

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ИНДУСТРИАЛЬНЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВПО «МГИУ»)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

по специальности 010503 «Математическое обеспечение и
администрирование информационных систем»

на тему «Разработка программного обеспечения для анализа эффекта
Холла в полупроводниках»

Группа		101151
Студент-дипломник	_____	А.А. Овсянников
Руководитель работы	_____	к.ф.-м.н., доцент И.М. Белова

		Допускается к защите
Заведующий кафедрой	_____	доцент, к.ф.-м.н Е.А. Роганов

Москва 2015

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ИНДУСТРИАЛЬНЫЙ
УНИВЕРСИТЕТ»
(ФГБОУ ВПО «МГИУ»)

Кафедра информационных систем и технологий

_____ / Роганов Е.А. /
«_____» _____ 2015г

ЗАДАНИЕ

на выпускную квалификационную работу по специальности 010503
«Математическое обеспечение и администрирование информационных
систем»

студента Овсянникова Александра Александровича группы 101151

1. Тема ВКР: Разработка программного обеспечения для анализа эффекта Холла в полупроводниках.
2. Сроки начала работы: 16.03.2015, защиты: 25.06.2015.
3. Руководитель ВКР: к.ф.-м.н., доцент И.М. Белова
4. Задание ВКР:
 - 4.1. Исходные данные: файлы формата .xls
 - 4.2. Цель работы: разработать программное обеспечение для анализа ЭДС Холла в полупроводниках, разработать программное обеспечение для хранения и обработки экспериментальных данных.
 - 4.3. Содержание работы:

1. Введение	4
2. Литературный обзор	6
3. Постановка задачи	18
4. Модель расчета электродвижущей силы Холла	19
5. Структура разработанного программного обеспечения . .	23
6. Программная реализация и обсуждение результатов . . .	29
7. Выводы	66
8. Приложение	68
 - 4.4. Используемые математические методы: метод последовательно верхней релаксации, численное вычисление производных, численное вычисление интегралов.

Студент-дипломник	_____	А.А.Овсянников
Руководитель работы	_____	к.ф.-м.н., доцент И.М. Белова

Аннотация

Ключевые слова: эффект Холла, уравнение Пуассона, метод последовательной верхней релаксации, ЭДС Холла, QT.

Работа посвящена разработке программного обеспечения для анализа эффекта Холла в полупроводниках. Рассматривается построение модели для анализа результатов проведенных экспериментов и базы данных для хранения этих результатов.

Содержание

1.	Введение	4
2.	Литературный обзор	6
3.	Постановка задачи	18
4.	Модель расчета электродвижущей силы Холла	19
5.	Структура разработанного программного обеспечения	23
6.	Программная реализация и обсуждение результатов	29
7.	Выводы	66
8.	Приложение	68

1. Введение

Полупроводники - это широкий класс веществ, в которых концентрация подвижных носителей заряда ниже, чем атомов. Их ширина запрещенной зоны порядка нескольких электронвольт. Преимущество таких веществ в том, что они, занимая промежуточное место между проводниками и диэлектриками, могут обладать свойствами и тех, и других. Так, например, вблизи абсолютного нуля проводимости у полупроводникового материала практически нет, тогда как при высоких температурах его свойство проводить электричество значительно повышается.

Благодаря наличию достаточно узкой запрещенной зоны, в полупроводниках может наблюдаться огромное количество эффектов, которых невозможно добиться ни в металлах, ни в диэлектриках. В следствие этого физические свойства полупроводниковых веществ изучены гораздо больше, чем проводниковых и диэлектрических. Одной из основных областей применения полупроводников - это производство полупроводниковых приборов и интегральных микросхем. В первую очередь это относится к кремнию, но затрагивает и другие соединения (Ge, GaAs, InP, InSb).

Проводимость полупроводника обеспечивается явлениями переноса. Это явления, обусловленные движением носителей заряда под действием электрического и магнитного полей, градиента температуры и градиента концентрации носителей заряда. К явлениям переноса относятся электропроводность, электронная теплопроводность, гальваномагнитные, термоэлектрические и термомагнитные эффекты.

Как известно, под воздействием электрического поля в полупроводнике возникает такое явление переноса, как электрический ток. Квазичастицы приходят в упорядоченное движение. При перемещении носителей заряда, электрическое поле совершает работу, называемую потенциалом. В этом случае, при помещении кристалла в поле постоянного магнита, возникает поперечная разность потенциалов, называемая холловским напряжением. Это явление было названо эффектом Холла, в честь американского физика Эдвина Герберта Холла, открывшем его в 1879 году. Свой эксперимент он проводил на золотой пластинке, размещенной на стекле, при пропускании через которую магнитного потока возникала разность потенциалов на боковых краях пластины. Разница потенциалов возникала вследствие приложения магнитного поля перпендикулярно к плоскости пластинки (холловского элемента). Отношение холловского напряжения к величине продольного тока, сегодня известное как «холловское сопротивление», характеризует материал, из которого изготовлен элемент Холла.

Эффект Холла имеет очень широкое применение. С его помощью производятся измерения линейных и угловых перемещений объектов, сильных токов и магнитных полей. Существуют датчики давления и температуры, механические переключатели, датчики скорости и системы зажигания автомобиля, ионные реактивные двигатели, созданные на его основе.

В ряде случаев, проводя эксперименты, основанных на этом явлении, необходимо проводить много гальваномагнитных измерений в различных магнитных полях. Очень часто такие эксперименты связаны со значительными техническими трудностями. Трудности эти связаны со способами получения магнитных полей. Слабые

поля легко получают при помощи всякого рода соленоидов, тогда как для создания сильного поля, как правило, прибегают к мощным электромагнитам, в которых индукция поля в зазоре сердечника регулируется значением силой тока, протекающем через обмотки. Значительное увеличение действия электрического поля приводит к остаточной намагниченности сердечника, что в последствии мешает уверенно работать в области малых полей. Также из-за увеличения силы тока нагреваются обмотки электромагнита, что приводит к уменьшению тока даже при стабильном источнике питания.

В ОАО «Гиредмет» была предпринята попытка создания установки, в которой изменение индукции магнитного поля осуществляется за счет вращения кристалла полупроводникового материала в криостате в поле постоянного магнита. Это устройство было названо «Цилиндр».

При помощи эффекта Холла определяют концентрацию носителей заряда в полупроводниковом материале. Для этого крайне необходимо программное обеспечение, позволяющее изучать изменение электродвижущей силы Холла, возникающее при изменении величины и формы контактов, а также хранить накопившиеся экспериментальные данные.

2. Литературный обзор

Эффект Холла в полупроводниках с несколькими типами носителей

В теории явлений переноса хорошо изучены ситуации, в которых электрохимический потенциал газа носителей заряда сравнительно медленно изменяется в пространстве. При этом, в случае если электрохимический потенциал также мало меняется и на расстояниях порядка характерной длины волны носителей заряда, то систему свободных электронов и других квазичастиц можно рассматривать как газ классически движущихся чакстиц. Для описания процессов в этой системе следует разобрать такое понятие, как функция распределения носителей заряда по квазиимпульсам и координатам. Средняя концентрация электронов проводимости в единице объема $dp \equiv dp_x dp_y dp_z$ выражается как

$$\frac{2}{(2\pi\hbar)^3} f(p, r) dp \quad (1)$$

Плотность потока и сила тока носителей определяется следующим выражением:

$$j = -\frac{2}{(2\pi\hbar)^3} \int v(p) f(p, r) dp \quad (2)$$

$$I = -\frac{2}{(2\pi\hbar)^3} \int v(p) [E(p) - e\varphi] f(p, r) dp \quad (3)$$

Правые части обоих уравнений не равны нулю только при наличии факта нарушения термодинамического равновесия. Это нарушение проявляется в изменении вида функции распределения. Нахождение величин в формулах (2) и (3) сводится к нахождению вида неравновесной функции распределения. Для этого существует универсальное уравнение, полученное Л.Больцманом. Оно учитывает как условия опыта, так и свойства материала.

Это уравнение представляет собой уравнение непрерывности в фазовом пространстве, в качестве координат которого выступают компоненты векторов p и r . Для того, чтобы вывести это уравнение следует в фазовом пространстве выделить элемент объема, например параллелепипед со сторонами $dx, dy, dz, dp_x, dp_y, dp_z$. В нем будет происходить изменения числа частиц за бесконечно малый промежуток времени dt . В момент времени t число частиц в параллелепипеде есть

$$dn = f(p, r, t) \frac{dpdr}{(2\pi\hbar)^3} \quad (4)$$

Спустя время dt это число равно

$$dn' = f(p, r, t + dt) \frac{dpdr}{(2\pi\hbar)^3} = \left\{ f(p, r, t) + \frac{\partial f(p, r, t)}{\partial t} dt \right\} \frac{dpdr}{(2\pi\hbar)^3} \quad (5)$$

Разница количества частиц соответственно равна:

$$dn' - dn = \frac{\partial f}{\partial t} \frac{dpdr dt}{(2\pi\hbar)^3} \quad (6)$$

Эта разность дает представление о том, сколько пришло в элемент объема частиц с данной проекцией спина. Для получения уравнения Больцмана нужно выразить левую часть уравнения (6) через функцию распределения. По скольку в этом уравнении учитываются конкретные условия опыта, необходимо указать процессы, приводящие к изменению числа частиц с данными координатами и квазиимпульсами.

Число частиц в элементе объема может изменяться под влиянием следующих явлений:

1. трансляции (перемещения в пространстве координат);
2. ускорение частиц (под действием каких-либо полей);
3. рассеяние (столкновения с диффектами кристаллической решетки);
4. рекомбинации и генерации носителей заряда.

Перемещение частиц в пространстве - это совокупность трех перемещений вдоль трех взаимно-перпендикулярных осей координат. В момент времени dt в левую грань элемента объема войдут частицы, находящиеся на расстоянии не более чем $v_x dt$ от этой грани и движущиеся направо. Их число выражается следующим выражением:

$$f(p, x, y, z, t) \frac{v_x dt dy dz dp}{(2\pi\hbar)^3} \quad (7)$$

За это же время через левую грань выходит число частиц, равное:

$$f(p, x + dx, y, z, t) \frac{v_x dt dy dz dp}{(2\pi\hbar)^3} \quad (8)$$

Если выражение (8) вычесть из выражения (7) (предварительно разложив его в ряд Тейлора по dx) можно получить:

$$- \frac{\partial f}{\partial x} v_x \frac{dt dr dp}{(2\pi\hbar)^3} \quad (9)$$

Выражение (9) является результирующим «приходом», связанным с движением частиц вдоль оси. Баланс частиц за счет перемещений по осям y и z вычисляется аналогичным образом. Следующее выражение характеризует изменение числа частиц с данной проекцией спина за время dt за счет перемещений их в пространстве координат:

$$(dn' - dn) = -(v, \nabla f) \frac{dt dr dp}{(2\pi\hbar)^3} \quad (10)$$

Теперь следует провести похожие вычисления для изменения числа частиц в рассматриваемом фазовом пространстве под действием электрического и магнитного полей. Силы, обусловленные этими полями, вызывают изменение квазиимпульса, или, другими словами, трансляцию в пространстве квазиимпульсов. В этом случае скорость $v = \frac{dp}{dt}$. Вместо градиента в пространстве координат следует использовать градиент в пространстве квазиимпульсов $\nabla_p f$.

$$(dn' - dn) = - \left(\frac{dp}{dt}, \nabla f \right) \frac{dt dr dp}{(2\pi\hbar)^3} \quad (11)$$

Согласно уравнениям движения:

$$\frac{dp}{dt} = F = -eE - \frac{e}{c}[v \times B] \quad (12)$$

Под E и B здесь следует понимать напряженность электрического и индукцию магнитного полей.

Рассеяние частиц может происходить по разным причинам. Например, при наличии рассеивателей - диффектов кристаллической решетки. Рассеяние - это изменение состояния носителя заряда в результате взаимодействия с прассеивателями; состояние характеризуется квазиимпульсами и проекцией спина. Пусть квазиимпульс частицы до столкновения равен p' , а после столкновения p . Тогда:

$$P_1(p', p)f(p', r)[1 - f(p, r)] \frac{dpdp' dr}{(2\pi\hbar)^3} \quad (13)$$

P_1 - коэффициент пропорциональности, который может зависеть и от r . Проинтегрировав это выражение по p' и умножая на dt , можно получить полное число носителей заряда, проходящих за время dt в рассматриваемый элемент объема за счет столкновений:

$$\frac{drdpdt}{(2\pi\hbar)^3} \int P_1(p', r)f(p', p)[1 - f(p, r)] \frac{dp'}{(2\pi\hbar)^3} = A \frac{dtdr dp}{(2\pi\hbar)^3} \quad (14)$$

Интеграл A описывает «приход» частиц в единичный элемент фазового объема в единицу времени.

Для «ухода» соответственно имеется интеграл B :

$$\frac{drdpdt}{(2\pi\hbar)^3} \int P_2(p, p')f(p, r)[1 - f(p', r)] \frac{dp'}{(2\pi\hbar)^3} = B \frac{dtdr dp}{(2\pi\hbar)^3} \quad (15)$$

Комбинируя эти 2 выражения можно получить полное изменение частиц в результате рассеяний:

$$(dn' - dn) == J[f] \frac{dtdr dp}{(2\pi\hbar)^3} \quad (16)$$

Где:

$$J \equiv A - B = \frac{1}{(2\pi\hbar)^3} \int dp' \left\{ P_1(p', p)f(p', r)[1 - f(p, r)] - P_2(p, p')f(p, r)[1 - f(p', r)] \right\} \quad (17)$$

При рассеянии частиц друг на друге формула (17) остается в силе, но изменяется интеграл рассеяния, так как в этом случае меняется состояние не одной частицы, а двух. Пусть квазиимпульсы частиц до столкновения равны p'_1 и p'_2 , а после столкновения p_1 и p_2 . Число таких столкновений в единицу времени равно:

$$P_1(p'_1, p'_2; p_1, p_2)f(p'_1, r) \frac{dp'_2}{(2\pi\hbar)^3} f(p'_2, r) \frac{dp'_2}{(2\pi\hbar)^3} \times [1 - f(p_1, r)] \frac{dp_1}{(2\pi\hbar)^3} [1 - f(p_2, r)] \frac{dp_2}{(2\pi\hbar)^3} \quad (18)$$

Проинтегрировав это выражение по p'_1, p_2 и p'_2 можно получить полное число столкновений, связанных с приходом носителя заряда в рассматриваемый элемент фазового объема.

При вычислении кинетических коэффициентов можно вообще не принимать во внимание рекомбинационные явления. Рекомбинация и захват становятся заметными лишь спустя длительное время после того, как процессы рассеяния сформируют функцию распределения носителей по квазиимпульсам.

Если комбинировать формулы (4), (10), (11) и (16), можно получить уравнения для функции распределения:

$$\frac{\partial f}{\partial t} = -(v, \nabla f) - (F, \nabla_p f) + j[f] \quad (19)$$

Выражение (19) называют кинетическим уравнением, или уравнением Больцмана. Интеграл столкновений $J[f]$ задается формулами (17) или (18) (или их суммой, если существует оба вида рассеяния), а сила F - выражением (12).

Уравнение (19) написано для полупроводника с одним типом носителей. Если носители несколько (электроны, легкие и тяжелые дырки), необходимо ввести свою функцию распределения для каждого типа. Следовательно, для полупроводника с несколькими типами носителей будет несколько кинетических уравнений, при этом в правой части каждого из них появится сумма по всем типам частиц, на которых рассматриваемый носитель может рассеиваться. [2]

Если допустить, что в полупроводнике ток переносится двумя типами носителей (электроны и дырки) и предположить, что вектор напряженности электрического поля E находится в плоскости плупроводника, то эффект Холла можно изобразить так, как показано на рисунке ниже. Здесь и далее под слабым магнитным полем понимается случай, в котором длина свободного пробега квазичастицы гораздо больше, чем радиус траектории ее движения ($l \ll r$). Подвижности и концентрации электронов и дырок равны соответственно n, μ_n, p, μ_p . Векторы индукции и магнитного поля обозначены как B и J .

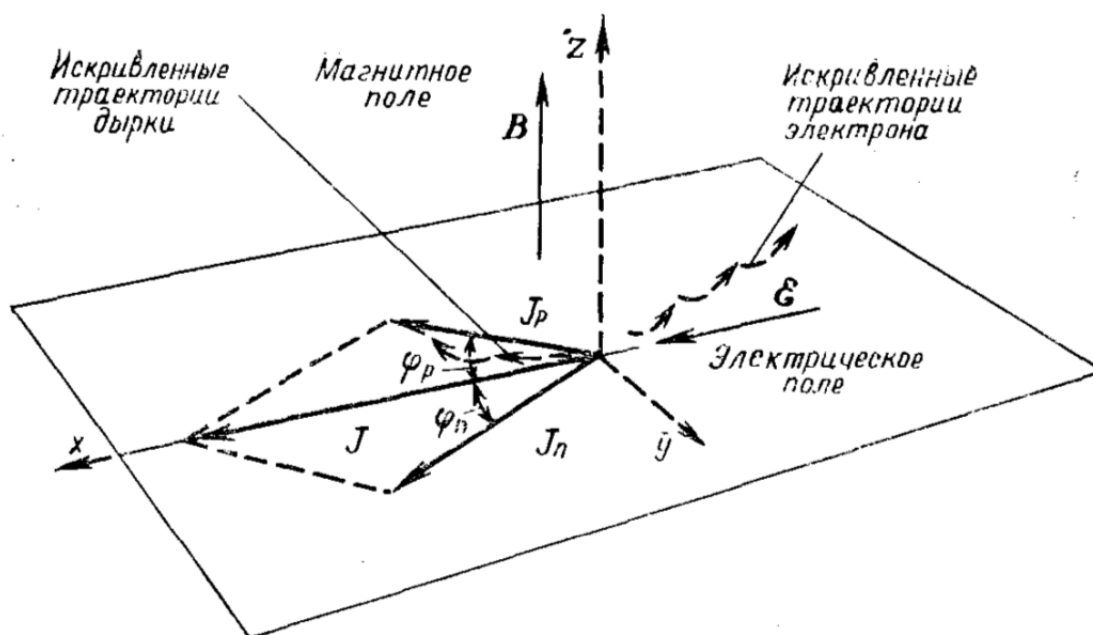


Рис. 1. Эффект Холла при наличии двух типов носителей заряда

Под действием электрического поля заряды начинают ускоряться. По мере увеличения дрейфовой скорости на квазичастицы начинает действовать отклоняющая сила лоренца. Траектория движения носителя заряда искривляется до тех пор, пока он не наткнется на дефект кристаллической решетки. Далее, передав ей часть своей энергии, он начинается двигаться по новой траектории под действием электрического поля. Таким образом, как видно на рисунке, траектории носителя образуют серию дуг. Векторы плотности тока J_n (электронов) и J_p (дырок) будут отклоняться относительно вектора напряженности электрического поля на углы φ_n (электроны) и φ_p (дырки).

Если, например, неравновесная функция равна:

$$f_n(k) = f_0 + f_1^n(k) = f_0 - \frac{\partial f_0}{\partial E} \chi_n(k) \quad (20)$$

То плотность тока можно записать как:

$$J = -\frac{e}{4\pi^3} \int_V v_n f_1^n(k) d\tau_k + \frac{e}{4\pi^3} \int_V v_p f_1^p(k') d\tau'_k = \frac{e\hbar}{4\pi^3 m_n^*} \quad (21)$$

$$\int_V \frac{\partial f_0}{\partial E} \chi_n(k) k d\tau_k - \frac{e\hbar}{4\pi^3 m_p^*} \int_V \frac{\partial f_0}{\partial E} \chi_n(k') k' d\tau'_k \quad (22)$$

В случае однородного полупроводника со сферическими изоэнергетическими поверхностями при отсутствии магнитного поля и градиента температуры можно сказать, что:

$$\chi_n = \frac{e\hbar\tau_e}{m_n^*} \left\{ E + \frac{e\tau_e}{m_n^*} [BE] \right\} \quad (23)$$

$$\chi_p = \frac{e\hbar\tau_h}{m_p^*} \left\{ E - \frac{e\tau_h}{m_p^*} [BE] \right\} \quad (24)$$

Тогда электронная составляющая плотности тока равна:

$$J_n = -\frac{e^2\hbar^2}{4\pi^3 m_n^{*2}} \int_V \frac{\partial f_0}{\partial E} \left\{ \tau_e(kE) + \frac{e\tau_e}{m_n^*} [BE] \right\} k d\tau_k \quad (25)$$

$$J_n = \frac{e^2 n}{m_n^*} E \frac{4}{3\sqrt{\pi}} \int_0^\infty \tau_e e^{-\alpha} \alpha^{\frac{3}{2}} d\alpha + \frac{e^2 n}{m_n^*} [EB] \frac{4}{3\sqrt{\pi}} \int_0^\infty \tau_e e^{-\alpha} \alpha^{\frac{3}{2}} d\alpha \quad (26)$$

Вводя среднее значение времени релаксации для электронов:

$$\langle \tau_e \rangle = \frac{4}{3\sqrt{\pi}} \int_0^\infty \tau_e e^{-\alpha} \alpha^{\frac{3}{2}} d\alpha \quad (27)$$

и среднее значение квадрата времени релаксации:

$$\langle \tau_e^2 \rangle = \frac{4}{3\sqrt{\pi}} \int_0^\infty \tau_e^2 e^{-\alpha} \alpha^{\frac{3}{2}} d\alpha \quad (28)$$

а также учитывая, что $\mu_n = \frac{e\langle \tau_e \rangle}{m_n^*}$ можно для электронной составляющей плотности тока получить:

$$J_n = en\mu_n E + \frac{\langle \tau_e^2 \rangle}{|\langle \tau_e \rangle|^2} en\mu_n^2 [BE] = en\mu_n E + r_n en\mu_n^2 [BE] \quad (29)$$

где

$$r_n = \frac{\langle \tau_e^2 \rangle}{|\langle \tau_e \rangle|^2} \quad (30)$$

Дырочная составляющая плотности тока запишется аналогично:

$$J_p = en\mu_p E - r_n en\mu_p^2 [BE] \quad (31)$$

где

$$r_p = \frac{\langle \tau_h^2 \rangle}{|\langle \tau_h \rangle|^2} \quad (32)$$

Полная плотность тока равна:

$$J = e(n\mu_n + p\mu_p)E - r_n e\left(\frac{r_p}{r_n} p\mu_p^2 - n\mu_n^2\right) [BE] \quad (33)$$

В случае упругого рассеяния

$$\frac{r_p}{r_n} = \frac{\langle \tau_h^2 \rangle}{|\langle \tau_h \rangle|^2} \frac{|\langle \tau_e \rangle|^2}{\langle \tau_e^2 \rangle} = 1 \quad (34)$$

Обозначая $r_n = r$ плотности тока в полупроводнике с двумя типами носителей равна:

$$J = e(n\mu_n + p\mu_p)E - re(p\mu_p^2 - n\mu_n^2) [BE] \quad (35)$$

Тогда, в случае слабого магнитного поля для напряженности Холла можно записать выражение:

$$\xi = \frac{r}{e} \frac{p\mu_p^2 - n\mu_n^2}{(p\mu_p + n\mu_n)^2} JB = RJB \quad (36)$$

[1]

Использование эффекта Холла в гальваномагнитных измерениях

Для ряда практических приложений необходимо проводить гальваномагнитные измерения образцов полупроводниковых материалов в различных магнитных полях, перекрывая возможно более широкий интервал значений индукции магнитного поля. Такие измерения позволяют различить между собой образцы КРТ п- и р- типов. Анализ полевой зависимости коэффициента Холла позволяет получить информацию о значениях концентраций и подвижностей свободных носителей заряда.

Реализация гальваномагнитных измерений в широком интервале магнитных полей связана со значительными техническими трудностями. Малые поля легко получить используя соленоиды, а сильные поля - при помощи электромагнитов, в которых индукция магнитного поля регулируется увеличением силы тока, текущего сквозь обмотки электромагнита, нагревая их. При этом из-за остаточной намагниченности сердечника не удастся уверенно работать в области малых полей.

