**BC/NW 2021№ 2 (38):4.1**

**РАЗРАБОТКА ПРОГРАММЫ ОБРАБОТКИ РЕЗУЛЬТАТОВ МОДЕЛЬНЫХ ЭКСПЕРИМЕНТОВ**

Абросимов Л.И Полушин

Современная геополитическая обстановка в мире ставит перед Российской Федерацией задачу создания максимально эффективной системы обороны от любых средств нападения как из воздушной среды, так и из космического пространства [1]. Решение этой задачи требует непрерывного наблюдения за воздушной обстановкой и быстрой обработки информации с последующей передачей её вышестоящим пунктам для принятия решения о дальнейших действиях.

Необходимо разработать программу, которая будет производить селекцию движущейся в трехмерном пространстве цели. Для этого необходимо обработать результаты модельных экспериментов. Суть обработки состоит в усреднении параметров с привязкой к другому параметру. Программа будет являться компонентом для тестирования. В ней будет происходить селекция движущейся цели по заданным параметрам, и моделирование параметров. Для наглядности параметры модельных экспериментов необходимо выводить на графики. Все параметры хранятся в базе данных SQLite.

Воздушный объект можно охарактеризовать различным набором параметров, многие из которых не представляют полезной информации. Таким образом, чтобы абстрагироваться от ненужных переменных, свойств и взаимосвязей воздушного объекта нужно создать его модель.

В качестве исследуемого модельного эксперимента будет выступать движущийся в пространстве воздушный объект. Его модель представляет собой набор параметров, описывающих движение объекта в пространстве.

Всякий воздушный объект обладает свойством отражать падающую на него электромагнитную энергию. Чтобы оценить отражательную способность воздушного объекта вводится следующий параметр – ЭПР. ЭПР является дополнительным параметром для тестирования работоспособности программы.

Таким образом, сформирована модель с основными параметрами для решения задачи обработки результатов модельных экспериментов. По этой модели проектируется база данных.

В соответствии с техническим заданием решены следующие задачи:

– Определить параметры воздушных объектов .

– Составить структуру базы данных.

– Разработать программу заполнения базы данных.

– Разработать программу обработки информации путем привязки и усреднения заданных параметров.

– Сделать вывод по результатам работы.

Исходными данными для разработки являются:

– Фреймворк Qt C++.

– Система управления базами данных SQLite.

Ожидаемыми результатами являются: программа заполнения базы данных и программа обработки полученной информации с удобными и понятными интерфейсами.

В данной работе необходимо использовать Qt – фреймворк для разработки кроссплатформенного программного обеспечения на языке программирования C++. Преимущества Qt в его открытости и доступности, кроссплатформенности, наличии хорошей и полной документации [2]. Также для хранения и структурирования данных необходимо использовать СУБД SQLite. SQLite – это система управления базами данных, отличительной особенностью которой является ее встраиваемость в приложения [3].

**Структура базы данных**

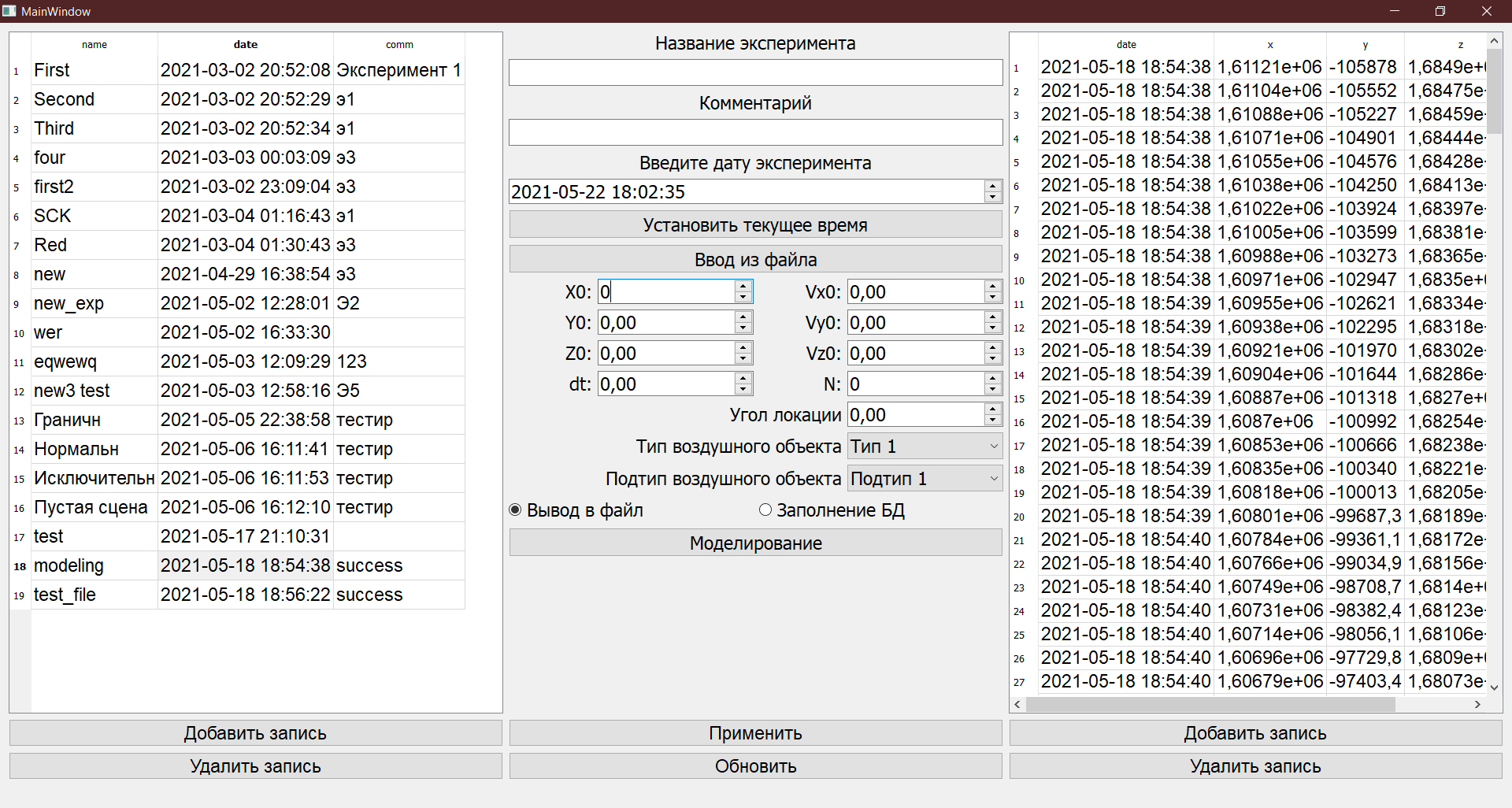
В модели объекта, были выделены основные параметры: координаты в трехмерном пространстве, ЭПР, время привязки информации, номер объекта. Эти параметры будут основой структуры базы данных. Каждый эксперимент содержит в себе множества этих параметров. Эксперименты отличаются временем начала и названием с комментарием пользователя.

