Globalizer

1.1

Создано системой Doxygen 1.14.0

Глава 1

Иерархический список классов

1.1 Иерархия классов

Иерархия классов.

Глава 2

Алфавитный указатель классов

2.1 Классы

Классы с их кратким описанием.

Глава 3

Список файлов

3.1 Файлы

Полный список документированных файлов.

Глава 4

Классы

4.1 Шаблон структуры _binary_function< _Arg1, _Arg2, _Result >

Открытые типы

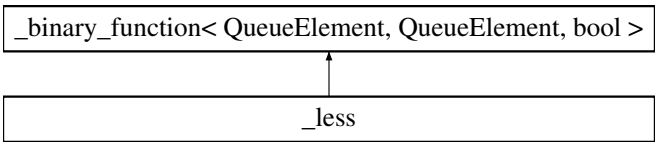
- typedef _Arg1 first_argument_type
- typedef _Arg2 second_argument_type
- typedef _Result result_type

Объявления и описания членов структуры находятся в файле:

- globalizer/method/include/QueueCommon.h

4.2 Структура _less

Граф наследования:_less:



Открытые члены

- bool operator() (const QueueElement &_Left, const QueueElement &_Right) const

Дополнительные унаследованные члены

Открытые типы унаследованные от _binary_function< QueueElement, QueueElement, bool >

- typedef QueueElement first_argument_type
- typedef QueueElement second_argument_type
- typedef bool result_type

Объявления и описания членов структуры находятся в файле:

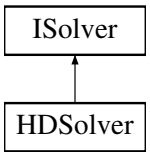
- globalizer/method/include/QueueCommon.h

4.3 Класс HDSolver

Базовый класс для решателя задач большой размерности

#include <HDSolver.h>

Граф наследования:HDSolver:



Открытые члены

- HDSolver (IProblem *problem, std::vector< int > _dimentions={})
- virtual int Solve ()

  Решение задачи по умолчанию
- SolutionResult * GetSolutionResult ()
- virtual void SetPoint (std::vector< Trial * > &points)

  Добавляет точки испытаний
- virtual std::vector< Trial * > & GetAllPoint ()

  Возвращает все имеющиеся точки испытаний

Защищенные члены

- void SetDimentions (std::vector< int > _dimentions)

  Задачть размерности
- void CreateStartPoint ()

  Создать начальную точку решения задач
- void Construct ()

  заполняет основные поля класса
- void AddPoint (Solver *solver, int i, std::vector< Trial * > &points, int startParameterNumber)

  Добавляет вычесленные точки в общий массив с точками
- void UpdateStartPoint (SolutionResult *solution, double &bestValue, int curDimensions, int start↩ ParameterNumber, std::vector< Trial * > &points, HDTask *curTask)

  Обновляет стартовую точку

Защищенные данные

- std::vector< Solver * > solvers

  Решатели для оптимизации по группам параметров
- Solver * finalSolver

  Решатель для объединения всех остальных Решателей
- std::vector< int > dimensions

  Размерости групп парамеров, по умолчанию по 1.
- SolutionResult * solutionResult

  Решение задачи оптимизации
- int originalDimension

Размерность исходной задачи

- std::vector< HDTask * > tasks

  Задачи для оптимизации по группам параметров

- IProblem * problem

  задача оптимизации

- std::vector< double > alternativeStartingPoint

  альтернативная стартовая точка на случай если не удалось улучшить решение

4.3.1 Подробное описание

Базовый класс для решателя задач большой размерности

4.3.2 Методы

4.3.2.1 GetAllPoint()

std::vector< Trial * > & HDSolver::GetAllPoint () [virtual]

Возвращает все имеющиеся точки испытаний

Замещает ISolver.

4.3.2.2 SetPoint()

void HDSolver::SetPoint (
std::vector< Trial * > & points) [virtual]

Добавляет точки испытаний

Замещает ISolver.

4.3.2.3 Solve()

int HDSolver::Solve () [virtual]

Решение задачи по умолчанию

curr_child = parameters.parallel_tree.ProcChild[i];!!!!!

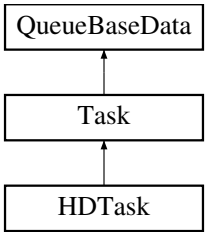
Замещает ISolver.

Объявления и описания членов классов находятся в файлах:

- globalizer/include/HDSolver.h
- globalizer/src/HDSolver.cpp

4.4 Класс HDTask

Граф наследования:HDTask:



Открытые члены

- HDTask (IProblem *_problem, int _ProcLevel)
- virtual Task * Clone ()

  Создает копию класса
- virtual const double * GetA () const

  Возвращает левую границу области поиска
- virtual const double * GetB () const

  Возвращает правую границу области поиска
- virtual const double * GetOptimumPoint () const

  Возвращает априори известные координаты точки глобального минимума Перед первым вызовом нужно вызвать resetOptimumPoint()
- virtual double CalculateFuncs (const double *y, int fNumber)

  Вычисляет значение функции с номером fNumber в точке y.
- virtual void CalculateFuncsInManyPoints (double *y, int fNumber, int numPoints, double *values)

  Вычисляет numPoints значений функции с номером fNumber, в координатах y в массив values Работает только если задача является наследником IGPUProblem.
- void SetStartParameterNumber (int _startParameterNumber)

  Задает начало блока переменных
- virtual void CopyPoint (double *y, Trial *point)

  Копирует координаты точки из массива, согласно имеющимся правилам

Открытые члены унаследованные от Task

- Task (IProblem *_problem, int _ProcLevel)

  Конструктор.
- Task ()

  Конструктор по умолчанию.
- virtual ~Task ()

  Деструктор.
- virtual Task * CloneWithNewData ()

  Создает клон текущего объекта с новыми данными (в данной реализации эквивалентно Clone).
- virtual void Init (IProblem *_problem, int _ProcLevel)

  Инициализирует объект данными задачи.
- virtual int GetN () const

  Возвращает общую размерность задачи.
- virtual double GetOptimumValue () const

  Возвращает априори известное значение глобального минимума.

- virtual void resetOptimumPoint ()

  Обновляет координаты точки глобального минимума из объекта #IProblem.
- virtual bool GetIsOptimumValueDefined () const

  Проверяет, известно ли для задачи значение глобального минимума.
- virtual bool GetIsOptimumPointDefined () const

  Проверяет, известны ли для задачи координаты глобального минимума.
- virtual IProblem * getProblem ()

  Возвращает указатель на текущую задачу.
- virtual int GetNumOfFunc () const

  Возвращает число функций (ограничения и критерии).
- virtual void SetNumofFunc (int nf)

  Задает число функций.
- int GetProcLevel ()

  Возвращает уровень процесса в дереве процессов.
- virtual int GetNumOfFuncAtProblem () const

  Возвращает число функций в исходной задаче.
- virtual int GetNumberOfDiscreteVariable ()

  Возвращает число дискретных параметров.
- virtual int GetNumberOfValues (int discreteVariable)

  Возвращает число допустимых значений для дискретного параметра.
- virtual int GetAllDiscreteValues (int discreteVariable, double *values)

  Определяет все допустимые значения дискретного параметра.
- virtual bool IsPermissibleValue (double value, int discreteVariable)

  Проверяет, является ли значение допустимым для дискретного параметра.
- virtual double * getMin ()

  Возвращает минимальные значения функций (для многокритериальной оптимизации).
- virtual double * getMax ()

  Возвращает максимальные значения функций (для многокритериальной оптимизации).
- virtual bool IsInit ()

  Проверяет, был ли объект инициализирован.
- virtual bool IsLeaf ()

  Проверяет, является ли задача листом в дереве процессов.

Открытые члены унаследованные от QueueBaseData

- virtual void SetQueueElementa (QueueElement *q)
- virtual QueueElement * GetQueueElementa ()

Защищенные данные

- int startParameterNumber

  Начало блока переменных

Защищенные данные унаследованные от Task

- double A [MaxDim]

  Левая граница области поиска
- double B [MaxDim]

  Правая граница области поиска
- int NumOfFunc

  Число функционалов (последний - критерий)
- IProblem * pProblem

  Указатель на саму задачу оптимизации
- double OptimumValue

  Оптимальное значение целевой функции (определено, если известно из задачи)
- double OptimumPoint [MaxDim]

  Координаты глобального минимума целевой функции (определено, если известно)
- bool IsOptimumValueDefined

  true, если в задаче известно оптимальное значение критерия
- bool IsOptimumPointDefined

  true, если в задаче известна точка глобального минимума
- int ProcLevel

  Уровень процесса в дереве процессов
- bool isInit

  Флаг, указывающий, был ли класс инициализирован

Защищенные данные унаследованные от QueueBaseData

- QueueElement * queueElementa

  Элемент очереди, хранящий этот интервал

Дополнительные унаследованные члены

Открытые атрибуты унаследованные от Task

- int num

4.4.1 Методы

4.4.1.1 CalculateFuncs()

```
double HDTask::CalculateFuncs (
            const double * y,
            int fNumber)   [virtual]
```

Вычисляет значение функции с номером fNumber в точке y.

Переопределяет метод предка Task.

4.4.1.2 CalculateFuncsInManyPoints()

```
void HDTask::CalculateFuncsInManyPoints (
                double * y,
                int fNumber,
                int numPoints,
                double * values)   [virtual]
```

Вычисляет numPoints значений функции с номером fNumber, в координатах y в массив values Работает только если задача является наследником IGPUProblem.

Переопределяет метод предка Task.

4.4.1.3 Clone()

```
Task * HDTask::Clone ()   [virtual]
```

Создает копию класса

Переопределяет метод предка Task.

4.4.1.4 CopyPoint()

```
void HDTask::CopyPoint (
                double * y,
                Trial * point)   [virtual]
```

Копирует координаты точки из массива, согласно имеющимся правилам

Аргументы

in	y	имеющисся координаты.	
out	point	точка назначения.	

Возвращает

true, если значение допустимо, иначе false.

Переопределяет метод предка Task.

4.4.1.5 GetA()

```
const double * HDTask::GetA () const   [virtual]
```

Возвращает левую границу области поиска

Переопределяет метод предка Task.

4.4.1.6 GetB()

const double * HDTask::GetB () const [virtual]

Возвращает правую границу области поиска

Переопределяет метод предка Task.

4.4.1.7 GetOptimumPoint()

const double * HDTask::GetOptimumPoint () const [virtual]

Возвращает априори известные координаты точки глобального минимума Перед первым вызовом нужно вызвать resetOptimumPoint()

Переопределяет метод предка Task.

Объявления и описания членов классов находятся в файлах:

- globalizer/method/include/HDTask.h
- globalizer/method/src/HDTask.cpp

4.5 Class I Method

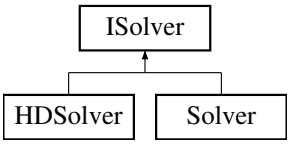
Этот раздел временно недоступен.

4.6 Класс ISolver

Интерфейс, базового класса

#include <SolverInterface.h>

Граф наследования:ISolver:



Открытые члены

- virtual int Solve ()=0
  Решить задачу
- virtual void SetPoint (std::vector< Trial * > &points)=0
  Добавляет точки испытаний
- virtual std::vector< Trial * > & GetAllPoint ()=0
  Возвращает все имеющиеся точки испытаний

4.6.1 Подробное описание

Интерфейс, базового класса

4.6.2 Методы

4.6.2.1 GetAllPoint()

virtual std::vector< Trial * > & ISolver::GetAllPoint () [pure virtual]

Возвращает все имеющиеся точки испытаний

Замещается в HDSolver и Solver.

4.6.2.2 SetPoint()

virtual void ISolver::SetPoint (
std::vector< Trial * > & points) [pure virtual]

Добавляет точки испытаний

Замещается в HDSolver и Solver.

4.6.2.3 Solve()

virtual int ISolver::Solve () [pure virtual]

Решить задачу

Замещается в HDSolver и Solver.

Объявления и описания членов класса находятся в файле:

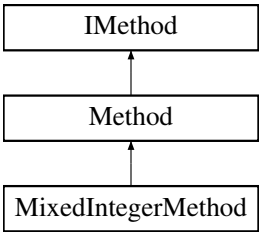
- globalizer/include/SolverInterface.h

4.7 Класс Method

Базовый класс, реализующий алгоритм глобального поиска.

#include <Method.h>

Граф наследования:Method:



Открытые члены

- Method (Task &_pTask, SearchData &_pData, Calculation &_Calculation, Evolvent &_↩ Evolvent)
- virtual void FirstIteration ()

  Функция выполняет первую итерацию метода
- virtual void CalculateIterationPoints ()

  Вычисления точек очередной итерации
- virtual void CalculateFunctionals ()

  Вычисление функций задачи
- virtual void RenewSearchData ()

  Обновление поисковой информации
- virtual bool CheckStopCondition ()

  Проверка выполнения критерия остановки метода
- virtual bool EstimateOptimum ()

  Оценить текущее значение оптимума
- virtual void FinalizeIteration ()

  Функция вызывается в конце проведения итерации
- virtual int GetIterationCount ()

  Получить число испытаний
- virtual Trial * GetOptimEstimation ()

  Получить текущую оценку оптимума
- virtual int GetNumberOfTrials ()

  Сбор статистики
- virtual void PrintLevelPoints (const std::string &fileName)

  сохраняем точки с уровня
- virtual void PrintPoints (const std::string &fileName)

  Сохраняем все точки, со всех уровней, в файл
- void HookeJeevesMethod (Trial &point, std::vector< Trial * > &localPoints)

  Метод Хука-Дживса
- virtual std::vector< int > GetFunctionCalculationCount ()

  Возвращает Число вычислений каждой функции
- virtual double GetAchievedAccuracy ()

  Возвращает достигнутую точность
- virtual void ResetSearchData ()

  Обновление поисковой информации
- void InsertPoints (const std::vector< Trial * > &points)

  Добавляет испытания в поисковую информацию, при этом обновляя константу Гёльдера и оценку оптимума
- virtual void InsertLocalPoints (const std::vector< Trial * > &points, Task *task=0)

  Добавляет испытания полученные локальным методом в поисковую информацию, при этом обновляя константу Гёльдера и оценку оптимума
- virtual void LocalSearch ()

  Запускает локальный метод
- virtual int GetLocalPointCount ()

  Возвращает число точек полученное от локальныго метода
- virtual int GetNumberLocalMethodtStart ()

  Возвращает число запусков локально метода
- virtual void PrintSection ()

  Печатает информацию о сечениях

Защищенные члены

- virtual void SavePoints ()

  Метод сохраняющий точки в статический массив
- virtual double CalculateGlobalR (SearchInterval *p)

  Вычисление "глобальной" характеристики
- virtual double CalculateLocalR (SearchInterval *p)

  Вычисление "локальной" характеристики
- virtual void CalculateM (SearchInterval *p)

  Вычисление оценки константы Липшица
- virtual IterationType GetIterationType (int iterationNumber, int localMixParameter)

  Определение типа текущей итерации: локальная или глобальная
- virtual int IsBoundary (SearchInterval *p)

  Определение, является интервал граничным или нет 0 - Не граничный; 1 - Левая граница; 2 - Правая граница
- virtual void UpdateM (double newValue, int index, int boundaryStatus, SearchInterval *p)

  Обновление константы Липшица для функции с заданным индексом
- virtual bool UpdateOptimumEstimation (Trial &trial)

  Обновление текущей оценки оптимума
- virtual void CalculateCurrentPoint (Trial &pCurTrialsj, SearchInterval *BestIntervalsj)

  Вычисление координат точек испытания для основной\единственной развертки
- virtual void CalculateCurrentPoints (std::vector< SearchInterval * > &BestIntervals)

  Вычисляем координаты точек которые будем использовать на текущей итерации
- virtual bool IsIntervalInSegment (SearchInterval *basicInterval, SearchInterval *newInterval)

  Пренадлежит ли newInterval отрезку в котором находится basicInterval.
- virtual double Update_r (int iter=-1, int procLevel=-1)

  Изменение при динамичеки изменяемом r, r = r + rDynamic / (Iteration ^ (1/N))
- virtual void CalculateImage (Trial &pCurTrialsj)

  x--> y; Вычисляет координаты y в гиперкубе по x из отрезка
- virtual SearchInterval * AddCurrentPoint (Trial &pCurTrialsj, SearchInterval *BestIntervalsj)

  Добавление основных (из основной\единственной развертки) точек испытания в базу
- virtual void Recalc ()

  Перерасчет характеристик и перестройка очереди
- virtual SearchData * GetSearchData (Trial *trial)

  Получаем поисковую информацию, важно для адаптивного метода
- virtual void SetNumPoints (int newNP)

  Изменить количество текущих точек испытаний, переписывает #iteration.pCurTrials и #iteration.BestIntervals.

Защищенные данные

- int MaxNumOfTrials

  Максимальное число испытаний
- int StartLocalIteration

  число итераций до включения смешанного алгоритма
- bool isGlobalMUpdate

  Обновлено глобальное M
- bool isLocalZUpdate

  Обновлена лучшая точка в текущей задаче
- Task & pTask

  Указатель на решаемую задачу

- SearchData * pData

  Указатель на матрицу состояния поиска
- Calculation & calculation

  Вычислитель
- Evolvent & evolvent

  Указатель на развертку
- InformationForCalculation inputSet

  Входные данные для вычислителя, формирубтся в CalculateFunctionals()
- TResultForCalculation outputSet

  Выходные данные вычислителя, обрабатывается в CalculateFunctionals()
- SearchIteration iteration

  информация о данных текущей итерации
- bool isFoundOptimalPoint

  Была получена точка в окрестности глобального оптимума
- double AchievedAccuracy

  достигнутая точность
- double alfa

  Коэффициент локальной адаптации
- std::vector< int > functionCalculationCount

  Число вычисленных значений каждой функции
- bool isFindInterval

  нужно ли искать интервал
- bool isSetInLocalMinimumInterval

  Новая точка устанавливается в интервал принадлежащий окрестности локального минимума
- int localPointCount

  количество точек вычисленных локальным методом
- int numberLocalMethodtStart

  число запусков локально метода
- bool isStop

  Нужно останавливаться
- std::vector< Trial * > localMinimumPoints

  найденные локальные минимумы
- double * Xmax

  Максимальные длины интервалов для разных индексов правой точки
- double * mu

  Значения оценки константы Липшица для разных индексов правой точки
- bool isSearchXMax

  Инициализирован ли Xmax.
- std::vector< Trial * > printPoints

  Массив для сохранения точек для последующей печати и рисования

4.7.1 Подробное описание

Базовый класс, реализующий алгоритм глобального поиска.

В классе Method реализованы основные функции, определяющие работу алгоритма глобального поиска.

4.7.2 Конструктор(ы)

4.7.2.1 Method()

```
Method::Method (
            Task & _pTask,
            SearchData & _pData,
            Calculation & _Calculation,
            Evolvent & _Evolvent)
```

количество точек вычисленных локальным методом

число запусков локально метода

4.7.3 Методы

4.7.3.1 AddCurrentPoint()

```
SearchInterval * Method::AddCurrentPoint (
            Trial & pCurTrialsj,
            SearchInterval * BestIntervalsj)   [protected], [virtual]
```

Добавление основных (из основной\единственной развертки) точек испытания в базу

Добавление основных (из основной\единственной развертки) точек испытания в базу, возвращает правый

4.7.3.2 CalculateCurrentPoint()

```
void Method::CalculateCurrentPoint (
            Trial & pCurTrialsj,
            SearchInterval * BestIntervalsj)   [protected], [virtual]
```

Вычисление координат точек испытания для основной\единственной развертки

Вычисляет координаты на отрезке 0..1 для всех разверток по образу проведенного испытания

Переопределяется в MixedIntegerMethod.

4.7.3.3 CalculateFunctionals()

```
void Method::CalculateFunctionals ()   [virtual]
```

Вычисление функций задачи

Проводятся испытания в точках из массива #iteration.pCurTrials, результаты проведенных испытаний записываются в тот же массив

Замещает IMethod.

4.7.3.4 CalculateGlobalR()

```
double Method::CalculateGlobalR (
            SearchInterval * p)   [protected], [virtual]
```

Вычисление "глобальной" характеристики

Аргументы

	in	p	указатель на интервал, характеристику которого надо вычислить

Возвращает

"Глобальная" характеристика интервала

4.7.3.5 CalculateIterationPoints()

void Method::CalculateIterationPoints () [virtual]

Вычисления точек очередной итерации

Вычисленные точки очередной итерации записываются в массив #iteration.pCurTrials

Замещает IMethod.

Переопределяется в MixedIntegerMethod.

