МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафедра: Математического обеспечения ЭВМ и суперкомьютерных технологий**

Направление подготовки: «Фундаментальная информатика и информационные технологии»

Профиль подготовки: «Название профиля подготовки бакалавра»

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

**Тема:**

**«Архитектура учебно-исследовательской системы Абсолют»**

Допущена к защите Выполнил:

Заведующий кафедрой: студент группы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Пронина Мария Валерьевнаученая. степень, ученое звание

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ подпись подпись

Научный руководитель:

ассистент Козинов Евгений Александрович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

Рецензент:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ученая степень, ученое звание

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород  
2017

# Содержание

[Введение 4](#_Toc482864962)

[Постановка задачи 5](#_Toc482864963)

[Глава 1. Глобальная оптимизация 6](#_Toc482864964)

[Задача глобальной оптимизации 6](#_Toc482864965)

[Общая схема численных методов оптимизации 6](#_Toc482864966)

[Характеристические алгоритмы глобального поиска 7](#_Toc482864967)

[Решение многомерной задачи глобальной оптимизации 8](#_Toc482864968)

[Глава 2. Архитектура программного комплекса 9](#_Toc482864969)

[Компоненты системы. 9](#_Toc482864970)

[Компонента алгоритмов глобальной оптимизации Algorithms. 12](#_Toc482864971)

[Компонента функций Functions. 13](#_Toc482864972)

[Компонента визуализации функций Rendering. 14](#_Toc482864973)

[Компонента модели данных AbsolutKernel. 14](#_Toc482864974)

[Компонента обработки пользовательского ввода Presenters. 16](#_Toc482864975)

[Компонента пользовательского интерфейса View. 16](#_Toc482864976)

[Взаимодействие компонент системы. 16](#_Toc482864977)

[Расширение системы с разработанной архитектурой 18](#_Toc482864978)

[Добавление алгоритмов глобальной оптимизации 18](#_Toc482864979)

[Добавление нового алгоритма оптимизации из сторонней системы 22](#_Toc482864980)

[Добавление новой функции 25](#_Toc482864981)

[Добавление нового способа задания функции 26](#_Toc482864982)

[Глава 3. Разработка компоненты отображения алгоритмов глобального поиска 28](#_Toc482864983)

[Выбор технологии 28](#_Toc482864984)

[Описание графической компоненты. 29](#_Toc482864985)

[Пользовательские элементы управления для инициализации OpenGL. 35](#_Toc482864988)

[Внедрение отображений в основную программу 38](#_Toc482864989)

[Заключение 41](#_Toc482864990)

[Список литературы 42](#_Toc482864991)

# Введение

Сейчас существует много задач, которые требуют применения сложных вычислений и суперкомпьютерных технологий. Одним из таких ресурсоемких методов является глобальная оптимизация, применяемая для решения задач в механике, экономике, проектировании инженерных сооружений. Применение глобальной оптимизации позволяет найти такие параметры систем различных типов, которые позволяет достичь требуемых значений каких-либо характеристик, например, сейсмоустойчивости задания или веса конструкции.

Глобальная оптимизация - задача нахождения экстремума целевой функции в некоторой области конечномерного векторного пространства, ограниченной набором неравенств. Сейчас разработано множество алгоритмов глобального поиска, например, метод Пиявского[12], генетический алгоритм[5], метод Стронгина [1;3]. Многие из них обладают различными наборами параметров и следуют разным стратегиям поиска, но все они достаточно сложны и требуют тщательного изучения перед их практическим использованием.

Для помощи в изучении данной темы требуется создать приложение Абсолют, которое создало бы условия для понимания студентами принципов работы основных алгоритмов глобальной оптимизации и помогало бы проводить небольшие вычислительные эксперименты для лучшего понимания сфер применения тех или иных алгоритмов. Уже было создан вариант Абсолюта для решения оптимизационных задач, в которых требуется найти экстремум одномерной функции.

Сейчас требуется продолжить создание этой системы и расширить ее на случай многомерных функций. Основная задача, поставленная перед приложением Абсолют - создать программный комплекс, обеспечивающий возможность изучения алгоритмов глобальной оптимизации, их сравнения и анализа, в удобной визуальной форме.

# Постановка задачи

Требуется создать демонстрационное приложение Абсолют для изучения алгоритмов глобальной оптимизации, позволяющее в наглядном виде демонстрировать их работу. Тестовое приложение будет поддерживать алгоритмы поиска без использования ограничений на многомерных функциях с функциональностью, аналогичной вычислительному комплексу Абсолют для оптимизации одномерных функций [4].

Так как планируется в дальнейшем расширять исходную систему путем добавления новых методов решения, поддержку новых размерностей и отображений процесса решения, большое внимание требуется уделить архитектуре программы, что и будет рассмотрено в данной работе.

Задача, стоящая на текущем этапе разработки, заключается в создании общей схемы архитектуры программной системы Абсолют, которая должна обладать следующей функциональностью:

1. добавление новых алгоритмов оптимизации без существенной переработки существующей системы;
2. поддержка изменения параметров алгоритмов для всех алгоритмов системы;
3. возможность работы с функциями различных размерностей;
4. возможность добавления новых визуальных отображений;
5. одновременное проведение нескольких экспериментов;
6. создание пользовательских функций.

# Глава 1. Глобальная оптимизация

Так как проектирование системы основывается на выполняемых ею функциях, перед непосредственным проектированием нужно изучить решаемую задачу и специфику алгоритмов глобальной оптимизации. Подробнее о решении задач глобальной оптимизации описано в [1].

## Задача глобальной оптимизации

Пусть задана некоторая многоэкстремальная функция на своей области определения D. Функция действительна и принимает во всех точках области определения конечные значения. Также она должна удовлетворять обобщенному условию Липшица на всем интервале поиска:

где и - любые числа из области определения, K - константа, , f имеет обратную себе

Кроме области определения функции, может быть задан набор ограничений, позволяющий более точно описать область поиска.

Задачей оптимизации будем называть задачу следующего вида: найти заданные экстремальные характеристики функции на множестве D. В зависимости от искомых экстремальных характеристик возможны различные конкретные постановки задачи оптимизации.

## Общая схема численных методов оптимизации

Методы представляются как итерационная процедура, которая осуществляет вычисление в области определения определенные характеристики минимизируемой функции. В самом общем виде численный метод решения задачи представляет собой набор

в котором - правила выбора точек испытаний, - правила построения оценки экстремума, - правила остановки вычислительного процесса.

Порядок проведения испытаний состоит в следующем:

1. Выбирается точка первого испытания.
2. По выбранной точке k-го испытания производится вычисление значения функции. На основе поисковой информации с предыдущих испытаний сужается класс функций, к которому принадлежит g, до множества
3. Определятся текущая оценка экстремума
4. Вычисляется точка очередного испытания
5. Определяется величина принимающая два значения - 0 или 1. Если 1, то номер испытания увеличивается на 1 и переходим к шагу 2. Если 0, то вычисления прекращаются.

## Характеристические алгоритмы глобального поиска

Хотя сейчас разработано множество различных методов решения рассматриваемой задачи, проектируемая программа рассчитана на такой класс алгоритмов глобального поиска, как характеристические.

Алгоритм решения задачи называется характеристическим, если начиная с некоторого шага поиска , выбор координаты следующего испытания можно описать в виде следующей последовательности действий:

1. Задается набор конечного числа точек из области определения исследуемой функции и упорядочивается по возрастанию координаты. В него могут входить и точки, в которых не проводились испытания, например, границы области определения оптимизируемой функции.
2. Каждому интервалу из этого набора ставится в соответствие некоторое число , называемое характеристикой этого интервала
3. Определяется интервал с максимальной характеристикой
4. Следующее испытание проводится в точке интервала с максимальной характеристики, которая определяется по некоторому правилу.

Характеристическими являются такие алгоритмы, как метод перебора, метод Кушнера, метод ломаных и базовый информационно-статистический алгоритм глобального поиска Стронгина, подробно описанные в [1;2;3]. В последнем в качестве функции используется

где величина вычисляется следующим образом:

Где - параметр метода.

## Решение многомерной задачи глобальной оптимизации

Для использования методов решения задач оптимизации одномерных функций для многомерных используется приведение функции многих переменных к функции одной переменной. Для этого применяются развертки типа кривой Пеано.

Многомерная область определения функции переводится в отрезок вещественной оси. Подобласти отображаются в одномерные интервалы. В зависимости от степени разбиения m число подобластей, на которые делится исходная область, различно, оно составляет , где N — размерность отображаемой области. Таким образом, можно, выбрав параметр разбиения m, отобразить точку в центре каждой подобласти (сетку на гиперкубе) в точку в центре каждого интервала на отрезке (сетку на отрезке). Такое отображение будет взаимно однозначным. Многомерная целевая функция будет определена на одномерной сетке.

