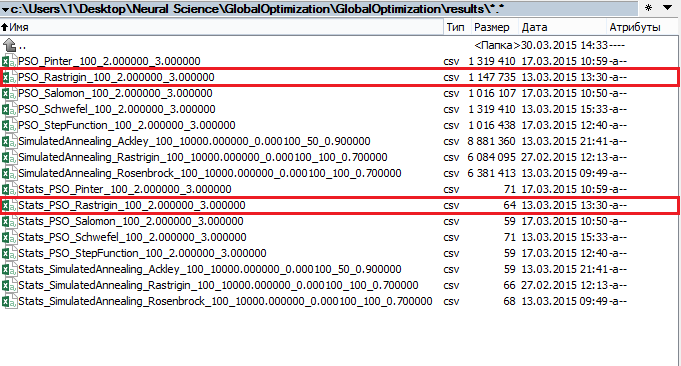


Рисунок 4. Каталог с файлами результатов тестирования

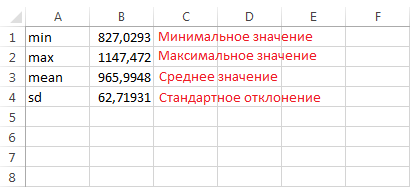


Рисунок 5. Файл со статистикой

## **Добавление метода**

Для добавления нового метода необходимо выполнить следующие действия:

1. Создать нужное количество заголовочных файлов (с расширением .h) для разрабатываемого метода. Имена заголовочных файлов должны отражать суть. Например, файл метода имитации отжига (Simulated Annealing) может быть назван sa.h.
2. Прописать программный код в созданный(е) файл(ы). Сам метод должен быть реализован в виде функции, которая возвращает объект класса FullReport. Эта функция содержит следующие обязательные параметры:

* порядковый номер целевой функции;
* размерность задачи;
* стартовая точка типа point (для одноточечных методов) или вектор стартовых точек (для многоточечных);
* максимальное число вызовов целевой функции.

Остальные параметры определяются разработчиком алгоритма. Рекомендуется передавать параметры по ссылке и указывать перед типом const.

Тело функции начинается с создания объекта класса FullReport, который содержит информацию для отчетности по всему одинарному запуску, а именно количество вызовов целевой функции и её значение на каждой итерации. Также создается объект класса Report – туда записываются данные одной итерации. Добавление его в полный отчет производится с помощью метода insert\_into\_report. Функция разрабатываемого метода возвращает объект класса FullReport, созданный в начале.

В качестве генератора псевдослучайных чисел рекомендуется использовать Вихрь Мерсенна, который обеспечивает быструю генерацию высококачественных псевдослучайных чисел.

В качестве примера ниже приведен программный код функции метода роя частиц (см. Листинг 3).

Листинг 3. Функция метода

|  |
| --- |
| FullReport pso(const int &fcode, const int &n, vector<point> &init\_swarm, const int &max\_call\_f, const double &a1, const double &a2)  {  FullReport fr;  int swarm\_size = init\_swarm.size();  vector<Particle> w;  for (int i = 0; i < swarm\_size; i++)  {  Particle x(fcode, init\_swarm[i]);  w.push\_back(x);  }  Swarm p\_s(w);  testfun::call\_f = 0;  Report r(testfun::call\_f, p\_s.calc\_averageValue());  fr.insert\_into\_report(r);  while (testfun::call\_f < max\_call\_f)  {  p\_s.swarmMove(a1, a2, fcode);  Report r(testfun::call\_f, p\_s.calc\_averageValue());  fr.insert\_into\_report(r);  }  return fr;  } |

1. В папку config добавить конфигурационный файл формата txt реализованного метода с соответствующим названием и прописать его параметры.
2. В main.cpp добавить функцию для тестирования реализованного метода (см. Листинг 4), которая должна иметь следующие параметры:

* порядковый номер целевой функции;
* размерность задачи;
* число запусков для усреднения;
* максимальное число вызовов целевой функции.