4.7.3.6 CalculateLocalR()

double Method::CalculateLocalR (
SearchInterval * p) [protected], [virtual]

Вычисление "локальной" характеристики

Данная функция должна вызываться только для интервала, у которого вычислена глобальная характеристика, т.е. после вызова функции CalculateGlobalR

Аргументы

	in	p	указатель на интервал, характеристику которого надо вычислить

Возвращает

"Локальная" характеристика интервала

4.7.3.7 CalculateM()

void Method::CalculateM (
SearchInterval * p) [protected], [virtual]

Вычисление оценки константы Липшица

Обновленная оценка константы Липшица записывается в базе алгоритма

Аргументы

	in	p	указатель на интервал

4.7.3.8 CheckStopCondition()

bool Method::CheckStopCondition () [virtual]

Проверка выполнения критерия остановки метода

Метод прекращает работу в следующих случаях:

- число испытаний превысило максимально допустимое значение
- если решается одна задача и выполнен критерий $x_t - x_{t-1} < \epsilon$
- если решается серия задач и выполнен критерий $\|y^k - y^*\| < \epsilon$

Возвращает

истина, если критерий остановки выполнен; ложь - в противном случае.

Замещает IMethod.

4.7.3.9 EstimateOptimum()

bool Method::EstimateOptimum () [virtual]

Оценить текущее значение оптимума

Возвращает

истина, если оптимум изменился; ложь - в противном случае

Замещает IMethod.

4.7.3.10 FinalizeIteration()

void Method::FinalizeIteration () [virtual]

Функция вызывается в конце проведения итерации

Замещает IMethod.

4.7.3.11 FirstIteration()

void Method::FirstIteration () [virtual]

Функция выполняет первую итерацию метода

Необходимо сосчитать значения на границах

Замещает IMethod.

Переопределяется в MixedIntegerMethod.

4.7.3.12 GetAchievedAccuracy()

double Method::GetAchievedAccuracy () [virtual]

Возвращает достигнутую точность

Замещает IMethod.

4.7.3.13 GetFunctionCalculationCount()

std::vector< int > Method::GetFunctionCalculationCount () [virtual]

Возвращает Число вычислений каждой функции

Замещает IMethod.

4.7.3.14 GetIterationCount()

int Method::GetIterationCount () [virtual]

Получить число испытаний

Возвращает

число испытаний

Замещает IMethod.

4.7.3.15 GetIterationType()

IterationType Method::GetIterationType (
int iterationNumber,
int localMixParameter) [protected], [virtual]

Определение типа текущей итерации: локальная или глобальная

Аргументы

	in	iterationNumber	
	in	localMixParameter	параметр смешивания локального и глобального алгоритмов. Возможны три вариант

Возвращает

- тип итерации

4.7.3.16 GetLocalPointCount()

int Method::GetLocalPointCount () [virtual]

Возвращает число точек полученное от локальныго метода

Замещает IMethod.

4.7.3.17 GetNumberLocalMethodtStart()

int Method::GetNumberLocalMethodtStart () [virtual]

Возвращает число запусков локально метода

Замещает IMethod.

4.7.3.18 GetNumberOfTrials()

int Method::GetNumberOfTrials () [virtual]

Сбор статистики

Функция возвращает общее число испытаний, выполненных при решении текущей задачи и всех вложенных подзадач

Возвращает

общее число испытаний

Замещает IMethod.

4.7.3.19 GetOptimEstimation()

Trial * Method::GetOptimEstimation () [virtual]

Получить текущую оценку оптимума

Возвращает

испытание, соответствующее текущему оптимуму

Замещает IMethod.

4.7.3.20 GetSearchData()

SearchData * Method::GetSearchData (
Trial * trial) [protected], [virtual]

Получаем поисковую информацию, важно для адаптивного метода

Переопределяется в MixedIntegerMethod.

4.7.3.21 InsertLocalPoints()

void Method::InsertLocalPoints (
const std::vector< Trial * > & points,
Task * task = 0) [virtual]

Добавляет испытания полученные локальным методом в поисковую информацию, при этом обновляя константу Гёльдера и оценку оптимума

Аргументы

	in	points	точки испытаний, которые будут добавлены

4.7.3.22 InsertPoints()

void Method::InsertPoints (
const std::vector< Trial * > & points) [virtual]

Добавляет испытания в поисковую информацию, при этом обновляя константу Гёльдера и оценку оптимума

Аргументы

	in	points	точки испытаний, которые будут добавлены

Замещает IMethod.

4.7.3.23 LocalSearch()

void Method::LocalSearch () [virtual]

Запускает локальный метод

Замещает IMethod.

4.7.3.24 PrintPoints()

void Method::PrintPoints (
const std::string & fileName) [virtual]

Сохраняем все точки, со всех уровней, в файл

Замещает IMethod.

4.7.3.25 PrintSection()

void Method::PrintSection () [virtual]

Печатает информацию о сечениях

Замещает IMethod.

4.7.3.26 RenewSearchData()

void Method::RenewSearchData () [virtual]

Обновление поисковой информации

Замещает IMethod.

4.7.3.27 SavePoints()

void Method::SavePoints () [protected], [virtual]

Метод сохраняющий точки в статический массив

Замещает IMethod.

4.7.3.28 UpdateM()

void Method::UpdateM (
double newValue,
int index,
int boundaryStatus,
SearchInterval * p) [protected], [virtual]

Обновление константы Липшица для функции с заданным индексом

Если константа обновлена, поднимает флаг #recalc. Данная функция используется в функции CalculateM

Аргументы

	in	newValue	новое значение константы Липшица
	in	index	индекс функции
	in	boundaryStatus	является ли интервал граничным
	in	p	рассматриваемый интервал

4.7.3.29 UpdateOptimumEstimation()

bool Method::UpdateOptimumEstimation (
Trial & trial) [protected], [virtual]

Обновление текущей оценки оптимума

Если переданная точка лучше текущей оценки оптимума, то эта оценка обновляется и поднимается флаг #recalc.

Аргументы

	in	trial	точка, которую необходимо сравнить с текущим оптимумом

Возвращает

true, если оптимум обновлён, иначе false

4.7.4 Данные класса

4.7.4.1 alfa

double Method::alfa [protected]

Коэффициент локальной адаптации

Диапазон значений параметра alfa от 1 (глобальный) до 20 (локальный) поиск Рекомендуемое значение alfa = 15.

4.7.4.2 evolvent

Evolvent& Method::evolvent [protected]

Указатель на развертку

В зависимости от вида отображения это может быть:

- единственная развертка
- множественная сдвиговая развертка
- множественная вращаемая развертка

Объявления и описания членов классов находятся в файлах:

- globalizer/method/include/Method.h
- globalizer/method/src/Method.cpp

4.8 Класс MethodFactory

Открытые статические члены

- static IMethod * CreateMethod (Task &_pTask, SearchData &_pData, Calculation &_↩ Calculation, Evolvent &_Evolvent)

Объявления и описания членов классов находятся в файлах:

- globalizer/method/include/MethodFactory.h
- globalizer/method/src/MethodFactory.cpp

4.9 Шаблон класса MinMaxHeap< T, Compare >

Открытые члены

- MinMaxHeap (unsigned heapsize)
- void clear ()
- bool empty () const
- unsigned int size () const
- T * push (const T &val)
- void TrickleUp (const T *ptr)
- void TrickleDown (const T *ptr)
- T & findMax () const
- const T & findMin () const
- T popMax ()
- T pop ()
- T popMin ()
- void deleteElement (const T *ptr)
- T * getHeapMemPtr () const

Объявления и описания членов класса находятся в файле:

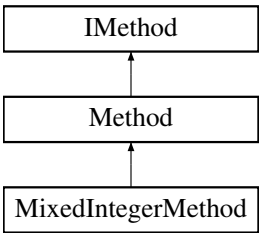
- globalizer/method/include/MinMaxHeap.h

4.10 Класс MixedIntegerMethod

Базовый класс, реализующий алгоритм глобального поиска.

#include <MixedIntegerMethod.h>

Граф наследования:MixedIntegerMethod:



Открытые члены

- MixedIntegerMethod (Task &_pTask, SearchData &_pData, Calculation &_Calculation, Evolvent &_Evolvent)
- virtual void FirstIteration ()

  Функция выполняет первую итерацию метода
- virtual void CalculateIterationPoints ()

  Вычисления точек очередной итерации

Открытые члены унаследованные от Method

- Method (Task &_pTask, SearchData &_pData, Calculation &_Calculation, Evolvent &_↩ Evolvent)
- virtual void CalculateFunctionals ()

  Вычисление функций задачи
- virtual void RenewSearchData ()

  Обновление поисковой информации
- virtual bool CheckStopCondition ()

  Проверка выполнения критерия остановки метода
- virtual bool EstimateOptimum ()

  Оценить текущее значение оптимума
- virtual void FinalizeIteration ()

  Функция вызывается в конце проведения итерации
- virtual int GetIterationCount ()

  Получить число испытаний
- virtual Trial * GetOptimEstimation ()

  Получить текущую оценку оптимума
- virtual int GetNumberOfTrials ()

  Сбор статистики
- virtual void PrintLevelPoints (const std::string &fileName)

  сохраняем точки с уровня
- virtual void PrintPoints (const std::string &fileName)

  Сохраняем все точки, со всех уровней, в файл
- void HookeJeevesMethod (Trial &point, std::vector< Trial * > &localPoints)

  Метод Хука-Дживса
- virtual std::vector< int > GetFunctionCalculationCount ()

  Возвращает Число вычислений каждой функции
- virtual double GetAchievedAccuracy ()

  Возвращает достигнутую точность
- virtual void ResetSearchData ()

  Обновление поисковой информации
- void InsertPoints (const std::vector< Trial * > &points)

  Добавляет испытания в поисковую информацию, при этом обновляя константу Гёльдера и оценку оптимума
- virtual void InsertLocalPoints (const std::vector< Trial * > &points, Task *task=0)

  Добавляет испытания полученные локальным методом в поисковую информацию, при этом обновляя константу Гёльдера и оценку оптимума
- virtual void LocalSearch ()

  Запускает локальный метод
- virtual int GetLocalPointCount ()

  Возвращает число точек полученное от локальныго метода
- virtual int GetNumberLocalMethodtStart ()

  Возвращает число запусков локально метода
- virtual void PrintSection ()

  Печатает информацию о сечениях

Защищенные члены

- virtual void CalculateCurrentPoint (Trial &pCurTrialsj, SearchInterval *BestIntervalsj)

  Вычисление координат точек испытания для основной\единственной развертки
- virtual void SetDiscreteValue (int u, std::vector< std::vector< double > > dvs)

  Задать значения дискретного параметра
- virtual SearchData * GetSearchData (Trial *trial)

  Получаем поисковую информацию, важно для адаптивного метода

Защищенные члены унаследованные от Method

- virtual void SavePoints ()

  Метод сохраняющий точки в статический массив
- virtual double CalculateGlobalR (SearchInterval *p)

  Вычисление "глобальной" характеристики
- virtual double CalculateLocalR (SearchInterval *p)

  Вычисление "локальной" характеристики
- virtual void CalculateM (SearchInterval *p)

  Вычисление оценки константы Липшица
- virtual IterationType GetIterationType (int iterationNumber, int localMixParameter)

  Определение типа текущей итерации: локальная или глобальная
- virtual int IsBoundary (SearchInterval *p)

  Определение, является интервал граничным или нет 0 - Не граничный; 1 - Левая граница; 2 - Правая граница
- virtual void UpdateM (double newValue, int index, int boundaryStatus, SearchInterval *p)

  Обновление константы Липшица для функции с заданным индексом
- virtual bool UpdateOptimumEstimation (Trial &trial)

  Обновление текущей оценки оптимума
- virtual void CalculateCurrentPoints (std::vector< SearchInterval * > &BestIntervals)

  Вычисляем координаты точек которые будем использовать на текущей итерации
- virtual bool IsIntervalInSegment (SearchInterval *basicInterval, SearchInterval *newInterval)

  Пренадлежит ли newInterval отрезку в котором находится basicInterval.
- virtual double Update_r (int iter=-1, int procLevel=-1)

  Изменение при динамичеки изменяемом r, r = r + rDynamic / (Iteration ^ (1/N))
- virtual void CalculateImage (Trial &pCurTrialsj)

  x--> y; Вычисляет координаты y в гиперкубе по x из отрезка
- virtual SearchInterval * AddCurrentPoint (Trial &pCurTrialsj, SearchInterval *BestIntervalsj)

  Добавление основных (из основной\единственной развертки) точек испытания в базу
- virtual void Recalc ()

  Перерасчет характеристик и перестройка очереди
- virtual void SetNumPoints (int newNP)

  Изменить количество текущих точек испытаний, переписывает #iteration.pCurTrials и #iteration.BestIntervals.

Защищенные данные

- int mDiscreteValuesCount

  Количество дискретных значений Произведение числа значений всех дискретных переменных.
- std::vector< std::vector< double > > mDiscreteValues

  Значения дискретных параметров
- int startDiscreteVariable

  Индекс первого дискретного параметра
- std::vector< Trial * > localMinimumPoints

  найденные локальные минимумы

Защищенные данные унаследованные от Method

- int MaxNumOfTrials
  Максимальное число испытаний
- int StartLocalIteration
  число итераций до включения смешанного алгоритма
- bool isGlobalMUpdate
  Обновлено глобальное M
- bool isLocalZUpdate
  Обновлена лучшая точка в текущей задаче
- Task & pTask
  Указатель на решаемую задачу
- SearchData * pData
  Указатель на матрицу состояния поиска
- Calculation & calculation
  Вычислитель
- Evolvent & evolvent
  Указатель на развертку
- InformationForCalculation inputSet
  Входные данные для вычислителя, формирубтся в CalculateFunctionals()
- TResultForCalculation outputSet
  Выходные данные вычислителя, обрабатывается в CalculateFunctionals()
- SearchIteration iteration
  информация о данных текущей итерации
- bool isFoundOptimalPoint
  Была получена точка в окрестности глобального оптимума
- double AchievedAccuracy
  достигнутая точность
- double alfa
  Коэффициент локальной адаптации
- std::vector< int > functionCalculationCount
  Число вычисленных значений каждой функции
- bool isFindInterval
  нужно ли искать интервал
- bool isSetInLocalMinimumInterval
  Новая точка устанавливается в интервал принадлежащий окрестности локального минимума
- int localPointCount
  количество точек вычисленных локальным методом
- int numberLocalMethodtStart
  число запусков локально метода
- bool isStop
  Нужно останавливаться
- std::vector< Trial * > localMinimumPoints
  найденные локальные минимумы
- double * Xmax
  Максимальные длины интервалов для разных индексов правой точки
- double * mu
  Значения оценки константы Липшица для разных индексов правой точки
- bool isSearchXMax
  Инициализирован ли Xmax.
- std::vector< Trial * > printPoints
  Массив для сохранения точек для последующей печати и рисования

4.10.1 Подробное описание

Базовый класс, реализующий алгоритм глобального поиска.

В классе Method реализованы основные функции, определяющие работу алгоритма глобального поиска.

4.10.2 Методы

4.10.2.1 CalculateCurrentPoint()

```
void MixedIntegerMethod::CalculateCurrentPoint (
                Trial & pCurTrialsj,
                SearchInterval * BestIntervalsj)   [protected], [virtual]
```

Вычисление координат точек испытания для основной\единственной развертки

..1

Переопределяет метод предка Method.

4.10.2.2 CalculateIterationPoints()

```
void MixedIntegerMethod::CalculateIterationPoints ()   [virtual]
```

Вычисления точек очередной итерации

Вычисленные точки очередной итерации записываются в массив #iteration.pCurTrials

Переопределяет метод предка Method.

4.10.2.3 FirstIteration()

```
void MixedIntegerMethod::FirstIteration ()   [virtual]
```

Функция выполняет первую итерацию метода

Переопределяет метод предка Method.

4.10.2.4 GetSearchData()

```
SearchData * MixedIntegerMethod::GetSearchData (
                Trial * trial)   [protected], [virtual]
```

Получаем поисковую информацию, важно для адаптивного метода

Переопределяет метод предка Method.

4.10.3 Данные класса

4.10.3.1 mDiscreteValuesCount

int MixedIntegerMethod::mDiscreteValuesCount [protected]

Количество дискретных значений Произведение числа значений всех дискретных переменных.

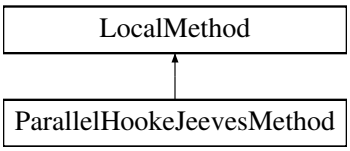
Равно числу интервалов.

Объявления и описания членов классов находятся в файлах:

- globalizer/method/include/MixedIntegerMethod.h
- globalizer/method/src/MixedIntegerMethod.cpp

4.11 Класс ParallelHookeJeevesMethod

Граф наследования:ParallelHookeJeevesMethod:



Открытые члены

- ParallelHookeJeevesMethod (Task *_pTask, Trial _startPoint, Calculation &_calculation, int logPoints=0)

Защищенные члены

- virtual double MakeResearch (OBJECTIV_TYPE *)
- double CheckCoordinate (const OBJECTIV_TYPE *x)
- virtual double EvaluateObjectiveFunctiuon (const OBJECTIV_TYPE *)

Защищенные данные

- Calculation & calculation
- InformationForCalculation inputSet
  lculateFunctionals()
- TResultForCalculation outputSet
  lculateFunctionals()

Объявления и описания членов классов находятся в файлах:

- globalizer/method/include/ParallelHookeJeevesMethod.h
- globalizer/method/src/ParallelHookeJeevesMethod.cpp

4.12 Класс Performance

Открытые члены

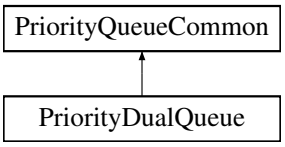
- void Start ()
- double GetTime ()

Объявления и описания членов класса находятся в файле:

- globalizer/method/include/Performance.h

4.13 Класс PriorityDualQueue

Граф наследования:PriorityDualQueue:



Открытые члены

- PriorityDualQueue (int _MaxSize=DefaultQueueSize)
- int GetLocalSize () const
- int GetSize () const
- int GetMaxSize () const
- bool IsLocalEmpty () const
- bool IsLocalFull () const
- bool IsEmpty () const
- bool IsFull () const
- QueueElement * Push (double globalKey, double localKey, void *value)
- QueueElement * PushWithPriority (double globalKey, double localKey, void *value)
- void Pop (double *key, void **value)
- void DeleteByValue (void *value)
- virtual void DeleteElement (QueueElement *item)

  .
- void PopFromLocal (double *key, void **value)
- void Clear ()
- void Resize (int size)
- virtual QueueElement & FindMax ()
- void TrickleUp (QueueElement *item)
- void TrickleDown (QueueElement *item)

Защищенные члены

- void DeleteMinLocalElem ()
- void DeleteMinGlobalElem ()
- void ClearLocal ()
- void ClearGlobal ()

Защищенные данные

- int MaxSize
- int CurLocalSize
- int CurGlobalSize
- MinMaxHeap< QueueElement, _less > * pGlobalHeap
- MinMaxHeap< QueueElement, _less > * pLocalHeap

4.13.1 Методы

4.13.1.1 Clear()

void PriorityDualQueue::Clear () [virtual]

Замещает PriorityQueueCommon.

4.13.1.2 DeleteByValue()

void PriorityDualQueue::DeleteByValue (
void * value) [virtual]

Замещает PriorityQueueCommon.

4.13.1.3 DeleteElement()

void PriorityDualQueue::DeleteElement (
QueueElement * item) [virtual]

.

Замещает PriorityQueueCommon.

4.13.1.4 FindMax()

virtual QueueElement & PriorityDualQueue::FindMax () [inline], [virtual]

Замещает PriorityQueueCommon.

4.13.1.5 GetMaxSize()

int PriorityDualQueue::GetMaxSize () const [virtual]

Замещает PriorityQueueCommon.

4.13.1.6 GetSize()

int PriorityDualQueue::GetSize () const [virtual]

Замещает PriorityQueueCommon.

4.13.1.7 IsEmpty()

bool PriorityDualQueue::IsEmpty () const [virtual]

Замещает PriorityQueueCommon.

4.13.1.8 IsFull()

bool PriorityDualQueue::IsFull () const [virtual]

Замещает PriorityQueueCommon.

4.13.1.9 Pop()

void PriorityDualQueue::Pop (
double * key,
void ** value) [virtual]

Замещает PriorityQueueCommon.

4.13.1.10 Push()

QueueElement * PriorityDualQueue::Push (
double globalKey,
double localKey,
void * value) [virtual]

Замещает PriorityQueueCommon.

4.13.1.11 PushWithPriority()

QueueElement * PriorityDualQueue::PushWithPriority (
double globalKey,
double localKey,
void * value) [virtual]

Замещает PriorityQueueCommon.

4.13.1.12 Resize()

void PriorityDualQueue::Resize (
int size) [virtual]

Замещает PriorityQueueCommon.

