Министерство образования Российской Федерации
Сибирский Государственный Аэрокосмический университет
имени академика М.Ф. Решетнёва
Институт информатики и телекоммуникаций
Кафедра информатики и вычислительной техники

КУРСОВОЙ ПРОЕКТ

По курсу «Объектно-ориентированное программирование»

Пояснительная записка

Тема: Обучающая программа по дисциплине «Методы оптимизации» «МОСТ»

Выполнил: студенты гр. БИ72
Проскурин А.В.
Почкаенко В.Ю.
Белоконь А.В.
Проверил: к.т.н., доцент кафедры ИВТ
Вдовенко В.В.

Красноярск, 2010

Содержание

Введение

Обучающая программа по дисциплине «Методы оптимизации» «МОСТ» разрабатывается по заказу Л.И. Лыткиной, доцента кафедры Прикладной математики Сибирского Государственного Аэрокосмического университета им. академика М.Ф. Решетнёва. Основной целью программы является проверка в автоматизированном режиме знаний студентов об основных методах оптимизации:

- методе покоординатного спуска с дискретным шагом;
- методе покоординатного спуска с минимизацией по направлению;
- методе наискорейшего спуска;
- методе Хука-Дживса;
- методе Нелдера-Мида;
- методе Ньютона,

путём проверки состояний приложения.

Следует отметить, что подобные приложения уже существуют, однако они имеют ряд недостатков (работоспособны только под ОС Windows XP, не безопасны,имеют неудобный интерфейс, отсутствует дополнительный функционал). Таким образом, программа «МОСТ» должна вобрать все достоинства уже существующих приложений и устранить их недостатки.

В ходе работы над программой предполагается глубже изучить кросплатформенную библиотеку QT в целях реализации простой переносимости системы под платформы, отличные от Windows. Также предполагается получить и закрепить навыки по работе с UML диаграммами и их созданию, а также приложениями для реализации этих процессов.

1. Техническое задание

1.1.Введение

1.1.1. Назначение и общая характеристика продукта

ПП предназначен для проверки в автоматизированном режиме знаний студентов об основных методах оптимизации на основе прохождения ими двух типов функций: квадратичной $(A \bullet (x_1-B)^2+C \bullet (x_2-D)^2+E \bullet (x_1-F) \bullet (x_2-G))$ и овражной $(A \bullet (x_2-x_1^2)^2+B \bullet (1-x_1)^2)$.

1.1.2. Ссылки и использование литературы

Список источников, к которым можно обратиться за справками в процессе разработки

- http://doc.crossplatform.ru/qt/
- http://ru.wikipedia.org/

Список, использованной литературы.

- Qt4.5. Профессиональное программирование на C++. – Спб.: БХВ-Петербург, 2010. – 896 с.: ил. + DVD – (В подлиннике) ISBN 978-5-9775-0398-3
- Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб: Питер, 2001. — 368 с.: ил. (Серия ≪Библиотека программиста≫) ISBN 5-272-00355-1
- Васильев Ф.П., Методы оптимизации. – М: Факториал Пресс, 2002. – 824 с.: ил. ISBN: 5-88688-056-9

1.1.Описание пользователей

Виды пользователей и их краткие описания

ПП предполагает наличие двух видов пользователей:

- Студент – объект проверки. Получает у преподавателя вариант задания и пытается его выполнить с минимальным количеством ошибок. Студенту доступны все функции основной программы кроме разблокирования овражной функции по паролю;
- Преподаватель – при помощи дополнительной программы устанавливает данные вариантов и некоторые другие настройки (задаёт максимально допустимое число ошибок, пароль для разблокировки функции). В основной программе может разблокировать овражную функцию с помощью пароля.

Среда пользователя

Поддерживаемые ОС: семейства Windows и Linux.

1.2.Состояние рынка и конкурирующие продукты

Характеристика рынка

Данный ПП является узкоспециализированной программой, предназначенной для использования заказчиком, в связи с чем были рассмотрены только подобные программы, уже имеющиеся у заказчика – две версии программы «Методы оптимизации 2008».

Конкурирующие программные продукты

Основными конкурирующими продуктами являются:

- «Методы оптимизации 2008» 1.0:

“+” – масштабируемая при помощи кнопок карта высот функции с отображением дискретных уровней;

“–“ – отсутствует возможность выбора готовых вариантов;

“–” – отсутствует возможность разблокирования овражной функции по паролю;

“–” – неудобный интерфейс (слишком много окон, они имеют фиксированный размер);

“–” – справка имеется не ко всем частям программы.

- «Методы оптимизации 2008» 1.1:

“+” – масштабируемая при помощи кнопок карта высот функции с отображением дискретных уровней;

“–” – отсутствует возможность разблокирования овражной функции по паролю;

“–” – неудобный интерфейс (слишком много окон, они имеют фиксированный размер);

“–” – справка имеется не ко всем частям программы.

Определение позиций продукта на рынке

Продукт не претендует на высокие позиции на рынке. Предполагается его внутривузовское использование заказчиком.

1.3.Функции продукта

Пользовательские требования

- Возможность выбора метода оптимизации;
- Возможность выбора типа функции;
- Возможность выбора готового варианта, либо ввода своего;
- Наличие справки ко всем частям ПП;

- Возможность более удобной тренировки на квадратичной функции (начать заново на открытом окне метода (без повторного открытия));
- Разблокирование овражной функции при определённом минимуме допущенных ошибок при прохождении квадратичной функции;
- Отображение в конце прохождения функций числа допущенных ошибок (при прохождении овражной функции, отображать количество ошибок в квадратичной);
- Упрощённый ввод производных в методах наискорейшего спуска и Ньютона (убрать ряд ограничений);
- Прорисовка на карте высот функции следа, пройденного алгоритмом;
- Корректное отображение записи функции (когда часть коэффициентов равна 0 или 1);

Дополнительные функции

- Для преподавателя возможность разблокирования овражной функции при помощи пароля;
- Для преподавателя возможность в удобной форме задавать данные вариантов и др. (реализовать отдельным приложением).

1.4. Нефункциональные требования

1.4.1. Требования к продукту

Требования к инсталляции

Для установки требуется скопировать все файлы программы на локальный ПК (возможно отсутствие файла setvariants.exe (необходим только для преподавателя)).

Требования к удобству эксплуатации

- Как можно меньше окон (объединение функций выбора в одно окно);
- Интуитивно понятный интерфейс;
- Как можно больше пространства под карту высот функции;
- Возможность менять шрифты окон.

1.4.2. Требования к эффективности

Время ответа для транзакций (сред. и макс.)

Среднее и максимальное времена ответа для транзакций: зависит от мощности ПК.

Пропускная способность (транзакции в сек.)

Ограничивается скоростью человека.

Емкость

Предусмотрена работа с одним пользователем.

Время обновления экрана

Зависит от установленной частоты обновления монитора (по умолчанию 1/60 секунды).

Время реакции на действие пользователя

50 мс (ограничивается работой системного таймера).

Требования к ресурсам

Для работы данной программы достаточно компьютера на базе процессора Intel Pentium 3.

Среднее время между отказами (частота сбоев)

Не анализировалось.

Среднее время восстановления после сбоя

Автоматически при перезапуске.

Вероятность порчи данных при сбое

Промежуточные данные (выбранный вариант, степень прохождения функций и т.д.) не сохраняются.

Поддерживаемые ОС

Семейства Windows и Linux.

Объем машинно-зависимых операторов и подсистем

Отсутствует.

Прочие требования к переносимости

Возможна работа под другими ОС, поддерживаемыми библиотекой Qt (необходима компиляция исходного кода без внесения каких-либо изменений под необходимую ОС).

Требования к защищенности

Пароль для разблокирования хранится в бинарном файле в шифрованном виде.

Требования к удобству сопровождения

Открытый код с комментариями.

1.4.3. Организационные требования

Сроки разработки и изготовления

4 месяца.

Сопутствующая документация

Предполагается в виде справок к ПП, комментариев в исходном коде и пояснительной записки.

Модель организации разработки

Эволюционная модель.

Методы проектирования и документирования разработки

Электронный вариант документации.

Языки программирования и инструментальные средства

Основной язык программирования – C++.

Для написания кода будет использоваться среда разработки QDevelop, поддерживающая кроссплатформенную библиотеку Qt.

Требования к лицензированию

Данный ПП распространяется по лицензии GNU GPL, версия 3.

Требования к распространению и вопросы цены

Продукт распространяется бесплатно.

Вопросы авторизации прав

Авторские права принадлежат разработчикам программы.

1.5. Требования к документации

Руководство пользователя

Предполагается в виде справки к ПП. Также будет описано в пояснительной записке.

Руководство программиста

Программа с открытым кодом, содержит комментарии. Также будет описано в пояснительной записке.

Интерактивная подсказка

Предполагается.

Руководство по инсталляции и конфигурированию

Для установки требуется скопировать все файлы программы на локальный ПК (возможно отсутствие файла setvariants.exe). Дополнительной настройки не требуется.

1.6. Глоссарий

ОС – Операционная система;

ПК – Персональный компьютер;

ПП – Программный продукт;

GNU GPL – GNU General Public License (открытое лицензионное соглашение GNU).

2. Состав рабочей группы

Состав рабочей группы и распределение обязанностей приведены в таблице 1.

Таблица 1 Состав рабочей группы

1.	Управляющий	Организует процесс разработки и	Почкаенко В.Ю.

проектом:	является главным администратором.	
2. Архитектор:	Главный технический специалист, который руководит процессом принятия технических решений.	Почкаенко В.Ю.
3. Системный аналитик:	Специалист, отвечающий за точное установление множества требований, предъявляемых к программному продукту, а также за создание технического проекта.	Проскурин А.В.
4. Ведущий программист:	Программист, отвечающий за проектирование программ и за кодирование самых сложных модулей.	Почкаенко В.Ю.
5. Дизайнер:	Программист, отвечающий за графический интерфейс программы.	Проскурин А.В. Почкаенко В.Ю.
6. Программист-кодировщик:	программист, выполняющий кодирование простых модулей под руководством ведущего программиста.	Проскурин А.В. Белоконь А.В.
7. Библиотекарь:	Ведет сбор и учет всей информации, касающейся процесса разработки программного продукта.	Проскурин А.В.
8. Ответственный за тестирование:	специалист (программист), составляющий тесты для проверки программ.	Белоконь А.В.
9. Ответственный за документацию:	Специалист, составляющий проектную (отчеты о выполнении этапов разработки) и рабочую документацию (руководства программиста и пользователя).	Проскурин А.В. Почкаенко В.Ю. Белоконь А.В.

3. План по качеству

Поскольку программа будет представлена в виде приложения, необходимо учесть следующие факторы:

1) Поскольку целевая аудитория программы – это студенты-программисты, то необходимо добиться того, чтобы система была максимально защищена от не запланированного вмешательства.

2) Необходимо предусмотреть систему перехода от одного типа функции к другому (при помощи прохождения квадратичной функции и с помощью пароля, доступного только преподавателю)

3) Необходимо наличие системы составления вариантов преподавателем (отдельного приложения, являющегося дополнением к основному).

4) Программа работает в режиме реального времени, поэтому необходимо обеспечить быструю обработку действий пользователя.

Таблица 2 План выполнения работ

Объединение в группу	10.09.2010
Разработка технического задания	10.09.2010 – 1.10.2010
Разработка технического проекта	1.10.2010 – 15.10.2010
Программирование основного кода	15.10.2010 – 17.11.2010
Тестирование готового программного продукта	17.11.2010 – 21.11.2010
Составление проектной документации	15.11.2010 – 2.12.2010
Защита проекта	18.12.2010

4. Руководство пользователя

Для запуска программы на компьютере пользователя требуется открыть файл optimizationmethods.exe.

Главное окно программы состоит из двух частей:

- Выбор метода оптимизации и типа функции (Рис. 1);
- Установка используемых значений (Ошибка: источник перекрестной ссылки не найден, Рис. 4Ошибка: источник перекрестной ссылки не найден).

4.1. Выбор метода оптимизации и типа функции

Метод оптимизации выбирается в выпадающем списке. Для каждого метода доступен выбор квадратичной функции. Для выбора овражной функции необходимо данным методом пройти квадратичную (либо разблокировать ее при помощи пароля в пункте меню «Разблокировать овражную функцию» (Рис. 2)).

Овражная функция снова блокируется сразу после открытия окна прохождения.

После выбора метода оптимизации и типа функции нажмите кнопку «Далее».

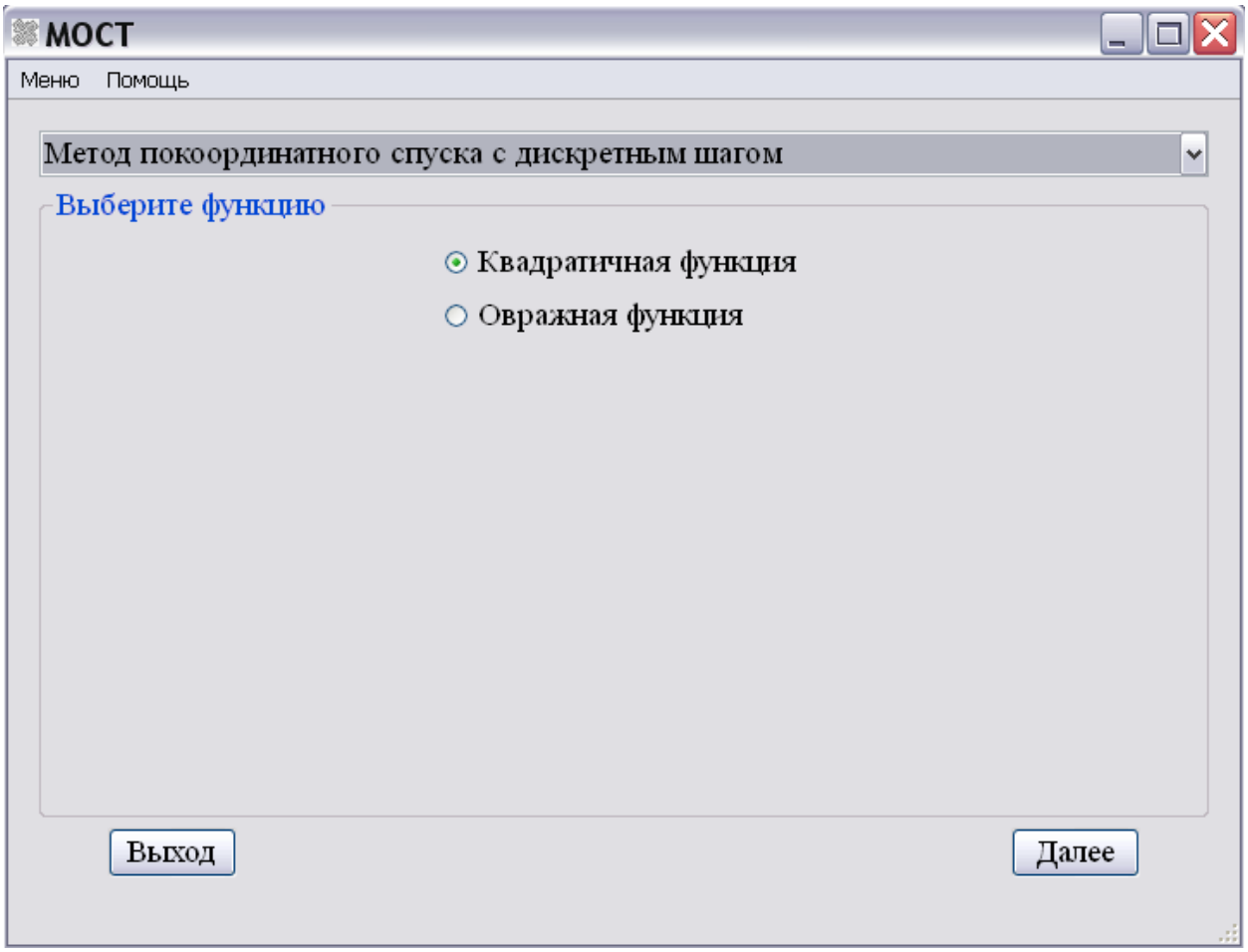

Рис. 1 Главное окно «МОСТ» – выбор метода оптимизации

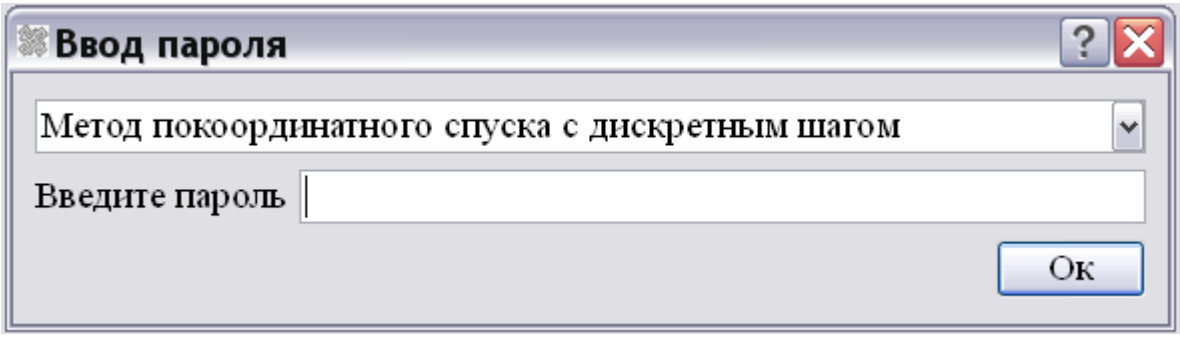

Рис. 2 Окно разблокирования овражной функции при помощи пароля

4.2. Установка используемых значений

Данная вкладка содержит выбор способа установки значений и их реализацию.