Листинг 4. Функция для тестирования метода

|  |
| --- |
| void start\_sa(const int &fcode, const int &n, const int &launches, const int &max\_call\_f)  {  ifstream config\_file("config\\sa.txt");  string s;  char c;  int i, num\_sets;  config\_file >> num\_sets;  vector<double> init\_t(num\_sets), end\_t(num\_sets), rate(num\_sets);  vector<int> inner\_iter(num\_sets);  config\_file >> s >> c;  for (i = 0; i < num\_sets; i++)  config\_file >> init\_t[i];  config\_file >> s >> c;  for (i = 0; i < num\_sets; i++)  config\_file >> end\_t[i];  config\_file >> s >> c;  for (i = 0; i < num\_sets; i++)  config\_file >> inner\_iter[i];  config\_file >> s >> c;  for (i = 0; i < num\_sets; i++)  config\_file >> rate[i];  config\_file.close();  check\_points\_file(fcode, n);  for (int p = 0; p < num\_sets; p++)  {  ifstream points\_file("points\\" + namefun[fcode] + "-" + to\_string(n));  string output\_fname = "results\\SimulatedAnnealing\_" + namefun[fcode] + "\_" + to\_string(n) + "\_" + to\_string(init\_t[p]) + "\_" + to\_string(end\_t[p]) + "\_" + to\_string(inner\_iter[p]) + "\_" + to\_string(rate[p]) + ".csv";  FILE \*output\_file = freopen(output\_fname.c\_str(), "a", stdout);  vector<FullReport> fr(launches);  vector<double> final;  point st(n);  for (int j = 0; j < n; j++)  points\_file >> st[j];  fr[0] = SimulatedAnnealing(fcode, n, st, max\_call\_f, init\_t[p], end\_t[p], inner\_iter[p], rate[p]);  fr[0].print\_full\_report\_cf();  fr[0].print\_full\_report();  final.push\_back(fr[0].show\_report().back().show\_value\_function());  for (i = 1; i < launches; i++)  {  point st(n);  for (int j = 0; j < n; j++)  points\_file >> st[j];  fr[i] = SimulatedAnnealing(fcode, n, st, max\_call\_f, init\_t[p], end\_t[p], inner\_iter[p], rate[p]);  fr[i].print\_full\_report();  final.push\_back(fr[i].show\_report().back().show\_value\_function());  }  print\_mean(fr, launches);  points\_file.close();  fclose(output\_file);  stats stts = get\_stats(final);  string stats\_fname = "results\\Stats\_SimulatedAnnealing\_" + namefun[fcode] + "\_" + to\_string(n) + "\_" + to\_string(init\_t[p]) + "\_" + to\_string(end\_t[p]) + "\_" + to\_string(inner\_iter[p]) + "\_" + to\_string(rate[p]) + ".csv";  FILE \*stats\_file = freopen(stats\_fname.c\_str(), "a", stdout);  cout\_stats(stts);  fclose(stats\_file);  final.clear();  }  } |

1. Добавить вызов функции из предыдущего этапа в main() (см. Рис. 6) и подключить заголовочный файл метода (см. Рис. 7).