4.13.1.13 TrickleDown()

void PriorityDualQueue::TrickleDown (
QueueElement * item) [inline], [virtual]

Замещает PriorityQueueCommon.

4.13.1.14 TrickleUp()

void PriorityDualQueue::TrickleUp (
QueueElement * item) [inline], [virtual]

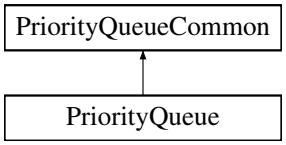
Замещает PriorityQueueCommon.

Объявления и описания членов классов находятся в файлах:

- globalizer/method/include/DualQueue.h
- globalizer/method/src/DualQueue.cpp

4.14 Класс PriorityQueue

Граф наследования:PriorityQueue:



Открытые члены

- PriorityQueue (int _MaxSize=DefaultQueueSize)
- int GetSize () const
- int GetMaxSize () const
- bool IsEmpty () const
- bool IsFull () const
- QueueElement * Push (double globalKey, double localKey, void *value)
- QueueElement * PushWithPriority (double globalKey, double localKey, void *value)
- void Pop (double *key, void **value)
- void DeleteByValue (void *value)
- void DeleteElement (QueueElement *item)

  Удаляет элемент
- void Clear ()
- void Resize (int size)
- QueueElement & FindMax ()
- void TrickleUp (QueueElement *item)
- void TrickleDown (QueueElement *item)

Защищенные члены

- int GetIndOfMinElem ()
- void DeleteMinElem ()

Защищенные данные

- int MaxSize
- int CurSize
- MinMaxHeap< QueueElement, _less > * pMem

4.14.1 Методы

4.14.1.1 Clear()

void PriorityQueue::Clear () [virtual]

Замещает PriorityQueueCommon.

4.14.1.2 DeleteByValue()

void PriorityQueue::DeleteByValue (
void * value) [virtual]

Замещает PriorityQueueCommon.

4.14.1.3 DeleteElement()

void PriorityQueue::DeleteElement (
QueueElement * item) [virtual]

Удаляет элемент

Замещает PriorityQueueCommon.

4.14.1.4 FindMax()

QueueElement & PriorityQueue::FindMax () [virtual]

Замещает PriorityQueueCommon.

4.14.1.5 GetMaxSize()

int PriorityQueue::GetMaxSize () const [virtual]

Замещает PriorityQueueCommon.

4.14.1.6 GetSize()

int PriorityQueue::GetSize () const [virtual]

Замещает PriorityQueueCommon.

4.14.1.7 IsEmpty()

bool PriorityQueue::IsEmpty () const [virtual]

Замещает PriorityQueueCommon.

4.14.1.8 IsFull()

```
bool PriorityQueue::IsFull () const   [virtual]
```

Замещает PriorityQueueCommon.

4.14.1.9 Pop()

```
void PriorityQueue::Pop (
            double * key,
            void ** value)   [virtual]
```

Замещает PriorityQueueCommon.

4.14.1.10 Push()

```
QueueElement * PriorityQueue::Push (
            double globalKey,
            double localKey,
            void * value)   [virtual]
```

Замещает PriorityQueueCommon.

4.14.1.11 PushWithPriority()

```
QueueElement * PriorityQueue::PushWithPriority (
            double globalKey,
            double localKey,
            void * value)   [virtual]
```

Замещает PriorityQueueCommon.

4.14.1.12 Resize()

```
void PriorityQueue::Resize (
            int size)   [virtual]
```

Замещает PriorityQueueCommon.

4.14.1.13 TrickleDown()

```
void PriorityQueue::TrickleDown (
            QueueElement * item)   [inline], [virtual]
```

Замещает PriorityQueueCommon.

4.14.1.14 TrickleUp()

void PriorityQueue::TrickleUp (
QueueElement * item) [virtual]

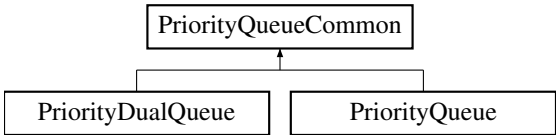
Замещает PriorityQueueCommon.

Объявления и описания членов классов находятся в файлах:

- globalizer/method/include/Queue.h
- globalizer/method/src/Queue.cpp

4.15 Класс PriorityQueueCommon

Граф наследования:PriorityQueueCommon:



Открытые члены

- virtual int GetSize () const =0
- virtual int GetMaxSize () const =0
- virtual bool IsEmpty () const =0
- virtual bool IsFull () const =0
- virtual QueueElement * Push (double globalKey, double localKey, void *value)=0
- virtual QueueElement * PushWithPriority (double globalKey, double localKey, void *value)=0
- virtual void Pop (double *key, void **value)=0
- virtual void DeleteByValue (void *value)=0
- virtual void DeleteElement (QueueElement *item)=0

  Удаляет элемент
- virtual void Clear ()=0
- virtual void Resize (int size)=0
- virtual QueueElement & FindMax ()=0
- virtual void TrickleUp (QueueElement *item)=0
- virtual void TrickleDown (QueueElement *item)=0

4.15.1 Методы

4.15.1.1 DeleteElement()

virtual void PriorityQueueCommon::DeleteElement (
QueueElement * item) [pure virtual]

Удаляет элемент

Замещается в PriorityDualQueue и PriorityQueue.

Объявления и описания членов класса находятся в файле:

- globalizer/method/include/QueueCommon.h

4.16 Класс Process

Открытые члены

- Process (SearchData &data, Task &task)
- double GetSolveTime ()

  Время решения
- void Solve ()

  Запуск решения задачи
- void Reset (SearchData *data, Task *task)

  Сброс параметров процесса
- virtual int GetIterationCount ()

  Получить число испытаний
- int GetNumberOfTrials ()
- virtual Trial * GetOptimEstimation ()

  Получить текущую оценку оптимума
- void InsertPoints (std::vector< Trial * > &points)

  Добавляет испытания в поисковую информацию, при этом обновляя константу Гёльдера и оценку оптимума

Защищенные члены

- void PrintOptimEstimationToFile (Trial OptimEstimation)

  печать текущего минимума в файл
- virtual void PrintOptimEstimationToConsole (Trial OptimEstimation)

  печать текущего минимума на экран
- virtual void OldPrintOptimEstimationToConsole (Trial OptimEstimation)

  Старый вариант печати в консоль
- virtual void PrintResultToFile (Trial OptimEstimation)

  Печать результата в файл
- virtual void BeginIterations ()

  Предварительные настройки, запускается только при первом запуске
- virtual void DoIteration ()

  Одна итерация
- virtual void EndIterations ()

  Окончание работы
- int GetProcLevel ()

  Место в дереве процессов
- bool CheckIsStop (bool IsStop)

  Проверяет остановились ли соседи

Защищенные данные

- bool isPrintOptimEstimation

  Печатать ли выходную информацию
- bool isFirstRun

  Общие данные для всех процессов
- Performance Timer

  Наш таймер
- double duration

время решения задачи
- bool IsOptimumFound

  Решилась ли задача
- Task * pTask

  Задача
- SearchData * pData

  Поисковая информация
- IMethod * pMethod

  Методы для одной итерации
- Evolvent * evolvent

  Указатель на развертку
- Calculation * calculation

  Вычислитель
- std::vector< int > Neighbours
- std::vector< int > functionCalculationCount

  Число вычисленных значений каждой функции
- std::vector< Trial * > * addPoints

  Точки которые будут добавлены после первой итерации

4.16.1 Методы

4.16.1.1 GetIterationCount()

virtual int Process::GetIterationCount () [inline], [virtual]

Получить число испытаний

Возвращает

число испытаний

4.16.1.2 GetOptimEstimation()

virtual Trial * Process::GetOptimEstimation () [inline], [virtual]

Получить текущую оценку оптимума

Возвращает

испытание, соответствующее текущему оптимуму

4.16.1.3 InsertPoints()

void Process::InsertPoints (
std::vector< Trial * > & points)

Добавляет испытания в поисковую информацию, при этом обновляя константу Гёльдера и оценку оптимума

Аргументы

	in	points	точки испытаний, которые будут добавлены

4.16.2 Данные класса

4.16.2.1 evolvent

Evolvent* Process::evolvent [protected]

Указатель на развертку

В зависимости от вида отображения это может быть:

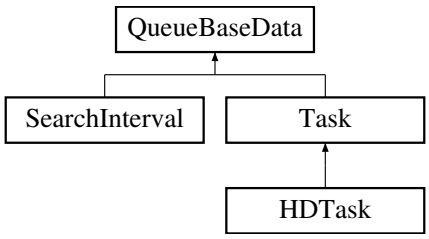
- единственная развертка
- множественная сдвиговая развертка
- множественная вращаемая развертка

Объявления и описания членов классов находятся в файлах:

- globalizer/include/GlobProcess.h
- globalizer/src/GlobProcess.cpp

4.17 Класс QueueBaseData

Граф наследования:QueueBaseData:



Открытые члены

- virtual void SetQueueElementa (QueueElement *q)
- virtual QueueElement * GetQueueElementa ()

Защищенные данные

- QueueElement * queueElementa

  Элемент очереди, хранящий этот интервал

Объявления и описания членов класса находятся в файле:

- globalizer/method/include/BaseInterval.h

4.18 Структура QueueElement

Открытые члены

- QueueElement (double _Key, void *_pValue)
- QueueElement (double _Key, void *_pValue, QueueElement *_pLinkedElement)

Открытые атрибуты

- QueueElement * pLinkedElement
- double Key
- void * pValue

Объявления и описания членов структуры находятся в файле:

- globalizer/method/include/QueueCommon.h

4.19 Класс SearcDataIterator

Открытые члены

- SearcDataIterator & operator++ ()
- SearcDataIterator operator++ (int)
- SearcDataIterator & operator-- ()
- SearcDataIterator operator-- (int)
- operator void * () const
- SearchInterval * operator-> ()
- SearchInterval * operator* () const

Защищенные данные

- SearchData * pContainer
- TreeNode * pObject

Друзья

- class SearchData

Объявления и описания членов классов находятся в файлах:

- globalizer/method/include/SearcDataIterator.h
- globalizer/method/src/SearcDataIterator.cpp

4.20 Класс SearchData

Открытые члены

- SearchData (int _NumOfFuncs, int _MaxSize=DefaultSearchDataSize)
- SearchData (int _NumOfFuncs, int _MaxSize, int _queueSize)
- void Clear ()

  Очищает и дерево и очередь интервалов
- SearchInterval * InsertInterval (SearchInterval &pInterval)

  Новый интервал (по xl)
- void UpdateInterval (SearchInterval &pInterval)

  Обновление интервала (по xl)
- SearchInterval * GetIntervalByX (Trial *x)

  Ищет интервал у которого левой точкой является x.
- SearchInterval * FindCoveringInterval (Trial *x)

  Поиск интервала, в котором содержится x, т.
- SearchInterval * GetIntervalWithMaxR ()

  Получение интервала с максимальной хар-кой.
- SearchInterval * GetIntervalWithMaxLocalR ()

  Получение интервала с максимальной локальной хар-кой.
- SearchInterval * InsertPoint (SearchInterval *coveringInterval, Trial &newPoint, int iteration, int methodDimension)

  Вставка испытания в заданный интервал.
- SearcDataIterator GetIterator (SearchInterval *p)

  Получить итератор
- SearcDataIterator GetBeginIterator ()

  Получить следующий итератор
- void PushToQueue (SearchInterval *pInterval)

  Получить интервал, предыдущий к указанному не относится к итератору, не меняет текущий узел в итераторе SearchInterval* GetPrev(SearchInterval &pInterval);.
- void RefillQueue ()

  Перезаполнение очереди (при ее опустошении или при смене оценки константы Липшица)
- void DeleteIntervalFromQueue (SearchInterval *i)

  Удалить интервал из очереди
- void PopFromGlobalQueue (SearchInterval **pInterval)

  Берет из очереди один интервал
- void PopFromLocalQueue (SearchInterval **pInterval)

  Берет из очереди локальных характеристик один интервал
- void ClearQueue ()

  Очистить очередь интервалов
- void ResizeQueue (int size)

  Изменить размер очереди интервалов
- int GetCount ()

  Возвращает текущее число интервалов в дереве
- void GetBestIntervals (SearchInterval **intervals, int count)

  Получить count лучших интервалов
- void GetBestLocalIntervals (SearchInterval **intervals, int count)

  Получить count лучших локальных интервалов
- std::vector< Trial * > & GetTrials ()

  Получить испытания
- SearchInterval & FindMax ()

Возвращает максимальный элемент без извлечения

- bool IsRecalc ()

  Истина, если нужен пересчет характеристик
- void SetRecalc (bool f)

  Задает нужно ли пересчитывать характеристики
- Trial * GetBestTrial ()

  Лучшая точка, полученная для данной поисковой информации
- void SetBestTrial (Trial *trial)

  Задает лучшую точку
- void TrickleUp (SearchInterval *intervals)

  Всплытие для интервала
- int GetQueueSize ()

  Возвращает размер очереди

Открытые атрибуты

- double M [MaxNumOfFunc]

  Оценки констант Липшица
- double Z [MaxNumOfFunc]

  Минимальные значения функций задачи (для индексного метода)
- double local_r

  Вычисляемое r.

Защищенные члены

- void DeleteTree (TreeNode *pNode)

  Удалить дерево
- unsigned char GetHeight (TreeNode *p)

  Получить высоту
- int GetBalance (TreeNode *p)

  Отбалансировать
- void FixHeight (TreeNode *p)

  Исправить высоту
- TreeNode * RotateRight (TreeNode *p)

  Правый поворот вокруг p.
- TreeNode * RotateLeft (TreeNode *p)

  Левый поворот вокруг p.
- TreeNode * Balance (TreeNode *p)

  Балансировка узла p.
- TreeNode * Maximum (TreeNode *p) const

  Поиск самого левого интервала в поддереве
- TreeNode * Minimum (TreeNode *p) const

  Поиск самого правого интервала в поддереве
- TreeNode * Previous (TreeNode *p) const

  Получение предыдущего за p интервала
- TreeNode * Next (TreeNode *p) const

  Получение следующего за p интервала
- TreeNode * Insert (TreeNode *p, SearchInterval &pInterval)

  Вставка в дерево с корнем p (рекурсивная)
- TreeNode * Find (TreeNode *p, Trial *x) const

Поиск узла с нужным x в дереве с корнем p по левой границе интервала (рекурсивный)

- TreeNode * FindR (TreeNode *p, Trial *x) const

  Поиск узла по правой границе интервала
- TreeNode * FindIn (TreeNode *p, Trial *x) const

  Поиск узла с нужным x по левой и правой границам интервала(рекурсивный) xl() < x < xr.

Защищенные данные

- int NumOfFuncs

  Число функций задачи
- int MaxSize

  Максимальный размер МСП = максимальному числу итераций метода
- int Count

  Текущее число интервалов в дереве
- int CurIndex

  Текущий индекс, используется в итераторе
- TreeNode * pRoot

  Корень дерева
- TreeNode * pCur

  Текущая вершина дерева
- std::stack< TreeNode * > Stack

  Стек для итератора
- PriorityQueueCommon * pQueue

  Очередь характеристик
- std::vector< Trial * > trials

  Список всех точек, для их последующего удаления
- bool recalc

  Истина, если нужен пересчет характеристик
- Trial * BestTrial

  Лучшая точка, полученная для данной поисковой информации

Друзья

- class SearcDataIterator

4.20.1 Методы

4.20.1.1 FindCoveringInterval()

SearchInterval * SearchData::FindCoveringInterval (
Trial * x)

Поиск интервала, в котором содержится x, т.

е. xl() < x < xr нужен для вставки прообразов при использовании множественной развертки

4.20.1.2 GetIntervalWithMaxLocalR()

SearchInterval * SearchData::GetIntervalWithMaxLocalR ()

Получение интервала с максимальной локальной хар-кой.

Интервал берется из очереди. Если очередь пуста, то сначала будет вызван Refill()

4.20.1.3 GetIntervalWithMaxR()

SearchInterval * SearchData::GetIntervalWithMaxR ()

Получение интервала с максимальной хар-кой.

Интервал берется из очереди. Если очередь пуста, то сначала будет вызван Refill()

4.20.1.4 InsertPoint()

```
SearchInterval * SearchData::InsertPoint (
            SearchInterval * coveringInterval,
            Trial & newPoint,
            int iteration,
            int methodDimension)
```

Вставка испытания в заданный интервал.

Нужна для множественной развертки и добавления поисковой информации локального метода возвращает указатель на интервал с левым концом в newPoint

4.20.1.5 PushToQueue()

```
void SearchData::PushToQueue (
            SearchInterval * pInterval)
```

Получить интервал, предыдущий к указанному не относится к итератору, не меняет текущий узел в итераторе SearchInterval* GetPrev(SearchInterval &pInterval);.

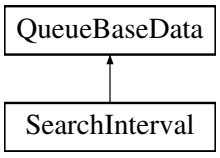
Для работы с очередью характеристик вставка, если новый элемент больше минимального в очереди если при этом очередь полна, то замещение минимального

Объявления и описания членов классов находятся в файлах:

- globalizer/method/include/SearchData.h
- globalizer/method/src/SearchData.cpp

4.21 Класс SearchInterval

Граф наследования:SearchInterval:



Открытые члены

- Extended xl ()
  левая граница интервала
- Extended xr ()
  правая граница интервала
- double zl ()
  значение последнего вычисленного функционала в xl
- int izl ()
  индекс последнего вычисленного фукнционала в xl
- double zr ()
  значение последнего вычисленного функционала в xr
- int izr ()
  индекс последнего вычисленного фукнционала в xr
- double * z ()
  значения вычисленных функционалов в izl, кол-во - izl() + 1
- int discreteValuesIndex ()
  Индекс значения дискретного параметра
- virtual bool operator== (SearchInterval &p)
- virtual bool operator> (SearchInterval &p)
- virtual bool operator< (SearchInterval &p)
- void CreatePoint ()
- SearchInterval (const SearchInterval &p)

Открытые члены унаследованные от QueueBaseData

- virtual void SetQueueElementa (QueueElement *q)
- virtual QueueElement * GetQueueElementa ()

Открытые атрибуты

- Trial * LeftPoint
  Левая точка интервала
- Trial * RightPoint
  Правая точка интервала
- double delta
  "гельдеровская" длина
- int ind
  номер итерации
- int K
  число "вложенных" итераций
- double R
  характеристика интервала (xl, xr)
- double locR
  локальная характеристика интервала (xl, xr)
- TreeNode * treeNode
  Элемент дерева хранящий этот интервал

Дополнительные унаследованные члены

Защищенные данные унаследованные от QueueBaseData

- QueueElement * queueElementa
  Элемент очереди, хранящий этот интервал

Объявления и описания членов классов находятся в файлах:

- globalizer/method/include/SearchInterval.h
- globalizer/method/src/SearchInterval.cpp

4.22 Класс SearchIntervalFactory

Открытые статические члены

- static SearchInterval * CreateSearchInterval (SearchInterval &interval)
- static SearchInterval * CreateSearchInterval ()

Объявления и описания членов классов находятся в файлах:

- globalizer/method/include/SearchIntervalFactory.h
- globalizer/method/src/SearchIntervalFactory.cpp

4.23 Структура SearchIteration

Открытые атрибуты

- int IterationCount
  номер выполненных итераций
- std::vector< Trial * > pCurTrials
  Указатель на массив испытаний, выполняемых на данной итерации

4.23.1 Данные класса

4.23.1.1 pCurTrials

std::vector<Trial*> SearchIteration::pCurTrials

Указатель на массив испытаний, выполняемых на данной итерации

В зависимости от типа метода в данном массиве может быть:

- одна компонпонента (последовательный алгоритм)
- #NumPoints компонент (параллельный синхронный и асинхронный алгоритмы)

Объявления и описания членов структуры находятся в файле:

- globalizer/method/include/SearchIteration.h

4.24 Структура SolutionResult

Результаты работы системы

#include <SolutionResult.h>

Открытые атрибуты

- Trial * BestTrial
  Лучшая итерация, полученная при данном запуске метода
- int IterationCount
  число выполненных итераций
- int TrialCount
  количество испытаний

4.24.1 Подробное описание

Результаты работы системы

Объявления и описания членов структуры находятся в файле:

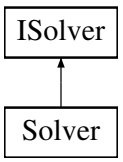
- globalizer/include/SolutionResult.h

4.25 Класс Solver

Базовые классы для решения задач глобальной оптимизации

#include <Solver.h>

Граф наследования:Solver:



Открытые члены

- Solver (IProblem *problem)
- virtual int Solve ()

  Решение задачи по умолчанию
- virtual int Solve (Task *task)

  Решение подзадачи с указанными параметрами
- void SetProblem (IProblem *problem)

  Задает задачу для решения
- IProblem * GetProblem ()

  Возвращает решаемую задачу
- SolutionResult * GetSolutionResult ()

  Возвращает полученное решение
- virtual void SetPoint (std::vector< Trial * > &points)

  Добавляет точки испытаний
- virtual std::vector< Trial * > & GetAllPoint ()

  Возвращает все имеющиеся точки испытаний
- Task * GetTask ()

  Возвращает задачу решателя
- SearchData * GetData ()

  Возвращает поисковую информацию

Защищенные члены

- virtual void ClearData ()

  Очистить данные
- virtual void InitAutoPrecision ()

  Инициализация чисел с расширенной точностью
- virtual int CreateProcess ()

  Создание процесса и всего остального
- int CheckParameters ()

  Проверяет что параметры солвера соответствуют решаемой задаче
- void MpiCalculation ()

  Решатель производящий вычисление при параллельном АГП (распараллеливание по точкам) на MPI.
- void AsyncCalculation ()

  Решатель используемый при ассинхронной схеме

Защищенные данные

- Process * mProcess

  Процесс решающий задачу
- IProblem * mProblem

  Задача
- Task * pTask

  Общее описание задачи
- bool isExternalTask

  Задача пораждена в Solver или пришла извне
- SearchData * pData

  База данных(поисковая информация)
- SolutionResult * result

  Результат работы системы
- std::vector< Trial * > * addPoints

  Точки которые будут добавлены после первой итерации

4.25.1 Подробное описание

Базовые классы для решения задач глобальной оптимизации

4.25.2 Методы

4.25.2.1 CreateProcess()

int Solver::CreateProcess () [protected], [virtual]

Создание процесса и всего остального

Создание задачи (Task) // перенести в фабрику

Создаём данные для поисковой информации

4.25.2.2 GetAllPoint()

std::vector< Trial * > & Solver::GetAllPoint () [virtual]

Возвращает все имеющиеся точки испытаний

Замещает ISolver.

