**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

**Кафедра технологий программирования**

**РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ НА PYQT ДЛЯ АНАЛИЗА СВЕТОВЫХ КРИВЫХ ЗВЁЗД**

Курсовая работа

|  |
| --- |
| Михайлова Антона Артуровича |
| студента 3 курса,  специальность «прикладная информатика» |
| **Научный руководитель:** старший преподаватель Карпович Н.А.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |

Минск, 2020

РЕФЕРАТ

Курсовая работа, 34 c., 25 рис..

**Ключевые слова:** PYQT, PYQTGRAPH, ЭКЗОПЛАНЕТА, ТРАНЗИТНЫЙ МЕТОД, АСТРОНОМИЯ, PYTHON.

**Объект исследования** — объектом исследования являются возможности PyQt для создания удобного интерфейса для исследователей световых кривых звёзд. В качестве предмета исследования выбираем разработку и тестирование возможностей библиотек PyQt и PyQtGraph на Python.

**Цели работы —** спроектировать приложение для обработки и анализа световых кривых и реализовать его на языке Python при помощи библиотек PyQt и PyQtGraph.

**Методы исследования —** а) теоретические: изучение литературы, посвященной обработке и анализу временных последовательностей; б) практические: проектирование приложения при помощи PyQt Designer и его разработка на Python при помощи библиотек PyQt и PyQtGraph.

**Результатами являются —** приложение для обработки и анализа световых кривых и нахождения транзитов экзопланет на Python.

**Область применения** — предсказание существования экзопланет по световым кривым звёзд нашей галактики.

РЭФЕРАТ

Курсавая работа, 34 ст., 25 мал..

**Ключавыя словы**: PYQT, PYQTGRAPH, ЭКЗАПЛАНЕТА, ТРАНЗІТНЫ МЕТАД, АСТРАНОМІЯ, PYTHON.

**Аб'ект даследавання** **—** аб’ектам даследавання з’яўляюцца магчымасці PyQt ў стварэнні зручнага інтерфейсу для даследчыкаў светлавых крывых зорак. У якасці прадмета даследавання выбіраем распрацоўку і тэставанне магчымасцяў бібліятэк PyQt і PyQtGraph на Python.

**Мэты працы** **—** спраектаваць прыкладанне для апрацоўкі і аналізу светлавых крывых і рэалізаваць яго на Python пры дапамозе бібліятэк PyQt і PyQtGraph.

**Метады даследавання** **—** а) тэарэтычныя: вывучэнне літаратуры, прысвечанай апрацоўцы і аналізу часавых паслядоўнасцяў; б) практычныя: праектаванне прыкладання пры дапамозе PyQt Designer і яго распрацоўка на Python пры дапамозе бібліятэк PyQt і PyQtGraph.

**Вынікамі з'яўляюцца** **—** прыкладанне для апрацоўкі светлавых крывых і знаходжання транзітаў экзопланет на Python.

**Вобласць ужывання** **—** прадказанне існавання экзапланет па светлавым крывым зорак нашай галактыкі.

ESSAY

Course work, 34 p., 25 illustrations.

**Keywords**: PYQT, PYQTGRAPH, EXOPLANET, TRANSIT METHOD, ASTRONOMY, PYTHON.

**Object of research —** the object of research is the capabilities of PyQt to create a convenient interface for researchers of light curves of stars. As the subject of study, we select the development and testing of the capabilities of the PyQt and PyQtGraph libraries in Python.

**Purpose —** to design an application for processing and analysis of light curve and to implement it in Python using the PyQt and PyQtGraph libraries.

**Methods of research** **—** a) theoretical: the study of the literature devoted to the processing and analysis of time series; b) practical: designing an application using PyQt Designer and developing it in Python using PyQt and PyQtGraph libraries.

**The results are** **—** an application for processing light curves and finding transits of exoplanets in Python.

**Scope** **—** prediction of the existence of exoplanets from the light curves of the stars of our galaxy.

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 7](#_Toc39328975)

[**Глава 1** **ЭКЗОПЛАНЕТЫ И МЕТОДЫ ИХ ОБНАРУЖЕНИЯ** 9](#_Toc39328976)

[**1.1 Экзопланеты** 9](#_Toc39328977)

[**1.1.1** **Типы экзопланет** 9](#_Toc39328978)

[**1.2. Методы обнаружения экзопланет** 10](#_Toc39328979)

[**1.2.1 Метод Доплера (радиальных скоростей)** 10](#_Toc39328980)

[**1.2.2 Гравитационное микролинзирование** 11](#_Toc39328981)

[**1.2.3 Транзитный метод** 12](#_Toc39328982)

[**Глава 2** **ГРАФИЧЕСКИЙ ФРЕЙМВОРК QT И ОСОБЕННОСТИ ЕГО ПРИМЕНЕНИЯ В ЯЗЫКЕ PYTHON** 15](#_Toc39328983)

[**2.1. Фреймворк Qt** 15](#_Toc39328984)

[**2.1.1 Общий обзор Qt** 15](#_Toc39328985)

[**2.1.2 История развития фреймворка Qt** 17](#_Toc39328986)

[**2.2. Особенности применения фреймворка Qt на языке Python и библиотека PyQt** 18](#_Toc39328987)

[**2.2.1 Общий обзор PyQt** 18](#_Toc39328988)

[**2.2.2 PyQt vs PySide2** 19](#_Toc39328989)

[**Глава 3** **ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРИЛОЖЕНИЯ НА PYQT ДЛЯ АНАЛИЗА СВЕТОВЫХ КРИВЫХ** 22](#_Toc39328990)

[**3.1. Проектирование приложения** 22](#_Toc39328991)

[**3.1.1 Пользовательские истории** 22](#_Toc39328992)

[**3.1.2 Графический интерфейс пользователя** 23](#_Toc39328993)

[**3.1.3 Внутренняя архитектура приложения** 24](#_Toc39328994)

[**3.2. Разработка приложения** 25](#_Toc39328995)

[**3.2.1 Реализация основных методов класса MainWindow** 25](#_Toc39328996)

[**3.2.2 Реализация устранения долгопериодических колебаний звезды** 26](#_Toc39328997)

[**3.2.3 Реализация нарезки графика для изучения периодичности транзита** 27](#_Toc39328998)

[**3.2.4 Реализация сохранения показателей выбранных областей** 27](#_Toc39328999)

[**Глава 4** 30](#_Toc39329000)

[**ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ** 30](#_Toc39329001)

[**3.1. Тестирование основных методов приложения по работе с файлами и графиками** 30](#_Toc39329002)

[**3.2. Тестирование устранения долгопериодических колебаний звезды** 33](#_Toc39329003)

[**3.3. Тестирование нарезки графика по выделенным областям** 34](#_Toc39329004)

[**3.4. Тестирование сохранения результатов исследований** 36](#_Toc39329005)

[**ЗАКЛЮЧЕНИЕ** 40](#_Toc39329006)

[**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ** 42](#_Toc39329007)

**ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ**

|  |  |
| --- | --- |
| а.е. | астрономическая единица, равная 149 597 871 км или радиусу орбиты Земли, |
| TCE | события изменения потока звезды, похожие на транзиты(**T**hreshold **C**rossing **E**vent), |
| WYSIWYG | что видишь, то и получишь(**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et), |
| MOC | метаобъектный компилятор(**M**eta **O**bject **C**ompiler), |
| GUI | графический интерфейс пользователя(**G**raphical **U**ser **I**nterface), |
| IDE | интегрированная среда разработки(**I**ntegrated **D**evelopment **E**nvironment), |
| SQL | язык структурированных запросов(**S**tructured **Q**uery **L**anguage), |
| XML | расширяемый язык разметки(e**X**tensible **M**arkup **L**anguage), |
| HTML | язык гипертекстовой разметки(**H**yper**T**ext **M**arkup **L**anguage), |
| SVG | масштабируемая векторная графика(**S**calable **V**ector **G**raphics); |
| GPL | универсальная общедоступная лицензия(**G**eneral **P**ublic **L**icense); |
| LGPL | меньшая универсальная общедоступная лицензия(**L**esser **G**eneral **P**ublic **L**icense). |

# **ВВЕДЕНИЕ**

4 октября 1957 года в СССР был произведён запуск первого в мире искусственного спутника Земли – ПС-1. Этот день был признан Международной федерацией астронавтики Днём начала космической эры человечества. С началом новой эры пришли также новые задачи и вопросы. Как сформировалась Земля? Почему её климатические условия являются такими, какие они есть? Какие параметры и конфигурации привели к зарождению жизни? Почему жизнь на Земле такая, какая она есть? Существуют ли другие планеты, условия которых благоприятствуют зарождению жизни?

Но есть намного более важный аспект, особенно актуальный в последнее время. Экология Земли резко ухудшается. Геополитическая ситуация также оставляет желать лучшего. Если на Земле произойдёт катаклизм, как человечество его переживёт? Если Земля станет непригодной на жизни, человечеству придётся найти новый дом. Приведу цитату Илона Маска: «Самое главное – создать самостоятельно развивающийся город на Марсе. Это, я думаю, самое важное для того, чтоб максимально продлить существование нашей цивилизации». Конечно, даже в пределах Солнечной системы есть несколько вариантов и Марс один из них. Но они и близко не повторяют условия Земли.

На все эти вопросы человечеству поможет ответить наблюдения за звёздными системами, похожими на нашу. Большое количество наземных и орбитальных телескопов направлены на далёкие звезды в попытках найти новые экзопланеты и среди них отыскать ту, которая окажется обитаемой. Существует множество волонтёрских проектов, например, Planet Hunters или Project Discovery, в которых любой желающий может принять участие в попытках обработать то громадное количество данных, которое за всё это время создали различные наблюдения за звёздами.

Разрабатываемое приложение должно помочь решить проблему по обработке данных для нахождения экзопланет. Оно помогает найти транзиты по данным, полученным при помощи транзитного метода, который более подробно будет описан в главе 1.

Объектом исследования являются возможности PyQt для создания удобного интерфейса для исследователей световых кривых звёзд, а также разработка и тестирование возможностей библиотек PyQt и PyQtGraph на Python.

Методы исследования, которые применялись в работе, включают в себя изучение литературы, посвященной обработке и анализу временных последовательностей, проектирование приложения на PyQt Designer и последующая его разработка на Python при помощи библиотек PyQt и PyQtGraph.

В главе 2 будет рассмотрен графический фреймворк Qt и особенности его применения в языке Python, в частности особенности библиотеки PyQt.

В главе 3 будут рассмотрены процессы проектирования и разработки приложения на PyQt для анализа световых кривых, в частности использование вспомогательной библиотеки PyQtGraph, предназначенной для высокопроизводительной обработки графиков и временных последовательностей.

В главе 4 будет проведено несколько экспериментов с реальными данными для тестирования нашего приложения.

# **ЭКЗОПЛАНЕТЫ И МЕТОДЫ ИХ ОБНАРУЖЕНИЯ**

## **1.1 Экзопланеты**

Экзопланета, или внесолнечная планета – это планета, находящаяся вне Солнечной системы. На сегодняшний день в общей сложности достоверно подтверждено существование 2355 экзопланет, из которых 361 находятся в так называемой «зоне Златовласки» - промежуток возможных орбит вокруг родительской звезды, в которой планета будет иметь условия близкие к условиям Земли, позволяющие воде существовать в жидком виде. Но кандидатов в экзопланеты, которых ещё повторно не зарегистрировали при помощи телескопа после анализа световых кривых, намного больше [2].

Общее количество экзопланет в нашей галактике оценивается примерно в 100 миллиардов, из которых от 5 до 20 миллиардов могут являться «землеподобными».

* + 1. **Типы экзопланет**

Среди экзопланет выделяются несколько типов. Перечислим наиболее часто встречающиеся:

1. «горячие юпитеры» - экзопланеты, представляющие собой газовые гиганты наподобие Юпитера, которые в ходе своей эволюции затормозились об протопланетарное облако и сместились к родительской звезде на дистанцию в диапазоне от 0,04 а.е. до 0,10 а.е. Они представляют собой большинство открытых экзопланет, за счёт их лёгкого обнаружения из-за заметных короткопериодических возмущений в движение звезды. Кроме того, достаточно велика вероятность прохождения планеты перед звездой, что позволяет обнаружить её за счёт уменьшения светимости звезды. Первая экзопланета 51 Pegasi b, открытая в 1995 году астрономами Мишелем Майором и Дидье Кело в Обсерватории Верхнего Прованса, также является «горячим юпитером»;
2. «суперземля» - класс планет, превышающих Землю по массе в 5 – 10 раз. Обнаружены преимущественно на орбитах звёзд небольшой массы, менее солнечной, относящихся к оранжевым и красным карликам. Предполагаются основными кандидатами на наличие внеземной жизни. Первая планета такого класса была обнаружена на орбите звезды Gliese 876. Масса планеты была равна 7,5 массам Земли, а её орбитальный период был равен всего 2 суткам;
3. «супер-юпитер» - класс планет, имеющих массу от 10 и более масс Юпитера. Они находятся на гипотетической границе между планетой и звездой. В большинстве своём они имеют радиус, близкий к радиусу Юпитера, но их плотность достигает невероятных значений. Так, например, Corot-3b с массой равной 22 массам Юпитера, имеет предсказанную плотность, равную 26400 кг/куб.м (к примеру, наша Земля имеет плотность 5510 кг/куб.м, а Юпитер – 1330 кг/куб.м).

## **1.2. Методы обнаружения экзопланет**

Существует множество методов обнаружения планет, но здесь мы рассмотрим только те, которые принесли наибольшее количество результатов.

### **1.2.1 Метод Доплера (радиальных скоростей)**

Данный метод основывается на наблюдении за спектром звезды. За счёт притяжения планеты родительская звезда будет двигаться по небольшой орбите вокруг общего центра масс – барицентра. Если плоскость орбиты планеты не находится перпендикулярно лучу нашего зрения, это приведёт к изменению её радиальной скорости. Это можно зарегистрировать при помощи эффекта Доплера, за счёт которого линии спектра звезды начинают смещаться при изменении её радиальной скорости относительно нас (Рисунок 1.1).

Данный метод имеет большие недостатки. Некоторые звёзды имеют активную фотосферу, т.е. они могут расширяться и уменьшаться в размере с течением времени. Такую активность можно легко спутать с движением звезды за счёт присутствия массивной планеты. На сегодняшний день он практически исчерпал свои возможности и в основном используется для дополнительного подтверждения открытий, сделанных при помощи транзитного метода.

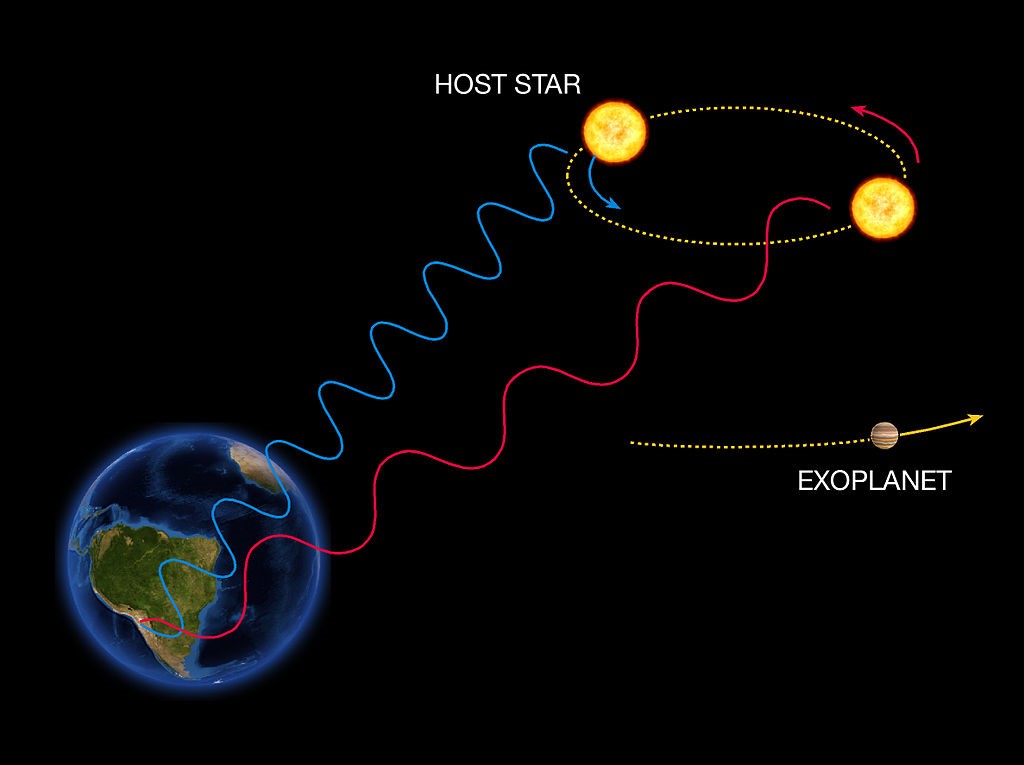


Рисунок 1.1 - Эффект Доплера

### **1.2.2 Гравитационное микролинзирование**

Суть гравитационного линзирования заключается в искривлении пространства гравитационными полями массивного тела. За счёт искривления пространства свет объектов, находящихся за массивным телом, преломляется и то место, где мы видим объект, отличается от его реального положения. Однако если массивное тело и объект находятся на одной прямой с нами, то может получиться эффект линзирования, когда свет от объекта размазывает по кольцу вокруг массивного тела (Рисунок 1.2).

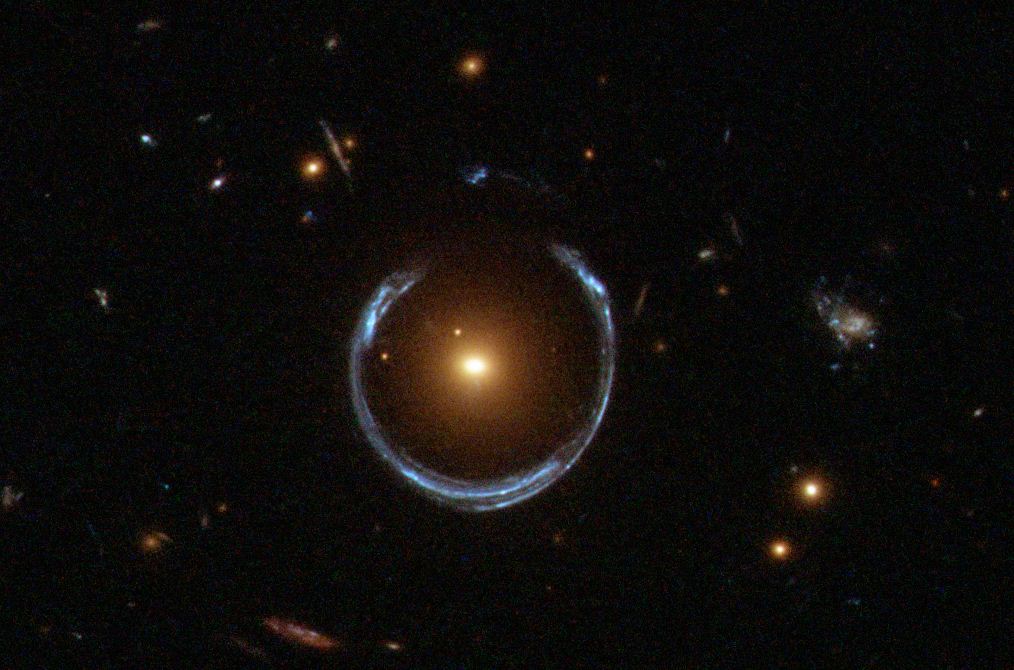


Рисунок 1.2 – Эллиптическая галактика линзирует другую галактику

Гравитационное микролинзирование возникает в том случае, если две звезды находятся на одной прямой вместе с нами. Планета ближней звезды за счёт своей массы вносит дополнительный эффект искривление света от дальней звезды, который можно зарегистрировать при помощи достаточно точных приборов (Рисунок 1.3).