Есть два способа установки значений:

- Выбор варианта (Ошибка: источник перекрестной ссылки не найден);

- Ввод вручную (Рис. 4).

Для переключения между ними необходимо щёлкнуть левой кнопкой мыши необходимому пункту.

Выбор варианта

На форме имеется выпадающий список для выбора уже существующего варианта. Все необходимые данные уже введены и остается только выбрать вариант и нажать кнопку «Далее».

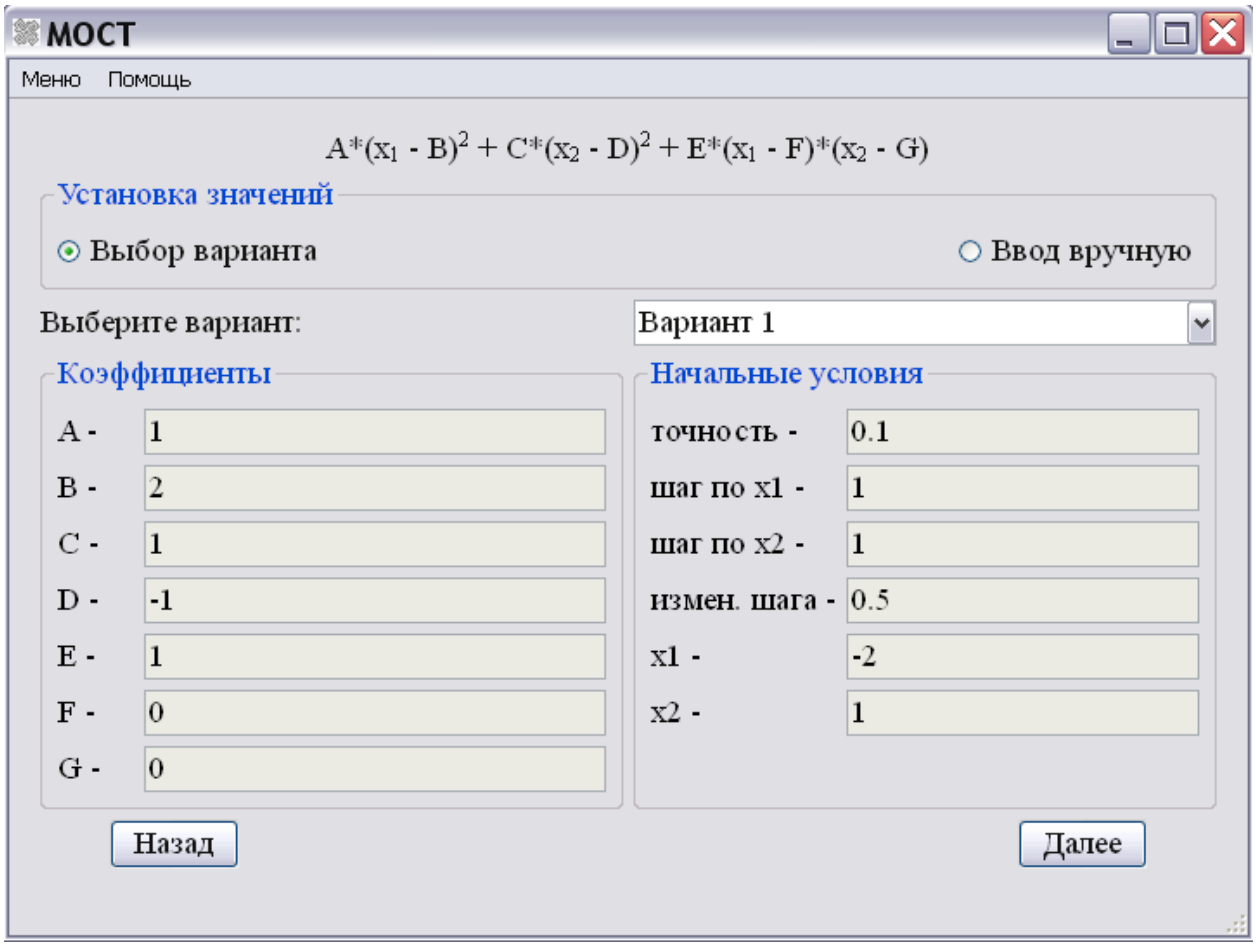

Рис. 3 Главное окно «МОСТ» – установка используемых значений (выбор варианта)

Ввод вручную

Вам необходимо вручную заполнить поля формы. Поля сгруппированы в две колонки:

- Коэффициенты – содержит поля, предназначенные для ввода коэффициентов функции (формула функции расположена в верхней части окна программы);

- Начальные условия – содержит поля задания точности, установки длин шагов и коэффициента изменения шага, а также задание начальной точки.

В поля можно вводить только числа (целые и десятичные). При вводе десятичных чисел в качестве разделителя можно использовать как символ «,», так и символ «.».

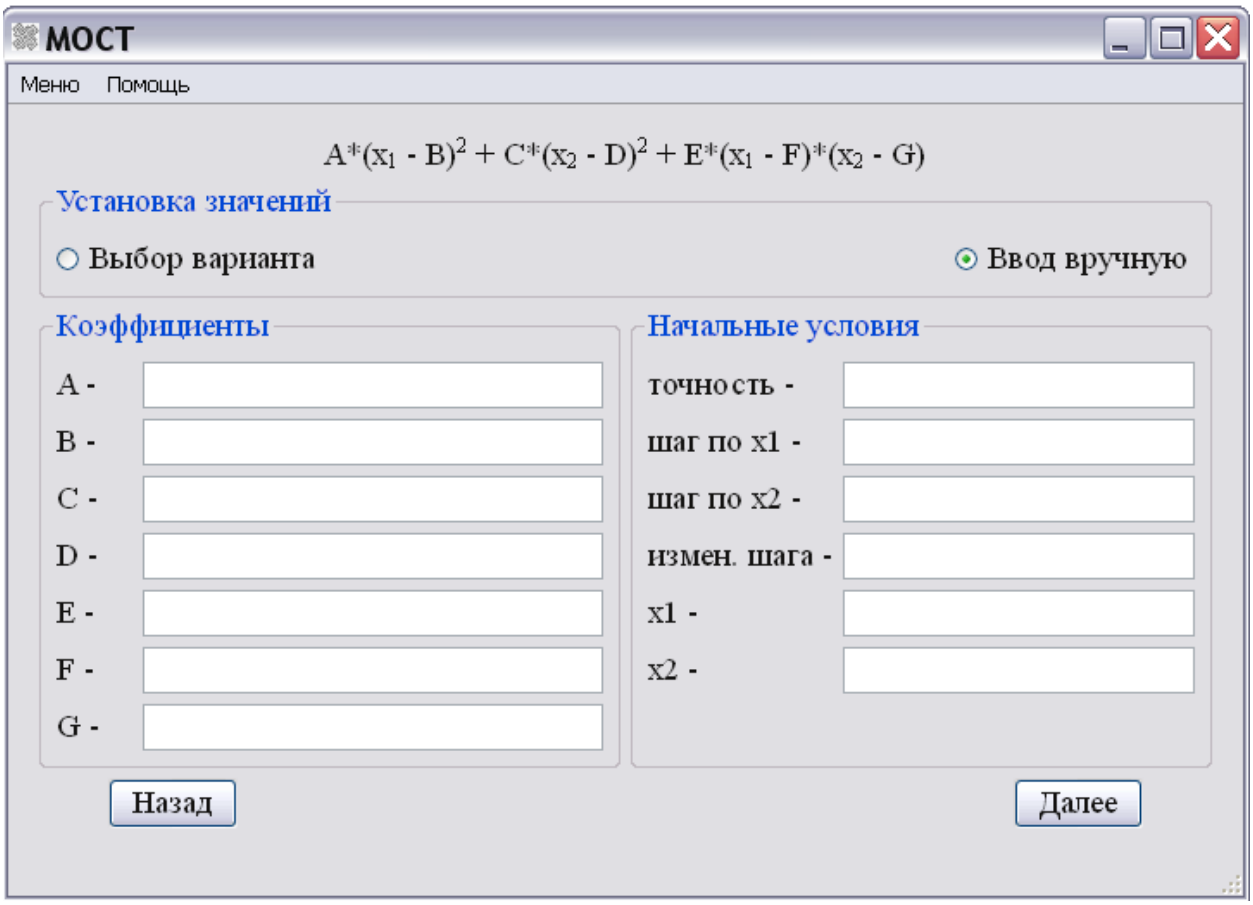

Рис. 4 Главное окно «МОСТ» – установка используемых значений (ввод вручную)

Когда будут заполнены все поля, нажмите кнопку «Далее».

4.3. Структура меню главного окна

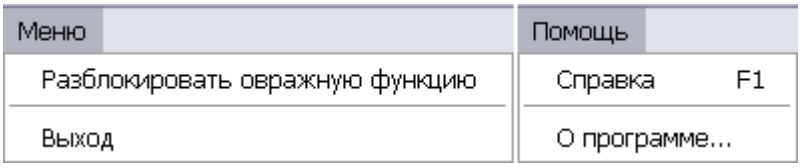

Рис. 5 Структура меню главного окна «МОСТ»

- Разблокировать овражную функцию – открытие окна вводы пароля для разблокирования овражной функции (пароль имеется у преподавателя) (Рис. 2);

- Выход – закрытие программы;
- Справка – открытие окна справки. Автоматически открывается справка о главном окне (чтобы просмотреть меню справки необходимо перейти по ссылке «На главную») (Рис. 6);
- О программе – содержит три вкладки: о программе (общие сведения), исходный код (ссылка на исходный код), авторы (имена разработчиков и их электронные ящики) (Рис. 7).

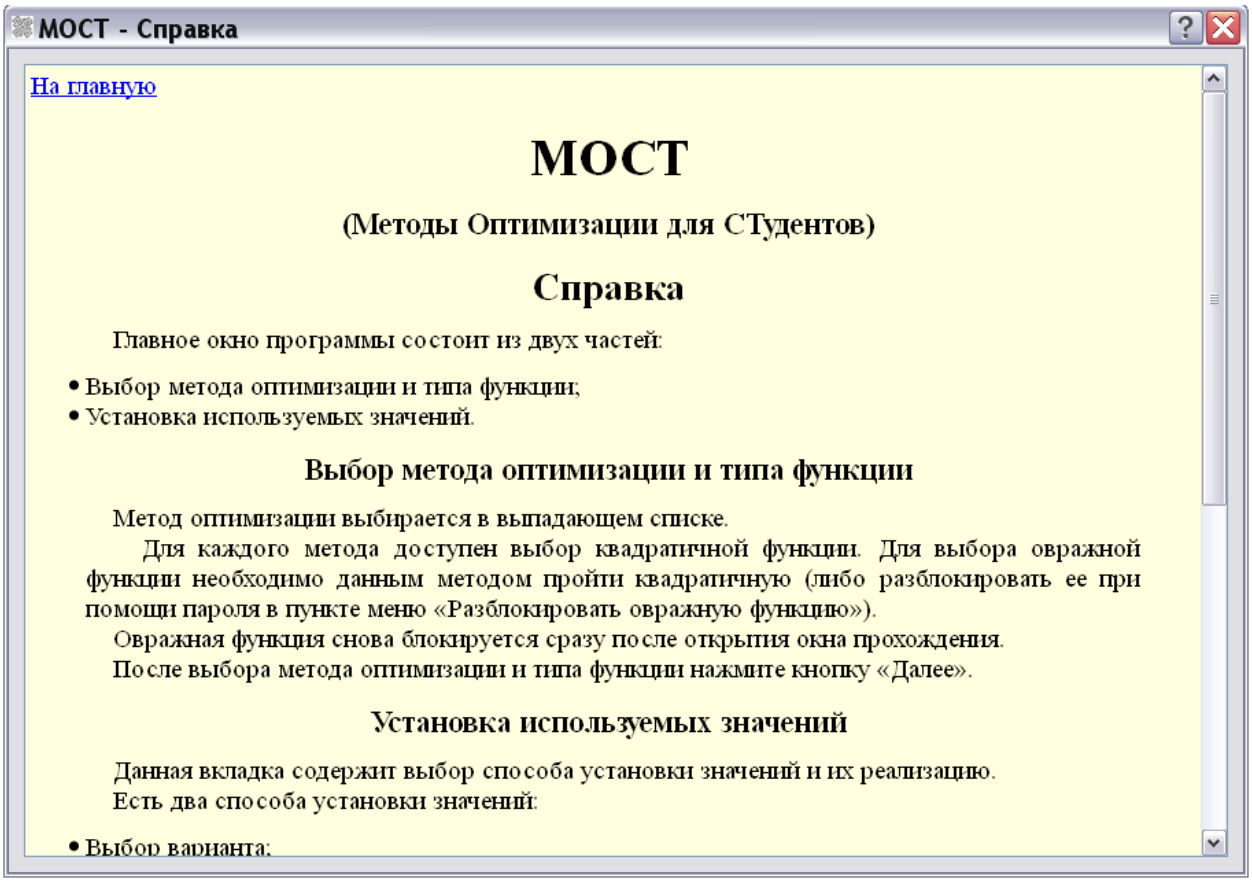

Рис. 6 Окно справки к главному окну «МОСТ»