Таким образом, структура базы данных принимает следующий вид: (рис. 1), где, соответственно обозначены, таблица экспериментов с первичным ключом *expid*, названием эксперимента *name*, датой и временем начала эксперимента *date* и комментарием пользователя – *comm*. Таблица параметров с первичным ключом *paramid*, внешним ключом *expid*, временем привязки параметров – *date*, координатами *X, Y* и *Z,* ЭПР – *epr*, и номером объекта - *number*. На столбцах *expid* таблицы экспериментов и *paramid* таблицы параметров установлен автоинкремент для упрощения создания новых записей. Столбцы *date* двух таблиц по умолчанию добавляют текущую дату и время. Столбец *number* содержит проверку вводимых значений. Значение этого столбца должно быть больше 0.

Исходя из технического задания, выбранная база данных – SQLite.



Рис. 1. Структура базы данных



**Заполнение базы данных из файла**

Чтобы добавить новый эксперимент из файла необходимо ввести название эксперимента, комментарий (не обязательно) и дату со временем в соответствующие поля и нажать на кнопку «Ввод из файла». Поле даты и времени автоматически проставляется текущей датой и временем, также это можно сделать, нажав на кнопку «Установить текущее время».

Далее пользователю откроется диалоговое окно, где необходимо выбрать файл с расширением txt. Если структура файла соответствует структуре, описанной в п. 2.2, то в первой таблице отобразится новый эксперимент. Схема алгоритма заполнения базы данных из файла представлен на рис. 2. Алгоритм построчно читает файл и составляет динамический запрос вставки данных в базу данных. Вначале происходит добавление нового эксперимента, и получение его *idexp*. Дальше происходит чтение названий столбцов и составление части запроса вставки. После этого в запрос добавляются значения. Если какой-либо запрос не выполнился, или не удалось считать файл или данные, то переменная логического типа *success* устанавливается в логический ноль. В конце происходит проверка значения этой переменной. Если она равна логическому нулю, то все добавленные данные удаляются, и пользователю выводится сообщение об ошибке.



Рис. 2. Схема алгоритма заполнения БД из файла

**Моделирование траектории полета воздушного объекта**

Для моделирования параметров эксперимента потребуется ввести в соответствующие поля следующее: название эксперимента, комментарий, дату и время, проекция векторов начального положения и начальной скорости объекта на оси координат РЛС, шаг времени, количество параметров, тип объекта, подтип объекта, угол локации. Затем необходимо выбрать куда выгружать данные: в базу данных или в *txt* файл, и нажать на кнопку «Моделирование».

Моделирование производится методом Рунге-Кутты четвёртого порядка по следующим формулам:

где xn+1, yn+1, zn+1 – значения координат в момент времени n+1;

xn, yn, zn – значения координат в момент времени n;

Vxn+1, Vyn+1, Vzn+1 – значения проекций вектора скорости в момент времени n+1;

Vxn, Vyn, Vzn – значения проекций вектора скорости в момент времени n;

h – величина шага;

tn+1 – время на n+1 шаге;

tn – время на n-ом шаге;

dt – шаг времени.

Для нахождения значений проекций вектора скорости в момент времени n+1 служат функции K1(x,y,z,Vx,Vy,Vz,dt), К2(x,y,z,Vx,Vy,Vz,dt), К3(x,y,z,Vx,Vy,Vz,dt). К1(x,y,z,Vx,Vy,Vz,dt) – функция нахождения коэффициентов k11, k21, k31, k41 для проекции вектора скорости на ось X. К2(x,y,z,Vx,Vy,Vz,dt) – функция нахождения коэффициентов k12, k22, k32, k42 для проекции вектора скорости на ось Y. К3(x,y,z,Vx,Vy,Vz,dt) – функция нахождения коэффициентов k13, k23, k33, k43 для проекции вектора скорости на ось Z. Остальные коэффициенты для xn+1, yn+1 и zn+1 находятся по рассмотренным ранее коэффициентам. Функции K1, K2 и K3 будут описаны ниже.

Рассмотрим нахождение коэффициентов на примере нахождения следующего значения координаты xn+1 и значения проекции вектора скорости в момент времени n+1 на ось X, формулы (8 – 23):

где *kx* – поправка к текущему значению координаты x;

*kvx* – поправка к проекции вектора скорости на ось X.

Аналогично находятся остальные значения yn+1, zn+1, Vyn+1 и Vzn+1.

Рассмотрим функции K1, K2, K3 [9]. Входными параметрами являются значения xn, yn, zn, Vxn, Vyn и Vzn. На выходе получается соответствующий коэффициент для Vxn+1,Vyn+1 или Vzn+1. Функция K1:

где Wz – угловая скорость вращения Земли, равная 0.7292115\*10-4 [c-1];

R – средний радиус Земли, равный 6371000 [м];

r – модуль вектора положения (25);

a1,2 – параметры для учета гравитации (26,27);

J2 – коэффициент 2-ой зональной гармоники, равный 108262.575\*10-8;

*BigSemiAxis* – экваториальный радиус Земли, равный 6378136 [м];

*GeocentricalGravityConst* – геоцентрическая гравитационная постоянная Земли, равная 39860044.18\*107 [м3/с2].