4.25.2.3 GetSolutionResult()

SolutionResult * Solver::GetSolutionResult ()

Возвращает полученное решение

best point

4.25.2.4 InitAutoPrecision()

void Solver::InitAutoPrecision () [protected], [virtual]

Инициализация чисел с расширенной точностью

Функция автоматически выбирает используемый тип данных в зависимости от размерности решаемой задачи и плотности развертки

4.25.2.5 MpiCalculation()

void Solver::MpiCalculation () [protected]

Решатель производящий вычисление при параллельном АГП (распараллеливание по точкам) на MPI.

Входные данные для вычислителя, формируются в CalculateFunctionals()

Выходные данные вычислителя, обрабатываются в CalculateFunctionals()

4.25.2.6 SetPoint()

void Solver::SetPoint (
std::vector< Trial * > & points) [virtual]

Добавляет точки испытаний

Замещает ISolver.

4.25.2.7 Solve()

int Solver::Solve () [virtual]

Решение задачи по умолчанию

curr_child = parameters.parallel_tree.ProcChild[i];!!!!!

Замещает ISolver.

Объявления и описания членов классов находятся в файлах:

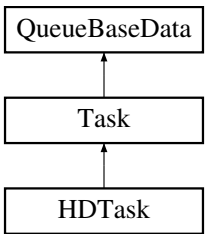
- globalizer/include/Solver.h
- globalizer/src/Solver.cpp

4.26 Класс Task

Класс, инкапсулирующий информацию о задаче оптимизации.

#include <Task.h>

Граф наследования:Task:



Открытые члены

- Task (IProblem *_problem, int _ProcLevel)

  Конструктор.
- Task ()

  Конструктор по умолчанию.
- virtual ~Task ()

  Деструктор.
- virtual Task * Clone ()

  Создает клон текущего объекта.
- virtual Task * CloneWithNewData ()

  Создает клон текущего объекта с новыми данными (в данной реализации эквивалентно Clone).
- virtual void Init (IProblem *_problem, int _ProcLevel)

  Инициализирует объект данными задачи.
- virtual int GetN () const

  Возвращает общую размерность задачи.
- virtual const double * GetA () const

  Возвращает левую границу области поиска.
- virtual const double * GetB () const

  Возвращает правую границу области поиска.
- virtual double GetOptimumValue () const

  Возвращает априори известное значение глобального минимума.
- virtual void resetOptimumPoint ()

  Обновляет координаты точки глобального минимума из объекта #IProblem.
- virtual const double * GetOptimumPoint () const

  Возвращает априори известные координаты точки глобального минимума.
- virtual bool GetIsOptimumValueDefined () const

  Проверяет, известно ли для задачи значение глобального минимума.
- virtual bool GetIsOptimumPointDefined () const

  Проверяет, известны ли для задачи координаты глобального минимума.
- virtual IProblem * getProblem ()

  Возвращает указатель на текущую задачу.
- virtual int GetNumOfFunc () const

  Возвращает число функций (ограничения и критерии).
- virtual void SetNumofFunc (int nf)

  Задает число функций.
- int GetProcLevel ()

  Возвращает уровень процесса в дереве процессов.
- virtual int GetNumOfFuncAtProblem () const

  Возвращает число функций в исходной задаче.
- virtual double CalculateFuncs (const double *y, int fNumber)

  Вычисляет значение функции с номером fNumber в точке y.
- virtual void CalculateFuncsInManyPoints (double *y, int fNumber, int numPoints, double *values)

  Вычисляет значения функции в нескольких точках (для GPU).
- virtual int GetNumberOfDiscreteVariable ()

  Возвращает число дискретных параметров.
- virtual int GetNumberOfValues (int discreteVariable)

  Возвращает число допустимых значений для дискретного параметра.
- virtual int GetAllDiscreteValues (int discreteVariable, double *values)

  Определяет все допустимые значения дискретного параметра.
- virtual bool IsPermissibleValue (double value, int discreteVariable)

Проверяет, является ли значение допустимым для дискретного параметра.

- virtual double * getMin ()

  Возвращает минимальные значения функций (для многокритериальной оптимизации).
- virtual double * getMax ()

  Возвращает максимальные значения функций (для многокритериальной оптимизации).
- virtual bool IsInit ()

  Проверяет, был ли объект инициализирован.
- virtual bool IsLeaf ()

  Проверяет, является ли задача листом в дереве процессов.
- virtual void CopyPoint (double *y, Trial *point)

  Копирует координаты точки из массива, согласно имеющимся правилам

Открытые члены унаследованные от QueueBaseData

- virtual void SetQueueElementa (QueueElement *q)
- virtual QueueElement * GetQueueElementa ()

Открытые атрибуты

- int num

Защищенные данные

- double A [MaxDim]

  Левая граница области поиска
- double B [MaxDim]

  Правая граница области поиска
- int NumOfFunc

  Число функционалов (последний - критерий)
- IProblem * pProblem

  Указатель на саму задачу оптимизации
- double OptimumValue

  Оптимальное значение целевой функции (определено, если известно из задачи)
- double OptimumPoint [MaxDim]

  Координаты глобального минимума целевой функции (определено, если известно)
- bool IsOptimumValueDefined

  true, если в задаче известно оптимальное значение критерия
- bool IsOptimumPointDefined

  true, если в задаче известна точка глобального минимума
- int ProcLevel

  Уровень процесса в дереве процессов
- bool isInit

  Флаг, указывающий, был ли класс инициализирован

Защищенные данные унаследованные от QueueBaseData

- QueueElement * queueElementa

  Элемент очереди, хранящий этот интервал

4.26.1 Подробное описание

Класс, инкапсулирующий информацию о задаче оптимизации.

Task является оберткой над интерфейсом #IProblem и предоставляет доступ к параметрам задачи, таким как границы поиска, число функций, известные оптимальные значения, а также предоставляет методы для вычисления значений функций.

4.26.2 Конструктор(ы)

4.26.2.1 Task() [1/2]

```
Task::Task (
            IProblem * _problem,
            int _ProcLevel)
```

Конструктор.

Аргументы

	in	_problem	Указатель на объект задачи оптимизации.
	in	_ProcLevel	Уровень процесса в дереве процессов.

4.26.2.2 Task() [2/2]

```
Task::Task ()
```

Конструктор по умолчанию.

Создает неинициализированный объект.

4.26.3 Методы

4.26.3.1 CalculateFuncs()

```
double Task::CalculateFuncs (
            const double * y,
            int fNumber)   [virtual]
```

Вычисляет значение функции с номером fNumber в точке y.

Аргументы

	in	y	Указатель на массив с координатами точки.
	in	fNumber	Номер вычисляемой функции.

Возвращает

Значение функции.

Переопределяется в HDTask.

4.26.3.2 CalculateFuncsInManyPoints()

```
void Task::CalculateFuncsInManyPoints (
                double * y,
                int fNumber,
                int numPoints,
                double * values)   [virtual]
```

Вычисляет значения функции в нескольких точках (для GPU).

Работает только если задача является наследником #IGPUProblem.

Аргументы

	in	y	Указатель на массив координат точек.
	in	fNumber	Номер функции.
	in	numPoints	Количество точек.
	out	values	Массив для записи результатов.

Переопределяется в HDTask.

4.26.3.3 Clone()

```
Task * Task::Clone ()   [virtual]
```

Создает клон текущего объекта.

Возвращает

Указатель на новый объект Task.

Переопределяется в HDTask.

4.26.3.4 CloneWithNewData()

```
Task * Task::CloneWithNewData ()   [virtual]
```

Создает клон текущего объекта с новыми данными (в данной реализации эквивалентно Clone).

Возвращает

Указатель на новый объект Task.

4.26.3.5 CopyPoint()

```
void Task::CopyPoint (
                double * y,
                Trial * point)   [virtual]
```

Копирует координаты точки из массива, согласно имеющимся правилам

Аргументы

	in	y	имеющисся координаты.
	out	point	точка назначения.

Возвращает

true, если значение допустимо, иначе false.

Переопределяется в HDTask.

4.26.3.6 GetA()

```
const double * Task::GetA () const    [virtual]
```

Возвращает левую границу области поиска.

Возвращает

Указатель на массив со значениями левой границы.

Переопределяется в HDTask.

4.26.3.7 GetAllDiscreteValues()

```
int Task::GetAllDiscreteValues (
            int discreteVariable,
            double * values)    [virtual]
```

Определяет все допустимые значения дискретного параметра.

Аргументы

	in	discreteVariable	Индекс дискретной переменной.
	out	values	Массив, в который будут сохранены значения.

Возвращает

Код ошибки (#IProblem::OK или #IProblem::ERROR).

4.26.3.8 GetB()

```
const double * Task::GetB () const    [virtual]
```

Возвращает правую границу области поиска.

Возвращает

Указатель на массив со значениями правой границы.

Переопределяется в HDTask.

4.26.3.9 GetIsOptimumPointDefined()

bool Task::GetIsOptimumPointDefined () const [virtual]

Проверяет, известны ли для задачи координаты глобального минимума.

Возвращает

true, если координаты известны, иначе false.

4.26.3.10 GetIsOptimumValueDefined()

bool Task::GetIsOptimumValueDefined () const [virtual]

Проверяет, известно ли для задачи значение глобального минимума.

Возвращает

true, если значение известно, иначе false.

4.26.3.11 getMax()

double * Task::getMax () [virtual]

Возвращает максимальные значения функций (для многокритериальной оптимизации).

Возвращает

NULL в текущей реализации.

4.26.3.12 getMin()

double * Task::getMin () [virtual]

Возвращает минимальные значения функций (для многокритериальной оптимизации).

Возвращает

NULL в текущей реализации.

4.26.3.13 GetN()

int Task::GetN () const [virtual]

Возвращает общую размерность задачи.

Возвращает

Размерность задачи.

4.26.3.14 GetNumberOfDiscreteVariable()

int Task::GetNumberOfDiscreteVariable () [virtual]

Возвращает число дискретных параметров.

Дискретные параметры всегда последние в векторе y.

Возвращает

Число дискретных переменных или 0, если задача не является целочисленной.

4.26.3.15 GetNumberOfValues()

int Task::GetNumberOfValues (
int discreteVariable) [virtual]

Возвращает число допустимых значений для дискретного параметра.

Аргументы

	in	discreteVariable	Индекс дискретной переменной.

Возвращает

Число значений или -1, если задача не является целочисленной.

4.26.3.16 GetNumOfFunc()

int Task::GetNumOfFunc () const [virtual]

Возвращает число функций (ограничения и критерии).

Возвращает

Число функций.

4.26.3.17 GetNumOfFuncAtProblem()

int Task::GetNumOfFuncAtProblem () const [virtual]

Возвращает число функций в исходной задаче.

Возвращает

Число функций.

4.26.3.18 GetOptimumPoint()

const double * Task::GetOptimumPoint () const [virtual]

Возвращает априори известные координаты точки глобального минимума.

Перед первым вызовом рекомендуется вызвать resetOptimumPoint().

Возвращает

Указатель на массив с координатами точки глобального минимума.

Переопределяется в HDTask.

4.26.3.19 GetOptimumValue()

double Task::GetOptimumValue () const [virtual]

Возвращает априори известное значение глобального минимума.

Возвращает

Значение глобального минимума.

4.26.3.20 getProblem()

IProblem * Task::getProblem () [virtual]

Возвращает указатель на текущую задачу.

Возвращает

Указатель на объект #IProblem.

4.26.3.21 GetProcLevel()

int Task::GetProcLevel ()

Возвращает уровень процесса в дереве процессов.

Возвращает

Уровень процесса.

4.26.3.22 Init()

void Task::Init (
IProblem * _problem,
int _ProcLevel) [virtual]

Инициализирует объект данными задачи.

Аргументы

	in	_problem	Указатель на объект задачи оптимизации.
	in	_ProcLevel	Уровень процесса в дереве процессов.

4.26.3.23 IsInit()

```
bool Task::IsInit ()  [virtual]
```

Проверяет, был ли объект инициализирован.

Возвращает

true, если объект инициализирован, иначе false.

4.26.3.24 IsLeaf()

```
bool Task::IsLeaf ()  [virtual]
```

Проверяет, является ли задача листом в дереве процессов.

Возвращает

true, если ProcLevel не равен 0, иначе false.

4.26.3.25 IsPermissibleValue()

```
bool Task::IsPermissibleValue (
            double value,
            int discreteVariable)  [virtual]
```

Проверяет, является ли значение допустимым для дискретного параметра.

Аргументы

	in	value	Проверяемое значение.
	in	discreteVariable	Индекс дискретной переменной.

Возвращает

true, если значение допустимо, иначе false.

4.26.3.26 SetNumofFunc()

```
void Task::SetNumofFunc (
            int nf)  [virtual]
```

Задает число функций.

Аргументы

	in	nf	Новое число функций.

Объявления и описания членов классов находятся в файлах:

- globalizer/method/include/Task.h
- globalizer/method/src/Task.cpp

4.27 Класс TaskFactory

Открытые статические члены

- static Task * CreateTask (IProblem *_problem, int _ProcLevel)
- static Task * CreateTask ()
- static Task * CreateTask (Task *t)

Статические открытые данные

- static int num = 0
- static std::vector< int > permutations

Объявления и описания членов классов находятся в файлах:

- globalizer/method/include/TaskFactory.h
- globalizer/method/src/TaskFactory.cpp

4.28 Класс TreeNode

Открытые члены

- TreeNode (SearchInterval &p)

Открытые атрибуты

- SearchInterval * pInterval

  Интервал поиска
- unsigned char Height

  Высота дерева
- TreeNode * pLeft

  Левая ветвь дерева
- TreeNode * pRight

  Правая ветвь дерева
- TreeNode * pParent

  Родительский узел

Объявления и описания членов классов находятся в файлах:

- globalizer/method/include/TreeNode.h
- globalizer/method/src/TreeNode.cpp

4.29 Класс Trial

Открытые члены

- Trial ()

  Создает не вычесленное испытание в координате x=0.
- Trial (const Trial &trial)

  Копия точки
- virtual Trial * Clone ()

  Создает копию точки
- void SetX (Extended d)

  Задаем координату в одномерном пространстве перещет в многомерное не производится!
- virtual Trial & operator= (Extended d)

  Присвоение координаты точки в одномерном прогстранстве
- virtual Extended X ()

  Возвращает координату точки
- virtual double GetFloor ()

  Возвращает левую границу отрезка == 0.
- virtual double GetValue ()

  Возвращает значение испытания (с учетом индексной схемы)
- virtual Trial * GetLeftPoint ()

  Возврящает соседнюю с лева точку
- virtual Trial * GetRightPoint ()

  Возврящает соседнюю с права точку
- virtual Trial & operator= (const Trial &trial)

  Копирование точки
- virtual bool operator== (Trial &t)

  Сравнение точек в одномерном пространстве
- virtual bool operator> (Trial &t)

  Сравнение точек в одномерном пространстве
- virtual bool operator< (Trial &t)

  Сравнение точек в одномерном пространстве

Открытые атрибуты

- int discreteValuesIndex

  Индекс значения дискретного параметра
- double y [MaxDim]

  точка в многомерном пространстве
- double FuncValues [MaxNumOfFunc]

  значения функций задачи, вычисленные до первого нарушенного
- int index

  индекс точки
- int K

  число "вложенных" итераций
- int lowAndUpPoints

  Количество нужных для локального метода точек (обновляется только в случае потенциального локального минимума)
- Task * generatedTask

  Задача порождаемая точкой при адаптивной схеме редукции

- SearchInterval * leftInterval

  Интервал слева, эта точка для него правая
- SearchInterval * rightInterval

  Правый интервал, эта точка для него левая(главная)
- int TypeColor

  Цвет рисования точки

Защищенные данные

- Extended x

  точка на одномерном отрезке

4.29.1 Методы

4.29.1.1 SetX()

void Trial::SetX (
Extended d)

Задаем координату в одномерном пространстве перещет в многомерное не производится!