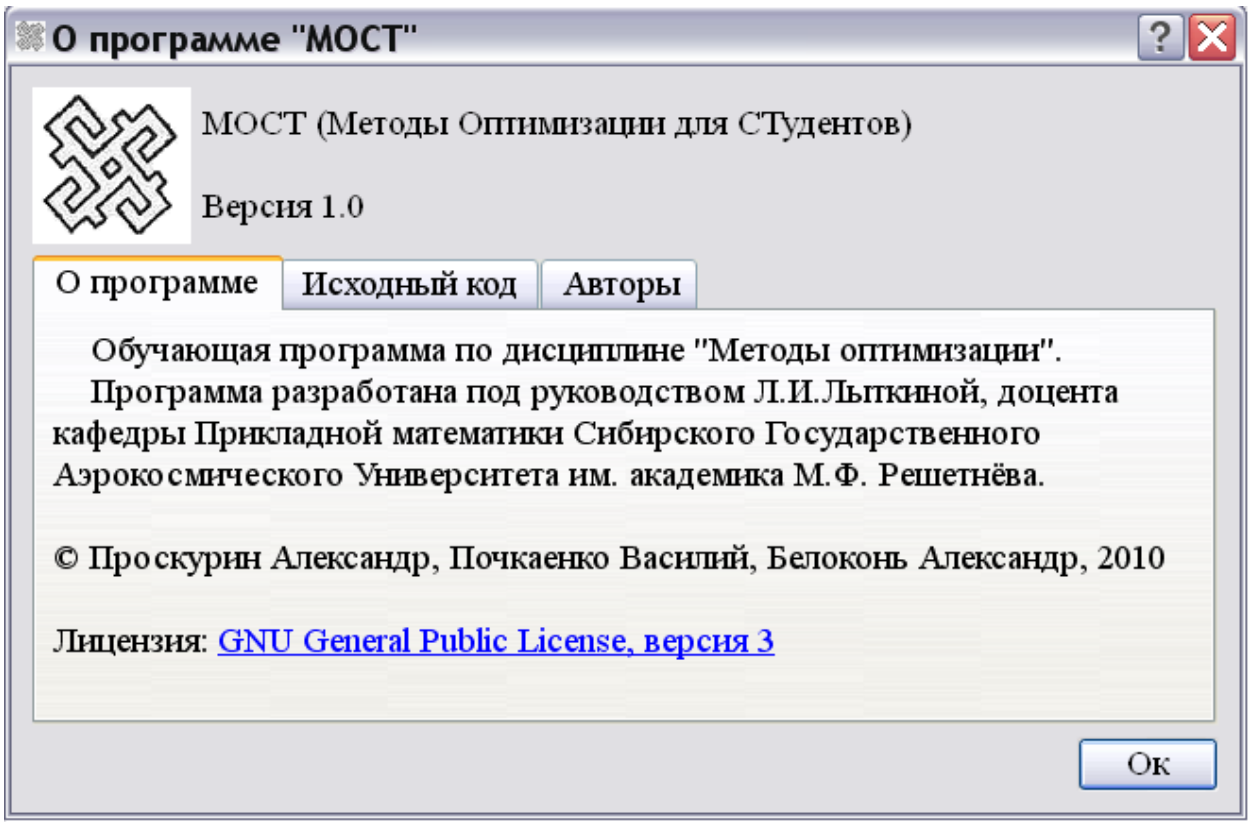

Рис. 7 Окно о программе «МОСТ»

4.4. Прохождение методов оптимизации

Окна методов оптимизации разбиты на 4 функциональные части (Рис. 8):

- Информация о выполнении – содержит информацию о текущем состоянии прохождения (координаты точек, значения функции в них и т.д.) и необходимую для дальнейшего прохождения алгоритма (например, расстояние между двумя последними принятыми точками);
- Отдел выбора – содержит кнопки, пункты выбора, а также строки для ввода данных. Необходимы для изменения состояния программы согласно алгоритму оптимизации (в противном случае регистрируется ошибка);
- Отдел логов – отображает действия пользователя и состояния программы;
- Карта высот – отображает карту высот функции и след, пройденный алгоритмом. Также содержит кнопки масштаба (можно менять при помощи мыши – для этого необходимо зажав клавишу Ctrl нажать левую кнопку мыши, затем перемещая мышь выбрать необходимую область масштабирования и зажав клавишу Ctrl вновь

нажать на левую кнопку мыши). Перемещаться по карте можно при помощи мыши (зажать левую кнопку и потянуть мышь в противоположную необходимой для просмотра сторону). Для сброса масштаба нажмите правую кнопку мыши.

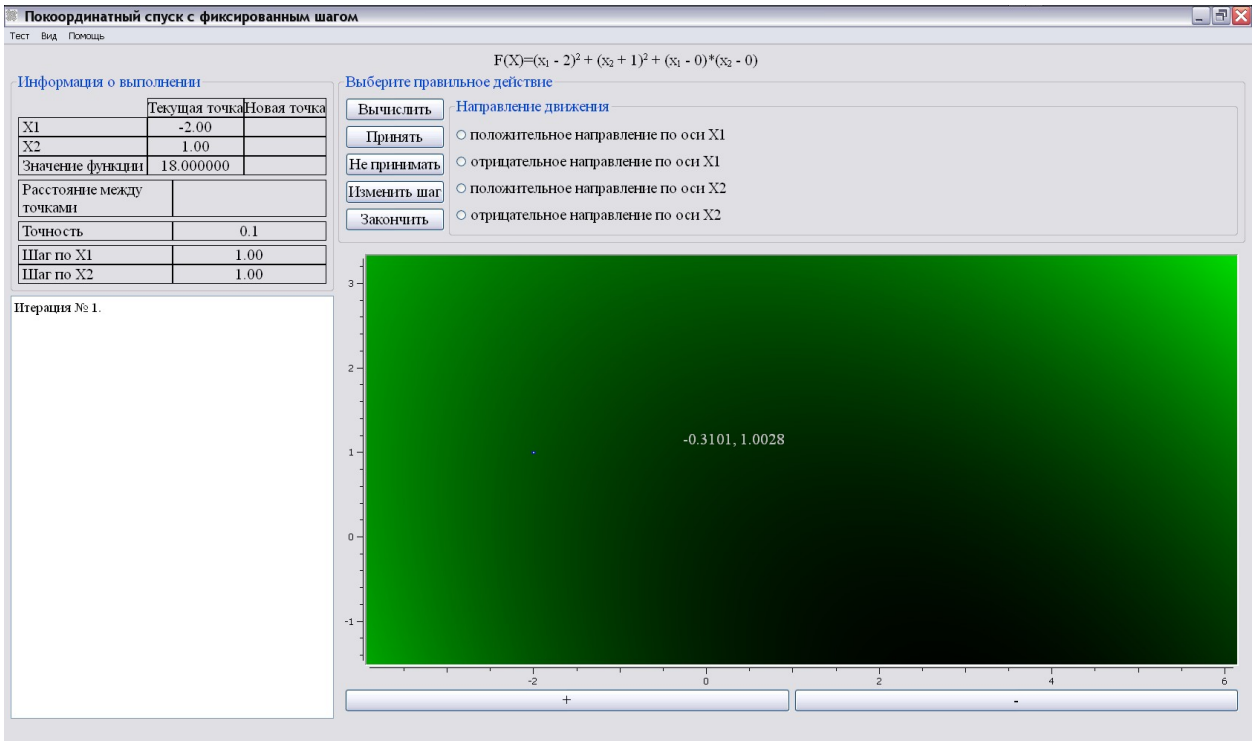

Рис. 8 Окно метода оптимизации «МОСТ»

Для прохождения методов необходимо с минимальным количеством ошибок изменять состояния программы согласно алгоритму (который можно просмотреть в пункте меню «Помощь -> Справка», либо нажав клавишу F1).

4.5. Структура меню окон методов оптимизации

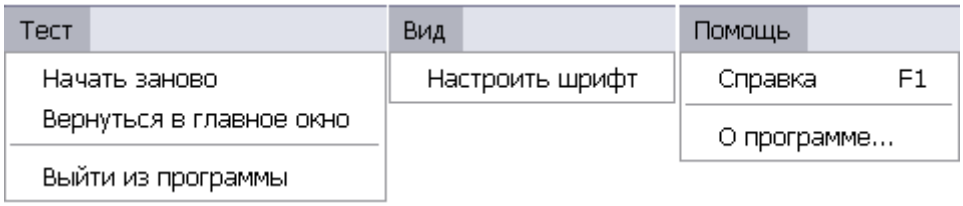

Рис. 9 Структура меню окон методов оптимизации

- Начать заново – сбросить текущее состояние программы на исходное (не активно при прохождении овражной функции);
- Вернуться в главное окно – закрыть окно метода оптимизации и открыть главное окно;
- Выйти из программы – закрытие программы;

- Настроить шрифт – открытие диалогового окна для настройки шрифтов окна (Рис. 10);
- Справка – открытие окна справки. Автоматически открывается справка по текущему методу оптимизации;
- О программе – аналогично соответствующему пункту меню главного окна (Рис. 7).

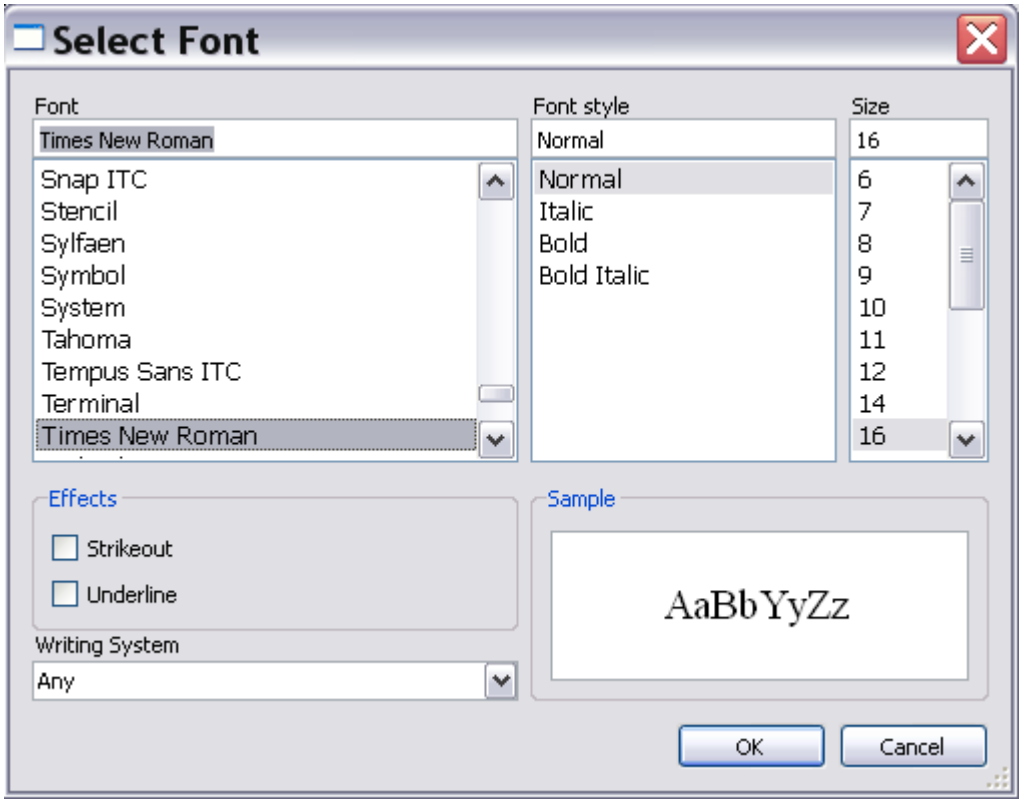

Рис. 10 Диалоговое окно настройки шрифтов в окнах МО

4.6. Приложение для установки вариантов

Главное окно программы состоит из четырех вкладок (Рис. 11):

- Квадратичная функция – предназначена для создания и редактирования вариантов квадратичной функции;
- Овражная функция – предназначена для создания и редактирования вариантов овражной функции;
- Количество ошибок – предназначена для установки максимально допустимого количества ошибок, которые пользователь может совершить, чтобы перейти от квадратичной функции к овражной;
- Установка пароля – предназначена для установки нового пароля, необходимого для открытия овражной функции через пункт меню.

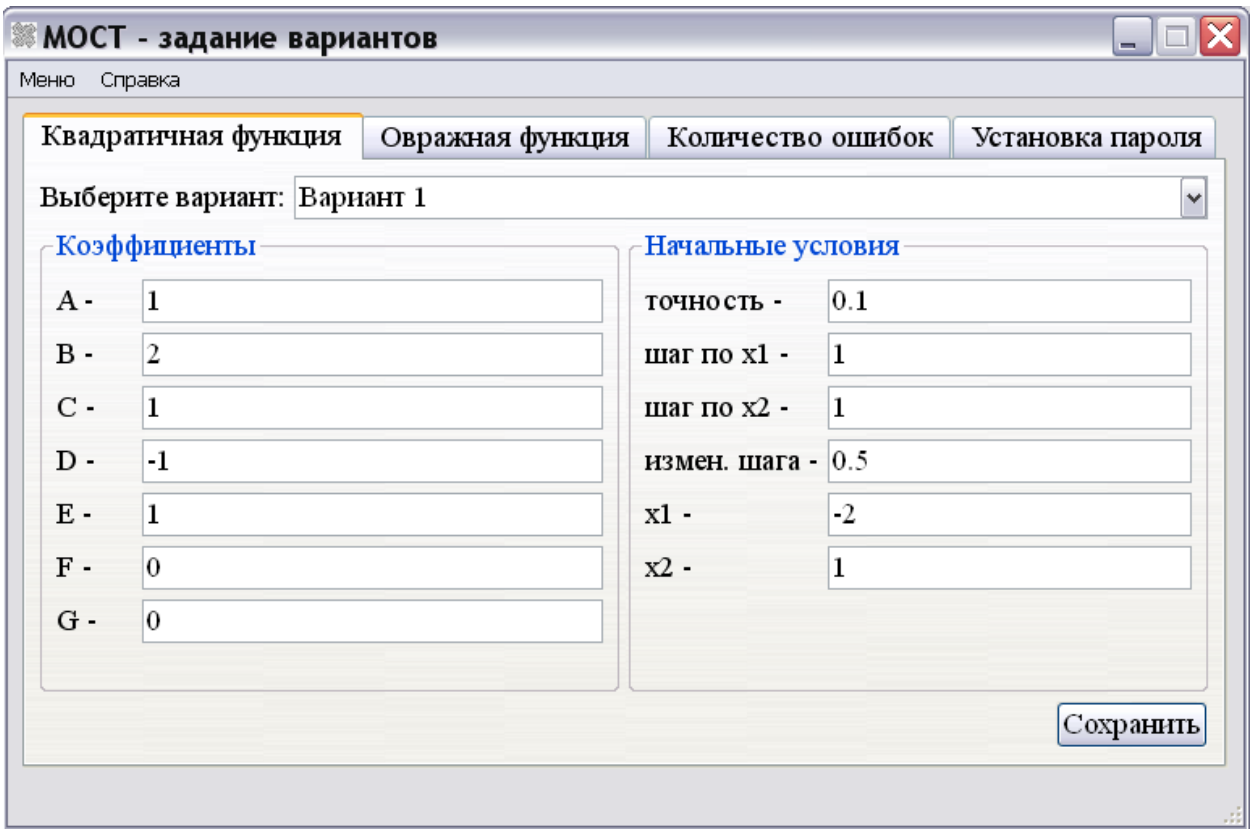

Рис. 11 Главное окно «МОСТ – задание вариантов»
(Квадратичная функция)

Квадратичная функция

На данной вкладке расположен выпадающий список для выбора уже существующего варианта и его последующего редактирования. Последний в списке вариант предназначен для добавления нового и записей не содержит.

Оставшиеся поля сгруппированы в две колонки:

- Коэффициенты – содержит поля, предназначенные для ввода коэффициентов квадратичной функции;
- Начальные условия – содержит поля задания точности, установки длин шагов и коэффициента изменения шага, а также задание начальной точки.

В поля можно вводить только числа (целые и десятичные). При вводе десятичных чисел в качестве разделителя можно использовать как символ «,», так и символ «.».

Данные вариантов хранятся во вложенной директории variants/quadFunction в файлах без расширения, именованных цифрой, равной номеру варианта минус один. Открыть файл можно любым текстовым редактором.

Овражная функция

На данной вкладке расположен выпадающий список для выбора уже существующего варианта и его последующего редактирования. Последний в списке вариант предназначен для добавления нового и записей не содержит.

Оставшиеся поля сгруппированы в две колонки:

- Коэффициенты – содержит поля, предназначенные для ввода коэффициентов овражной функции;
- Начальные условия – содержит поля задания точности, установки длин шагов и коэффициента изменения шага, а также задание начальной точки.

В поля можно вводить только числа (целые и десятичные). При вводе десятичных чисел в качестве разделителя можно использовать как символ «,», так и символ «.».