Функция K2:

где Wz – угловая скорость вращения Земли, равная 0.7292115\*10-4 [c-1];

R – средний радиус Земли, равный 6371000 [м];

r – модуль вектора положения (25);

a1,2 – параметры для учета гравитации (26,27);

J2 – коэффициент 2-ой зональной гармоники, равный 108262.575\*10-8;

*BigSemiAxis* – экваториальный радиус Земли, равный 6378136 [м];

*GeocentricalGravityConst* – геоцентрическая гравитационная постоянная Земли, равная 39860044.18\*107 [м3/с2].

Функция K3:

где Wz – угловая скорость вращения Земли, равная 0.7292115\*10-4 [c-1];

R – средний радиус Земли, равный 6371000 [м];

r – модуль вектора положения (25);

a1,2 – параметры для учета гравитации (26,27);

J2 – коэффициент 2-ой зональной гармоники, равный 108262.575\*10-8;

*BigSemiAxis* – экваториальный радиус Земли, равный 6378136 [м];

*GeocentricalGravityConst* – геоцентрическая гравитационная постоянная Земли, равная 39860044.18\*107 [м3/с2].

Для моделирования параметров эксперимента создан отдельный класс *prognoz*. Описание класса приведено в табл. 1.

Таблица 1

Описание класса *prognoz.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод или свойство | Модификатор доступа | Тип | | Аргументы | Описание |
| prognoz | public | void\* | | - | Конструктор |
| Wz | private | double const | | - | Угловая скорость Земли |
| J2 | private | double const | | - | Коэффициент 2-ой зональной гармоники |
| R | private | double const | | - | Средний радиус Земли |
| Метод или свойство | Модификатор доступа | Тип | Аргументы | | Описание |
| a00 | private | double const | - | | Параметр для учета гравитации |
| a20 | private | double const | - | | Параметр для учета гравитации |
| cz | private | double const | - | | Параметр для учета гравитации |
| K1 | private | double | const double &x, const double &y, const double &z, double &Vx, const double &Vy, const double &Vz, double &dt | | функция нахождения коэффициентов k11, k21, k31, k41 для проекции вектора скорости на ось X |
| K2 | private | double | const double &x, const double &y, const double &z, double &Vx, const double &Vy, const double &Vz, double &dt | | функция нахождения коэффициентов k12, k22, k32, k42 для проекции вектора скорости на ось Y |
| K3 | private | double | const double &x, const double &y, const double &z, double &Vx, const double &Vy, const double &Vz, double &dt | | функция нахождения коэффициентов k13, k23, k33, k43 для проекции вектора скорости на ось Z |
| Метод или свойство | Модификатор доступа | Тип | Аргументы | | Описание |
| prop | public | double | double &t, double &x, double &y, double &z, &Vx, double &Vy, double &Vz, double &dt,const double angle,double &epr, int type,int subtype | | Моделирование параметров на один шаг |
| density | private | double[2][35] | - | | Массив соответствия высоты и плотности атмосферы |
| getDensity | private | double | const double &z | | Расчет высоты и получение плотности |
| SRZNACH\_TYPE1 | private | double[3] | - | | Массив средних значение ЭПР для типа 1 (экспоненциальное распределение) |
| MIN\_MAX\_TYPE2 | private | double[2][2][10] | - | | Массив соответствия угла локации и минимального с максимальным значением ЭПР для типа 2 (равномерное распределение) |
| Метод или свойство | Модификатор доступа | Тип | Аргументы | | Описание |
| my\_uniform | private | double | - | | Функция получения случайной величины |
| my\_uniformMINtoMAX | private | double | double min, double max | | Функция получения случайной величины с равномерным законом распределения |
| my\_exponential | private | double | double lambda | | Функция получения случайной величины с экспоненциальным законом распределения |
| calcEPR\_TYPE1 | public | double | int subtype | | Функция получения ЭПР для типа 1 |
| calcEPR\_TYPE2 | public | double | double angle, int subtype | | Функция получения ЭПР для типа 2 |

Экземпляр класса *prognoz* моделирует координаты и вектор скорости на один шаг, поэтому вызов метода *prop* происходит в циклеот 0 до N, где N – заданное количество параметров. Для расчета начальной ЭПР предназначены методы *calcEPR\_TYPE1* и *calcEPR\_TYPE2*. Последующий расчет значений ЭПР производятся в методе *prop*. Схема алгоритма метода *prop* (рис. 3) высчитывает значения по формулам (1 – 23).



Рис. 3. Схема алгоритма метода *prop*

**Реализация программы обработки результатов модельных экспериментов**

Обработки результатов модельных экспериментов состоит из нескольких этапов. Для начала нужно выбрать эксперименты, по которым будет производиться обработка. Для удобства нахождения нужного эксперимента предусмотрен фильтр записей. Как и в случае с программой заполнения базы данных, номер эксперимента определяется автоматически и параметры соответствующего эксперимента отображаются во второй таблице *QTableView*, за которой закрепляется модель *QSqlTableModel*. Чтобы выбрать эксперимент необходимо сначала нажать на него в таблице экспериментов, после этого нажать на кнопку добавить. Выбранный эксперимент отобразится в *QTableView* в нижней части экрана. Чтобы отменить выбор нужно выбрать его в *QTableView* в нижней части экрана и нажать на кнопку «удалить». Эксперимент удалится только из выбранных, изменение базы данных не предусмотрено в этой программе. Чтобы полностью очистить выбранные эксперименты необходимо нажать на кнопку «удалить все». Затем указать какой параметр усреднить и к какому параметру привязаться, а также задать значение дельты или выбрать «плавающую» дельту. После этого нужно нажать на кнопку «обработать». Затем происходит обработка и вывод информации в таблицу. Очистить таблицу можно с помощью кнопки «очистить». Чтобы сохранить таблицу на диске нужно нажать на кнопку «сохранить в файл» и задать расположение на диске в диалоговом окне выбора файлов.

Дополнительно можно построить графики параметров от времени нужного эксперимента или графики по результатам обработки. Графиков можно выводить любое количество. Размер графиков подстраивается под размер интерфейса. Увеличить изображение на графиках можно с помощью колесика мыши. Очистить графики возможно также по кнопке «очистить».

В программе не предусмотрено изменение базы данных, поэтому редактирование *QTableView* запрещено.

Интерфейс программы представлен на рис. 4 и рис. 5.