Аргументы

in	d	-	новая координата

Объявления и описания членов классов находятся в файлах:

- globalizer/method/include/Trial.h
- globalizer/method/src/Trial.cpp

4.30 Класс TrialFactory

Открытые статические члены

- static Trial * CreateTrial ()
- static Trial * CreateTrial (const OBJECTIV_TYPE *startPoint)
- static Trial * CreateTrial (Trial *point)

Объявления и описания членов класса находятся в файле:

- globalizer/method/include/TrialFactory.h

Глава 5

Файлы

5.1 Common.h

```
00001
00002 //                                                      //
00003 //           LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD          //
00004 //                                                      //
00005 //                  Copyright (c) 2015 by UNN.                  //
00006 //                    All Rights Reserved.                      //
00007 //                                                      //
00008 //  File:      common.h                                     //
00009 //                                                      //
00010 //  Purpose:   Common Header file                               //
00011 //                                                      //
00012 //  Author(s): Sysoyev A., Barkalov K.                          //
00013 //                                                      //
00015
00016 #ifndef __COMMON_H__
00017 #define __COMMON_H__
00018
00019 #include "Defines.h"
00020
00021 /*
      ============================================================================================
      *\
00022 **  Constants                                                 **
00023 \*
      ============================================================================================
      */
00024
00025 //const int MaxPathLength = 512
00026 const int MaxNumOfTaskLevels = 5;
00027 const int MaxNumOfFunc = MAX_NUMBER_OF_FUNCTION;
00028 const int MaxDim = MAX_TRIAL_DIMENSION;
00029 const int MaxNumOfGlobMinima = MAX_NUM_MIN;
00030 const int MaxM = 20;
00031 const int MaxL = 10;
00032
00033 const double MaxDouble = 1.7e308;
00034 const double MinDouble = -1.7e308;
00035 const double AccuracyDouble = 1.0e-8;
00036
00037 const int DefaultQueueSize =
      32767;//1023;//65535;//32767;//16383;//16777215;//8388607;//1048575;//8388607;//524287; //262143; // должно быть
      равно 2^k - 1
00038 const int DefaultSearchDataSize = 100000;
00039
00040 const int TagChildStartSolve = 101;
00041 const int TagChildSolved = 102;
00042 const int ChildStopMsg = -101;
00043
00044 /*
      ============================================================================================
      *\
00045 **  Types                                                     **
00046 \*
      ============================================================================================
      */
00047 class Process;
00048 #ifdef WIN32
```

```
00049 typedef void(__cdecl* tIterationHandler)(Process* pProcess);
00050 typedef OBJECTIV_TYPE(__cdecl* tFunction)(const OBJECTIV_TYPE*);
00051 #else
00052 typedef void(*tIterationHandler)(Process* pProcess);
00053 typedef OBJECTIV_TYPE(*tFunction)(const OBJECTIV_TYPE*);
00054 #endif
00055
00056
00063 enum IterationType
00064 {
00066   Global,
00068   Local
00069 };
00070
00071
00072 enum EParameterType
00073 {
00074   Pbool,
00075   Pint,
00076   Pdouble,
00077   Pstring,
00078   PETypeMethod,
00079   PETypeCalculation,
00080   PELocalMethodScheme,
00081   PESeparableMethodType,
00082   PEStopCondition,
00083   PETypeProcess,
00084   PETypeSolver,
00085   PEMapType,
00086   Pints,
00087   Pdoubles,
00088   Pflag
00089 };
00090
00091 enum ETypeMethod
00092 {
00093   StandartMethod,
00094   IntegerMethod
00095 };
00096
00097 enum ESeparableMethodType
00098 {
00099   Off,
00100   GridSearch,
00101   GlobalMethod
00102 };
00103
00104 enum ELocalMethodScheme
00105 {
00107   None,
00109   FinalStart,
00111   UpdatedMinimum
00112 };
00113
00114 enum EStopCondition
00115 {
00116   Accuracy,
00117   OptimumVicinity,
00118   OptimumVicinity2,
00119   OptimumValue,
00120   AccuracyWithCheck,
00121   InLocalArea
00122 };
00123
00124 enum ETypeCalculation
00125 {
00127   OMP,
00129   CUDA,
00131   MPI_calc,
00133   AsyncMPI,
00135   OneApi
00136 };
00137
00138 enum ETypeProcess
00139 {
00141   SynchronousProcess
00142 };
00143
00147 enum ETypeSolver
00148 {
00150   SingleSearch,
00152   HDSearch
00153 };
00154
00156 enum EMapType
00157 {
00159   mpBase
```

```
00160 };
00161
00163 enum ETypeDistributionStartingPoints
00164 {
00166   Evenly
00167 };
00168
00169 enum ETypeLocalMethod
00170 {
00172   HookeJeeves,
00174   LeastSquareMethod,
00176   ParallelHookeJeeves
00177 };
00178
00179 enum ETypeStartLocalMethod
00180 {
00182   AnyPoints,
00184   EqualNumberOfPoints
00185 };
00186
00187 enum ETypeAddLocalPoint
00188 {
00190   RegularPoints,
00192   NotTakenIntoAccountInStoppingCriterion,
00194   IntegratedOnePoint,
00196   IntegratedAllPoint,
00198   IntegratedBestPath
00199 };
00200
00202 enum ETypeLocalMinInterval
00203 {
00205   NPoints,
00207   DecisionTrees,
00209   AllMinimum
00210 };
00211
00213 enum ELocalTuningType
00214 {
00216   WithoutLocalTuning,
00218   MiniMax,
00220   Adaptive,
00222   AdaptiveMiniMax
00223 };
00224
00225 #endif
00226 // - end of file ----------------------------------------------------------------------------
```

5.2 Defines.h

```
00001 #ifndef __DEFINES_H__
00002 #define __DEFINES_H__
00003
00004
00005 /*
      ================================================================================
      *\
00006 **  Constants                                                   **
00007 \*
      ================================================================================
      */
00008
00010 #ifdef GLOBALIZER_MAX_DIMENSION
00011 #define MAX_TRIAL_DIMENSION GLOBALIZER_MAX_DIMENSION
00012 #else
00013 #define MAX_TRIAL_DIMENSION 200
00014 #endif
00015
00017 #ifdef GLOBALIZER_MAX_Number_Of_Function
00018 #define MAX_NUMBER_OF_FUNCTION GLOBALIZER_MAX_Number_Of_Function
00019 #else
00020 #define MAX_NUMBER_OF_FUNCTION 20
00021 #endif
00022
00023
00024 // Максимальное количество глобальных минимумов
00025 #define MAX_NUM_MIN 20
00026 // Если константа определена то на GPU будет использоваться double иначе float
00027 #define CUDA_VALUE_DOUBLE_PRECISION
00029
00030 #define CPU_VALUE_DOUBLE_PRECISION
00031
00038 #define _M_ZERO_LEVEL 1e-12
00039
```

```
00040
00041 /*
      ==========================================================================
      *\
00042 **  Types                                                   **
00043 \*
      ==========================================================================
      */
00045 #ifdef CUDA_VALUE_DOUBLE_PRECISION
00046 #define CUDA_VALUE double
00047 #else
00048 #define CUDA_VALUE float
00049 #endif
00051 #ifdef CPU_VALUE_DOUBLE_PRECISION
00052 #define CPU_VALUE double
00053 #else
00054 #define CPU_VALUE float
00055 #endif
00057 #define ints int*
00059 #define doubles double*
00061 #define FLAG bool
00062
00063 /*
      ==========================================================================
      *\
00064 **  Defines                                                 **
00065 \*
      ==========================================================================
      */
00066
00067 /*
      ==========================================================================
      *\
00068 **  Параметры спецификаторы и прочее,  для разных версий компиляторов      **
00069 \*
      ==========================================================================
      */
00070
00071 #ifdef _GPU_CUDA_
00072
00073 #define perm 1
00074 #define SPECIFIER __shared__
00075 #define ARRAY_SPECIFIER __constant__
00076 #define concatenation cuda
00077 #define F_DEVICE __device__
00078 #define parameter_const
00079 #define OBJECTIV_TYPE CUDA_VALUE
00080 #define GET_FUNCTION_PARAMETERS OBJECTIV_TYPE* x, OBJECTIV_TYPE* f
00081 #define FUNCTION_CALCOLATION_PREF (x)
00082
00083 #define GKLS_VARIABLES_SPECIFIER __shared__
00084 //Формирование констант правильной точности
00085 #ifdef CUDA_VALUE_DOUBLE_PRECISION
00086 #define PRECISION(x) x##0
00087 #else
00088 #define PRECISION(x) x##f
00089 #endif
00090
00091 #else
00092
00093
00094 #define SPECIFIER extern
00095 #define ARRAY_SPECIFIER extern
00096 #define concatenation
00097 #define F_DEVICE inline
00098 #define OBJECTIV_TYPE CPU_VALUE
00099 #define GET_FUNCTION_PARAMETERS tFunction* f
00100 #define FUNCTION_CALCOLATION_PREF
00101
00102 #define GKLS_VARIABLES_SPECIFIER extern
00103 //Формирование констант правильной точности
00104 #ifdef CPU_VALUE_DOUBLE_PRECISION
00105 #define PRECISION(x) x##0
00106 #else
00107 #define PRECISION(x) x##f
00108 #endif
00109
00110 #endif
00111
00112 /*
      ==========================================================================
      *\
00113 **  Служебные                                               **
00114 \*
      ==========================================================================
      */
00115 #ifndef GLOBALIZER_MAX
```

```
00116 #define GLOBALIZER_MAX(a,b) ((a) > (b) ? a : b)
00117 #endif
00118
00119 #ifndef GLOBALIZER_MIN
00120 #define GLOBALIZER_MIN(a,b) ((a) < (b) ? a : b)
00121 #endif
00122
00124 #define CAT(x, y) x##y
00126 #define CAT4(a, b, c, d) a##b##c##d
00128 #define CONCATENATION2(name, console) CAT(console,name)
00132 #define CONCATENATION(name) CONCATENATION2(name, concatenation)
00134 #define CAT_COM(x) //##x
00135
00137 #define ParType(type) P##type
00139 #define LinkParameter(name) link##name
00141 #define ComParameter(name) com##name
00143 #define HelpParameter(name) help##name
00145 #define IncParameter(name) inc##name
00147 #define make_str(bar) # bar
00149 #define IsChange(name) IS_##name
00150
00152 #define InitParameter(type, name, defVal, com, help, sizeVal)                          \
00153 IsChange(name) = false;                                                                \
00154 Inc(ParType(name), ParType(type), LinkParameter(name), com,                            \
00155   HelpParameter(name), help, (void*)(&name), make_str(defVal),                         \
00156   make_str(name), &IsChange(name), IncParameter(name), make_str(name), sizeVal);
00157
00158 #define OWNER_NAME Owner
00159
00161 #define InitOption(name, defVal, com, help, sizeVal)                                   \
00162   InitializationOption((BaseProperty<OWNER_NAME>*)(&name), make_str(name), make_str(defVal), com, help,
      sizeVal);
00163
00164 /*
      ============================================================================
      *\
00165 **  Директиы для объявления переменных                                        **
00166 \*
      ============================================================================
      */
00167
00173 #define VARIABLES(name, type, specifier) specifier type name
00177 #define GKLS_VARIABLES(name, type) VARIABLES(name, type, GKLS_VARIABLES_SPECIFIER)
00181 #define CONSTANT_VARIABLES(name, type) SPECIFIER type name
00183 #define NEW_FUNC_DEF(name) CONCATENATION(name##_func_def())
00185 #define STATIC_ARRAY(type, name, count) type name[count]
00189 #define NEW_ARRAY(type, name, count) ARRAY_SPECIFIER STATIC_ARRAY(type, name, count)
00193 #define NEW_ARRAY_MAX_SIZE(name) NEW_ARRAY(OBJECTIV_TYPE, name, MAX_TRIAL_DIMENSION)
00197 #define FLOAT_VARIABLES(name) SPECIFIER OBJECTIV_TYPE name
00198
00199
00206 #define NEW_VARIABLES(type, name, val) CONSTANT_VARIABLES(name, type);\
00207  F_DEVICE type NEW_FUNC_DEF(name)\
00208   {\
00209   name = val;\
00210 return name;\
00211 }\
00212
00214 #define CreateParameter(type, name)       \
00215 public: type name;                        \
00216 public: FLAG IsChange(name);              \
00217 protected: int IncParameter(name);        \
00218 protected: EParameterType ParType(name);  \
00219 protected: std::string LinkParameter(name);   \
00220 protected: std::string HelpParameter(name);
00221
00223 #define BasicMethods(ClassType, Type)                                                  \
00224   virtual void operator =(Type data) {TypedProperty<Type, Owner>::operator=(data);}              \
00225   virtual void operator=(ClassType<Owner>& data) {ParameterProperty<Type, Owner>::operator=(data);}     \
00226   virtual void operator = (ClassType<Owner>* data) {ParameterProperty<Type, Owner>::operator=(data);}  \
00227   virtual std::string ToString() {return operator std::string();}                      \
00228   virtual void FromString(std::string val) {operator = (val);}                        \
00229   virtual void operator = (BaseProperty<Owner>& data) { ParameterProperty<Type,
      Owner>::operator=((ParameterProperty<Type, Owner>*)(& data)); } \
00230   virtual void operator=(char* data) {operator = (std::string(data));}
00231
00232
00233
00238 #define PROPERTY(T, N)     \
00239   T Get ## N() const;     \
00240   void Set ## N(T value);
00241
00242 /*
      ============================================================================
      *\
00243 **  Прочие константы                                                          **
00244 \*
```

```
    ========================================================================
    */
00245
00246 #define PI PRECISION(3.14159265359)
00247
00248
00249
00250
00251
00252
00253
00254 #endif
```

5.3 Globalizer.h

```
00001
00002 //                                                      //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD          //
00004 //                                                      //
00005 //                    Copyright (c) 2025 by UNN.                //
00006 //                      All Rights Reserved.                  //
00007 //                                                      //
00008 //  File:      Globalizer.h                                 //
00009 //                                                      //
00010 //  Purpose:   Console version of Globalizer system                 //
00011 //                                                      //
00012 //  Author(s): Lebedev I.                                   //
00013 //                                                      //
00015
00016 #pragma once
00017
00018
00019 #ifndef _CRT_SECURE_NO_WARNINGS
00020 #define _CRT_SECURE_NO_WARNINGS
00021 #endif
00022
00023 #include <algorithm>
00024 #include <fstream>
00025 #include <vector>
00026 #include <string>
00027 #include <cmath>
00028 #include <iostream>
00029
00030 #include "Solver.h"
00031 #include "GlobalizerProblem.h"
00032 #include "HDSolver.h"
00033
00034 #ifdef _GLOBALIZER_BENCHMARKS
00035 #include "IGlobalOptimizationProblem.h"
00036 #include "GlobalOptimizationProblemManager.h"
00037 #endif // _GLOBALIZER_BENCHMARKS
00038
00039 #ifndef WIN32
00040 #include <unistd.h>
00041 #endif
00042
00059 void GlobalizerInitialization(int argc=0, char* argv[]=nullptr,
00060   bool isMPIInit = false, bool isPrintParameters = false,
00061   std::string mLogFileName = "", int processCount = -1,
00062   int processNumber = -1, bool isPrintToFile = false,
00063   std::string* errorsName = nullptr, int* errorsCode = nullptr,
00064   int errorsCount = 0);
00065
00070 void CreateCurentProblemsParameters(int argc, char* argv[]);
```

5.4 GlobProcess.h

```
00001
00002 //                                                      //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD          //
00004 //                                                      //
00005 //                    Copyright (c) 2015 by UNN.                //
00006 //                      All Rights Reserved.                  //
00007 //                                                      //
00008 //  File:      process.h                                  //
00009 //                                                      //
00010 //  Purpose:   Header file for optimization process class           //
00011 //                                                      //
00012 //  Author(s): Sysoyev A.                                   //
```

```
00013 //                                                                      //
00015
00016 #ifndef __PROCESS_H__
00017 #define __PROCESS_H__
00018
00019 #include "SearchData.h"
00020 #include "MethodInterface.h"
00021 #include "Task.h"
00022 #include "Exception.h"
00023 #include "Parameters.h"
00024 #include "Performance.h"
00025 #include "ProblemInterface.h"
00026 #include "Evolvent.h"
00027 #include "CalculationFactory.h"
00028
00029 //extern const int MaxNumOfTaskLevels;
00030
00031 // ----------------------------------------------------------------------------------------
00032 class Process
00033 {
00034 protected:
00036   bool isPrintOptimEstimation;
00038   bool isFirstRun;
00039
00041   Performance Timer;
00043   double duration;
00044
00046   bool IsOptimumFound;
00048   Task* pTask;
00050   SearchData* pData;
00051
00053   IMethod* pMethod;
00061   Evolvent* evolvent;
00063   Calculation* calculation;
00064
00065   std::vector<int> Neighbours;
00066
00068   std::vector<int> functionCalculationCount;
00069
00071   void PrintOptimEstimationToFile(Trial OptimEstimation);
00073   virtual void PrintOptimEstimationToConsole(Trial OptimEstimation);
00075   virtual void OldPrintOptimEstimationToConsole(Trial OptimEstimation);
00077   virtual void PrintResultToFile(Trial OptimEstimation);
00078
00080   virtual void BeginIterations();
00082   virtual void DoIteration();
00084   virtual void EndIterations();
00086   int GetProcLevel() { return pTask->GetProcLevel(); }
00088   bool CheckIsStop(bool IsStop);
00089
00091   std::vector<Trial*>* addPoints;
00092 public:
00093   Process(SearchData& data, Task& task);
00094   virtual ~Process();
00096   double GetSolveTime();
00098   void Solve();
00099
00101   void Reset(SearchData* data, Task* task);
00102
00107   virtual int GetIterationCount() { return pMethod->GetIterationCount(); }
00108   int GetNumberOfTrials() { return pMethod->GetNumberOfTrials(); }
00109
00114   virtual Trial* GetOptimEstimation() { return pMethod->GetOptimEstimation(); }
00115
00116
00122   void InsertPoints(std::vector<Trial*>& points);
00123 };
00124
00125 void ShowIterResults(Process *pProcess);
00126
00127 #endif
00128 // - end of file ------------------------------------------------------------------------
```

5.5 HDSolver.h

```
00001 #pragma once
00002
00003 #include "Solver.h"
00004 #include "HDTask.h"
00005
00006
00010 class HDSolver : public ISolver
00011 {
```

```
00012 protected:
00013
00015   std::vector< Solver*> solvers;
00017   Solver* finalSolver;
00019   std::vector<int> dimensions;
00021   SolutionResult* solutionResult;
00023   int originalDimension;
00025   std::vector <HDTask*> tasks;
00026
00028   IProblem* problem;
00029
00031   std::vector<double> alternativeStartingPoint;
00032
00034   void SetDimentions(std::vector<int> _dimentions);
00035
00037   void CreateStartPoint();
00038
00040   void Construct();
00041
00043   void AddPoint(Solver* solver, int i, std::vector<Trial*>& points, int startParameterNumber);
00044
00046   void UpdateStartPoint(SolutionResult* solution, double& bestValue, int curDimensions,
00047     int startParameterNumber, std::vector<Trial*>& points, HDTask* curTask);
00048
00049 public:
00050   HDSolver(IProblem* problem, std::vector<int> _dimentions = {});
00051
00052 #ifdef _GLOBALIZER_BENCHMARKS
00053
00054   HDSolver(IGlobalOptimizationProblem* problem, std::vector<int> _dimentions = {});
00055
00056 #endif
00057
00058   virtual ~HDSolver();
00059
00061   virtual int Solve();
00062
00063
00064
00065   SolutionResult* GetSolutionResult();
00066
00068   virtual void SetPoint(std::vector<Trial*>& points);
00070   virtual std::vector<Trial*>& GetAllPoint();
00071 };
```

5.6 SolutionResult.h

```
00001 #ifndef __SOLUTION_RESULT_H__
00002 #define __SOLUTION_RESULT_H__
00003
00004 #include "Common.h"
00005 #include "SearchData.h"
00009 struct SolutionResult
00010 {
00012   Trial* BestTrial;
00014   int IterationCount;
00016   int TrialCount;
00017 };
00018
00019 #endif
```

5.7 Solver.h

```
00001 #ifndef __SOLVER_H__
00002 #define __SOLVER_H__
00003
00004 #ifndef _CRT_SECURE_NO_WARNINGS
00005 #define _CRT_SECURE_NO_WARNINGS
00006 #endif
00007
00008 #include <mpi.h>
00009 #include <exception>
00010
00011 #include "Common.h"
00012 #include "GlobProcess.h"
00013 #include "Exception.h"
00014 #include "InitProblem.h"
00015 #include "OutputSystem.h"
00016 #include "Messages.h"
```

```
00017 #include "SolutionResult.h"
00018 #include "SolverInterface.h"
00019 #include "GlobalizerProblem.h"
00020
00021 #ifdef _GLOBALIZER_BENCHMARKS
00022 #include "IGlobalOptimizationProblem.h"
00023 #include "GlobalOptimizationProblemManager.h"
00024 #endif // _GLOBALIZER_BENCHMARKS
00025
00026
00030
00031 class Solver : public ISolver
00032 {
00033 protected:
00035   Process* mProcess;
00037   IProblem* mProblem;
00038
00040   Task* pTask;
00042   bool isExternalTask;
00044   SearchData* pData;
00045
00047   SolutionResult* result;
00048
00050   virtual void ClearData();
00056   virtual void InitAutoPrecision();
00058   virtual int CreateProcess();
00060   int CheckParameters();
00061
00063   void MpiCalculation();
00065   void AsyncCalculation();
00067   std::vector<Trial*>* addPoints;
00068
00069 public:
00070   Solver(IProblem* problem);
00071
00072 #ifdef _GLOBALIZER_BENCHMARKS
00073
00074   Solver(IGlobalOptimizationProblem* problem);
00075
00076 #endif
00077
00079   virtual int Solve();
00081   virtual int Solve(Task* task);
00082
00083   virtual ~Solver();
00084
00086   void SetProblem(IProblem* problem);
00087
00089   IProblem* GetProblem();
00090
00092   SolutionResult* GetSolutionResult();
00093
00095   virtual void SetPoint(std::vector<Trial*>& points);
00096
00098   virtual std::vector<Trial*>& GetAllPoint();
00099
00101   Task* GetTask();
00102
00104   SearchData* GetData();
00105 };
00106
00107 #endif //solver.h
```

5.8 Файл globalizer/include/SolverInterface.h

#include <vector>
#include "Trial.h"

Классы

- class ISolver

  Интерфейс, базового класса

5.8.1 Подробное описание

Авторы

Дата

Авторство

ННГУ им. Н.И. Лобачевского

5.9 SolverInterface.h

См. документацию.

```
00001
00002 //                                                      //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD          //
00004 //                                                      //
00005 //                 Copyright (c) 2015 by UNN.                 //
00006 //                   All Rights Reserved.                   //
00007 //                                                      //
00008 //  File:                                          //
00009 //                                                      //
00010 //  Purpose:   Header file for method class                 //
00011 //                                                      //
00012 //  Author(s):                          //
00013 //                                                      //
00015
00027
00028 #include <vector>
00029 #include "Trial.h"
00030
00031
00032 #ifndef __SOLVER_INTERFACE_H__
00033 #define __SOLVER_INTERFACE_H__
00034
00035 // -----------------------------------------------------------------------------
00039 class ISolver
00040 {
00041 public:
00043   virtual int Solve() = 0;
00045   virtual void SetPoint(std::vector<Trial*>& points) = 0;
00047   virtual std::vector<Trial*>& GetAllPoint() = 0;
00048 };
00049
00050 #endif
00051 // - end of file ------------------------------------------------------------------
```