Данные вариантов хранятся во вложенной директории variants/ravinFunction в файлах без расширения, именованных цифрой, равной номеру варианта минус один. Открыть файл можно любым текстовым редактором.

Количество ошибок

В поля можно вводить только числа (целые и десятичные).

Данная вкладка содержит поля для ввода максимально допустимого количества ошибок для каждого метода.

Данные о количестве ошибок хранятся во вложенной папке variants в файле me.dll. Открыть файл можно любым текстовым редактором.

Установка пароля

Данная вкладка содержит поле для ввода старого пароля, и два поля для ввода нового.

Для установки нового пароля необходимо ввести старый пароль, а затем дважды новый.

В случае если пароль будет забыт, установить новый можно удалив во вложенной папке variants файл ps.dll и введя в данной программе новый пароль в соответствующие поля (поле старого пароля можно оставить не заполненным).

Пароль хранится в хешированном виде во вложенной папке variants в файле ps.dll. Данный файл является бинарным и при попытке открыть его текстовым редактором пароль перестает быть действительным – необходима его переустановка или замена файла ps.dll на рабочий.

5. Руководство программиста

Программа поставляется с открытым исходным кодом. Код тщательно прокомментирован. Документацию, собранную системой «Doxygen», см. в приложении 1.

Заключение

В ходе работы над программой были достигнуты цели, поставленные перед началом работ. А именно была разработана обучающая программа по дисциплине «Методы оптимизации» «МОСТ». Соблюдены все требования, представленные в техническом задании к проекту. А именно было разработано приложение, лишенное всех недостатков уже существующих программ данного класса, в которое ко всему прочему были применены все пожелания, высказанные заказчиком, например кросплатформенность приложения.

В ходе работы над программой была глубоко изучена кросплатформенная библиотека QT и при помощи нее реализована переносимости системы под платформы, отличные от Windows. Были изучены и закреплены навыки по работе с UML диаграммами и их созданию, а также приложениями для реализации этих процессов.

Список литературы

1) Qt4.5. Профессиональное программирование на C++. – Спб.: БХВ-Петербург, 2010. – 896 с.: ил. + DVD – (В подлиннике) ISBN 978-5-9775-0398-3
2) Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. — СПб: Питер, 2001. — 368 с.: ил. (Серия ≪Библиотека программиста≫) ISBN 5-272-00355-1
3) Васильев Ф.П., Методы оптимизации. – М: Факториал Пресс, 2002. – 824 с.: ил. ISBN: 5-88688-056-9

Приложение 1

Руководство программиста

Класс AlgoritmoWin

Базовый класс для окна прохождения алгоритма.

Производные классы:**CWdescent_mdImpl**, **CWdescentWinImpl**, **FasterDescentImpl**, **HuGiImpl**, **NeMiImpl** и **NotWenImpl**.

Сигналы

void **usiloPlenumis** (int, int)

Пользователь прошёл тест.

Открытые члены

AlgoritmoWin (**funkcio** *f, QVector< double > d, QWidget *parent=0, Qt::WindowFlags flags=0)

Защищенные члены

QString **textoFunkcio** ()

Получить текст функции.

Защищенные данные

funkcio * F

Целевая функция.

QVector< double > **D**

Данные задания.

MapoPorFunkcioImpl * MapoWdg

Указатель на виджет - карту высот целевой функции.

spuro * Sp

Указатель на след.

qreal **strikteco**

Точность.

int **KvantoEraroj**

Количество ошибок .

int **NumeroIteracio**

Номер итерации.

Подробное описание

Базовый класс для окна прохождения алгоритма.

Конструктор(ы)

AlgoritmoWin (funkcio * *f*, QVector< double > *d*, QWidget * *parent* = 0, Qt::WindowFlags *flags* = 0)

Аргументы:

f Указатель на целевую функцию. **ScenoPorMapo** не заботится о назначении Funkcio родителя.
d Массив с данными задания.
parent Родитель.
flags Флаги парамметров окна.

Методы

QString textoFunkcio () [protected]

Получить текст функции.

Формирует текст функции в иде html страницы для отображения в верху окна.

Класс CWdescent_mdImpl

Окно для прохождения Покоординатного спуска с минимизацией по направлению.

Базовые классы:**AlgoritmoWin**.

Открытые члены

CWdescent_mdImpl (**funkcio** *f, QVector< double > d, QWidget *parent=0, Qt::WFlags flags=0)

Защищенные члены

DemonstrataQPointF LengthOfStepX1 (**DemonstrataQPointF** X) const

Возвращает точку, полученную при оптимизации функции одной переменной по оси X1.

DemonstrataQPointF LengthOfStepX2 (**DemonstrataQPointF** X) const

Возвращает точку, полученную при оптимизации функции одной переменной по оси X2.

Защищенные данные

int **quanError**

Количество ошибок допустимых.

DemonstrataQPointF BP

Базовая точка итерации.

DemonstrataQPointF MP

Текущая базовая точка.

Подробное описание

Окно для прохождения Покоординатного спуска с минимизацией по направлению.

Алгоритм:

Операции алгоритма повторяются циклически, пока значение не приблизится к минимуму на заданную точность.

Каждая итерация содержит следующие шаги: Шаг1 Вычисляем минимум целевой функции F(X1; X2) вдоль оси X1 (идет оптимизация одной переменной). Шаг2 Вычисляем минимум целевой функции F(X1; X2) вдоль оси X2 (идет оптимизация одной переменной). Шаг3 Если max(dF, dX) < E, то завершаем процесс минимизации. Иначе - переходим к шагу 1.

Реализация:

Реализовано в виде конечного автомата. См. обзор "Каркас конечного автомата" в документации Qt.

Код написан согласно диаграмме состояний.

тт - текущая точка. бт - базовая точка; в ней сохраняется значение тт в начале итерации. Это позволяет вычислить расстояние на которое сместилась тт за итерацию.

Состояния алгоритма заключены в сложное состояние. Дочернии состояния наследуют переходы родителя, но переходы назначенные непосредственно состоянию имеют приоритет перед унаследованными. so имеет переходы без цели. Они не изменяют состояние, а только высылают сигнал. По этому сигналу регистрирую ошибку пользователя.

Перед запуском автомата задаю переменным начальные значения. См. init(). Использую этот слот и при переходе вызванным действием "Начать заново".

Применяю **DemonstrataQPointF** для BP, MP. В конструкторе окна соединяю их с надписями и картой, а затем использую как обычные QPointF, не заботясь об отображении.

Конструктор(ы)

CWdescent_mdImpl (funkcio * *f*, QVector< double > *d*, QWidget * *parent* = 0, Qt::WFlags *flags* = 0)

Аргументы:

f Указатель на целевую функцию. **CWdescent_mdImpl** не заботится о назначении Funkcio родителя.
d Массив с данными задания.Структура: 0 - точность; 1 - шаг по оси x1; 2 - шаг по оси x2; 3 - коэффициент изменения шага; 4 - x1; 5 - x2; 6 - максимальное количество ошибок.
parent Родитель.
flags Флаги параметров окна.

Методы

DemonstrataQPointF LengthOfStepX1 (DemonstrataQPointF *X*) const [protected]

Возвращает точку, полученную при оптимизации функции одной переменной по оси X1.

Возвращает длину шага для оптимизации функции одной переменной по оси X1.

DemonstrataQPointF LengthOfStepX2 (DemonstrataQPointF *X*) const [protected]

Возвращает точку, полученную при оптимизации функции одной переменной по оси X2.

Возвращает длину шага для оптимизации функции одной переменной по оси X2.

Класс CWdescentWinImpl

Окно для прохождения Покоординатного спуска с фиксированным шагом.

Базовые классы:**AlgoritmoWin**.

Сигналы

void **stateHasEntered** ()

Использую сигнал для прехода, который не требует действий пользователя, а только проверяет условие.

Открытые члены

CWdescentWinImpl (**funkcio** *f, QVector< double > d, QWidget *parent=0, Qt::WFlags flags=0)

Конструктор.

Защищенные данные

DemonstrataQPointF BP

Базовая точка итерации.

DemonstrataQPointF MP

Текущая базовая точка.

DemonstrataQPointF NP

Новая точка.

DemonstrataQPointF PX1

Шаг по x1.

DemonstrataQPointF PX2

Шаг по x2.

qreal **ModPX**

Модификатор шага.

Подробное описание

Окно для прохождения Покоординатного спуска с фиксированным шагом.

Алгоритм:

Операции алгоритма повторяются циклически, пока значение не приблизится к минимуму на заданную точность.

Каждая итерация содержит следующие шаги:

1) Изменяем x1 в положительную сторону на значение h1.

2) Если значение F(x1 + h1, x2) < F(x1, x2), то необходимо принять новую точку и проверить условие выхода, иначе не принимаем её.

3) Если не приняли, то изменяем x1 в отрицательную сторону на значение h1.

4) Если значение F(x1 - h1, x2) < F(x1, x2), то необходимо принять новую точку и проверить условие выхода, иначе не принимаем её.

5) Изменяем x2 в положительную сторону на значение h2.

6) Если значение F(x1, x2 + h2) < F(x1, x2), то необходимо принять новую точку и проверить условие выхода, иначе не принимаем её.

7) Если не приняли, то изменяем x2 в отрицательную сторону на значение h2.

8) Если значение F(x1, x2 - h2) < F(x1, x2), то необходимо принять новую точку и проверить условие выхода, иначе не принимаем её.

9) Если не приняли ни одно новое значения для x1 и x2, то уменьшаем длину шага.

10) Проверяем условие завершения:

Если расстояние от предыдущей базовой точки до текущей меньше требуемой точности, то завершаем поиск, иначе переходим к следующей итерации.

1. Конец поиска
2. Реализация:

Реализовано в виде конечного автомата. См. обзор "Каркас конечного автомата" в документации Qt.

Код написан согласно диаграмме состояний.

тт - текущая точка. нт - новая точка. бт - базовая точка; в ней сохраняется значение тт в начале итерации. Это позволяет вычислить расстояние на которое сместилась тт за итерацию.

Состояния алгоритма заключены в сложное состояние. Дочернии состояния наследуют переходы родителя, но переходы назначенные непосредственно состоянию имеют приоритет перед унаследованными. so имеет переходы без цели. Они не изменяют состояние, а только высылают сигнал. По этому сигналу регистрирую ошибку пользователя.

Перед запуском автомата задаю переменным начальные значения. См. init(). Использую этот слот и при переходе вызванным действием "Начать заново".

Применяю **DemonstrataQPointF** для BP, MP, NP, pX1, pX2. В конструкторе окна соединяю их с надписями и картой, а затем использую как обычные QPointF, не заботясь об отображении.

Конструктор(ы)

CWdescentWinImpl (funkcio * *f*, QVector< double > *d*, QWidget * *parent* = 0, Qt::WFlags *flags* = 0)

Конструктор.

Аргументы:

f Указатель на целевую функцию. **CWdescentWinImpl** не заботится о назначении Funkcio родителя.
d Массив с данными задания.Структура: 0 - точность; 1 - шаг по оси x1; 2 - шаг по оси x2; 3 - коэффициент изменения шага; 4 - x1; 5 - x2; 6 - максимальное количество ошибок.
parent Родитель.
flags Флаги параметров окна.

Данные класса

DemonstrataQPointF PX1 [protected]

Шаг по x1.

Для удобства задаю в виде точки (длина, 0).

DemonstrataQPointF PX2 [protected]

Шаг по x2.

Для удобства задаю в виде точки (0, длина).

Класс DemonstrataQPointF

Расширение QPointF, для использования совместно с **SignalantoPorPointF**.

Открытые члены

void **difiniSignalantoPorPointF** (**SignalantoPorPointF** *SP)

Устанавливает указатель на "сигнализатор".

void **setX** (qreal x)

Устанавливает новое значение X.

void **setY** (qreal y)

Устанавливает новое значение Y.

Подробное описание

Расширение QPointF, для использования совместно с **SignalantoPorPointF**.

Совместное использование **SignalantoPorPointF** и **DemonstrataQPointF** позволяет высылать сигналы со значениями точки, при каждом её изменении. Это позволяет соединнить объект **DemonstrataQPointF** с элементами отображения в окне и затем работать с ним, как с обычной точкой(QPointF), не заботясь об отображении. MOC сам всё сделает.

Пример:

```
KvadratigantoFunkcio f;
DemonstrataQPointF p;
SignalantoPorPointF sp(&p, &f);
connect(&sp, SIGNAL(proviziValoro(const QString &)), lb, SLOT(setTEXT(const QString &)));
...
p *= 6;
```

В этом классе переопределены все методы, изменяющие точку. В них добавлен вызов **SignalantoPorPointF::SendiSignaloj()**. Для самого изменения используется реализация из QPointF. Указатель на **SignalantoPorPointF** передаётся в метод difiniSignalantoPorPointF. Обычно он

вызывается в конструкторе **SignalantoPorPointF**. Если объекту **DemonstrataQPointF** не назначен объект **SignalantoPorPointF**, то он ни чем не отличается от QPointF.

См. также:

SignalantoPorPointF.

Класс FasterDescentImpl

Окно для прохождения метода наискорейшего спуска.

Базовые классы:**AlgoritmoWin**.

Открытые слоты

static bool **DerivativeQuad** (**funkcio** ***F**, **DemonstrataQPointF** ***BP**, QLineEdit *dfdx1, QLineEdit *dfdx2)

Слот, для выполнения скрипта проверки производной в квадратичной функции.

static bool **DerivativeRavin** (**funkcio** ***F**, **DemonstrataQPointF** ***BP**, QLineEdit *dfdx1, QLineEdit *dfdx2)

Слот, для выполнения скрипта проверки производной в овражной функции.

Сигналы

void **stateHasEntered** ()

Использую сигнал для прехода, который не требует действий пользователя, а только проверяет условие.

Открытые члены

FasterDescentImpl (**funkcio** *f, QVector< double > d, QWidget *parent=0, Qt::WFlags flags=0)

Конструктор.

Открытые статические члены

static QString **otrNumberSign** (double a)

Возвращает противоположный знак числа.

static QString **numberSign** (double a)

Возвращает знак числа.

Защищенные данные

int **quanError**

Количество ошибок допустимых.

DemonstrataQPointF BP

Базовая точка итерации.

DemonstrataQPointF grad

Градиент.

DemonstrataQPointF lengthStep

Длина шага.

Подробное описание

Окно для прохождения метода наискорейшего спуска.

Алгоритм:

Операции алгоритма повторяются циклически, пока значение не приблизится к минимуму на заданную точность.

Каждая итерация содержит следующие шаги: Шаг1 Выбираем направление для минимизации как антиградиент в исходной точке с координатами (X1; X2) для функции F(X1; X2). Шаг2 Вводим координаты градиента равные частным производным функции F(X). Если уже введены, то переходим к шагу 3. Шаг3 Если длина антиградиента меньше E, то минимум найден. Иначе - переходим к шагу 4. Шаг4 Вычисляем длину шага a = min F(X + a*S) (Ищется любым

методом одномерной оптимизации). Шаг5 Делаем шаг и принимаем новую точку. Переходим к шагу 1.

Реализация:

Реализовано в виде конечного автомата. См. обзор "Каркас конечного автомата" в документации Qt.

Код написан согласно диаграмме состояний.

бт - базовая точка; в ней сохраняется значение тт в начале итерации. Это позволяет вычислить расстояние на которое сместилась тт за итерацию.

Состояния алгоритма заключены в сложное состояние. Дочернии состояния наследуют переходы родителя, но переходы назначенные непосредственно состоянию имеют приоритет перед унаследованными. so имеет переходы без цели. Они не изменяют состояние, а только высылают сигнал. По этому сигналу регистрирую ошибку пользователя.

Перед запуском автомата задаю переменным начальные значения. См. init(). Использую этот слот и при переходе вызванным действием "Начать заново".

Применяю **DemonstrataQPointF** для BP, grad. В конструкторе окна соединяю их с надписями и картой, а затем использую как обычные QPointF, не заботясь об отображении.