Однако данный метод обладает значительными недостатками. Такой эффект можно получить тогда и только тогда, когда две звезды находятся на одной прямой, и длится он всего несколько дней, поскольку звёзды и Земля постоянно движутся относительно друг друга. Кроме того, найденные по этому методу планеты зачастую находятся на расстоянии в несколько тысяч световых лет от нас, так что проверка достоверности при помощи других методов является невозможной.

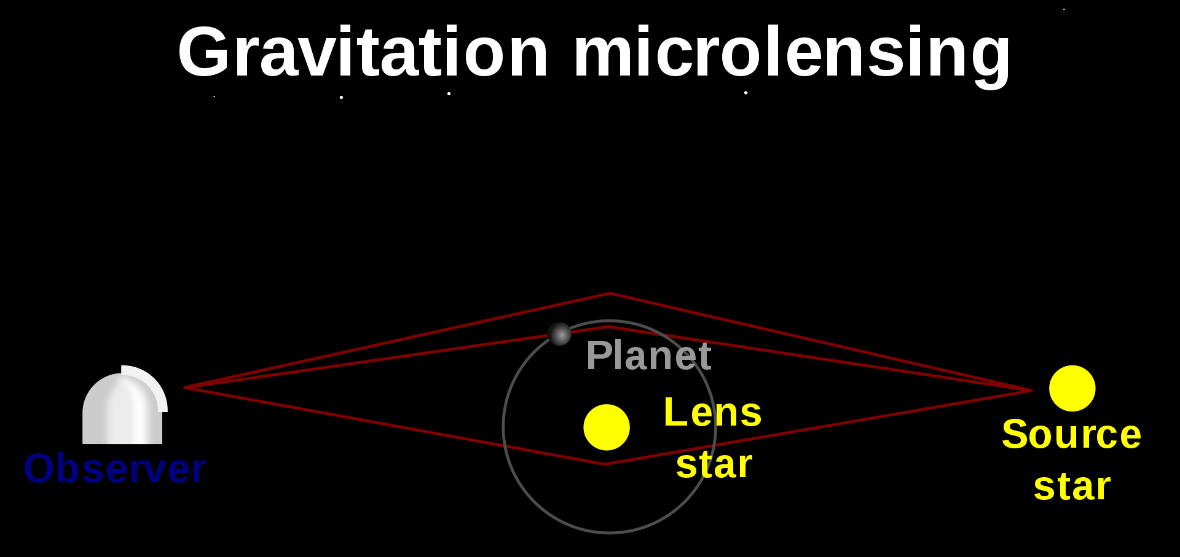


Рисунок 1.3 – Пример эффекта гравитационного микролинзирования

Чаще всего этот метод используют по отношению к звёздам, расположенным между Землёй и центром нашей галактики, так как галактический центр имеет большую плотность звёзд. За счёт этого можно увидеть, как одна и та же звезда проходит перед большим количеством других звёзд, что позволяет пронаблюдать эффект микролинзирования несколько раз.

### **1.2.3 Транзитный метод**

А теперь перейдём к основному методу обнаружения экзопланет и поговорим о нём поподробнее, так как данный метод лежит в основе данной курсовой работы.

Данный метод основывается на наблюдении за светимостью звёзд на продолжительном отрезке времени. Когда перед звездой пройдёт планета, она заслонит собой часть звезды и значение светимости, которое наблюдается, упадёт (Рисунок 1.4). На графике мы это увидим в виде небольшого углубления (Рисунок 1.5).

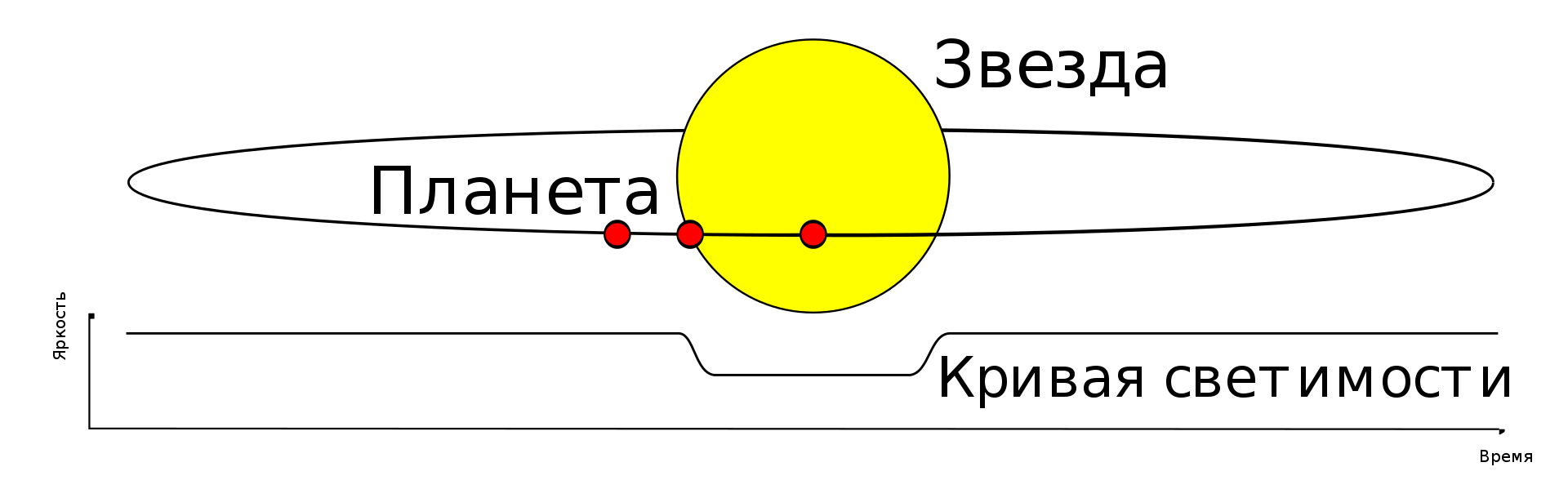


Рисунок 1.4 – Пример транзита экзопланеты

Данный метод имеет дополнительное применение. Во время транзита свет звезды проходит сквозь атмосферу планеты, что позволяет нам при помощи анализа спектра звезды без транзита и спектра с транзитом понять состав атмосферы планеты.

Однако и у него есть свои недостатки. Первый недостаток – плоскость орбиты планеты должна иметь крайне маленький угол наклонения относительно нашего луча зрения. Но количество звёзд настолько велико, что даже несмотря на эту проблему всё равно было найдено значительное количество экзопланет. Второй недостаток заключается в большом количестве «ложных транзитов», связанных с активностями звёзды, например, пульсациями или периодичные затмения в случае двойных звёздных систем.

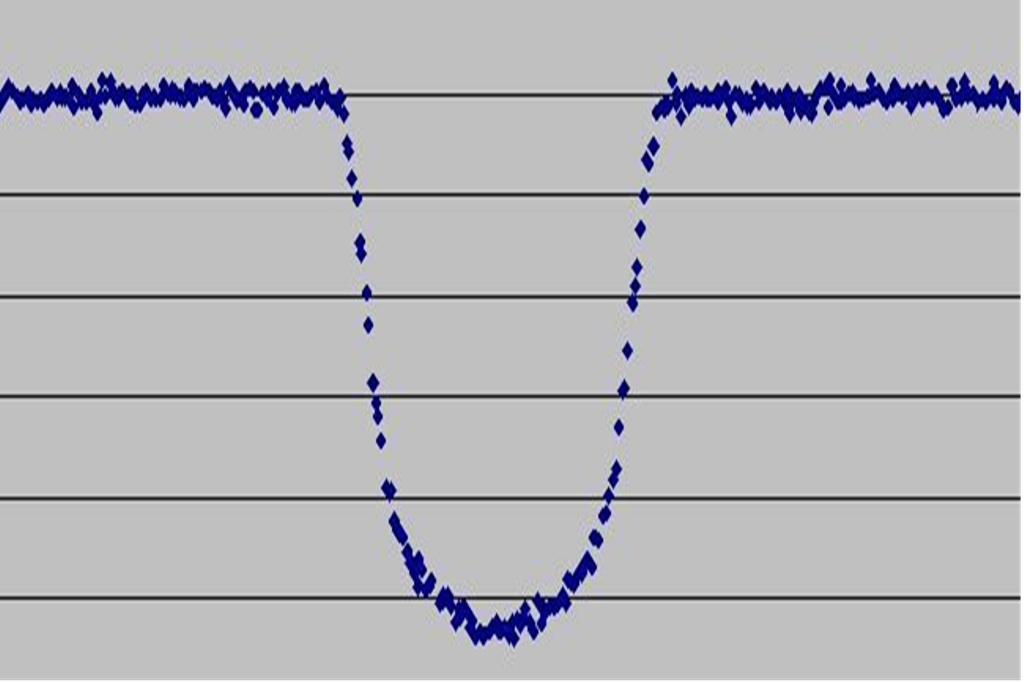


Рисунок 1.5 – Пример транзита экзопланеты на графике светимости звезды

На сегодняшний день данный метод является единственным методом обнаружения экзопланет с высокой достоверностью.

Для изучения транзитов было создано множество проектов. Среди них COROT и TRAPPIST. Наиболее известным из них является Kepler. В ходе этого проекта в 2009 году на солнечную орбиту был выведен телескоп Kepler со сверхчувствительным фотометром, специально предназначенный для поиска экзопланет. Его размеченные учёными данные будут использоваться в ходе данной курсовой работы для обучения и тестирования нейронной сети.

**ВЫВОДЫ**

1. Пояснена предметная область исследования для данного курсового проекта. Описаны основные типы экзопланет, встречающихся в полученных данных.
2. Описаны методы открытия новых экзопланет. Выделены достоинства и недостатки основных методов открытия экзопланет. Проведено относительное сравнение методов и выявлен наилучший – транзитный метод.

# **ГРАФИЧЕСКИЙ ФРЕЙМВОРК QT И ОСОБЕННОСТИ ЕГО ПРИМЕНЕНИЯ В ЯЗЫКЕ PYTHON**

## **2.1. Фреймворк Qt**

### **2.1.1 Общий обзор Qt**

Qt (произносится как “cute”, т.е. “кьют”, а не “кьюти”, как многие произносят по ошибке) – это кроссплатформенный фреймворк с открытым исходным кодом для разработки программного обеспечения, который сперва появился как исключительно фреймворк для языка C++, но позже был распространён и на многие другие языки. Появился он в 1996 благодаря разработкам компании Trolltech. Также стоит отметить, что он лежит в основе рабочей среды KDE, которой пользуются для разработки множества различных дистрибьютивов Linux.

Основная особенность Qt заключается в его кроссплатформенности. Он позволяет запускать программного обеспечение, написанное на нём, на большинстве существующих операционных систем за счёт компиляции программы без каких-либо изменений в исходном коде. Достигается это при помощи использования метаобъектного компилятора – предварительной системы обработки исходного кода. Исходный код обрабатывается MOC, который ищет в классах программы макрос Q\_OBJECT и переводит код в мета-объектный, после чего мета-объектный код компилируется на языке С++.

Включает Qt в себя не только классы, необходимые для создания элементов графических интерфейсов, но и классы, необходимые, например, для работы с базами, данных, сетевыми протоколами и т.п.

В комплекте с Qt также идёт среда разработки графических интерфейсов Qt Designer по принципу WYSIWYG (Рисунок 2.1), т.е. в процессе редактирования и создания видим интерфейс, который будет максимально похож на финальную продукцию. Кроме этого, в комплекте с ним идут такие инструменты как Qt Linguist (графическая утилита, позволяющая упростить процесс локализации и перевода на другие языки), Qt Assistant (справочная система Qt, позволяющая удобно работать с документацией), Qt Creator (редактор кода, который также включает в себя справку, графические средства Qt Designer и отладчик приложений).

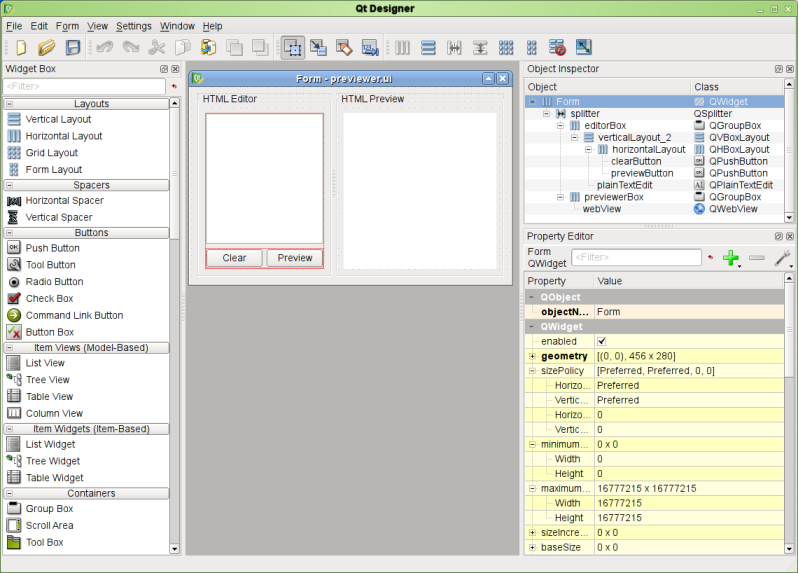


Рисунок 2.1. Qt Designer

Подытоживая, в Qt можно выделить следующие ключевые концепции:

* Полная абстракция GUI – когда Qt был впервые выпущен, он использовал свой собственный механизм рисования и элементы управления, эмулируя внешний вид различных платформ, на которых он работает, когда он отрисовывает свои виджеты. Это облегчило работу по портированию, потому что очень мало классов в Qt действительно зависело от целевой платформы; однако, это иногда приводило к небольшим несоответствиям там, где эта эмуляция была несовершенной. Последние версии Qt включают в себя нативные API для различных платформ.
* Сигналы и слоты – удобная и эффективная схема взаимодействия между объектами, которая позволяет легко реализовать такой шаблон проектирования как наблюдатель, а также позволяет избежать множественных дубликаций кода. Элементы графического интерфейса посылают сигналы, содержащие в себе информацию о произошедшем событии (например, нажатие кнопки), которые перехватываются другими контрольными элементами при помощи специальных функций, называемых “слоты”;
* Использование MOC – исходный код обрабатывается MOC, который ищет в классах программы макрос Q\_OBJECT и переводит код в мета-объектный, после чего мета-объектный код компилируется на языке С++;
* Языковые привязки (language bindings) – API, позволяющий языку программирования использовать библиотеки других языков программирования, а также сервисы и функции операционных систем, не являющихся нативными для данного языка программирования.

### **2.1.2 История развития фреймворка Qt**

История Qt началась летом 1990 года, когда Хаарварт Норт и Эрик Чамбе-Энг работали вместе над проектом, связанным с базой данных ультразвуковых изображений, на C++. Именно они начали разработку Qt и 3 года спустя данный проект был взят под крыло компанией Quasar Technologies, которая затем сменила имя на Trolltech. Сперва Qt предоставлялся лишь на 2-х платформах: Unix и Windows.

Самая первая версия Qt 0.90 for X11/Linux была выпущена 20 мая 1995 года под лицензией Qt Free Edition License. Данная лицензия отличалась от Free Software Foundation лицензии, т.к. в то время как исходный код был доступен открыто, перераспространение изменённых версий было запрещено. Кроме того, версия Qt для Windows была доступна только под несвободной лицензией, что означало, что приложения, написанные для X11/Linux невозможно портировать на Windows без приобретения платной версии.

Это вызывало некоторые разногласия, которые привели к компромиссу между FSF и Trolltech, заключающийся в создании новой лицензии, согласно которой, если в течении года не будет выпущена версия Qt под лицензией GNU GPL, то Qt попадёт под BSD лицензию, которая накладывает минимальные ограничения на распространение исходного кода. В результате в 2000 году была выпущена версия Qt 2.2 для X11/Linux под GNU GPL лицензией. В 2001 году была выпущена версия Qt 3.0, добавляющая поддержку Mac OS, но только в платной версии, а в 3.2 поддержка Mac OS стала доступной под лицензией GNU GPL.

Версия Qt 4.0 принесла обновление лицензии для платформы Windows, позволяющее использовать данное программное обеспечение под лицензией GNU GPL. С данного момента можно считать, что фреймворк Qt окончательно стал проектом с открытым исходным кодом.

После выпуска версии Qt 4.0 Trolltech была куплена компанией Nokia в 2008 году. Однако в связи с провалом продвижения Symbian, в 2011 году Nokia продала коммерческую лицензию Qt компании Digia.

Последнее глобальное изменение архитектуры Qt произошло с выпуском версии Qt 5.0 19 декабря 2012 года, которая принесла с собой аппаратное ускорение графики, QML и JavaScript. Традиционные QWidgets только на C++ продолжали поддерживаться, но не получали выгоды от повышения производительности, доступное благодаря новой архитектуре.

В 2014 году Digia передала права на Qt своей дочерней компании, The Qt Company, которая владела 25 брендами, связанными с Qt.

В последнее время большое количество различных компаний участвуют в открытой разработке Qt. В их число входят Intel, AudioCodes, KDAB, KDE и другие.

## **2.2. Особенности применения фреймворка Qt на языке Python и библиотека PyQt**

Несмотря на то что сперва Qt был задуман для использования на языке С++, со временем благодаря его популярности и широкому функционалу появились и наборы расширений – языковые привязки – которые позволяют использовать данный фреймворк и на других языках. В частности, для языка Python была создана библиотека PyQt.

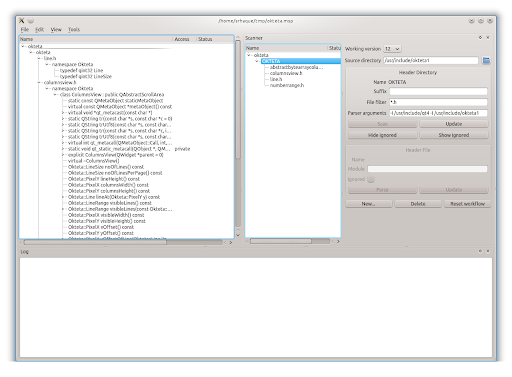
### **2.2.1 Общий обзор PyQt**

Итак, PyQt – это библиотека привязок фреймворка PyQt для языка Python. Она была разработана и поддерживается компанией Riverbank Computing. При разработке PyQt была использована технология SIP. SIP – это инструментарий для автоматической генерации привязок на языке Python к C и C++ библиотекам. Данная технология была разработана в 1998 году как раз для создания PyQt. Сейчас данную технологию можно использовать для создания привязок к любым библиотекам из C и C++.

Для генерации привязок SIP берёт набор инструкций, который хранится в файле формата .sip, описывающий код и генерирует C++ код. Данный код компилируется и конвертируется в код на Python. В частности, для последних версий PyQt используется внутренний инструмент metasip, который представляет собой что-то наподобие IDE для SIP (Рисунок 2.2).

PyQt включает в себя практически все возможности фреймворка Qt:

* набор виджетов графического интерфейса;
* стили виджетов;
* доступ к базам данных с помощью SQL;
* поддержку интернационализации;
* парсер XML;
* интеграцию с WebKit, движком рендеринга HTML;
* поддержку воспроизведения видео и аудио;
* поддержка SVG.

Рисунок 2.2. Metasip

У PyQt есть лишь один небольшой недостаток – по сравнению с Qt, данная библиотека распространяется не под LGPL лицензией, а под GPL лицензией, что немного ограничивает разработчиков при создании нового программного обеспечения на основе PyQt. Но в реальности для Qt существует 2 привязки – PyQt и PySide2. Между ними существует несколько существенных различий, в том числе и различия в лицензии.