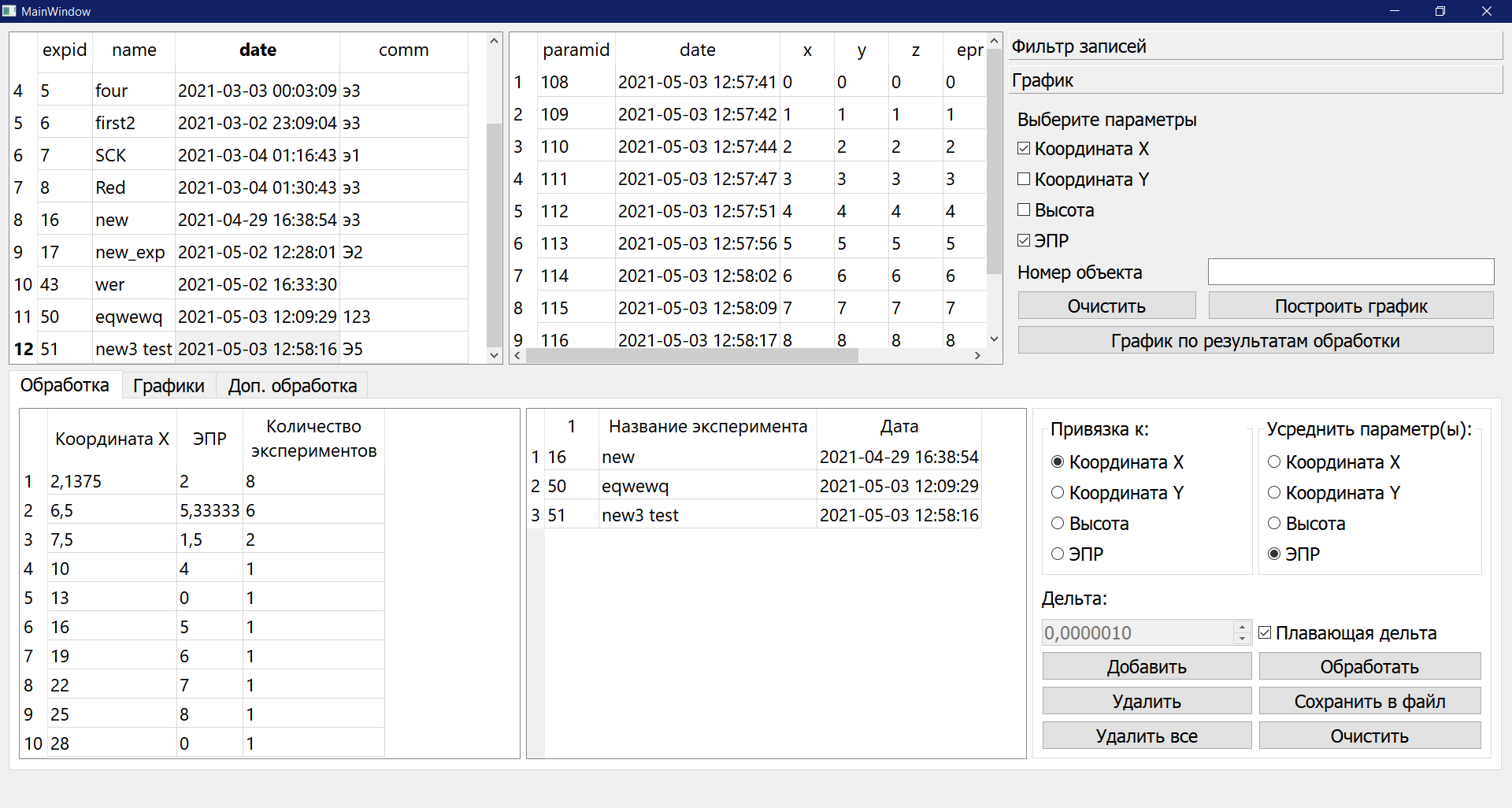


Рис. 4. Интерфейс программы (обработка результатов)