В ОАО Гиредмет была создана установка, лишенная этих недостатков. В ней индукция магнитного поля регулируется с помощью поворота полупроводникового материала в кристате. Ее рисунок изображен ниже. Дьюар из нержавеющей стали (1) укрепляется на поддоне (2), который может поворачиваться с помощью шагового двигателя (3), смонтированного на пластине (4). Образец (5) монтируется на медной вставке (6), которая с помощью специального устройства фиксируется на опорный подпятник (7) на дне дьюара. Сверху дьюар закрывается заглушкой (8), имеющей отверстие для заливки в дьюар жидкого азота. Образец (5) на вставке опускается в трубу (9) и фиксируется относительно дьюара, но может вместе с дьюаром поворачиваться вокруг вертикальной оси. Образец (5) помещается между полюсами постоянного магнита (10), снабженного съемными наконечниками (11), сверху постоянный магнит закрывается крышкой (12). Трубка (9) жестко соединена с диском (13), имеющим деления для контроля угла поворота образца (вместе с дьюаром) относительно нулевого положения. Сверху вся конструкция закрывается разъемом (14), через который подводятся контактные провода к образцу. В корпусе (15) просверлено отверстие (16) для вывода проводов с шагового двигателя. К днищу корпуса (17) прикреплены 3 ножки (18). При низких температурных измерениях образец обдувается парами жидкого азота, выходящего по специальной канавке вдоль вставки (6) и затем - из-под нижней части разъема - в атмосферу. Температура образца контролируется с помощью термопары «медьконстант», спай которой укреплен в непосредственной близости от образца.

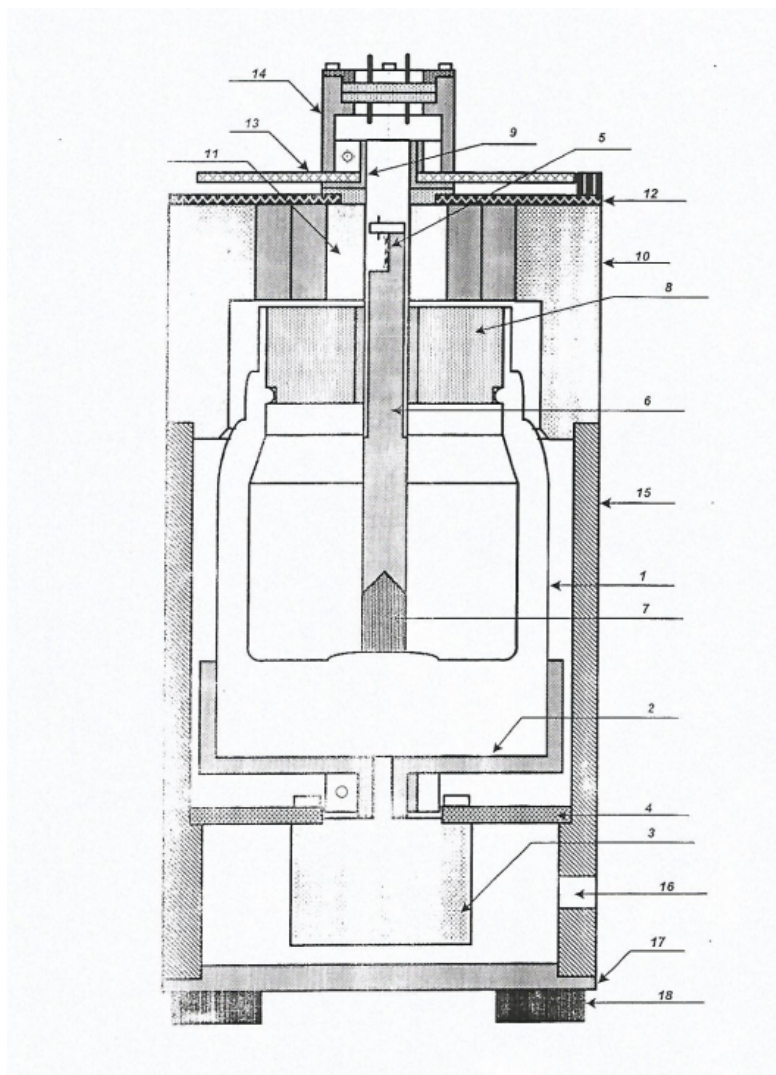


Рис. 2. Схема установки «Цилиндр»

Датчик начального положения (3) служит для фиксации начала отсчета угла поворота образца (1) в поле постоянного магнита (2). Поворот осуществляется с помощью блока управления и индикации (4), изготовленного на базе микроконтроллера ATMEGA U8-20P4, который управляет шаговым двигателем ДШИ-200-2-1 (5). Одно нажатие кнопки контроллера вызывает поворот образца на угол 1.8° (в ту или другую сторону). Сигнал с термопары измеряется цифровым вольтметром В7-13 (7) и коммутируется с помощью коммутатора (8). Измеряемый сигнал фиксируется цифровым вольтметром В7-65/5 (9) через интерфейс RS232 (10) поступает на персональный компьютер (11), где фиксируется и обрабатывается с помощью специальной программы.

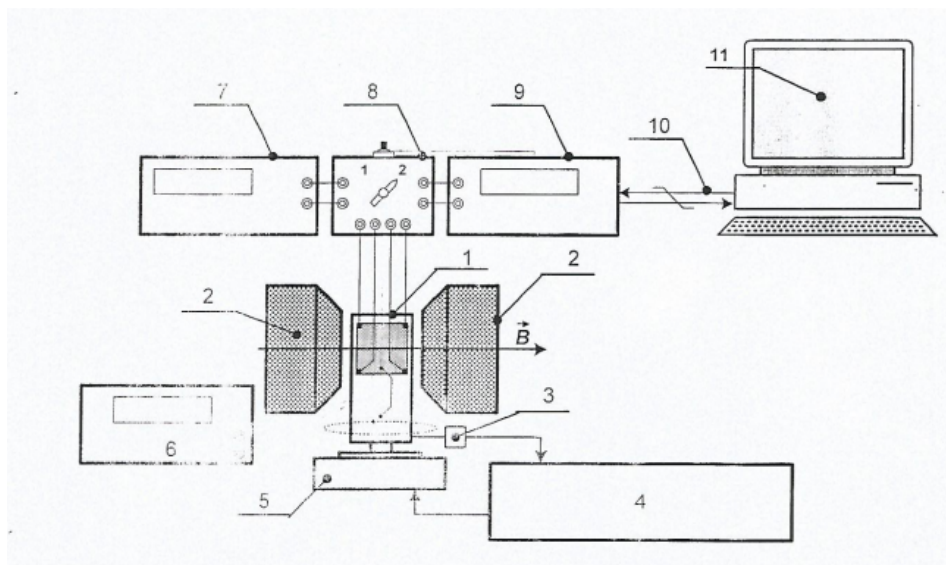


Рис. 3. Блок-схема установки «Цилиндр»

Для практического применения результатов подобных исследований крайне необходимо изучить зависимость электродвижущей силы Холла от величины и размера контактов. Исследования, проводимые в ОАО «Гиредмет», проводятся для большого количества образцов полупроводниковых материалов. Большое количество экспериментальных данных необходимо хранить. В данной работе рассматривалось построение базы данных для хранения подобной информации, интерфейс для ее использования и модуль для графического отображения вычислений, а также было разработано программное обеспечение, необходимое для исследования зависимости электродвижущей силы Холла от контактов.

Выбор программного обеспечения для реализации решения описанных выше проблем

После проведения испытаний на установке Цилиндр, в ОАО Гиредмет были получены экспериментальные данные. Для хранения этих данных необходимо составить базу. Для построения базы данных для хранения экспериментальной информации было решено использовать систему управления базой данных PostgreSQL.

Дипломный проект было решено реализовывать на языках C и C++. C++ — компилируемый статически типизированный язык программирования общего назначения. Поддерживает такие парадигмы программирования как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование, обеспечивает модульность, отдельную компиляцию, обработку исключений, абстракцию данных, объявление типов (классов) объектов, виртуальные функции. Стандартная библиотека включает, в том числе, общеупотребительные контейнеры и алгоритмы. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков. В сравнении с его предшественником — языком C — наибольшее внимание уделено поддержке объектно-ориентированного и обобщённого программирования.

C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (игр). Существует множество реализаций языка C++, как бесплатных, так и коммерческих и для различных платформ. Например, на платформе x86 это GCC, Visual C++, Intel C++ Compiler, Embarcadero (Borland) C++ Builder и другие. Синтаксис C++ унаследован от языка C. Одним из принципов разработки было сохранение совместимости с C. Тем не менее, C++ не является в строгом смысле надмножеством C; множество программ, которые могут одинаково успешно транслироваться как компиляторами C, так и компиляторами C++, довольно велико, но не включает все возможные программы на C. [19]

Для построения пользовательского интерфейса для работы с выполненным проектом было решено использовать Qt. Qt - кроссплатформенный инструментальный разработки ПО на языке программирования C++. Есть также «привязки» ко многим другим языкам программирования: Python — PyQt, PySide; Ruby — QtRuby; Java — Qt Jambi; PHP — PHP-Qt и другие. Со времени своего появления в 1996 году библиотека Qt легла в основу тысяч успешных проектов во всём мире. Кроме того, Qt является фундаментом популярной рабочей среды KDE, входящей в состав многих дистрибутивов Linux. Qt позволяет запускать написанное с его помощью ПО в большинстве современных операционных систем путём простой компиляции программы для каждой ОС без изменения исходного кода. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Qt является полностью объектно-ориентированным, легко расширяемым и поддерживающим технику компонентного программирования. Отличительная особенность Qt от других библиотек — использование Meta Object Compiler (MOC) — предварительной системы обработ-

ки исходного кода (в общем-то, Qt — это библиотека не для чистого C++, а для его особого наречия, с которого и «переводит» МОС для последующей компиляции любым стандартным C++ компилятором). МОС позволяет во много раз увеличить мощь библиотек, вводя такие понятия, как слоты и сигналы. Кроме того, это позволяет сделать код более лаконичным. Утилита МОС ищет в заголовочных файлах на C++ описания классов, содержащие макрос `Q_OBJECT`, и создаёт дополнительный исходный файл на C++, содержащий метаобъектный код. Qt позволяет создавать собственные плагины и размещать их непосредственно в панели визуального редактора. Также существует возможность расширения привычной функциональности виджетов, связанной с размещением их на экране, отображением, перерисовкой при изменении размеров окна. Qt комплектуется визуальной средой разработки графического интерфейса «Qt Designer», позволяющей создавать диалоги и формы в режиме WYSIWYG. В поставке Qt есть «Qt Linguist» — графическая утилита, позволяющая упростить локализацию и перевод программы на многие языки; и «Qt Assistant» — справочная система Qt, упрощающая работу с документацией по библиотеке, а также позволяющая создавать кросс-платформенную справку для разрабатываемого на основе Qt ПО. Начиная с версии 4.5.0 в комплект Qt включена среда разработки «Qt Creator», которая включает в себя редактор кода, справку, графические средства «Qt Designer» и возможность отладки приложений. «Qt Creator» может использовать GCC или Microsoft VC++ в качестве компилятора и GDB в качестве отладчика. Для Windows версий библиотека комплектуется компилятором, заголовочными и объектными файлами MinGW. Существуют версии библиотеки для Microsoft Windows, систем класса UNIX с графической подсистемой X11, Android, iOS, Mac OS X, Microsoft Windows CE, QNX, встраиваемых Linux-систем и платформы S60. Идет портирование на Windows Phone и Windows RT. Также идет портирование на Haiku и Tizen. До недавнего времени библиотека Qt также распространялась ещё в одной версии: Qt/Embedded. Теперь эта платформа переименована в Qtoria Core и распространяется как отдельный продукт. Qtoria Core обеспечивает базовую функциональность для всей линейки платформ, предназначенных для разработки приложений для встраиваемых и мобильных устройств (КПК, смартфонов и т. п.). [13]

Для визуализации результатов работы проекта были выбраны OpenGL и Gnuplot. OpenGL (Open Graphics Library) — спецификация, определяющая независимый от языка программирования платформонезависимый программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику. Включает более 300 функций для рисования сложных трёхмерных сцен из простых примитивов. Используется при создании компьютерных игр, САПР, виртуальной реальности, визуализации в научных исследованиях. На платформе Windows конкурирует с Direct3D. На базовом уровне, OpenGL — это просто спецификация, то есть документ, описывающий набор функций и их точное поведение. Производители оборудования на основе этой спецификации создают реализации — библиотеки функций, соответствующих набору функций спецификации. Реализация призвана эффективно использовать возможности оборудования. Если аппаратура не позволяет реализовать какую-либо возможность, она должна быть эмулирована программно. Производители должны пройти специфические тесты (conformance tests — тесты на соответствие) прежде чем реализация будет классифицирована как OpenGL-реализация. Таким образом, разработчикам программного обеспечения достаточно

научиться использовать функции, описанные в спецификации, оставив эффективную реализацию последних разработчикам аппаратного обеспечения. Эффективные реализации OpenGL существуют для Windows, Unix-платформ, PlayStation 3 и Mac OS. Эти реализации обычно предоставляются изготовителями видеоадаптеров и активно используют возможности последних. Существуют также открытые реализации спецификации OpenGL, одной из которых является библиотека Mesa. Из лицензионных соображений Mesa является «неофициальной» реализацией OpenGL, хотя полностью с ней совместима на уровне кода и поддерживает как программную эмуляцию, так и аппаратное ускорение при наличии соответствующих драйверов. Спецификация OpenGL пересматривается консорциумом ARB (Architecture Review Board), который был сформирован в 1992 году. Консорциум состоит из компаний, заинтересованных в создании широко распространённого и доступного API. Согласно официальному сайту OpenGL, членами ARB с решающим голосом на ноябрь 2004 года являются производители профессиональных графических аппаратных средств SGI, 3Dlabs, Matrox, производители потребительских графических аппаратных средств ATI и NVIDIA, производитель процессоров Intel, и изготовители компьютеров и компьютерного оборудования IBM, Apple, Dell, Hewlett-Packard и Sun Microsystems, а также один из лидеров компьютерной игровой индустрии id Software. Microsoft, один из основоположников консорциума, покинула его в марте 2003 года. Помимо постоянных членов, каждый год приглашается большое количество других компаний, становящихся частью OpenGL ARB в течение одного года. Такое большое число компаний, вовлеченных в разнообразный круг интересов, позволило OpenGL стать прикладным интерфейсом широкого назначения с большим количеством возможностей.

gnuplot — свободная программа для создания двух- и трёхмерных графиков. gnuplot имеет собственную систему команд, может работать интерактивно (в режиме командной строки) и выполнять скрипты, читаемые из файлов. Также используется в качестве системы вывода изображений в различных математических пакетах: GNU Octave, Maxima, Reduce и других. gnuplot выводит графики как непосредственно на экран (интерактивный режим), так и в файлы различных графических форматов (командный режим работы), таких как PNG, EPS, SVG, JPEG и множество других. Программа также может генерировать код на LaTeX, позволяя использовать шрифты и формулы LaTeX. [9]

3. Постановка задачи

Данная дипломная работа включает в себя две задачи:

1. Разработка программного обеспечения для анализа ЭДС Холла в полупроводниках;
2. Разработка базы для хранения экспериментальных данных. Необходимо составить базу для хранения информации, интерфейс для работы с ней и модуль для графического отображения результатов.

4. Модель расчета электродвижущей силы Холла

В процессе переноса частиц из одной точки в другую электрическое поле совершает работу. Эту работу называют потенциалом. Для построения модели требуется рассчитать векторное поле, состоящее из векторов электрического поля. Каждый вектор этого поля можно выразить через потенциал. Тогда:

$$\vec{E} = -grad\varphi' \quad (37)$$

Здесь:

- φ - это потенциал электрического поля.

Поскольку вектор электрического поля лежит в плоскости полупроводника, его компонента по оси z равна нулю. Поэтому, задачу вычисления векторов электрического поля можно считать двумерной. В этом случае, вектор электрического поля можно записать в виде следующих скалярных выражений:

$$E_x = -\frac{\partial\varphi}{\partial x} \quad E_y = -\frac{\partial\varphi}{\partial y} \quad (38)$$

Для отображения векторного поля на скалярное, существует дифференциальный оператор, называемый дивергенцией:

$$div \vec{E} = \frac{\partial E_x}{\partial x} + \frac{\partial E_y}{\partial y} = \frac{\rho}{\epsilon_0} \quad (39)$$

Если подставить выражение (37) в выражение (39), то можно получить следующее уравнение:

$$\frac{\partial^2\varphi}{\partial x^2} + \frac{\partial^2\varphi}{\partial y^2} = -\frac{\rho}{\epsilon_0} \quad (40)$$

Уравнение (40) называют уравнением Пуассона. Это дифференциальное уравнение в частных производных эллиптического типа, которое описывает электрическое поле. Здесь

- φ - это потенциал электрического поля.
- ρ - величина, описывающая положение заряда в данной точке

Исходя из всего вышесказанного, для расчета электрического поля, необходимо, путем решения уравнения Пуассона, найти потенциал в каждой точке поля и вычислить его градиент.

Для численного решения Уравнения Пуассона и нахождения значений потенциала на узлах двумерной сетки использовался метод последовательной верхней релаксации. Для этого двумерную прямоугольную область покрывалась двумерной сеткой узлов, с шагом, равным отношению максимального значения соответствующей координаты к числу узлов сетки:

$$dx = \frac{x_{max}}{n} \quad dy = \frac{y_{max}}{n} \quad (41)$$

Согласно основным идеям метода сеток, для дискретизации обеих пространственных производных в уравнении (40) следует использовать по три соседних узла вдоль каждой из координат. Поэтому уравнение Пуассона может быть записано в разностной форме при помощи шаблона типа "крест" (рис. 4).

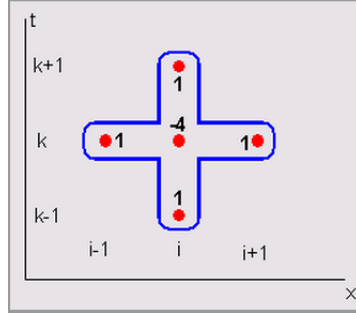


Рис. 4. Схема «Крест»

Уравнение Пуассона в конечно-разностном виде:

$$\frac{\varphi_{i+1,j} + \varphi_{i-1,j} - \varphi_{i,j}}{h^2} + \frac{\varphi_{i,j+1} + \varphi_{i,j-1} - \varphi_{i,j}}{h^2} = -\rho_{i,j} \quad (42)$$

После приведения подобных слагаемых в разностных уравнениях коэффициенты разностной схемы будут такими, как показано возле узлов шаблона на рисунке 4:

$$a\varphi_{i+1,j} + b\varphi_{i-1,j} + c\varphi_{i,j+1} + d\varphi_{i,j-1} + e\varphi_{i,j} = \rho_{i,j} \quad (43)$$

$$a = b = c = d = 1, e = -4. \quad (44)$$

В методе последовательной верхней релаксации узел сетки вычисляется в 2 этапа. На первом этапе вычисляется невязка:

$$\xi_{j,l} = a_{i,j}\varphi_{j+1,l} + b_{j,l}\varphi_{j-1,l} + c_{j,l}\varphi_{j,l+1} + d_{j,l}\varphi_{j,l-1} + e_{j,l}\varphi_{j,l} - dx * dy * f_{j,l} \quad (45)$$

На втором этапе вычисляется средневзвешенное значение узла:

$$\varphi_{j,l}^{new} = \varphi_{j,l}^{old} - \omega \frac{\xi_{j,l}}{e_{j,l}} \quad (46)$$

В данной работе будет применяться метод последовательной верхней релаксации с ускорением Чебышева. Поэтому будет выполнять динамическое вычисления параметра релаксации

$$\omega^0 = 1 \quad (47)$$

$$\omega^{\frac{1}{2}} = \frac{1}{1 - \frac{\rho_{jacob}^2}{2}} \quad (48)$$

$$\omega^{\frac{n+1}{2}} = \frac{1}{1 - \frac{\omega^n \rho_{jacob}^2}{2}} \quad (49)$$

$$n = \frac{1}{2}, 1, \dots \infty \quad (50)$$

$$\omega^{\infty} \rightarrow \omega_{optimal} \quad (51)$$

ρ_{jacob_i} - это спектральный радиус. В данной работе он равен

$$\rho_{jacob_i} = \cos\left(\frac{2\pi}{n}\right) \quad (52)$$

Из формулы (45) следует, что этот метод не вычисляет узлы на границах сетки. Поэтому значения этих узлов нужно задать при помощи граничных условий. В данной работе краевая задача смешанная - по всем четырем границам задается нормальная производная потенциала (граничные условия Неймана), а в левом нижнем и правом верхнем углах задаются квадраты 4x4 с заранее известными значениями потенциала (граничные условия Дирихле). Поэтому по всем четырем границам значения узлов равны значениям на предграничных узлах, а в левом нижнем и правом верхнем углах сетки есть квадраты или уголки с определенным размером со значениями потенциала, заданными при помощи пользовательского интерфейса, который будет описан позже. Условием сходимости метода будет служить значением нормы невязки, равное или большее заданного числа. [3]

Получив значения потенциала в каждом узле двумерной сетки, необходимо вычислить две компоненты вектора градиента потенциала (по оси x и по оси y). Для этого для каждого значения потенциала численно вычисляется производная по четырем точкам:

$$\frac{\partial \varphi}{\partial x} = \frac{-\varphi(x_0 + 2dx) + 8\varphi(x_0 + dx) - 8\varphi(x_0 - dx) + \varphi(x_0 - 2dx)}{12dx} \quad (53)$$

$$\frac{\partial \varphi}{\partial y} = \frac{-\varphi(y_0 + 2dy) + 8\varphi(y_0 + dy) - 8\varphi(y_0 - dy) + \varphi(y_0 - 2dy)}{12dy} \quad (54)$$

Компоненты вектор напряжения электрического поля E - это взятые со знаком минус компоненты вектора градиента потенциала.

Вычислив векторное поле, необходимо построить векторное поле, состоящее из векторов плотности тока. Вектор плотности тока вычисляется по формуле:

$$J = e(n\mu_n + p\mu_p)E - re(p\mu_p^2 - n\mu_n^2)[BE] \quad (55)$$

где

- e - заряд электрона, равный $1.6 * 10^{-19}$
- n - концентрация электронов
- μ_n - подвижность электронов
- p - концентрация дырок
- μ_p - подвижность дырок

В правой части формулы для вычисления плотности тока есть векторное произведение векторов индукции магнитного поля и электрической напряженности. [1] Вектор магнитной индукции направлен под углом α к плоскости полупроводника. Его компонента B_z будет вычислять по формуле $B_z = B * \sin(\alpha)$. Получив необходимые векторы для каждого узла сетки, необходимо вычислить их векторное произведение. Их векторное произведение вычисляется следующим образом:

$$[BE] = \begin{vmatrix} i & j & k \\ E_x & E_y & 0 \\ B_x & B_y & B_z \end{vmatrix} = i(E_y * B_z) - j(E_x * B_z) + k(E_x * B_y - E_y * B_x) \quad (56)$$

При замере ЭДС Холла во время экспериментов с установкой Цилиндр использовалась схема Ван-Дер_Пау, в следствие чего третье слагаемое равняется 0. Поэтому векторным произведением векторов напряженности электрического поля и индукции магнитного поля можно считать вектор

$$[BE] = \{E_y * B_z, E_x * B_z, 0\} \quad (57)$$

Вычислив вектор плотности тока в каждой точке двумерной сетки, необходимо вычислить проекцию каждого вектора плотности тока на диагональ сетки. Сложив все скалярные проекции, полученное значение необходимо численно проинтегрировать (умножить на длину диагонали сетки). Полученное значение и будет являться электродвижущей силой Холла.