### **2.2.2 PyQt vs PySide2**

Самой большой и важной разницей для большинства компаний между двумя данными привязками является лицензия. PyQt распространяется под лицензией GPL, в то время как PySide2 поставляется под лицензией LGPL. Если сравнивать 2 данные лицензии:

* GPL – приложения созданные при помощи библиотеки под данной лицензией должны обязательно иметь открытый доступ к коду и, что более важно, созданные приложения должны иметь GPL-совместимую лицензию, что в большинстве случаев является неприемлемым для коммерческих проектов;
* LGPL – приложения созданные при помощи библиотеки под данной лицензией могут быть закрытыми, единственным требованием является создание инструкции по замене данной библиотеке на библиотеку с GPL лицензией.

Как мы видим, в случае разработки коммерческого проекта существует 2 варианта: либо купить коммерческую лицензию у Riverbank Computing и использовать PyQt, либо использовать PySide2, однако в обоих случаях может также возникнуть необходимость купить коммерческую лицензию Qt.

Для некоммерческих проектов стоит учитывать и другие различия между этими привязками. Самым важным является то, что для PyQt создано большое количество удобных инструментов и библиотек (в том числе PyQtGraph, которая будет обсуждаться немного позже), в то время как у PySide2 нет даже специализированного инструмента для разработки.

Кроме этого, API в PyQt является чуть более широким и покрывающим чуть больше фреймворка Qt (как например в случае функции qmlRegisterSingletonType, которая отсутствует в PySide2).

Если рассматривать данные привязки со стороны поддержки и сообщества, то здесь также есть существенная разница. PySide2 является ныне проектом The Qt Company, которая является достаточно крупной и имеет большое количество иных проектов, связанных с Qt, под своим крылом, в то время как Riverbank Computing (которая по сути является всего лишь небольшой консалтинговой фирмой) по некоторой информации содержит в себе штат всего в 1 сотрудника. Отказность поддержки у такой системы является крайне большой.

Также у данных привязок есть различие в генераторе привязок для Python. В случае PyQt, как уже упоминалось в этой работе, им является Metasip, в случае же PySide2 им является Shiboken. Между ними есть большая разница в процессе генерации и получаемых шаблонах, но ключевой разницой является то, что Metasip может работать как с библиотеками C++, так и с библиотеками C, в то время как Shiboken был разработан специально под C++.

При проектировании приложения нередко бывает сложно сделать выбор между 2-мя данными привязками. Данную сложность разрешают наборы обёрток, которые позволяют практически на ходу “поменять колёса” и перейти с PyQt на PySide2 и обратно. Одним из наиболее известных наборов является проект QtPy.

В данном проекте выбор был сделан в пользу PyQt на основании как раз-таки количества удобных инструментов и библиотек. Для PyQt была создана библиотека PyQtGraph, которая является высокопроизводительным инструментов для построения и редактирования графиков и временных кривых.

**ВЫВОДЫ**

* + - 1. Был проведён общий обзор возможностей и особенностей фреймворка Qt.
      2. Была рассмотрена история развития фреймворка Qt и пояснена его нынешняя популярность.
      3. Рассмотрены особенности применения фреймворка Qt на языке Python.
      4. Рассмотрены две различные привязки для Qt для языке Python, проведено их сравнение и сделан выбор одной из привязок для разработки данного проекта.
      5. Рассмотрены особенности выбранной привязки PyQt и её возможности.

# **ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРИЛОЖЕНИЯ НА PYQT ДЛЯ АНАЛИЗА СВЕТОВЫХ КРИВЫХ**

Перед тем как приступить к разработке приложения всегда сперва необходимо определить необходимый функционал для имплементации и проанализировать технические возможности доступных инструментов для определения возможных реализаций затребованного функционала.

## **3.1. Проектирование приложения**

Для данного проекта описание функционала можно осуществить через пользовательские истории. В случае пользовательских историй существует возможность создать примерное описание пользовательского интерфейса: необходимых элементов управления, показателей и окон, на основании которых можно впоследствии спроектировать и внутреннюю архитектуру приложения.

### **3.1.1 Пользовательские истории**

В ходе анализа существующих решений, а также известных проблем, связанных с исследованиями световых кривых, были созданы следующие пользовательские истории:

* как пользователь, я могу открыть файл формата .fits, содержащий в себе информацию о световой кривой и её временную последовательность;
* как пользователь, я могу увидеть отображённую световую кривую на своём экране;
* как пользователь, я могу выделить область, которая является подозрительной на транзит или TCE, на графике;
* как пользователь, я могу выделить множество подозрительных на транзит или TCE областей на графике;
* как пользователь, я могу сложить график по выделенным подозрительным областям графика для более детального анализа возможности существования транзита в данных подозрительных областях;
* как пользователь, я могу увидеть показатели подозреваемых на транзит областей, такие как период и смещение первой найденной области относительно начала наблюдений;
* как пользователь, я могу изменять показатели подозреваемых на транзит областей с последующим отображением новых полученных подозреваемых на транзит областей;
* как пользователь, я могу сохранить показатели для выбранных на графике областей;
* как пользователь, я могу удалить сохранённые показатели для выбранных на графике областей;
* как пользователь, я могу очистить график от отмеченных как подозрительных областей;
* как пользователь, я могу убрать долгопериодические колебания звезды из графика для упрощения нахождения подозрительных на транзит областей.

На основании данных пользовательских историй стали известны общая концепция и функционал приложения, на основании которых стало возможным построить приблизительный макет для GUI данного приложения.

### **3.1.2 Графический интерфейс пользователя**

GUI для данного приложения был создан при помощи Qt Designer – инструмента фреймворка Qt, который упоминался ранее в главе 1. На рисунке 3.1 можно увидеть спроектированный GUI.

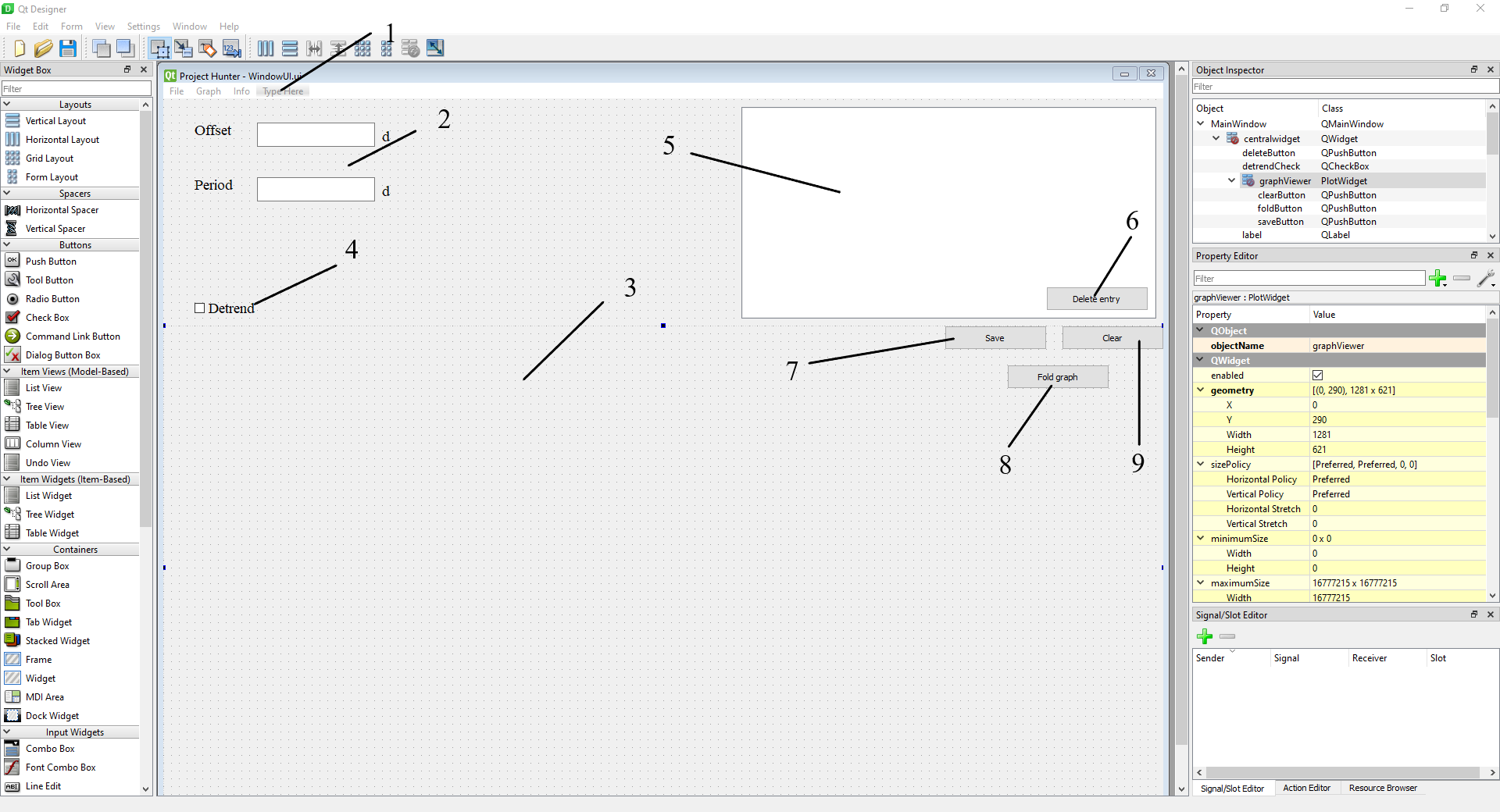


Рисунок 3.1. GUI приложения

Стоит заметить, что на данном шаблоне можно видеть все контрольные элементы проектируемого приложения:

Меню приложения. В нём есть опции работы с приложением, файлами, а также загруженной световой кривой.

Элементы для вывода и изменения показателей выделенных подозрительных областей.

Элемент для отображения графика. В данном случае использован особый элемент из библиотеки PyQtGraph, поэтому Qt Designer не может его корректно отобразить. Для создания данного элемента необходимо сперва создать элемент класса QWidget, после чего “повысить” его до необходимого нам объекта.

Элемент для управления детрендом графика. Именно этот элемент позволит отбрасывать долгопериодические колебания звезды на графике.

Элемент для отображения сохранённых транзитов и их показателей.

Элемент управления для удаления ненужных пользователю сохранённых транзитов.

Элемент управления для сохранения выделенных областей как информации о транзите.

Элемент управления для контроля складывания/раскладывания графика по выделенным областям.

Элемент управления для очистки графика от выделенных областей.

На основе выделенных пользовательских историй и шаблона GUI можно приступить к проектированию внутренней архитектуры и имплементации приложения.

### **3.1.3 Внутренняя архитектура приложения**

Как упоминалось в главе 1, Qt является полностью объектно-ориентированным фреймворком, из чего следует, что для нашего главного окна будет создан отдельный класс, методы которого будут являться реакциями на совершение каких-то действий пользователями. В данном проекте будет всего одно окно и оно будет соответствовать классу MainWindow.

Для объединения нашей внутренней части приложения с графическим интерфейсом, необходимо сгенерировать из .ui файла, который был создан во время проектирования GUI, .py файл, в котором будет содержаться инициализация и связывание всех элементов GUI c классом MainWindow.

Для работы с графиками световых кривых была выбрана библиотека PyQtGraph. Данная библиотека написана полностью на языке Python на основе Qt и numpy. Она предназначена для использования в математических, научных и инженерных приложениях. Распространяется по открытой лицензии MIT. Данная библиотека имеет множество различных применений в построении как 2D, так и 3D представлений.

По сравнению с базовым matplotlib, который используется почти во всех проектах, можно выделить 3 больших различия, за счёт которых и было принято решение использовать PyQtGraph:

* скорость – в случае необходимости быстрого обновления данных или графика, matplotlib не является лучшим выбором, он сильно проигрывает по данному параметру библиотеке PyQtGraph;
* портативность / простота установки – так как PyQtGraph написана полностью на языке Python, это означает, что она работает на любой платформе, на которой поддерживается numpy и Qt без необходимости в предварительной компляции;
* многие дополнительные функции – библиотека PyQtGraph стремится охватить многие аспекты разработки научных и инженерных приложений с помощью более продвинутых функций, таких как нарезка данных на основе областей интереса, параметрические деревья, блок-схемы, многопроцессорная обработка и т.п.;

## **3.2. Разработка приложения**

На основе спроектированного графического интерфейса и внутренней архитектуры приложения можно начать имплементацию необходимого функционала для полного достижения цели, представленной в данной курсовой работе – удобное приложение для анализа световых кривых.

### **3.2.1 Реализация основных методов класса MainWindow**

Класс MainWindow осуществляет реализацию функционала главного окна данного приложения (Приложение Б). Он наследуется от двух классов: QMainWindow и Ui\_MainWindow. QMainWindow является частью библиотеки PyQt, который предоставляет методы и функционал главного окна приложения. Ui\_MainWindow (Приложение А) является сгенерированным инструментом pyuic5 классом из файла WindowUI.ui, который был создан при помощи инструмента Qt Designer, который был использован при создании шаблона графического интерфейса приложения.

Особенное внимание стоит обратить на функцию \_\_init\_\_ и init\_open. В функции \_\_init\_\_ происходит декларация и инициализация полей класса MainWindow. В свою очередь \_\_init\_\_ вызывает 2 функции: setupUI и init\_open. setupUI является унаследованной функцией от класса Ui\_MainWindow, которая производит инициализацию и заполнение графического интерфейса приложения элементами. init\_open осуществляет необходимые преобразования графического интерфейса, в особенности тех элементов, которые будут доступны только в определённых состояниях приложения. Кроме этого в этой функции происходит соединение сигналов со слотами – реализация основной концепции Qt.

Кроме функций основного окна, также существует потребность в переиспользовании некоторых объектов, таких как QPen для рисования линий графика. В данных целях был создан файл utils.py.

Для отображения графика и работы с ним используется объект PlotWidget из библиотеки PyQtGraph. Построение самого графика является очень простым действием – необходимо вызвать функцию plot() либо на самом объекте PlotWidget, либо на объекте PlotItem, который можно извлечь из объекта PlotWidget при помощи функции getPlotItem(). Для добавления выделения (или любых других объектов на график) используется функция addItem() на объекте PlotWidget.

Для реализации базового выделения областей на графике я воспользовался объектом InfiniteLine, которая позволяет помещать на график прямые линии. Для выделения множества областей, а также отрисовки их в реальном времени я воспользовался одним из главных преимуществ библиотеки PyQtGraph – её скорость. Суть подхода заключается в том, что при создании данных линий и графика самой световой кривой в том числе на выходе получается объект, модифицируя который есть возможность также изменять и то, что отрисовывается на рабочей области графика. За счёт этого появляется возможность реагировать в реальном времени на действия пользователя в рабочей области графика.

За счёт возможностей Qt и PyQtGraph реализация базового функционала является достаточно простой и высокоэффективной. Обратим внимание на функционал, реализация которого потребовала использования некоторых алгоритмов и, в первую очередь, дополнительных исследований.

### **3.2.2 Реализация устранения долгопериодических колебаний звезды**

После небольшого исследования для решения проблемы долгопериодических колебаний звёзд был выбран алгоритм скользящего среднего.

Суть его заключается в следующем. Выбирается размер окна, при помощи которого мы будет нормализовать график. Желательно, чтобы размер данного окна был нечётным. В случае световых кривых достаточно эффективным подходом является выбор окна, размер которого равен квадратному корню от размера самой световой кривой. Данное окно скользит по графику и на каждом шаге происходят следующие действия:

* рассчитывается среднее значение координаты Y для точек, входящих в окно;
* идёт перерасчёт координаты Y для точки, находящейся по середине окна по формуле:
* окно сдвигается на 1 точку дальше по графику.

Таким образом обрабатывается весь график и на выходе получается видоизменённый график, который будет иметь в себе только кратковременные перепады, что по сути и является задачей данного приложения: нахождения кратковременных перепадов, которые могут являться транзитами.

### **3.2.3 Реализация нарезки графика для изучения периодичности транзита**

В случае достаточно широкой по времени световой кривой транзит одной и той же планеты может повторяться несколько раз. В таком случае для более подробного изучения мест, имеющих схожие по размеру и форме перепады светимости имеется необходимость в добавлении функционала нарезки графика по данным местам.

Алгоритм сложения достаточно прост – программа проходится по световой кривой и осуществляет преобразование X координаты каждой из точек по формуле:

где – значение координаты Х первого выделенной области,

– расстояние между выделенными областями по координате Х,

– функция округления.

В итоге получается множество точек со значением координаты Х в пределах [, ].

### **3.2.4 Реализация сохранения показателей выбранных областей**

В случае, когда исследователь обнаруживает транзит, но исследование данной световой кривой не было завершено, было бы удобно сохранить предыдущий результат.

Самый базовый вариант сохранения крайне прост – когда исследователь нажимает кнопку “Save”, программа считывает поля смещения первого транзита и периода и записывает их в объект QListWidget. Исследователь также может двойным кликом поднять данную запись и отобразить её на графике.

Однако данный вариант сохраняет результаты исследований исключительно в оперативной памяти компьютера, что означает, что после закрытия программы эти данные будут утеряны. Для решения этой проблемы для этого проекта было решено использовать систему кэширования результатов.

В данном проекте для каждого открываемого файла рассчитывается хеш его содержимого на основе алгоритма криптографического хеширования SHA-1. При завершении работы с файлом программа записывает в файл с именем равным значению хеша данного файла результаты исследований. Данное дополнение к процессу сохранения показателей позволяет исследователям даже при повторном открытии файла после завершения работы получить предыдущие результаты исследований световой кривой, хранящейся в данном файле.

Несмотря на то что в 2017 году Google представил алгоритм получения коллизий для SHA-1, в случае данного проекта этот алгоритм вполне удовлетворяет требования по скорости и шансу случайной коллизии. Здесь нет необходимости полной безопасности, как в случае цифровых подписей на основе SHA-1, от которых настоятельно рекомендуют отказываться и переходить на другие алгоритмы криптографического хеширования.