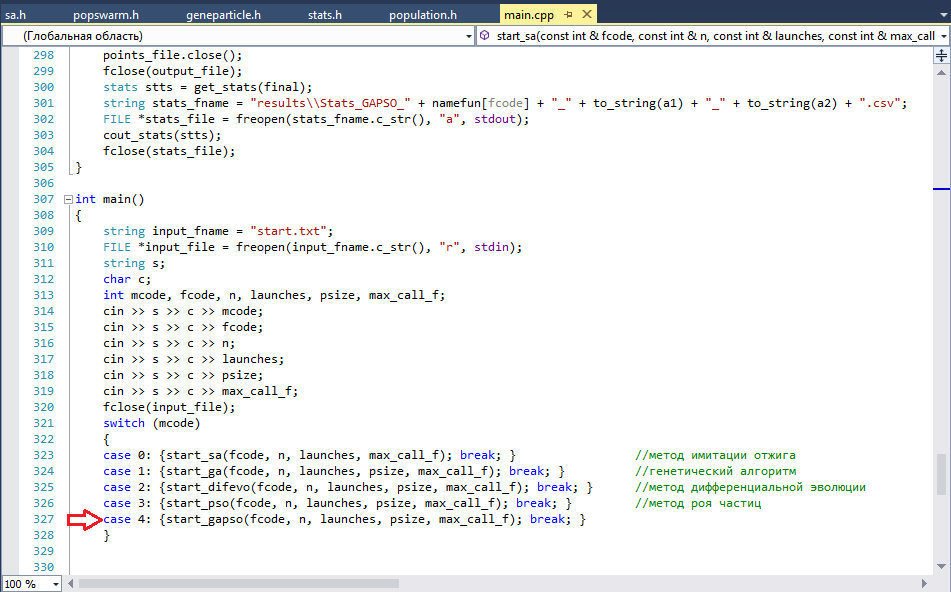


Рисунок 6. Вызов функции для тестирования метода в файле

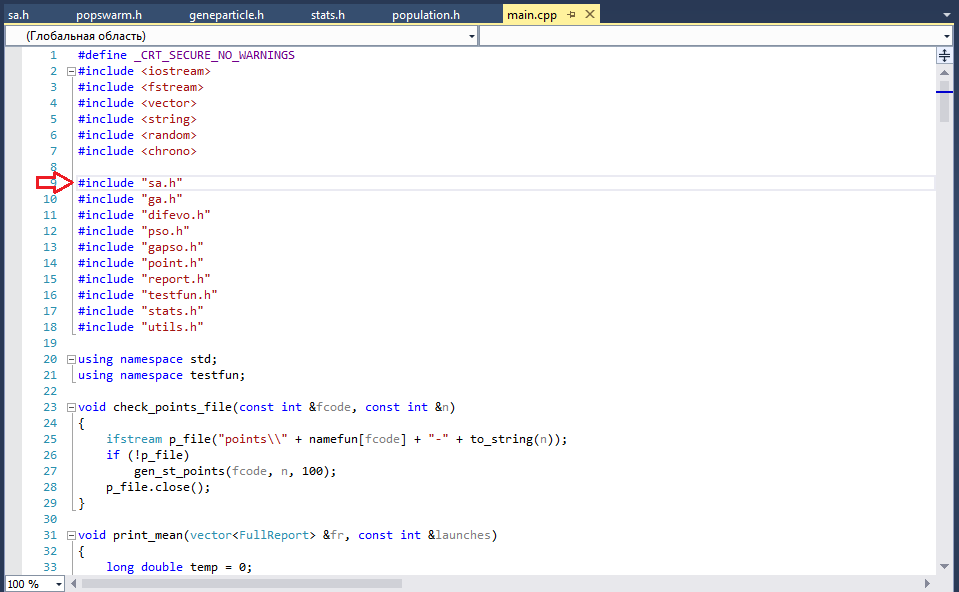


Рисунок 7. Подключение заголовочного файла метода

1. **Добавление тестовой функции**

Для того, чтобы добавить тестовую функцию, необходимо сделать следующее:

1. Добавить код функции в пространство имён testfun (см. Листинг 5).

Листинг 5. Пример функции

|  |
| --- |
| double f\_sphere(const int &n, const point &x)  {  double s = 0;  for (int i = 0; i < n; i++)  s += x[i] \* x[i];  return s;  } |

1. Добавить значения левой и правой границ области определения в векторы lb и rb соответственно. В вектор namefun добавить название функции, заключенное в кавычки (см. Рисунок 8).

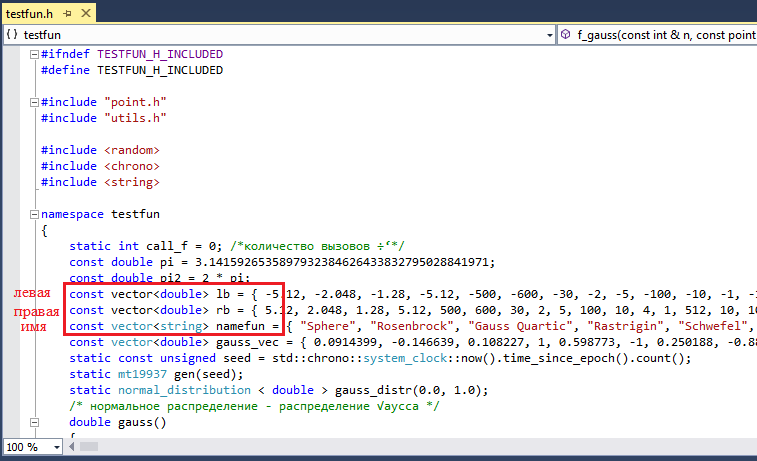


Рисунок 8. Ограничения тестовых функций

1. Прописать в главной функции вызова f новую функцию с соответствующим порядковым номером (см. Рисунок 9). Стоит обратить внимание на то, что порядковые номера функции в вызове и в векторах границ области определения, названий должны совпадать.

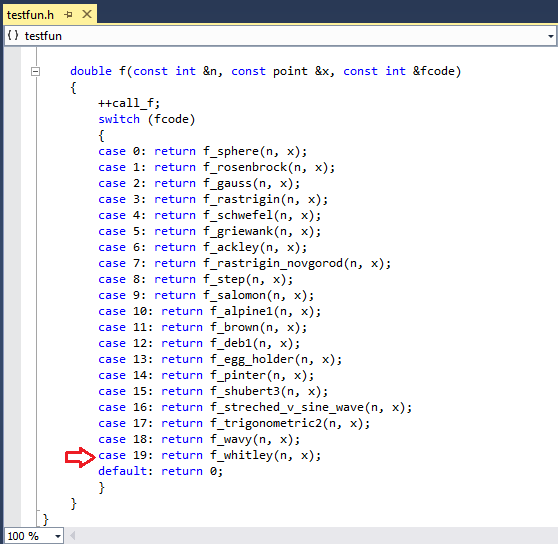


Рисунок 9. Вызов тестовых функций