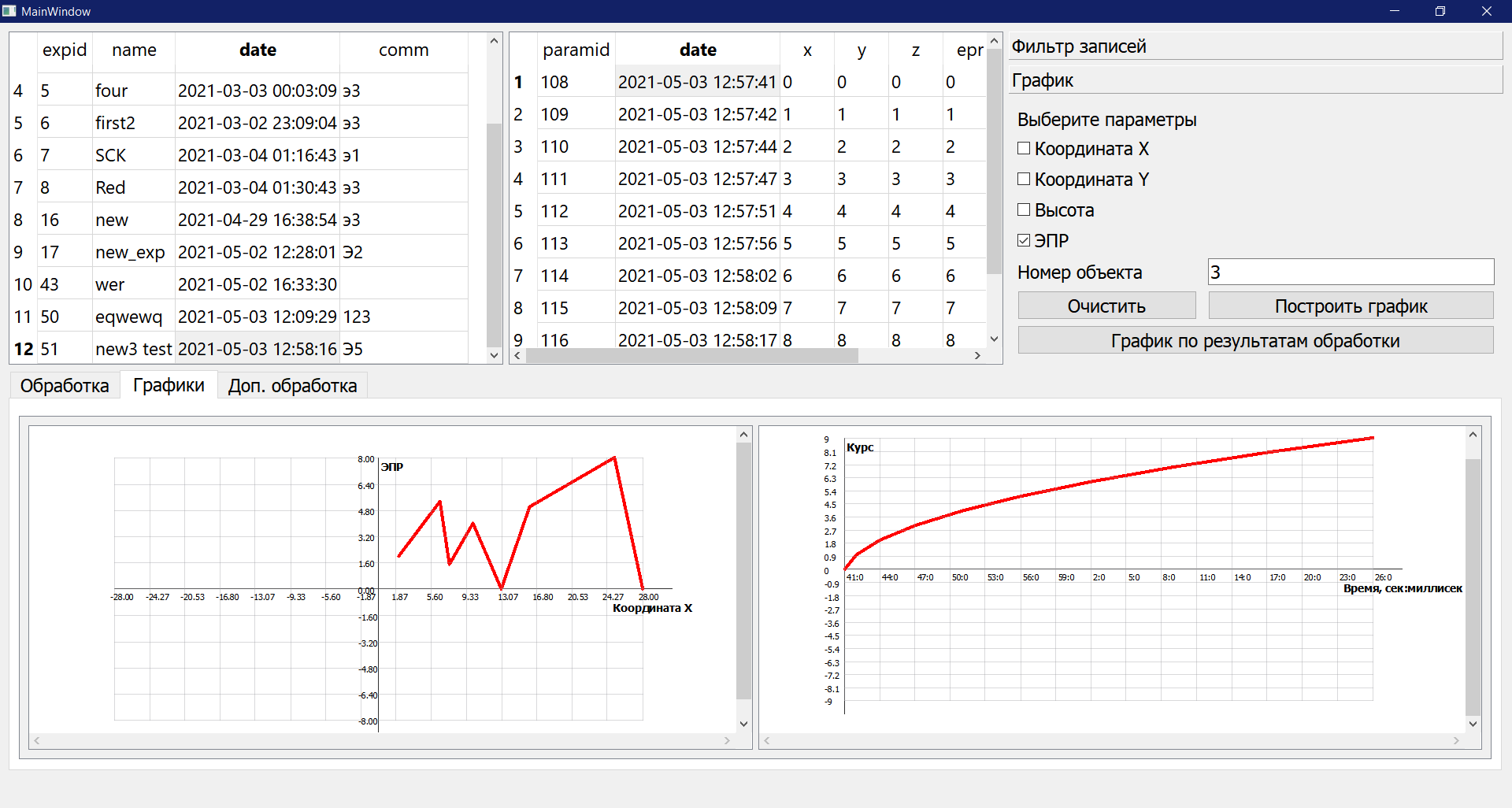


Рис. 5. Интерфейс программы (построение графиков)

**Обработка результатов**

После выбора экспериментов необходимо выбрать параметр усреднения и параметр привязки и задать значение дельты или указать «плавающую» дельту. Параметры выбираются с помощью *QRadioButton* – переключатель с текстом. Указать «плавающую» дельту можно с помощью *QCheckBox –* элемент графического интерфейса, у которого есть два состояния: включено и выключено (установлено и не установлено). При установке *QCheckBox* блокируется ввод значения дельты.

Для обработки пользователю нужно нажать на кнопку «обработать». После нажатия происходит проверка: выбраны ли минимум два эксперимента. По умолчанию выбран параметр «Координата X» для привязки и усреднения. Если проверка не прошла, то пользователь получает соответствующее сообщение.

Далее происходит создание динамического запроса в базу данных, и вывод информации в *QTableView*, за которым закреплена модель *QStandardItemModel*, которая связывается с *QTableView* посредством вызова методов:

– QStandardItemModel::QStandardItemModel(int rows, int columns, QObject \*parent = nullptr) – конструктор, в который передается количество строк и столбцов;

– void QTableView::setModel(QAbstractItemModel \*model) – метод, в который передается указатель на модель.

Все изменения в модели отображаются на связанном с ней экземпляром класса *QTableView.*

Алгоритм создания динамического запроса и выдачи результатов представлен на рис. 6 и рис. 7 [10 – 12].



Рис. 6. Схема алгоритм обработки и выдачи информации



Рис. 7. Алгоритм обработки и выдачи информации

Для усреднения выбирается основной эксперимент – первый эксперимент в списке выбранных экспериментов. Количество строк таблицы результата обработки равно количеству строк параметров основного эксперимента. Алгоритм проходится по каждой строке параметров основного эксперимента и находит соответствующие значения привязки, в зависимости от дельты, в таблице параметров следующего эксперимента в списке выбранных, затем складывает параметры усреднения и привязки, и увеличивает счетчик совпадений на единицу. После прохода по всем строкам параметров не основного эксперимента, сохраняет параметры в *model3* – модель таблицы обработанной информации. В конце, после прохода по всем параметрам всех выбранных экспериментов, происходит усреднение значений в *model3* и вывод результатов в табличном виде.

**Построение графиков**

Для построения графиков был создан отдельный класс *MyGraphicView,* унаследованный от *QGraphicsView –* предоставляет виджет для отображения содержимого *QGraphicsScene*. *QGraphicsScene –* предоставляет поверхность для управления большим числом графических 2D элементов. Описание класса приведено в табл. 2.

Таблица 2

Описание класса *MyGraphicView.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод или свойство | Модификатор доступа | Тип | Аргументы | Описание |
| MyGraphicView | public | QGraphicsView | QWidget\*parent | Конструктор |
| scene | private | QGraphicsScene\* | - | Указатель на сцену для рисования элементов |
| group\_1,  group\_2 | private | QGraphicsItemGroup\* | - | Указатель на группы элементов |
| timer | private | QTimer\* | - | Указатель на таймер |
| data\_time | private | QVector<QDateTime>\* | - | Динамический массив для хранения дат и времен |
| data\_param | private | QVector<QString> \* | - | Динамический массив для хранения значений параметров |
| data\_flag | private | bool | - | Флаг, который устанавливается в логическую единицу при вызове метода SetData |
| count\_data | private | int | - | Содержит количество данных |
| difDate | private | qint64 | - | Разница между самой ранней и поздней датами в секундах |
| Метод или свойство | Модификатор доступа | Тип | Аргументы | Описание |
| zero, zero2 | private | int | - | Начальное значение времени |
| max, max2 | private | double | - | Максимальное значение параметров по модулю |
| param\_str, param\_str2 | private | QString | - | Название параметра |
| no\_time | private | bool | - | Флаг установки данных без зависимости от времени |
| vect\_par1, vect\_par2 | private | QVector <double>\* | - | Динамический массив для хранения значений параметров без зависимости от времени |
| resizeEvent | private | void | QResizeEvent \*event | Перехватывает событие изменение размера виджета |
| deleteItemsFromGroup | private | void | QGraphicsItemGroup \*group | Удаляет группу элементов |
| eventFilter | protected | bool | QObject \* obj , QEvent \* event | Перехватывает события от колесика мыши и изменяет масштаб |
| Метод или свойство | Модификатор доступа | Тип | Аргументы | Описание |
| no\_time\_graph | private | void | - | Метод построения графика по значениям, полученным в результате обработки (не зависящим от времени) |
| setData | public | void | QVector<QDateTime> \*datetime,QVector<QString> \*param,QString par\_str | Устанавливает данные (зависимость параметра от времени) |
| setData (перегружен) | public | void | QVector<double> \*vect\_param1, QVector<double> \*vect\_param2,QString par\_str, QString par\_str2 | Устанавливает данные (зависимость параметра от другого параметра) |
| slotAlarmTimer | private | void | - | Слот для обработчика переполнения таймера. В нем производится перерисовка графика. |

Для того, чтобы построить график зависимости параметра от времени привязки пользователю необходимо выбрать параметр(ы), эксперимент и указать номер объекта. Затем нажать на кнопку «построить график».