5. Структура разработанного программного обеспечения

Внешний вид пользовательского интерфейса для использования описанной в этой главе модели и диаграмма вариантов использования, показывающая его возможности, приводятся ниже [5]:

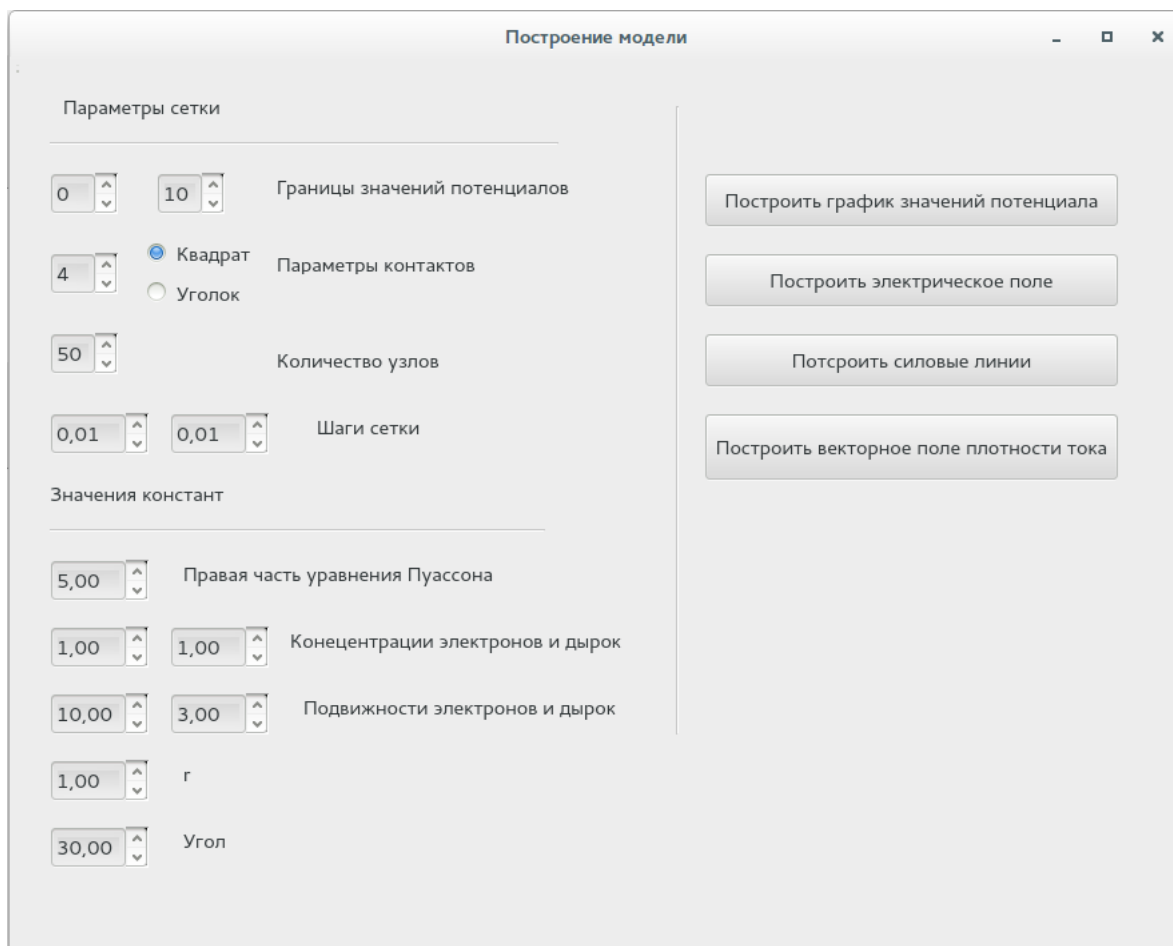


Рис. 5. Внешний вид интерфейса для работы с моделью

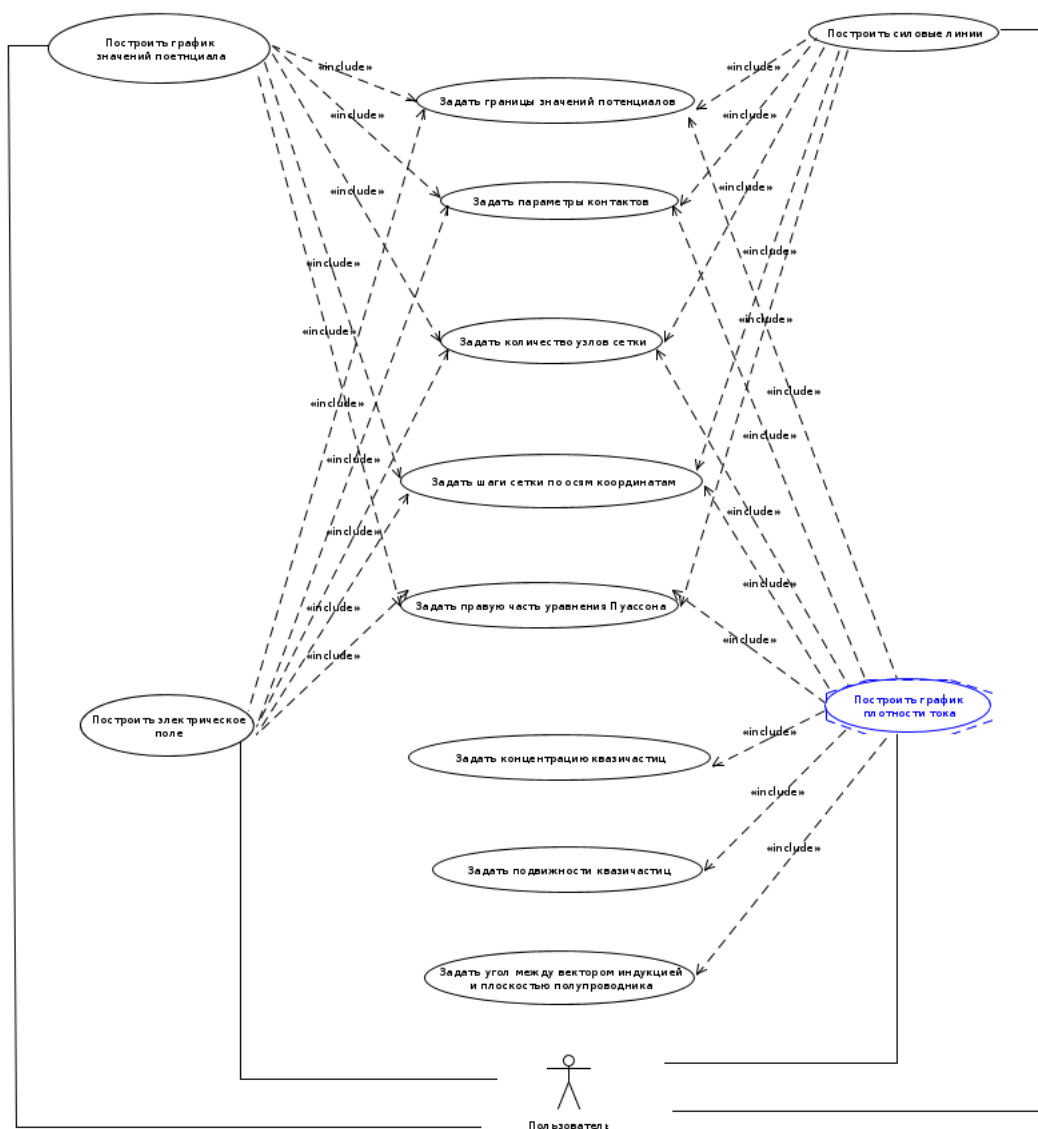


Рис. 6. Диаграмма прецедентов для программной реализации модели

В данной работе строилась база для хранения экспериментальных данных, оформленных в виде файлов формата .xls. Как известно, у каждого такого файла данные приводятся на отдельных листах, собранных в одну книгу. Каждый файл - серия листов. Файлы, содержащие в себе данные, проведенные в рамках одного эксперимента, собраны в отдельные папки. Ниже приводится диаграмма классов [5], описывающая структуру построенной базы:

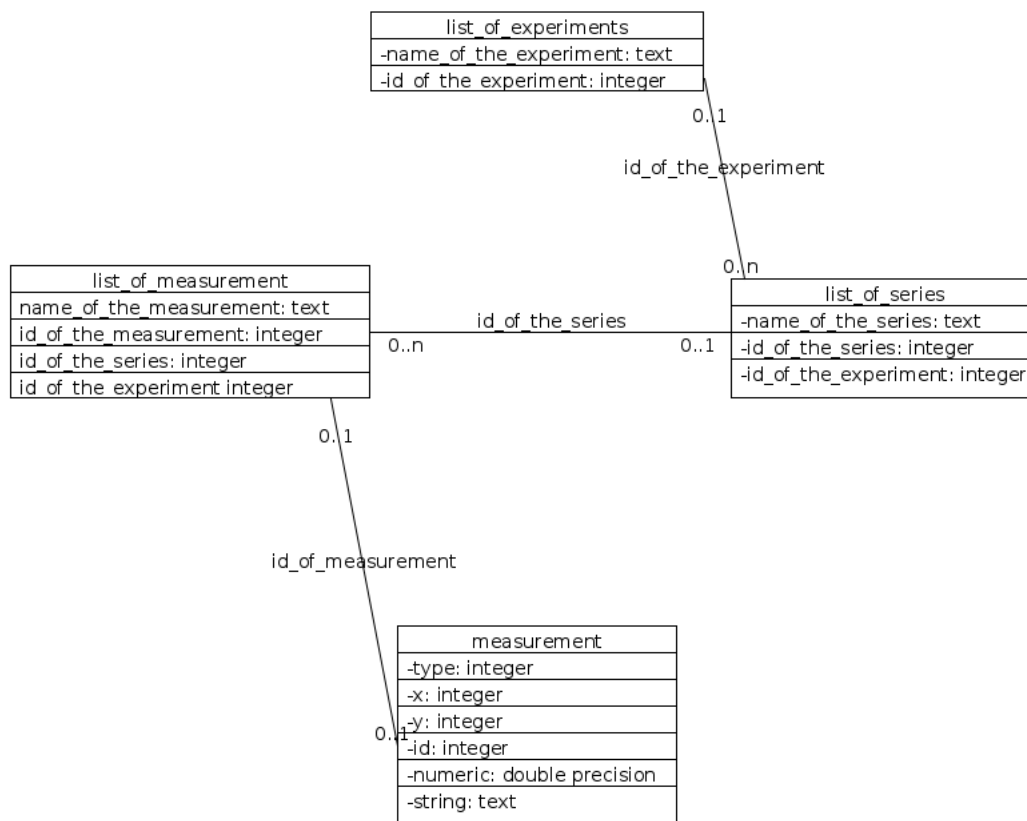


Рис. 7. Диаграмма классов. Структура базы

В данной диаграмме название класса соответствует названию таблицы в базе, а каждый атрибут класса соответствует колонке таблицы. Папка с файлами формата .xls рассматривается как один эксперимент. Название папки - название эксперимента. Соответственно, при загрузке файла в базу в таблице list_of_experiment (название эксперимента) создается запись, в которой в колонку name_of_the_experiment (название эксперимента), записывается указанное пользователем название эксперимента (предполагается, что это название папки), а в колонку id_of_the_experiment вносится индивидуальный идентификатор эксперимента. Идентификатор генерируется в интерфейсе без участия пользователя (об этом будет рассказано ниже) [11].

В рамках одного эксперимента может быть проведено много серий измерений. Каждое измерение соответствует листу загружаемого файла. Каждый файл соответствует серии этих измерений. При загрузке файла в базу создается запись в таблице list_of_series (список серий). Таблица содержит в себе две колонки. Первая колонка - name_of_the_series (название серии). Сюда записывается название файла .xls. Вторая колонка - id_of_the_series (номер серии) содержит в себе индивидуальный идентификатор серии измерений (также генерируется без участия пользователя). Третья колонка - id_of_the_experiment (номер эксперимента) содержит в себе номер эксперимента, в рамках которого проводилась данная серия измерений.

Как уже говорилось выше, в рамках одной серии измерений может проводиться большое количество измерений. Для контроля этой информации в базе существует таблица `list_of_measurement` (список измерений), которая имеет 4 колонки. В первой колонке указывается название измерения (название листа файла `.xls`). Эта колонка называется `name_of_the_measurement`. Во вторую колонку вносится индивидуальный идентификатор каждого измерения. Эта колонка называется `id_of_the_measurement`. Третья колонка - `id_of_the_series` (идентификатор серии). В нее вносится индивидуальный идентификатор серии, в рамках которой проводилось измерение. Он соответствует идентификатору серии из таблицы `list_of_the_series`. Четвертая колонка существует для того, чтобы для каждого измерения.

Непосредственно экспериментальные данные оформлены в виде таблиц, находящихся на листах. Для хранения этих таблиц в базе есть таблица, которая называется `measurement` (измерение). Основная сложность в том, что информацию, занесенную в эту таблицу, необходимо по требованию пользователя выгружать в интерфейс в том виде, в котором она была до загрузки в базу. Каждая запись в этой таблице несет в себе информацию, заключенную в одной ячейке. Первая колонка называется `type`. В зависимости от того, чем являлась информация в ячейке (числом или строкой) в эту колонку записывается 1 или 2 соответственно. В следующие 2 колонки заносятся номера строки и столбца (координаты `x` и `y` ячейки в таблице). Четвертая колонка - индивидуальный идентификатор измерения, по которому всегда можно определить, на каком листе находится ячейка. Следующие 2 колонки взаимоисключаемы для каждой записи - `numeric` заполняется числом из ячейки (если в ячейке хранится число), а `string` заполняется если в ячейке хранилась строка. Ниже приводится интерфейс для работы с базой и диаграмма вариантов [5] использования для этого интерфейса:

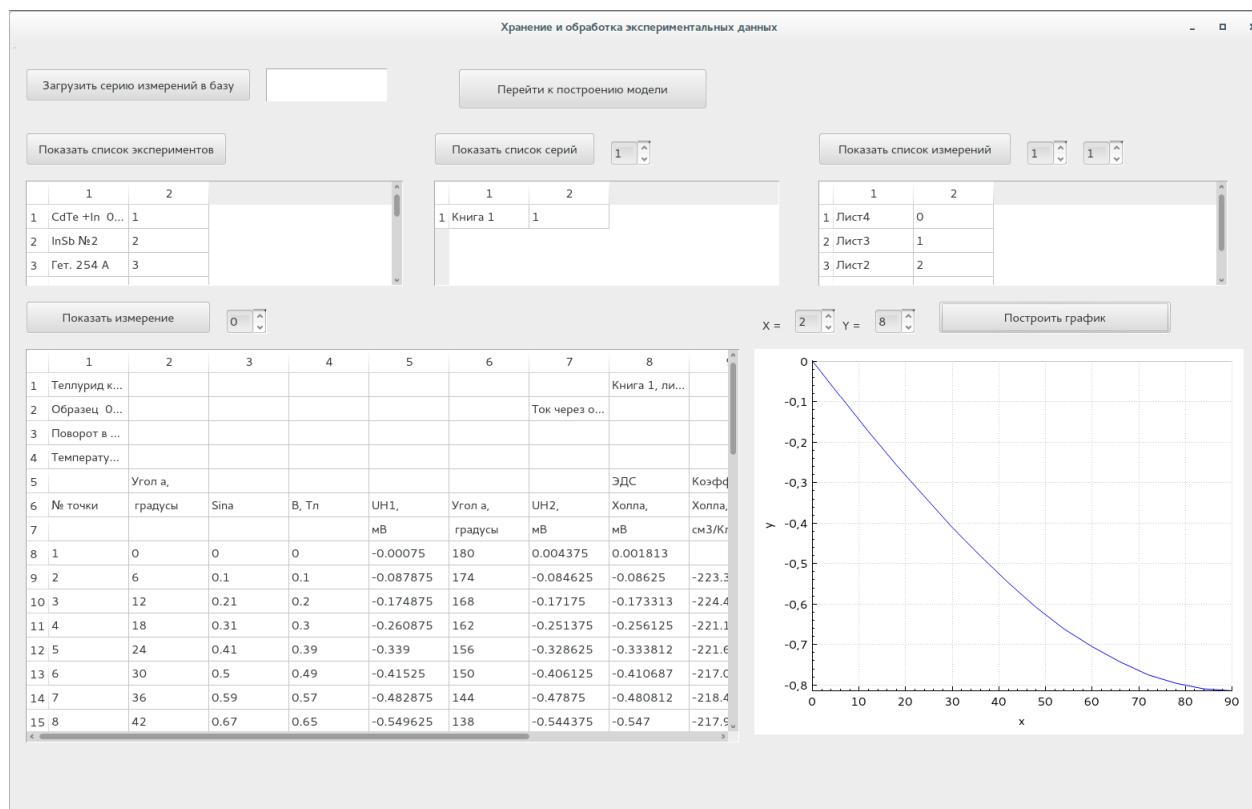


Рис. 8. Внешний вид интерфейса для работы с базой

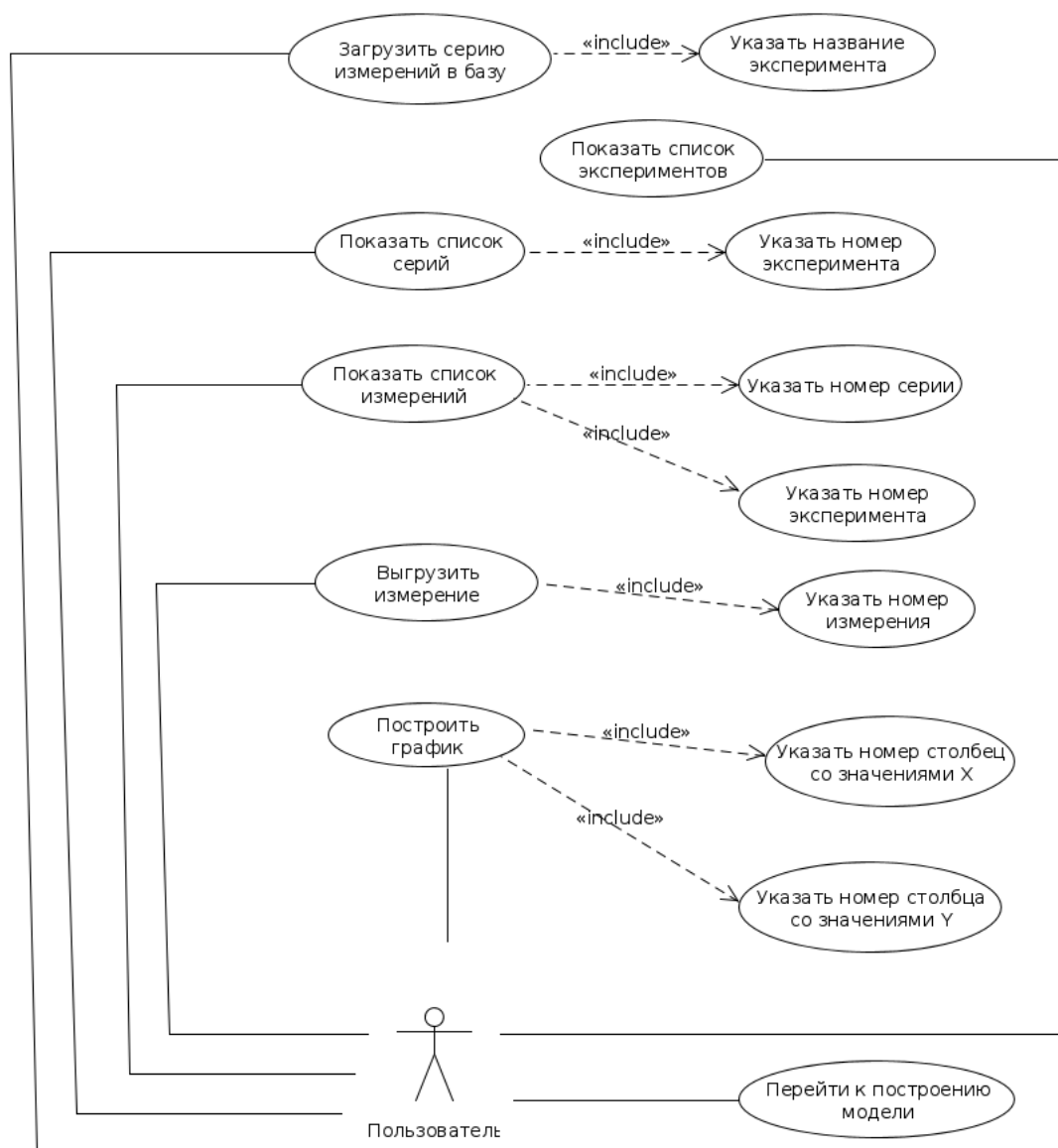


Рис. 9. Диаграмма прецедентов для приложения, работающего с базой

6. Программная реализация и обсуждение результатов

Интерфейс был реализован на языке C++ с использованием кроссплатформенного инструментария разработки ПО QT. Программа, описывающая этот интерфейс, была названа ModelingHoll3 и состоит из 5 файлов:

1. ModelingHoll3.pro
2. mainwindow.h
3. main.cpp
4. mainwindow.cpp
5. mainwindow.ui

UI-файл Qt Designer представляет собой дерево виджетов формы в формате XML. Формы могут быть обработаны:

- Во время компиляции, что означает, что формы будут преобразованы в код C++, который может быть скомпилирован
- Во время выполнения, что означает, что формы обрабатываются классом QUiLoader, который динамически создаёт дерево виджетов при анализе XML-файла.

mainwindow.ui в Qt Designer выглядит следующим образом:

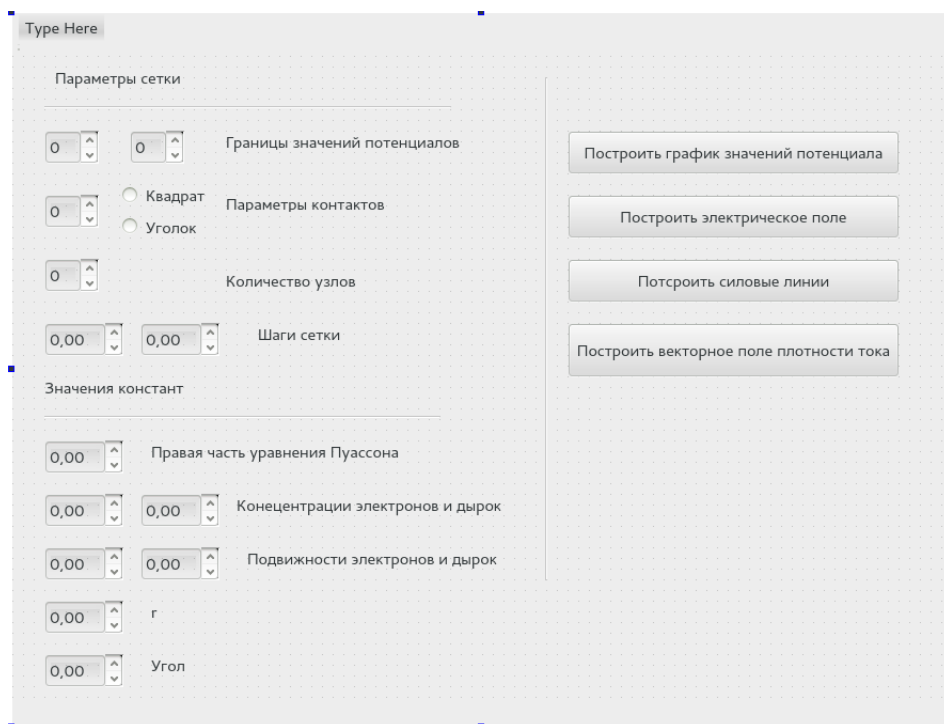


Рис. 10. Файл mainwindow.ui, открытый в Qt Designer

Для подключения его к программе использовался подход одиночного наследования. То есть создавался подкласс от стандартного виджета Qt и подключался закрытый экземпляр объекта формы пользовательского интерфейса, имеющий вид переменной-члена. Используемые в этом случае компоненты делают видимыми виджеты и компоновщики, используемые в форме, для подкласса виджета Qt и предоставляют стандартную систему для создания соединений сигналов и слотов между пользовательским интерфейсом и другими объектами в вашем приложении [12].

Подкласс определяется следующим образом:

```
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
private slots:
    void on_pushButton_clicked();
    void on_pushButton_2_clicked();
    void on_pushButton_3_clicked();
    void on_pushButton_4_clicked();
private:
    Ui::MainWindow *ui;
};
```

Сигналы и слоты соединены в пользовательском интерфейсе виджетов. При нажатии кнопки, запускается соответствующая функция.

В файле main.cpp происходит создание объекта класса MainWindow и запуск его свойства show(), в результате чего при запуске программы пользователю показывается интерфейс [13].

Дипломный проект, описываемый в этой работе, позволяет визуализировать результаты работы построенной выше модели с помощью четырех различных способов, а именно:

1. визуализация тенденции роста значений потенциала;
2. визуализация векторного поля, состоящего из векторов напряжения электрического поля;
3. визуализация силовых линий электрического поля;
4. визуализация векторного поля, состоящего из векторов плотности тока.

Для того, чтобы использовать тот или иной способ визуализации, пользователь должен указать необходимую информацию в разделах «Параметры сетки» и «Значения констант», а именно:

- границы значений потенциала (минимальное и максимальное значения)
- параметры контактов (форма и размер)
- количество узлов сетки
- Шаги сетки (по x и по y)
- Правая часть уравнения Пуассона
- концентрации электронов и дырок
- подвижности электронов и дырок
- τ
- угол

После этого пользователь должен нажать на одну из четырех кнопок. Каждая кнопка соответствует одному типу визуализации. При нажатии на кнопку запускается соответствующая функция (свойство класса MainWindow).

Ниде приводится таблица, в которой показано соответствие кнопок и вызываемых ими функций.