# Глава 2. Архитектура программного комплекса

Необходимость в разработке архитектуры для приложения Абсолют вытекает из требования к универсальности разрабатываемого приложения. Различные алгоритмы глобальной оптимизации используют широкий набор параметров и значительно отличаться друг от друга процессом поиска решения. Также требуется предусмотреть свободу формирования новых постановок задач оптимизации - система не должна ограничиваться встроенным набором задач. Добавление нового алгоритма или задачи в программный комплекс не должно вызывать значительных изменений.

В рамках работы для разработки архитектуры приложения широко использовались шаблоны проектирования. Рассматривается проблема разработки архитектуры вычислительного комплекса – выбор архитектурного стиля, способа его реализации и выделение основных модулей комплекса. Далее приводится подробное описание выделенных компонент и способы их взаимодействия. В конце главы описываются примеры расширения системы путем добавления новых задач и методов.

## Компоненты системы.

Программно-вычислительный комплекс проектировался как набор связанных между собой компонент (компонентная архитектура). Такой подход позволил вести независимую разработку отдельных модулей приложения.

Выделение основных блоков в программе происходило при помощи построения и изучения use-case диаграммы (рисунок 1).

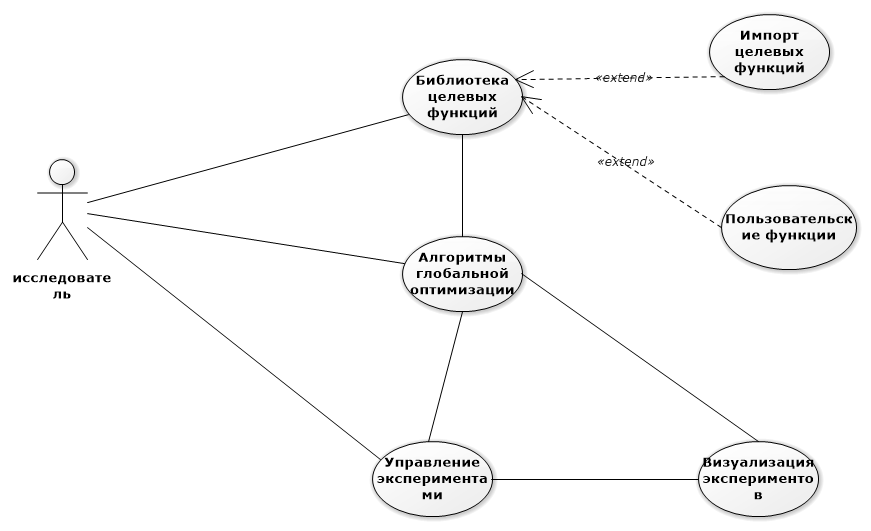


Рис. 1: Use-case диаграмма.

Естественным образом отделяются друг от друга графический интерфейс системы и вычислительное ядро. Вычислительная часть, согласно исследованиям предметной области, распадается на компоненту алгоритмов глобальной оптимизации и библиотеку целевых функций. Проектирование на основе предметной области является одним из подходов к архитектуре программного обеспечения, который успешно применяется для разработки вычислительных комплексов [11]. Разбиение пользовательского приложения на составные части произведено в соответствии с паттерном проектирования MVP и включает в себя отдельные компоненты для модели данных, визуальных элементов и обработки пользовательского ввода. В графической части выделен самостоятельный модуль визуализации функций и процесса работы численных методов для его повторного использования в других приложениях.

Основной особенностью паттерна МVP является отсутствие коммуникации между слоем данных и слоем отображений - они полностью независимы (рисунок 2). Это позволит в дальнейшем изменить пользовательский интерфейс программного комплекса без изменения работы с алгоритмами оптимизации, например, разработать приложение для смартфона или изменить API программирования пользовательского интерфейса. Разделение приложения на независимые слои является отличительной чертой послойной архитектуры [23].



Рис. 2: Взаимодействие основных компонент в паттерне MVP

Таким образом, программный комплекс состоит из следующих компонент:

1. Algorithms – алгоритмы оптимизации
2. Functions – целевые функции
3. AbsolutKernel – модель данных приложения
4. View – пользовательский интерфейс
5. Presenters – взаимодействие между моделью данных и пользовательским интерфейсом
6. Rendering – визуализация функций.

Отношения между ними представлены на диаграмме компонент (рисунок 3).

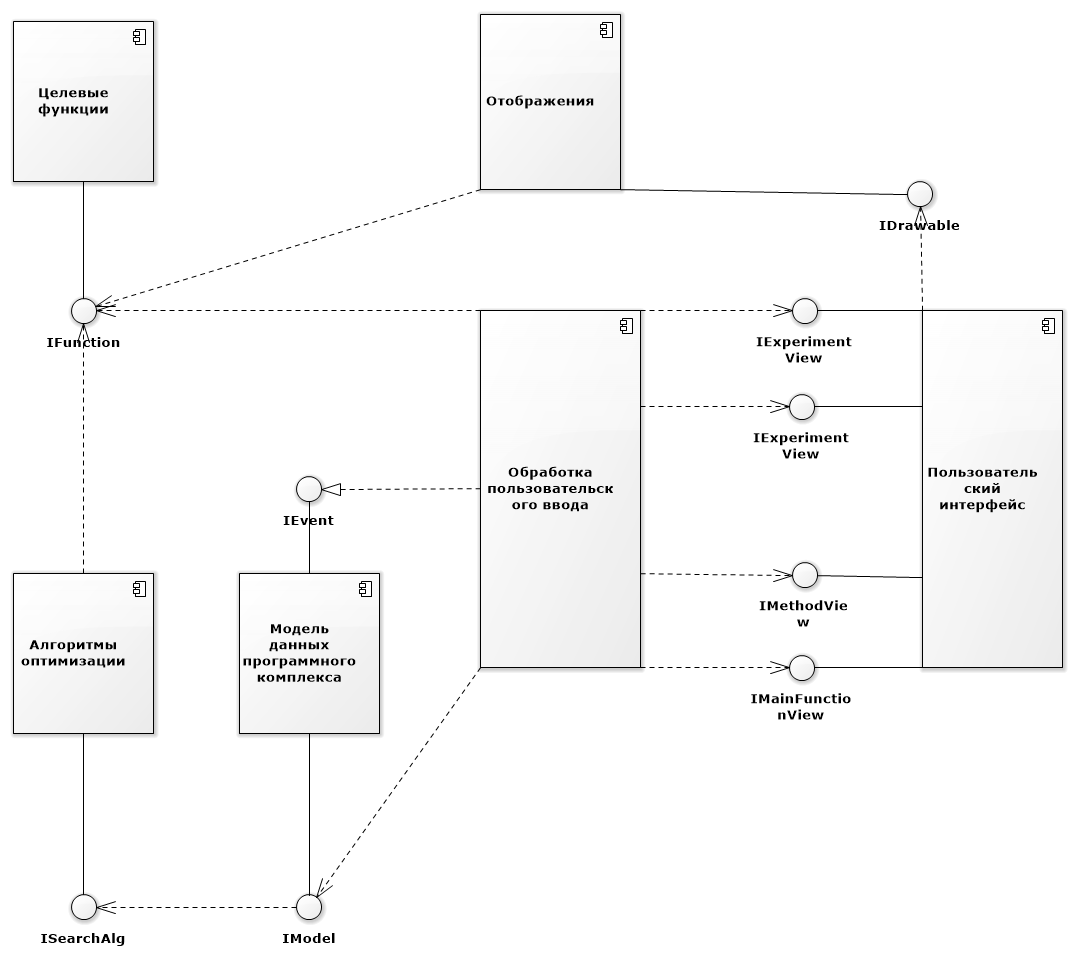


Рис. 3: Диаграмма компонент.

## Компонента алгоритмов глобальной оптимизации Algorithms.

Содержит реализацию основных методов глобальной оптимизации и предоставляет интерфейс для взаимодействия с состоянием поиска.

Листинг 1. Интерфейс ISearchAlg.

public interface ISearchAlg

{ string Name { get; }  *//итерация по вычисленным значениям* void Restart(); void NextStep(); void PrevStep(); bool HasNext(); int CurrentIter(); IEnumerable<MethodPoint> CalculatedPoints(int start = 0); MethodPoint LastPoint(); *//получение оценки решения задачи* MethodPoint Argmin(); MethodPoint ArgminTask();  *// установка свойств алгоритма* void SetProperty(string name, object value); void SetProperties(Dictionary<string, object> propsValues); object GetProperty(string name); List<PropertyInfo> GetPropertiesInfo(); *// информация о достижении известного минимума* int IterOverMin(); bool IsIterOverMin(); *// перерасчет точек и информация о завершении работы* bool IsEnded(); void RecalcPoints();  *// клонирование алгоритма* ISearchAlg Clone(); }

Все разработанные алгоритмы реализуют интерфейс, представленный выше (листинг 1). Это позволяет создать универсальное представление численного метода, независимое от его внутренней организации. ISearchAlg предоставляет информацию об оценке решения, вычисленных точках и позволяет устанавливать свойства метода – данные операции характерны для всех алгоритмов оптимизации.