Для указания параметров используется *QCheckBox*. При нажатии на *QCheckBox* заполняется массив логического типа *check\_graph[4]*, где: *true* – параметр выбран, *false* – параметр не выбран.

Алгоритм построения графиков сначала производит проверку: выбрал ли пользователь эксперимент, указал ли параметр(ы) и номер объекта.

После проверки производится запрос в базу данных «SELECT» выборки столбцов даты и выбранного параметра с фильтром по номеру эксперимента и номеру объекта, упорядоченные по дате. Затем создается новый элемент типа *QVector<MyGraphicView\*>,* в котором содержатся указатели на экземпляры класса *MyGraphicView*, и заполняются две структуры типа *QVector<QVector<QDateTime>\*>* и *QVector<QVector<QString>\*>*, в них содержатся указатели на динамические массивы *QVector<QDateTime>* и *QVector<QString>*, в которых содержатся значения даты и параметров, полученных после выполнения запроса в базу данных. Если нужно построить график по результатам обработки, то заполняются две структуры типа *QVector<QVector<double>\*>,* в них содержатся указатели на динамические массивы *QVector< double >*. Таким образом реализовано добавление любого количества графиков.

Метод *SetData* перегружен, это нужно для того, чтобы устанавливать параметры различных типов: первый метод принимает *QVector<QDateTime>\** и *QVector<QString>\** – зависимость выбранного параметра от времени, и аргумент типа *QString* – название параметра, второй принимает два аргумента типа QVector<double> \* – зависимость параметра от другого параметра и два аргумента *QString* – названия параметров.

Алгоритм метода *SetData,* кроме инициализации данных, определяет интервал времени и записывает его в свойство *difDate,* а также находит максимальное значение параметров по модулю, и устанавливает свойство *data\_flag* в логическую единицу, а *no\_time* в логический ноль. Если вызван перегруженный *SetData,* то алгоритм находит два максимальных значения по модулю, устанавливает флаг *no\_time* и *data\_flag* в логическую единицу.

Когда создается экземпляр класса *MyGraphicView,* начинает отчет таймер, установленный на 50 миллисекунд. После переполнения таймера вызывается слот *slotAlarmTimer()*, в котором происходит прорисовка графика. Если таймер переполнится раньше, чем установятся данные, то свойство *data\_flag* не позволит произвести прорисовку. Алгоритм прорисовки автоматически подбирает масштаб исходя из временных параметров, и определяет размерность времени: секунды, минуты или часы. Если флаг *no\_time* установлен в логическую единицу, то *slotAlarmTimer()* вызывает метод *no\_time\_graph().* Метод *no\_time\_graph()* аналогичен методу *slotAlarmTimer()*, с тем отличием, что *no\_time\_graph()* не делает поправку на время, и ось ординат находится посередине, так как значения на оси абсцисс могут быть отрицательными. Схема алгоритма метода *slotAlarmTimer()* представлена на рис.8, рис.9 и рис. 10. Вначале алгоритма идет инициализация сцены и делителя сцены. Делитель сцены нужен для упрощения построения графиков. Далее идет построение осей. После этого расчет точек на оси абсцисс и времени для каждой точки, одновременно с этим строятся вертикальные линии сетки. Время выбирается в зависимости от разности самой поздней и самой ранней даты и времени в секундах. Затем алгоритм рассчитывает точки на оси ординат и значения в этих точках. Расчет на оси ординат зависит от количества точек и максимального значения параметров по модулю. В конце проставляются названия осей, на оси абсцисс также в зависимости от разницы дат, на оси ординат в соответствии с принятым параметром [13, 14].



Рис. 8. Схема алгоритма метода *slotAlarmTimer()* (стр. 1)



Рис. 9. Схема алгоритма метода *slotAlarmTimer()* (стр. 2)



Рис. 10. Схема алгоритма метода *slotAlarmTimer()* (стр. 3)

Данный класс может строить графики не только с зависимостью параметра от времени, но и с зависимостью параметра от другого параметра, с помощью метода *no\_time\_graph()*. Этот метод аналогичен методу *slotAlarmTimer()*, отличается построением оси абсцисс. Ось ординат находится посередине, так как значения на оси абсцисс могут быть отрицательными. Для того, чтобы установить зависимость параметра от другого параметра необходимо вызвать перегруженный метод *SetData ( QVector < double > \* vect\_param1 , QVector < double > \*vect\_param2, QString par\_str, QString par\_str2)*. Его аргументы принимают два вектора (динамических массива) типа double, и названия параметров. Перегруженный метод *SetData* также высчитывает разницу между максимальным и минимальным значением параметров и сохраняет в *max* и *max2* для одного вектора и другого. Этот метод устанавливает *no\_time* в логическую единицу, тем самым блокирует выполнение метода *slotAlarmTimer()* и вызывает метод *no\_time\_graph()*.

Расчет координат точек графика на сцене в методе *no\_time\_graph()* происходит следующим формулам:

, где X1,2 – координата X первой и второй точек на сцене;

Y1,2 – координата Y первой и второй точек на сцене;

width – ширина сцены;

height – высота сцены;

n – делитель ширины сцены;

max, max2 – разности между максимальными и минимальными значениями параметров для vect\_par1 и vect\_par2 соответственно.

x1,2, y1,2 – соседние значения параметров на каждом шаге.

Расчет координат точек графика на сцене в методе *slotAlarmTimer()* происходит следующим формулам:

, где X1,2 – координата X первой и второй точек на сцене;

Y1,2 – координата Y первой и второй точек на сцене;

width – ширина сцены;

height – высота сцены;

n – делитель ширины сцены;

max – разность между максимальным и минимальным значениями параметров.

difDate\_res – разность между самой ранней и самой поздней датой со временем в секундах.

x1,2, y1,2 – соседние значения параметров на каждом шаге.