5.10 BaseInterval.h

```
00001
00002 //                                                      //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD          //
00004 //                                                      //
00005 //                 Copyright (c) 2015 by UNN.                 //
00006 //                   All Rights Reserved.                   //
00007 //                                                      //
00008 //  File:      data.h                                  //
00009 //                                                      //
00010 //  Purpose:   Header file for search data classes          //
00011 //                                                      //
00012 //  Author(s): Sysoyev A., Barkalov K., Sovrasov V.             //
00013 //                                                      //
00015
00016 #ifndef __BASE_INTERVAL_H__
```

```
00017 #define __BASE_INTERVAL_H__
00018
00019 #include "QueueCommon.h"
00020
00021 class QueueBaseData
00022 {
00023 protected:
00025   QueueElement* queueElementa;
00026 public:
00027   virtual void SetQueueElementa(QueueElement* q)
00028   {
00029     queueElementa = q;
00030   }
00031   virtual QueueElement* GetQueueElementa()
00032   {
00033     return queueElementa;
00034   }
00035 };
00036
00037 #endif //__BASE_INTERVAL_H__
```

5.11 DualQueue.h

```
00001
00002 //                                                      //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD          //
00004 //                                                      //
00005 //                    Copyright (c) 2015 by UNN.                    //
00006 //                      All Rights Reserved.                      //
00007 //                                                      //
00008 //  File:      dual_queue.h                                        //
00009 //                                                      //
00010 //  Purpose:   Header file for priority dual queue class               //
00011 //                                                      //
00012 //  Author(s): Sysoyev A., Barkalov K., Sovrasov V.                    //
00013 //                                                      //
00015
00016 #ifndef __DUAL_QUEUE_H__
00017 #define __DUAL_QUEUE_H__
00018
00019 #include "MinMaxHeap.h"
00020 #include "Common.h"
00021 #include "QueueCommon.h"
00022
00023 class PriorityDualQueue : public PriorityQueueCommon
00024 {
00025 protected:
00026   int MaxSize;
00027   int CurLocalSize;
00028   int CurGlobalSize;
00029
00030   MinMaxHeap< QueueElement, _less >* pGlobalHeap;
00031   MinMaxHeap< QueueElement, _less >* pLocalHeap;
00032
00033   void DeleteMinLocalElem();
00034   void DeleteMinGlobalElem();
00035   void ClearLocal();
00036   void ClearGlobal();
00037 public:
00038
00039   PriorityDualQueue(int _MaxSize = DefaultQueueSize); // _MaxSize must be qual to 2^k - 1
00040   ~PriorityDualQueue();
00041
00042   int GetLocalSize() const;
00043   int GetSize() const;
00044   int GetMaxSize() const;
00045   bool IsLocalEmpty() const;
00046   bool IsLocalFull() const;
00047   bool IsEmpty() const;
00048   bool IsFull() const;
00049
00050   QueueElement* Push(double globalKey, double localKey, void *value);
00051   QueueElement* PushWithPriority(double globalKey, double localKey, void *value);
00052   void Pop(double *key, void **value);
00053   void DeleteByValue(void *value);
00055   virtual void DeleteElement(QueueElement * item);
00056   void PopFromLocal(double *key, void **value);
00057
00058   void Clear();
00059   void Resize(int size);
00060   virtual QueueElement& FindMax()
00061   {
00062     return pGlobalHeap->findMax();
```

```
00063   }
00064   void TrickleUp(QueueElement * item)
00065   {
00066     pGlobalHeap->TrickleUp(item);
00067   }
00068
00069   void TrickleDown(QueueElement * item)
00070   {
00071     pGlobalHeap->TrickleDown(item);
00072   }
00073 };
00074 #endif
00075 // - end of file ----------------------------------------------------------------------------
```

5.12 HDTask.h

```
00001 #pragma once
00002
00003 #include "Task.h"
00004
00005 class HDTask : public Task
00006 {
00007 protected:
00008
00010   int startParameterNumber;
00011
00012
00013 public:
00014   HDTask(IProblem* _problem, int _ProcLevel);
00015   HDTask();
00016
00018   virtual Task* Clone();
00019
00021   virtual const double* GetA() const;
00023   virtual const double* GetB() const;
00024
00029   virtual const double* GetOptimumPoint() const;
00031   virtual double CalculateFuncs(const double* y, int fNumber);
00036   virtual void CalculateFuncsInManyPoints(double* y, int fNumber, int numPoints, double* values);
00037
00039   void SetStartParameterNumber(int _startParameterNumber);
00040
00047   virtual void CopyPoint(double* y, Trial* point);
00048
00049 };
```

5.13 Файл globalizer/method/include/Method.h

Объявление класса Method.

#include "MethodInterface.h"
#include "Calculation.h"
#include "InformationForCalculation.h"
#include "SearchIteration.h"
#include "SearchInterval.h"

Классы

- class Method

  Базовый класс, реализующий алгоритм глобального поиска.

5.13.1 Подробное описание

Объявление класса Method.

class$_m$ethodMethod.

Авторы

Баркалов К., Сысоев А.

Дата

2015-2016

Авторство

ННГУ им. Н.И. Лобачевского

Объявление класса Method и сопутствующих типов данных

Авторы

, .

Дата

2015-2016

Авторство

class$_m$ethodMethod

5.14 Method.h

См. документацию.

```
00001
00002 //                                                      //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD          //
00004 //                                                      //
00005 //                 Copyright (c) 2015 by UNN.                 //
00006 //                   All Rights Reserved.                   //
00007 //                                                      //
00008 //  File:      method.h                                    //
00009 //                                                      //
00010 //  Purpose:   Header file for method class                 //
00011 //                                                      //
00012 //  Author(s): Barkalov K., Sysoyev A.                      //
00013 //                                                      //
00015
00016
00028
00029
00030 #ifndef __METHOD_H__
00031 #define __METHOD_H__
00032
00033 #include "MethodInterface.h"
00034 #include "Calculation.h"
00035 #include "InformationForCalculation.h"
```

```
00036 #include "SearchIteration.h"
00037 #include "SearchInterval.h"
00038
00039
00040
00041 // ----------------------------------------------------------------------------------------
00042
00043
00049 class Method : public IMethod
00050 {
00051 protected:
00052   // ------------------------------------------------------------------------
00053   // Копия параметров для конкретного уровня дерева
00054   // ------------------------------------------------------------------------
00056   int MaxNumOfTrials;
00058   int StartLocalIteration;
00059
00060
00062   bool isGlobalMUpdate;
00064   bool isLocalZUpdate;
00065
00066   // ------------------------------------------------------------------------
00067   // Ссылки на объекты используемых методом
00068   // ------------------------------------------------------------------------
00070   Task& pTask;
00072   SearchData* pData;
00073
00075   Calculation& calculation;
00083   Evolvent& evolvent;
00084
00085   // ------------------------------------------------------------------------
00086   // Внутренние данные метода
00087   // ------------------------------------------------------------------------
00088
00090   InformationForCalculation inputSet;
00092   TResultForCalculation outputSet;
00094   SearchIteration iteration;
00096   bool isFoundOptimalPoint;
00097
00099   double           AchievedAccuracy;
00105   double alfa;
00106
00108   std::vector<int> functionCalculationCount;
00109
00111   bool isFindInterval;
00112
00114   bool isSetInLocalMinimumInterval;
00115
00117   int localPointCount;
00119   int numberLocalMethodtStart;
00121   bool isStop;
00122
00124   std::vector<Trial*> localMinimumPoints;
00125
00126
//=============================================================================================
00127   //Для методов локального уточнения нужны миксимумы
00128
00130   double* Xmax;
00132   double* mu;
00134   bool isSearchXMax;
00135
//=============================================================================================
00136
00138   std::vector<Trial*> printPoints;
00139
00141   virtual void  SavePoints();
00142
00148   virtual double CalculateGlobalR(SearchInterval* p);
00156   virtual double CalculateLocalR(SearchInterval* p);
00162   virtual void CalculateM(SearchInterval* p);
00173   virtual IterationType GetIterationType(int iterationNumber, int localMixParameter);
00177   virtual int IsBoundary(SearchInterval* p);
00178
00188   virtual void UpdateM(double newValue, int index, int boundaryStatus, SearchInterval* p);
00189
00197   virtual bool UpdateOptimumEstimation(Trial& trial);
00198
00200   virtual void CalculateCurrentPoint(Trial& pCurTrialsj, SearchInterval* BestIntervalsj);
00201
00203   virtual void CalculateCurrentPoints(std::vector<SearchInterval*>& BestIntervals);
00204
00206   virtual bool IsIntervalInSegment(SearchInterval* basicInterval, SearchInterval* newInterval);
00207
00208
00212   virtual double Update_r(int iter = -1, int procLevel = -1);
00213
```

```
00215   virtual void CalculateImage(Trial& pCurTrialsj);
00216
00217
00218
00220   virtual SearchInterval* AddCurrentPoint(Trial& pCurTrialsj, SearchInterval* BestIntervalsj);
00221
00223   virtual void Recalc();
00224
00225
00227   virtual SearchData* GetSearchData(Trial* trial);
00228
00229
00233   virtual void SetNumPoints(int newNP);
00234
00235
00236 public:
00237
00238   Method(Task& _pTask, SearchData& _pData,
00239     Calculation& _Calculation, Evolvent& _Evolvent);
00240   virtual ~Method();
00241
00244   virtual void FirstIteration();
00245
00250   virtual void CalculateIterationPoints();
00251
00257   virtual void CalculateFunctionals();
00258
00259
00262   virtual void RenewSearchData();
00263
00273   virtual bool CheckStopCondition();
00274
00279   virtual bool EstimateOptimum();
00280
00283   virtual void FinalizeIteration();
00284
00289   virtual int GetIterationCount();
00290
00291
00296   virtual Trial* GetOptimEstimation();
00297
00304   virtual int GetNumberOfTrials();
00305
00307   virtual void PrintLevelPoints(const std::string& fileName);
00308
00310   virtual void PrintPoints(const std::string & fileName);
00311
00313   void HookeJeevesMethod(Trial& point, std::vector<Trial*>& localPoints);
00314
00316   virtual std::vector<int> GetFunctionCalculationCount();
00317
00319   virtual double GetAchievedAccuracy();
00320
00323   virtual void ResetSearchData() {};
00324
00325
00331   void InsertPoints(const std::vector<Trial*>& points);
00337   virtual void InsertLocalPoints(const std::vector<Trial*>& points, Task* task = 0);
00338
00340   virtual void LocalSearch();
00341
00343   virtual int GetLocalPointCount();
00344
00346   virtual int GetNumberLocalMethodtStart();
00347
00349   virtual void PrintSection();
00350
00351 };
00352
00353 #endif
00354 // - end of file ----------------------------------------------------------------------------
```

5.15 MethodFactory.h

```
00001
00002 //                                                      //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD          //
00004 //                                                      //
00005 //                  Copyright (c) 2015 by UNN.                  //
00006 //                    All Rights Reserved.                    //
00007 //                                                      //
00008 //  File:      method_factory.h                                  //
00009 //                                                      //
00010 //  Purpose:   Header file for method factory class                  //
```

```
00011 //                                                                      //
00012 //  Author(s): Lebedev I.                                               //
00013 //                                                                      //
00015
00016 #ifndef __METHOD_FACTORY_H__
00017 #define __METHOD_FACTORY_H__
00018
00019 #include "Method.h"
00020
00021 class MethodFactory
00022 {
00023 public:
00024   static IMethod* CreateMethod(Task& _pTask, SearchData& _pData,
00025     Calculation& _Calculation, Evolvent& _Evolvent);
00026 };
00027
00028 #endif
00029 // - end of file ----------------------------------------------------------------------------
```

5.16 MethodInterface.h

```
00001
00002 //                                                                      //
00003 //           LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD            //
00004 //                                                                      //
00005 //                  Copyright (c) 2015 by UNN.                          //
00006 //                    All Rights Reserved.                              //
00007 //                                                                      //
00008 //  File:      method.h                                                 //
00009 //                                                                      //
00010 //  Purpose:   Header file for method class                             //
00011 //                                                                      //
00012 //  Author(s): Barkalov K., Sysoyev A.                                  //
00013 //                                                                      //
00015
00016
00028
00029
00030 #ifndef __METHOD_INTERFACE_H__
00031 #define __METHOD_INTERFACE_H__
00032
00033 #include "Common.h"
00034 #include "Task.h"
00035 #include "Trial.h"
00036 #include "Evolvent.h"
00037 #include "Parameters.h"
00038
00039 // ---------------------------------------------------------------------------------------
00040
00046 class IMethod
00047 {
00048 public:
00051   virtual void FirstIteration() = 0;
00052
00057   virtual void CalculateIterationPoints() = 0;
00058
00064   virtual void CalculateFunctionals() = 0;
00065
00068   virtual void RenewSearchData() = 0;
00069
00079   virtual bool CheckStopCondition() = 0;
00080
00085   virtual bool EstimateOptimum() = 0;
00086
00089   virtual void FinalizeIteration() = 0;
00090
00095   virtual int GetIterationCount() = 0;
00096
00101   virtual Trial* GetOptimEstimation() = 0;
00102
00109   virtual int GetNumberOfTrials() = 0;
00110
00112   virtual std::vector<int> GetFunctionCalculationCount() = 0;
00113
00115   virtual double GetAchievedAccuracy() = 0;
00116
00117
00123   virtual void InsertPoints(const std::vector<Trial*>& points) = 0;
00124
00126   virtual void PrintPoints(const std::string & fileName) = 0;
00127
00129   virtual void LocalSearch() = 0;
00130
00132   virtual void  SavePoints() = 0;
```

```
00133
00135   virtual int GetLocalPointCount() = 0;
00137   virtual int GetNumberLocalMethodtStart() = 0;
00139   virtual void PrintSection() = 0;
00140 };
00141
00142 #endif
00143 // - end of file ------------------------------------------------------------------------
```