Для удобства разработки существует два базовых класса с реализациями методов интерфейса по умолчанию. Это классы GenericAlg и CharacteristicAlg. GenericAlg берет на себя работу с хранением свойств и их получением, дает базовые возможности по заполнению структуры данных с точками испытаний. Во втором базовом классе представлена характеристическая схема алгоритмов глобальной оптимизации, что позволяет быстро добавлять новые алгоритмы такого типа.

Создание объектов-алгоритмов происходит через класс-фабрику AlgorithmFactory по имени алгоритма или через специальные методы (сторонние системы с параметрами запуска). Паттерны проектирования «абстрактная фабрика» и «фабричный метод», использованные при проектировании AlgorithmFactory, подробно описаны в [6; 13].

## Компонента функций Functions.

Предоставляет набор из готовых функций для проведения оптимизационных экспериментов и функциональность по созданию пользовательских функций: импорт из динамических библиотек и ввод математических выражений. Функции для тестового набора выбраны из опубликованных списков тестовых задач для глобальной оптимизации и сборников функций двух переменных [10;16;18].

Интерфейс компоненты (листинг 2) отражает основные действия над функциями, необходимые для работы алгоритмов оптимизации.

Листинг 2. Интерфейс компоненты Functions.

public interface IFunction  
 {  
 // границы области определения  
 List<double> Right { get; set; }  
 List<double> Left { get; set; }  
 // известный минимум  
 List<double> Min\_x { get; set; }  
 // подсчет значения  
 double Calc(List<double> arg);  
 double Calc(double x);  
 // операции с разверткой Пеано  
 List<double> GetImage(double x);  
 double GetPrototype(List<double> x);  
 // поддержка DLL с несколькими функциями  
 int GetNumberOfFunctions();  
 void SetFunctionNumber(int index);  
 }

Как и в предыдущей компоненте, для ускорения добавления новых функций существует специальный класс MultidimFunction, в которым основные методы уже реализованы – это методы для работы с развертками (GetImage и GetPrototype).

Создание объектов происходит аналогично компоненте Algorithms.

## Компонента визуализации функций Rendering.

Содержит различные графические объекты на технологии OpenGL, связанные с отображением функций и точек испытаний и предоставляет специальные пользовательские элементы управления для их встраивания в приложения. Более подробно компонента описана в главе 3.

## Компонента модели данных AbsolutKernel.

Включает в себя модель данных программы и интерфейс слушателя изменений данных. Интерфейс компоненты (листинг 3) описывает действия с экспериментами, дает доступ к результатам вычислений и позволяет настраивать алгоритмы оптимизации.

Листинг 3. Интерфейс модели данных.

public interface IModel  
 {  
 // подписка на изменение данных  
 void Subscribe(IEventListener e);  
 void Unsubcribe(IEventListener e);  
 void SubscribeExperiment(IEventListener e, Guid experimentId);  
 // получение имен экспериментов и их номеров  
 Dictionary<Guid, string> GetExperimentsNames();  
 // создание и удаление экспериментов  
 Guid AddExperiment(string name, ISearchAlg alg);  
 void DeleteExperiment(Guid experimentId);  
 // управление свойствами метода и установка нового метода   
 ISearchAlg GetAlg(Guid experimentId);   
 void SetAlg(Guid experimentId, ISearchAlg alg);  
 void SetProperty(string name, object value, Guid experimentId);  
 object GetProperty(string name, Guid experimentId);  
 void SetProperties(Guid experimentId, Dictionary<string, object> propsValues);  
 List<PropertyInfo> GetPropsNames(Guid experimentId);  
 // итератор по точкам испытания  
 void NextStep(Guid experimentId);  
 void PrevStep(Guid experimentId);  
 MethodPoint GetLastPoint(Guid experimentId);  
 bool HasNext(Guid experimentId);  
 int GetCurrentInter(Guid experimentId);  
 IEnumerable<MethodPoint> GetExpEnumerator(Guid experimentId, int start = 0);  
 // текущая оценка решения  
 MethodPoint GetBestPoint(Guid experimentId);  
 // установка срезов для многомерных функций   
 void SetSlicesDictionary(Dictionary<int, IFunction> slices, Guid expId);  
 Dictionary<int, IFunction> GetSlicesDictionary(Guid expId);  
 }

Разработанный интерфейс для модели данных позволяет управлять экспериментами и алгоритмами оптимизации, получать результаты работы численных методов. При изменении данных (например, при смене типа алгоритма во время работы метода SetAlg). Модель реализована при помощи паттерна singleton, его характеристики описаны в [6; 13].

При подписке на модели можно наблюдать за событиями, описанными в интерфейсе слушателя (листинг 4).

Листинг 4. Интерфейс слушателя модели.

public interface IEventListener  
 {  
 void OnPointsReset();  
 void OnMainFunctionChanged();  
 void OnSlicesChanged();  
 void OnMethodChanged();  
 void OnNextStep();  
 void OnPrevStep();  
 void OnExperimentAdded();  
 void OnExperimentDeleted();  
 }

При необходимости возможно добавить какое-то новое событие - достаточно изменить интерфейс, реализовать оповещение в модели и добавить реализацию нового метода из интерфейса там, где это необходимо.

## Компонента обработки пользовательского ввода Presenters.

Организует связь между слоем данных и слоем отображений. Основными задачами этой компоненты является получение сигналов от визуального интерфейса при вводе пользователя (действия при нажатии кнопок, выборе пунктов меню) и передача данных к отображениям (например, при изменении целевой функции уведомить графики функции и передать новый объект). Для получения событий об изменении данных классы этой компоненты реализуют описанный выше интерфейс IEventListener.

## Компонента пользовательского интерфейса View.

Состоит из приложения WindowsForms и представляет собой пользовательский интерфейс программного комплекса. Содержит в себе различные меню, окна для изменения параметров объектов системы, проведения экспериментов и задания функций.

## Взаимодействие компонент системы.

Для организации взаимодействия компонент между собой используется их объединение в слои (рисунок 4). Каждый слой взаимодействует с другим слоем при помощи интерфейсов компонент – конкретная реализация классов скрыта.



Рис. 4: Распределение компонент по слоям.

В общем случае процесс взаимодействия слоев выглядит следующим образом (рис. 5):



Рис. 5: Общая схема взаимодействия компонент

1. Пользователь взаимодействует с визуальным интерфейсом приложения
2. Слой отображения передает данные от пользователя слою представления
3. Слой представления обновляет данные в слое модели
4. Внутри слоя модели происходит взаимодействие с компонентами алгоритмов глобальной оптимизации и целевых функций
5. Далее слой модели посредством событий уведомляет слой представления об изменении данных
6. Слой представления обновляет визуальный интерфейс

Схема полностью идентична стандартному взаимодействию в паттерне MVP.

Рассмотрим реализацию этой схемы на конкретном примере обработки нажатия пользователя на кнопку перехода на следующий шаг работы алгоритма ">>" в окне эксперимента.

1. Пользователь нажимает на кнопку ">>", что приводит к вызову у формы эксперимента метода OnNextStepClick.
2. Форма эксперимента является отображением для точек эксперимента и вызывает метод своего презентера OnNextStep().
3. В методе OnNextStep() презентер вызывает метод модели NextStep, указывая привязанный номер эксперимента.
4. Модель обращается к численному методу этого эксперимента и вызывает у него метод для изменения текущей итерации NextStep.
5. После этого модель уведомляет слушателей эксперимента об изменении состояния данных.
6. Презентер эксперимента получает уведомление и вызывает у связанных с ним отображений (к ним относится и форма эксперимента) метод для добавления новой точки LastPoint. В нем он передает полученную от модели точку новой итерации. Форма эксперимента обновляется.

Описанный процесс проиллюстрирован на диаграмме последовательности (рис.6).

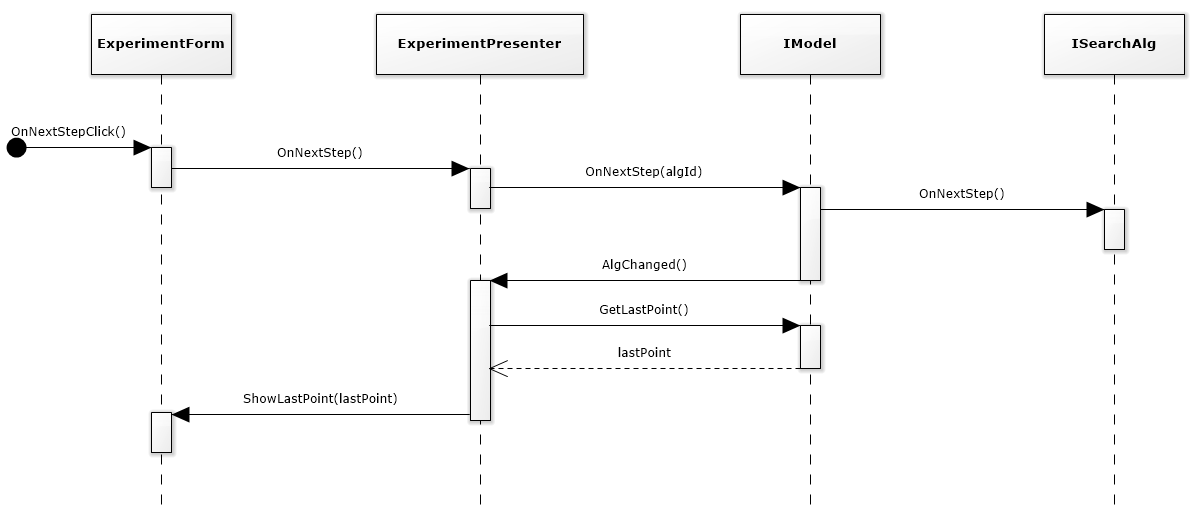


Рис. 6: Диаграмма последовательности взаимодействия частей приложения.