Таблица 1. Таблица соответствия кнопки и вызываемой ею функции

Название кнопки	Название функции
Построить график значений потенциала	on_pushButton_clicked()
Построить векторное поле градиента	on_pushButton_2_clicked()
Построить векторное поле плотности тока	on_pushButton_3_clicked()
Построить силовые линии	on_pushButton_4_clicked()

В этих функциях в специально объявленные для этого переменные записываются параметры сетки и значения констант, занесенные пользователем в интерфейс. В данной дипломной работе при решении уравнения Пуассона и вычислении значений потенциала на узлах двумерной сетки происходит с учетом двух типов граничных условий:

1. значения потенциала на граничных узлах сетки равно значениям потенциалов на предграницных узлах сетки;
2. заданы значение потенциала в левом нижнем углу сетки (минимальное значение) и правом верхнем углу сетки (максимальное значение) и параметры контактов.

В экспериментах с установкой Цилиндр к полупроводниковому материалу прикрепляются контакты, имеющие определенные форму и размер. Эти контакты необходимо учитывать при решении уравнения Пуассона. Для этого пользователь в интерфейсе указывает минимальное и максимальное значение потенциала в левом нижнем и правом верхнем углах сетки (предполагается, что именно там прикреплены контакты), и указывает их параметры, а именно форму (квадраты или уголки), и размеры (длину одной из сторон контакта).

Например, значение правой границы потенциала введено в спин-бокс в левом верхнем углу. Это значение считывается следующим образом:

```
int potential2 = ui ->spinBox_2->value();
```

Использование виджета придворяется префиксом `ui`, каждый виджет имеет уникальное название и множество свойств. Для считывания значение используется `value()` [13]. Аналогичным образом считываются все остальные данные, введенные в спин-боксы. Форма контактов, отмеченная пользователем в специальной конпке, считывается в переменную `form` следующим образом:

```
if (ui -> radioButton-> isChecked() == 1) form = 1;
if (ui -> radioButton_2 -> isChecked() == 1) form = 2;
```

В переменную `grafic` записывается значение, которое даст возможность программе понять, какой способ визуализации выбрал пользователь. Значение переменной `grafic` может быть:

- 1 - если пользователь нажал на кнопку «Построить график значений потенциала»;
- 2 - если пользователь нажал на кнопку «Построить векторное поле градиента»;
- 3 - если пользователь нажал на кнопку «Построить силовые линии»;
- 4 - если пользователь нажал на кнопку «Построить векторное поле плотности тока».

После считывания всей введенной пользователем информации, она записывается в файл `parametres.dat`, после чего осуществляется системный вызов, запускающей программу `modeling_holl8`, в которой, помимо описанной выше модели, реализуется визуализация всех выше перечисленных типов.

Программа начинает свою работу с функции `parser()`, которая считывает из файла `parametres.dat` все необходимые данные. Поскольку значения потенциала ищется на узлах двумерной сетки, коэффициенты уравнения Пуассона, значения правой части, решения уравнения Пуассона, значения компонент векторов градиента по осям x и y и значения компонент плотности тока по осям x и y будут заноситься в двумерные массивы, количество элементов которых равно количеству узлов сетки. Поэтому, после выполнения функции `parser()`, выделяется память под двумерные массивы. Далее в цикле по всем элементам массивов, происходит заполнение массивов коэффициентов уравнения Пуассона и значений правой части значениями, равными $a=b=c=d=1$, $e = -4$, $f = f_prav$ (переменная, в которой лежит введенное пользователем в соответствующий спин-бокс значение правой части). Массив `u[x][y]`, в котором будут лежать значения потенциала в узлах сетки, заполняется значениями 0. Далее запускается функция `sor()`, реализующая метод последовательной верхней релаксации, а значит решающая уравнения Пуассона с учетом всех граничных условий. Текст функции приводится в разделе Приложение.

В начале метода вычисляется спектральный радиус в соответствии с формулой (41):

$$r_{jac} = \cos(2 * \pi / n);$$

Как уже было сказано, условием сходимости является равносильность или превышение значения нормы невязки по отношению к заданному числу. Это число равно произведению суммы всех значений элементов массивов значений правой части (сложенной в переменную `anormf`) на $EPS = 1.0e-5$. После вычисления переменной `anormf`, начинаются итерации метода. Расчет узлов производится по всей сетке, кроме граничных узлов. Сначала вычисляется формула (45):

$$resid = a[j][1] * u[j+1][1] + b[j][1] * u[j-1][1] + c[j][1] * u[j][1+1] + d[j][1] * u[j][1-1] + e[j][1] * u[j][1] - dx * dy * f[j][1];$$

Далее формула (46):

$$u[j][1] -= omega * resid / e[j][1];$$

После пересчета узлов выполняются граничные условия - граничным узлам даются значения предграничных, в левом верхнем и правом нижнем углу восстанавливаются значения потенциалов. В зависимости от того, что указывает пользователь, граничные условия в узлах сетки могут быть двух разных форм - квадраты и углы. Далее происходит динамическое вычисление параметра сходимости и проверка условия сходимости метода. После выполнения функции `sor()`, в массиве `u[x][y]` записаны значения потенциалов на узлах двумерной сетки. Причем индексы элементов массива совпадают с индексами узлов сетки. Далее запускается функция `calculate_gradient()`, вычисляющая значения компонент вектора градиента по осям x и y по формулам (53) и (54):

```
void calculate_gradient(){
    for(x=2; x<n-2; x++){
        for (y=2; y<n-2; y++){
            ux[x][y] = (-u[x+2][y]+8*u[x+1][y]
                -8*u[x-1][y]+u[x-2][y])/(12*dx);
            uy[x][y] = (-u[x][y+2]+8*u[x][y+1]
                -8*u[x][y-1]+u[x][y-2])/(12*dy);
        }
    }
}
```

После чего, запускается функция `calculate_current_density()`, вычисляющая компоненты вектора плотности тока по осям x и y и электродвижущую силу Холла. Текст функции приводится в разделе Приложение. В переменную `e1` записывается значение заряда электрона в экспоненциальной форме. Далее, в соответствии с формулой (55) происходит вычисление двух констант: $e(n\mu_n + p\mu_p)$ и $re(p\mu_p^2 - n\mu_n^2)$. После чего, в цикле по всем узлам сетки происходит вычисление двух компонент плотности тока по той же формуле (55). Векторное произведение вычисляется по формуле (57). На каждом шаге цикла, после вычисления двух компонент вектора, вычисляется проекция данного вектора плотности тока на диагональ сетки. Полученное значение домножается на длину диагонали и суммируется с переменной `ads`. На выходе цикла получены компоненты всех векторов плотности тока сетки (необходимые для визуализации векторного поля), и значение электродвижущей силы Холла для данных граничных условий.

Далее начинается визуализация результатов. Если значение переменной `grafic` равно 1, то запускаются две функции. Первая, `create_file_with_coordinate_u()`, записывает в файл `data1.dat` значение потенциала `u[x][y]` и его координаты на сетке. Далее запускается функция `challenge_gnuplot()` записывает в файл `main.gnu` команды для `gnuplot` [9]:

```
fprintf( main_gnu, "set pm3d map\nset size
square\nsplot 'data1.dat' palette\n pause mouse keypress \n" );
```

После этого программа `modeling_holl8` завершает свою работу, и в функции `on_pushButton_2_clicked()` системным вызовом запускается `gnuplot`:

```
system("gnuplot 'main.gnu' &");
```

Все примеры различных визуализаций будут приводиться в конце этой главы, как и все примеры работы проекта. Если же значение функции `grafic` не равно 1, то запускаются функции `opengl`, так как именно с помощью `opengl` были реализованы остальные три вида визуализации. Функция `Display()` приводится в разделе Приложение. В зависимости от значения переменной `grafic`, функция `Display` рисует тот или иной график, но до этой работы она рисует оси координат. Состоят оси из 6 линий красного цвета и двух букв. Далее, если значенике переменной `grafic` равно 2, начинается отрисовка векторного поля. Вектор рисуется в виде стрелочки. Стрелочка в точке (0,0) рисует функция `arrog()` по тому же принципу, по котором отрисовывались оси. Эта функция вызывается в цикле по всем узлам сетки, но перед этим при помощи команд `Translatef` и `Rotatef` будущий вектор переносится в позицию текущего узла и поворачивается на определенный угол, который вычисляется по формуле [4]:

$$\varphi = \arctan \frac{u_y}{u_x} \quad (58)$$

Если значение переменной `grafic` оказалось равным 3, значит начинается процесс построения силовых линий. Силовые линии электрического поля строятся на основании значений градиента в каждом узле. Таким образом, в случае расположения контактов в левом нижнем и правом верхнем углах силовые линии, в виде дуг, также, как и векторы градиента, из левого нижнего угла следуют в правый верхний угол. Фрагмент функции `Display()`, отображающий силовые линии при значении переменной `grafic`, равном 3, приводится в разделе Приложение. Построение дуг осуществляется в три этапа:

1. проверка принадлежности рассматриваемого узла диагонали сетки;
2. в случае принадлежности, построение дуги от рассматриваемого узла в правый верхний угол до границы сетки;
3. построение дуги от рассматриваемого узла в нижний левый угол к границе сетки.

В цикле по координатам сетки на принадлежность диагонали проверяется каждый узел. В случае, если его x координата равна выражению $n-y-1$, для этого узла выполняются два следующих этапа. Из ячейки массива значений компонент ux вектора градиента и массива значений компонент uy вектора градиента с номерами строки и столбца, равными координате найденной диагональной точки берутся значения компонент вектора градиента в этом узле. Оба значение нормируются по формулам:

$$ux_n = \frac{ux}{\sqrt{ux^2 + uy^2}} \quad (59)$$

$$uy_n = \frac{uy}{\sqrt{ux^2 + uy^2}} \quad (60)$$

После чего координаты второй точки дуги определяются, как:

$$x = x + ux_n * dl \quad (61)$$

$$y = y + uy_n * dl \quad (62)$$

где dl - длина отрезка от текущей точки к новой.

Новая точка хоть и находится в рамках сетки, имеет нецелочисленные координаты, а значит точного значения компонент вектора градиента в этой точке нет. Для того, чтобы найти следующую точку, нужно усреднить значения потенциалов в четырех соседних точках. Для этого, значения координат полученных точек округляются до первого целого значения, меньшего чем округляемое. Из полученных координат простыми арифметическими действиями можно получить координаты других трех точек:

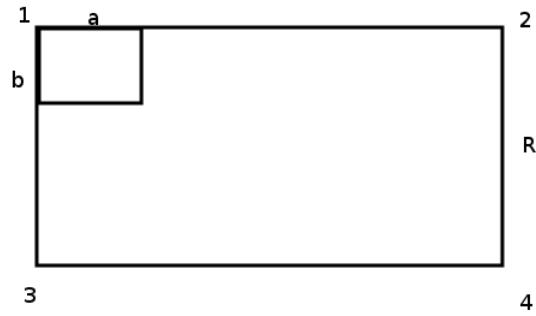


Рис. 11. Положение четырех соседних точек

$$x_3 = \lfloor x \rfloor \quad (63)$$

$$y_3 = \lfloor y \rfloor \quad (64)$$

$$x_1 = x_3 \quad (65)$$

$$y_1 = y_3 + 1 \quad (66)$$

$$x_2 = x_3 + 1 \quad (67)$$

$$y_2 = y_3 + 1 \quad (68)$$

$$x_4 = x_3 + 1 \quad (69)$$

$$y_4 = y_3 \quad (70)$$

Получив новые точки, нужно усреднить значения компонент вектора градиента по формуле:

$$ux_{average} = \frac{ux[x_1][y_1]\sqrt{a^2 + b^2} + ux[x_2][y_2]\sqrt{(R - a)^2 + b^2}}{4} + \quad (71)$$

$$\frac{ux[x_3][y_3]\sqrt{a^2 + (R - b)^2} + ux[x_4][y_4]\sqrt{(R - a)^2 + (R - b)^2}}{4} \quad (72)$$

$$uy_{average} = \frac{uy[x_1][y_1]\sqrt{a^2 + b^2} + uy[x_2][y_2]\sqrt{(R - a)^2 + b^2}}{4} + \quad (73)$$

$$\frac{uy[x_3][y_3]\sqrt{a^2 + (R - b)^2} + uy[x_4][y_4]\sqrt{(R - a)^2 + (R - b)^2}}{4} \quad (74)$$

После нахождения приблизительного значения компонент вектора градиента в этой точке, эти компоненты нормализуются по формулам 26 и 27. Координаты следующей точке можно получить как:

$$x = x + ux_{average} * dl \quad (75)$$

$$y = y + uy_{average} * dl \quad (76)$$

Силовые линии строятся вдоль направления увеличения значений градиента.

Этот алгоритм повторяется в цикле с необходимым размером счетчика. По мере нахождения точек, они соединяются кривой, образуя необходимую дугу.

Третий этап построения дуги (ее нижняя половина) выполняется аналогичным образом. Однако, полученная дуга должна быть развернута по отношению к верхней половине дуги на 180° . Поэтому при нормализации полученных компонент вектора градиента формулы 26 и 27 берутся со знаком минус. Каждая точка добавляется со своим цветом (для окрашивания дуг). Как известно, функции `glColor3f` передаются на вход значения красной, зеленой и синей компонент цвета. Красная и синяя компоненты цвета передаются как модуль нормализованного приблизительного значения градиента в данной точке

Если же значение переменной `grafic` было равно 4, значит начинается процесс построения векторного поля, состоящего из векторов плотности тока. Это поле строится аналогично векторному полю, состоящему из векторов градиента. Угол поворота вектора определяется как арктангенс отношения компонент вектора плотности тока по y и по x .

$$\varphi = \arctan \frac{J_y}{J_x} \quad (77)$$

Построенная и реализованная выше модель является частью дипломного проекта, решающая задачу №1. В задаче №2 требовалось составить базу данных, реализовать для нее пользовательский интерфейс и модуль для графического отображения результатов. Экспериментальные данные оформлены в виде файлов формата `.xls`. Для анализа этих файлов в данной работе была разработана отдельная программа. Она получает на вход путь к файлу, который необходимо проанализировать и путь, по которому нужно создать файл формата `.txt` с разобранными файлами.

Как уже говорилось выше, файл формата .xls представляет собой некоторое подобие книги, содержащей в себе листы с таблицами. Все собранные данные будут складываться в массив структур `cells_of_file`, каждый элемент которого представляет собой содержимое ячейки и ее характеристики (координаты `x` и `y` положения ячейки в таблице, тип хранящихся данных и название листа, на котором расположена ячейка). Такая структура в программе описывается следующим образом:

```
typedef struct cell_of_file{
    int x;
    int y;
    int type;
    union {
        double numeric_value;
        char *string_value;
    };
    char *sheet_name;
}cell_of_excel;
```

В характеристику элемента структуры `type` записывается 1, если в ячейке хранится число, и 2 если в ячейке хранится строка. И, естественно, отдельным параметром является информация, хранящаяся в ячейке. Это может быть число, строка или строка, но не то и другое одновременно. Поэтому параметры `numeric_value` и `string_value` взаимоисключаемые. Для работы с файлами формата .xls была выбрана библиотека `libxls` [18]. Открытие файла осуществляется при помощи команды `xls_open`, вторым аргументом которой передается кодировка открываемого файла:

```
pWB = xls_open(argv[1], encoding);
```

Далее вызывается функция `xls_parser()`, которая и анализирует открытый файл. Текст этой функции приводится в приложении. В этой функции в ходе выполнения трех циклов (по листам, по строкам и по столбцам) заполняется структурный массив `cells_of_file`, каждый элемент которого является экземпляром структуры, описанной выше. Запускается цикл, который будет выполняться столько раз, сколько есть листов в открытом файле (`pWB->sheets.count`). Открыв лист, необходимо выделить память для структурного массива с параметрами каждой ячейки. Размер этого массива будет равен количеству используемых ячеек листа. Для того, чтобы узнать это количество, номер последней используемой строки умножает на номер последнего используемого столбца (`int number_of_cells = number_of_rows * number_of_cols`). Далее запускается цикл, тело которого будет выполняться столько раз, сколько в данном листе используется строк (`cellRow <= pWS->rows.lastrow`). Ранее был создан экземпляр `row` структуры `st_row_data` (хранящей в себе параметры строки). На каждом шаге этого цикла в эту переменную записываются параметры текущей строки (`row = xls_row(pWS, cellRow)`). Далее запускается третий цикл (цикл по столбцам), в начале тела которого аналогичным образом создается экземпляр структуры, хранящей параметры ячейки (`xlsCell *cell = xls_cell(pWS, cellRow, cellCol)`). Таким образом, в ходе тройного цикла, рассматриваются все ячейки файла. Для индексации по массиву `cells_of_file` используется переменная `counter`, инкрементируемая

в конце тела третьего цикла (по столбцам). После создания экземпляра структуры `xlsCell`, у текущего элемента массива `cells_of_file` заполняются параметры `x` и `y` номером шага по массива по строкам и массива по столбцам соответственно. Далее в параметр `sheet_name` текущего элемента записывается название рассматриваемого листа. В ходе выполнения различных условных операторов заполняются параметры `type` и `numeric_value` (`string_value`) текущего элемента массива. Если в ячейке хранится целое или дробное число (`if (cell->id == 0x27e || cell->id == 0x0BD || cell->id == 0x203)`), в параметр `type` записывается число 1, а в параметр `numeric_value` записывается содержимое ячейки. Аналогичные действия производятся и в случае, если содержимое ячейки является формулой (`cell->id == 0x06`), дающей число (`if (cell->l == 0)`). Если же это, так называемая, булевская формула, то в `string_value` записывается соответственно `true` или `false`. Также, если формула в файле дает значение `error`, то в `string_value` записывается `error`. И, наконец, если в ячейке лежит строка или что-то, что не является формулой, то ее содержимое также записывается в `string_value`. Далее, как уже отмечалось выше, в конце тела третьего цикла значение переменной `counter` (которая является итератором массива `cells_of_file`) увеличивается на 1. После выполнения двух внутренних циклов, запускается функция `spase()`. Эта функция проходит по массиву `cells_of_file` и, в случае, если в ячейке не содержалось никакой информации, дает параметру `type` данного элемента значение 3. При этом переменная `count_for_qt_array` увеличивается на 1 только в том случае, если в ячейке хранилась какая-то информация. Это требуется, чтобы в будущем сформировать подобный массив, но только несущими какую-либо информацию ячейками. Эта переменная записывается в файл `counter.dat`, находящийся среди файлов проекта. В этот же файл записывается и количество листов. После чего запускается функция `print_massive()`, которая осуществляет печать массива `cells_of_file`. Благодаря символу `>` между путем до файла `.xls` и путем до файла `.txt` в команде запуска программы, печать производится в файл, лежащему в той же папке, что и файл `.xls`. Далее начинается новый шаг цикла первого цикла (по всем листам файла).

```

560      8
561      -221.121250
562      Лист4
563
564      1
565      11
566      0
567      5.000000
568      Лист4
569
570      1
571      11
572      1
573      24.000000
574      Лист4
575
576      1
577      11
578      2
579      0.410000
580      Лист4
581
582      1
583      11
584      3
585      0.390000
586      Лист4
587
588      1
589      11
590      4
591      -0.339000
592      Лист4
593

```

Рис. 12. Пример полученного текстового файла

С помощью СУБД PostgreSQL создается база со структурой, описанной в предыдущей главе [11]. Проект, реализующий интерфейс для работы с этой базой и графический модуль для построения графиков был назван finalwork15, и состоит из следующих файлов:

1. finalwork15.pro
2. mainwindow.h
3. functional_for_work_with_database.cpp
4. functional_for_work_with_files.cpp
5. mainwindow.cpp
6. main.cpp

Также к проекту были подключены файлы qcustomplot.h и qcustomplot.cpp, в которых реализован функционал для построения графиков в виджете QT и который являются сторонней библиотекой [14]. Как и в прошлый раз, прежде всего стоит привести внешний вид файла mainwindow.ui в QTDesigner:

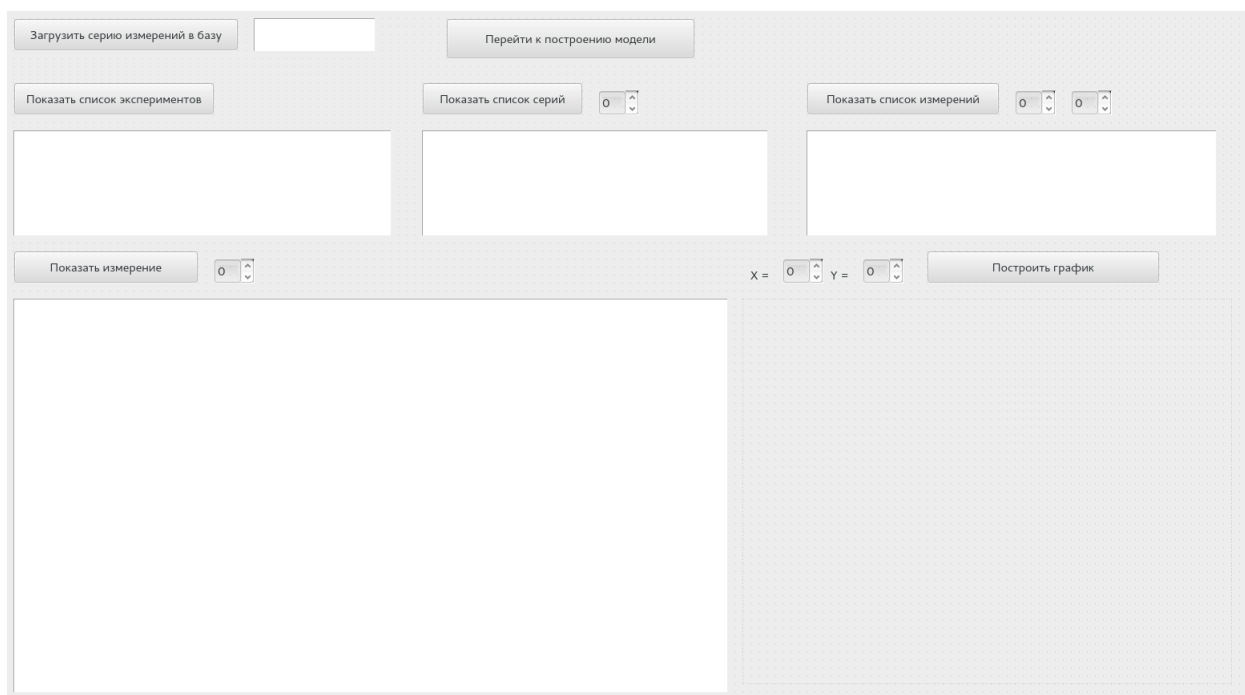


Рис. 13. Файл mainwindow.ui, открытый в QTDesigner

Интерфейс состоит из 7 кнопок, четырех таблиц tableview, 6 спин-боксов и одного виджета для построения графиков. По аналогии с предыдущем проектом QT, существует класс MainWindow:

```
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:

    void on_pushButton_clicked();
    void on_pushButton_2_clicked();
    void on_pushButton_3_clicked();
    void on_pushButton_4_clicked();
    void on_pushButton_5_clicked();
    void on_pushButton_6_clicked();
    void on_pushButton_7_clicked();

private:
    Ui::MainWindow *ui;
};
```

и существует соответствие между кнопками и функциями. Ниде приводится таблица, в которой показано соответствие кнопок и вызываемых ими функций.

Таблица 2. Таблица соответствия кнопки и вызываемой ею функции

Название кнопки	Название функции
Загрузить серию измерений в базу	on_pushButton_clicked()
Показать список измерений	on_pushButton_2_clicked()
Показать список серий	on_pushButton_3_clicked()
Показать список экспериментов	on_pushButton_4_clicked()
Показать измерение	on_pushButton_5_clicked()
Перейти к построению модели	on_pushButton_6_clicked()
Построить график	on_pushButton_7_clicked()

В файле Mainwindow.h объявлены следующие функции для работы с базой:

- void create_a_record_of_the_experiment(); - функция создает запись в таблице list_of_experiments (записывает название и идентификатор)

- `void create_a_record_of_the_series();` - функция создает запись в таблице `list_of_series` (записывает название, идентификатор серии и идентификатор эксперимента)
- `void create_records_in_measurement();` - создает запись в таблице `list_of_mesurement` (записывает название, идентификатор измерения, идентификатор серии и идентификатор эксперимента)
- `void add_records_of_the_measurement(int counter_of_series, int counter_of_experiments);` - добавляет в таблицу `mesurement` новые записи (содержимые ячеек и все их параметры)
- `void upload_list_of_experiments();` - функция выгружает в интерфейс содержимое таблицы `list_of_experiments()`;
- `void upload_list_of_the_series(int id_of_the_experiemnt);` - функция выгружает в интерфейс содержимое таблицы `list_of_series()`;
- `void upload_list_of_measurement(int id_of_the_experiment, int id_of_the_series);` - функция выгружает в интерфейс содержимое таблицы `list_of_mesurement()`;
- `void upload_measurement(int id_of_the_measurement);` - функция выгружает в интерфейс запрашиваемую пользователем часть таблицы `mesurement()`;

Следующие функции вызываются при необходимости работы с файлами:

- `void xls_parser();` - функция подготавливает необходимые данные для анализа файла `.xls`
- `void analyzing_txt_file(char *txt_file);` - функция анализирует полученный текстовый файл

Были созданы два дополнительных класса. Класс `cell` - это класс, описывающий ячейку. У каждой ячейки есть такие параметры, как тип хранящихся в ячейке данных (1, если хранится число и 2, если хранится строка), координаты `x` и `y` ее положения в таблице, взаимоисключаемые параметры `numeric_value` (для числового содержимого ячейки) и `string_value` (для строкового содержимого ячейки), название и номер листа:

```
class cell{
public:
    int type;
    int x;
    int y;
    union {
        double numeric_value;
        char string_value[100];
    };
    char sheet[100];
    int sheet_id;
};
```

Класс `Cell_off_main_table` описывает ячейку таблицы `list_of_experiments`, поэтому гораздо проще предыдущего класса. Любое содержимое ячейки будет представлено строкой. Поэтому у каждой ячейки есть координаты `x` и `y` положения ячейки в таблице и ее строковое содержимое:

```

class Cell_off_main_table{
public:
    int x;
    int y;
    char string_value[100];
};

```

Для написания программы на с++, осуществляющей запросы к базе, была выбрана ибиблиотека libpq [16].