5.17 MinMaxHeap.h

```
00001
00002 //                                                                      //
00003 //            LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD           //
00004 //                                                                      //
00005 //                    Copyright (c) 2015 by UNN.                        //
00006 //                      All Rights Reserved.                          //
00007 //                                                                      //
00008 //  File:      minmaxheap.h                                             //
00009 //                                                                      //
00010 //  Purpose:   Header file for minmaxheap class                         //
00011 //                                                                      //
00012 //  Author(s): ???                                                    //
00013 //                                                                      //
00015
00016 #ifndef __MINMAXHEAP_H__
00017 #define __MINMAXHEAP_H__
00018
00019 #include "Exception.h"
00020
00021 #include <algorithm>
00022 #include <functional>
00023
00024 #include "BaseInterval.h"
00025
00026   // возвращает двоичный логарифм от val
00027 inline unsigned int log2(unsigned int val)
00028 {
00029   //if (val == 0)
00030   //    throw exception("log2(0) не определен\n");
00031
00032   unsigned int result = 0;
00033
00034   while (val)
00035   {
00036     val >>= 1;
00037     ++result;
00038   }
00039
00040   return result - 1;
00041 }
00042
00043 //  T - тип элементов в куче
00044 //  Container - тип контейнера, используемый для хранения кучи
00045 //  Compare - функция сравнения, чтобы определить как сравнивать элементы в куче
00046
00047 // Считаем, что корень - Max Level
00048 template<class T, class Compare = std::less<T> >
00049 class MinMaxHeap
00050 {
00051   T* m_heap; // здесь хранится содержимое кучи
00052   Compare m_compare; // объект для сравнения
00053   unsigned m_heapsize; // размер кучи
00054   unsigned m_currentheapsize; // текущее количество элементов в куче
00055
00056
00057   static inline unsigned int parent(unsigned int index)
00058   {
00059     return (index - 1) / 2; // возвращает индекс родителя узла, заданного index-ом
00060   }
00061
00062   inline void swapLinkedElems(T& arg1, T& arg2)
00063   {
00064     if (arg1.pLinkedElement != NULL && arg2.pLinkedElement != NULL)
00065       std::swap(arg1.pLinkedElement->pLinkedElement, arg2.pLinkedElement->pLinkedElement);
00066     else if (arg1.pLinkedElement != NULL)
00067       arg1.pLinkedElement->pLinkedElement = &arg2;
00068     else if (arg2.pLinkedElement != NULL)
00069       arg2.pLinkedElement->pLinkedElement = &arg1;
00070   }
00071
00072   // ------------------------------------------------------------------------------------
00073   static inline unsigned int leftChild(unsigned int index)
00074   {
```

```
00075     return 2 * index + 1;  // возвращает индекс левого сына узла, заданного index-ом
00076   }
00077
00078   // ----------------------------------------------------------------------------------------
00079   static inline unsigned int rightChild(unsigned int index)
00080   {
00081     return 2 * index + 2;  // возвращает индекс правого сына узла, заданного index-ом
00082   }
00083
00084   // ----------------------------------------------------------------------------------------
00085   static inline bool isOnMinLevel(unsigned int index)
00086   {
00087     return log2(index + 1) % 2 == 1; // true - если вершина, соответствующая index находится на Min Level
00088   }
00089
00090   // ----------------------------------------------------------------------------------------
00091   static inline bool isOnMaxLevel(unsigned int index)
00092   {
00093     return !isOnMinLevel(index); // true - если вершина, соответствующая index находится на Max Level
00094   }
00095
00096   // ----------------------------------------------------------------------------------------
00097   template<bool MaxLevel>
00098   int trickleUp_(unsigned int index) // вспомогательный метод для всплытия
00099   {
00100     if (index == 0) // не можем всплывать дальше
00101       return index;
00102
00103     unsigned int index_grandparent = parent(index); // найдем первый родительский пройденный уровень
00104
00105     if (index_grandparent == 0) // если такого нет, выходим
00106       return index;
00107
00108     index_grandparent = parent(index_grandparent); // найти прародителя
00109
00110     if (m_compare(m_heap[index], m_heap[index_grandparent]) ^ MaxLevel) // убедимся, что нужно поменяться
местами с прародителем
00111     {
00112       QueueBaseData* i1 = static_cast<QueueBaseData*>(m_heap[index].pValue);
00113       QueueBaseData* i2 = static_cast<QueueBaseData*>(m_heap[index_grandparent].pValue);
00114
00115       swapLinkedElems(m_heap[index_grandparent], m_heap[index]);
00116       std::swap(m_heap[index_grandparent], m_heap[index]);
00117
00118       i1->SetQueueElementa(&m_heap[index_grandparent]);
00119       i2->SetQueueElementa(&m_heap[index]);
00120
00121       return trickleUp_<MaxLevel>(index_grandparent);
00122     }
00123     return index;
00124   }
00125
00126   // ----------------------------------------------------------------------------------------
00127   int trickleUp(unsigned int index) // размещаем узел на соответствующем уровне (Min или Max)
00128   {
00129     if (index == 0)  // не можем всплывать дальше
00130       return index;
00131
00132     unsigned int index_parent = parent(index); // найдем первый родительский пройденный уровень
00133
00134     if (isOnMinLevel(index))
00135     {
00136       // убедимся, что нужно поменяться местами с родителем
00137       if (m_compare(m_heap[index_parent], m_heap[index]))
00138       {
00139         QueueBaseData* i1 = static_cast<QueueBaseData*>(m_heap[index_parent].pValue);
00140         QueueBaseData* i2 = static_cast<QueueBaseData*>(m_heap[index].pValue);
00141
00142         swapLinkedElems(m_heap[index_parent], m_heap[index]);
00143         std::swap(m_heap[index_parent], m_heap[index]);
00144
00145         i1->SetQueueElementa(&m_heap[index]);
00146         i2->SetQueueElementa(&m_heap[index_parent]);
00147
00148         return trickleUp_<true>(index_parent);
00149       }
00150       else
00151         return trickleUp_<false>(index);
00152     }
00153     else
00154     {
00155       // убедимся, что нужно поменяться местами с родителем
00156       if (m_compare(m_heap[index], m_heap[index_parent]))
00157       {
00158         QueueBaseData* i1 = static_cast<QueueBaseData*>(m_heap[index_parent].pValue);
00159         QueueBaseData* i2 = static_cast<QueueBaseData*>(m_heap[index].pValue);
00160
```

```
00161       swapLinkedElems(m_heap[index_parent], m_heap[index]);
00162       std::swap(m_heap[index_parent], m_heap[index]);
00163
00164       i1->SetQueueElementa(&m_heap[index]);
00165       i2->SetQueueElementa(&m_heap[index_parent]);
00166
00167       return trickleUp_<false>(index_parent);
00168     }
00169     else
00170       return trickleUp_<true>(index);
00171   }
00172 }
00173
00174 // ----------------------------------------------------------------------------------------------
00175 template<bool MaxLevel>
00176 void trickleDown_(unsigned int index) // вспомогательный метод для погружения
00177 {
00178   //  if ( index >= m_currentheapsize ) // убедимся, что элемент существует
00179   //    throw exception("Элемент с таким индексом не существует\n");
00180
00181   unsigned int smallestNode = index; // храним индекс наименьшего узла
00182   unsigned int left = leftChild(index); // получаем правого сына
00183
00184   if (left < m_currentheapsize && (m_compare(m_heap[left], m_heap[smallestNode]) ^ MaxLevel)) //
проверяем левого и правого сыновей
00185     smallestNode = left;
00186   if (left + 1 < m_currentheapsize && (m_compare(m_heap[left + 1], m_heap[smallestNode]) ^ MaxLevel))
00187     smallestNode = left + 1;
00188
00189   unsigned int leftGrandchild = leftChild(left); // проверяем внуков
00190   for (unsigned int i = 0; i < 4 && leftGrandchild + i < (unsigned int)m_currentheapsize; ++i)
00191     if (m_compare(m_heap[leftGrandchild + i], m_heap[smallestNode]) ^ MaxLevel)
00192       smallestNode = leftGrandchild + i;
00193
00194   if (index == smallestNode) // если текущий узел наименьший, ничего не делаем и выходим
00195     return;
00196
00197   QueueBaseData* i1 = static_cast<QueueBaseData*>(m_heap[index].pValue);
00198   QueueBaseData* i2 = static_cast<QueueBaseData*>(m_heap[smallestNode].pValue);
00199
00200   swapLinkedElems(m_heap[index], m_heap[smallestNode]);
00201   std::swap(m_heap[index], m_heap[smallestNode]); // меняем местами текущий узел и наименьший
00202
00203   i1->SetQueueElementa(&m_heap[smallestNode]);
00204   i2->SetQueueElementa(&m_heap[index]);
00205
00206   if (smallestNode - left > 1)
00207   {
00208     // если родитель наименьшего узла больше, чем сам узел, меняем местами
00209     if (m_compare(m_heap[parent(smallestNode)], m_heap[smallestNode]) ^ MaxLevel)
00210     {
00211
00212       int par0 = parent(smallestNode);
00213
00214       i1 = static_cast<QueueBaseData*>(m_heap[par0].pValue);
00215       i2 = static_cast<QueueBaseData*>(m_heap[smallestNode].pValue);
00216
00217       swapLinkedElems(m_heap[parent(smallestNode)], m_heap[smallestNode]);
00218       par0 = parent(smallestNode);
00219       std::swap(m_heap[parent(smallestNode)], m_heap[smallestNode]);
00220
00221       i1->SetQueueElementa(&m_heap[smallestNode]);
00222       i2->SetQueueElementa(&m_heap[par0]);
00223     }
00224
00225     trickleDown_<MaxLevel>(smallestNode);
00226   }
00227 }
00228
00229 // ----------------------------------------------------------------------------------------------
00230 void trickleDown(unsigned int index) // погружение
00231 {
00232   if (isOnMinLevel(index))
00233     trickleDown_<false>(index);
00234   else
00235     trickleDown_<true>(index);
00236 }
00237
00238 // ----------------------------------------------------------------------------------------------
00239 unsigned int findMinIndex() const // поиск индекса наименьшего узла
00240 {
00241   switch (m_currentheapsize)
00242   {
00243   case 0:
00244     // куча пуста
00245     throw EXCEPTION("Куча пуста\n");
00246     break;
```

```
00247     case 1:
00248       // в куче только один элемент
00249       return 0;
00250     case 2:
00251       // в куче 2 элемента => сын должен быть минимумом
00252       return 1;
00253     default:
00254       // в куче больше 2х элементов
00255       return m_compare(m_heap[1], m_heap[2]) ? 1 : 2;
00256     }
00257   }
00258
00259   // ----------------------------------------------------------------------------------------
00260   void deleteElement(unsigned int index) // удаление элемента из кучи
00261   {
00262     //     if (index >= (unsigned int)m_currentheapsize) // проверить существование элемента
00263      //        throw exception("Элемент с таким индексом не существует\n");
00264
00265         // если мы удаляем последний элемент из кучи
00266     if (index == m_currentheapsize - 1)
00267     {
00268       m_currentheapsize--;
00269       return;
00270     }
00271
00272     //TBaseInterval* i1 = (TBaseInterval*)m_heap[index].pValue;
00273     //TBaseInterval* i2 = (TBaseInterval*)m_heap[m_currentheapsize - 1].pValue;
00274
00275     QueueBaseData* i1 = static_cast<QueueBaseData*>(m_heap[index].pValue);
00276     QueueBaseData* i2 = static_cast<QueueBaseData*>(m_heap[m_currentheapsize - 1].pValue);
00277
00278     swapLinkedElems(m_heap[index], m_heap[m_currentheapsize - 1]);
00279     std::swap(m_heap[index], m_heap[m_currentheapsize - 1]); // меняем местами элемент с последним в куче
00280
00281     i1->SetQueueElementa(&m_heap[m_currentheapsize - 1]);
00282     i2->SetQueueElementa(&m_heap[index]);
00283
00284     m_currentheapsize--; // удаляем последний элемент из кучи
00285
00286     trickleDown(index); // погружаем тот элемент, который поместили на место index
00287   }
00288
00289 public:
00290   MinMaxHeap(unsigned heapsize) : m_heap(NULL)
00291   {
00292     m_heapsize = heapsize;
00293     m_heap = new T[m_heapsize];
00294     m_currentheapsize = 0;
00295   }
00296
00297   ~MinMaxHeap()
00298   {
00299     if (m_heap != NULL)
00300       delete[] m_heap;
00301   }
00302
00303   void clear()
00304   {
00305     m_currentheapsize = 0;
00306   }
00307
00308   bool empty() const
00309   {
00310     return m_currentheapsize == 0; // проверка на пустоту кучи
00311   }
00312
00313   unsigned int size() const
00314   {
00315     return (unsigned int)m_currentheapsize; // количество элементов в куче
00316   }
00317
00318
00319   T* push(const T & val) // добавление элемента в кучу
00320   {
00321     if (m_currentheapsize < m_heapsize)
00322     {
00323       m_heap[m_currentheapsize] = val; // добавляем в конец
00324       m_currentheapsize++;
00325       return m_heap + trickleUp(m_currentheapsize - 1); // всплываем
00326     }
00327     return NULL;
00328     //  else
00329     //      throw exception("Вставка в кучу невозможна! Переполнение!\n");
00330   }
00331
00332   void TrickleUp(const T* ptr)
00333   {
```

```
00334   unsigned index = unsigned(ptr - m_heap);
00335   trickleUp(index);
00336  }
00337
00338  // ---------------------------------------------------------------------------------------
00339  void TrickleDown(const T* ptr)
00340  {
00341   unsigned index = unsigned(ptr - m_heap);
00342   trickleDown(index);
00343  }
00344
00345  T & findMax() const // возвращает максимум за O(1) без удаления из кучи
00346  {
00347   //      if (empty()) // проверка на пустоту
00348    //         throw exception("Куча пуста \n");
00349
00350   return m_heap[0];
00351  }
00352
00353  const T & findMin() const
00354  {
00355   return m_heap[findMinIndex()];  // возвращает минимум за O(1) без удаления из кучи
00356  }
00357
00358  T popMax() // извлечение (с удалением из кучи) максимального элемента
00359  {
00360   //   if (empty()) // проверяем кучу на пустоту
00361    //      throw exception("Куча пуста \n");
00362
00363   T temp = m_heap[0]; // сохраняем максимум
00364   int delIndx = 0;
00365   deleteElement(delIndx); // удаляем
00366
00367   return temp;
00368  }
00369
00370  T pop()
00371  {
00372   return popMax(); // извлечение (с удалением из кучи) максимального элемента
00373  }
00374
00375
00376  T popMin() // извлечение (с удалением из кучи) минимального элемента
00377  {
00378   //   if (empty()) // проверяем не пуста ли куча
00379    //      throw exception("Куча пуста \n");
00380
00381   unsigned int smallest = findMinIndex(); // сохраняем индекс минимума
00382   T temp = m_heap[smallest]; // сохраняем минимальное значение
00383   deleteElement(smallest); // удаляем
00384
00385   return temp;
00386  }
00387
00388  void deleteElement(const T* ptr)
00389  {
00390   unsigned index = unsigned(ptr - m_heap);
00391   //if(index >= 0 && index < m_heapsize)
00392   deleteElement(index);
00393   //  else
00394   //      throw std::exception("Куча пуста \n");
00395  }
00396
00397  T* getHeapMemPtr() const
00398  {
00399   return m_heap;
00400  }
00401 };
00402
00403 #endif
00404 // - end of file ------------------------------------------------------------------------
```

5.18 Файл globalizer/method/include/MixedIntegerMethod.h

Объявление класса MixedIntegerMethod.

#include "Method.h"

Классы

- class MixedIntegerMethod
  Базовый класс, реализующий алгоритм глобального поиска.

5.18.1 Подробное описание

Объявление класса MixedIntegerMethod.

Авторы

Лебедев И.

Дата

2025-2026

Авторство

ННГУ им. Н.И. Лобачевского

Объявление класса MixedIntegerMethod и сопутствующих типов данных

5.19 MixedIntegerMethod.h

См. документацию.

```
00001
00002 //                                                      //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD          //
00004 //                                                      //
00005 //                 Copyright (c) 2025 by UNN.                 //
00006 //                   All Rights Reserved.                   //
00007 //                                                      //
00008 //  File:      MixedIntegerMethod.h                        //
00009 //                                                      //
00010 //  Purpose:   Header file for method class                 //
00011 //                                                      //
00012 //  Author(s): Lebedev.i                                    //
00013 //                                                      //
00015
00016
00028
00029
00030 #ifndef __MIXED_INTEGER_METHOD_H__
00031 #define __MIXED_INTEGER_METHOD_H__
00032
00033 #include "Method.h"
00034
00035
00041 class MixedIntegerMethod : public Method
00042 {
00043 protected:
00044
00045
00051   int mDiscreteValuesCount;
00053   std::vector< std::vector< double > > mDiscreteValues;
00055   int startDiscreteVariable;
00056
00057
00059   std::vector<Trial*> localMinimumPoints;
00060
00062   virtual void CalculateCurrentPoint(Trial& pCurTrialsj, SearchInterval* BestIntervalsj);
00063
00064
00066   virtual void SetDiscreteValue(int u, std::vector< std::vector <double> > dvs);
00067
00069   virtual SearchData* GetSearchData(Trial* trial);
00070
00071
00072 public:
00073
00074   MixedIntegerMethod(Task& _pTask, SearchData& _pData,
00075     Calculation& _Calculation, Evolvent& _Evolvent);
00076   virtual ~MixedIntegerMethod();
```

```
00077
00080   virtual void FirstIteration();
00081
00086   virtual void CalculateIterationPoints();
00087
00088
00089
00090 };
00091
00092 #endif
```

5.20 ParallelHookeJeevesMethod.h

```
00001
00002 //                                                      //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD          //
00004 //                                                      //
00005 //                    Copyright (c) 2015 by UNN.                    //
00006 //                      All Rights Reserved.                      //
00007 //                                                      //
00008 //  File:      local_method.h                                       //
00009 //                                                      //
00010 //  Purpose:   Header file for local method class                   //
00011 //                                                      //
00012 //  Author(s): Sovrasov V.                                          //
00013 //                                                      //
00015
00016 #ifndef __PARALLEL_HOOKE_JEEVES_METHOD_H__
00017 #define __PARALLEL_HOOKE_JEEVES_METHOD_H__
00018
00019 #include "Parameters.h"
00020 #include "Task.h"
00021 #include "Trial.h"
00022 #include "Common.h"
00023 #include <vector>
00024
00025 #include "LocalMethod.h"
00026 #include "Calculation.h"
00027
00028 class ParallelHookeJeevesMethod : public LocalMethod
00029 {
00030 protected:
00031
00032   Calculation& calculation;
00034   InformationForCalculation inputSet;
00036   TResultForCalculation outputSet;
00037
00038   virtual double MakeResearch(OBJECTIV_TYPE*);
00039
00040   //virtual double EvaluateObjectiveFunctiuon(const OBJECTIV_TYPE*);
00041
00042   double CheckCoordinate(const OBJECTIV_TYPE* x);
00043
00044   virtual double EvaluateObjectiveFunctiuon(const OBJECTIV_TYPE*);
00045
00046 public:
00047
00048   ParallelHookeJeevesMethod(Task* _pTask, Trial _startPoint, Calculation& _calculation, int logPoints = 0);
00049
00050 //  virtual Trial StartOptimization();
00051 };
00052
00053 #endif //__PARALLEL_HOOKE_JEEVES_METHOD_H__
00054 // - end of file ----------------------------------------------------------------------------
```

5.21 Performance.h

```
00001
00002 //                                                      //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD          //
00004 //                                                      //
00005 //                    Copyright (c) 2015 by UNN.                    //
00006 //                      All Rights Reserved.                      //
00007 //                                                      //
00008 //  File:      performance.h                                        //
00009 //                                                      //
00010 //  Purpose:   Performance measurement class                        //
00011 //                                                      //
00012 //  Author(s): Sysoyev A.                                           //
00013 //                                                      //
00015
```

```
00016 #ifndef __PERFORMANCE_H__
00017 #define __PERFORMANCE_H__
00018
00019 #ifdef WIN32
00020 #include <windows.h>
00021 #else
00022 #include <sys/time.h>
00023  //TODO: add Linux specific things
00024 #endif
00025
00026 // ----------------------------------------------------------------------------------------
00027 class Performance
00028 {
00029 private:
00030 #ifdef WIN32
00031   LARGE_INTEGER startCount;
00032   LARGE_INTEGER s_freq;
00033 #else
00034 // TODO: add Linux specific things
00035   struct timeval tv1,tv2,dtv;
00036   struct timezone tz;
00037 #endif
00038
00039 public:
00040   Performance()
00041   {
00042 #ifdef WIN32
00043     s_freq.QuadPart = 0;
00044 #else
00045 // TODO: add Linux specific things
00046 #endif
00047   }
00048
00049   void Start()
00050   {
00051 #ifdef WIN32
00052     QueryPerformanceCounter(&startCount);
00053 #else
00054 // TODO: add Linux specific things
00055     gettimeofday(&tv1, &tz);
00056 #endif
00057   }
00058
00059   double GetTime()
00060   {
00061 #ifdef WIN32
00062     LARGE_INTEGER finishCount;
00063     QueryPerformanceCounter(&finishCount);
00064
00065     if(s_freq.QuadPart == 0)
00066       QueryPerformanceFrequency(&s_freq);
00067
00068     return ((double)(finishCount.QuadPart - startCount.QuadPart) /
00069       (double)s_freq.QuadPart);// * 1000.0;
00070 #else
00071 // TODO: add Linux specific things
00072     gettimeofday(&tv2, &tz);
00073   dtv.tv_sec= tv2.tv_sec -tv1.tv_sec;
00074   dtv.tv_usec=tv2.tv_usec-tv1.tv_usec;
00075   if(dtv.tv_usec<0) { dtv.tv_sec--; dtv.tv_usec+=1000000; }
00076   long res = dtv.tv_sec*1000+dtv.tv_usec/1000;
00077   return (double)res / 1000.0;
00078 #endif
00079   }
00080 };
00081
00082 #endif // __PERFORMANCE_H__
00083 // - end of file ----------------------------------------------------------------------------
```

5.22 Queue.h

```
00001
00002 //                                                      //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD          //
00004 //                                                      //
00005 //                    Copyright (c) 2015 by UNN.                    //
00006 //                      All Rights Reserved.                      //
00007 //                                                      //
00008 //  File:      Queue.h                                    //
00009 //                                                      //
00010 //  Purpose:   Header file for priority queue class                 //
00011 //                                                      //
00012 //  Author(s): Sysoyev A., Barkalov K., Sovrasov V.                 //
00013 //                                                      //
```

```
00015
00016 #ifndef __QUEUE_H__
00017 #define __QUEUE_H__
00018
00019 #include "MinMaxHeap.h"
00020 #include "Common.h"
00021 #include "QueueCommon.h"
00022
00023 class PriorityQueue : public PriorityQueueCommon
00024 {
00025 protected:
00026   int MaxSize;
00027   int CurSize;
00028   MinMaxHeap< QueueElement, _less >* pMem;
00029
00030   int GetIndOfMinElem();
00031   void DeleteMinElem();
00032
00033 public:
00034   PriorityQueue(int _MaxSize = DefaultQueueSize); // _MaxSize must be qual to 2^k - 1
00035   ~PriorityQueue();
00036
00037   int GetSize() const;
00038   int GetMaxSize() const;
00039   bool IsEmpty() const;
00040   bool IsFull() const;
00041
00042   //localKey value is not really used by the Push and PushWithPriority methods
00043   QueueElement* Push(double globalKey, double localKey, void *value);
00044   QueueElement* PushWithPriority(double globalKey, double localKey, void *value);
00045   void Pop(double *key, void **value);
00046   void DeleteByValue(void *value);
00047
00049   void DeleteElement(QueueElement * item);
00050
00051   void Clear();
00052   void Resize(int size);
00053
00054   QueueElement& FindMax();
00055   void TrickleUp(QueueElement * item);
00056   void TrickleDown(QueueElement * item)
00057   {
00058     pMem->TrickleDown(item);
00059   }
00060 };
00061 #endif
00062 // - end of file ----------------------------------------------------------------------------
```

5.23 QueueCommon.h

```
00001
00002 //                                                      //
00003 //           LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD          //
00004 //                                                      //
00005 //                  Copyright (c) 2015 by UNN.                  //
00006 //                    All Rights Reserved.                    //
00007 //                                                      //
00008 //  File:      queue_common.h                                  //
00009 //                                                      //
00010 //  Purpose:   Header file for abstract queue class                //
00011 //                                                      //
00012 //  Author(s): Sysoyev A., Barkalov K., Sovrasov V.                 //
00013 //                                                      //
00015
00016 #ifndef __QUEUECOMMON_H__
00017 #define __QUEUECOMMON_H__
00018
00019 struct QueueElement
00020 {
00021   QueueElement *pLinkedElement;
00022   double Key;
00023   void *pValue;
00024
00025   QueueElement() {}
00026   QueueElement(double _Key, void *_pValue) :
00027     Key(_Key), pValue(_pValue), pLinkedElement(0)
00028   {}
00029   QueueElement(double _Key, void *_pValue, QueueElement* _pLinkedElement) :
00030     Key(_Key), pValue(_pValue), pLinkedElement(_pLinkedElement)
00031   {}
00032 };
00033
00034 template<class _Arg1,
00035   class _Arg2,
```

```
00036   class _Result>
00037   struct _binary_function
00038 {   // base class for binary functions
00039   typedef _Arg1 first_argument_type;
00040   typedef _Arg2 second_argument_type;
00041   typedef _Result result_type;
00042 };
00043
00044 struct _less : public _binary_function<QueueElement, QueueElement, bool>
00045 {   // functor for operator<
00046   bool operator()(const QueueElement& _Left, const QueueElement& _Right) const
00047   { // apply operator< to operands
00048     return (_Left.Key < _Right.Key);
00049   }
00050 };
00051
00052 class PriorityQueueCommon
00053 {
00054 public:
00055   virtual ~PriorityQueueCommon() {}
00056
00057   virtual int GetSize() const = 0;
00058   virtual int GetMaxSize() const = 0;
00059   virtual bool IsEmpty() const = 0;
00060   virtual bool IsFull() const = 0;
00061
00062   virtual QueueElement* Push(double globalKey, double localKey, void *value) = 0;
00063   virtual QueueElement* PushWithPriority(double globalKey, double localKey, void *value) = 0;
00064   virtual void Pop(double *key, void **value) = 0;
00065   virtual void DeleteByValue(void *value) = 0;
00067   virtual void DeleteElement(QueueElement * item) = 0;
00068   virtual void Clear() = 0;
00069   virtual void Resize(int size) = 0;
00070   virtual QueueElement& FindMax() = 0;
00071   virtual void TrickleUp(QueueElement * item) = 0;
00072   virtual void TrickleDown(QueueElement * item) = 0;
00073 };
00074
00075 #endif
00076 // - end of file ----------------------------------------------------------------------------
```