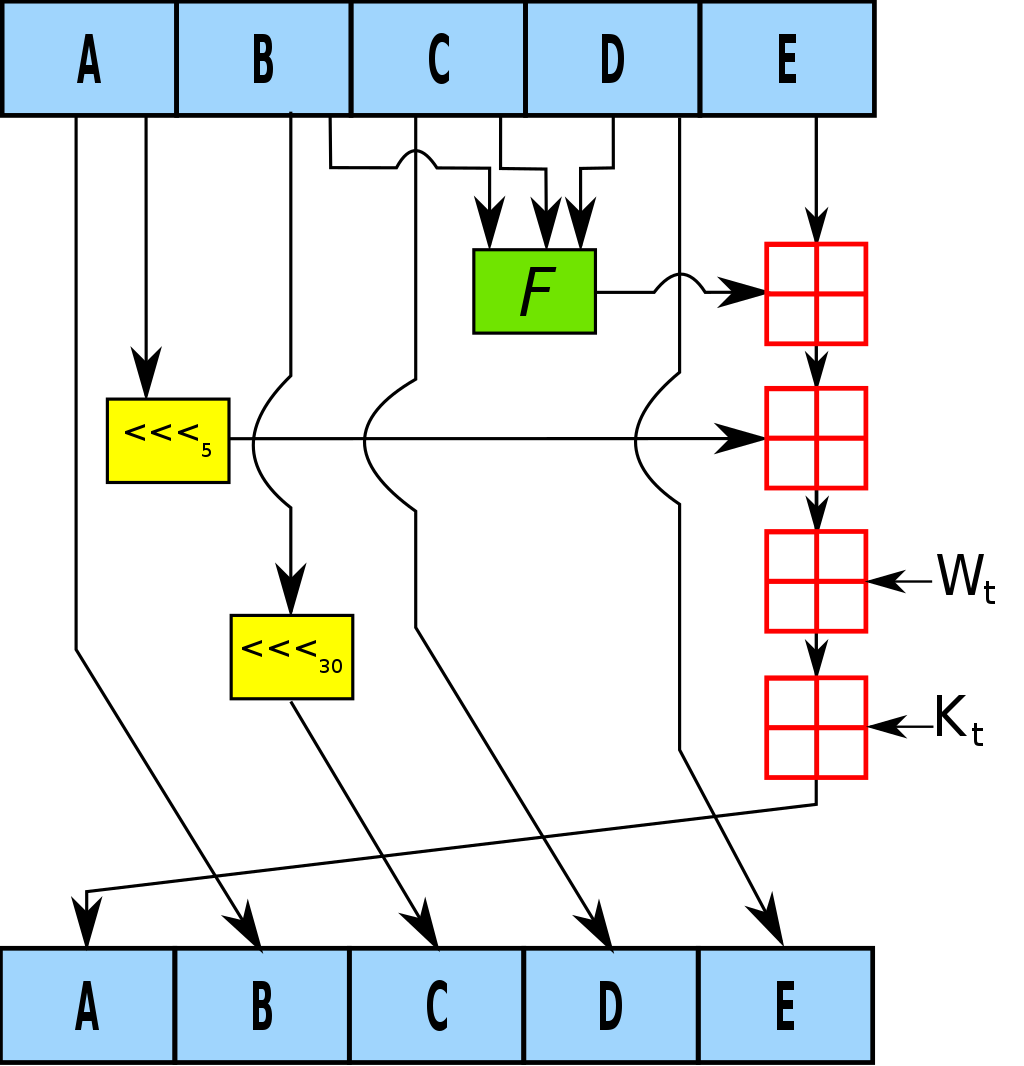


Рисунок 3.2. Итерация алгоритма SHA-1

Краткое описание алгоритма SHA-1 (Рисунок 3.2):

* входной поток разбивается на блоки по 512 бит в каждом, в последнем блоке в последние 64 бита записывается длина входного потока (это накладывает ограничение на максимальную длину потока - бит;
* инициализируется 5 переменных, 4 нелинейные операции и 4 вспомогательные константы, значения которых изначально определены алгоритмом;
* блок сообщения преобразуется из 16 32-битовых слов в 80 32-битовых;
* производятся операции над 5 переменными, инициализированных в начале;
* после совершения всех необходимых итераций над всеми блоками, 5 переменных объединяют и получают результирующий хеш.

**ВЫВОДЫ**

* + - 1. Был проведён анализ проблем исследователей световых кривых и созданы пользовательские истории для решения этих проблем.
      2. Был построен графический интерфейс на основании пользовательских историй, созданных в пункте 1.
      3. Была спроектирована внутренняя архитектура приложения для реализации необходимого функционала.
      4. Были реализованы основные методы класса MainWindow для решения большинства простейших проблем.
      5. Было реализовано устранение долгопериодических колебаний звезды из графика световой кривой.
      6. Была реализована нарезка графика по выделенным подозрительным областям.
      7. Было реализовано сохранение результатов исследований как в оперативной, так и постоянной памяти компьютера.

# 

# **ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ**

После разработки данного проекта необходимо тщательное тестирование, ведь даже мельчайшая недоработка может привести либо к неправильному отображению результатов исследования, либо к повреждению сохраняемых результатов, либо к вовсе неправильным результатам.

## **3.1. Тестирование основных методов приложения по работе с файлами и графиками**

Сценарий: исследователь заходит в программу и открывает файл формата .fits, содержащий данные по световой кривой звезды.

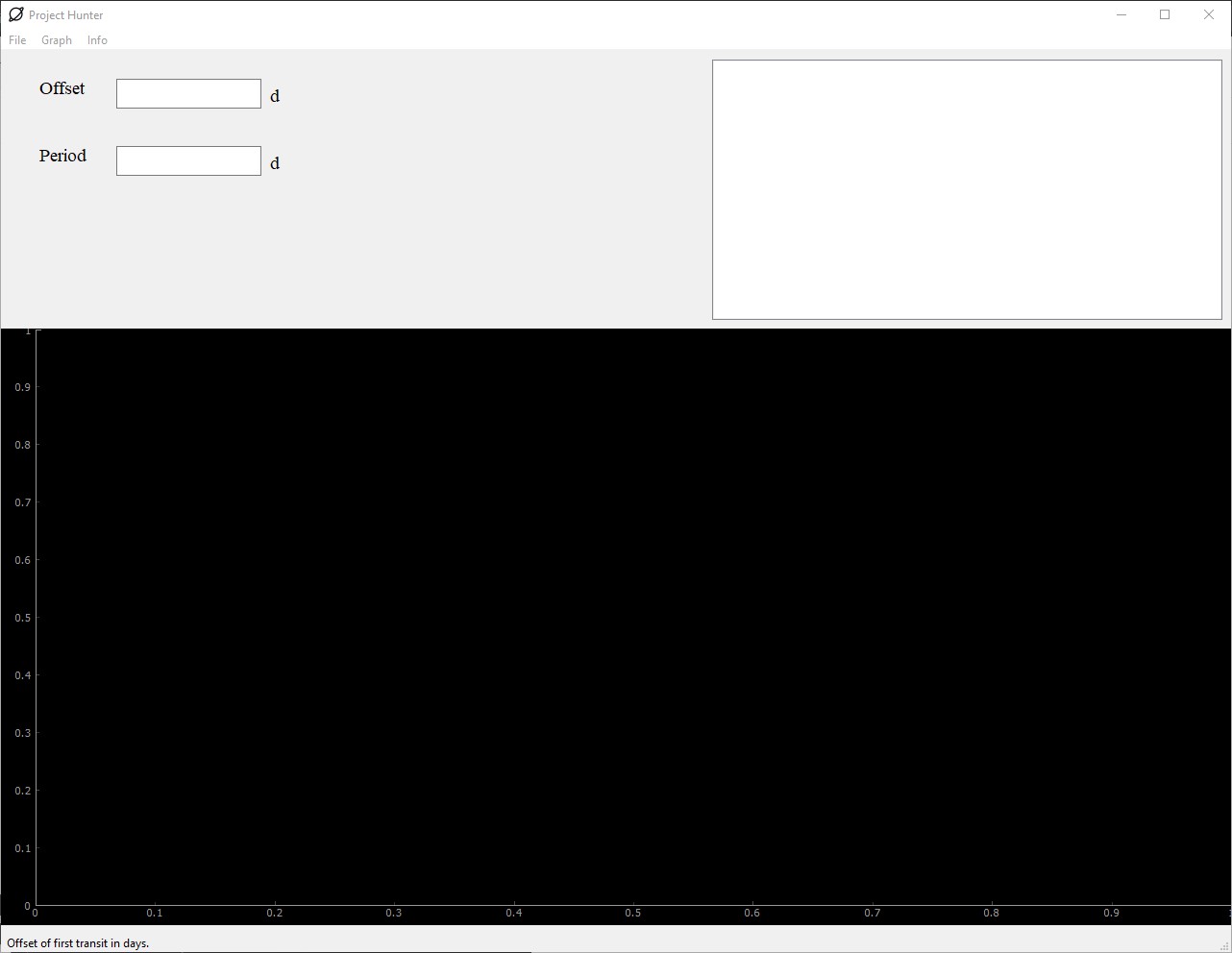


Рисунок 4.1. Начальное состояние приложения

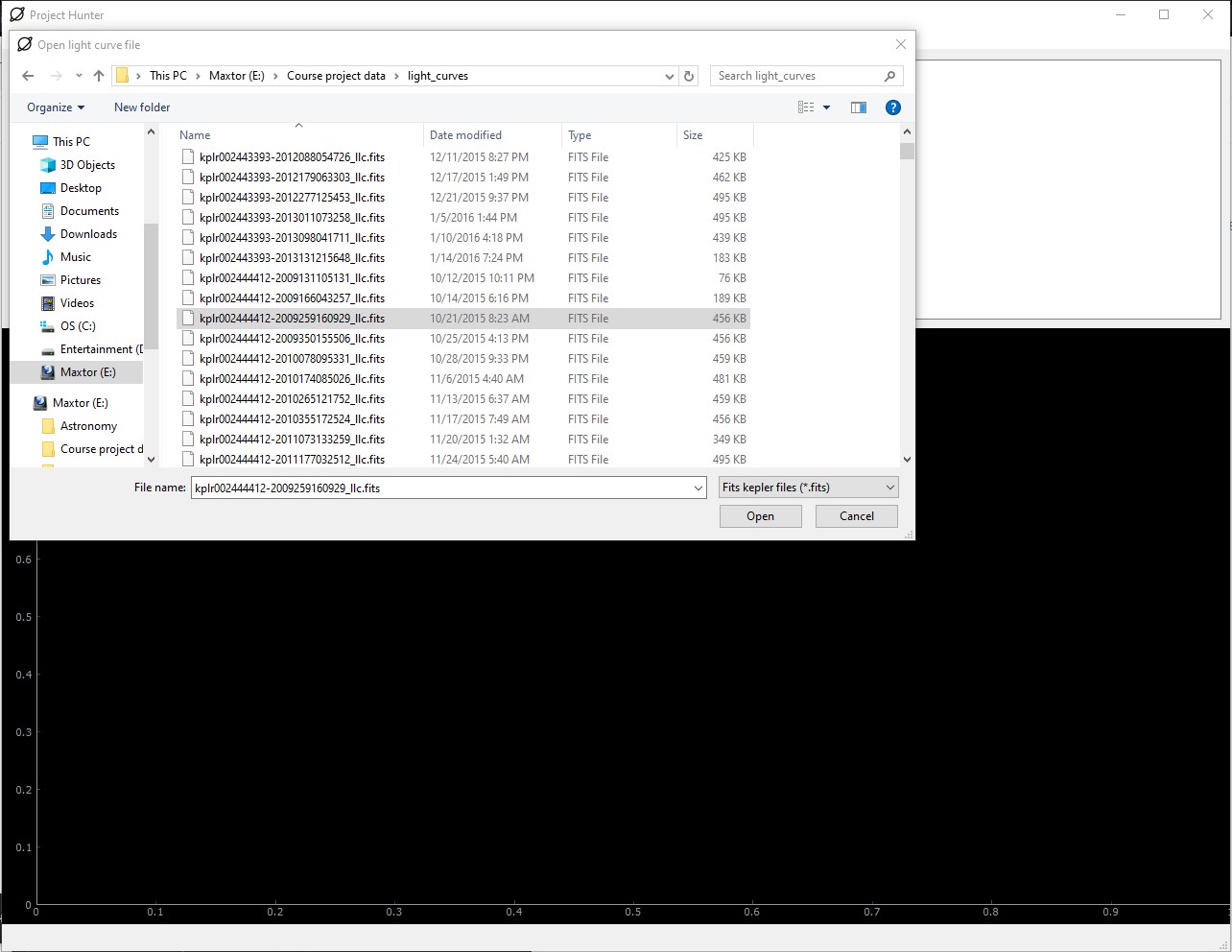


Рисунок 4.2. Открытие файла формата .fits

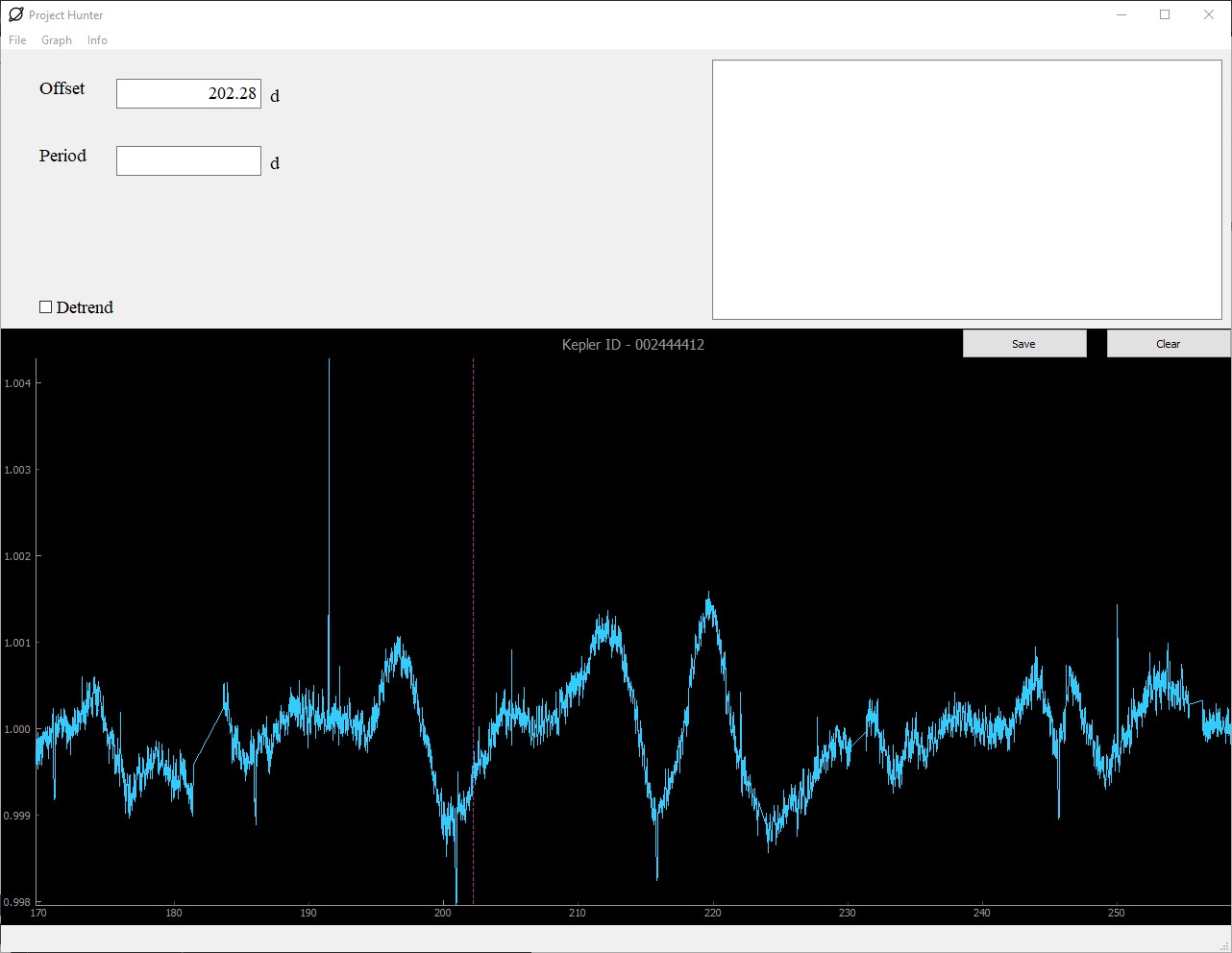


Рисунок 4.3. Построенный график световой кривой

Сценарий: исследователь открыл файл световой кривой и хочет отметить подозрительные на транзиты области за счёт интеракций с рабочей областью графика при помощи кнопок мышки. После исследователь очищает рабочую область.