Здесь x1 и x2 предварительно делятся на коэффициент времени в зависимости от difDate\_res:

– если difDate\_res<60, то x1,2 = x1,2;

– если difDate\_res/60>60, то x1,2 = x1,2/360;

– если difDate\_res/60<60, то x1,2 = x1,2/60;

Таким образом происходит преобразование значений параметров в координаты линии на сцене. Линия графика рисуется красным цветом, линии осей – черным, линии сетки – серым. Результат построения графика представлен на рис. 11.

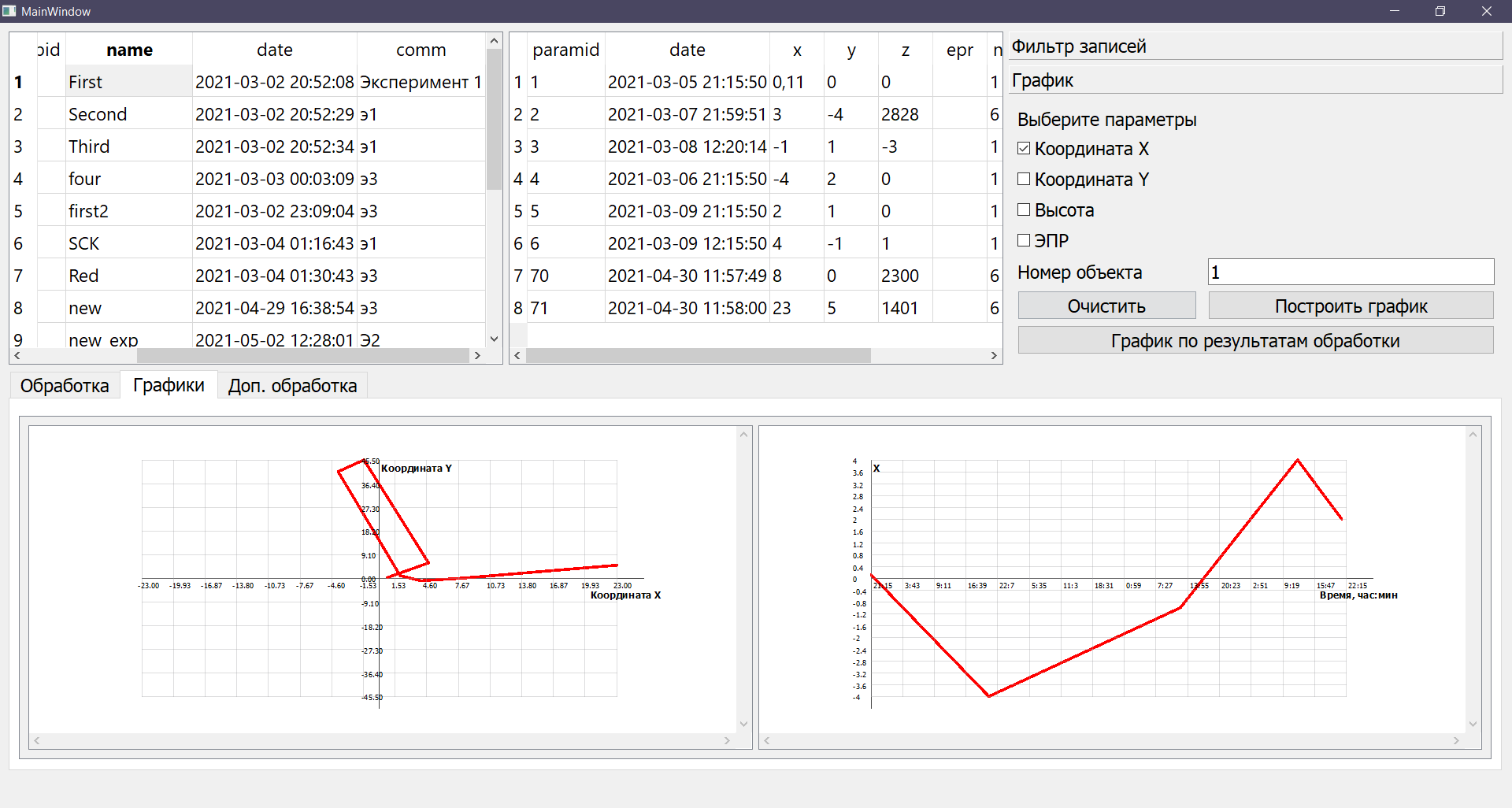


Рис. 11. Результат построения графика (Левый график – график по результатам обработки, правый – зависимость параметра “Координата X” от времени, эксперимент “First”, номер объекта 1)

Чтобы построить график по результатам обработки модельных экспериментов нужно сначала провести обработку, затем нажать на кнопку «График по результатам обработки».

Как только происходит изменение размера, то вызывается метод *resizeEvent(QResizeEvent \*event)*, который запускает таймер на 10 миллисекунд, и после переполнения происходит повторная прорисовка графика.

Когда происходит событие колёсика мыши, то вызывается метод *eventFilter(QObject \*watched, QEvent \*event)*, который изменяет масштаб сцены *QGraphicsScene*.

Графики добавляются в *QGridLayout* – класс, который выравнивает виджеты по сетке. Для удобства чтения информации с графиков в одной строке *QGridLayout* содержится не более двух графиков.

Чтобы удалить все графики необходимо нажать на кнопку «очистить». Алгоритм обработки нажатия этой кнопки удаляет содержимое структур типов QVector < MyGraphicView \*>, *QVector< QVector < QDateTime >\*>*, *QVector < QVector < QString > \*>* и *QVector<double>\**, и очищает занятую этими структурами память*.*

**Тестирование программ**

Для оценки быстродействия программ используется встроенная библиотека *QTimer*, которая позволяет засечь время моделирования и время обработки параметров. Результат отображается с помощью *QDebug* – класс, который предоставляет выходной поток для отладочной информации. Время засекается с момента нажатия на кнопку и до мометра выхода программы из слота обработки нажатия кнопки.

Согласно стандарта ГОСТ 19.301-79 испытания проводились по следующей схеме:

1. Объект испытаний – программа обработки результатов модельных экспериментов и программа заполнения базы данных.
2. Цель испытаний – выявление ошибок, проверка работоспособности программ, оценка быстродействия.
3. Средства испытаний.