## Расширение системы с разработанной архитектурой

Для иллюстрации возможностей системы к расширению функциональности далее описаны процессы добавления новых элементов различных типов.

## Добавление алгоритмов глобальной оптимизации

В общем случае для добавления нового алгоритма необходимы следующие шаги:

1. создать класс, реализующего интерфейс ISearchAlg.
2. в конструкторе класса зарегистрировать его параметры.
3. если необходимо, создать новые типы параметров в Properties.
4. создать пользовательский элемент управления для ввода нового типа параметра.
5. добавить новый тип параметра в код окна для настройки параметров метода.
6. добавить в фабрику алгоритмов новую опцию с именем алгоритма.

Так как все характеристические алгоритмы обладают одной вычислительной схемой, для них создан специальный базовый класс CharacteristicAlg. Он упрощает расширение системы алгоритмами с характеристической представимостью. Достаточно переопределить методы расчета характеристики и расчет новой точки. Новые свойства добавляются аналогично предыдущей схеме.

В качестве примера рассмотрим создание характеристического алгоритма, определяющего точку решения путем последовательного сканирования области.

Первым шаг заключается в создании нового класса ScanningAlg. В качестве базового указывается CharacteristicAlg.

Листинг 5. ScanningAlg.

public class ScanningAlg : CharacteristicAlg  
 {  
 protected override double CalcNewPoint()  
 {  
 return 0.5\*(ExperimentPoints[maxCharIndex].evolventX +  
 ExperimentPoints[maxCharIndex - 1].evolventX);  
 }  
  
 protected override double CalcCharacteristic(int curIntervalIndex)  
 {  
   
 var right = ExperimentPoints[curIntervalIndex + 1];  
 var left = ExperimentPoints[curIntervalIndex];  
 return right.evolventX - left.evolventX;  
  
 }  
 }

Далее переопределяем виртуальные методы характеристического алгоритма для высчитывания характеристики и поиска новой точки. Доступ к уже вычисленным точкам происходит через сортированный список ExperimentPoints из CharacteristicAlg.

Последний шаг - добавление в фабрику алгоритмов новой опции с только что созданным классом для использования метода в вычислительном комплексе.

Листинг 6. Создание нового алгоритма для его использования в вычислительном комплексе.

public static class AlgFactory  
{  
 // в список опций добавляется Scanning alg  
 public static List<string> Options = new List<string>() {"Agp", "Scanning alg", "AGP\_paral", "Scanning\_alg\_paral", "Piavskiy" };  
 public static ISearchAlg Build(string algName)  
 {  
 if(algName == "Agp")  
 return new Agp();  
 // создание алгоритма нового типа  
 if(algName == "Scanning alg")  
 return new ScanningAlg();  
 if (algName == "AGP\_paral")  
 return new Algorithms.Methods.AGP\_2();  
 if (algName == "Scanning\_alg\_paral")  
 return new Algorithms.Methods.ScanningAlg\_Paral();  
 if(algName == "Piavskiy")  
 return new Piyavskiy();   
 throw new ArgumentException(algName);  
 }  
}

Если алгоритм не удовлетворяет характеристической схеме или нужно встроить уже существующий код, то для таких случаев используется другой базовый класс GenericAlg. Рассмотрим его применение при внедрении кода простого генетического алгоритма[[1]](#footnote-1). Первым шагом станет регистрация свойств в конструкторе GeneticSimpleAlg:

Листинг 7. Регистрация свойств в генетическом алгоритме.

class GeneticSimpleAlg : GenericAlg  
 {  
 protected IFunction f;  
 GA ga;  
  
 public GeneticSimpleAlg()  
 {  
  
 RegisterProperty(new DoubleProperty("crossoverRate", 0.0, 100.0, 1, DoubleDeltaType.Log, 0.8));  
 RegisterProperty(new FuncProperty("Function", null));  
 RegisterProperty(new DoubleProperty("mutationRate", 0.0, 100.0, 1, DoubleDeltaType.Log, 0.05));  
 RegisterProperty(new IntProperty("populationSize", 2, 300, 10, 100));  
 RegisterProperty(new IntProperty("generationSize", 1, 5000, 100, 20));  
   
 }

}

Далее создается специальный метод для расчета значений функции, который адаптирует используемый класс IFunction к требованиям используемого кода метода.

Листинг 8. Адаптирование расчета значений функции.  
 public double theActualFunction(params double[] values)  
 {  
 if (values.Length != 2)  
 throw new ArgumentOutOfRangeException("should only have 2 args");  
  
 double x = values[0];  
 double y = values[1];  
  
 List<double> values\_list = new List<double>();  
 values\_list.Add(x);  
 values\_list.Add(y);  
 return (-1.0) \* f.Calc(values\_list);  
  
 }

Затем требуется описать основную логику работы алгоритма в методе RecalcPoints, используя классы из добавляемого кода, обеспечив заполнение массива точек испытания.

Листинг 9. Реализация метода расчета точек испытания.  
 public override void RecalcPoints()  
 {  
 realIteration = 0;  
 ga = new GA((double)GetProperty("crossoverRate"), (double)GetProperty("mutationRate"),  
 (int)GetProperty("populationSize"), (int)GetProperty("generationSize"), 2);  
 if (f == null) return;  
 Restart();  
 ExperimentPoints.Clear();  
 ExperimentPointsReal.Clear();   
 ga.FitnessFunction = new GAFunction(theActualFunction);  
 ga.Elitism = true;  
 ga.Go(ref ExperimentPoints, ref realIteration);  
 ExperimentPointsReal = ExperimentPoints;  
 realIteration--;  
 }

Получается следующее деление функциональности - класс GenericAlg реализует работу со свойствами и хранение точек испытания, класс GeneticAlgSimple адаптирует интерфейс ISearchAlg к интерфейсу стороннего класса GA, выполняющего вычисления.

Все свойства нового алгоритма регистрируются в конструкторе класса-адаптера и затем передаются в класс GA.

## Добавление нового алгоритма оптимизации из сторонней системы

Другим вариантом добавления может стать создание специального окна:

1. создаем класс с интерфейсом ISearchAlg;
2. регистрируем параметры;
3. вместо добавления новой опции в фабрику добавляем в фабрику новый метод для сборки алгоритма из необходимых параметров
4. создаем новое окно с нужными пользовательскими элементами управления
5. добавляем его открытие в меню
6. в MethodPresenter добавляем новый метод для создания алгоритма из параметров, в котором вызываем новый метод фабрики
7. обрабатываем информацию, полученную из окна и передаем ее в MethodPresenter в метод создания нового типа алгоритма.

Таким способом встроен запуск системы Globalizer – он запускается из программно-вычислительного комплекса со специальными параметрами и его результаты отображаются в основном окне.

Как и в предыдущих реализациях, в конструкторе класса описываются все свойства метода (листинг 10).

Листинг 10. Конструктор класса Examin\_alg

class Examin\_alg : GenericAlg  
{   
 public Examin\_alg()  
 {  
 // регистрируем свойства для передачи их в аргументы запуска  
 RegisterProperty(new DoubleProperty("Precision", 0.000001, 0.1, 5, DoubleDeltaType.Log, 0.00001));  
 RegisterProperty(new FuncProperty("Function", null));  
 RegisterProperty(new StringProperty("Examin\_dll", ""));  
 RegisterProperty(new StringProperty("Examin\_conf", ""));  
 RegisterProperty(new StringProperty("Examin\_exe", ""));  
 RegisterProperty(new IntProperty("Dimension", 2, 2, 0, 2));  
 RegisterProperty(new IntProperty("Evolvent", 8, 12, 1, 10));  
 RegisterProperty(new DoubleProperty("r", 1.000001, 100, 100, DoubleDeltaType.Step, 2));  
 }

}

В основном методе класса запускается процесс с приложением Examin и анализируется сгенерированный им файл.