Конструктор(ы)

FasterDescentImpl (funkcio * *f*, QVector< double > *d*, QWidget * *parent* = 0, Qt::WFlags *flags* = 0)

Аргументы:

f Указатель на целевую функцию. **FasterDescentImpl** не заботится о назначении Funkcio родителя.
d Массив с данными задания.Структура: 0 - точность; 1 - шаг по оси x1; 2 - шаг по оси x2; 3 - коэффициент изменения шага; 4 - x1; 5 - x2; 6 - максимальное количество ошибок.
parent Родитель.
flags Флаги параметров окна.

Класс funkcio

Класс для целевой функции.

Производные классы:**KvadratigantoFunkcio** и **RavinaFunkcio**.

Открытые члены

virtual double **df_dx1** (const QVector< double > X) const =0

Возвращает значение частной производной первого порядка по x1.

virtual double **df_dx1** (const QPointF X) const =0

Возвращает значение частной производной первого порядка по x1.

virtual double **df_dx2** (const QVector< double > X) const =0

Возвращает значение частной производной первого порядка по x2.

virtual double **df_dx2** (const QPointF X) const =0

Возвращает значение частной производной первого порядка по x2.

double **lengthOfStep** (const QVector< double > X, const double e) const

Возвращает длину шага для оптимизации функции одной переменной.

double **lengthOfStep** (const QPointF X) const

Возвращает длину шага для оптимизации функции одной переменной.

virtual double **detGessian** (const QPointF *X) const =0

Возвращает определитель гессиана.

virtual double **rezulto** (const double x1, const double x2) const =0

Возвращает результат вычисления функции в точке.

virtual double **rezulto** (const QVector< double > X) const =0

Возвращает результат вычисления функции в точке.

virtual double **rezulto** (const QPointF &X) const

Возвращает результат вычисления функции в точке.

virtual QVector< double > **minPoint** (const double e) const

Возвращает точку минимума функции.

funkcio (QObject *parent=0)

Подробное описание

Класс для целевой функции.

Базавый класс для всех целевых функций.

Если хотите добавить свою, унаследуите этот класс и реализуйте все виртуальные методы. Совместимость проекта с наследниками этого класса гарантирована.

При разработки собственных прохождений какого либо алгоритма используите только этот класс для целевой функции.

Пример:

```
CWdescentWinImpl( funkcio *f,
                  QVector<double> *d,
                  QWidget * parent = 0,
                  Qt::WFlags flags = 0
                );
```

Конструктор(ы)

funkcio (QObject * *parent* = 0) [inline]

Аргументы:

parent Родитель.

Методы

double lengthOfStep (const QVector< double > *X*, const double *e*) const

Возвращает длину шага для оптимизации функции одной переменной.

Реализация методов класса-родитекля для целевых функций.

QVector< double > minPoint (const double *e*) const [virtual]

Возвращает точку минимума функции.

Аргументы:

e Точность.

virtual double rezulto (const QPointF & *X*) const [inline, virtual]

Возвращает результат вычисления функции в точке.

Аргументы:

X Точка.

Переопределяется в **KvadratigantoFunkcio** и **RavinaFunkcio.**

virtual double rezulto (const QVector< double > *X*) const [pure virtual]

Возвращает результат вычисления функции в точке.

Аргументы:

X Точка.

Замещается в **KvadratigantoFunkcio** и **RavinaFunkcio**.

virtual double rezulto (const double *x1*, const double *x2*) const [pure virtual]

Возвращает результат вычисления функции в точке.

Аргументы:

x1 Первая координата точки.
x2 Вторая координата точки.

Замещается в **KvadratigantoFunkcio** и **RavinaFunkcio** .

Класс HuGiImpl

Окно для прохождения метода Хука-Дживса.

Базовые классы:**AlgoritmoWin**.

Открытые члены

HuGiImpl (**funkcio** *f, QVector< double > d, QWidget *parent=0, Qt::WFlags flags=0)

Конструктор.

Подробное описание

Окно для прохождения метода Хука-Дживса.

Конструктор(ы)

HuGiImpl (funkcio * *f*, QVector< double > *d*, QWidget * *parent* = 0, Qt::WFlags *flags* = 0)

Конструктор.

Аргументы:

f Указатель на целевую функцию. **HuGiImpl** не заботится о назначении Funkcio родителя.
d Массив с данными задания.Структура: 0 - точность; 1 - шаг по оси x1; 2 - шаг по оси x2; 3 - коэффициент изменения шага; 4 - x1; 5 - x2; 6 - максимальное количество ошибок.
parent Родитель.
flags Флаги параметров окна.

Класс KvadratigantoFunkcio

Реализация целевой квадратичной функции.

Базовые классы:**funkcio**.

Открытые члены

double **df_dx1** (const QVector< double > X) const

Возвращает значение частной производной первого порядка по x1.

double **df_dx1** (const QPointF X) const

Возвращает значение частной производной первого порядка по x1.

double **df_dx2** (const QVector< double > X) const

Возвращает значение частной производной первого порядка по x2.

double **df_dx2** (const QPointF X) const

Возвращает значение частной производной первого порядка по x2.

double **detGessian** (const QPointF *X) const

Возвращает определитель гессиана.

double **rezulto** (const double x1, const double x2) const

Возвращает результат вычисления функции в точке.

double **rezulto** (const QVector< double > X) const

Возвращает результат вычисления функции в точке.

double **rezulto** (const QPointF &X) const

Возвращает результат вычисления функции в точке.

KvadratigantoFunkcio (QObject *parent=0)

*Конструктор класса **KvadratigantoFunkcio** по умолчанию.*

KvadratigantoFunkcio (double A, double B, double C, double D, double E, double F, double G, QObject *parent=0)

*Конструктор класса **KvadratigantoFunkcio**.*

KvadratigantoFunkcio (QVector< double > data, QObject *parent=0)

*Конструктор класса **KvadratigantoFunkcio**.*

Подробное описание

Реализация целевой квадратичной функции.

A*pow((x1 - B), 2) + C*pow((x2 - D), 2) + E*(x1 - F)*(x2 - G)

Конструктор(ы)

KvadratigantoFunkcio (QObject * *parent* = 0) [inline]

Аргументы:

parent Родитель.

KvadratigantoFunkcio (double *A*, double *B*, double *C*, double *D*, double *E*, double *F*, double *G*, QObject * *parent* = 0) [inline]

Аргументы:

parent Родитель.

KvadratigantoFunkcio (QVector< double > *data*, QObject * *parent* = 0) [inline]

Аргументы:

parent Родитель.
data Массив с параметрами функции.

Методы

double rezulto (const QPointF & *X*) const [inline, virtual]

Возвращает результат вычисления функции в точке.

Аргументы:

X Точка.

Переопределяет метод предка **funkcio**.

double rezulto (const QVector< double > *X*) const [inline, virtual]

Возвращает результат вычисления функции в точке.

Аргументы:

X Точка.

Замещает **funkcio**.

double rezulto (const double *x1*, const double *x2*) const [inline, virtual]

Возвращает результат вычисления функции в точке.

Аргументы:

x1 Первая координата точки.
x2 Вторая координата точки.

Замещает **funkcio**.

Класс MapoPorFunkcioImpl

Виджет для отображения карты высот целевой функции и "следа алгоритма на ней".

Открытые слоты

void **difiniFonaKoloro** (QColor)

Устанавливает цвет карты.

void **setScale** (qreal factor)

Устанавливает масштаб.

void **difiniFunkcio** (**funkcio** *f)

Устанавливает целевую функцию.

void **difiniSpuro** (**spuro** *Spuro)

Устанавливает на сцене "след" алгоритма.

Сигналы

void **MusaPosX** (const qreal)

Передаёт координату X мыши.

void **MusaPosY** (const qreal)

Передаёт координату Y мыши.

void **MusaPos** (const QString &)

Передаёт координаты мыши в виде строки текста.

Открытые члены

QColor **fonaKoloro** () const

Возвращает цвет карты.

MapoPorFunkcioImpl (const **funkcio** *Funkcio, QWidget *parent=0, Qt::WFlags f=0)
MapoPorFunkcioImpl (QWidget *parent=0, Qt::WFlags f=0)
const **spuro** * **proviziSpuro** () const

Возвращает указатель на "след" прохождения алгоритма.

qreal **Scale** () const

Возвращает множитель масштабирования.

Подробное описание

Виджет для отображения карты высот целевой функции и "следа алгоритма на ней".

Карта высот отображается на фоне сцены.

Поверх её рисуется "след" оставляемый алгоритмом. Каждый алгоритм имеет свой "след". Базовым для всех "следов" является класс spuro.

При создании объект **MapoPorFunkcioImpl** не имеет "следа". Необходимо самостоятельно создать "след" нужного алгоритма и передать указатель на него в объект **MapoPorFunkcioImpl** с помощью метода **difiniSpuro(spuro * S)**.

Руководство по работе со следом конкретного алгоритма см. в описании соответствующего класса.

spuroSinkoLauxKoordinatoj - "след" по координатного спуска с фиксированным шагом.

Метод const spuro * **proviziSpuro()** возвращает указатель на существующий "след" прохождения алгоритма.

Конструктор(ы)

MapoPorFunkcioImpl (const funkcio * *Funkcio*, QWidget * *parent* = 0, Qt::WFlags *f* = 0)

Аргументы:

Funkcio Указатель на целевую функцию. **ScenoPorMapo** не заботится о назначении Funkcio родителя.
parent Родитель.
f Флаги параметров окна.

MapoPorFunkcioImpl (QWidget * *parent* = 0, Qt::WFlags *f* = 0)

Аргументы:

parent Родитель.
f Флаги параметров окна.

Методы

const spuro * proviziSpuro () const

Возвращает указатель на "след" прохождения алгоритма.

"Следы" разных алгоритмов имеют разные слоты.

См. также:

spuro, **spuroSinkoLauxKoordinatoj**.

void setScale (qreal *factor*) [slot]

Устанавливает масштаб.

Аргументы:

factor Множитель масштабирования. factor = 1 - масштаб 1:1.

Класс NeMiImpl

Окно для прохождения метода Нелдора-Мида.

Базовые классы:**AlgoritmoWin**.

Сигналы

void **stateHasEntered** ()

Использую сигнал для прехода, который не требует действий пользователя, а только проверяет условие.

Защищенные данные

DemonstrataQPointF P1

Первая точка основного триугольника.

DemonstrataQPointF P2

Вторая точка основного триугольника.

DemonstrataQPointF P3

Третья точка основного триугольника.

DemonstrataQPointF * Ph

Указатель на наибольшую точку.

DemonstrataQPointF * Pm

Указатель на средную точку.

DemonstrataQPointF * Pl

Указатель на наименьшую точку.

QPointF **Pc**

Центр тяжести триугольника.

DemonstrataQPointF PR

Отражённая точка.

DemonstrataQPointF PK

Сжатая точка.

DemonstrataQPointF PD

Растянутая точка.

Подробное описание

Окно для прохождения метода Нелдора-Мида.

Класс RavinaFunkcio

Реализация целевой "овражной" функции.

Базовые классы:**funkcio**.

Открытые члены

double **df_dx1** (const QVector< double > X) const

Возвращает значение частной производной первого порядка по x1.

double **df_dx1** (const QPointF X) const

Возвращает значение частной производной первого порядка по x1.

double **df_dx2** (const QVector< double > X) const

Возвращает значение частной производной первого порядка по x2.

double **df_dx2** (const QPointF X) const

Возвращает значение частной производной первого порядка по x2.

double **detGessian** (const QPointF *X) const

Возвращает определитель гессиана.

double **df_dx1dx1** (const QPointF *X) const

Возвращает значение частной производной второго порядка.

double **df_dx1dx2** (const QPointF *X) const

Возвращает значение частной производной второго порядка.

double **df_dx2dx2** () const

Возвращает значение частной производной второго порядка.

double **rezulto** (const double x1, const double x2) const

Возвращает результат вычисления функции в точке.

double **rezulto** (const QVector< double > X) const

Возвращает результат вычисления функции в точке.

double **rezulto** (const QPointF &X) const

Возвращает результат вычисления функции в точке.

RavinaFunkcio (QObject *parent=0)

*Конструктор класса **RavinaFunkcio** по умолчанию.*

RavinaFunkcio (double A, double B, QObject *parent=0)

*Конструктор класса **RavinaFunkcio**.*

RavinaFunkcio (QVector< double > data, QObject *parent=0)

*Конструктор класса **RavinaFunkcio**.*

Подробное описание

Реализация целевой "овражной" функции.

A*pow((x2 - pow(x1, 2)), 2) + B*pow((1-x1), 2)

Конструктор(ы)

RavinaFunkcio (QObject * *parent* = 0) [inline]

Аргументы:

parent Родитель.

RavinaFunkcio (double *A*, double *B*, QObject * *parent* = 0) [inline]

Аргументы:

parent Родитель.

RavinaFunkcio (QVector< double > *data*, QObject * *parent* = 0) [inline]

Конструктор класса **RavinaFunkcio**.

Аргументы:

parent Родитель.
data Массив с параметрами функции.

Методы

double rezulto (const QPointF & *X*) const [inline, virtual]

Возвращает результат вычисления функции в точке.

Аргументы:

X Точка.

Переопределяет метод предка **funkcio**.

double rezulto (const QVector< double > *X*) const [inline, virtual]

Возвращает результат вычисления функции в точке.

Аргументы:

X Точка.

Замещает **funkcio**.

double rezulto (const double *x1*, const double *x2*) const [inline, virtual]

Возвращает результат вычисления функции в точке.

Аргументы:

x1 Первая координата точки.
x2 Вторая координата точки.

Замещает **funkcio**.

Класс ScenoPorMapo

Сцена для карты высот целевой функции.

Открытые слоты

void **difiniKoloro** (QColor &)

Устанавливает цвет карты.

void **setScale** (qreal factor)

Устанавливает масштаб.

Сигналы

void **MusaPosX** (const qreal)

Передаёт координату X мыши.

void **MusaPosY** (const qreal)

Передаёт координату Y мыши.

void **MusaPos** (const QString &)

Передаёт координаты мыши в виде строки текста.

Открытые члены

QColor **Koloro** () const

Возвращает цвет карты.

qreal **scale** () const

Возвращает масштаб.

ScenoPorMapo (const **funkcio** *Funkcio, QObject *parent=0)

ScenoPorMapo (const **funkcio** *Funkcio, const QRectF &sceneRect, QObject *parent=0)
ScenoPorMapo (const **funkcio** *Funkcio, qreal x, qreal y, qreal width, qreal height, QObject *parent=0)

Защищенные члены

void **mouseMoveEvent** (QGraphicsSceneMouseEvent *mouseEvent)

Обработчик перемещения мыши.

void **drawBackground** (QPainter *painter, const QRectF &rect)

Отрисовывает фон сцены с использованием painter перед отрисовкой любого элемента или переднего плана.

Подробное описание

Сцена для карты высот целевой функции.

Отличается от стандартной QGraphicsScene тем, что рисует на фоне карту высот.

В классе собственная реализация масштабирования.

Конструктор(ы)

ScenoPorMapo (const funkcio * *Funkcio*, QObject * *parent* = 0)

Аргументы:

Funkcio Указатель на целевую функцию. **ScenoPorMapo** не заботится о назначении Funkcio родителя.
parent Родитель.

ScenoPorMapo (const funkcio * *Funkcio*, const QRectF & *sceneRect*, QObject * *parent* = 0)

Аргументы:

Funkcio Указатель на целевую функцию. **ScenoPorMapo** не заботится о назначении Funkcio родителя.
sceneRect Область сцены.
parent Родитель.

ScenoPorMapo (const funkcio * *Funkcio*, qreal *x*, qreal *y*, qreal *width*, qreal *height*, QObject * *parent* = 0)

Аргументы:

Funkcio Указатель на целевую функцию. **ScenoPorMapo** не заботится о назначении Funkcio родителя.
x Горизонтальная координата левого верхнего угла сцены.
y Вертиктальная координата левого верхнего угла сцены.
width Ширина сцены.
height Высота сцены.
parent Родитель.