Также для работы проекта объявлены следующие глобальные переменные:

- extern int real_counter; - размер массива ячеек, сформированного после анализа файла .xls;
- extern cell *Cells; - массив, каждый элемент которого является экземпляром класса cell;
- extern QString filename; - строка, содержащая путь до файла .xls
- extern char *experiment_name; - строка, содержащая введенное пользователем с клавиатуры название эксперимента
- extern char *series_name; - строка, соедржащая в себе назование серии измерений
- extern PGresult *res; - экземпляру структуры, в котором хранится статус запроса и данные, полученные от базы
- extern PGconn *conn; - экземпляр структуры, описывающий новое соединение с базой
- extern Cell_off_main_table *main_table; - массив, каждый элемент которого является экземпляром класса Cell_off_main_table
- extern int size_of_main_table; - размер выгружаемой системной таблицы
- extern cell *database_val; - массив из выгруженных из таблицы mesurement записей
- extern int size_of_database_val; - размер этого массив

Подключение интерфейса к базе

При запуске проекта запускается функция int main(int argc, char *argv[]). Создается объект класса MainWindow w. База данных, составленная в рамках данной дипломной работы, называется TestData. В специально отведенные для этого массив копируется название базы, имя пользователя, который имеет права доступа к ней, и пароль для того, чтобы программа могла от имени этого пользователя подключиться к базе:

```

conn = PQsetdbLogin(NULL, NULL, NULL, NULL, dbname, user, pwd);
if(PQstatus(conn) == CONNECTION_BAD) {
    fprintf(stderr, "Connection to database failed.\n");
    fprintf(stderr, "%s", PQerrorMessage(conn));
    exit(1);
}

```

После чего при помощи функции PQsetdbLogin осущетсяявляется подключение к базе. Вызывается функции show() и пользователь видит интерфейс для работы с базой.

Загрузка серии измерений в базу

Для того, чтобы загрузить серию измерений в базу, пользователь должен ввести в специально отведенное для этого текстовое поле название эксперимента, в рамках которого была проведена загружаемая серия измерений и нажать на кнопку «Загрузить серию измерений в базу». Запускается функция `on_pushButton_clicked()`. В ней вызывается функция `getOpenFilename`, благодаря которой пользователь видит меню для выбора файла и путь до выбранного пользователем файла записывается в глобальную переменную `filename`:

```
filename = QFileDialog::getOpenFileName(this, tr("Open File"), "", tr(""));
```

Далее запускается функция `xls_parser()`, текст которой описан в файле `functional_for_work_with_files.cpp`. В начале функции в переменную `xls_file` складывается путь до файла `.xls`, извлеченный из переменной `filename` путем следующих действий:

```
QByteArray ba = filename.toLocal8Bit();  
char *xls_file = ba.data();
```

Далее в переменную `txt_file` складывается содержимое `xls_file`, за исключением трех последних символов. Они заменяются на символы `.txt`. Таким образом, полученный в ходе анализа файл `.txt` будет лежать в той же папке, что и файл `.xls`. Далее, в глобальную переменную `series_name` записывается название файла `.xls`, полученное копированием необходимого участка строки `xls_file` (все, кроме пути до файла и последних четырех символов формата). Поэтому в таблицу `list_of_series`, в качестве названия серии измерений, будет записано название файла `.xls`. Далее, с помощью команды `system` осуществляется системный вызов. В качестве команды для системного вызова передается 4 аргумента: путь до программы, осуществляющей анализ `.xls` файла, путь до анализируемого файла, аргумент, говорящий о том, что все команды `printf` необходимо выводить не в терминал, а в файл, и путь до файла, в который необходимо выводить результат:

```
sprintf(command, "./libxls/src/xls2csv '%s' > '%s'", xls_file, txt_file);  
system(command);
```

После чего начинает свою работу программа, содержащая в себе функцию `xls_parser()`, описанную выше. По окончании ее работы по адресу, записанному в переменной `txt_file` лежит файл с данными из файла формата `.xls`. Эта переменная подается на вход функции `analyzing_txt_file(txt_file)`, анализирующей полученные текстовый файл. Работа функции начинается с выделения памяти под массив `Cells`, который был глобально объявлен ранее. Размер его берется из файла `counter.dat` (о нем рассказывалось выше). После чего начинается анализ текстового файла с данными. Как уже говорилось выше, в текстовом файле параметры ячейки сложены следующим образом:

1. тип
2. координата x

3. координата y
4. содержимое ячейки (строковое или текстовое)
5. название листа

После чего следует пробельная строка, говорящая о том, что начинаются параметры новой ячейки. Объявляются 5 переменных, которые по своей сути являются флажками:

- `type` - флаг говорит о том, что в следующей строке файла лежит значение параметра `type` ячейки.
- `coordinate_x` - флаг говорит о том, что в следующей строке лежит значение координаты `x` ячейки
- `coordinate_y` - флаг говорит о том, что в следующей строке лежит значение координаты `y` ячейки
- `value` - флаг говорит о том, что в следующей строке лежит содержимое ячейки
- `sheet` - флаг говорит о том, что в следующей строке лежит название листа, на котором расположена ячейка

Значение переменной `type` изначально равно 1 (так как первая строка текстового файла - всегда тип первой ячейки). Все остальные переменные равны 0. Запускается цикл `while`, тело которого будет выполняться до тех пор, пока не будет достигнут конец файла. Далее, на каждом шаге цикла в переменную `buf` считывается новая строка из файла (перед этим переменная `buf` на всякий случай заполняется нулями). Если был поднят флаг `type` (значение переменной равно 1) и если первый символ считанной строки не является символом конца строки, то в переменную `type_new` с помощью функции `atoi` записывается число, которое ранее являлось строкой. Если полученное число не равнялось 3 (то есть если ячейка не была пустой), то в параметр `type` текущего элемента массива `Cells` записывается это число и поднимается флаг `coordinate_x`. Флаг `type` опускается (переменной дается значение 0). Если был поднят флаг `coordinate_x`, аналогичным образом считанное число записывается в параметр `x` текущего элемента `Cells`. Флаг `coordinate_x` опускается, флаг `coordinate_y` поднимается и происходят аналогичные действия. В случае поднятого флага `value` начинается считывание содержимого ячейки. Если в текущем элементе массива `Cells` параметр `type` равняется 1, значит считывается числовое значение. Поскольку в QT числа формата `double` имеют не запятую, а точку, перед преобразованием строки в число, запятая в переменной `buf` меняется на точку. Если значение переменной `type` равняется 2, то значение переменной `buf` записывается в строковый параметр элемента массива `Cells`. Далее флаг `value` опускается и поднимается флаг `sheet`. Следующая считываемая строка является названием листа. Оно записывается в параметр `sheet` элемента массива `Cells`. Флаг `sheet` опускается. Далее поднимается флаг `type` и следующая строка вновь будет являться значением переменной `type` для следующего элемента массива `Cells`. В итоге после работы функции массив `Cells` оказывается заполненным информацией из файла `.xls`, который пользователь выбрал для загрузки в базу.

Продолжается работа функции `on_pushButton_clicked()`. Необходимо считать из текстового поля название эксперимента, в рамках которого была проведена загружаемая серия измерений. Поскольку название эксперимента было введено в `textEdit`, считывание из него строки происходит следующим образом:

```
QString str= ui -> textEdit ->toPlainText();
QByteArray ba = str.toLocal8Bit();
experiment_name = ba.data();
```

Теперь необходимо в таблице `list_of_experiments` создать новую запись об эксперименте. Для этого запускается функция `create_a_record_of_the_experiment()`. Функция создает новую запись об эксперименте в случае, если эксперимента с таким названием в базе еще нет. Вначале функция осуществляет запрос к базе на выгрузку всех имеющих в таблице идентификаторов [17]:

```
sprintf(query, "select id_of_the_experiment from list_of_experiments;");
res = PQexec(conn, query);
```

Функция `PQntuples` вернет число прочитанных из таблицы строк, а функция `PQnfields` – число ячеек в одной строке. Если выгруженных идентификаторов больше, чем 0, берет последний выгруженный идентификатор (`PQgetvalue(res, i, n)`) и в переменную `counter_of_experiments` записывает его значение, увеличенное на 1 (при помощи функции `atoi`). Если же количество выгруженных идентификаторов меньше, чем 0 (в таблице нет ни одной записи), то переменной `counter_of_experiments` присваивается значение 1. Далее осуществляется запрос на выгрузку всех имен названий экспериментов из базы [17]:

```
sprintf(query, "select name_of_the_experiment from list_of_experiments");
res = PQexec(conn, query);
```

Также, как и идентификаторы, выгруженные имена по одному записываются в строку. Далее они сравниваются со строкой `experiment_name`, и в случае совпадения значение переменной `new_flag` увеличивается на 1. Если же значение переменной `new_flag` ни разу не увеличилось, осуществляется запрос к базе на вставку новой записи с названием эксперимента `experiment_name` и идентификатором `counter_of_the_experiment` [17]:

```
sprintf(query, "insert into
list_of_experiments (name_of_the_experiment, id_of_the_experiment)
values ('%s', %d);", experiment_name, counter_of_experiments);
res = PQexec(conn, query);
```

После создания записи об эксперименте (в случае, если такого эксперимента еще не было), необходимо создать запись в таблице `list_of_the_series`. Для этого запускается функция `create_a_record_of_the_series()`. В рамках одного эксперимента не может быть проведено двух серий с одинаковым названием. Но одна серия измерений может проводиться в рамках нескольких экспериментов. Поэтому исключено создание в этой таблице записей, у которых название серии и идентификатор эксперимента, в рамках которого она проведена, совпадают с названием и идентификатором другой записи. В начале работы функция осуществляет запрос к базе на выгрузку идентификатора эксперимента с названием `experiment_name` из таблицы `list_of_experiments`. Элемент полученного массива с индексами (0,0) преобразуется

к числу и записывается в переменную `counter_of_experiments`. Далее осуществляется запрос к базе на выгрузку всех названий серий из таблицы `list_of_series`. Проходя в двойном цикле по всем элементам полученного массива, ищутся совпадения тех названий, что уже имелись в таблице, со строкой `series_name`. В случае совпадения поднимается флаг `flag_new`. В этом случае осуществляется запрос к базе на выгрузку идентификаторов эксперимента из таблицы `list_of_series` из записей, названия экспериментов которых совпадают с `name_of_the_experiment` и, в случае совпадения этих идентификаторов с `counter_of_experiments` поднимается еще один флаг (другими словами пользователь пытается загрузить серию измерений, которая была уже загружена в рамках этого эксперимента). В этом случае работа функции завершается. Если же идентификаторы не совпали, то при помощи механизма, аналогичному тому, что был в функции `create_a_record_of_the_experiment()`, вычисляется идентификатор для загружаемой серии измерений. После чего осуществляется запрос к базе на вставку в таблицу `list_of_series` серии измерений [17]:

```
sprintf(query, "insert into
list_of_series values ('%s', %d, %d)", series_name,
counter_of_series, counter_of_experiments );
res = PQexec(conn, query);
```

и запускается функция `add_records_of_the_measurement(counter_of_series, counter_of_experiments)`, осуществляющая создание записи об измерениях в таблице `list_of_measurement`. Функция получает на вход идентификаторы серии и эксперимента, в рамках которых проводились измерения. Перед описанием работы функции стоит напомнить, что измерениями считаются листы загружаемого файла `.xls`. Эта функция вставит в таблицу несколько записей (столько, сколько есть листов у файла `.xls`). Для первой вставляемой записи идентификатор измерения определяется по тому же механизму, по которой определялся идентификатор эксперимента. Далее запускается цикл по всем элементам массива `Cells`. На первом шаге осуществляется запрос к базе на создание записи в таблице с именем листа, записанным `Cells[0].sheet`, идентификатором, найденным до этого и записанным в `Cells[0].sheet_id`, и переданными в функцию идентификаторами серии и эксперимента [17].

```
sprintf(query, "insert into list_of_measurement values ('%s', %d , %d, %d)",
Cells[i].sheet, Cells[i].sheet_id,
counter_of_series, counter_of_experiments );
res = PQexec(conn, query);
```

Если же счетчик цикла не равен 0, происходит сравнение названия листа текущего элемента с названием листа у предыдущего элемента. В случае, если они отличаются друг от друга, осуществляется новая запись в таблицу `list_of_measurement`. Перед этим в параметр `sheet_id` текущего листа записывается идентификатор, равный `counter_of_measurement+1`. После выполнения функции идентификаторы добавленных измерений в таблице `list_of_mesurement` совпадают с параметрами `sheet_id` элементов массива `Cells`.

Далее свою работу начинает одна из самых важных функций - `create_records_in_measurement()`. Она создает новые записи в таблице `measurement`. Стоит напомнить, что каждая строка в таблице `measurement` соответствует одной ячейке загруженной таблице (соответственно, и выгруженной тоже), поэтому именно в этой функции происходит загрузка в базу информации, находящейся в таблицах файла `excell`. Текст этой функции приводится в разделе Приложение. В каждой строке таблицы `list_of_mesurement` есть позиции `type` (тип хранящейся в ячейке информации), `x` (номер строки в таблице `excell`, на которой находится чейка), `y` (номер столбца в таблице `excell`, на котором находится ячейка), `id` (номер листа, на котором находится ячейка), `numeric` (числовое содержимое ячейки, если оно есть), `string` (строковое содержимое ячейки, если оно есть). Номер листа соответствует идентификатору измерения, которому принадлежит данная ячейка. Запускается цикл от 0 до `real_counter` (которая объявлена глобально и является размером массива `Cells`). На каждом шаге цикла формируется запрос на вставку записи с информацией по всем перечисленным позициям. Если параметр `type` элемента массива `Cells` равен 1, значит у этого элемента есть `numeric_value`, но нет `string_value`. Формируется запрос `query` со всеми необходимыми членами. Поскольку в PostgreSQL у чисел формата `double precision` не запятые, а точки, в `for` с конца строки к ее началу просматривается строка `query` с сформированным запросом. Первая попавшаяся запятая и будет запятой параметра `numeric_value` элемента массива `Cells`. она меняется на точку. После чего выполняется запрос. Колонка `string` остается незаполненной. Если же значение параметра `type` равно 2, значит у элемента массива `Cells` есть `string_value`, но нет `numeric_value`. В этом случае формируется запрос на создание записи со всеми необходимыми параметрами (перечисленными выше), но колонка `numeric` остается незаполненной.

На этом загрузка файла в базу считается завершенной. Функция `on_pushButton_clicked()` завершает свою работу.

Выгрузка списка существующих экспериментов из базы

Для просмотра списка существующих в базе экспериментов, пользователю необходимо нажать на кнопку «Показать список экспериментов». С нажатием кнопки начинает свою работу функция `on_pushButton_4_clicked()`. Эта функция начинает свою работу с вызова функции `upload_list_of_experiments()`, описанной в файле `functional_for_work_with_database()`. Эта функция осуществляет запрос к базе на выгрузку всей информации из таблицы `list_of_experiments`:

```
sprintf(query, "SELECT * FROM list_of_experiments;");
res = PQexec(conn, query);
```

Далее выделяется память под массив `main_table`, каждый элемент которого является экземпляром класса `Cell_off_main_table`. Этот массив был объявлен глобально. Его размер - это произведение количества строк и количества ячеек выгруженной таблицы (`PQnfields(res) * PQntuples(res)`). Далее в двойном цикле по строкам и полям выгруженной таблицы в параметр `string_value` каждого элемента массива `main_table` копирует содержимое полей таблицы (`PQgetvalue(res, i, n)`), а номер строки и столбца ячейки (параметры `x` и `y` элемента массива) даются номера

строки и поля соответственно. По окончании циклов работа функции `upload_list_of_experiments()` завершается. Продолжается работа функции `on_pushButton_4_clicked()`. В созданное заранее `tableview` транслируется модель, созданная с помощью `QStandardItemModel`, у которой содержимое ячейки - это `string_value` элемента массива `main_table`, а координаты ячейки - это `x` и `y` элемента массива `main_table`.

Выгрузка списка существующих серий измерений, проведенных в рамках эксперимента

По идентификатору эксперимента пользователь может узнать названия серий измерений, проведенных в рамках этого эксперимента. Для этого он должен ввести в специальный спин-бокс идентификатор эксперимента и нажать на кнопку «Показать список серий». По нажатию этой кнопки запускается функция `on_pushButton_3_clicked()`. Считав идентификатор эксперимента из спин-бокса:

```
int id_of_the_experiemnt = ui -> spinBox_3-> value();
```

После чего запускается функция `upload_list_of_the_series(id_of_the_experiemnt)`, описанная в файле `functional_for_work_with_database.cpp`. Функция получает на вход идентификатор эксперимента, по которому нужно найти и выгрузить необходимые записи из таблицы `list_of_series`. Функция осуществляет запрос к базе на выгрузку из этой таблицы название и идентификаторо серий из записей, в которых идентификатор эксперимента равен тому, что был передан в функцию:

```
sprintf(query, "select name_of_the_series, id_of_the_series  
from list_of_series where id_of_the_experiment = %d", id_of_the_experiemnt );  
res = PQexec(conn, query);
```

Далее выделяется память под массив `main_table`, каждый элемент которого является экземпляром класса `Cell_off_main_table`. Размер, как и в прошлый раз - это произведение количества строк и количества ячеек выгруженной таблицы. Далее в двойном цикле по строкам и полям выгруженной таблицы в параметр `string_value` каждого элемента массива `main_table` копирует содержимое полей таблицы. Заполняются параметры `x` и `y` элемента массива `main_table`. По окончании циклов работа функции `upload_list_of_series()` завершается. Продолжается работа функции `on_pushButton_4_clicked()`. В созданное заранее `tableview` транслируется модель, созданная с помощью `QStandardItemModel`, у которой содержимое ячейки - это `string_value` элемента массива `main_table`, а координаты ячейки - это `x` и `y` элемента массива `main_table`.

Выгрузка списка существующих измерений, проведенных в рамках серии измерений эксперимента

По идентификаторам эксперимента и серии, введенным в специальные спин-боксы, пользователь может выгрузить список измерений, проведенных в рамках серии. Для этого он должен нажать на кнопку «Показать список измерений». Запустит-

ся функция `on_pushButton_2_clicked()`. Эта функция, считав из спин-бокса идентификаторы серии и эксперимента, запустит функцию `upload_list_of_measurement(id_of_the_experiment, id_of_the_series)`. Эта функция формирует и осуществляет запрос к базе на выгрузку из таблицы `list_of_measurement` названия измерения и их идентификаторы из записей, в которых идентификаторы серии измерений и эксперимента совпадают с теми, что были переданы функции на вход:

```
sprintf(query, "select name_of_the_measurement, id_of_the_measurement
from list_of_measurement where id_of_the_experiment
= %d and id_of_the_series = %d",
    id_of_the_experiment, id_of_the_series );
res = PQexec(conn, query);
```

Далее выделяется память под массив `main_table`, каждый элемент которого является экземпляром класса `Cell_off_main_table`. Размер, как и в прошлый раз - это произведение количества строк и количества ячеек выгруженной таблицы. Далее в двойном цикле по строкам и полям выгруженной таблицы в параметр `string_value` каждого элемента массива `main_table` копирует содержимое полей таблицы. Заполняются параметры `x` и `y` элемента массива `main_table`. По окончании циклов работа функции `upload_list_of_series()` завершается. Продолжается работа функции `on_pushButton_4_clicked()`. В созданное заранее `tableview` транслируется модель, созданная с помощью `QStandardItemModel`, у которой содержимое ячейки - это `string_value` элемента массива `main_table`, а координаты ячейки - это `x` и `y` элемента массива `main_table`.

Выгрузка измерения

По идентификатору измерения пользователь может выгрузить из базы содержимое соответствующего листа файла `excell`. Для этого он должен ввести в соответствующий спин-бокс идентификатор измерения и нажать на кнопку «Показать измерение». Запустится функция `on_pushButton_5_clicked()`. Она запустит функцию `upload_measurement(id_of_the_measurement)` и по полученным данным выстроит таблицу, идентичную той, что была на листе файла `excell`. Поскольку эти функции наиболее важные, их текст приводится в разделе Приложение.

Функция запускает работу функции `upload_measurement(id_of_the_measurement)`, передав ей на вход идентификатор измерения, считанный до этого из спин-бокса. Функция осуществляет запрос к базе на выгрузку полей, содержащих тип содержимого ячейки, номера строки и столбца, на которых она находилась, строковые и числовые содержимое из тех записей, в которых идентификатор измерения равняется тому, что ввел пользователь. Далее выделяется память под массив, объявленный заранее глобально для измерений, выгруженных из базы - `database_val`. Его размер - это количество выгруженных строк таблицы (так как каждая строка - это одна ячейка, а одна ячейка - это один элемент массива, каждый элемент которого является экземпляром класса `cell`). Далее запускается двойной цикл по всем строкам и полям выгруженной таблицы. На каждом шаге цикла содержимое поля копируется в переменную `str`. Если счетчик цикла по полям строки равен 0, значит выгруженное поле содержит тип ячейки будущей таблицы, если 1 - то координату `x`, и 2 - координату

у, если 3 - числовое содержимое ячейки, если 4 - строковое содержимое. Поля type, x и y записываются в соответствующие параметры при помощи atoi. Если значение database[i].type равно одному, значит нужно заполнять numeric_value. Для этого в полученной строке последняя точка меняется на запятую. Строка преобразовывается в число при помощи atof. Если же значение database[i].type равно 2, значит на следующем шаге цикла произойдет копирование содержимого строки в database_val[i].string_value. После чего продолжается работа функции on_pushButton_5_clicked(). В качестве размера будущей таблицы выбираются максимальные значения параметров x и y среди элементов массива database_val. Далее, при помощи QStandardItemModel строится модель таблицы с заданными размерами. В зависимости от значения параметра type элементов массива database_val, в ячейку копируется либо параметр numeric_value, либо string_value. Номером строки и столбца выступают соответственно параметры x и y массива. Далее построенная таблица транслируется в tableView_4. Пользователь выгружает таблицу, идентичную той, что была на листе файла excell. Двигаясь по номерам измерений, пользователь получает информацию, идентичную той, которую бы он получал, двигаясь по листам файла excell.

Модуль для графического отображения результатов

Пользователь может построить двумерный график только в том случае, если выгрузил какое-либо измерение. Для этого он должен ввести в специально отведенные для этого спин-боксы номера столбца, содержащих значения координат x и y точек графика, и нажать на кнопку «Построить график». После нажатия запустится функция on_pushButton_7_clicked(). Текст этой функции приводится в разделе Приложение. Функция начинает свою работу со считывания номеров столбцов из спин-боксов. Далее, по скольку в столбце таблицы, помимо чисел, могут быть ячейки, содержащие строки, запускается цикл по всем элементам массива database_val. В этом цикле вычисляется количество всех элементов массива database_val, у которых значение параметра type равно 1 (ячейка содержит число) и параметр x равен тому, что было считано из соответствующего спин-бокса. Полученное число является размером вектора, в который будут сложены числа из нужного столбца таблицы. Для вычисления размера вектора, содержащего в себе числа из столбца, который пользователь выбрал для значений координаты у точек графика осуществляются аналогичные действия. Объявив два вектора нужного размера, начинается их заполнение. В них записываются значения параметра numeric_value тех элементов массива database_val, у которых значение параметра type равно 1, а значение параметра x (или y, в зависимости от вектора), равно тому, что пользователь ввел в соответствующий спин-бокс. В процессе записи информации в векторы, происходит поиск минимального и максимального значения параметров x и y. Построение графика осуществляется с помощью библиотеки qcustomplot [14]. Создается график, добавляются данные (векторы x и y). Задаются размеры осей (при помощи максимальных значений x и y). После чего вызывается функция replot() и пользователь видит в специально отведенном для этого widget построенный им график.