Листинг 11. Получение точек испытания при помощи сторонней системы.  
   
 public override void RecalcPoints()  
 {  
 if ((string) GetProperty("Examin\_conf") == "" ||  
 (string) GetProperty("Examin\_dll") == "" ||  
 (string) GetProperty("Examin\_exe") == "" ||  
 f == null) return;  
 Restart();  
 ExperimentPointsReal.Clear();

// запускаем процесс с прилолжением  
 Init();  
 string path = @"out\_point.txt";  
 if (File.Exists(path))  
  
 // считываем информацию из выходного файла, полученного после запуска системы и загружаем ее в структуру с точками испытаний  
 using (StreamReader sr = File.OpenText(path))  
 {  
 int iter = 0;  
 string firstLine = sr.ReadLine();  
 string[] info = firstLine.Split(' ');  
 realIteration = Convert.ToInt32(info[0]);  
  
 while (!sr.EndOfStream)  
 {  
 var s = sr.ReadLine();  
 string[] parts = s.Split('|');  
 if(parts.Length == 1) continue;  
 string[] point = parts[0].Split(' ');  
 MethodPoint mp;  
 mp = new MethodPoint  
 {  
 iteration = iter  
 };  
 mp.y = Convert.ToDouble(parts[1].TrimStart(' '), CultureInfo.InvariantCulture);  
 mp.x.Add(Convert.ToDouble(point[0], CultureInfo.InvariantCulture));  
 mp.x.Add(Convert.ToDouble(point[1], CultureInfo.InvariantCulture));  
 iter++;  
 mp.id\_process = 0;  
 ExperimentPointsReal.Add(mp);  
  
 }   
  
 }  
 }

Запуск системы представляет собой запуск нового процесса при помощи переданных в алгоритм параметров – имя исполняемого файла, путь до динамической библиотеки с целевой функцией, конфигурационный файл.  
  
  
Листинг 12. Запуск системы Examin.  
 private void Init()  
 {  
 if ((string) GetProperty("Examin\_conf") == "" || (string) GetProperty("Examin\_dll") == "" ||  
 (string) GetProperty("Examin\_exe") == "")  
 return;  
 //формируем аргументы для запуска exe-файла  
 var t = "-Dimension " + ((int) GetProperty("Dimension")).ToString() +  
 " -r " + ((double) GetProperty("r")).ToString() +  
 " -Eps " + ((double) GetProperty("Precision")).ToString() +  
 " -m " + (int) GetProperty("Evolvent") +  
 " -MaxNumOfPoints " + (int) GetProperty("Maximum Iteration") +  
 " -sip " + @"""out\_point.txt""" +  
 " - libConf " + @"""" + ((string)GetProperty("Examin\_conf")) + @"""" +  
 " -lib "+ @"""" +((string) GetProperty("Examin\_dll")) + @"""";  
 // запускаем процесс с системой  
 Process Examin\_Process = new Process() {StartInfo = new ProcessStartInfo()  
 {  
 FileName = (string)GetProperty("Examin\_exe"),  
 Arguments = t,  
 UseShellExecute = false,  
 RedirectStandardOutput = true,  
 CreateNoWindow = true  
 } };  
  
 Examin\_Process.Start();  
   
 while (!Examin\_Process.HasExited) ;  
 var output = "";  
 // записываем консольный вывод  
 var fileStream = File.Create(@"examinOutput.txt");  
  
 Examin\_Process.StandardOutput.BaseStream.CopyTo(fileStream);  
   
 }  
}

В фабрике алгоритмов добавляется новый метод, создающий объект ExaminAlg из пути до exe-файла, файла динамической библиотеки с задачей и конфигурационного файла для нее.

Листинг 13. Создание объекта реализованного алгоритма.

public static ISearchAlg BuildExaminAlg(string dll, string conf, string examin){ ISearchAlg a = new Examin\_alg(); a.SetProperty("Examin\_conf", conf); a.SetProperty("Examin\_dll", dll); a.SetProperty("Examin\_exe", examin); return a;}

Затем в программе реализуется специальный пункт меню с формой задания параметров и ее обработка в MethodPresenter:

Листинг 14. Обработка формы для запуска сторонней системы.

public void SetExaminMethod(string dllPath, string configPath, string examinPath)  
{  
 \_m.SetAlg(\_expId, AlgFactory.BuildExaminAlg(dllPath, configPath, examinPath));  
 \_m.SetProperty("Function", FunctionFactory.BuildFromDLL(dllPath, configPath), \_expId);  
}

После этого систему можно использовать для проведения экспериментов в программном комплексе Абсолют.

## Добавление новой функции

Добавление новой функции практически аналогично первому способу создания нового алгоритма.

1. создаем новый класс с интерфейсом IFunction. Если функция подходит под один из базовых классов, делаем новый класс наследником выбранного базового класса.
2. реализуем методы
3. добавляем в фабрику функций новую опцию.

В качестве образца такого типа расширения создадим двумерную плоскость f(x,y) = x+y в квадрате |x|<1, |y|<1.

Листинг 15.Пример реализации функции f(x,y) = x+y.

public class Plane : MultidimFunction  
 {   
 //в конструкторе задаем область определения функции  
 // Left - левая нижняя точка допустимой области, Right - верхняя правая   
 // свойства Left и Right определены в классе MultidimFunction  
 public Plane()  
 {   
 Left = new List<double>(){-1,-1};  
 Right = new List<double>(){1,1};  
 }  
 //в перегруженном методе Calc подсчитывается значение функции в точке arg  
 public override double Calc(List<double> arg)  
 {  
 return arg[0] + arg[1];  
 }  
 }

Так как функция двумерная, воспользуемся существующим классом MultidimFunction для реализации операций с развертками. После этого остается только создать новый конструктор и определить правило подсчета значения в точке. Многомерные точки задаются при помощи списков вещественных чисел.

Редактирование фабрики функции происходит аналогично добавлению нового алгоритма.

## Добавление нового способа задания функции

Если необходимо сделать более сложный вариант создания функции (ввод формулы, графический редактор и т.д.), снова реализуется класс с интерфейсом. Затем в фабрику добавляется новый метод, конструирующий функцию из нового типа параметров. В основном меню создаем новую опцию, под нее создаем окно. В MethodPresenter добавляем специальный метод для обработки ввода из нового окна и создания функции новым методом фабрики.

Такой способ был использован в задании функции при помощи вводимой пользователем математической формулы. Вычисления пользовательских формул проводятся при помощи библиотеки mxparser.

Как и в предыдущем случае, создается наследник класса MultidimFunction.

Листинг 16. Создание функции из пользовательского ввода.

public class CustomFunction : MultidimFunction  
  
 {  
 // границы допустимой области  
 private List<double> \_right;  
 private List<double> \_left;  
 // вспомогательные переменные для вычисления выражения  
 private Argument \_x;  
 private Argument \_y;  
 private Argument \_z;  
  
 // formula - строка с формулой  
 public CustomFunction(List<double> left, List<double> right, string formula)  
 {  
 \_right = right;  
 \_left = left;  
 Right = \_right;  
 Left = \_left;  
 formula = "z =" + formula;  
 \_x = new Argument("x");  
 \_y = new Argument("y");  
 \_z = new Argument(formula, \_x, \_y);   
  
 }  
 // подсчет значения в точке arg производится средствами mxparser  
 public override double Calc(List<double> arg)  
 {  
 \_x.setArgumentValue(arg[0]);  
 \_y.setArgumentValue(arg[1]);  
   
 return \_z.getArgumentValue();  
 }  
 }

Особенность данного конкретного случая заключается в передаче в конструктор класса строки с формульным выражением. Следующим шагом идет редактирование порождающего класса FunctionFactory - создается новый метод CreateFunctionFromTextInput:

Листинг 17. Изменения в фабрике функций для нового способа ввода.

public static IFunction BuildFromTextInput(List<double> left, List<double> right, string expression)  
{  
 return new CustomFunction(left, right, expression);  
}

В пользовательском интерфейсе для доступа к новой функциональности нужно создать специальное диалоговое окно с возможностью ввода границ области и формульного выражения и добавить пункт меню в Problems.

# Глава 3. Разработка компоненты графического отображения алгоритмов глобального поиска

Обучающая система для проведения вычислительных экспериментов предполагает широкий выбор графических средств для наблюдения за ходом эксперимента. Особенно необходимы визуальные способы представления целевых функций и последовательности точек, порождаемых методом оптимизации. Многомерность целевых функций требует различного подхода к их отображению для лучшего понимания процесса поиска, поэтому в системе реализованы следующие графические представления:

1. объемный график функции и точки испытания
2. карта температур и линии уровня, также с точками испытания
3. одномерный график функции, полученной при помощи развертки Пеано
4. распределение точек в виде диаграмма
5. срезы многомерных целевых функций секущими плоскостями.

В данной главе описана программная реализация составных частей для визуальных представлений и их внедрение в основную программу.

## Выбор технологии

После исследования уже существующих средств для визуализации данных (Dislin [19], OxyPlot, Lighting Chart, Sci Chart), выяснилось, что несмотря на широкий набор предлагаемых возможностей, ни одна система не реализует в полной мере необходимую функциональность. Проблема заключается в связанных между собой визуальных объектов с динамическим обновлением (точки испытания на графике целевой функции) и в повороте графических элементов в реальном времени. Кроме того, некоторые из упомянутых библиотек не поддерживаются в используемом интерфейсе программирования приложений Windows Forms. Известная система Dislin позволяет реализовать часть необходимых отображений, но не поддерживает поворот графиков в реальном времени. Остальные библиотеки или поддерживают только двумерную графику, или не свободно распространяемые. Возникла необходимость разработки собственной библиотеки визуализации.