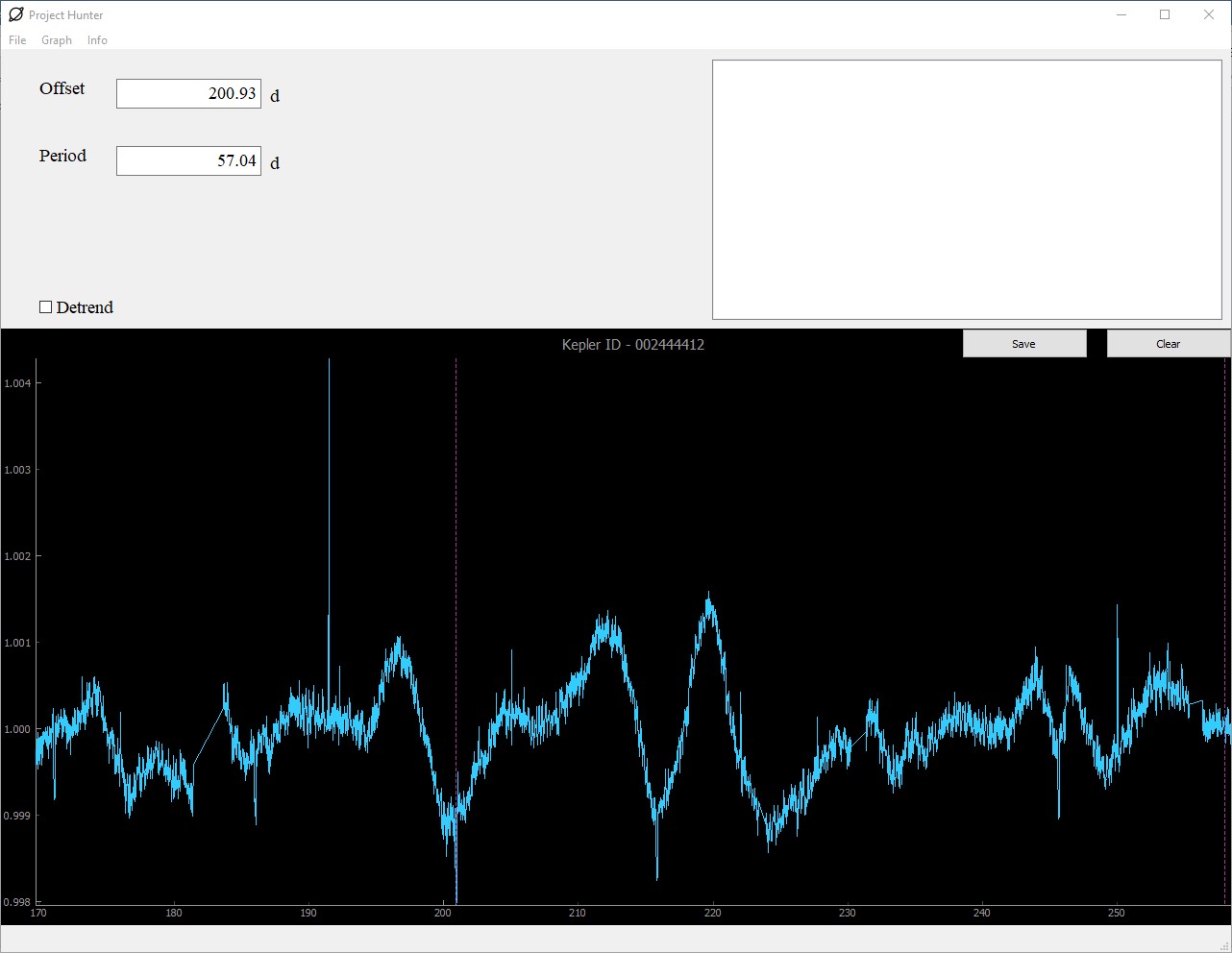


Рисунок 4.4. Нажатие левой кнопки мыши на область

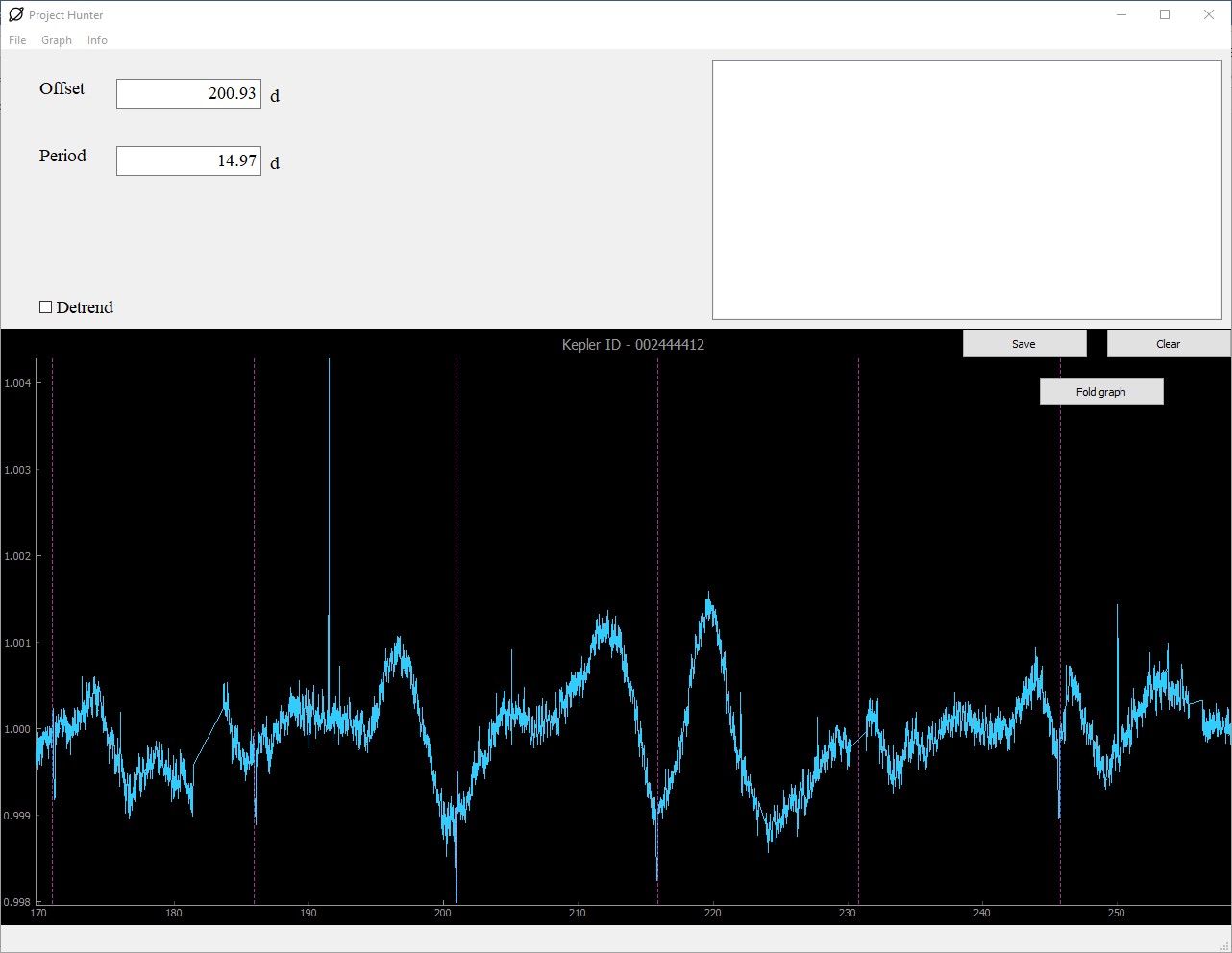


Рисунок 4.5. Повторное нажатие левой кнопки мыши на другую область

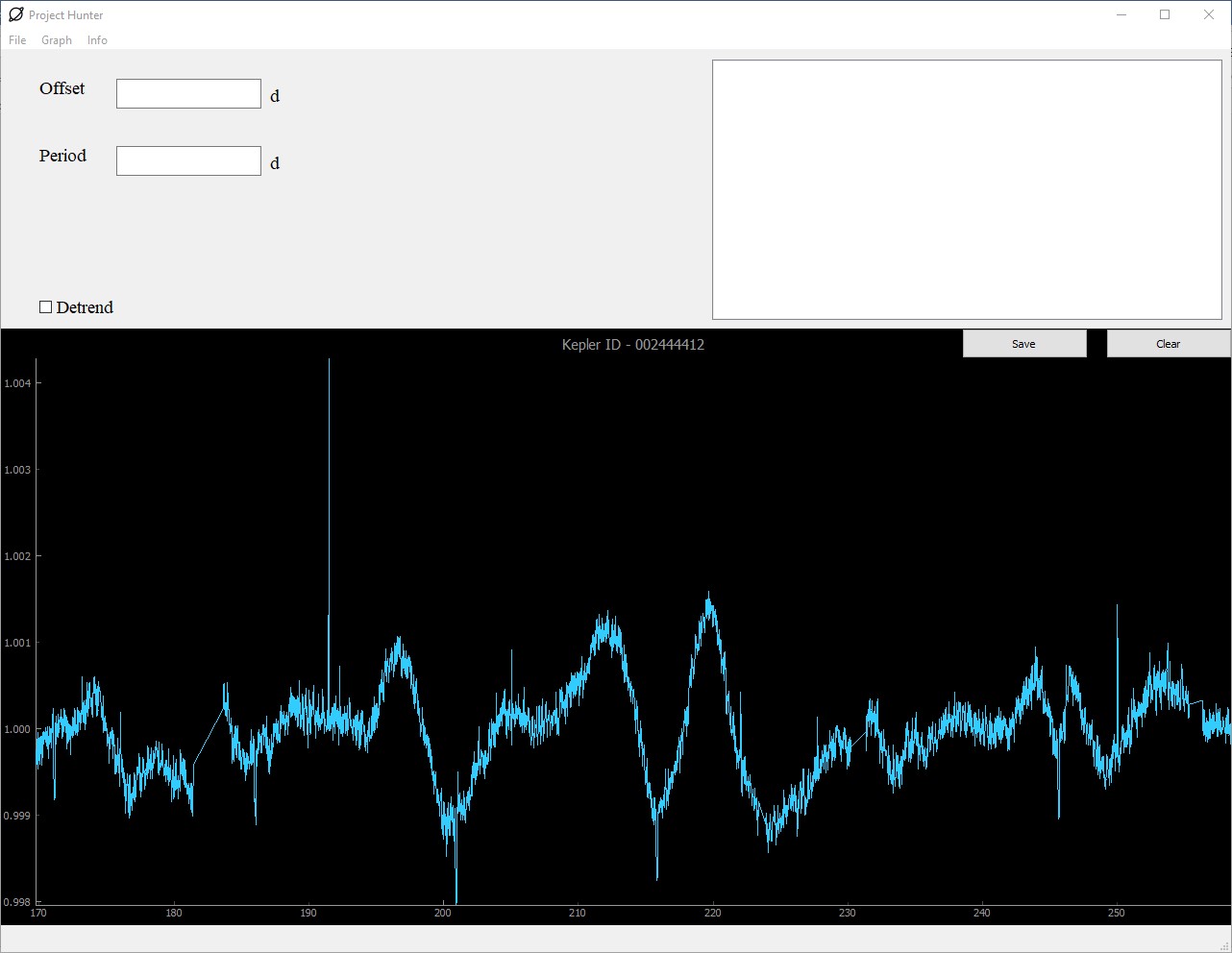


Рисунок 4.6. Чистка рабочей область при помощи кнопки “Clear”

## **3.2. Тестирование устранения долгопериодических колебаний звезды**

Сценарий: исследователь открыл файл и хочет устранить долгопериодические колебания звезды, не устраняя при этом подозрительные на транзит области.

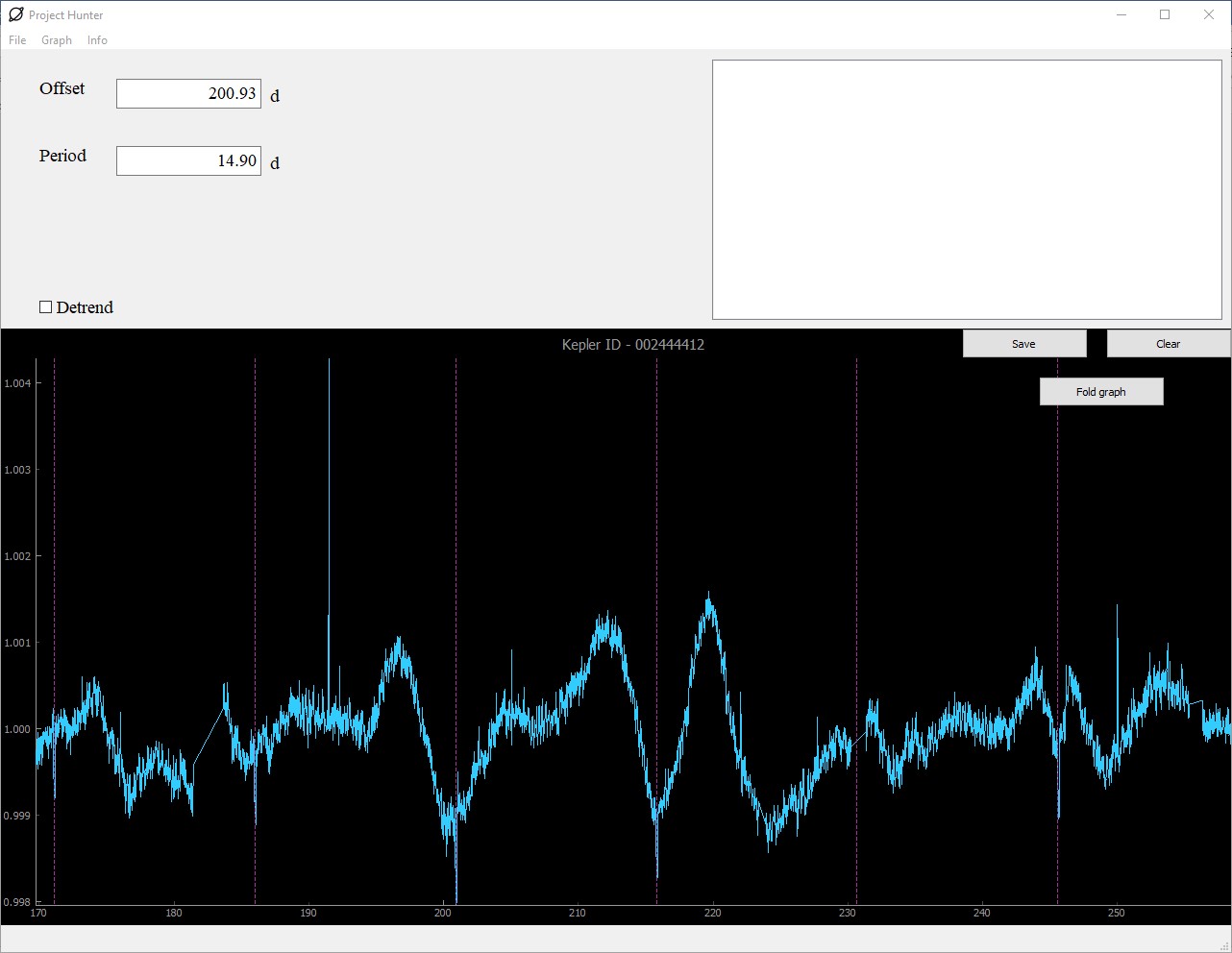


Рисунок 4.7. Изначальное состояние

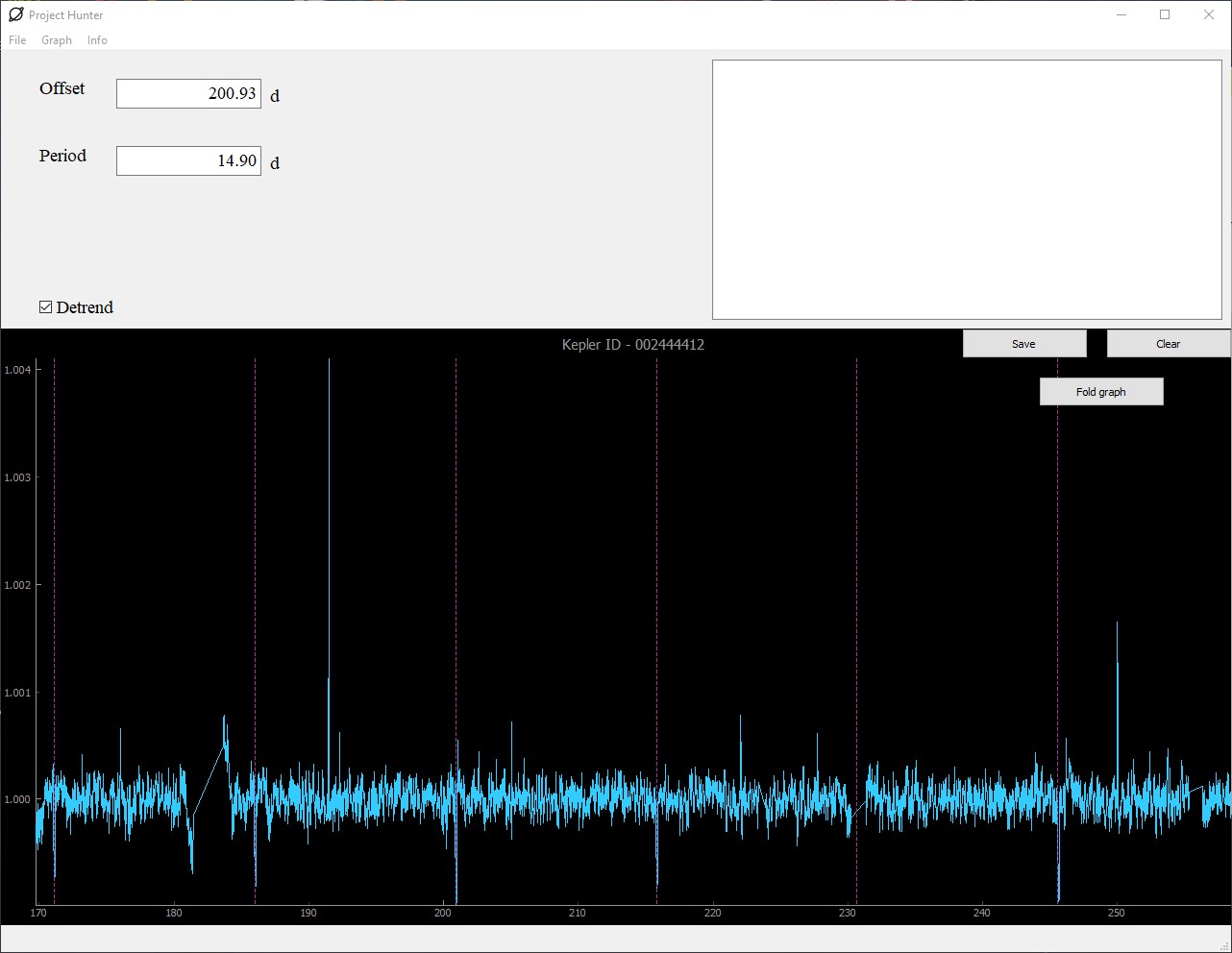


Рисунок 4.8. Состояние после устранения

## **3.3. Тестирование нарезки графика по выделенным областям**

Сценарий: исследователь выделил подозрительные области и хочет изучить данные области более подробно. Для этого он нажимает на кнопку “Fold graph”. Кроме этого после нарезки он хочет устранить долгопериодические колебания звезды для более чёткой картины.