Методы

void drawBackground (QPainter * *painter*, const QRectF & *rect*) [protected]

Отрисовывает фон сцены с использованием painter перед отрисовкой любого элемента или переднего плана.

Аргументы:

painter Контекст рисования фона сцены.

rect Область сцены.

void mouseMoveEvent (QGraphicsSceneMouseEvent * *mouseEvent*) [protected]

Обработчик перемещения мыши.

Высылаются сигналы о положении мыши. Затем вызывается реализация поумолчанию.

void setScale (qreal *factor*) [slot]

Устанавливает масштаб.

Аргументы:

factor Множитель масштабирования. factor = 1 - масштаб 1:1.

Класс SignalantoPorPointF

Используется совместно с **DemonstrataQPointF**.

Сигналы

void **proviziValoroFukcioEnPointo** (double)

Предоставляет значение функции в точке.

void **proviziValoroFukcioEnPointo** (const QString &)

Предоставляет значение функции в точке в виде строки.

void **proviziValoro** (const QPointF &)

Предоставляет значение в виде точки.

void **proviziValoro** (const QString &)

Предоставляет значение в виде строки содержащей точку.

void **proviziXValoro** (int)

Предоставляет значение в виде целого числа.

void **proviziXValoro** (double valoro)

Предоставляет значение в виде десятичной дроби.

void **proviziXValoro** (const QString &)

Предоставляет значение в виде строки.

void **proviziYValoro** (int)

Предоставляет значение в виде целого числа.

void **proviziYValoro** (double valoro)

Предоставляет значение в виде десятичной дроби.

void **proviziYValoro** (const QString &)

Предоставляет значение в виде виде строки.

Открытые члены

void **SendiSignaloj** ()

Высылает все сигналы.

Подробное описание

Используется совместно с **DemonstrataQPointF**.

Высылает сигналы при изменении точки.

Совместное использование **SignalantoPorPointF** и **DemonstrataQPointF** позволяет высылать сигналы со значениями точки, при каждом её изменении. Это позволяет соединить

объект **DemonstrataQPointF** с элементами отображения в окне и затем работать с ним, как с обычной точкой(QPointF), не заботясь об отображении. MOC сам всё сделает.

Пример:

```
KvadratigantoFunkcio f;
DemonstrataQPointF p;
SignalantoPorPointF sp(&p, &f);
connect(&sp, SIGNAL(proviziValoro(const QString &)), lb, SLOT(setTEXT(const QString &)));
...
p *= 6;
```

SignalantoPorPointF требует при создании указатель на объект **DemonstrataQPointF**. Если передать 0, то ни один сигнал не будет вылан.

Если установлен не обязательный параметр funkcio * F, то будут высылаться сигналы со значениями преданной целевой функци в точке.

Сигналы SignalojPorPointF::proviziValoroFukcioEnPointo(const QString &) и SignalojPorPointF::ValoroFukcioEnPointo(double) высылаются, только если установлена целевая функция.

Функция SendiSignaloj заставляет объект SignalojPorPointF выслать сигналы. Используется в **DemonstrataQPointF**. Можно вызвать её вручную, что бы обновить итображения точки. Обычно вызывать вручную не требуется.

Как и любой наследник QObject, **SignalantoPorPointF** может быть добавлен в иерархию объектов.

См. также:

DemonstrataQPointF.

Класс spuro

Базовый класс для отображения "следа" алгоритма.

Производные классы:spuroHuGi, **spuroNeMi**, **spuroSinkoLauxKoordinatoj** и **spuroSinkoLauxKoordinatoj_md**.

Открытые члены

virtual void **paint** (QPainter *, const QStyleOptionGraphicsItem *, QWidget *widget=0)

Реализует отрисовку элемента.

virtual QRectF **boundingRect** () const

Возвращает приблизительную площадь отрисовываемую элементом. Пустая реализция для заглушки.

qreal **scale** () const

Возвращает масштаб.

void **setScale** (qreal factor)

Устанавливает масштаб.

int **proviziIdAlgoritmo** () const

Возвращает порядковый номер алгоритма.

QColor **proviziBazaKoloro** () const

Возвращает значение основного цвета.

spuro (int IdAlgoritmo, QColor bazaKoloro, qreal Skalo=1, QGraphicsItem *parent=0)
spuro (QGraphicsItem *parent=0)

Конструктор для заглушки.

Защищенные члены

virtual QPolygonF **aplikiScalo** (QPolygonF p)

Применяет масштаб к полигонам.

Защищенные данные

int **IDAlgoritmo**

Порядковый номер алгоритма.

QColor **BazaKoloro**

Основной цвет "следа".

qreal **skalo**

Коэффициент масштаба.

bool **empty**

Флаг заглушки.

Подробное описание

Базовый класс для отображения "следа" алгоритма.

Этот класс не проверяет логику прохождения алгоритма.

Он лишь позволяет нарисовать "след" на сцене.

Конструктор(ы)

spuro (int *IdAlgoritmo*, QColor *bazaKoloro*, qreal *Skalo* = 1, QGraphicsItem * *parent* = 0)

Аргументы:

IdAlgoritmo Порядковый номер алгоритма(см. A).
bazaKoloro Основной цвет "следа".
Skalo Коэффициент масштаба.
parent Элемент родитель.

spuro (QGraphicsItem * *parent* = 0)

Конструктор для заглушки.

Аргументы:

parent Элемент родитель.

Методы

virtual void paint (QPainter *, const QStyleOptionGraphicsItem *, QWidget * *widget* = 0) [inline, virtual]

Реализует отрисовку элемента.

Аргументы:

widget Указывает на виджет, который отрисовывается; в противном случае он равен 0. Для кэшированного рисования widget всегда равен 0.

Переопределяется в **spuroNeMi** , **spuroSinkoLauxKoordinatoj** и **spuroSinkoLauxKoordinatoj_md**.

QColor proviziBazaKoloro () const [inline]

Возвращает значение основного цвета.

void setScale (qreal *factor*)

Устанавливает масштаб.

Аргументы:

factor Множитель масштабирования. factor = 1 - масштаб 1:1.

Данные класса

bool empty `[protected]`

Флаг заглушки.

true если объект - заглушка. false если объект след для конкретного алгоритма.

int IDAlgoritmo `[protected]`

Порядковый номер алгоритма.

Класс spuroNeMi

Отображает "след" по координатного спуска с дискретным шагом.

Базовые классы:**spuro**.

Открытые слоты

void **finisxiIteracio** ()

Завершить итерацию.

void **difiniP1** (const QPointF &)

Установить первую точку.

void **difiniP2** (const QPointF &)

Установить вторую точку.

void **difiniP3** (const QPointF &)

Установить третью точку.

void **difiniPRespegulo** (const QPointF &)

Установить точку отражения.

void **difiniPDilato** (const QPointF &)

Установить точку растяжения.

void **difiniPKompakto** (const QPointF &)

Установить точку сжатия.

void **difiniBazaKoloro** (QColor bazaKoloro)

Установить базовый цвет.

void **difiniMomentaKoloro** (QColor momentaKoloro)

Установить текущий цвет.

void **senspurigi** ()

Очищает "след".

Открытые члены

void **paint** (QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget=0)

Реализует отрисовку элемента.

QRectF **boundingRect** () const

Возвращает приблизительную площадь отрисовываемую элементом.

spuroNeMi (QColor momentaKoloro, QColor bazaKoloro, **funkcio** *f, qreal Skalo=1, QGraphicsItem *parent=0)

Подробное описание

Отображает "след" по координатного спуска с дискретным шагом.

Этот "след" представляет собой треугольник - "перекати поле".

Сам треугольник рисуется основным цветом(BazaKoloro), а точки поиска текущим цветом (MomentaKoloro). Для установки вершин треугольника используются методы difiniP1, difiniP2 и difiniP3. Для установки точки отражения, тоски растяжения и точки сжатия используются соответственно difiniPRespegulo, difiniPDilato и difiniPKompakto. В конце каждой итерации следует вызывать метод finisxiIteracio, который очистит точки поиска.

Конструктор(ы)

spuroNeMi (QColor *momentaKoloro*, QColor *bazaKoloro*, funkcio * *f*, qreal *Skalo* = `1`, QGraphicsItem * *parent* = `0`)

Аргументы:

momentaKoloro Текущий цвет "следа".
bazaKoloro Основной цвет "следа".
Skalo Коэффициент масштаба.
parent Элемент родитель.

Методы

void paint (QPainter * *painter*, const QStyleOptionGraphicsItem * *option*, QWidget * *widget* = `0`) `[virtual]`

Реализует отрисовку элемента.

Аргументы:

painter Контекст рисования элемента.
option Опции стилей для элементов, такие как его состояние, область отображения и подсказки степени его детализации.
widget Указывает на виджет, который отрисовывается; в противном случае он равен 0. Для кэшированного рисования widget всегда равен 0.

Переопределяет метод предка **spuro**.

Класс spuroSinkoLauxKoordinatoj

Отображает "след" по координатного спуска с дискретным шагом.

Базовые классы:**spuro**.

Открытые слоты

void **difiniUnuaPointo** (QPointF p)

Установить первую точку.

void **difiniUnuaPointo** (qreal x, qreal y)

Перегружает difiniUnuaPointo(QPointF &p).

void **finisxiIteracio** ()

Завершить итерацию.

void **reveniAlMomentoPointo** ()

Вернуться к текущей точке.

void **aldoniSercxantaPointo** (QPointF)

Добавить точку поиска.

void **difiniMomentaPointo** (QPointF)

Установить текущую точку.

void **difiniBazaKoloro** (QColor bazaKoloro)

Установить базовый цвет.

void **difiniMomentaKoloro** (QColor momentaKoloro)

Установить текущий цвет.

void **senspurigi** ()

Очищает "след".

Открытые члены

void **paint** (QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget=0)

Реализует отрисовку элемента.

QRectF **boundingRect** () const

Возвращает приблизительную площадь отрисовываемую элементом.

spuroSinkoLauxKoordinatoj (QColor momentaKoloro, QColor bazaKoloro, qreal Skalo=1, QGraphicsItem *parent=0)

Подробное описание

Отображает "след" по координатного спуска с дискретным шагом.

Этот "след" представляет собой ломаную линию, соединяющую установленные для него точки.

Рисует прошедшие итерации основным цветом - BazaKoloro, а текущую MomentaKoloro.

Этот класс не проверяет логику прохождения алгоритма. Он лишь позволяет нарисовать "след" на сцене.

Сразу же после создания "следа" необходимо задать начальную точку. Для этого используется слот void **difiniUnuaPointo(QPointF p)**.

В классе создана основная точка итерации - MomentaPointo. Это точка вокруг которой ведётся поиск. Для каждой итерации эта точка должна быть обновлена.

Чтобы добавить точку поиска воспользуйтесь слотом void **aldoniSercxantaPointo(QPointF)**. Если точка поиска не оказалась меньше основной точки итерации, то вызовите слот void **reveniAlMomentoPointo()**. Не забывайте возвращаться назад.

Если итерация закончена вызовите слот void **finisxiIteracio()**. Обратите внимание, что finisxiIteracio задаёт в качестве основной точки новой итерации последнюю точку из списка завершённой итерации. Не забывайте возвращаться назад с помощью **reveniAlMomentoPointo()**.

Конструктор(ы)

spuroSinkoLauxKoordinatoj (QColor *momentaKoloro*, QColor *bazaKoloro*, qreal *Skalo* = `1`, QGraphicsItem * *parent* = `0`)

Аргументы:

momentaKoloro Цвет текущей итерации.
bazaKoloro Основной цвет "следа".
Skalo Коэффициент масштаба.
parent Элемент родитель.

Методы

void difiniUnuaPointo (QPointF *p*) [slot]

Установить первую точку.

Устанавливает точку с которой начинается поиск.

void finisxiIteracio () [slot]

Завершить итерацию.

Делает последнюю точку из MomentaPointoj основной точкой текущей итерации. Переносит точки завершаемой итерации в "хвост". Заменяет список точек текущей итерации на вновь полученную основную точку текущей итерации.

void paint (QPainter * *painter*, const QStyleOptionGraphicsItem * *option*, QWidget * *widget* = 0) [virtual]

Реализует отрисовку элемента.

Аргументы:

painter Контекст рисования элемента.
option Опции стилей для элементов, такие как его состояние, область отображения и подсказки степени его детализации.
widget Указывает на виджет, который отрисовывается; в противном случае он равен 0. Для кэшированного рисования widget всегда равен 0.

Переопределяет метод предка **spuro**.

Класс spuroSinkoLauxKoordinatoj_md

Отображает "след" покоординатного спуска с минимизацией по направлению.

Базовые классы:**spuro**.

Открытые слоты

void **difiniUnuaPointo** (const QPointF &p)

Установить первую точку.

void **difiniUnuaPointo** (qreal x, qreal y)

Перегружает difiniUnuaPointo(QPointF &p).

void **finisxiIteracio** ()

Завершить итерацию.

void **aldoniPointo** (const QPointF &p)

Добавить точку.

void **difiniBazaKoloro** (QColor bazaKoloro)

Установить базовый цвет.

void **difiniMomentaKoloro** (QColor momentaKoloro)

Установить текущий цвет.

void **senspurigi** ()

Очищает "след".

Открытые члены

void **paint** (QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget=0)

Реализует отрисовку элемента.

QRectF **boundingRect** () const

Возвращает приблизительную площадь отрисовываемую элементом.

spuroSinkoLauxKoordinatoj_md (QColor bazaKoloro, qreal Skalo=1, QGraphicsItem *parent=0)

Подробное описание

Отображает "след" покоординатного спуска с минимизацией по направлению.

Конструктор(ы)

spuroSinkoLauxKoordinatoj_md (QColor *bazaKoloro*, qreal *Skalo* = `1`, QGraphicsItem * *parent* = `0`)

Аргументы:

bazaKoloro Основной цвет "следа".
Skalo Коэффициент масштаба.
parent Элемент родитель.

Методы

void difiniUnuaPointo (const QPointF & *p*) `[slot]`

Установить первую точку.

Устанавливает точку с которой начинается поиск.

void finisxilteracio () `[slot]`

Завершить итерацию.

Делает последнюю точку из MomentaPointoj основной точкой текущей итерации. Переносит точки завершаемой итерации в "хвост". Заменяет список точек текущей итерации на вновь полученную основную точку текущей итерации.

void paint (QPainter * *painter*, const QStyleOptionGraphicsItem * *option*, QWidget * *widget* = `0`) `[virtual]`

Реализует отрисовку элемента.

Аргументы:

painter Контекст рисования элемента.
option Опции стилей для элементов, такие как его состояние, область отображения и подсказки степени его детализации.
widget Указывает на виджет, который отрисовывается; в противном случае он равен 0. Для кэшированного рисования widget всегда равен 0.