Существует несколько популярных средств для разработки приложений с сложными графическими элементами. Это OpenGL, DirectX и Vulcan - современное развитие OpenGL. В качестве основы для разработки был выбран OpenGL за его кроссплатформенность, открытую реализацию и наличия опыта разработки с использованием этой платформы Основные особенности разработки приложений с использованием технологии OpenGL и базовые алгоритмы описаны в [8;9]. Сообществом и разработчиками поддерживается открытый учебник по OpenGL с подробными уроками и описанием современных подходов к компьютерной графике [15].

В программе будет использоваться библиотека-обертка над интерфейсом OpenGL для .NET - OpenTK, предоставляющая функционал, адаптированный к объектно-ориентированному языку программирования C# и математический аппарат для алгоритмов компьютерной графики.

## Описание графической компоненты.

Библиотека Rendering отвечает за графические представления точек испытания и целевой функции. В ней можно выделить три основные части: классы-утилиты для математических операций и работы с OpenGL, классы графических элементов и пользовательские элементы управления для использования в основной программе.

### **Работа с OpenGL и математические операции.**

Классы этого блока решают задачи инициализации контекста OpenGL и управления координатами пространства рисования и некоторых часто используемых операций с векторами и значениями функций. Здесь же производится настройка основных параметров графики - включение освещения, сглаживания, работа с буфером глубины и корректное масштабирование элементов управления.

Для реализации поворота фигур был создан специальный класс камеры, задачей которого становится применение матриц поворота. Камера всегда смотрит в начало координат. При нажатии на клавиши поворота камера начинается двигаться по окружностям вокруг графика - ее новое положение рассчитывается поворотом на соответствующий угол вектора от начала координат до камеры. При переходе через определенные углы меняется вектор направления вверх для естественного поворота без резкой смены ориентации фигуры на экране.

### **Графические элементы**

Для удобства каждый сложный графический элемент представлен отдельным классом. Условно их можно разделить на две группы:

1. отображения функции - классы Surface, TemperatureMap, Isobars, Curve.
2. отображения точек испытания - PointsSet2D, PointsSet3d, LinesOnAxis, LinesOnAxisOnScreen и Diagramm.

**Surface - график функции двух переменных.**

Представляет собой поверхность, заданную двумерной функцией z = f(x,y).

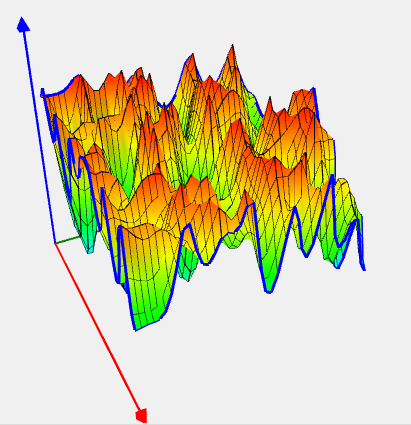


Рис. 7: Пример построенной поверхности

Функция задается объектом с интерфейсом IFunction (описан в главе 2). Поверхность строится следующей последовательностью шагов:

1. построение сетки значений функции в области определения
2. масштабирование узлов сетки
3. рисование полигонов с вершинами в узлах сетки средствами OpenGL.

Область определения разбивается на прямоугольники, в вершинах которых считаются значения функции. Далее сетка масштабируется для полного попадания графика в видимую область.

Листинг 18. Заполнение полигонов для рисования поверхности.

private void FillPolygons()  
{  
 \_polygons.Clear();  
 func = f;  
 double[] x = Utils.FillCoordArrays(func, 0,   
 (int) GetProperty("Grid points")),  
 y = Utils.FillCoordArrays(func, 1, (int) GetProperty("Grid points"));  
 var z = Utils.FillValues(func, (int) GetProperty("Grid points"));  
 Utils.FillZMatrix((int) GetProperty("Grid points"),   
 (int) GetProperty("Grid points"), z, x, y, func);  
 double[] min, max, center;  
 Utils.GetInfo(x, y, z, out min, out max, out center);  
 Utils.AdjustTo01(x, y, z, (int) GetProperty("Grid points"));  
 \_gridPoints = (int) GetProperty("Grid points");  
 \_x = x;  
 \_z = y;  
 \_y = z;  
  
 for (var xCell = 0; xCell < \_gridPoints - 1; xCell++)  
 for (var yCell = 0; yCell < \_gridPoints - 1; yCell++)  
 \_polygons.Add(new GridSquare(\_x, \_z, \_y, xCell, yCell));  
}

Для построения адаптивной сетки после начального разбиения происходит деление клеток на основании значения модуля градиента функции в узлах клетки. Если значение градиента в одном из четырех узлов и в центре ячейки превышает среднее значение по всем узлам сетки, то клетка делится на 4 клетки меньшего размера.

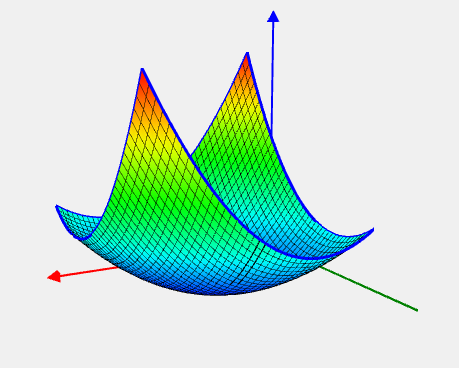
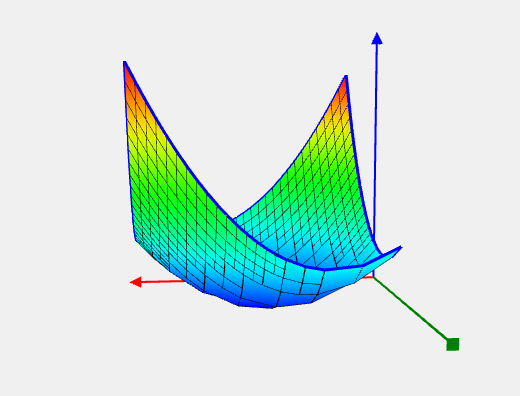


Рис. 8: Сравнение адаптивной сетки и традиционной – слева адаптивная сетка, справа традиционная.

Градиент в точке считается численно при помощи разностной схемы, где производная по одной переменной считается по формуле:

Процесс рисования графика после построения квадратов сетки:

1. меняется направление оси ординат;
2. рисуются координатные векторы;
3. выбирается адаптивное разбиение или обычное, в зависимости от значения параметра;
4. для каждого квадрата рисуется поверхность - рисуется два треугольника на узлах квадрата;
5. рисуется каркас графика.

Листинг 19. Рисование поверхности.

private void DrawSurface()  
{  
 GL.PushMatrix();  
 GL.Scale(1, 1, -1);  
 var source = \_polygons;  
 if ((bool) GetProperty("Axis")) DrawAxis();  
 if ((bool) GetProperty("Adaptive grid"))  
 source = \_adaptivePolygons;  
 foreach (var polygon in source)  
 {  
 if ((bool) GetProperty("Surface material")) DrawSurfaceMaterial(polygon);  
 if ((bool) GetProperty("Mesh")) DrawBorder(polygon);  
 }  
 GL.PopMatrix();  
}

Пользователь может настраивать вид графика - убирать поверхность или каркас, настраивать цвет каркаса, ставить нужное количество узлов сетки и способ построения сетки.

Так как все остальные классы для отображения функции построены по сходному принципу, рассмотрим структуру класса Surface более подробно.

Члены класса Surface:

* Поля
  + private readonly IPainter \_p - объект для рисования графических примитивов
  + private readonly List \_adaptivePolygons - сформированные полигоны для отображения функции в режиме адаптивной сетки
  + private readonly List \_polygons - полигоны для отображения функции в обычном режиме
  + private double[] \_center - положение центра поверхности, заданной функцией
  + private double[] \_max - максимальные значения координат поверхности
  + private double[] \_mins - минимальные значения координат поверхности
  + private bool \_isGridSet - флаг, показывающий, заполнены ли полигоны для рисования
  + private readonly PropertyProvider \_property - объек для работы со свойствами
  + private IFunction \_f - отображаемая функция.
* Свойства
  + public string Name - имя отображения.
* Конструктор
  + public Surface(IPainter p) - инициализирует новый экземпляр класса Surface
* Методы
  + public void Draw() - рисует поверхность на экране
  + public void SetProperty(string name, object value) - устанавливает значение свойства
    - name - имя свойства
    - value - значение
  + public void SetProperties(Dictionary propsValues) - устанавливает значение нескольких свойств одновременно
    - propsValues - словарь с именами свойств и их значениями
  + public object GetProperty(string name) - возвращает значение свойства
    - name - имя свойства
  + public List GetPropertiesInfo() - получение информации о доступных свойствах
  + public void RegisterProperty(PropertyInfo p) - добавление нового свойства
    - p - информация о свойстве
  + public void SetSurface(IFunction func) - установка функции для отрисовки
    - func - отображаемая функция
  + private void FillPolygons() - заполнение полигонов, используемых в обычном режиме рисования
  + private void FillAdaptivePolygons() - заполнение полигонов для адаптивного режима
  + private void Spleet(GridSquare gridSquare, Dictionary> tmp) - разделение квадрата сетки для адаптивного режима
    - gridSquare - разделяемый квадарт
    - tmp - коллекция для хранения исходного квадрата и полученных при разделении
  + private void DrawSurface() - рисование составных частей поверхности
  + private void DrawBorder(GridSquare polygon) - рисование границы полигона
    - polygon - текущий квадрат сетки
  + private void DrawSurfaceMaterial(GridSquare polygon) - рисование цветного материала поверхности
    - polygon - текущий квадрат сетки
  + private void DrawAxis() - отображение координатных осей

**TemperatureMap - карта температур.**

Построение карты температур происходит практические так же, как и поверхности. Отличием является координата Z - в плоском графике она учитывается только при выборе цвета. Третьей координатой вершин ставится 0. Не поддерживается адаптивная сетка. В остальном аналогична классу Surface.