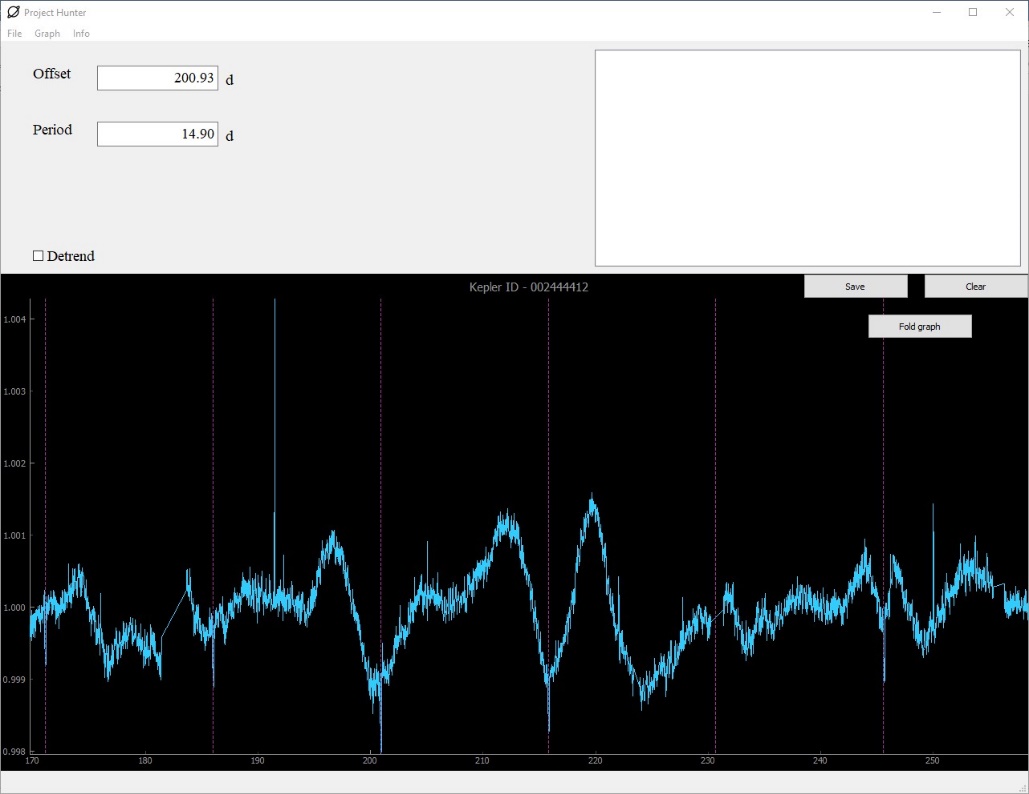


Рисунок 4.9. Изначальное состояние

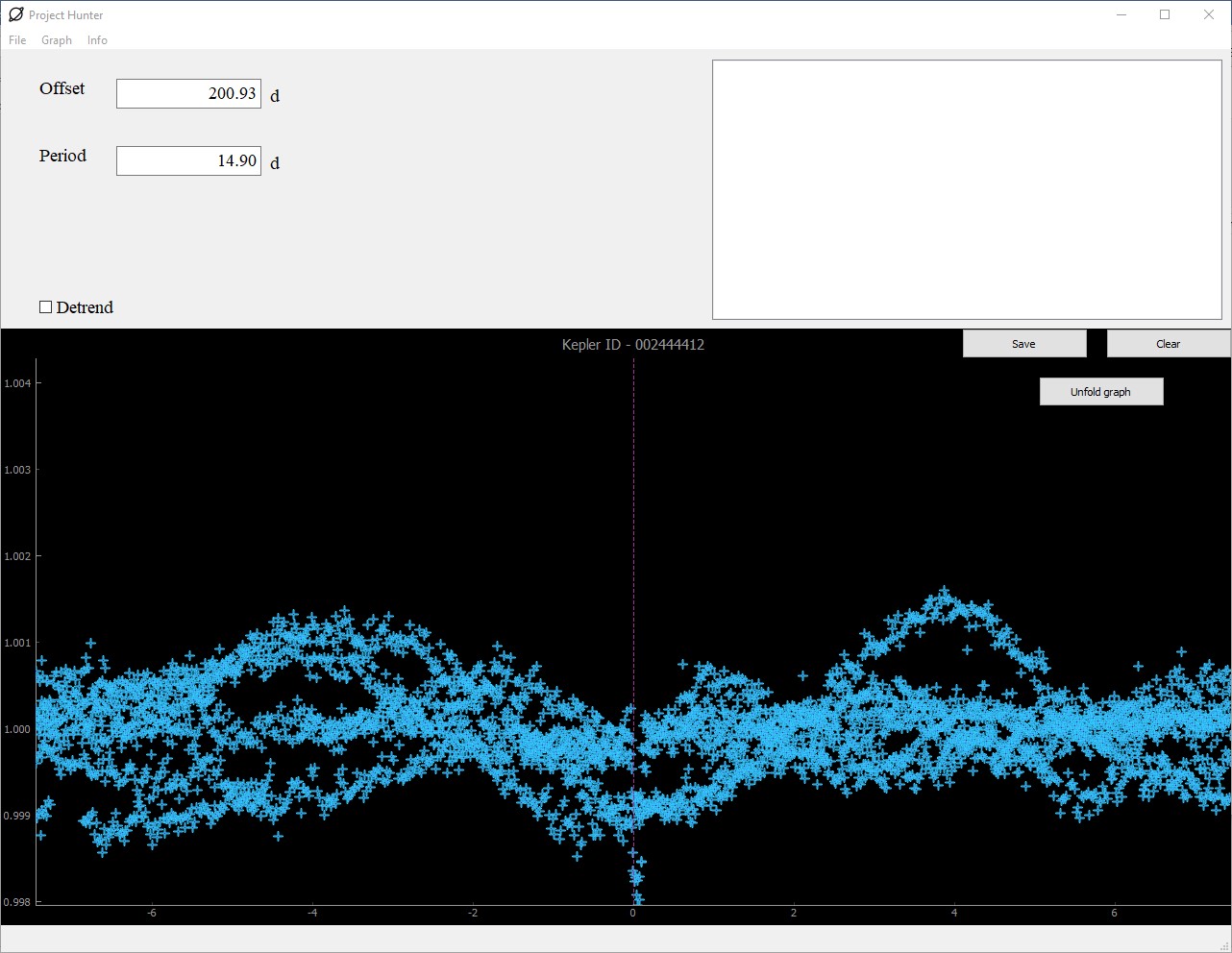


Рисунок 4.10. Нарезанный график

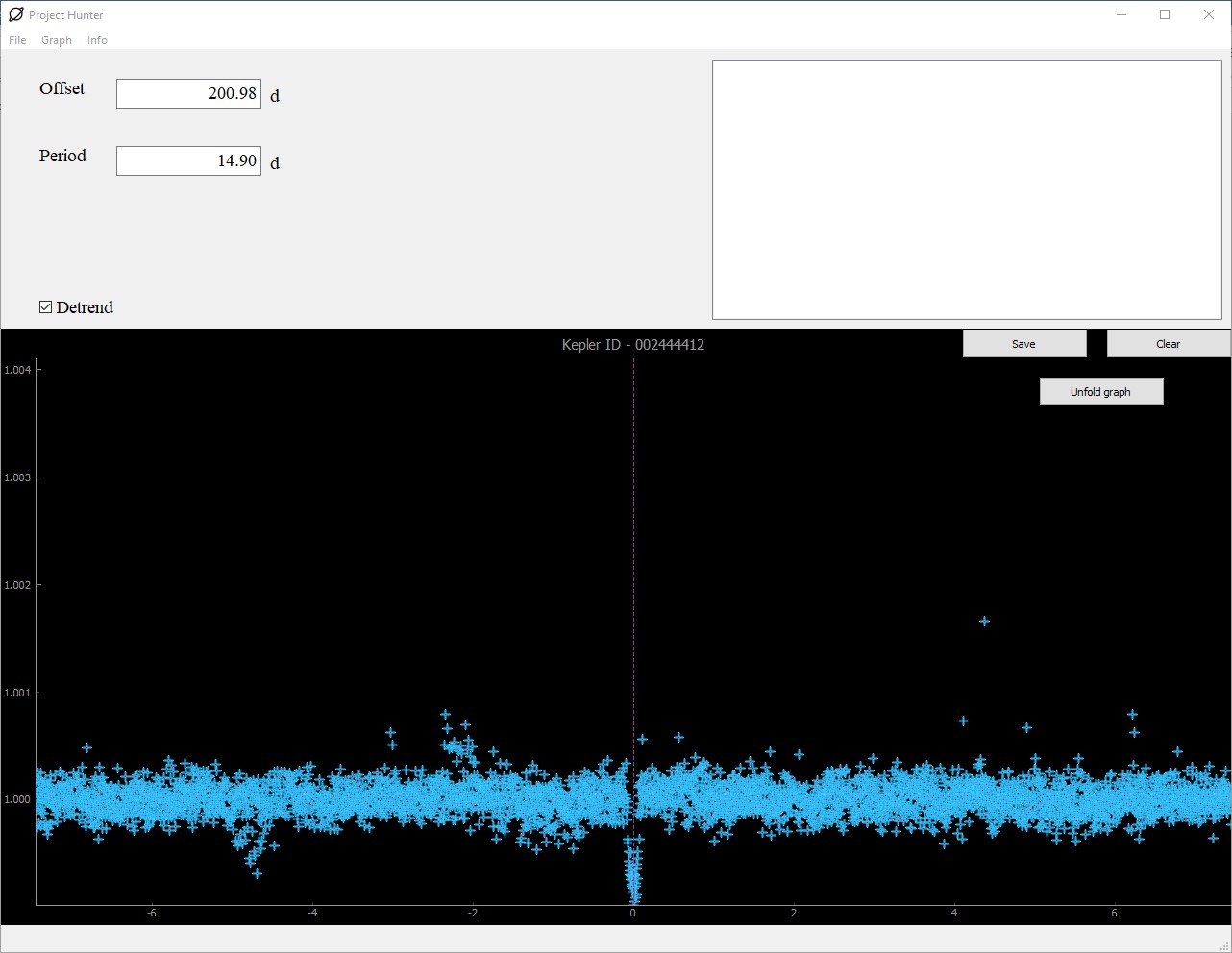


Рисунок 4.11. Нарезанный график с устранёнными долгопериодическими колебаниями звезды

## **3.4. Тестирование сохранения результатов исследований**

Сценарий: исследователь нашёл транзиты в световой кривой и теперь хочет сохранить данный результат. Он нажимает кнопку “Save” и открывает другой файл. Затем он снова открывает предыдущий файл и дважды кликает на результат исследования. Исследователь замечает, что допустил ошибку и удаляет данный результат.