Переопределяет метод предка **spuro**

Приложение 2

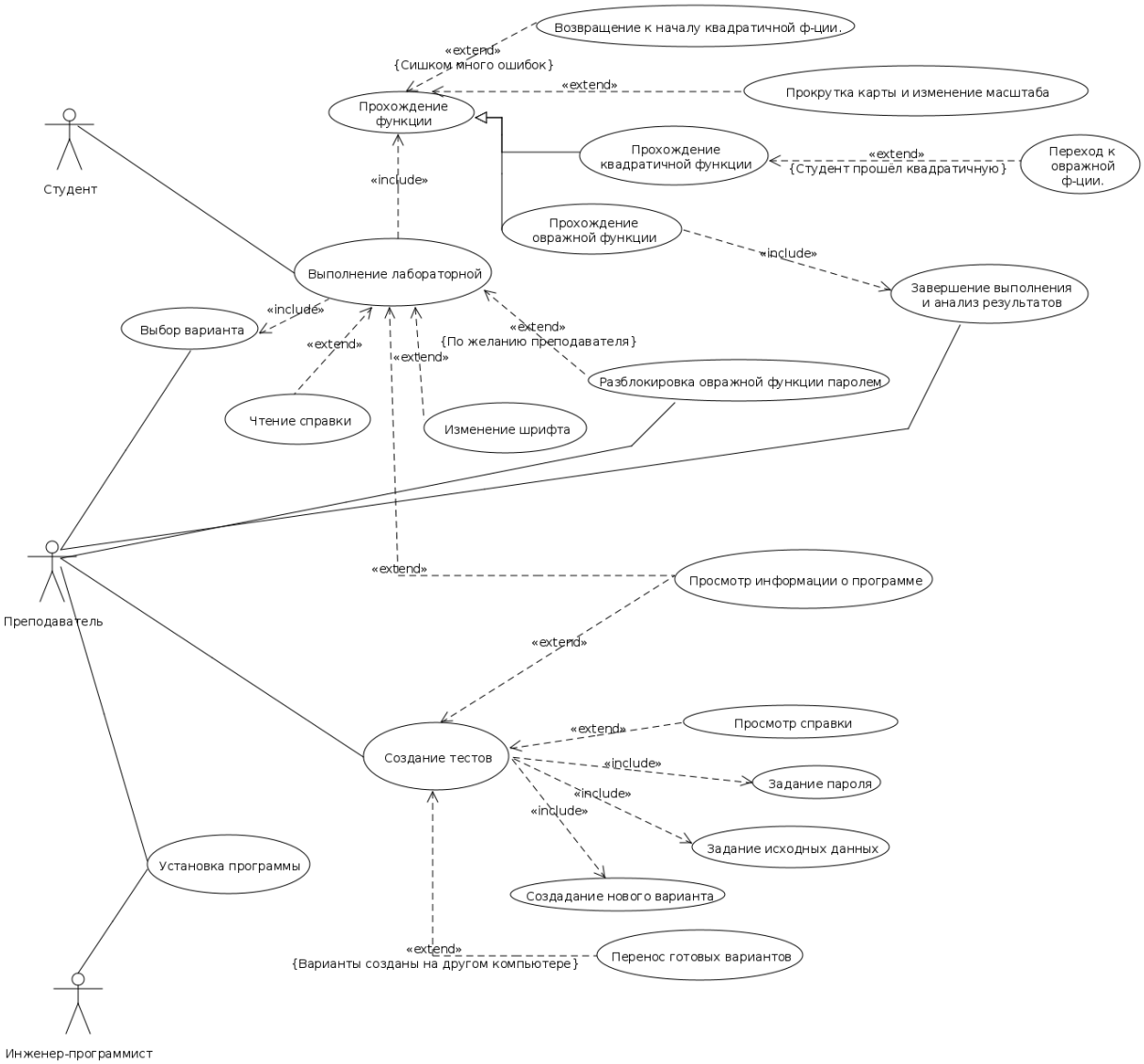

Рис12. Диаграмма прецедентов.

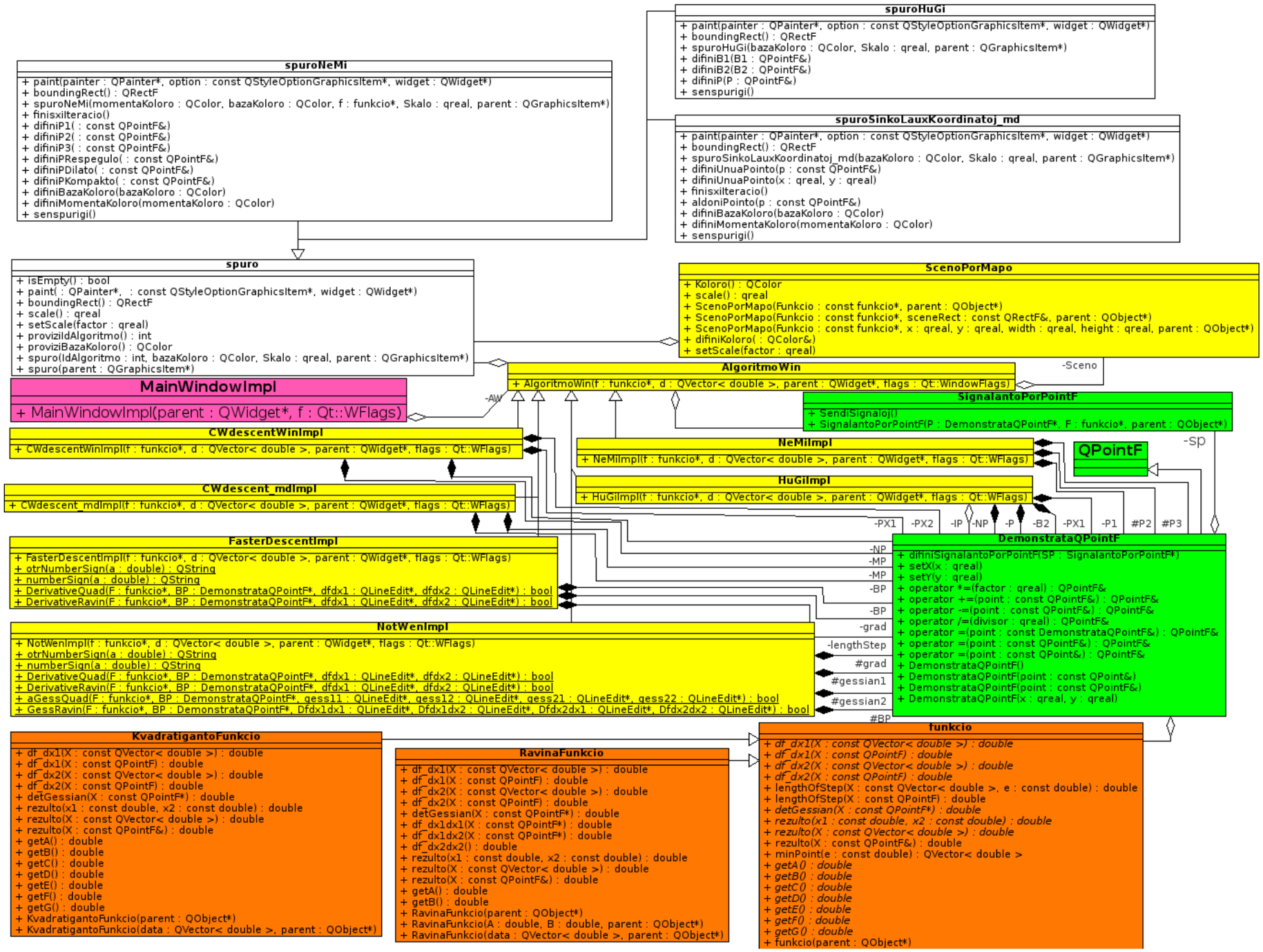

Рис. 13 Диаграмма классов

Прежде чем описать структуру классов программы необходимо указать особенности библиотеки Qt. Основная особенность это механизм сигналов и слотов. Любой наследник QObject может иметь сигналы и слоты.

```
class MyClass :  public QObject{
Q_OBJECT
public slots:
  void MySlot(int n);
signals:
  void MySignal(int n);
}
```

Слот это обычная функция с одной особенностью её можно соединить с сигналом. Сигнал синтаксически похож на функцию. Строка emit MySignal(235); равносильна вызову слота связанного с этим сигналом, с сигналом можно связать произвольное число слотов и можно изменять эти связи во время выполнения программы. Большинство классов Qt имеют слоты и высылают сигналы при изменении полей или других событиях.

Функции

Обозначено оранжевым цветом

Чтобы обеспечить работу с двумя типами функций применён паттерн стратегия. Класс funkcio обеспечивает общий интерфейс: возможность задания параметров функции и возврат её значения в точке. KvadratigantoFunkcio и RavinaFunkcio реализуют конкретные типы функций. Интерфейс funkcio избыточен, т.к. надо поддержать оба типа. Неиспользуемые параметры просто игнорируются.

Точка

Обозначено зелёным цветом

Стандартный класс двумерной точки — QPointF не высылает сигналы при своём изменении. Мы реализовали такую возможность. В результате можно связать объект точки и текстовые поля на форме при создании окна, и в остальном коде не заботится о выводе координат в текстовые поля.

Для наследников QObject нельзя создавать конструктор копирования. Поэтому был создан класс DemonstrataQPointF полностью совместимый с QPointF и вызывающий SignalantoPorPointF::SendiSignaloj() при каждом своём изменении. SignalantoPorPointF высылает сигналы с новыми значениями координат. Каждому объекту DemonstrataQPointF должен соответствовать один объект SignalantoPorPointF.

Окно выбора варианта.

Обозначено розовым цветом

MainWindowImpl — класс окна выбора алгоритма, варианта и типа функции. После выбора запускается один из наследников AlgoritmoWin.

Окна прохождения алгоритмов

Обозначено жёлтым цветом

Для обеспечения работы с разными алгоритмами был использован паттерн стратегия. Абстрактный класс AlgoritmoWin обеспечивает общее для всех поведение и базовый интерфейс: запуск из MainWindowImpl с заданными функцией, начальной точкой, числом возможных ошибок. В наследниках AlgoritmoWin реализованы алгоритмы: CWdescentWinImpl — покоординатный спуск с фиксированным шагом, CWdescent_mdImpl — покоординатный спуск с минимизацией по направлению, FasterDescentImpl — наискорейший спуск, HuGiImpl — метод Хука-Дживса, NeMiImpl — метод Нелдора-Мида, NotWenImpl — метод Ньютона. Описания алгоритмов см. в книге Васильев Ф.П., Методы оптимизации..

Для пошагового прохождения алгоритмов был использован паттерн конечный автомат. Библиотека Qt имеет каркас для поддержки этого паттерна (см. http://doc.crossplatform.ru/qt/4.6.x/statemachine-api.html). Диаграммы состояний для алгоритмов см. в приложении 4.

Карта высот функции

Обозначено жёлтым цветом

Для карты высот был создан собственный виджет — ScenoPorMapo. При создании ему передаётся функция, карту высот которой надо построить.

След прохождения

Обозначено белым цветом.

Для разных алгоритмов надо рисовать разные следы на карте. Это реализовано с помощью паттерна состояние. В конструкторе окна метода создаётся соответствующий объект следа. По средствам сигналов и слотов связывается с точками. И указатель на него передаётся в виджет карты.

Диаграммы состояний

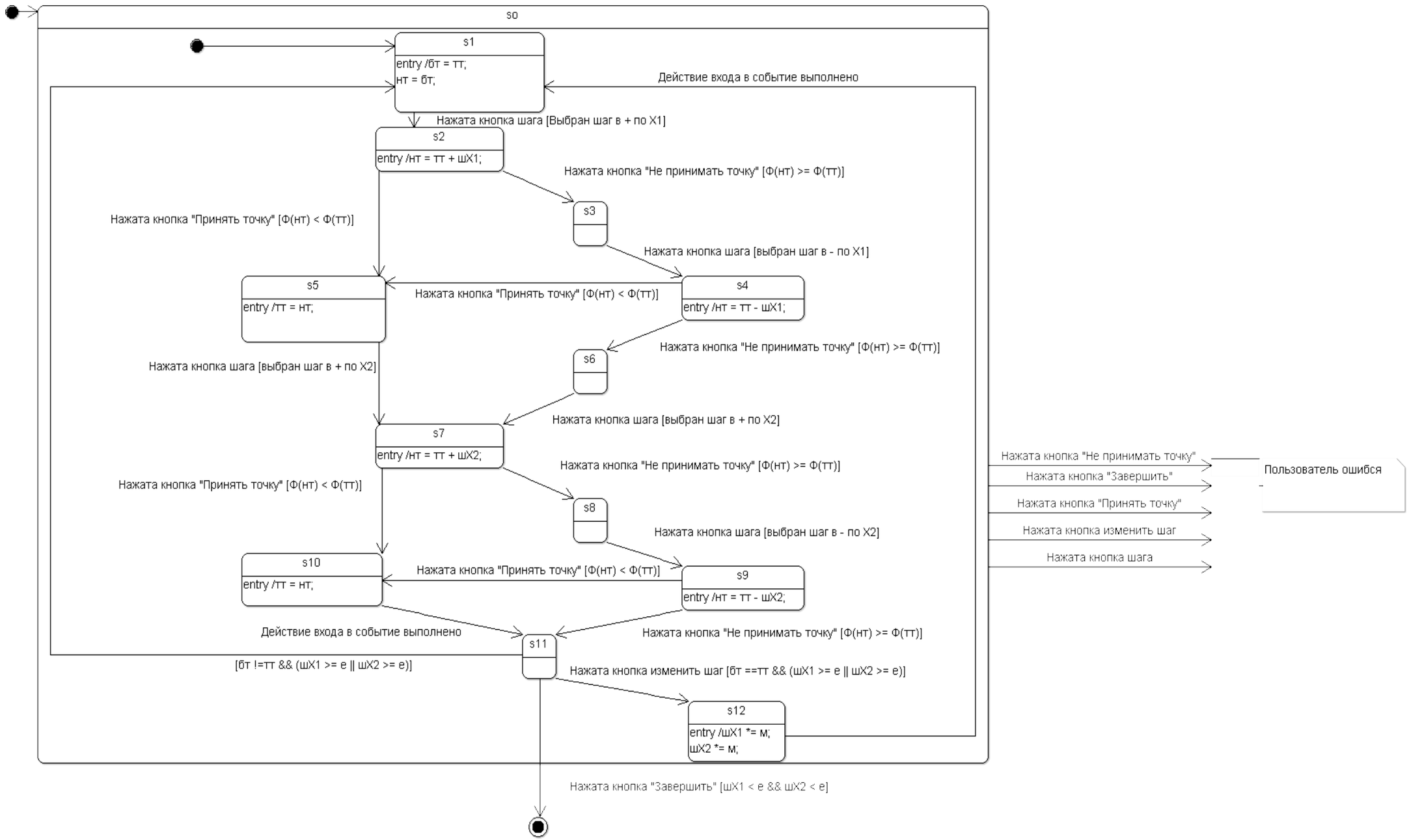

Рис. 14 Диаграмма состояний к методу покоординатного спуска с дискретным шагом

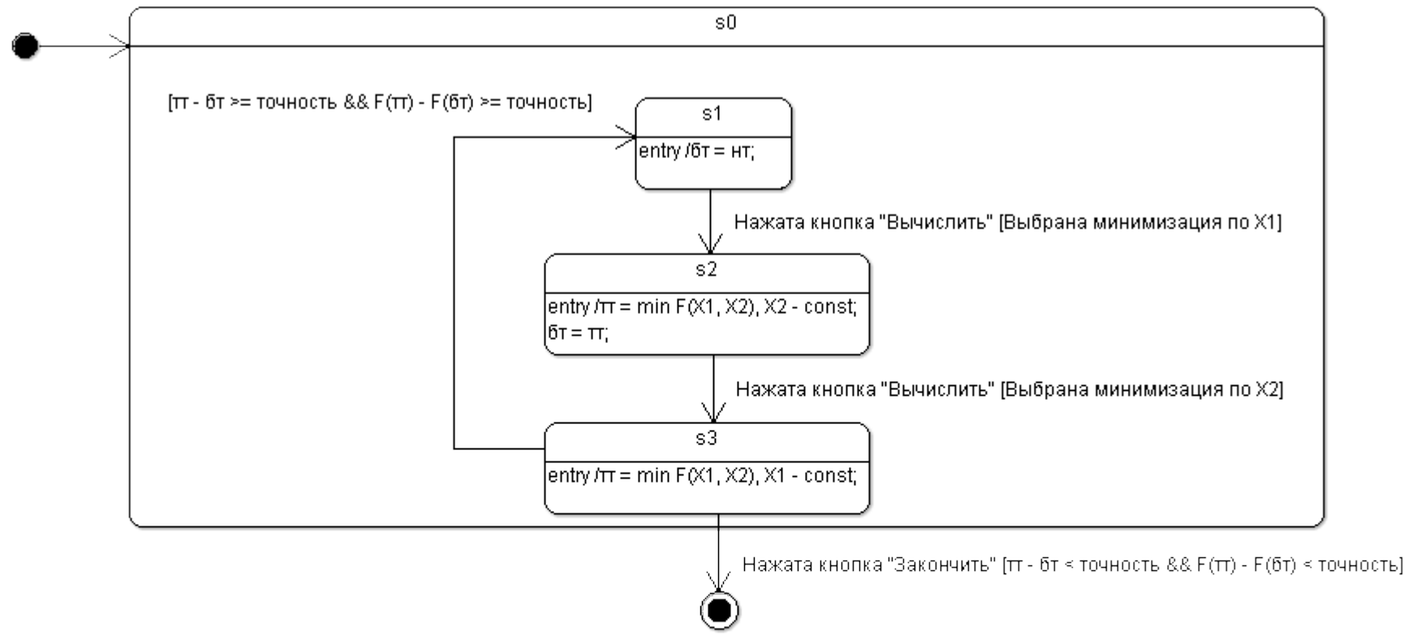

Рис. 15 Диаграмма состояний к методу покоординатного спуска с минимизацией по направлению