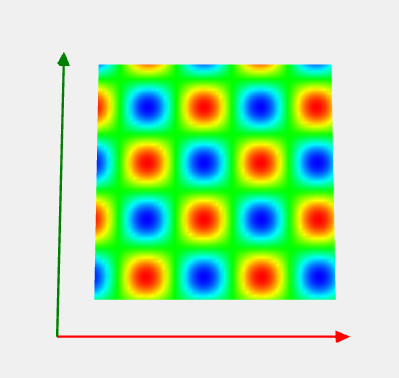


Рис. 9: Карта температур

**Curve - одномерный график**

Для расчета значений используется метод интерфейса IFunction, позволяющий вычислить значение функции через развертку Пеано. Построение происходит по тому же принципу, как в двух предыдущих отображениях. Вместо рисования полигонов рисуются линии между подсчитанными точками.

Листинг 20. Рисование кривой в классе Curve.

private void DrawCurve()  
{  
 if ((bool) GetProperty("Axis")) DrawAxis();  
 GL.LineWidth(1);  
 GL.Begin(BeginMode.LineStrip);  
 foreach (var point in \_points)  
 {  
 GL.Color3(Color.Black);  
 GL.Vertex3(point);  
 }  
 GL.End();  
}

Публичные методы совпадают с методами для класса Surface.

**Isobars - линии уровня**.

Алгоритм для построения линий уровня выбран на основе имеющихся данных о значениях функции. Существуют способы нахождения линий уровня для произвольного набора точек [17], но в данном конкретном случае возможен подсчет значений функции во всех точках ее области определения. Это позволило разбить область определения на равные части и применить метод марширующих квадратов [22]. После построения сетки происходит выявление изолиний при помощи вышеуказанного алгоритма. Одному уровню значений функции соответствует одна итерация этого процесса, он повторяется для всех значений линий уровня. В остальном структура класса не отличается от предыдущих.

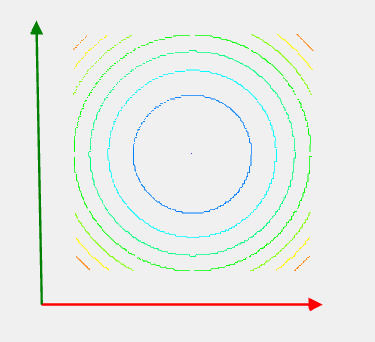


Рис. 10: Изолинии

**PointsSet3D и PointsSet2D - множества точек в пространстве и на плоскости**.

Два класса со схожей функциональностью - первый отображает точки в трехмерном пространстве, второй проецирует на плоскость z = 0. Так как потребуется отображать разные группы точек, оба множества поддерживают систему корзин. Точки из разных корзин отображаются разными цветами.

## Пользовательские элементы управления для инициализации OpenGL

Для связи основной программы на Windows Forms, и разработанных отображений использован специальный пользовательский элемент управления, предлагаемый библиотекой OpenTK. Но сам по себе он выполняет исключительно инициализацию openGL контекста. Требовалось с его помощью создать новый элемент управления, с поворотом рисуемых объектов при помощи клавиатуры и настройкой индивидуальных свойств. Причем свойства должны подгружаться из свойств объектов рисования автоматически. Новый user control получил название DrawablesControl. Визуально он включает в себя GLControl, контекстное меню для настройки параметров и кнопку для открытия меню (рис. 9).

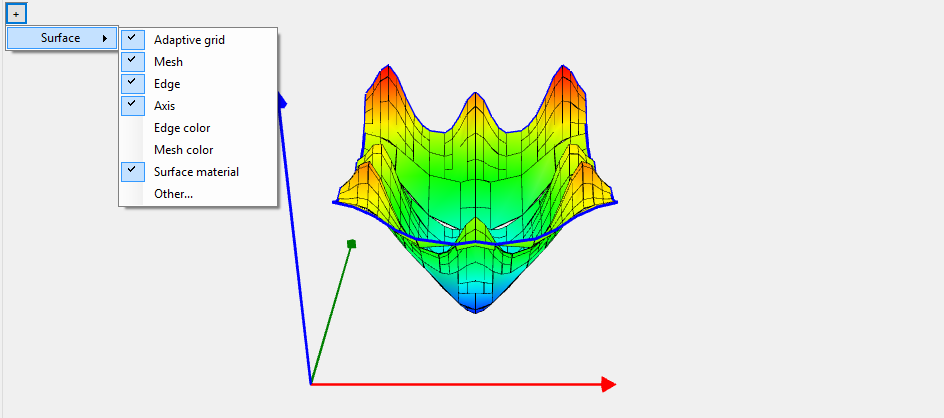


Рис. 11: Внешний вид DrawablesControl

При проектировании этого элемента управления ставились две основные задачи:

1. автоматическое формирование меню для изменения свойств объектов, рисуемых в этом окне
2. управление камерой и процессом обновления

Решение первой задачи было достаточно простым - так как все графические элементы с изменяемыми свойствами реализуют интерфейс IPropertyUser, достаточно было при добавлении нового элемента Drawables проверить его на реализацию этого интерфейса.

Листинг 21. Добавление объекта для рисования.

protected void AddDrawableObject(IDrawable drawableObject)  
{  
 // добавление объекта в список отображаемых  
 Drawables.Add(drawableObject);  
 // проверка на использование объектом настраиваемых параметров  
 var propsUser = drawableObject as IPropertyUser;  
 if (propsUser == null) return;  
 // добавление в список пользователей свойств  
 \_propertyUsers.Add(propsUser);  
 // если существует элемент с таким же именем, заменить на новый  
 if (\_propertyUsers.Find(p => p.Name == propsUser.Name) != null) return;  
 // создание пункта меню  
 var newObjectMenuItem = new ToolStripMenuItem(drawableObject.Name);  
 PropetiesMenuStrip.Items.Add(newObjectMenuItem);  
 // генерация пунктов меню для свойств   
 foreach (var p in propsUser.GetPropertiesInfo())  
 {  
 var menuItemFromProperty = CreateMenuItemFromProperty(p, propsUser);  
 if(menuItemFromProperty != null)  
 newObjectMenuItem.DropDownItems.Add(menuItemFromProperty);  
 }  
 // пункт меню для настройки числовых свойств, если такие имелись  
 if (!\_needToCreateChangeParamsDialog) return;  
 var paramsItem = new ToolStripMenuItem("Other...");  
 paramsItem.Click += OnClick;  
 newObjectMenuItem.DropDownItems.Add(paramsItem);  
 \_needToCreateChangeParamsDialog = false;  
}

Для разных типов свойств генерируются разные элементы управления - для свойств типа BoolProperty создаются пункты меню с возможностью поставить/снять галочку, для ColorProperty создаются окна ColorPicker для выбора нового цвета. Все числовые свойства изменяются в специально созданном для этого окне, доступного по пункту "Other..." в меню. За определения типа свойства и создания соответствующих ему пунктов меню отвечает метод CreateMenuItemFromProperty.