5.24 SearcDataIterator.h

```
00001
00002 //                                                                 //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD        //
00004 //                                                                 //
00005 //                    Copyright (c) 2015 by UNN.                   //
00006 //                      All Rights Reserved.                       //
00007 //                                                                 //
00008 //  File:      data.h                                              //
00009 //                                                                 //
00010 //  Purpose:   Header file for search data classes                  //
00011 //                                                                 //
00012 //  Author(s): Sysoyev A., Barkalov K., Sovrasov V.                //
00013 //                                                                 //
00015
00016 #ifndef __SEARC_DATA_ITERATOR_H__
00017 #define __SEARC_DATA_ITERATOR_H__
00018
00019 #include "SearchData.h"
00020 #include "TreeNode.h"
00021
00022
00023
00024 // ----------------------------------------------------------------------------------------
00025 class SearcDataIterator
00026 {
00027   friend class SearchData;
00028 protected:
00029   SearchData *pContainer;
00030   TreeNode *pObject;
00031
00032 public:
00033   SearcDataIterator();
00034
00035   SearcDataIterator& operator++();
00036   SearcDataIterator operator++(int);
00037   SearcDataIterator& operator--();
00038   SearcDataIterator operator--(int);
00039   operator void*() const;
00040   SearchInterval* operator->();
00041   SearchInterval* operator *() const;
00042 };
```

```
00043
00044
00045
00046 #endif
00047 // - end of file ----------------------------------------------------------------------
```

5.25 SearchData.h

```
00001
00002 //                                                                      //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD             //
00004 //                                                                      //
00005 //                    Copyright (c) 2015 by UNN.                        //
00006 //                      All Rights Reserved.                          //
00007 //                                                                      //
00008 //  File:      data.h                                                  //
00009 //                                                                      //
00010 //  Purpose:   Header file for search data classes                      //
00011 //                                                                      //
00012 //  Author(s): Sysoyev A., Barkalov K., Sovrasov V.                     //
00013 //                                                                      //
00015
00016 #ifndef __SEARCH_DATA_H__
00017 #define __SEARCH_DATA_H__
00018
00019 #include "Common.h"
00020 #include "Extended.h"
00021 #include "DualQueue.h"
00022 #include "Queue.h"
00023 #include <stack>
00024 #include "Parameters.h"
00025
00026 #include <string.h>
00027
00028 #include <list>
00029 #include <vector>
00030
00031
00032 class TreeNode;
00033 class SearchInterval;
00034 class SearcDataIterator;
00035 class Trial;
00036 // -------------------------------------------------------------------------------------------
00037 class SearchData
00038 {
00039   friend class SearcDataIterator;
00040 protected:
00042   int NumOfFuncs;
00044   int MaxSize;
00046   int Count;
00048   int CurIndex;
00050   TreeNode *pRoot;
00052   TreeNode *pCur;
00053
00055   std::stack<TreeNode*> Stack;
00057   PriorityQueueCommon *pQueue;
00058
00060   std::vector<Trial*> trials;
00061
00063   bool recalc;
00064
00066   Trial* BestTrial;
00067
00069   void DeleteTree(TreeNode *pNode);
00071   unsigned char GetHeight(TreeNode *p);
00073   int GetBalance(TreeNode *p);
00075   void FixHeight(TreeNode *p);
00077   TreeNode* RotateRight(TreeNode *p);
00079   TreeNode* RotateLeft(TreeNode *p);
00081   TreeNode* Balance(TreeNode *p);
00083   TreeNode* Maximum(TreeNode *p) const;
00085   TreeNode* Minimum(TreeNode *p) const;
00087   TreeNode* Previous(TreeNode *p) const;
00089   TreeNode* Next(TreeNode *p) const;
00091   TreeNode* Insert(TreeNode *p, SearchInterval &pInterval);
00093   TreeNode* Find(TreeNode *p, Trial* x) const;
00095   TreeNode* FindR(TreeNode *p, Trial* x) const;
00098   TreeNode* FindIn(TreeNode *p, Trial* x) const;
00099 public:
00100
00101   SearchData(int _NumOfFuncs, int _MaxSize = DefaultSearchDataSize);
00102   SearchData(int _NumOfFuncs, int _MaxSize, int _queueSize);
00103   ~SearchData();
00104
```

```
00106   void Clear();
00108   SearchInterval* InsertInterval(SearchInterval &pInterval);
00110   void UpdateInterval(SearchInterval &pInterval);
00112   SearchInterval* GetIntervalByX(Trial* x);
00115   SearchInterval* FindCoveringInterval(Trial* x);
00118   SearchInterval* GetIntervalWithMaxR();
00121   SearchInterval* GetIntervalWithMaxLocalR();
00122
00127   SearchInterval* InsertPoint(SearchInterval* coveringInterval, Trial& newPoint,
00128     int iteration, int methodDimension);
00129
00131   SearcDataIterator GetIterator(SearchInterval* p);
00133   SearcDataIterator GetBeginIterator();
00134
00143   void PushToQueue(SearchInterval *pInterval);
00145   void RefillQueue();
00147   void DeleteIntervalFromQueue(SearchInterval* i);
00148
00150   void PopFromGlobalQueue(SearchInterval **pInterval);
00152   void PopFromLocalQueue(SearchInterval **pInterval);
00154   void ClearQueue();
00156   void ResizeQueue(int size);
00157
00159   int GetCount();
00160
00162   double M[MaxNumOfFunc];
00164   double Z[MaxNumOfFunc];
00165
00167   void GetBestIntervals(SearchInterval** intervals, int count);
00169   void GetBestLocalIntervals(SearchInterval** intervals, int count);
00171   std::vector<Trial*>& GetTrials()
00172   {
00173     return trials;
00174   }
00175
00177   SearchInterval& FindMax();
00178
00180   bool IsRecalc()
00181   {
00182     return recalc;
00183   }
00185   void SetRecalc(bool f)
00186   {
00187     recalc = f;
00188   }
00190   Trial* GetBestTrial()
00191   {
00192     return BestTrial;
00193   }
00195   void SetBestTrial(Trial* trial);
00196
00198   void TrickleUp(SearchInterval* intervals);
00199
00201   int GetQueueSize();
00202
00204   double local_r;
00205 }; // SearchData
00206
00207
00208 #endif
00209 // - end of file ----------------------------------------------------------------------------
```

5.26 SearchInterval.h

```
00001
00002 //                                                          //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD          //
00004 //                                                          //
00005 //                 Copyright (c) 2015 by UNN.                   //
00006 //                   All Rights Reserved.                     //
00007 //                                                          //
00008 //  File:      data.h                                       //
00009 //                                                          //
00010 //  Purpose:   Header file for search data classes           //
00011 //                                                          //
00012 //  Author(s): Sysoyev A., Barkalov K., Sovrasov V.                  //
00013 //                                                          //
00015
00016 #ifndef __SEARCH_INTERVAL_H__
00017 #define __SEARCH_INTERVAL_H__
00018
00019
00020 #include "Extended.h"
00021 #include "BaseInterval.h"
```

```
00022 #include "Trial.h"
00023
00024
00025 struct TreeNode;
00026
00027 // ----------------------------------------------------------------------------------------
00028 class SearchInterval : public QueueBaseData
00029 {
00030 public:
00032   Trial* LeftPoint;
00034   Trial* RightPoint;
00035
00037   double   delta;
00038
00040   int      ind;
00042   int     K;
00044   double   R;
00046   double   locR;
00047
00049   TreeNode *treeNode;
00050
00051
00052
00054   Extended xl()
00055   {
00056     return LeftPoint->X();
00057   }
00059   Extended xr()
00060   {
00061     return RightPoint->X();
00062   }
00063
00065   double   zl()
00066   {
00067     if (LeftPoint->index >= 0)
00068       return LeftPoint->FuncValues[LeftPoint->index];
00069     else
00070       return MaxDouble;
00071   };
00073   int izl()
00074   {
00075     return LeftPoint->index;
00076   }
00078   double   zr()
00079   {
00080     if (RightPoint->index >= 0)
00081       return RightPoint->FuncValues[RightPoint->index];
00082     else
00083       return MaxDouble;
00084   };
00086   int izr()
00087   {
00088     return RightPoint->index;
00089   }
00090
00092   double* z()
00093   {
00094     //double   *z;
00095     return LeftPoint->FuncValues;
00096   }
00097
00099   int discreteValuesIndex()
00100   {
00101     if (LeftPoint->discreteValuesIndex != RightPoint->discreteValuesIndex)
00102       throw "Error SearchInterval discreteValuesIndex!!!\n";
00103     return LeftPoint->discreteValuesIndex;
00104   }
00105
00106
00107   virtual bool operator == (SearchInterval &p)
00108   {
00109     return *LeftPoint == *(p.LeftPoint);
00110   }
00111
00112   virtual bool operator > (SearchInterval &p)
00113   {
00114     return *LeftPoint > *(p.LeftPoint);
00115   }
00116
00117   virtual bool operator < (SearchInterval &p)
00118   {
00119     return *LeftPoint < *(p.LeftPoint);
00120   }
00121
00122   void CreatePoint();
00123
00124   SearchInterval();
```

```
00125   SearchInterval(const SearchInterval &p);
00126   ~SearchInterval();
00127 };
00128
00129 #endif
00130 // - end of file ----------------------------------------------------------------------------
```

5.27 SearchIntervalFactory.h

```
00001
00002 //                                                                      //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD             //
00004 //                                                                      //
00005 //                    Copyright (c) 2015 by UNN.                        //
00006 //                      All Rights Reserved.                            //
00007 //                                                                      //
00008 //  File:      data.h                                                   //
00009 //                                                                      //
00010 //  Purpose:   Header file for search data classes                       //
00011 //                                                                      //
00012 //  Author(s): Sysoyev A., Barkalov K., Sovrasov V.                     //
00013 //                                                                      //
00015
00016 #ifndef __SEARCH_INTERVAL_FACTORY_H__
00017 #define __SEARCH_INTERVAL_FACTORY_H__
00018
00019 #include "SearchInterval.h"
00020
00021 // ---------------------------------------------------------------------------------------------
00022 class SearchInterval;
00023 class SearchIntervalFactory
00024 {
00025 public:
00026   static SearchInterval* CreateSearchInterval(SearchInterval& interval);
00027   static SearchInterval* CreateSearchInterval();
00028 };
00029
00030
00031
00032 #endif
00033 // - end of file ----------------------------------------------------------------------------
```

5.28 Файл globalizer/method/include/SearchIteration.h

```
#include <vector>
#include <fstream>
#include <algorithm>
#include "Trial.h"
#include "SearchInterval.h"
```

Классы

- struct SearchIteration

5.28.1 Подробное описание

Авторы

Дата

Авторство

ННГУ им. Н.И. Лобачевского

5.29 SearchIteration.h

См. документацию.

```
00001
00002 //                                                          //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD          //
00004 //                                                          //
00005 //                 Copyright (c) 2015 by UNN.                   //
00006 //                   All Rights Reserved.                     //
00007 //                                                          //
00008 //  File:                                           //
00009 //                                                          //
00010 //  Purpose:   Header file for method class                  //
00011 //                                                          //
00012 //  Author(s):                          //
00013 //                                                          //
00015
00027
00028
00029 #ifndef __SEARCH_ITERATION_H__
00030 #define __SEARCH_ITERATION_H__
00031
00032 #include <vector>
00033 #include <fstream>
00034 #include <algorithm>
00035
00036 #include "Trial.h"
00037 #include "SearchInterval.h"
00038
00039 // -------------------------------------------------------------------------------------------
00040
00041
00045 struct SearchIteration
00046 {
00048   int IterationCount;
00049
00056   std::vector<Trial*> pCurTrials;
00057
00058 };
00059
00060 #endif
00061 // - end of file ------------------------------------------------------------------------------
```

5.30 Файл globalizer/method/include/Task.h

Объявление класса Task.

```
#include "Parameters.h"
#include "Common.h"
#include "ProblemInterface.h"
#include "Exception.h"
#include "BaseInterval.h"
```

Классы

- class Task
  Класс, инкапсулирующий информацию о задаче оптимизации.

5.30.1 Подробное описание

Объявление класса Task.

Авторы

Сысоев А., Баркалов К.

Дата

2015-2016

Авторство

ННГУ им. Н.И. Лобачевского

Объявление класса Task, который представляет собой обертку над решаемой задачей оптимизации (#IProblem).

5.31 Task.h

См. документацию.

```
00001
00002 //                                                      //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD          //
00004 //                                                      //
00005 //                 Copyright (c) 2015 by UNN.                 //
00006 //                   All Rights Reserved.                   //
00007 //                                                      //
00008 //  File:     task.h                                      //
00009 //                                                      //
00010 //  Purpose:  Header file for optimization task class          //
00011 //                                                      //
00012 //  Author(s): Sysoyev A., Barkalov K.                         //
00013 //                                                      //
00015
00016
00029
00030 #ifndef __TASK_H__
00031 #define __TASK_H__
00032
00033 #include "Parameters.h"
00034 #include "Common.h"
00035 #include "ProblemInterface.h"
00036 #include "Exception.h"
00037 #include "BaseInterval.h"
00038
00039 class Trial;
00040
00041 // ----------------------------------------------------------------------------------------------
00042
00051
00052 class Task: public QueueBaseData
00053 {
00054 protected:
00056   double     A[MaxDim];
00058   double     B[MaxDim];
00060   int        NumOfFunc;
00062   IProblem*  pProblem;
00064   double     OptimumValue;
00066   double     OptimumPoint[MaxDim];
00068   bool       IsOptimumValueDefined;
00070   bool       IsOptimumPointDefined;
00072   int ProcLevel;
00074   bool isInit;
00075
00076 public:
00077   int num;
00078
00084   Task(IProblem* _problem, int _ProcLevel);
00085
00090   Task();
00091
00095   virtual ~Task();
00096
00101   virtual Task* Clone();
00102
00107   virtual Task* CloneWithNewData();
00108
00114   virtual void Init(IProblem* _problem, int _ProcLevel);
00115
00120   virtual int GetN() const;
00121
00126   virtual const double* GetA() const;
00127
00132   virtual const double* GetB() const;
00133
00138   virtual double GetOptimumValue() const;
00139
00143   virtual void resetOptimumPoint();
00144
00150   virtual const double* GetOptimumPoint() const;
00151
00156   virtual bool GetIsOptimumValueDefined() const;
```

```
00157
00162
00163   virtual bool GetIsOptimumPointDefined() const;
00164
00169   virtual IProblem* getProblem();
00170
00175   virtual int GetNumOfFunc() const;
00176
00181   virtual void SetNumofFunc(int nf);
00182
00187   int GetProcLevel();
00188
00193   virtual int GetNumOfFuncAtProblem() const;
00194
00201   virtual double CalculateFuncs(const double* y, int fNumber);
00202
00211   virtual void CalculateFuncsInManyPoints(double* y, int fNumber, int numPoints, double* values);
00212
00218   virtual int GetNumberOfDiscreteVariable();
00219
00225   virtual int GetNumberOfValues(int discreteVariable);
00226
00233   virtual int GetAllDiscreteValues(int discreteVariable, double* values);
00234
00241   virtual bool IsPermissibleValue(double value, int discreteVariable);
00242
00247   virtual double* getMin();
00248
00253   virtual double* getMax();
00254
00259   virtual bool IsInit();
00260
00265   virtual bool IsLeaf();
00266
00273   virtual void CopyPoint(double* y, Trial* point);
00274 };
00275
00276 #endif
00277 // - end of file ----------------------------------------------------------------------------
```

5.32 TaskFactory.h

```
00001
00002 //                                                      //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD          //
00004 //                                                      //
00005 //                    Copyright (c) 2015 by UNN.                    //
00006 //                      All Rights Reserved.                      //
00007 //                                                      //
00008 //  File:      method_factory.h                                    //
00009 //                                                      //
00010 //  Purpose:   Header file for method factory class                 //
00011 //                                                      //
00012 //  Author(s): Lebedev I.                                           //
00013 //                                                      //
00015
00016 #ifndef __TASK_FACTORY_H__
00017 #define __TASK_FACTORY_H__
00018
00019 #include "Task.h"
00020
00021 class TaskFactory
00022 {
00023 public:
00024   static int num;
00025   static std::vector<int> permutations;
00026   static Task* CreateTask(IProblem* _problem, int _ProcLevel);
00027   static Task* CreateTask();
00028   static Task* CreateTask(Task* t);
00029 };
00030
00031 #endif
00032 // - end of file ----------------------------------------------------------------------------
```

5.33 TreeNode.h

```
00001
00002 //                                                      //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD          //
00004 //                                                      //
00005 //                    Copyright (c) 2025 by UNN.                    //
```

```
00006 //                    All Rights Reserved.                     //
00007 //                                                             //
00008 //  File:      TreeNode.h                                      //
00009 //                                                             //
00010 //  Purpose:   Header file for search data classes              //
00011 //                                                             //
00012 //  Author(s): Sysoyev A., Barkalov K., Sovrasov V., Zaitsev A.  //
00013 //                                                             //
00015
00016 #ifndef __TREE_NODE_H__
00017 #define __TREE_NODE_H__
00018
00019 #include "SearchInterval.h"
00020 #include "SearchIntervalFactory.h"
00021
00022 class SearchInterval;
00023 // ----------------------------------------------------------------------------------------
00024 class TreeNode
00025 {
00026 public:
00028   SearchInterval* pInterval;
00029
00031   unsigned char Height;
00032
00034   TreeNode* pLeft;
00036   TreeNode* pRight;
00038   TreeNode* pParent;
00039
00040   TreeNode(SearchInterval& p);
00041
00042   ~TreeNode();
00043 };
00044
00045 #endif
00046 // - end of file ------------------------------------------------------------------------------
```

5.34 Trial.h

```
00001
00002 //                                                             //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD          //
00004 //                                                             //
00005 //                    Copyright (c) 2025 by UNN.                   //
00006 //                      All Rights Reserved.                     //
00007 //                                                             //
00008 //  File:      Trial.h                                         //
00009 //                                                             //
00010 //  Purpose:   Header file for search data classes              //
00011 //                                                             //
00012 //  Author(s): Lebedev I. Sysoyev A., Barkalov K., Sovrasov V.   //
00013 //                                                             //
00015
00016 #ifndef __TRIAL_H__
00017 #define __TRIAL_H__
00018
00019 #include "Common.h"
00020 #include "Extended.h"
00021
00022 #include "Task.h"
00023
00024 #include <cstring>
00025
00026
00027
00028 class SearchInterval;
00029 // ----------------------------------------------------------------------------------------
00030 class Trial
00031 {
00032 protected:
00034   Extended x;
00035
00036 public:
00038   int discreteValuesIndex;
00040   double y[MaxDim];
00042   double FuncValues[MaxNumOfFunc];
00044   int index;
00046   int K;
00047
00049   int lowAndUpPoints;
00050
00052   Task* generatedTask;
00053
00054
00056   SearchInterval* leftInterval;
```

```
00057
00059   SearchInterval* rightInterval;
00060
00062   int TypeColor;
00063
00065   Trial();
00066
00068   Trial(const Trial& trial);
00069
00071   virtual Trial* Clone();
00072
00073   ~Trial();
00074
00080   void SetX(Extended d);
00081
00083   virtual Trial& operator = (Extended d);
00084
00086   virtual Extended  X();
00087
00089   virtual double GetFloor();
00090
00092   virtual double GetValue();
00093
00095   virtual Trial* GetLeftPoint();
00096
00098   virtual Trial* GetRightPoint();
00099
00101   virtual Trial& operator = (const Trial& trial);
00102
00104   virtual bool operator == (Trial& t);
00105
00107   virtual bool operator > (Trial& t);
00108
00110   virtual bool operator < (Trial& t);
00111
00112 };
00113
00114
00115
00116 #endif
00117 // - end of file ----------------------------------------------------------------------------
```

5.35 TrialFactory.h

```
00001
00002 //                                                                      //
00003 //          LOBACHEVSKY STATE UNIVERSITY OF NIZHNY NOVGOROD             //
00004 //                                                                      //
00005 //                    Copyright (c) 2015 by UNN.                        //
00006 //                      All Rights Reserved.                          //
00007 //                                                                      //
00008 //  File:     data.h                                                    //
00009 //                                                                      //
00010 //  Purpose:   Header file for search data classes                      //
00011 //                                                                      //
00012 //  Author(s): Sysoyev A., Barkalov K., Sovrasov V.                     //
00013 //                                                                      //
00015
00016 #ifndef __TRIAL_FACTORY_H__
00017 #define __TRIAL_FACTORY_H__
00018
00019 #include "Parameters.h"
00020 //#include "Trial.h"
00021
00022
00023 class Trial;
00024 class TrialFactory
00025 {
00026 public:
00027   //static Trial* CreateTrial(Trial& interval);
00028   static Trial* CreateTrial()
00029   {
00030     return new Trial();
00031   }
00032
00033   static Trial* CreateTrial(const OBJECTIV_TYPE* startPoint)
00034   {
00035     Trial* res;
00036     res = new Trial();
00037     memcpy(res->y, startPoint, parameters.Dimension * sizeof(double));
00038
00039     return res;
00040   }
00041
```

```
00042
00043   static Trial* CreateTrial(Trial* point)
00044   {
00045     return new Trial(*point);
00046   }
00047 };
00048
00049
00050
00051 #endif
00052 // - end of file ------------------------------------------------------------------------------
```