Переход от интерфейса базы к интерфейсу модели

Для того, чтобы перейти от интерфейса для базы данных к интерфейсу для программного обеспечения, моделирующего эффект Холла, пользователь должен нажать на кнопку «Перейти к построению модели». После нажатия кнопки запустится функция `on_pushButton_6_clicked()`. Эта функция осуществляет системный вызов на запуск проекта ModelingHoll3, описанный выше. После чего пользователь может пользоваться интерфейсом для построения модели.

Демонстрация результатов

Демонстрация работы базы, интерфейса и графического модуля

Итак, введя название эксперимента CdTe +In 06-56(16) пользователь нажимает на кнопку «Загрузить серию измерений в базу» и в открывшемся меню выбирает нужный файл:

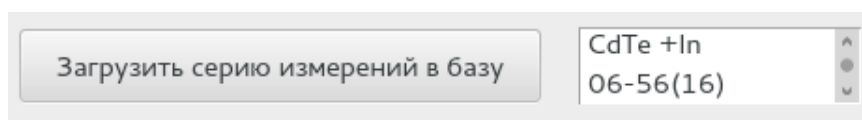


Рис. 14. Загрузка файла в базу

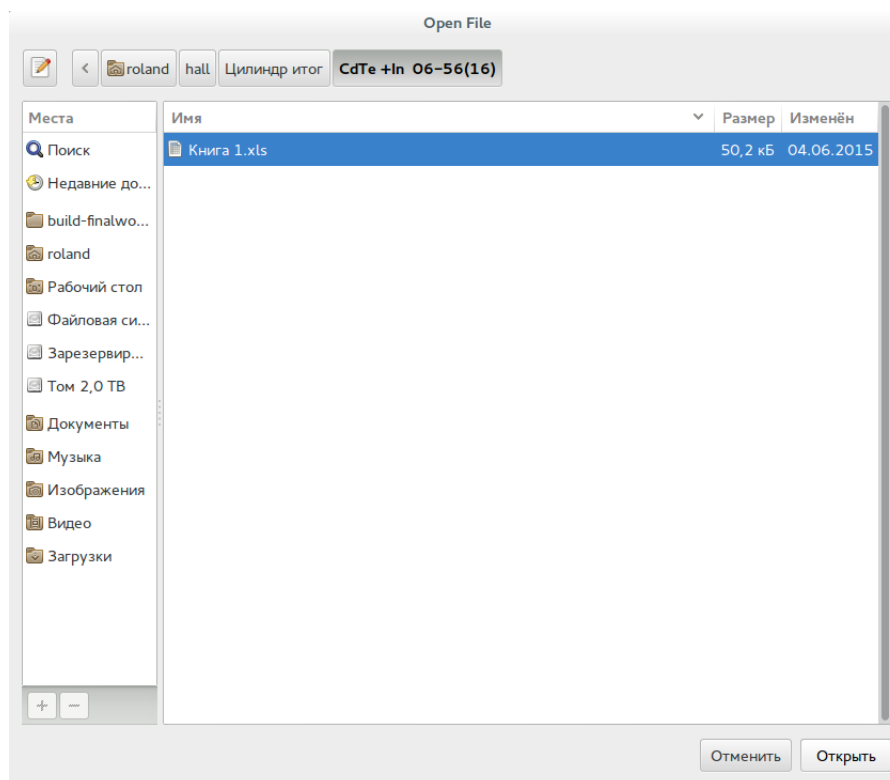
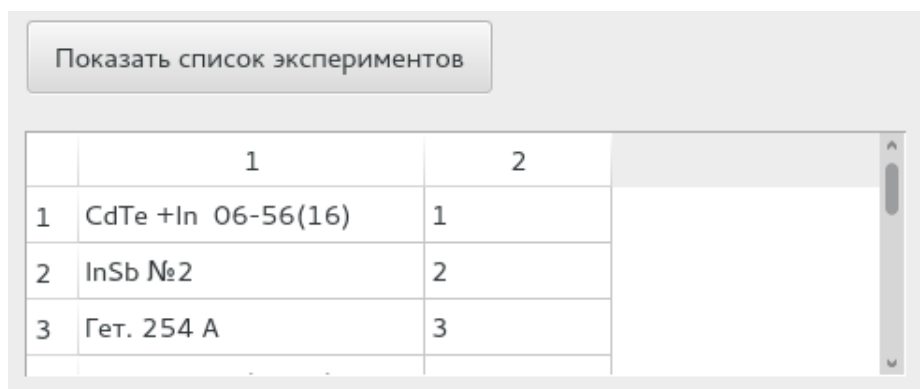


Рис. 15. Выбор загружаемого файла

Выделив нужный файл и нажав на кнопку «Открыть», пользователь начинает загрузку файла в базу. После нажатия на кнопку «Показать список экспериментов» можно увидеть, что в списке экспериментов появился созданный только что эксперимент:

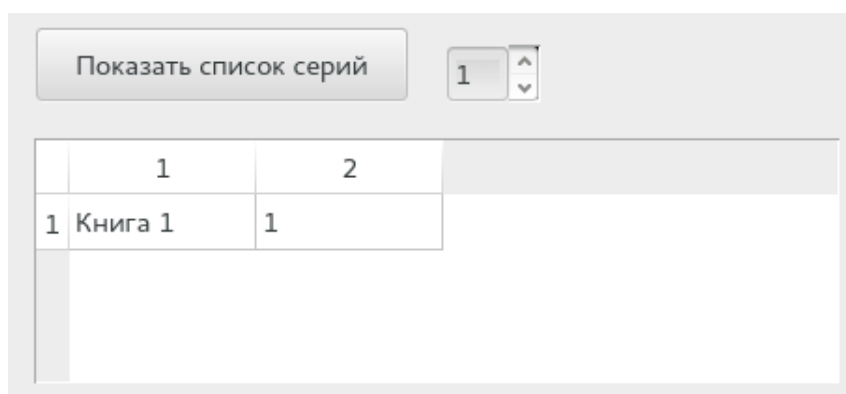


The screenshot shows a button labeled "Показать список экспериментов" and a table with three columns. The first column contains an index, the second contains the experiment name, and the third contains an identifier. The table has three rows of data.

	1	2
1	CdTe +In 06-56(16)	1
2	InSb №2	2
3	Гет. 254 А	3

Рис. 16. Список экспериментов базы

Названию эксперимента CdTe +In 06-56(16) соответствует идентификатор 1. Введя его в спин-бокс рядом с кнопкой «Показать список серий», можно увидеть список серий, проведенных в рамках этого эксперимента. Название серии соответствует названию загруженного файла:



The screenshot shows a button labeled "Показать список серий", a spinner box with the value 1, and a table with three columns. The first column contains an index, the second contains the series name, and the third contains an identifier. The table has one row of data.

	1	2
1	Книга 1	1

Рис. 17. Список серий измерений

Зная идентификатор серии (он написан справа от названия) и идентификатор эксперимента, можно выгрузить список измерений, проведенных в рамках серии (список листов файла excell):

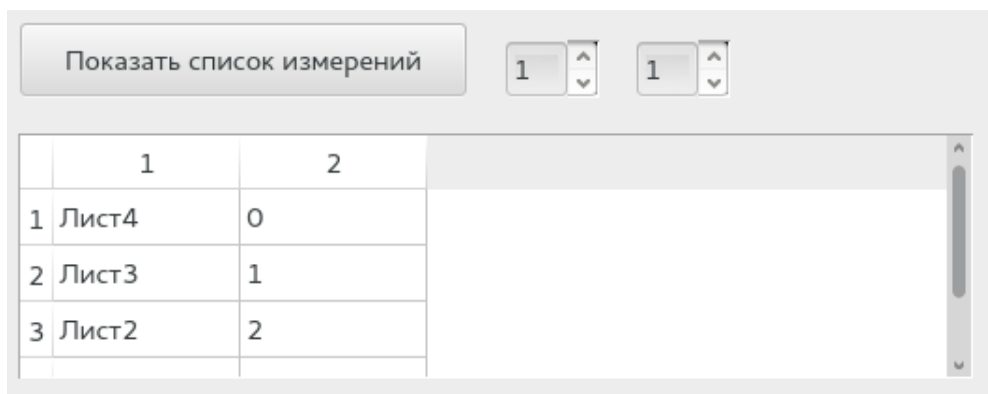


Рис. 18. Список измерений

Введя идентификатор измерений, можно увидеть содержимое листа файла excel:

	1	2	3	4	5	6	7	8	
1	Теллурид к...							Книга 1, ли...	
2	Образец 0...								
3	Поворот в ...								
4	Температу...								
5	Ток через о...								
6		Угол а,		U1,		Угол а,		U2,	(U1 - U
7	№ точки	градусы	Sina	мВ	№ точки	градусы	Sina	мВ	мВ
8	1	0	0	0.00375	31	180	0	0.00525	-0.000
9	2	6	0.1	-0.08475	32	186	-0.1	0.091	-0.087
10	3	12	0.21	-0.16825	33	192	-0.21	0.1815	-0.174
11	4	18	0.31	-0.25975	34	198	-0.31	0.262	-0.260
12	5	24	0.41	-0.3405	35	204	-0.41	0.3375	-0.339
13	6	30	0.5	-0.41225	36	210	-0.5	0.41825	-0.419
14	7	36	0.59	-0.4775	37	216	-0.59	0.48825	-0.482
15	8	42	0.67	-0.54675	38	222	-0.67	0.5525	-0.549

Рис. 19. Выгруженное из базы измерение

Введя в спин-боксы рядом с кнопкой «Построить график» в качестве координат у колонку с номером 9, а в качестве координат x колонку с номером 2, можно получить следующую зависимость:

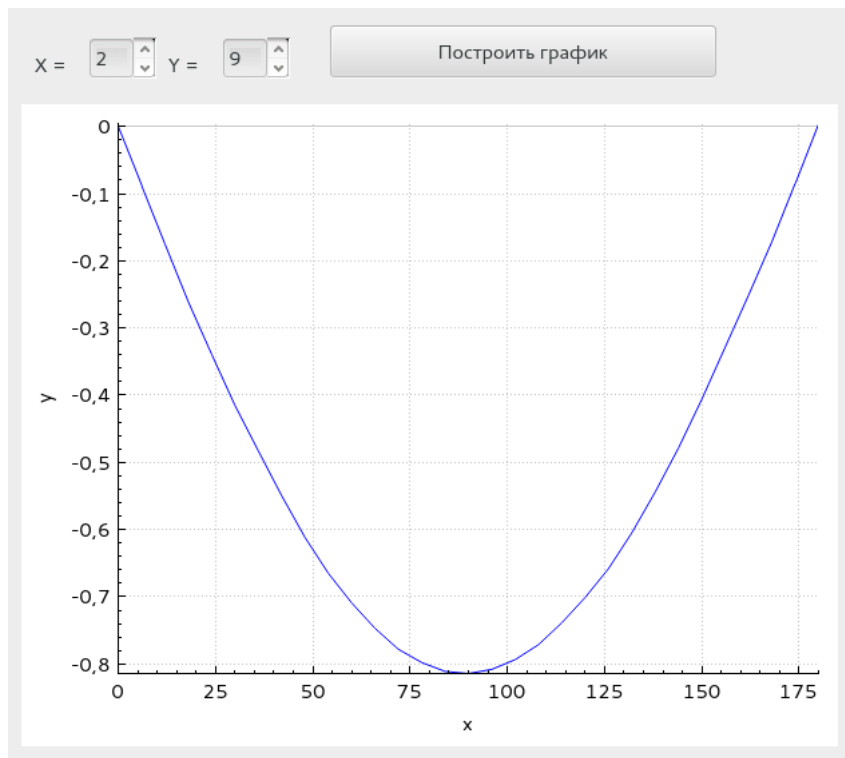


Рис. 20. Графический модуль для демонстрации результатов

Общий вид продемонстрированной процедуры:

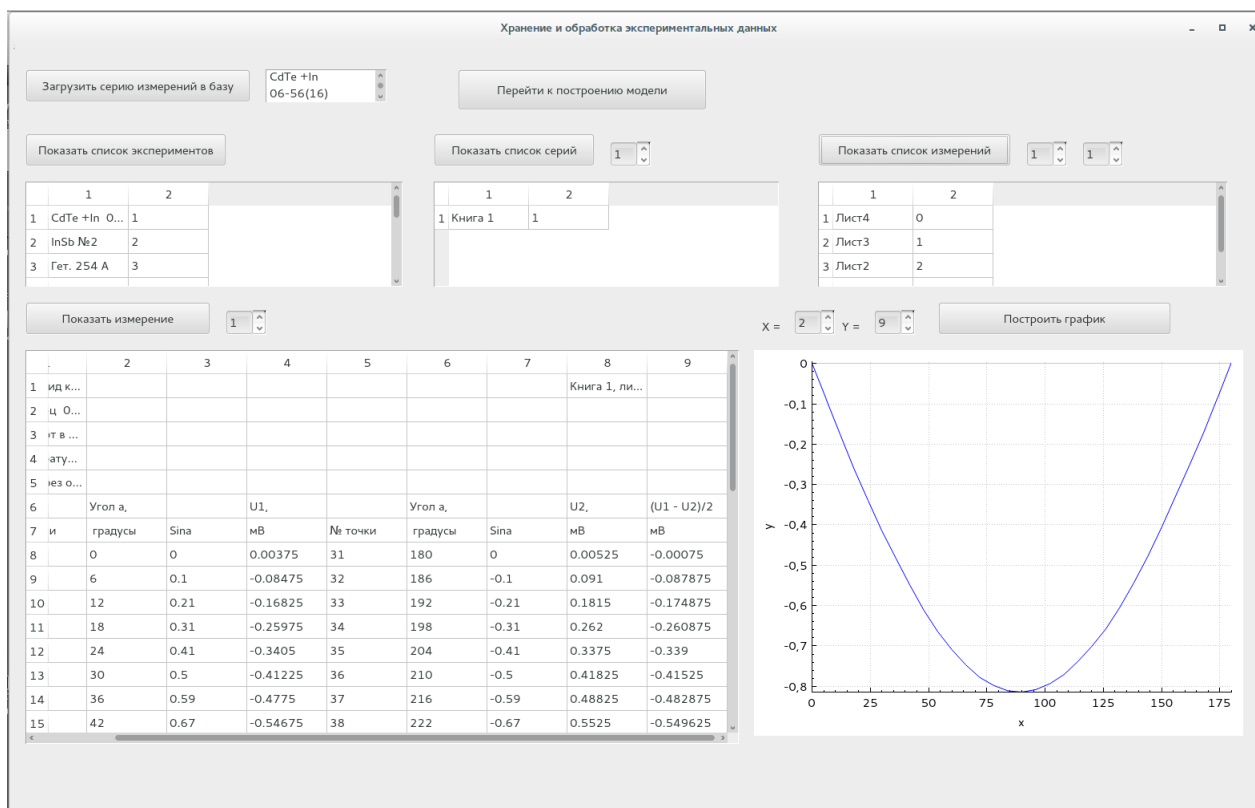


Рис. 21. Внешний вид интерфейса для работы с базой

Как уже говорилось ранее, для того, чтобы перейти к построению модели, необходимо нажать на кнопку «Перейти к построению модели»:

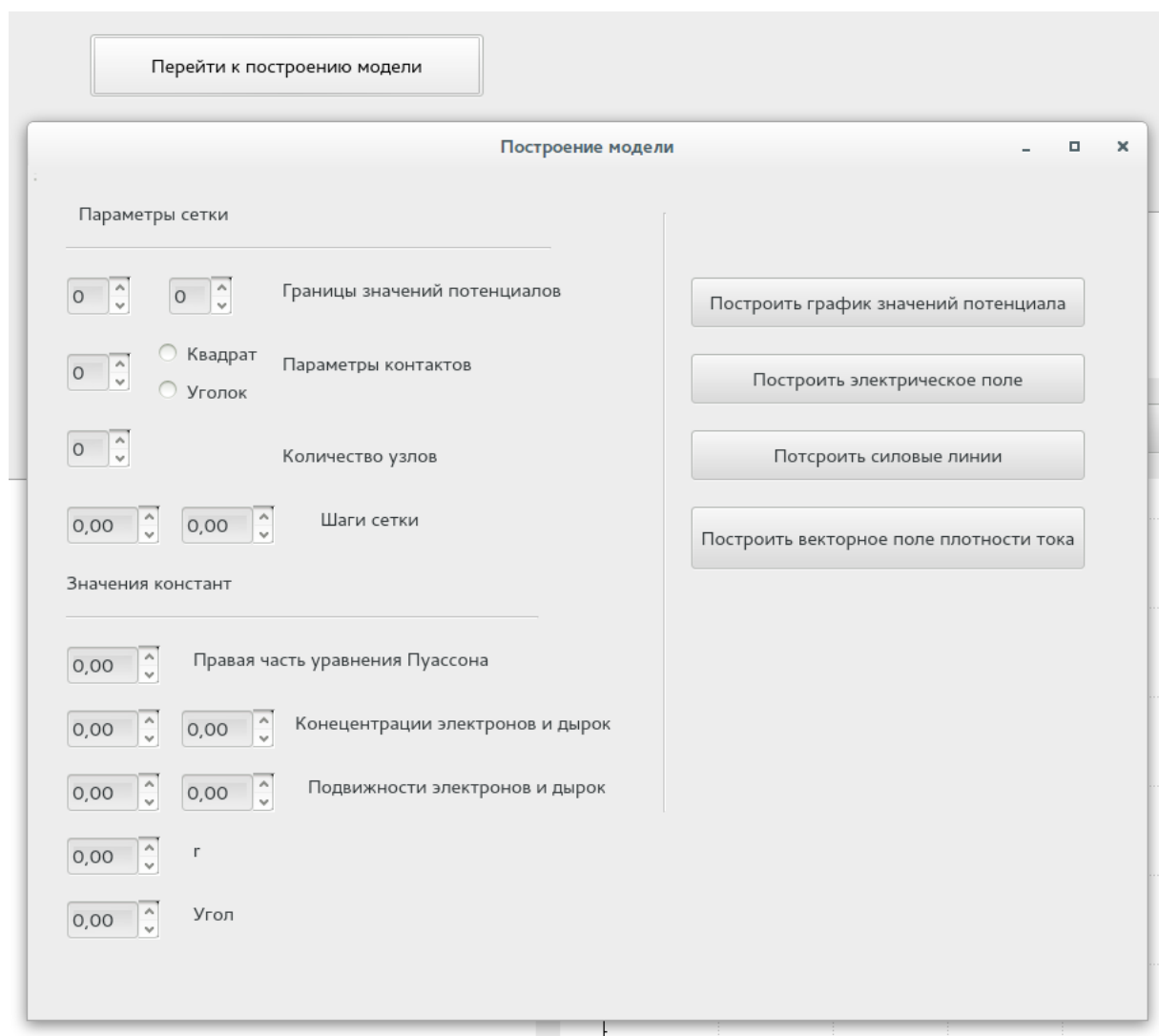


Рис. 22. Переход от работы с базой к работе с моделью

Демонстрация работы модели

Задав параметры сетки и значения констант, пользователь может увидеть график значений потенциала, векторное поле градиента, силовые линии электрического поля, векторное поле плотности тока. Для удобства, изменение электродвижущей силы Холла при изменении граничных условий будет приведено в конце этого раздела. Так, например:

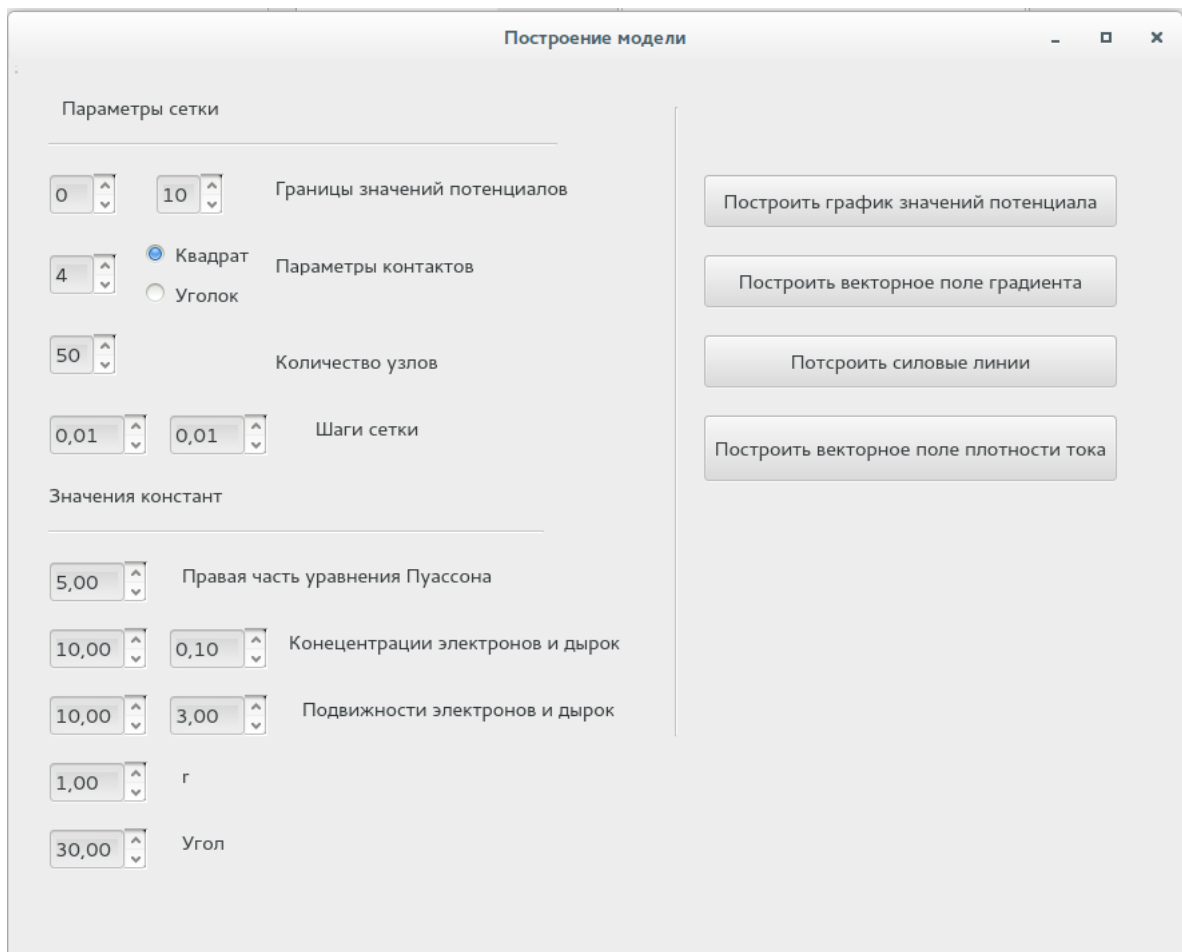


Рис. 23. Внешний вид интерфейса для работы с моделью

Если при таких параметрах сетки пользователь последовательно нажмет на кнопки «Построить график значений потенциала», «Построить векторное поле градиента», «Построить силовые линии» и «Построить векторное поле плотности тока», он получит следующий результат:

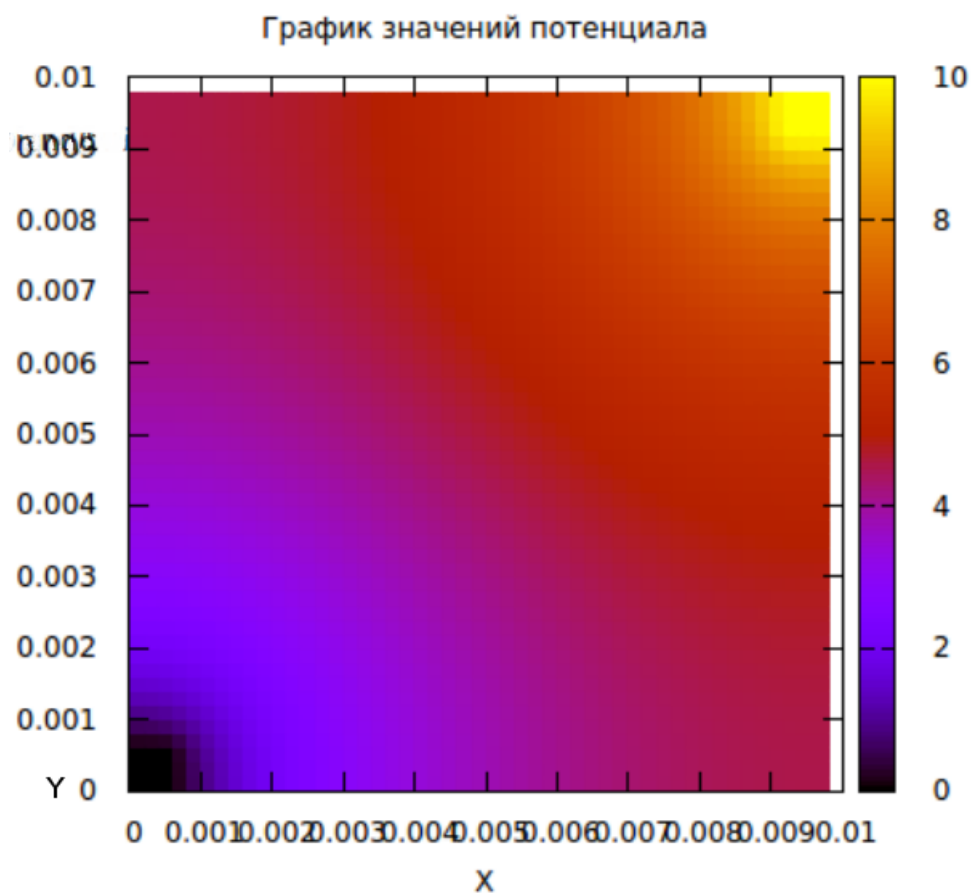


Рис. 24. График значений потенциала

Как видно на картинке, значения потенциала растут от левого нижнего угла в правый верхний угол, от минимального значения к максимальному. Значения потенциала как бы обволакивают максимальное значение, постепенно к нему приближаясь. При этом в левом нижнем и правом верхнем углах четко видны черный и желтый квадраты соответственно, демонстрирующие расположение и форму контактов.