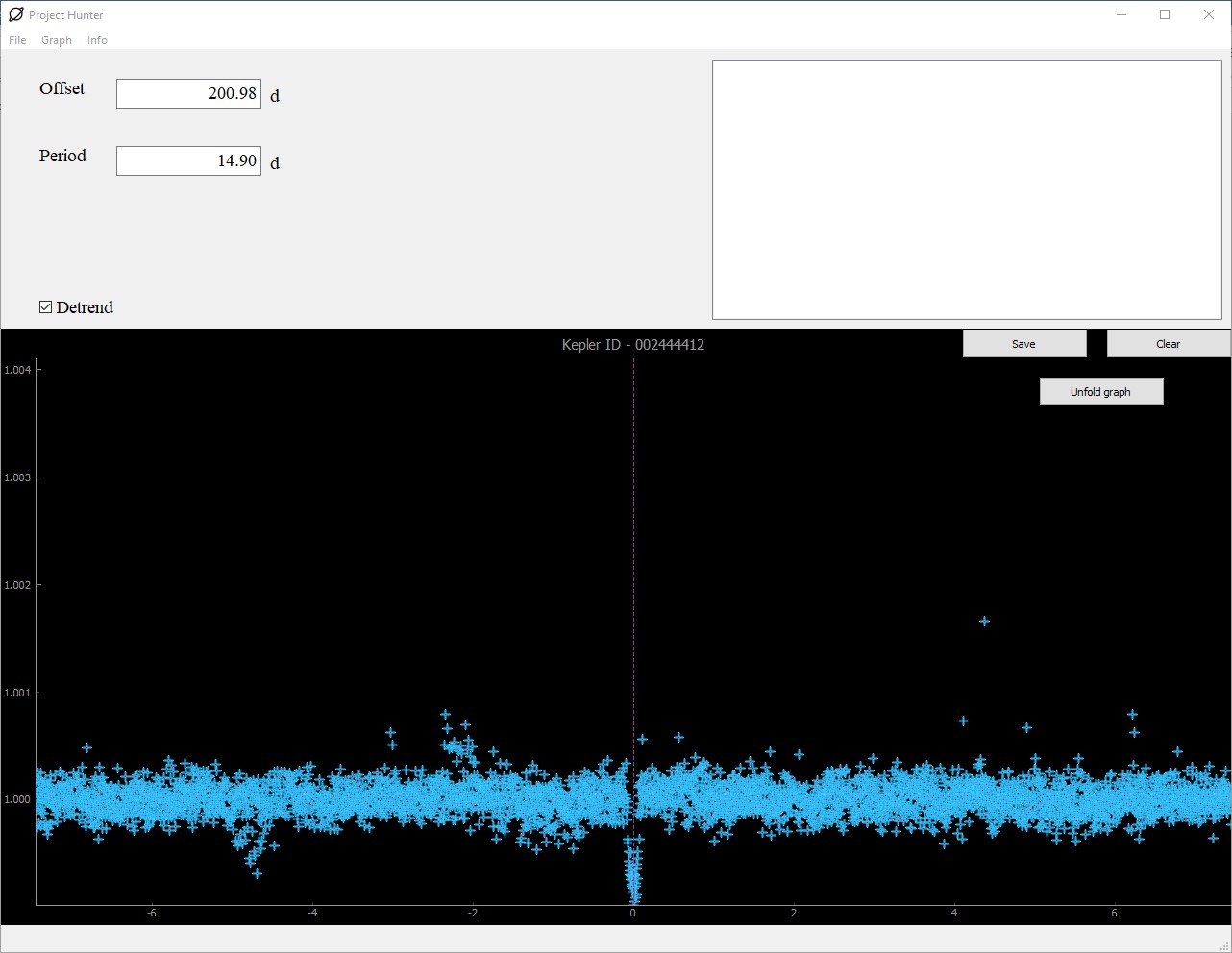


Рисунок 4.12. Исходное состояние

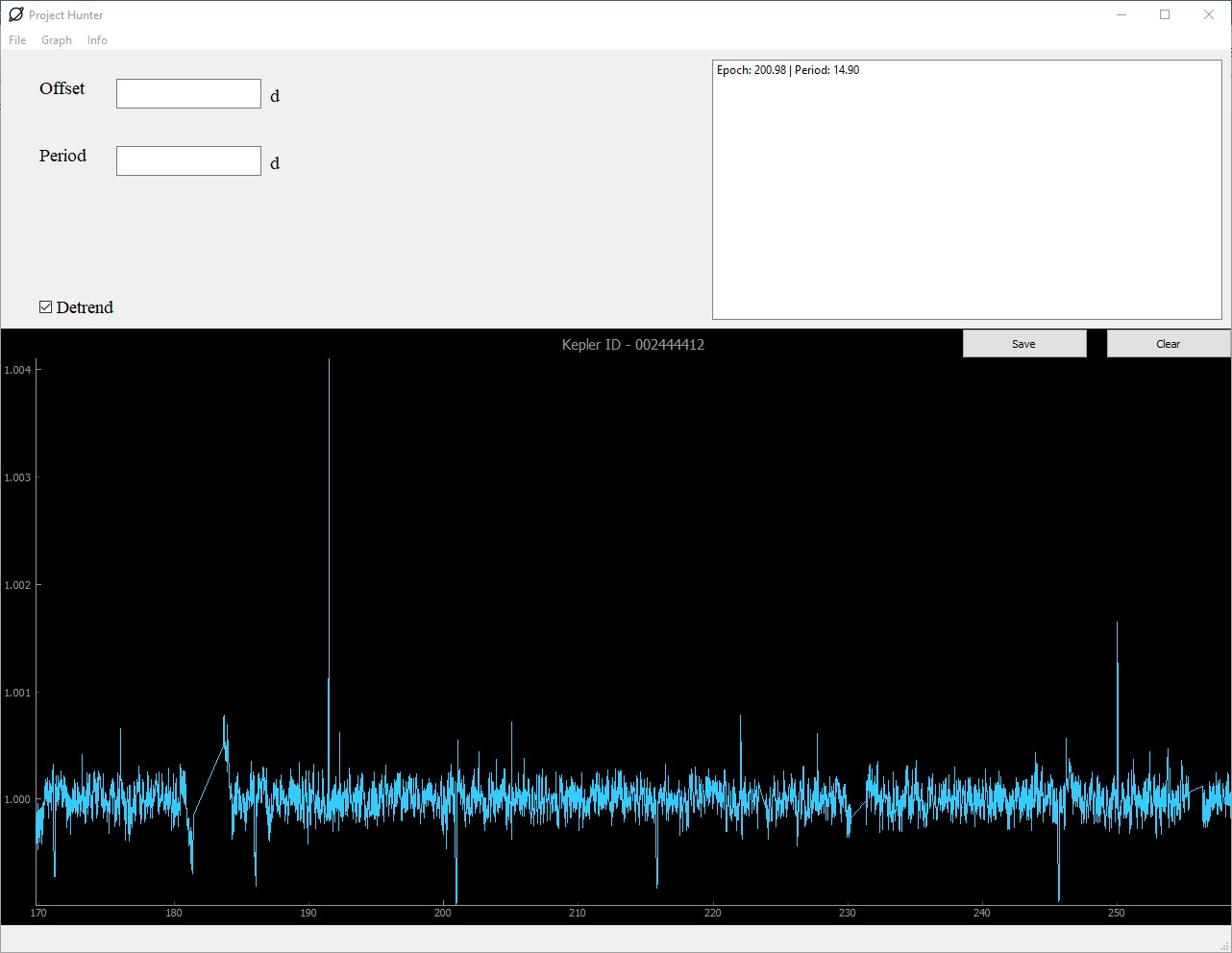


Рисунок 4.13. Сохранение результата

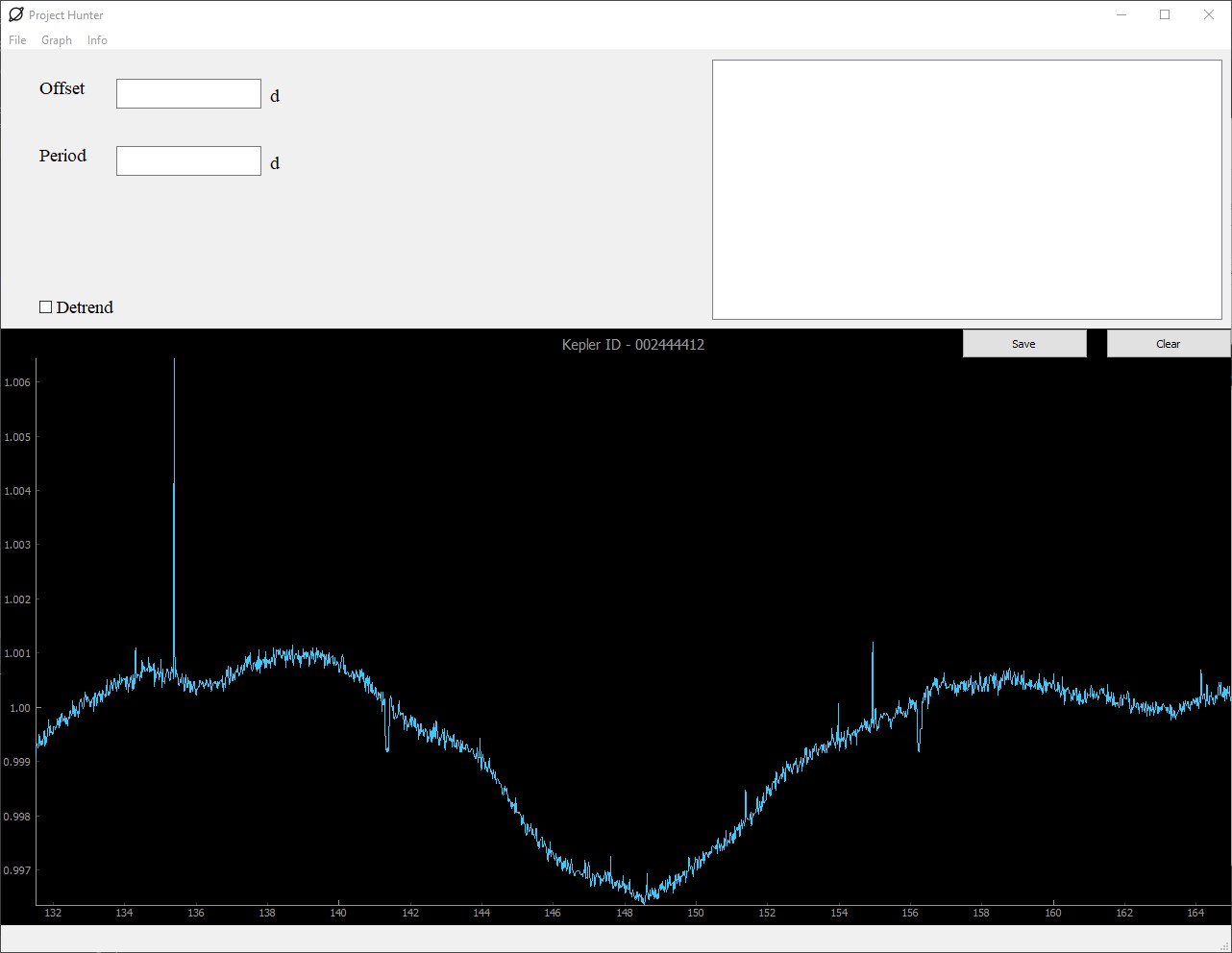


Рисунок 4.14. Открытие другого файла

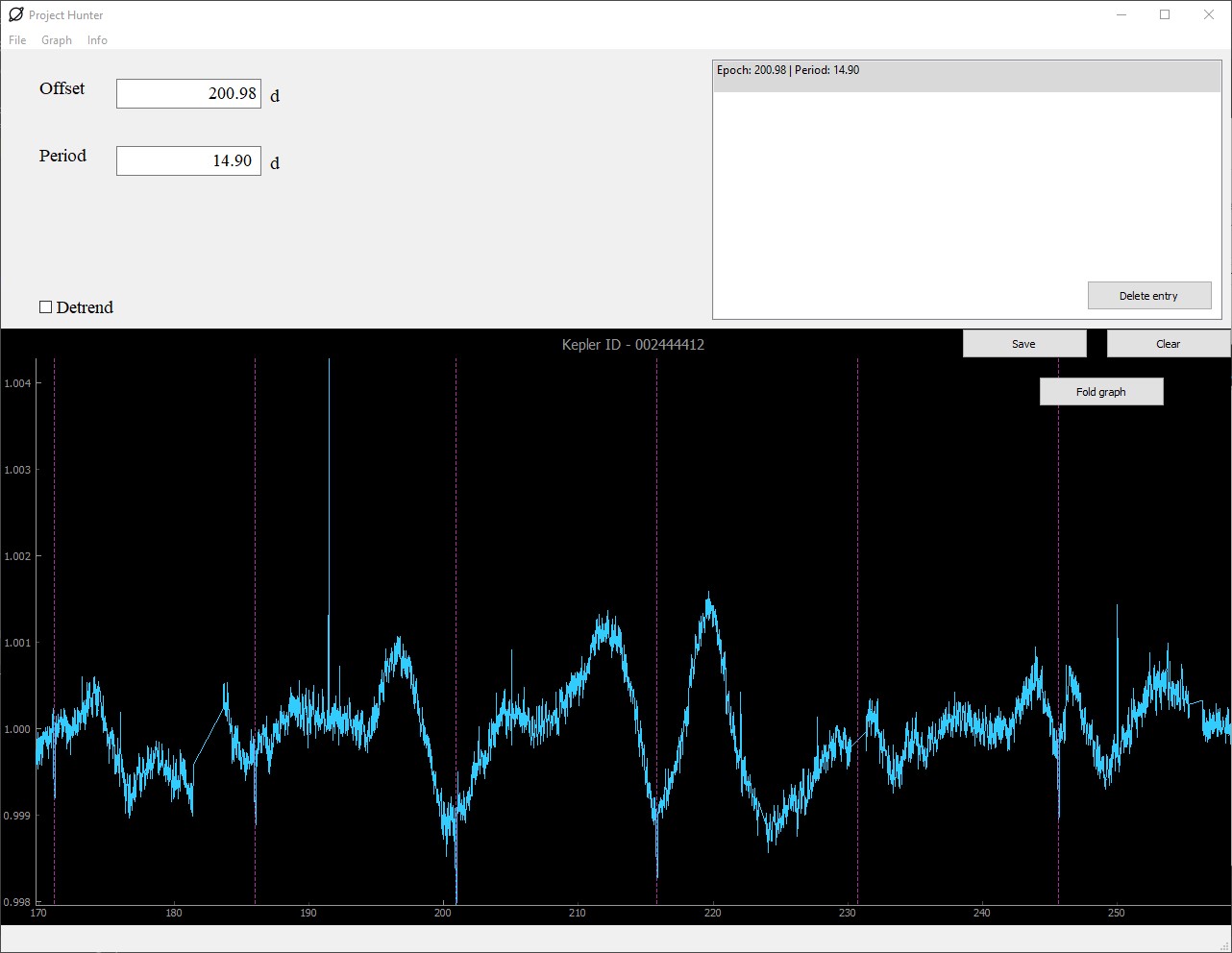


Рисунок 4.15. Открытие исходного файла и просмотр результата

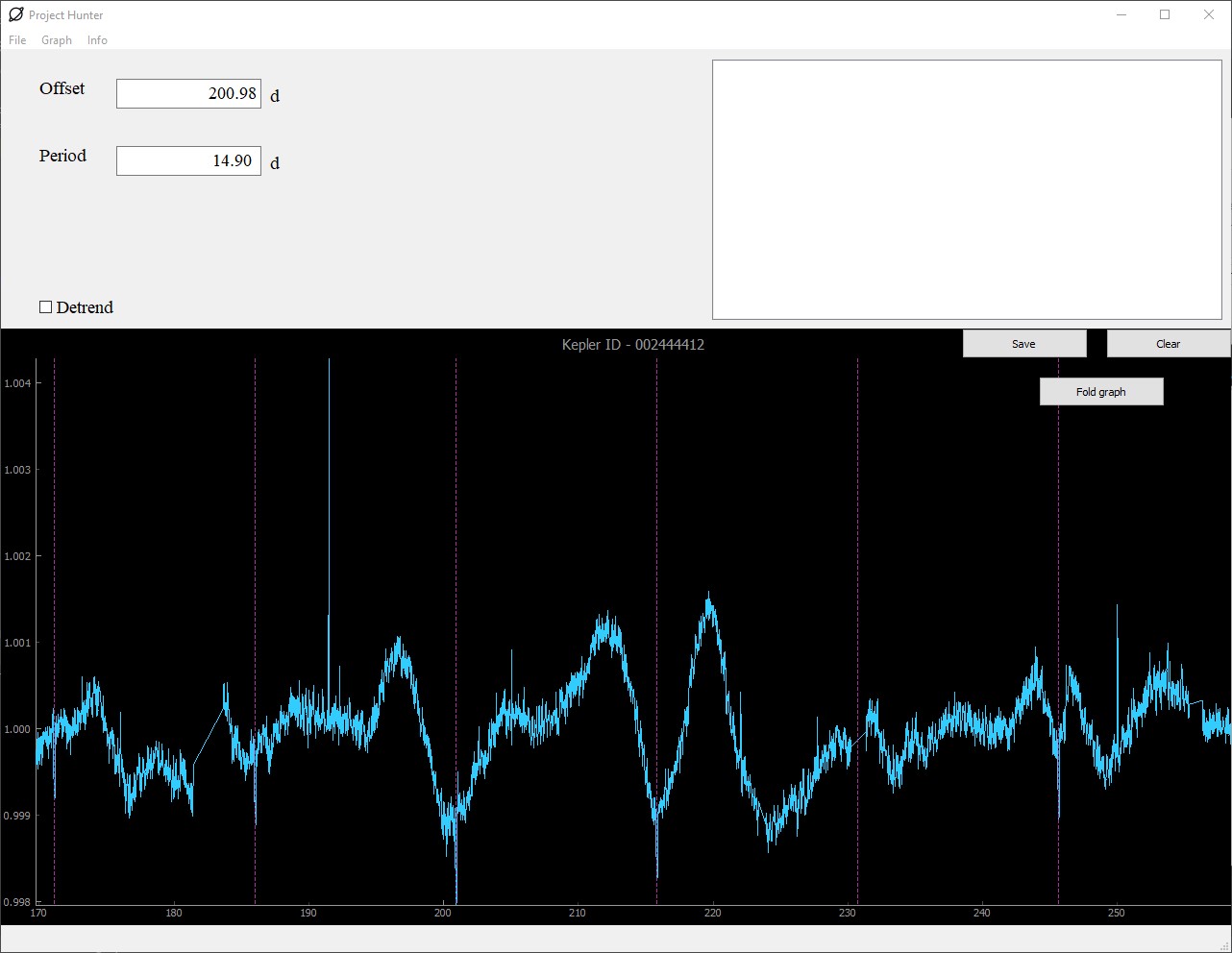


Рисунок 4.16. Удаление результата

**ВЫВОДЫ**

* + - 1. Было проведено тестирование основных методов приложения для работы с файлами и графиками.
      2. Было проведено тестирование устранения долгопериодических колебаний звезды из графика световой кривой.
      3. Было проведено тестирование нарезки графика по выделенным подозрительным областям.
      4. Было проведено тестирование сохранения результатов исследований как в оперативной, так и постоянной памяти компьютера.

# **ЗАКЛЮЧЕНИЕ**

Задача поиска экзопланет и жизни за пределами Солнечной системы на сегодняшний день является одной из самых актуальных в астрономии. Изучение физических условий на других планетах позволяет учёным лучше понять условия на Земле и спрогнозировать эволюцию этих условий с течением времени для понимания того, как это отразится на обществе и на потенциале выживания человечества в общем.

Кроме этого, поиск нового дома и эволюция человечества в межзвёздный вид, исследование новых горизонтов для дальнейшего углубления в законы физики данной Вселенной играют не меньший приоритет при поиске экзопланет. Хоть данные задачи в настоящее время не воспринимаются обществом как действительно важные из-за отсутствия прямых причин для беспокойства за состояние Земли и концентрировании современного общества на потребительстве, со временем они наверняка всплывут. Остаётся надеяться, что этот момент не наступит слишком поздно.

В данной работе были совершены следующие действия:

1. Пояснена предметная область исследования для данного курсового проекта. Описаны основные типы экзопланет, встречающихся в полученных данных.
2. Описаны методы открытия новых экзопланет. Выделены достоинства и недостатки основных методов открытия экзопланет. Проведено относительное сравнение методов и выявлен наилучший – транзитный метод.
3. Был проведён общий обзор возможностей и особенностей фреймворка Qt.
4. Была рассмотрена история развития фреймворка Qt и пояснена его нынешняя популярность.
5. Рассмотрены особенности применения фреймворка Qt на языке Python.
6. Рассмотрены две различные привязки для Qt для языке Python, проведено их сравнение и сделан выбор одной из привязок для разработки данного проекта.
7. Рассмотрены особенности выбранной привязки PyQt и её возможности.
8. Был проведён анализ проблем исследователей световых кривых и созданы пользовательские истории для решения этих проблем.
9. Был построен графический интерфейс на основании пользовательских историй, созданных в пункте 1.
10. Была спроектирована внутренняя архитектура приложения для реализации необходимого функционала.
11. Были реализованы основные методы класса MainWindow для решения большинства простейших проблем.
12. Было реализовано устранение долгопериодических колебаний звезды из графика световой кривой.
13. Было реализовано сложения графика по выделенным подозрительным областям.
14. Было реализовано сохранение результатов исследований как в оперативной, так и постоянной памяти компьютера.
15. Было проведено тестирование основных методов приложения для работы с файлами и графиками.
16. Было проведено тестирование устранения долгопериодических колебаний звезды из графика световой кривой.
17. Было проведено тестирование нарезки графика по выделенным подозрительным областям.
18. Было проведено тестирование сохранения результатов исследований как в оперативной, так и постоянной памяти компьютера.

# **СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

* 1. Уоссермен, Ф. Нейрокомпьютерная техника: Теория и практика = Neural Computing. Theory and Practice. / Уоссермен, Ф. — М.: Мир, 1992. — 240 с. — ISBN 5-03-002115-9.
  2. Exoplanets and Candidate Statistics [Electronic resource] / NASA Exoplanet Archive – Mode of access: http://exoplanetarchive.ipac.caltech.edu/docs/counts\_detail.html – Date of access: 01.05.2020
  3. К.А. Постнов, А.В. Засов. Курс общей астрофизики. – М.: Физический факультет МГУ, 2005. – 192 с. – ISBN 5-9900318-2-3.
  4. Bradley W. Carroll, Dale A. Ostlie. An Introduction to Modern Astrophysics. / Bradley W. Carroll, Dale A. Ostlie. – Harlow: Pearson, 2014. – 1470 стр. – ISBN 1-292-02293-0.
  5. Mikulski Archive for Space Telescopes [Electronic resource] – Mode of access: https://archive.stsci.edu – Date of access: 01.05.2020
  6. Q1 – Q17 DR24 TCE Table [Electronic resource] – Mode of access: https://exoplanetarchive.ipac.caltech.edu/cgi-bin/TblView/nph-tblView?app=ExoTbls&config=q1\_q17\_dr24\_tce – Date of access: 01.05.2020
  7. THE ROCHE LIMIT FOR CLOSE-ORBITING PLANETS: MINIMUM DENSITY, COMPOSITION CONSTRAINTS, AND APPLICATION TO THE 4.2 hr PLANET KOI 1843.03 [Electronic resource] – Mode of access: https://authors.library.caltech.edu/41045/2/2041-8205\_773\_1\_L15.pdf – Date of access: 01.05.2020
  8. Почему я люблю Qt и вы все тоже должны его любить [Electronic resource] – Mode of access: https://habr.com/ru/post/221523/ – Date of access: 01.05.2020
  9. Qt (software) [Electronic resource] – Mode of access: https://en.wikipedia.org/wiki/Qt\_(software) – Date of access: 01.05.2020
  10. Краткий обзор кроссплатформенного фреймворка Qt [Electronic resource] – Mode of access: https://nicknixer.ru/programmirovanie/kratkij-obzor-krossplatformennogo-frejmvorka-qt/ – Date of access: 01.05.2020
  11. Qt [Electronic resource] – Mode of access: https://www.qt.io/ – Date of access: 01.05.2020
  12. Qt Maintainers [Electronic resource] – Mode of access: https://wiki.qt.io/Maintainers – Date of access: 01.05.2020
  13. PyQt [Electronic resource] – Mode of access: https://ru.wikipedia.org/wiki/PyQt – Date of access: 01.05.2020
  14. What is PyQt? [Electronic resource] – Mode of access: https://riverbankcomputing.com/software/pyqt/intro – Date of access: 01.05.2020
  15. About PyQt [Electronic resource] – Mode of access: https://wiki.python.org/moin/PyQt – Date of access: 01.05.2020
  16. PyQt Whitepaper [Electronic resource] – Mode of access: https://www.riverbankcomputing.com/static/Docs/PyQt4-whitepaper/pyqt-whitepaper-us.pdf – Date of access: 01.05.2020
  17. PyQt vs Qt for Python (PySide2) [Electronic resource] – Mode of access: https://machinekoder.com/pyqt-vs-qt-for-python-pyside2-pyside/ – Date of access: 01.05.2020
  18. QtPy repository [Electronic resource] – Mode of access: https://github.com/spyder-ide/qtpy – Date of access: 01.05.2020
  19. PyQtGraph Scientific Graphics and GUI Library for Python [Electronic resource] – Mode of access: http://www.pyqtgraph.org/ – Date of access: 01.05.2020
  20. Moving average [Electronic resource] – Mode of access: https://en.wikipedia.org/wiki/Moving\_average – Date of access: 01.05.2020
  21. Detrending a light curve [Electronic resource] – Mode of access: https://oldtalk.planethunters.org/science/discussions/DPH100ht7b – Date of access: 01.05.2020
  22. Announcing the first SHA-1 collision [Electronic resource] – Mode of access: https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html – Date of access: 01.05.2020

**ПРИЛОЖЕНИЯ**

*Приложение А*

Файл WindowUI.py:

*# -\*- coding: utf-8 -\*-  
  
# Form implementation generated from reading ui file 'WindowUI.ui'  
#  
# Created by: PyQt5 UI code generator 5.13.0  
#  
# WARNING! All changes made in this file will be lost!***from** PyQt5 **import** QtCore, QtGui, QtWidgets  
  
  
**class** Ui\_MainWindow(object):  
 **def** setupUi(self, MainWindow):  
 MainWindow.setObjectName(**"MainWindow"**)  
 MainWindow.resize(1280, 960)  
 sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed, QtWidgets.QSizePolicy.Fixed)  
 sizePolicy.setHorizontalStretch(0)  
 sizePolicy.setVerticalStretch(0)  
 sizePolicy.setHeightForWidth(MainWindow.sizePolicy().hasHeightForWidth())  
 MainWindow.setSizePolicy(sizePolicy)  
 icon = QtGui.QIcon()  
 icon.addPixmap(QtGui.QPixmap(**"icon.png"**), QtGui.QIcon.Normal, QtGui.QIcon.Off)  
 MainWindow.setWindowIcon(icon)  
 self.centralwidget = QtWidgets.QWidget(MainWindow)  
 self.centralwidget.setObjectName(**"centralwidget"**)  
 self.graphViewer = PlotWidget(self.centralwidget)  
 self.graphViewer.setGeometry(QtCore.QRect(0, 290, 1281, 621))  
 self.graphViewer.setObjectName(**"graphViewer"**)  
 self.saveButton = QtWidgets.QPushButton(self.graphViewer)  
 self.saveButton.setGeometry(QtCore.QRect(1000, 0, 131, 31))  
 self.saveButton.setObjectName(**"saveButton"**)  
 self.clearButton = QtWidgets.QPushButton(self.graphViewer)  
 self.clearButton.setGeometry(QtCore.QRect(1150, 0, 131, 31))  
 self.clearButton.setObjectName(**"clearButton"**)  
 self.foldButton = QtWidgets.QPushButton(self.graphViewer)  
 self.foldButton.setEnabled(**True**)  
 self.foldButton.setGeometry(QtCore.QRect(1080, 50, 131, 31))  
 self.foldButton.setObjectName(**"foldButton"**)  
 self.label\_1 = QtWidgets.QLabel(self.centralwidget)  
 self.label\_1.setGeometry(QtCore.QRect(40, 30, 51, 20))  
 font = QtGui.QFont()  
 font.setFamily(**"Times New Roman"**)  
 font.setPointSize(14)  
 self.label\_1.setFont(font)  
 self.label\_1.setObjectName(**"label\_1"**)  
 self.label\_2 = QtWidgets.QLabel(self.centralwidget)  
 self.label\_2.setGeometry(QtCore.QRect(40, 100, 61, 21))  
 font = QtGui.QFont()  
 font.setFamily(**"Times New Roman"**)  
 font.setPointSize(14)  
 self.label\_2.setFont(font)  
 self.label\_2.setObjectName(**"label\_2"**)  
 self.detrendCheck = QtWidgets.QCheckBox(self.centralwidget)  
 self.detrendCheck.setGeometry(QtCore.QRect(40, 260, 81, 16))  
 font = QtGui.QFont()  
 font.setFamily(**"Times New Roman"**)  
 font.setPointSize(14)  
 self.detrendCheck.setFont(font)  
 self.detrendCheck.setObjectName(**"detrendCheck"**)  
 self.offsetText = QtWidgets.QLineEdit(self.centralwidget)  
 self.offsetText.setGeometry(QtCore.QRect(120, 30, 151, 31))  
 font = QtGui.QFont()  
 font.setFamily(**"Times New Roman"**)  
 font.setPointSize(14)  
 self.offsetText.setFont(font)  
 self.offsetText.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.AlignVCenter)  
 self.offsetText.setObjectName(**"offsetText"**)  
 self.periodText = QtWidgets.QLineEdit(self.centralwidget)  
 self.periodText.setGeometry(QtCore.QRect(120, 100, 151, 31))  
 font = QtGui.QFont()  
 font.setFamily(**"Times New Roman"**)  
 font.setPointSize(14)  
 self.periodText.setFont(font)  
 self.periodText.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.AlignVCenter)  
 self.periodText.setObjectName(**"periodText"**)  
 self.listOfTransits = QtWidgets.QListWidget(self.centralwidget)  
 self.listOfTransits.setGeometry(QtCore.QRect(740, 10, 531, 271))  
 self.listOfTransits.setObjectName(**"listOfTransits"**)  
 self.label = QtWidgets.QLabel(self.centralwidget)  
 self.label.setGeometry(QtCore.QRect(280, 40, 16, 16))  
 font = QtGui.QFont()  
 font.setFamily(**"Times New Roman"**)  
 font.setPointSize(14)  
 self.label.setFont(font)  
 self.label.setObjectName(**"label"**)  
 self.label\_3 = QtWidgets.QLabel(self.centralwidget)  
 self.label\_3.setGeometry(QtCore.QRect(280, 110, 16, 16))  
 font = QtGui.QFont()  
 font.setFamily(**"Times New Roman"**)  
 font.setPointSize(14)  
 self.label\_3.setFont(font)  
 self.label\_3.setObjectName(**"label\_3"**)  
 self.deleteButton = QtWidgets.QPushButton(self.centralwidget)  
 self.deleteButton.setGeometry(QtCore.QRect(1130, 240, 131, 31))  
 self.deleteButton.setObjectName(**"deleteButton"**)  
 MainWindow.setCentralWidget(self.centralwidget)  
 self.menubar = QtWidgets.QMenuBar(MainWindow)  
 self.menubar.setGeometry(QtCore.QRect(0, 0, 1280, 21))  
 self.menubar.setObjectName(**"menubar"**)  
 self.menuFile = QtWidgets.QMenu(self.menubar)  
 self.menuFile.setObjectName(**"menuFile"**)  
 self.menuGraph = QtWidgets.QMenu(self.menubar)  
 self.menuGraph.setObjectName(**"menuGraph"**)  
 self.menuInfo = QtWidgets.QMenu(self.menubar)  
 self.menuInfo.setObjectName(**"menuInfo"**)  
 MainWindow.setMenuBar(self.menubar)  
 self.statusbar = QtWidgets.QStatusBar(MainWindow)  
 self.statusbar.setObjectName(**"statusbar"**)  
 MainWindow.setStatusBar(self.statusbar)  
 self.actionOpen = QtWidgets.QAction(MainWindow)  
 self.actionOpen.setObjectName(**"actionOpen"**)  
 self.actionExit = QtWidgets.QAction(MainWindow)  
 self.actionExit.setObjectName(**"actionExit"**)  
 self.actionAbout = QtWidgets.QAction(MainWindow)  
 self.actionAbout.setObjectName(**"actionAbout"**)  
 self.actionDetrend = QtWidgets.QAction(MainWindow)  
 self.actionDetrend.setCheckable(**True**)  
 self.actionDetrend.setObjectName(**"actionDetrend"**)  
 self.menuFile.addAction(self.actionOpen)  
 self.menuFile.addSeparator()  
 self.menuFile.addAction(self.actionExit)  
 self.menuGraph.addAction(self.actionDetrend)  
 self.menuInfo.addAction(self.actionAbout)  
 self.menubar.addAction(self.menuFile.menuAction())  
 self.menubar.addAction(self.menuGraph.menuAction())  
 self.menubar.addAction(self.menuInfo.menuAction())  
  
 self.retranslateUi(MainWindow)  
 QtCore.QMetaObject.connectSlotsByName(MainWindow)  
  