Состав используемых во время испытаний технических средств:

– Ноутбук Lenovo Y500 с процессором Intel Core i7;

– ОЗУ 16 Гбайт;

– 4 Гбайт видеопамяти;

– наличие свободного места на жестком диске более 200 Мбайт;

1. Порядок испытаний. Сначала тестируется правильность вывода результатов моделирования, затем корректность введенной информации, и правильность вычислений с «плавающей» и константной (заданной пользователем) дельтой, затем производится оценка быстродействия программ: засекается время моделирования и время обработки результатов модельных экспериментов.
2. Методы испытаний. Испытания проводятся методом функционального тестирования.

Тесты предназначены для отладки функций программ и выявления следующих ситуаций:

– пользователь не ввел название эксперимента при моделировании параметров;

– пользователь не ввел необходимые данные при ручном заполнении;

– пользователь выбрал меньше двух экспериментов для обработки;

– произведена попытка удалить запись без её выбора;

– произведена попытка построить график по недостаточным данным.

Каждый тест характеризуется своим набором входных данных. Когда происходит одна из вышеперечисленных ситуаций происходит информирование пользователя о соответствующей ошибке с помощью диалогового окна Все тесты проведены успешно, новых ошибок не обнаружено.

Время моделирования параметров воздушного объекта зависит только от количества записей, поэтому оценка быстродействия моделирования производится на любом наборе начальных параметров, с изменением количества записей. Количество записей меняется от 1 до 10 с шагом 1, от 10 до 100 с шагом 10, от 100 до 1000 с шагом 100, от 1000 до 10000 с шагом 1000. Зависимость времени моделирования от количества записей представлена на рис. 12.

Рис. 12. Зависимость времени моделирования от количества записей

При моделировании параметров воздушного объекта количество записей обычно не превышает 1000. На основании полученных результатов можно сделать вывод, что моделирование параметров движения воздушного объекта происходит быстро для пользователя (меньше 10 секунд).

Время обработки результатов модельных экспериментов зависит от нескольких параметров: количество выбранных экспериментов, количество записей в каждом эксперименте, количество записей в основном эксперименте. При обработке результатов модельных экспериментов чаще всего выбирается меньше 10 экспериментов и меньше 1000 записей в каждом эксперименте. Исходя из этого оценим время обработки на следующих входных данных: количество выбранных экспериментов меняется от 2 до 10 с шагом 1, количество записей во всех экспериментах равно 1000, параметр привязки – координата X, параметр усреднения – ЭПР, дельта равна 100. Зависимость времени обработки от количества выбранных экспериментов представлена на рис 13.

Рис. 13. Зависимость времени обработки от количества выбранных экспериментов

На основании полученных результатов можно сделать вывод, что обработка результатов модельных экспериментов происходит практически мгновенно для пользователя (меньше 1 секунды).

В ходе использования и тестирования программы обработки результатов модельных экспериментов можно выделить часто повторяющийся ряд действий, а именно задание фильтра по экспериментам, выбор экспериментов и обработка информации. Для выполнения этих действий потребуется установить параметры фильтра во вкладке «Фильтр записей» (при запуске программы открыта именно эта вкладка) верхнего окна, нажать кнопку «применить» в той же вкладке, выбрать эксперимент, нажать на кнопку «Добавить» во вкладке «Обработка» нижнего окна (при запуске открыта), повторить выбор и нажатие кнопки «Добавить» для выбора остальных экспериментов, затем указать параметры обработки и нажать кнопку «Обработать» во вкладке «Обработка» нижнего окна. Все эти действия не требуют перехода на другие вкладки программы. В случае, если необходимо провести дополнительную обработку, или построить график необходимо переключиться на другие вкладки. В этом случае пользователю станут открытыми именно те блоки данных, с которыми он будет взаимодействовать, и не будет мешать лишняя информация. В программе заполнения базы данных все действия происходят в одном окне, и информация из базы данных всегда доступна пользователю. Таким образом, достигается максимальное удобство использования программ.

**Список использованных источников**

1. ПАО «НПО «Алмаз».: сайт. – URL: [www.raspletin.com](http://www.raspletin.com) (дата обращения 01.06. 2021). – Текст: электронный.
2. Qt (software) – Wikipedia.: сайт – URL: [www.wikipedia.org/wiki/Qt\_(software)](http://www.wikipedia.org/wiki/Qt_(software)) (дата обращения 31.05.2021). – Текст: электронный.
3. SQLite. сайт: – URL [www.younglinux.info/sqlite/sqlite](http://www.younglinux.info/sqlite/sqlite) (дата обращения 31.05.2021). – Текст: электронный.
4. Штофф В.А. Моделирование и философия. – М. Л.: Наука, 1966. 295 с.
5. Qt Plotting Widget QCustomPlot – Introduction.: сайт. – URL: www.qcustomplot.com (дата обращения: 07.05.2021). – Текст: электронный.
6. Qwt User's Guide: Qwt – Qt Widgets for Technical Applications.: сайт. – URL: qwt.sourceforge.io (дата обращения 07.05.2021). – Текст: электронный.
7. Сигналы и слоты в Qt. – URL: www.habr.com/ru/post/50812 (дата обращения 07.05.20210). – Текст: электронный.
8. Базовые контейнеры. сайт: – URL: [www.doc.crossplatform.ru/qt/4.3.2/containers.html](http://www.doc.crossplatform.ru/qt/4.3.2/containers.html) (дата обращения 01.06.2021). – Текст: электронный.
9. Сихарулидзе Ю.Г. Баллистика летательных аппаратов. — М.: Наука, 1982. — 352 с.
10. Прата С. Язык программирования C++ (C++11). Лекции и упражнения. - 6 изд. — М.: Вильямс, 2012. — 1248 с.
11. Страуструп Б. Язык программирования С++. Специальное издание. Пер. с англ. — М.: Издательство Бином, 2011 г. — 1136 с.
12. Лафоре Р. Объектно-ориентированное программирование в С++. Спб.: Питер, 2018. – 928 с.
13. Шлее М. Qt 4.8. Профессиональное программирование на C++ . — Спб.: БХВ-Петербург, 2012. — 894 с.
14. Боровский А.Н. Qt4.7+. Практическое программирование на C++. — СПб.: БХВ-Петербург, 2012. — 496 с.