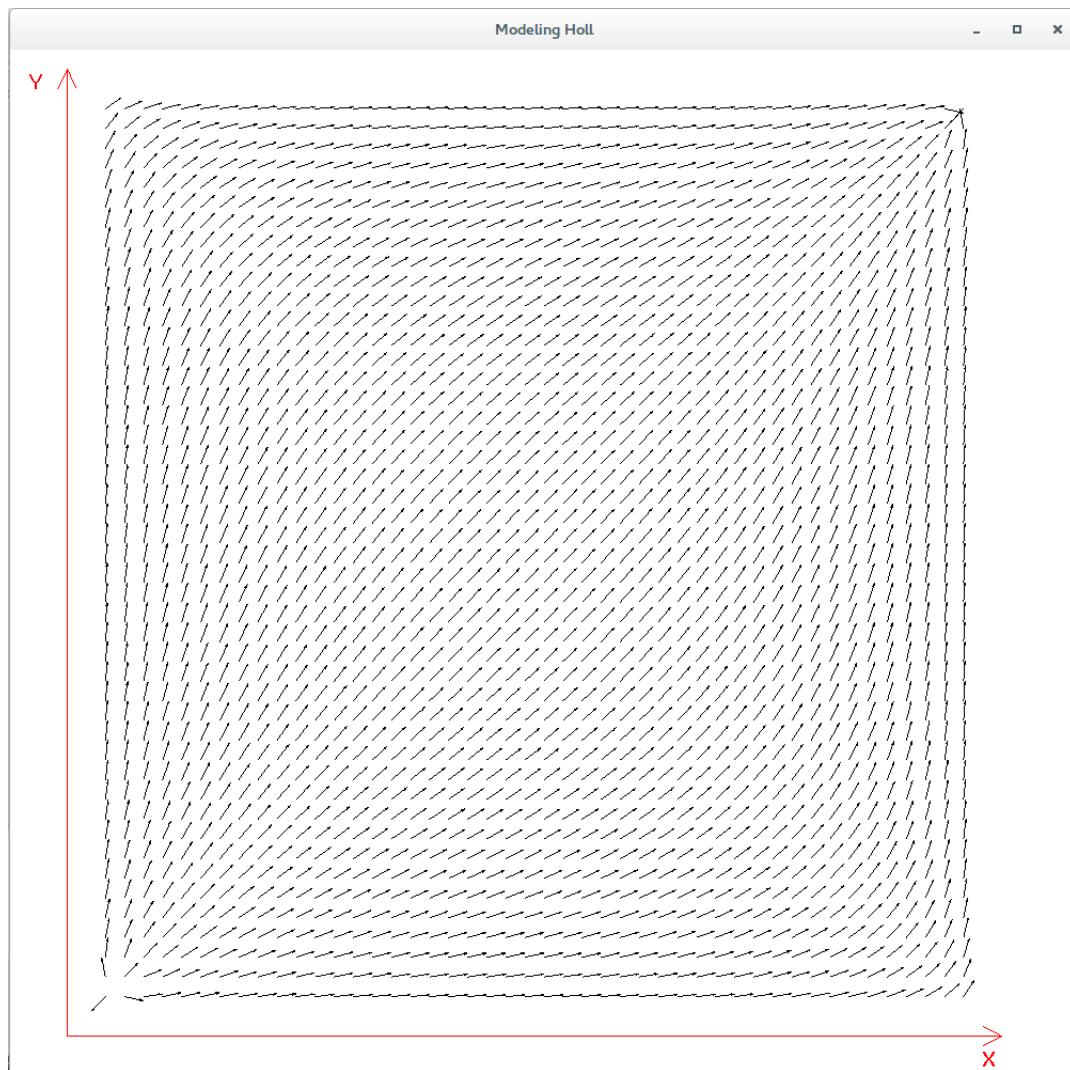


Рис. 25. Векторное поле градиента

На этой картинке видно, как векторы градиента направлены от левого нижнего угла к правому верхнему. Они образуют некие дуги по направлению роста значений потенциала.

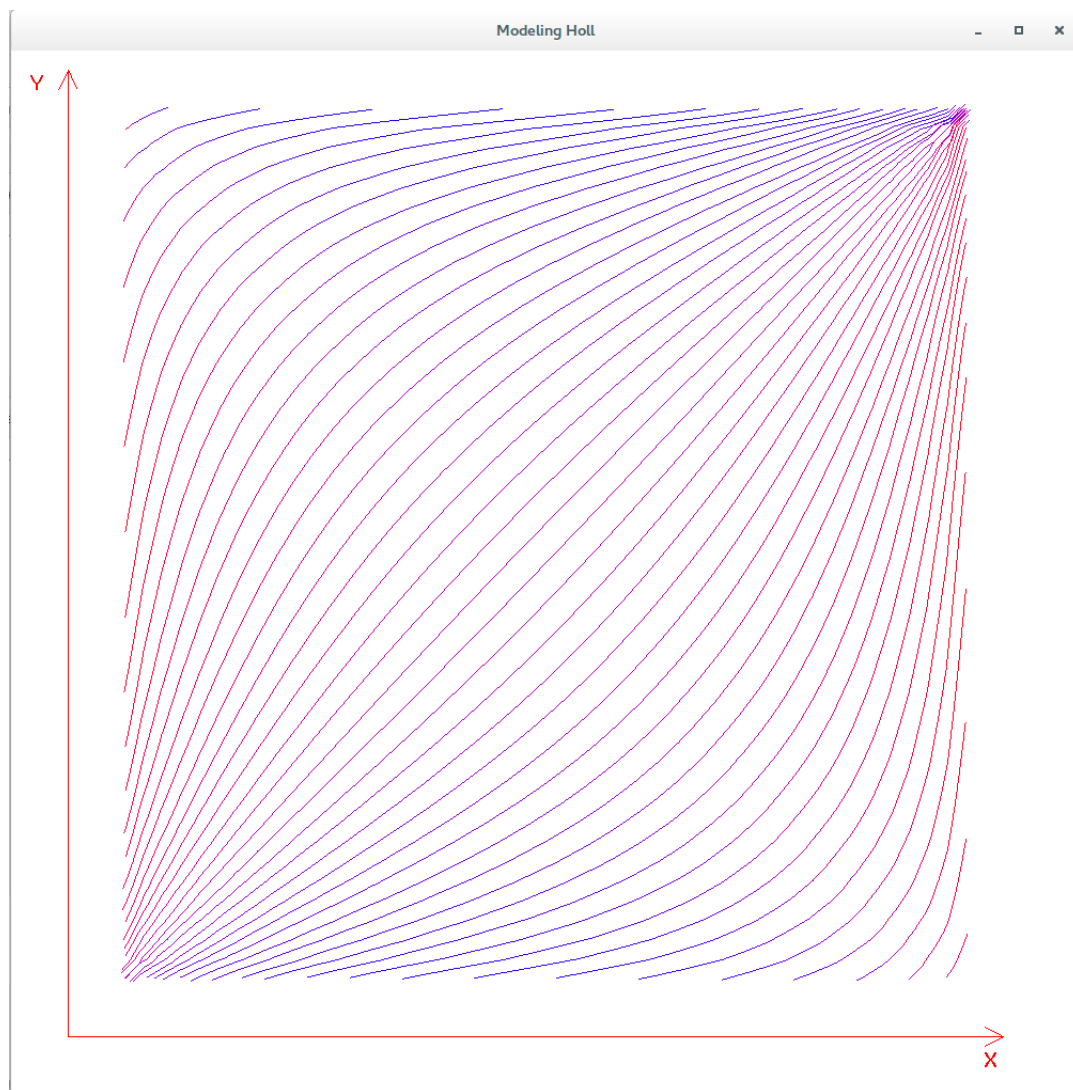


Рис. 26. Силовые линии электрического поля

Силовые линии строятся по направлению векторов градиента. Соответственно, они практически идентичны дугам, которые образуют векторы градиента. Они также растут от левого нижнего к правому верхнему углу.

При увеличении площади контактов, результаты изменятся. Так, например:

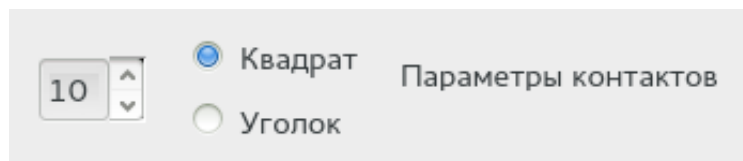


Рис. 27. Параметры контактов

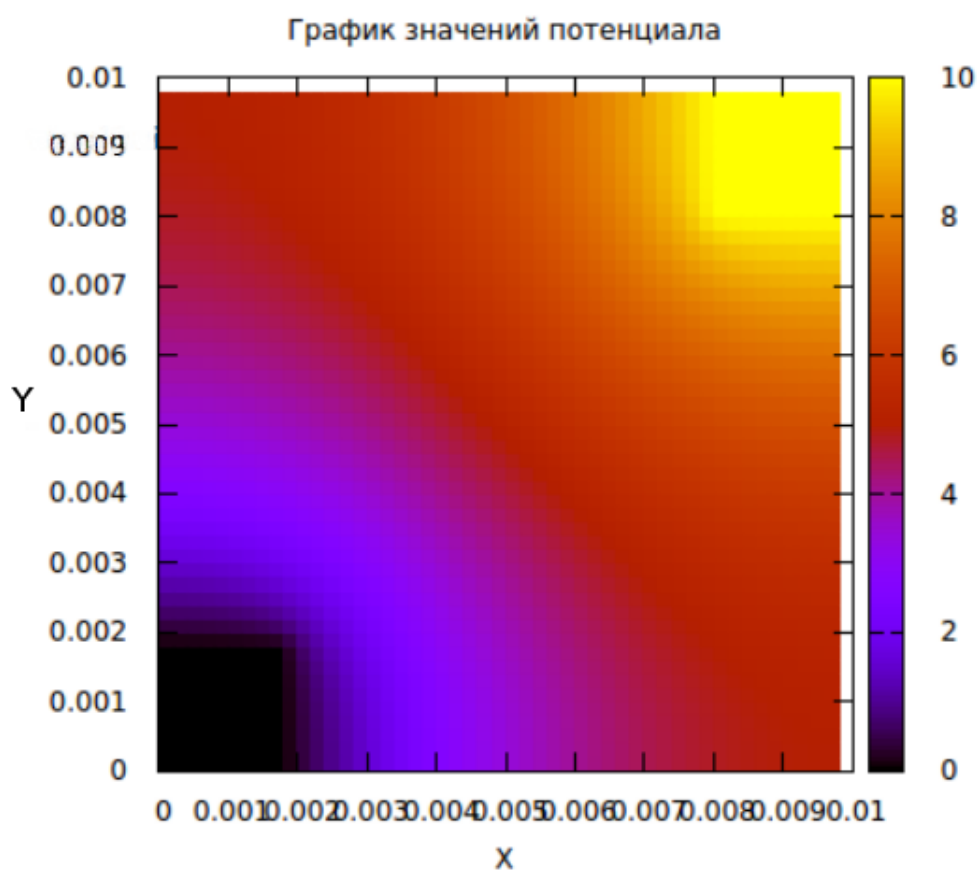


Рис. 28. Влияние изменения размера контактов

На графике видны ярковыраженные квадраты контактов. Тем не менее, характеристики графика потенциала сохранены, а значит уравнение Пуассона решено правильно.

Если поменять форму контактов с квадратов на уголки, можно получить следующий результат:

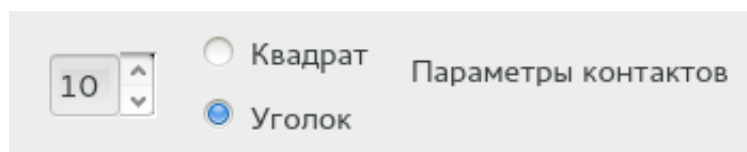


Рис. 29. Параметры контактов

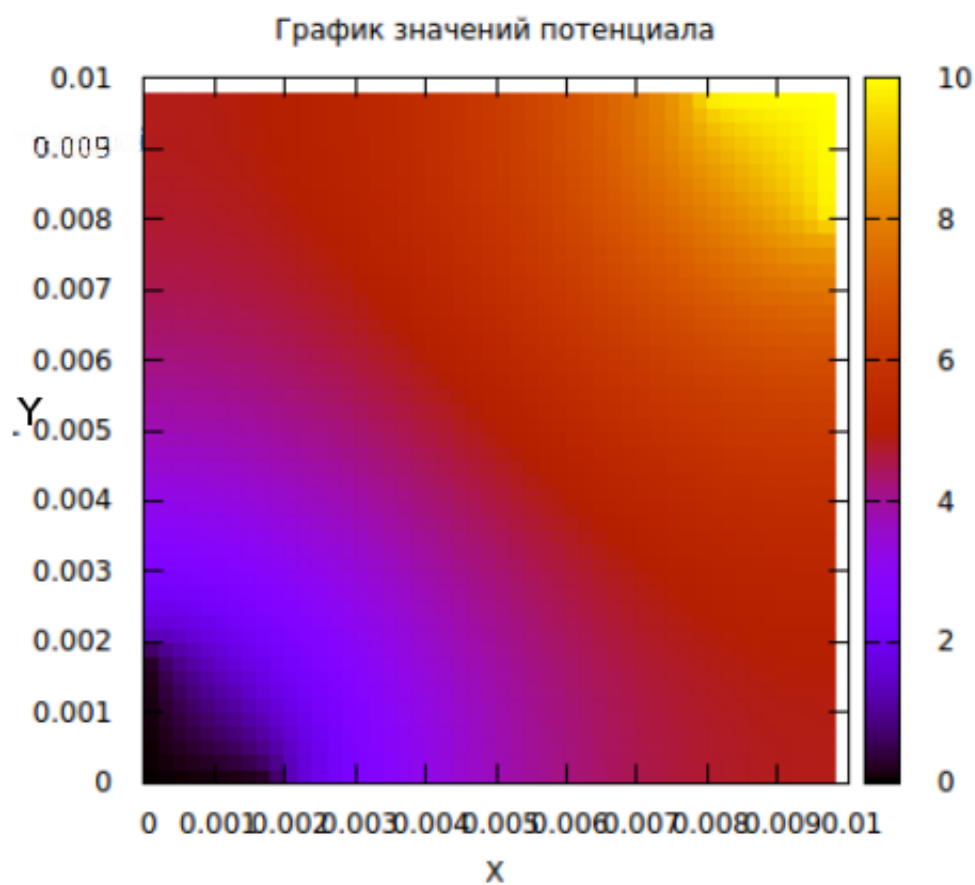


Рис. 30. Влияние изменения формы контактов

В левом нижнем и правом верхнем углах графика видны ярковыраженные углы.

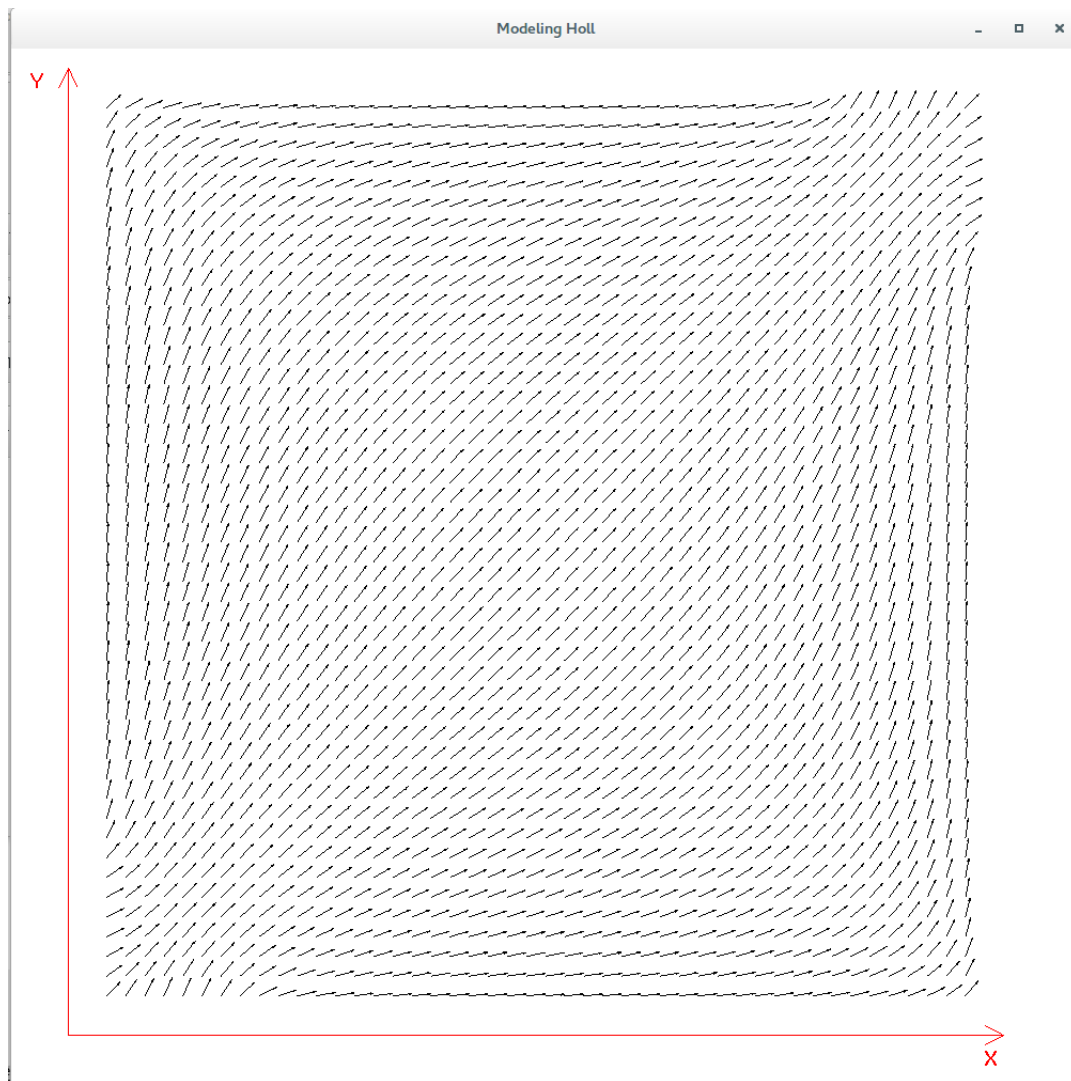


Рис. 31. Влияние изменения формы контактов

При изменении формы контактов на уголки, векторы в углах поворачиваются в правильном направлении, так как меняются значения потенциалов.

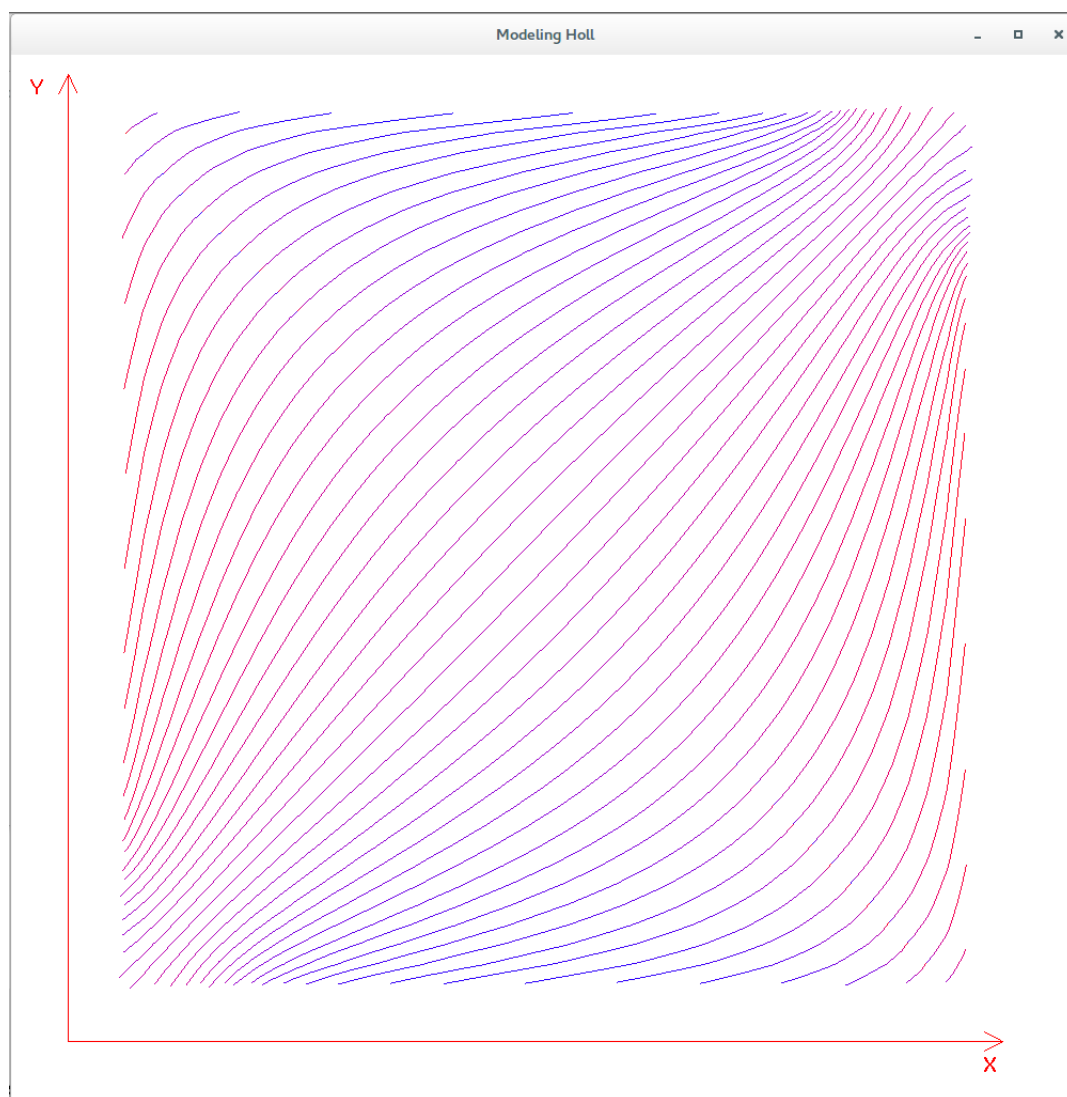


Рис. 32. Влияние изменения формы контактов

Благодаря изменению формы контактов на уголки, концы дуг сходятся не на правом верхнем углу в целом, а на правой верхней точке. Аналогичная ситуация наблюдается и в левом нижнем углу сетки. Эта точка является вершиной угла, являющегося контактом. Здесь и далее электродвижущая сила обозначается буквой e

Изменение электродвижущей силы Холла при изменении граничных условий

0 10 Границы значений потенциалов

4 Квadrat Уголок Параметры контактов

The interface shows two rows of controls. The first row has two numeric input fields with up/down arrows, containing '0' and '10', followed by the text 'Границы значений потенциалов'. The second row has a numeric input field with '4', followed by two radio buttons labeled 'Квadrat' and 'Уголок', and the text 'Параметры контактов'. The 'Уголок' radio button is selected.