 **def** retranslateUi(self, MainWindow):  
 \_translate = QtCore.QCoreApplication.translate  
 MainWindow.setWindowTitle(\_translate(**"MainWindow"**, **"Project Hunter"**))  
 self.saveButton.setText(\_translate(**"MainWindow"**, **"Save"**))  
 self.clearButton.setText(\_translate(**"MainWindow"**, **"Clear"**))  
 self.foldButton.setText(\_translate(**"MainWindow"**, **"Fold graph"**))  
 self.label\_1.setText(\_translate(**"MainWindow"**, **"Offset"**))  
 self.label\_2.setText(\_translate(**"MainWindow"**, **"Period"**))  
 self.detrendCheck.setText(\_translate(**"MainWindow"**, **"Detrend"**))  
 self.label.setText(\_translate(**"MainWindow"**, **"d"**))  
 self.label\_3.setText(\_translate(**"MainWindow"**, **"d"**))  
 self.deleteButton.setText(\_translate(**"MainWindow"**, **"Delete entry"**))  
 self.menuFile.setTitle(\_translate(**"MainWindow"**, **"File"**))  
 self.menuGraph.setTitle(\_translate(**"MainWindow"**, **"Graph"**))  
 self.menuInfo.setTitle(\_translate(**"MainWindow"**, **"Info"**))  
 self.actionOpen.setText(\_translate(**"MainWindow"**, **"Open"**))  
 self.actionExit.setText(\_translate(**"MainWindow"**, **"Exit"**))  
 self.actionAbout.setText(\_translate(**"MainWindow"**, **"About"**))  
 self.actionDetrend.setText(\_translate(**"MainWindow"**, **"Detrend"**))  
**from** pyqtgraph **import** PlotWidget  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 **import** sys  
 app = QtWidgets.QApplication(sys.argv)  
 MainWindow = QtWidgets.QMainWindow()  
 ui = Ui\_MainWindow()  
 ui.setupUi(MainWindow)  
 MainWindow.show()  
 sys.exit(app.exec\_())

* 1. *Приложение Б*

Файл Main.py:

**import** re  
**import** sys  
**import** numpy **as** np  
**import** pyqtgraph **as** pg  
**from** PyQt5.QtCore **import** QPointF  
**from** PyQt5.QtGui **import** QDoubleValidator  
**from** PyQt5.QtWidgets **import** QMainWindow, QApplication, QFileDialog, QListWidgetItem, QMessageBox  
**from** astropy.io **import** fits  
  
**from** WindowUI **import** Ui\_MainWindow  
**import** utils  
  
  
**class** MainWindow(QMainWindow, Ui\_MainWindow):  
  
 **def** \_\_init\_\_(self, \*args, \*\*kwargs):  
 super(MainWindow, self).\_\_init\_\_(\*args, \*\*kwargs)  
 self.setupUi(self)  
 self.maxTransitLines = 32  
 self.fileLoaded = **False** self.folded\_all\_time = []  
 self.detrended\_all\_flux = []  
 self.all\_time = []  
 self.all\_flux = []  
 self.kepID = **""** self.hashFile = **""** self.hashDir = **"cache\\"** transitLine = pg.InfiniteLine()  
 transitLine.setPen(utils.get\_transit\_pen())  
 self.firstTransitLine = transitLine  
 self.transitLines = []  
 self.presentLines = []  
 self.firstTransitSet = **False** self.firstLineAdded = **False** self.secondLinesSet = **False** self.isFolded = **False** self.init\_open()  
  
 **def** init\_open(self):  
  
 pen = utils.get\_transit\_pen()  
 **for** i **in** range(self.maxTransitLines):  
 transitLine = pg.InfiniteLine()  
 transitLine.setPen(pen)  
 self.transitLines.append(transitLine)  
 self.presentLines.append(**False**)  
  
 self.foldButton.setVisible(**False**)  
 self.foldButton.clicked.connect(self.click\_fold\_graph)  
  
 self.detrendCheck.setVisible(**False**)  
 self.detrendCheck.clicked.connect(self.detrend\_data)  
  
 self.deleteButton.setVisible(**False**)  
 self.deleteButton.clicked.connect(self.delete\_entry)  
  
 self.actionOpen.setShortcut(**"Ctrl+O"**)  
 self.actionOpen.setStatusTip(**"Open .fits file."**)  
 self.actionOpen.triggered.connect(self.open\_file)  
  
 self.actionDetrend.setEnabled(**False**)  
 self.actionDetrend.setShortcut(**"Ctrl+D"**)  
 self.actionDetrend.setStatusTip(**"Detrend graph"**)  
 self.actionDetrend.triggered.connect(self.menu\_detrend)  
  
 self.actionExit.setShortcut(**"Ctrl+Q"**)  
 self.actionExit.setStatusTip(**"Exit the application."**)  
 self.actionExit.triggered.connect(self.exit\_app)  
  
 self.actionAbout.setShortcut(**"Ctrl+I"**)  
 self.actionAbout.setStatusTip(**"About the application."**)  
 self.actionAbout.triggered.connect(self.show\_about)  
  
 self.offsetText.setValidator(QDoubleValidator())  
 self.periodText.setValidator(QDoubleValidator())  
 self.offsetText.textChanged.connect(self.text\_changed)  
 self.offsetText.setStatusTip(**"Offset of first transit in days."**)  
 self.periodText.textChanged.connect(self.text\_changed)  
 self.periodText.setStatusTip(**"Period of transits in days."**)  
  
 self.listOfTransits.itemClicked.connect(self.selection\_changed)  
 self.listOfTransits.itemDoubleClicked.connect(self.get\_saved\_transit)  
 self.listOfTransits.setStatusTip(**"Double click on transit entry to load it. "** +  
 **"Select entry and click Delete entry to remove."**)  
  
 self.clearButton.setVisible(**False**)  
 self.clearButton.clicked.connect(self.click\_clear\_graph)  
 self.clearButton.setStatusTip(**"Click to clear graph from transit lines."**)  
 self.saveButton.setVisible(**False**)  
 self.saveButton.clicked.connect(self.save\_transit\_entry)  
 self.saveButton.setStatusTip(**"Click to save transit entry."**)  
  
 self.graphViewer.sceneObj.sigMouseMoved.connect(self.show\_transit\_line)  
 self.graphViewer.sceneObj.sigMouseClicked.connect(self.set\_transit\_line)  
  
 **def** open\_file(self):  
 **if** self.hashFile != **""**:  
 self.save\_transits\_to\_file()  
 self.clear\_graph(rebuild=**False**)  
 fileName, \_ = QFileDialog.getOpenFileName(self, **"Open light curve file"**,  
 **"c:\\"**, **"Fits kepler files (\*.fits)"**)  
 **if** fileName:  
 **with** fits.open(fileName) **as** hdu\_list:  
 light\_curve = hdu\_list[**"LIGHTCURVE"**].data  
  
 match = re.search(**"kplr[0-9]{9}"**, fileName)  
 self.kepID = **""  
 if** match:  
 self.kepID = match[0][4:]  
  
 self.all\_time = light\_curve.TIME  
 self.all\_flux = light\_curve.PDCSAP\_FLUX  
  
 flux\_and\_time\_finite = np.logical\_and(np.isfinite(self.all\_flux), np.isfinite(self.all\_time))  
 self.all\_time = self.all\_time[flux\_and\_time\_finite]  
 self.all\_flux = self.all\_flux[flux\_and\_time\_finite]  
  
 self.all\_flux /= np.median(self.all\_flux)  
  
 self.hashFile = utils.hash\_file(fileName)  
 **try**:  
 **with** open(self.hashDir + self.hashFile, **"r"**) **as** transitFile:  
 **for** line **in** transitFile.readlines():  
 item = QListWidgetItem()  
 item.setText(line)  
 self.listOfTransits.addItem(item)  
 **except** OSError:  
 **pass** self.rebuild\_plot()  
 self.fileLoaded = **True** self.saveButton.setVisible(**True**)  
 self.clearButton.setVisible(**True**)  
 self.detrendCheck.setVisible(**True**)  
 self.actionDetrend.setEnabled(**True**)  
  
 **def** text\_changed(self):  
 **if not** self.isFolded:  
 **if** self.offsetText.text().strip(**" "**) != **""**:  
 xpos = float(self.offsetText.text())  
 p = QPointF(xpos, 1.0)  
 self.show\_first\_line(p)  
 **if** self.periodText.text().strip(**" "**) != **""**:  
 period = float(self.periodText.text())  
 p2 = QPointF(xpos + period, 1.0)  
 self.show\_secondary\_lines(p2)  
 **if** self.isFolded:  
 **if** self.offsetText.text().strip(**" "**) != **"" and** self.periodText.text().strip(**" "**) != **""**:  
 self.recalculate\_fold\_graph(float(self.offsetText.text()), float(self.periodText.text()))  
 **else**:  
 self.clear\_graph(rebuild=**True**)  
 self.text\_changed()  
  
 **def** get\_saved\_transit(self, item):  
 self.clear\_graph(rebuild=**True**)  
 epoch, period = item.text().split(**"|"**)  
 epoch = epoch[7:].strip(**" "**)  
 period = period[9:].strip(**" "**)  
 self.offsetText.setText(epoch)  
 **if** period != **"-"**:  
 self.periodText.setText(period)  
 self.secondLinesSet = **True** self.firstTransitSet = **True** self.isFolded = **False** self.foldButton.setVisible(**True**)  
  
 **def** save\_transit\_entry(self):  
 item = QListWidgetItem()  
 offset = self.offsetText.text().strip(**" "**)  
 period = self.periodText.text().strip(**" "**)  
 **if** offset == **"" or not** self.firstTransitSet:  
 QMessageBox.warning(self, **"Save transit"**, **"Offset of the first transit should be selected."**)  
 **return  
 if** period == **"" or not** self.secondLinesSet:  
 period = **"-"** item.setText(**"Epoch: {0} | Period: {1}\n"**.format(offset, period))  
 self.listOfTransits.addItem(item)  
 self.clear\_graph(rebuild=**True**)  
  
 **def** selection\_changed(self):  
 **if** len(self.listOfTransits.selectedItems()) == 0:  
 self.deleteButton.setVisible(**False**)  
 **return** self.deleteButton.setVisible(**True**)  
  
 **def** delete\_entry(self):  
 self.listOfTransits.takeItem(self.listOfTransits.currentRow())  
 self.deleteButton.setVisible(**False**)  
  
 **def** save\_transits\_to\_file(self):  
 **if** self.listOfTransits.count() == 0:  
 **return  
 with** open(self.hashDir + self.hashFile, **"w"**) **as** transitFile:  
 **for** i **in** range(self.listOfTransits.count()):  
 transitFile.write(self.listOfTransits.item(i).text())  
 self.listOfTransits.clear()  
 self.deleteButton.setVisible(**False**)  
  
 **def** click\_clear\_graph(self):  
 self.clear\_graph(rebuild=**True**)  
  
 **def** clear\_graph(self, rebuild=**False**):  
 self.graphViewer.getPlotItem().clear()  
 self.isFolded = **False** self.offsetText.setText(**""**)  
 self.periodText.setText(**""**)  
 **for** index **in** range(self.maxTransitLines):  
 self.presentLines[index] = **False** self.firstTransitSet = **False** self.firstLineAdded = **False** self.secondLinesSet = **False** self.foldButton.setVisible(**False**)  
 **if** rebuild:  
 self.rebuild\_plot()  
  
 **def** rebuild\_graph\_with\_transits(self):  
 self.graphViewer.getPlotItem().clear()  
 self.rebuild\_plot()  
 self.graphViewer.addItem(self.firstTransitLine)  
 **for** index, item **in** enumerate(self.transitLines):  
 **if** self.presentLines[index]:  
 self.graphViewer.addItem(item)  
  
 **def** rebuild\_plot(self):  
 x\_data = self.all\_time  
 y\_data = self.all\_flux  
 **if** self.kepID != **""**:  
 self.graphViewer.getPlotItem().setTitle(**"Kepler ID - {0}"**.format(self.kepID))  
 **if** self.detrendCheck.isChecked():  
 y\_data = self.detrended\_all\_flux  
 **if** self.isFolded:  
 x\_data = self.folded\_all\_time  
 self.graphViewer.getPlotItem().plot(x\_data, y\_data, pen=**None**, symbol=**'+'**, symbolPen=utils.get\_fold\_qpen())  
 **else**:  
 self.graphViewer.getPlotItem().plot(x\_data, y\_data, pen=utils.get\_qpen())  
 self.graphViewer.getPlotItem().vb.setLimits(yMin=np.min(y\_data), yMax=np.max(y\_data),  
 xMin=np.min(x\_data), xMax=np.max(x\_data))  
  
 **def** exit\_app(self):  
 choice = QMessageBox.question(self, **'Exit'**,  
 **"Exit the application?"**,  
 QMessageBox.Yes | QMessageBox.No)  
 **if** choice == QMessageBox.Yes:  
 sys.exit()  
 **else**:  
 **pass  
  
 def** show\_about(self):  
 QMessageBox.information(self, **'About'**, **"Application for analyzing light curves provided by Kepler telescope.\n"** +  
 **"Wrote on PyQt5 and pyqtgraph.\n"** +  
 **"Mikhailau Anton. 2020."**, )  
  
 **def** show\_transit\_line(self):  
 **if not** self.fileLoaded:  
 **return  
 if** self.isFolded:  
 **return** pos = self.graphViewer.lastMousePos  
 p = self.graphViewer.getPlotItem().vb.mapSceneToView(pos)  
 **if not** self.firstTransitSet:  
 self.offsetText.setText(**"%.2f"** % p.x())  
 **elif not** self.secondLinesSet:  
 offset = self.firstTransitLine.pos().x()  
 period = np.abs(offset - p.x())  
 self.periodText.setText(**"%.2f"** % period)  
  
 **def** show\_first\_line(self, p):  
 **if not** self.firstLineAdded:  
 self.graphViewer.addItem(self.firstTransitLine)  
 self.firstLineAdded = **True** self.firstTransitLine.setPos(p)  
  
 **def** show\_secondary\_lines(self, p):  
 offset = self.firstTransitLine.pos().x()  
 period = np.abs(offset - p.x())  
 **if** period == 0:  
 **return** left\_transit = offset - period  
 right\_transit = offset + period  
 last\_point = self.all\_time[len(self.all\_time) - 1]  
 **for** index, item **in** enumerate(self.transitLines):  
 **if** index % 2 == 0:  
 **if** left\_transit < self.all\_time[0]:  
 self.graphViewer.removeItem(item)  
 self.presentLines[index] = **False  
 continue** item.setPos(left\_transit)  
 left\_transit -= period  
 **if not** self.presentLines[index]:  
 self.presentLines[index] = **True** self.graphViewer.addItem(item)  
 **else**:  
 **if** right\_transit > last\_point:  
 self.graphViewer.removeItem(item)  
 self.presentLines[index] = **False  
 continue** item.setPos(right\_transit)  
 right\_transit += period  
 **if not** self.presentLines[index]:  
 self.presentLines[index] = **True** self.graphViewer.addItem(item)  
  
 **def** set\_transit\_line(self):  
 **if** self.isFolded:  
 **return  
 if not** self.firstTransitSet:  
 self.firstTransitSet = **True  
 else**:  
 self.secondLinesSet = **True** self.foldButton.setVisible(**True**)  
  
 **def** click\_fold\_graph(self):  
 xpos = float(self.offsetText.text())  
 period = float(self.periodText.text())  
 self.fold\_graph(xpos, period)  
  
 **def** fold\_graph(self, xpos, period):  
 **if** self.isFolded:  
 self.foldButton.setText(**"Fold graph"**)  
 self.isFolded = **False** self.firstTransitLine.setPos(xpos)  
 self.rebuild\_graph\_with\_transits()  
 **else**:  
 self.foldButton.setText(**"Unfold graph"**)  
 self.isFolded = **True** self.recalculate\_fold\_graph(xpos, period)  
  
 **def** recalculate\_fold\_graph(self, xpos, period):  
 self.folded\_all\_time = []  
 **for** point **in** self.all\_time:  
 self.folded\_all\_time.append(point - (xpos + np.round((point - xpos) / period) \* period))  
 self.graphViewer.getPlotItem().clear()  
 self.rebuild\_plot()  
 self.firstTransitLine.setPos(0.0)  
 self.graphViewer.addItem(self.firstTransitLine)  
  
 **def** menu\_detrend(self):  
 **if** self.actionDetrend.isChecked():  
 self.detrendCheck.setChecked(**True**)  
 self.detrend\_data()  
 **else**:  
 self.detrendCheck.setChecked(**False**)  
 self.detrend\_data()  
  
 **def** detrend\_data(self):  
 **if** self.detrendCheck.isChecked():  
 self.actionDetrend.setChecked(**True**)  
 dataSize = len(self.all\_time)  
 windowSize = int(np.sqrt(dataSize))  
 **if** windowSize % 2 == 0:  
 windowSize -= 1  
 self.detrended\_all\_flux = []  
 self.detrended\_all\_flux.extend(self.all\_flux[:int(windowSize / 2)])  
 window = []  
 window.extend(self.all\_flux[:windowSize])  
 **for** i **in** range(dataSize - windowSize + 1):  
 movingAverage = np.average(window)  
 self.detrended\_all\_flux.append(self.all\_flux[int(windowSize / 2) + i] - movingAverage + 1.0)  
 **if** i != dataSize - windowSize:  
 window = window[1:]  
 window.append(self.all\_flux[windowSize + i])  
 self.detrended\_all\_flux.extend(self.all\_flux[-int(windowSize / 2):])  
 **else**:  
 self.actionDetrend.setChecked(**False**)  
 self.rebuild\_graph\_with\_transits()  
  
  
**def** main():  
 app = QApplication(sys.argv)  
 GUI = MainWindow()  
 app.aboutToQuit.connect(GUI.save\_transits\_to\_file)  
 GUI.show()  
 sys.exit(app.exec\_())  
  
  
main()

* 1. *Приложение В*

Файл utils.py:

**import** hashlib  
  
**import** numpy **as** np  
**from** PyQt5 **import** QtCore  
**from** PyQt5.QtGui **import** QPen, QColor  
  
  
**def** reject\_outliers(data, m=3):  
 mean = np.mean(data)  
 sigma = np.std(data)  
 new\_data = []  
 rejected = []  
 **for** it **in** range(len(data)):  
 **if** (data[it] - mean) >= m \* sigma:  
 rejected.append(it)  
 **else**:  
 new\_data.append(data[it])  
 **return** new\_data, rejected  
  
  
**def** get\_qpen():  
 pen = QPen()  
 pen.setWidth(0.5)  
 pen.setStyle(QtCore.Qt.SolidLine)  
 pen.setColor(QColor(51, 204, 255))  
 pen.setCapStyle(QtCore.Qt.SquareCap)  
 pen.setJoinStyle(QtCore.Qt.RoundJoin)  
 **return** pen  
  
  
**def** get\_fold\_qpen():  
 pen = QPen()  
 pen.setWidth(0.1)  
 pen.setStyle(QtCore.Qt.SolidLine)  
 pen.setColor(QColor(51, 204, 255))  
 pen.setCapStyle(QtCore.Qt.SquareCap)  
 pen.setJoinStyle(QtCore.Qt.RoundJoin)  
 **return** pen  
  
  
**def** get\_transit\_pen():  
 pen = QPen()  
 pen.setWidth(0.3)  
 pen.setStyle(QtCore.Qt.DashLine)  
 pen.setColor(QColor(168, 50, 153))  
 **return** pen  
  
  
**def** hash\_file(filename):  
 h = hashlib.sha1()  
 **with** open(filename, **'rb'**) **as** file:  
 chunk = 0  
 **while** chunk != **b''**:  
 chunk = file.read(1024)  
 h.update(chunk)  
 **return** h.hexdigest()