Листинг 22. Определение типа свойства и создание пункта меню для него.

private ToolStripMenuItem CreateMenuItemFromProperty(PropertyInfo p,

IPropertyUser propsUser)  
 {  
 ToolStripMenuItem result = null;  
 if (p is BoolProperty)  
 {  
 result = new ToolStripMenuItem(p.name)  
 {  
 Checked = (bool) propsUser.GetProperty(p.name),  
 CheckOnClick = true  
 };  
 result.CheckedChanged += BoolPropertyOnCheckedChanged;  
 }  
 else if (p is ColorProperty)  
 {  
 result = new ToolStripMenuItem(p.name);  
 result.Click += ColorPropertyOnClick;  
 }  
 else if (p is DoubleProperty)  
 {  
 \_needToCreateChangeParamsDialog = true;  
 \_parameters.Add(p, (double) propsUser.GetProperty(p.name));  
 }  
 else if (p is IntProperty)  
 {  
 \_needToCreateChangeParamsDialog = true;  
 \_parameters.Add(p, (int) propsUser.GetProperty(p.name));  
 }  
 return result;  
 }

Также наследники класса DrawablesControl могут добавлять параметры окна, как это реализовано, например, в окне с 2d отображениями функций. Там пользователю предлагается выбрать текущий способ рисования - как карта температур или как линии уровня. Вторым логическим блоком во внутреннем коде окна для рисования становится организация обработки нажатий клавиатуры и перерисовки элементов. В первом варианте реализации вся обработка происходила при помощи обработчиков событий IsKeyDown и IsKeyUp, предоставляемых классом UserControl. При нажатии клавиш клавиатуры в специальном классе Keyboard ставили флаги нажатых клавиш. При обновлении, которая происходило в обработчике события Paint у GLControl, Keyboard меняла параметры камеры. При этом угол поворота камеры при нажатии всегда оставался постоянным. Обновление изображения происходило при помощи стандартного таймера из набора элементов Windows Forms. Во время события Timer.Tick вызывался метод GLControl.Refresh(). Такой способ обновления был использован несколько раз в простых приложениях с рисованием.

После добавления нескольких отображений функций в основную программу, стали очевидны недостатки такого подхода:

1. из-за постоянного вызова обновления по таймеру использование CPU программой стояло на 100%;
2. на компьютерах с малой частотой CPU поворот графиков функции становился очень медленным;
3. при повышении количества отображений те объекты, которые должны были вращаться синхронизировано, отставали друг от друга.

Новый подход используется в компьютерных играх. Вместо постоянного угла поворота камеры, постоянной является скорость поворота - угол в секунду. Время последнего обновления сохраняется, и камера поворачивается на угол, равный прошедшее время \* скорость поворота.

## Внедрение отображений в основную программу

После реализации объектов Drawables и элемента управления для рисования, внедрение не занимает особого труда.

Создаются наследники классов, реализующих интерфейс Drawables. Кроме этого, добавляется интерфейс нужного View для получения данных от Presenter требуемого вида. Например, для рисования основной функции в виде поверхности создается наследник класса Surface с интерфейсом IMainFunctionView.

В реализации интерфейса View полученные из модели данные обрабатываются. Это необходимо для приведения типов модели к типам конкретных фигур. Именно так происходит в классе PointsOnPlot (наследнике PointsSet3d) - в нем после получения новой MethodPoint происходит ее преобразование в Vector3d с масштабированием.

Следующим этапом становится создание наследника DrawablesControlThreading, имеющего конкретный набор объектов для рисования. Сейчас есть три таких элемента - график функции с точками испытаний, карта температур и линии уровня с точками испытаний, диаграмма распределения точек испытаний. Последним необходимым действием будет добавление соответствующих Presenters для связи с моделью.

После смены подхода к обновлению возникла проблема организации одновременного вращения - обработка нажатий клавиш каждым объектом DrawablesControlThreading приводила к рассинхронизации. Для решения этой проблемы в DrawablesControlThreading был убран захват нажатий клавиатуры. Теперь форма эксперимента перехватывает нажатия клавиш и уведомляет элементы управления о необходимости вращения на требуемый угол.

# Заключение

В ходе выполнения работы разработана архитектура приложения Абсолют. Разработанная архитектура позволяет решить задачи добавления различных алгоритмов глобальной оптимизации и целевых функций в систему и расширения системы путем разработки графических отображений. Разработка архитектуры проводилась с использованием технологий объектно-ориентированного программирования, компонентного подхода, послойного подхода и с применением паттернов проектирования. Реализовано приложение с использованием разработанной архитектуры.

Кроме того, реализованы графические отображения целевых функций и процесса поиска решения, среди которых график поверхности, карта температур и линии уровня. Все отображения успешно внедрены в систему и подготовлена база для расширения набора доступных визуальных элементов.

В приложение интегрирован запуск сторонних систем Examin и Direct и загрузка функций в виде подключаемых DLL файлов.

Вся разработка производилась в среде MS Visual Studio на языке С# с использованием библиотек .NET и библиотек для работы с OpenGL.

# Список литературы

1. Стронгин, Р.Г. Численные методы в многоэкстремальных задачах (информационно-статистические алгоритмы) / Р.Г. Стронгин. – М: Издательство «Наука», 1978 г. – 240 с.
2. Васильев, Ф.П. Методы оптимизации / Ф. П. Васильев – М.: Издательство «Факториал Пресс», 2002. – 824 с.
3. Параллельные вычисления в задачах глобальной оптимизации: монография / Р.Г. Стронгин [и др.] – М.: Издательство Московского университета, 2013. – 280 с.
4. Гергель В.П. АБСОЛЮТ. Программная система для исследования и изучения методов глобальной оптимизации : учебное пособие. / В.П. Гергель, Р.Г. Стронгин. – Нижний Новгород: Изд-во Нижегородского университета, 1998. – 141 с.
5. Гладков. Л.А. Генетические алгоритмы: Учебное пособие. / Л. А. Гладков, В.В. Курейчик, В.М. Курейчик— 2-е изд. — М: Физматлит, 2006. — 400 с.
6. Соммервилл, Иан. Инженерия программного обеспечения / Иан Соммервилл – 6-е издание – М.: Издательский дом «Вильямс», 2002. – 624 с.
7. Паттерны объектно-ориентированного программирования. Паттерны проектирования. / Эрих Гамма [и др.] – СПб.: Питер, 2016 – 366 с.
8. Никулин, Е.А. Компьютерная геометрия и алгоритмы машинной графики / Е.А. Никулин – СПб.: БХВ-Петербург, 2003 – 560 с.
9. Ву, Мейсон. OpenGL. Руководство по программированию. / Мейсон Ву [и др.] – СПб:Питер, 2006
10. Momin Jamil. A literature survey of benchmark functions for global optimization problems / Momin Jamil, Xin-She Yang // Int. Journal of Mathematical Modelling and Numerical Optimisation – 2013 - Vol. 4, No. 2 - pp. 150-194
11. Сысоев, А.В. Программный комплекс параллельных вычислений в задачах выбора глобально-оптимальных решений / А.В. Сысоев // Вестник Нижегородского университета им. Н.И. Лобачевского – 2012 - №5 [2] – c. 425-431
12. Пиявский С. А. Один алгоритм отыскания абсолютного экстремума функций / С.А. Пиявский // Журнал вычислительной математики и математической физики – 1972- т.12, № 4 - стр. 885—896.
13. Тепляков, С. Паттерны проектирования на платформе .NET / С. Тепляков. - СПб.: Питер, 2015. - 320 с.
14. Лабутина, А.А. ПараЛаб – среда для проведения вычислительных экспериментов для изучения и исследования параллельных методов решения сложных вычислительных задач / А. А. Лабутина, В.П. Гергель // Вестник Нижегородского университета им. Н.И. Лобачевского – 2011 - №3 [2] – с. 239-247
15. OpenGL Programming [Электронный ресурс]. – Режим доступа: <https://en.wikibooks.org/wiki/OpenGL_Programming> - Заглавие с экрана. – (Дата обращения – 15.05.2017).
16. Global optimization test problems [Электронный ресурс]. – Режим доступа: <http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm> – Заглавие с экрана – (Дата обращения – 15.05.2017).
17. Creating Contour Graphs [Электронный ресурс]. – Режим доступа: <http://www.originlab.com/doc/Origin-Help/Create-Contour-Graph> – Заглавие с экрана – (Дата обращения – 15.05.2017).
18. Functions 3D: Examples [Электронный ресурс]. – Режим доступа: <https://www.benjoffe.com/code/tools/functions3d/examples> – Заглавие с экрана – (Дата обращения – 15.05.2017).
19. Dislin Home Page [Электронный ресурс]. – Режим доступа: <http://www.mps.mpg.de/dislin> – Заглавие с экрана – (Дата обращения – 15.05.2017).
20. The Open Toolkit [Электронный ресурс]. – Режим доступа: <https://opentk.github.io/> – Заглавие с экрана – (Дата обращения – 15.05.2017).
21. Работа с OpenGL [Электронный ресурс]. – Режим доступа: <http://pmg.org.ru/nehe/index.html> – Заглавие с экрана – (Дата обращения – 15.05.2017).
22. Marching squares [Электронный ресурс]. – Режим доступа: <https://en.wikipedia.org/wiki/Marching_squares> – Заглавие с экрана – (Дата обращения – 15.05.2017).
23. Microsoft Application Architecture Guide, 2nd Edition [Электронный ресурс]. – Режим доступа: <https://msdn.microsoft.com/en-us/library/ff650706.aspx> – Заглавие с экрана – (Дата обращения – 15.05.2017).

1. Barry Lapthorn, A Simple C# Genetic Algorithm. [Electronic resource]. – URL: https://www.codeproject.com/Articles/3172/A-Simple-C-Genetic-Algorithm. [↑](#footnote-ref-1)