Рис. 33. $e_1 = 3.651646e + 10$

0 10 Границы значений потенциалов

4 Квadrat Уголок Параметры контактов

The interface is similar to the previous one, but the 'Квadrat' radio button is selected and highlighted with a dashed border. The 'Уголок' radio button is unselected.

Рис. 34. $e_2 = 4.525149e + 10$

0 10 Границы значений потенциалов

8 Квadrat Уголок Параметры контактов

The interface shows the numeric input field now contains '8'. The 'Уголок' radio button is selected.

Рис. 35. $e_3 = 5.120007e + 10$

0 12 Границы значений потенциалов

8 Квadrat Уголок Параметры контактов

The interface shows the top numeric input field now contains '12'. The 'Уголок' radio button is selected.

Рис. 36. $e_4 = 6.144008e + 10$

Ниже приводится таблица, в которой отображено отношение каждой новой величины ЭДС относительно первого случая граничных условий.

Таблица 3. Таблица изменений ЭДС Холла

Граничные условия	Изменение ЭДС Холла
Минимальное значение потенциала = 0 Максимальное значение потенциала = 10 Форма контакта - Квадрат Длина ребра квадрата - 4	$e_2 = 1,239208017 * e_1$
Минимальное значение потенциала = 0 Максимальное значение потенциала = 10 Форма контакта - Уголок Длина ребра уголка - 8	$e_3 = 1,40210935 * e_1$
Минимальное значение потенциала = 0 Максимальное значение потенциала = 12 Форма контакта - Уголок Длина ребра уголка - 8	$e_4 = 1,682531111 * e_1$

7. Выводы

В результате численных расчетов удалось показать, что ЭДС Холла зависит от величины и формы контактов. Было разработано полноценное приложение с полноценным интерфейсом, с помощью которого можно вычислять градиент электрического поля, силовые линии, электродвижущую силу Холла. Была составлена база данных на базе СУБД PostgreSQL с полноценным интерфейсом и графическим модулем для графического отображения различных данных. Приложение позволяет с помощью интерфейса загружать файлы формата .xls в базу, просматривать списки экспериментов, серий измерений (для каждого эксперимента), измерений (для каждой серии в рамках проекта), выводить содержимое измерения, строить графики по значениям из столбцов, и переходить к построению модели. При построении модели приложение позволяет интерактивно задавать параметры сетки, а именно:

- минимальное и максимальное значения потенциалов;
- форму и размер контактов;
- количество узлов сетки;
- шаги сетки;

а также значения констант, а именно:

- значение правой части уравнения Пуассона;
- концентрации электронов и дырок;
- подвижности электронов и дырок;
- значение коэффициента r ;
- размер угла между вектором индукции магнитного поля и напряженностью электрического поля

и по введенным данным строить график значений потенциала, векторное поле, состоящее из векторов градиента, силовые линии электрического поля, и производить расчет электродвижущей силы Холла.

Список литературы и интернет-ресурсов

- [1] Шалимова К.В. *Физика полупроводников*. — М.: Энергоатомиздат, 1985
- [2] Бонч-Бруевич В.Л., Калашников С.Г. *Физика полупроводников*. — М.: Наука, 1977
- [3] Белова И.М. *Компьютерное моделирование* — М.: МГИУ, 2007
- [4] Куприянов Д.Ю. *Использование библиотеки OpenGL. Моделирование трехмерной сцены* — М.: МГИУ, 2012
- [5] Фаулер М. *UML. Основыю 3 изд* —
- [6] D. E. Knuth *The TEX book. Addison-Wesley* — 1993
- [7] Львовоский С.М. *Набор и верстка в системе LATEX* — М. МЦНМО 2003
- [8] Павловская Т.А. *C/C++ Программирование на языке высокого уровня* — П. 2012
- [9] Lee Phillips *Gnuplot Cookbook* — П. 2012
- [10] Шлее М. *Qt 4.8. Профессиональное программирование на C++* — П. 2012
- [11] Радыгин В.Ю. *Базы данных и СУБД* — М. 2011
- [12] <http://doc.crossplatform.ru/qt/> - Все о кроссплатформенном программировании
- [13] <http://doc.qt.io/> - Qt Documentation
- [14] <http://blabla.ru/> - Вывод графиков в Qt 5 при помощи QCustomPlot
- [15] <http://www.effects.ru/science> - Виртуальный фонд естественнонаучных и научно-технических эффектов «Эффективная физика»
- [16] <http://postgresql.ru.net/docs/develc.html> - Как программы на Си взаимодействуют с сервером БД PostgreSQL
- [17] <http://www.postgresql.org/> - Documentation
- [18] <http://libxls.sourceforge.net/>
- [19] https://ru.wikipedia.org/wiki/C_-_C++

8. Приложение

Функция `sor()`, реализующая метод последовательной верхней релаксации:

```
void sor(){
    int p=0, p1 = 1;
    int ipass, j, jsw, l, lsw, nn, r, phi;
    double anorm, resid, rjac;
    double anormf = 0.0, omega = 1.0;
    rjac = cos( 2*M_PI/n );
    for( x = 1; x < n - 1; x++ )
    for( y = 1; y < n - 1; y++ )
        anormf += fabs( f[x][y] );
    for(nn = 1; nn <= MAXITS; nn++) {
        anorm = 0.0;
        jsw = 1;
        for(ipass = 1; ipass <= 2; ipass++) {
            lsw = jsw;
            for(j = 1; j < n-1; j++) {
                for(l = lsw; l < n-1; l += 2) {
                    resid = a[j][l] * u[j+1][l]
                        + b[j][l] * u[j-1][l]
                        + c[j][l] * u[j][l+1]
                        + d[j][l] * u[j][l-1]
                        + e[j][l] * u[j][l]
                        - dx*dy*f[j][l];
                    anorm += fabs(resid);
                    u[j][l] -= omega * resid / e[j][l];
                }
            }
            for( x = 0; x < n; x++ ) {
                u[x][0] = u[x][1];
                u[x][n-1] = u[x][n-2];
            }
            for( y = 0; y < n; y++ ) {
                u[0][y] = u[1][y];
                u[n-1][y] = u[n-2][y];
            }
            if (form == 1){
                for (x=0; x<edge_length; x++){
                    for (y=0; y<edge_length; y++){
                        u[x][y] = potential1;
                    }
                }
                for (x=n-edge_length; x<n; x++){
                    for (y=n-edge_length; y<n; y++){
                        u[x][y] = potential2;
                    }
                }
            }
        }
    }
}
```

```

    }
}
if (form == 2){
    for (x=0; x<edge_length; x++){
        u[x][0] = potential1;
    }
    for (y=0; y<edge_length; y++){
        u[0][y] = potential1;
    }
    for (x=n-edge_length; x<n; x++){
        u[x][n-1] = potential2;
    }
    for (y=n-edge_length; y<n; y++){
        u[n-1][y] = potential2;
    }
}
lsw = 3 - lsw;
}
jsw = 3 - jsw;
omega = (n == 1 && ipass == 1 ? 1.0
/ (1.0 - 0.5 * rjac * rjac)
: 1.0 / (1.0 - 0.25 * rjac * rjac * omega));
}
if( anorm < EPS*anormf ) return;
}
}

```

Функция `calculate_current_density()`, вычисляющая компоненты вектора плотности тока и электродвижущую силу Холла.

```

void calculate_current_density(){
    ads = 0;
    e1 = 1.60217657e-19;
    int i=0;
    double const1 = e1*(electron_density
* electron_mobility + hole_density * hole_mobility);
    double const2 = r*e1*(hole_density*(hole_mobility * hole_mobility))
- electron_density*(electron_mobility * electron_mobility);
    for (x=2; x<n-2; x++){
        for (y=2; y<n-2; y++){
            cur_density_x[x][y] = const1 * (-ux[x][y]) -
            const2 * (ux[x][y] * (1*sin((alpha*3.14)/180)));
            cur_density_y[x][y] = const1 * (-uy[x][y]) - const2
            * (uy[x][y] * (1*sin((alpha*3.14)/180)));
            ads += (cur_density_x[x][y] + cur_density_y[x][y]);
        }
    }
}

```

```

    ads = ads*(sqrt(n*n + n*n));
    printf("ads = %e\n", ads);
}

void Display(void) {
    double uxn_new_min, uxn_new_max, uyn_new_min, uyn_new_max;
    double x_point=0, y_point=0, x_point_down = 0, y_point_down = 0;
    int x_new3, y_new3, x_new1, y_new1, x_new2, y_new2, x_new4, y_new4;
    int k=0, i=0;
    if (grafic !=3) glClearColor(1, 1, 1, 1);
    if (grafic == 3) glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glPushMatrix();
    glColor3f(1.0, 0.0, 0.0);
    glRasterPos2f(-2, n-2);
    glutBitmapString(GLUT_BITMAP_HELVETICA_18, "Y");
    glRasterPos2f(n-2, -1.5);
    glutBitmapString(GLUT_BITMAP_HELVETICA_18, "X");
    glBegin(GL_LINES);
        glVertex2d(0,0);
        glVertex2d(0,n-1);
        glVertex2d(0,0);
        glVertex2d(n-1,0);
        glVertex2d(0,n-1);
        glVertex2d(-0.5,n-2);
        glVertex2d(0,n-1);
        glVertex2d(0.5,n-2);
        glVertex2d(n-1,0);
        glVertex2d(n-2,0.5);
        glVertex2d(n-1,0);
        glVertex2d(n-2,-0.5);
    glEnd();
    glPopMatrix();
    if (grafic == 2){
        for(x=2; x<n-2; x++){
            for(y=2; y<n-2; y++){
                glPushMatrix();
                double fi;
                fi = (atan2(uy[x][y],ux[x][y]))*180/3.14;
                glTranslatef(x,y,0);
                glRotatef( fi, 0,0,1);
                arrow();
                glPopMatrix();
            }
        }
    }
}

```

```

if(grafic == 3){
    for(x=3; x<n-2; x++){
        for(y=3; y<n-2; y++){
            x_point = x;
            y_point = y;
            x_point_down = x;
            y_point_down = y;
            if (x==n-y-1){
                glPushMatrix();
                glPointSize(2);
                glBegin(GL_LINE_STRIP);
                glVertex2d(x_point,y_point);
                k=1;
                while ((x_point<n-3)&&(y_point<n-3) && i< M){
                    if (k==1){
                        k=2;
                        double uxn = ux[x][y]
                        /sqrt(ux[x][y]*ux[x][y]+uy[x][y]*uy[x][y]);
                        double uyn = uy[x][y]
                        /sqrt(ux[x][y]*ux[x][y]+uy[x][y]*uy[x][y]);
                        x_point = x_point+uxn*dl;
                        y_point = y_point+uyn*dl;
                        glColor3f(my_abs(uyn), 0, my_abs(uxn));
                        glVertex2d(x_point, y_point);
                    }
                    i++;
                    x_new3 = floor(x_point);
                    y_new3 = floor(y_point);
                    x_new1 = x_new3;
                    y_new1 = y_new3+1;
                    x_new2 = x_new3+1;
                    y_new2 = y_new3+1;
                    x_new4 = x_new3+1;
                    y_new4 = y_new3;
                    if ( (x_new1 > 2) && (x_new1 < n-2)
                        && (x_new2 > 2) && (x_new2 < n-2)
                        && (x_new3 > 2) && (x_new3 < n-2)
                        && (y_new1 > 2) && (y_new1 < n-2)
                        && (y_new2 > 2) && (y_new2 < n-2)
                        && (y_new3 > 2) && (y_new3 < n-2)){
                        double a = (x_point) - x_new1;
                        double b = y_new1 - (y_point);
                        int R = 0.01;
                        double ux_new =
                            (ux[x_new1][y_new1]*sqrt(a*a+b*b)
                             + ux[x_new2][y_new2]
                             *sqrt((R-a)*(R-a) + b*b)

```

```

+ ux[x_new3][y_new3]
*sqrt(a*a+(R-b)*(R-b))
+ ux[x_new4][y_new4]
*sqrt((R-a)*(R-a) + (R-b)*(R-b)) ) /4;
double uy_new =
(uy[x_new1][y_new1]*sqrt(a*a+b*b)
+ uy[x_new2][y_new2]*sqrt((R-a)*(R-a) + b*b)
+ uy[x_new3][y_new3]*sqrt(a*a+(R-b)*(R-b))
+ uy[x_new4][y_new4]*sqrt((R-a)*(R-a)
+ (R-b)*(R-b)) ) /4;
double uxn_new = ux_new
/sqrt(ux_new*ux_new+uy_new*uy_new);
double uyn_new = uy_new
/sqrt(ux_new*ux_new+uy_new*uy_new);
x_point = x_point+uxn_new*dl;
y_point = y_point+uyn_new*dl;
glColor3f(my_abs(uyn_new), 0, my_abs(uxn_new));
glVertex2d(x_point, y_point);
}
}
glEnd();
glPopMatrix();
glPushMatrix();
i=0;
glPointSize(2);
glBegin(GL_LINE_STRIP);
glVertex2d(x_point_down ,y_point_down);
while((x_point_down > 3)&&(y_point_down > 3) && i < M){
    i++;
    if (k==1){
        k=3;
        double uxn = -ux[x][y]
/sqrt(ux[x][y]*ux[x][y]+uy[x][y]*uy[x][y]);
        double uyn = -uy[x][y]
/sqrt(ux[x][y]*ux[x][y]+uy[x][y]*uy[x][y]);
        x_point_down = x_point_down+uxn*dl;
        y_point_down = y_point_down+uyn*dl;
        glColor3f(my_abs(uyn), 0.0, my_abs(uxn));
        glVertex2d(x_point_down, y_point_down);
    }
    x_new3 = floor(x_point_down);
    y_new3 = floor(y_point_down);
    x_new1 = x_new3;
    y_new1 = y_new3+1;
    x_new2 = x_new3+1;
    y_new2 = y_new3+1;
    x_new4 = x_new3+1;

```



```

        y_new4 = y_new3;
        if ( (x_new1 > 2) && (x_new1 < n-2)
            && (x_new2 > 2) && (x_new2 < n-2)
            && (x_new3 > 2) && (x_new3 < n-2)
            && (y_new1 > 2) && (y_new1 < n-2)
            && (y_new2 > 2) && (y_new2 < n-2)
            && (y_new3 > 2) && (y_new3 < n-2)){
            double a = (x_point_down) - x_new1;
            double b = y_new1 - (y_point_down);
            int R = 0.01;
            double ux_new = (ux[x_new1][y_new1]*sqrt(a*a+b*b)
                + ux[x_new2][y_new2]*sqrt((R-a)*(R-a) + b*b)
                + ux[x_new3][y_new3]*sqrt(a*a+(R-b)*(R-b))
                + ux[x_new4][y_new4]*sqrt((R-a)*(R-a)
                + (R-b)*(R-b)) ) /4;
            double uy_new = (uy[x_new1][y_new1]*sqrt(a*a+b*b)
                + uy[x_new2][y_new2]*sqrt((R-a)*(R-a) + b*b)
                + uy[x_new3][y_new3]*sqrt(a*a+(R-b)*(R-b))
                + uy[x_new4][y_new4]*sqrt((R-a)*(R-a)
                + (R-b)*(R-b)) ) /4;
            double uxn_new = -ux_new
                /sqrt(ux_new*ux_new+uy_new*uy_new);
            double uyn_new = -uy_new
                /sqrt(ux_new*ux_new+uy_new*uy_new);
            x_point_down = x_point_down+uxn_new*d1;
            y_point_down = y_point_down+uyn_new*d1;
            glColor3f(my_abs(uyn_new), 0, my_abs(uxn_new));
            glVertex2d(x_point_down, y_point_down);
        }
    }
    glEnd();
    glPopMatrix();
}

}

}

if (grafic == 4 ){
    i=0;
    for(x=2; x<n-2; x++){
        for(y=2; y<n-2; y++){
            glPushMatrix();
            double fi;
            fi = (atan2(cur_density_y[x][y],
                cur_density_x[x][y]))*180/3.14;
            glTranslatef(x,y,0);
            glRotatef( fi, 0,0,1);
            arrow();
        }
    }
}

```

```

        glPopMatrix();
        i++;
    }
}
}
glFinish();
}

```

Функция, осуществляющая анализ файлов формата .xls.

```

void xls_parcer() {
    for (j = 0; j < pWB->sheets.count; j++) {
        pWS = xls_getWorkSheet(pWB, j);
        xls_parseWorkSheet(pWS);
        int k;
        k = pWS->rows.lastrow;
        number_of_rows = pWS->rows.lastrow + 1;
        number_of_colls = pWS->rows.lastcol;
        int number_of_cells = number_of_rows * number_of_colls;
        cells_of_file = (cell_of_excell*)
        malloc(number_of_cells*sizeof(cell_of_excell));
        for (cellRow = 0; cellRow <= pWS->rows.lastrow; cellRow++) {
            row = xls_row(pWS, cellRow);
            for (cellCol = 0; cellCol <= pWS->rows.lastcol; cellCol++) {
                xlsCell *cell = xls_cell(pWS, cellRow, cellCol);
                if ((!cell) || (cell->isHidden)) {
                    continue;
                }
                cells_of_file[counter].x = cellRow;
                cells_of_file[counter].y = cellCol;
                cells_of_file[counter].sheet_name = (char *)
                malloc(((int)strlen(pWB->sheets.sheet[j].name)+2));
                memcpy(cells_of_file[counter].sheet_name,
                pWB->sheets.sheet[j].name,
                (int)strlen(pWB->sheets.sheet[j].name) *sizeof(char));
                cells_of_file[counter].sheet_name[
                ((int)strlen(pWB->sheets.sheet[j].name))] = '\0';
                if (cell->id == 0x27e || cell->id == 0x0BD
                || cell->id == 0x203) {
                    cells_of_file[counter].numeric_value = cell->d;
                    cells_of_file[counter].type = 1;
                } else if (cell->id == 0x06) {
                    if (cell->l == 0){
                        cells_of_file[counter].numeric_value = cell->d;
                        cells_of_file[counter].type = 1;
                    } else {

```

```

        if (!strcmp((char *)cell->str, "bool")){
            if((int) cell->d){
                cells_of_file[counter].string_value
                = (char*)malloc(strlen("true")*sizeof(char));
                cells_of_file[counter].string_value = "true";
                cells_of_file[counter].type = 2;
            }else{
                cells_of_file[counter].string_value
                = (char*)malloc(strlen("false")*sizeof(char));
                cells_of_file[counter].string_value = "false";
                cells_of_file[counter].type = 2;
            }
        } else if (!strcmp((char *)cell->str, "error")){
            int len = strlen((char *)cell->str);
            cells_of_file[counter].string_value =
            (char*)malloc(strlen("*error")*sizeof(char));
            cells_of_file[counter].string_value = "*error*";
            cells_of_file[counter].type = 2;
        }
        else
        {
            int len = strlen((char *)cell->str);
            cells_of_file[counter].string_value =
            (char *)cell->str;
            cells_of_file[counter].type = 2;
        }
    }
} else if (cell->str != NULL) {
    int len = strlen((char *)cell->str);
    cells_of_file[counter].string_value =
    (char *)cell->str;
    cells_of_file[counter].type = 2;
}

    counter+=1;
}

}
space();
print_massiv();
counter = 0;
free(cells_of_file);
}
}

```

Функция, добавляющая в таблицу measurement базы информацию о загружаемой таблице файла excell

```
void create_records_in_measurement(){
    int QUERY_LEN = 1000, i, w;
    char query[QUERY_LEN];
    memset(query, 0, QUERY_LEN);
    for (i=0; i<real_counter; i++){
        if (Cells[i].type == 1) {
            memset(query, 0, QUERY_LEN);
            sprintf(query, "insert into
            measurement (type, x, y, id, numeric)
            values (%d, %d, %d, %d, %lf);", Cells[i].type, Cells[i].x,
            Cells[i].y, Cells[i].sheet_id, Cells[i].numeric_value);
            for (w = strlen(query); query[w] != ','; w--);
            query[w]='.';
            res = PQexec(conn, query);
        }
        if (Cells[i].type == 2) {
            memset(query, 0, QUERY_LEN);
            sprintf(query, "insert into
            measurement (type, x, y, id, string )
            values (%d, %d, %d, %d, '%s');", Cells[i].type, Cells[i].x,
            Cells[i].y, Cells[i].sheet_id, Cells[i].string_value);
            res = PQexec(conn, query);
        }
    }
}
```

Функция, осуществляющая построение таблицы по выгруженным из базы данным:

```
void MainWindow::on_pushButton_5_clicked()
{
    int id_of_the_measurement = ui -> spinBox_4->value();
    upload_measurement(id_of_the_measurement);
    int i, max_x=0, max_y=0;
    for (i=0; i<size_of_database_val; i++){
        if (database_val[i].x > max_x) {
            max_x = database_val[i].x;
        }
        if (database_val[i].y > max_y) {
            max_y = database_val[i].y;
        }
    }
    QStandardItemModel *model =
    new QStandardItemModel(max_x, max_y, this);
    for (i=0; i< size_of_database_val; i++){
        QStandardItem *firstRow;
```

```

        if(database_val[i].type==2){
            firstRow = new QStandardItem(database_val[i].string_value);
        }
        if(database_val[i].type==1){
            firstRow =
                new QStandardItem((QString::
                    number(database_val[i].numeric_value)));
        }
        int x = database_val[i].x ,y = database_val[i].y;
        model->setItem(x,y, firstRow);
    }
    ui->tableView_4->setModel(model);
}

```

Функция, осуществляющая выгрузку из таблицы measurement необходимой ее части:

```

void upload_measurement(int id_of_the_measurement){
    int QUERY_LEN = 1000, i,n;
    char query[QUERY_LEN];
    memset(query, 0, QUERY_LEN);
    sprintf(query, "select type, x, y,
        numeric, string from measurement where id = %d",
        id_of_the_measurement);
    res = PQexec(conn, query);
    size_of_database_val = PQntuples(res);
    database_val = (cell *)malloc(size_of_database_val * sizeof(cell));
    for(i = 0; i < PQntuples(res); i++) {
        for(n = 0; n < PQnfields(res); n++) {
            char *str;
            str = (char *)
                malloc((strlen(PQgetvalue(res, i, n))+2) * sizeof(char));
            sprintf(str, "%s", PQgetvalue(res, i, n));
            if (n==0){
                database_val[i].type = atoi(str);
            }
            if (n==1){
                database_val[i].x = atoi(str);
            }
            if (n==2){
                database_val[i].y = atoi(str);
            }
            if(n==3 && database_val[i].type == 1){
                int q=0;
                char k = ',';
                for (q = strlen(str)-1; str[q] != '.'; q--){
                    if (q<1) break;

```

```

        }
        if (q>=1) {
            str[q]=k;
        }
        database_val[i].numeric_value = atof(str);
    }
    if(n==4 && database_val[i].type == 2){
        strcpy(database_val[i].string_value,
            PQgetvalue(res, i, n) );
    }
    free(str);
}
}
}

```

Функция, осуществляющая построение графика по данным из двух столбцов выгруженной таблицы:

```

void MainWindow::on_pushButton_7_clicked()
{
    int a =ui->spinBox_5 ->value();
    int b =ui->spinBox_6->value();
    a--;
    b--;
    int i, q=0, size_x=0, size_y=0, w=0;
    for(i=0; i<size_of_database_val; i++){
        if (database_val[i].y==a && database_val[i].type == 1){
            size_x++;
        }
        if (database_val[i].y==b && database_val[i].type == 1){
            size_y++;
        }
    }
    QVector<double> x(size_x), y(size_y);
    double x_max = x[0], y_max=y[0], y_min = y[0];
    for(i=0; i<size_of_database_val; i++){
        if (database_val[i].y==a && database_val[i].type == 1){
            x[q]= database_val[i].numeric_value;
            if (x[q]>x_max) x_max=x[q];
            q++;
        }
        if (database_val[i].y==b && database_val[i].type == 1){
            y[w]= database_val[i].numeric_value;
            if (y[w]>y_max) y_max=y[w];
            if (y[w]<y_min) y_min=y[w];
            w++;
        }
    }
}

```

```
}  
ui->widget->addGraph();  
ui->widget->graph(0)->setData(x, y);  
ui->widget->xAxis->setRange(0, x_max);  
ui->widget->yAxis->setRange(y_min, y_max);  
ui->widget->replot();  
}
```