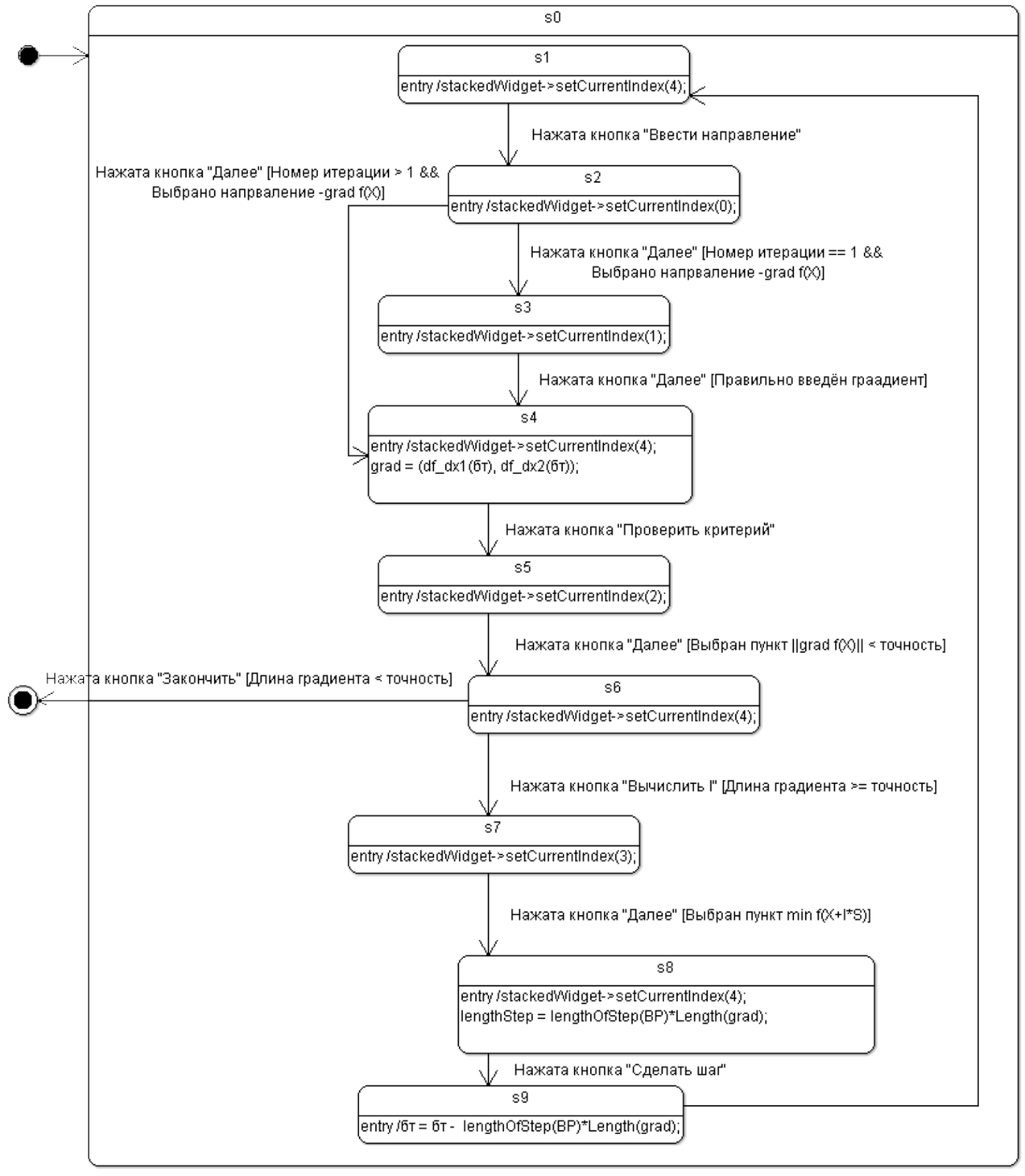

Рис. 16 Диаграмма состояний к методу наискорейшего спуска.

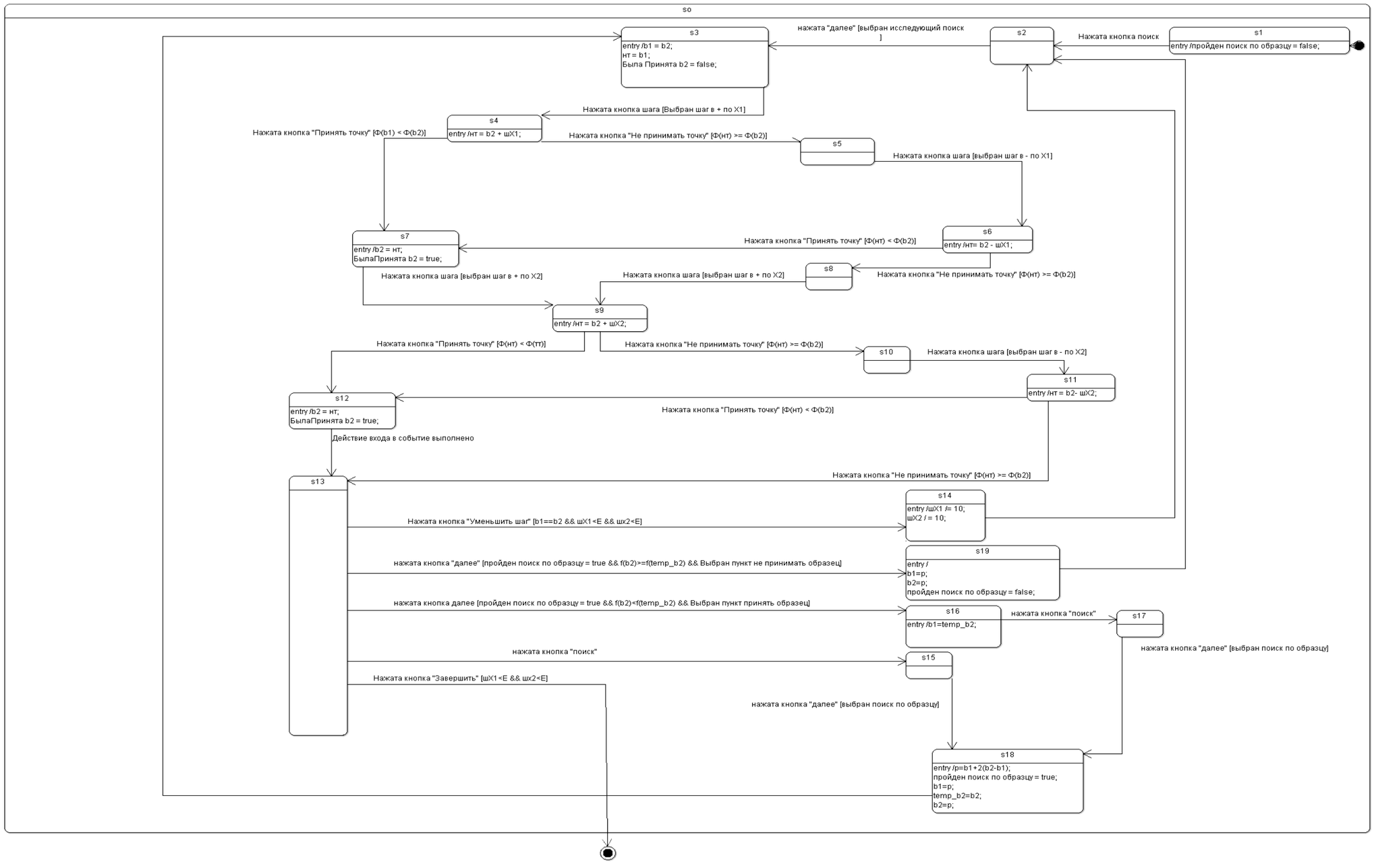

Рис. 17 Диаграмма состояний к методу Хука-Дживса

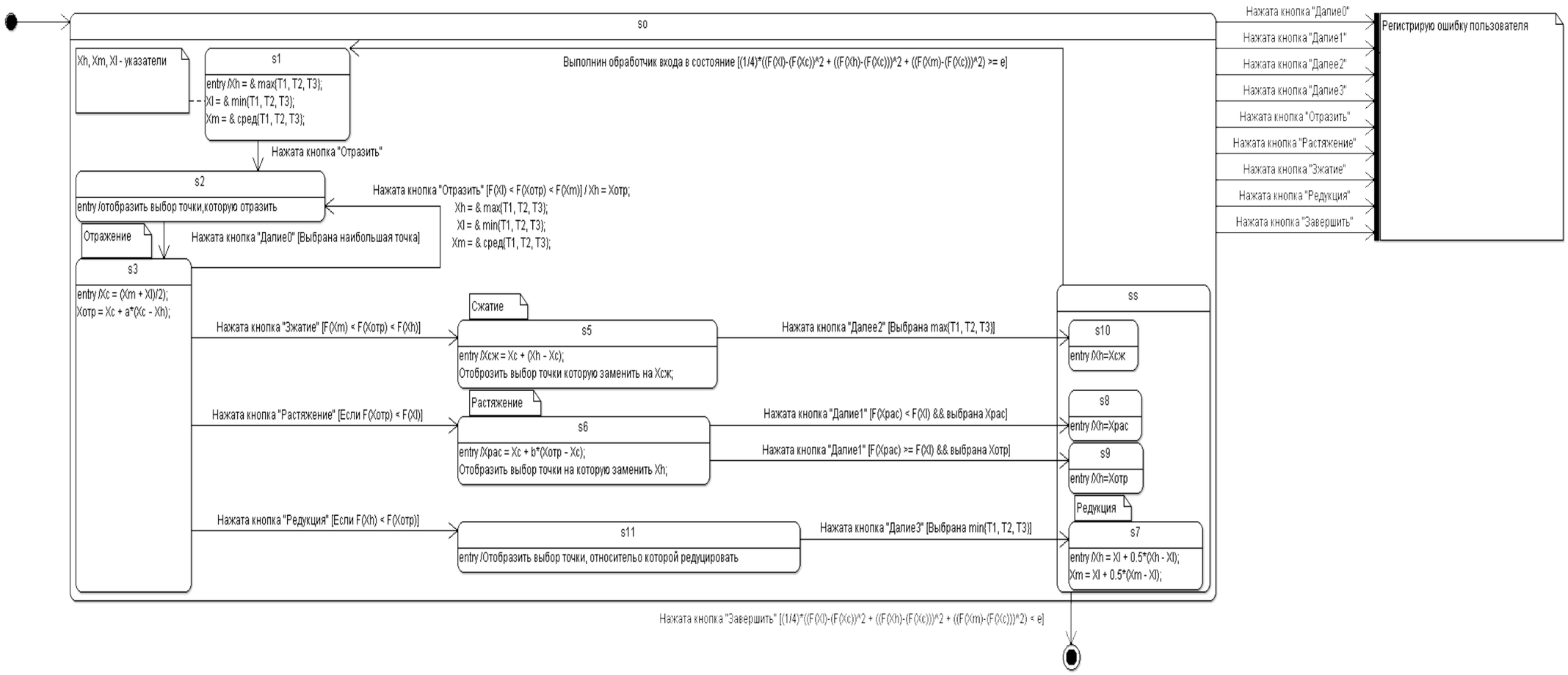

Рис. 18 Диаграмма состояний к методу Нелдера-Мида

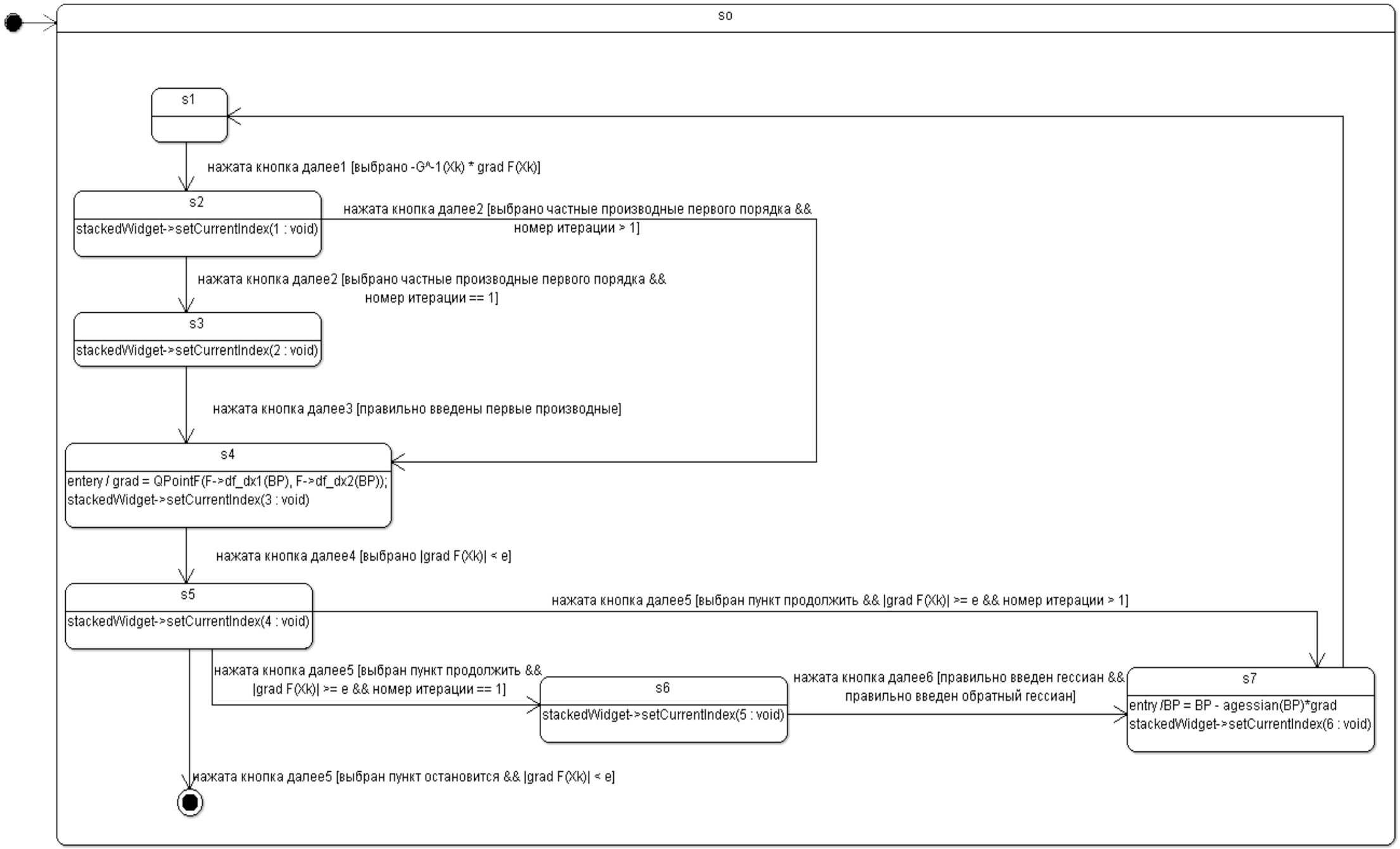

Рис. 19 Диаграмма состояний к методу Ньютона