ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ОБРАБОТКИ И ОТОБРАЖЕНИЯ ДАННЫХ ПРОФИЛОМЕТРА D-PACE WIRE SCANNER

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В новосибирском Институте ядерной физики осуществляется разработка ускорительного источника нейтронов, предназначенного для проведения бор-нейтронозахватной терапии (БНЗТ) злокачественных опухолей [1]. Одной из важных задач является диагностика пучка и автоматическая обработка данных.

Источник нейтронов представляет собой установку, состоящую из нескольких частей. Из ионного источника вытягиваются отрицательные ионы водорода с энергией 20 кэВ, затем пучок направляется в ускоритель-тандем, где ускоряется до потенциала 1 МВ, поданного на электроды ускорителя. Затем пучок перезаряжается, и его энергия удваивается на выходе из ускорителя. Затем пучок поворачивается поворотным магнитом и сбрасывается на литиевую мишень, которая генерирует нейтроны в результате реакции $^{7}\mathrm{Li}(p,n)^{7}\mathrm{Be}$.

Для диагностики пучка на входе в ускоритель был приобретен проволочный профилометр WS-30 (D-Pace, Канада), который позволяет провести измерение профиля пучка. Этот профилометр является единственной неразрушающей диагностикой пучка ионного источника на входе в ускоритель на данный момент.

Управление профилометром осуществляется посредством программы, которая позволяет запускать профилометр и считывать данные. Но эта программа не показывает такие параметры как полный ток и положение пучка, необходимые для оптимального ввода пучка в ускоритель. Поэтому требуется программное обеспечение, которое рассчитывает полный ток пучка, его положение и его размер, производя дополнительную обработку.

Помимо автоматической обработки данных требуется удобное представление данных в пользовательском интерфейсе.

1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Устройство профилометра

Схема проволочного сканера WS-30 показан на рисунке 1 [2]. Он имеет две вольфрамовые проволочки, закрепленные на общем стержне. Длина стержня равна 168.7 мм. Длина ножек составляет 49 мм, а диаметр 0,5 мм. Угол между ножками составляет 90 градусов.

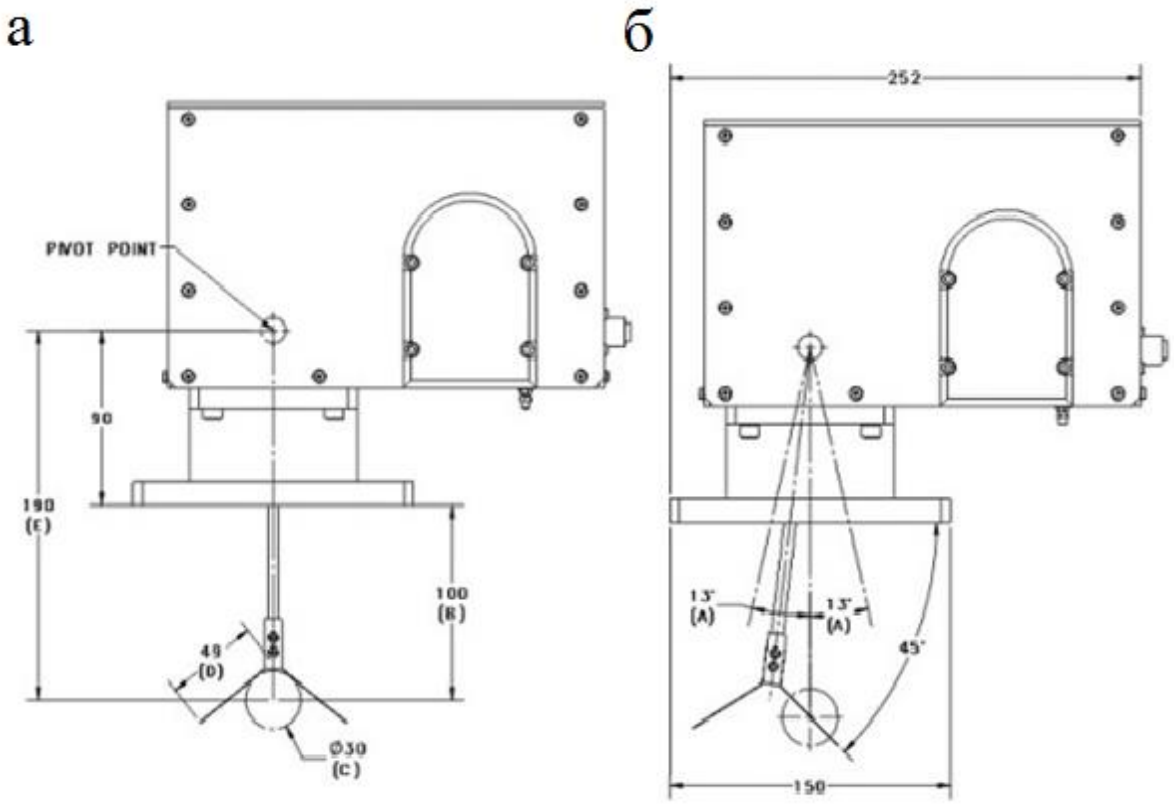

Рисунок 1 – Схема проволочного сканера WS-30: а – вертикальное положение профилометра, б – исходное положение профилометра

При проведении измерения стержень начинает свое движение с 13 градусов, затем поворачивается до минус 13 градусов и возвращается обратно, где 0 градусов – вертикальное положение стержня.

Ток, приходящий на ножки, стекает на землю и измеряется пикоамперметром Keithley 6485.

1.2 Схема экспериментальной установки

На рисунке 2 изображена схема низкоэнергетичного тракта: 1 – источник отрицательных ионов водорода, 2 – магнитные линзы, 3 – корректор, 4 – запирающие кольца, 5 – проволочный профилометр WS-30,6 – вакуумная камера, 7 – электроды ускорителя.

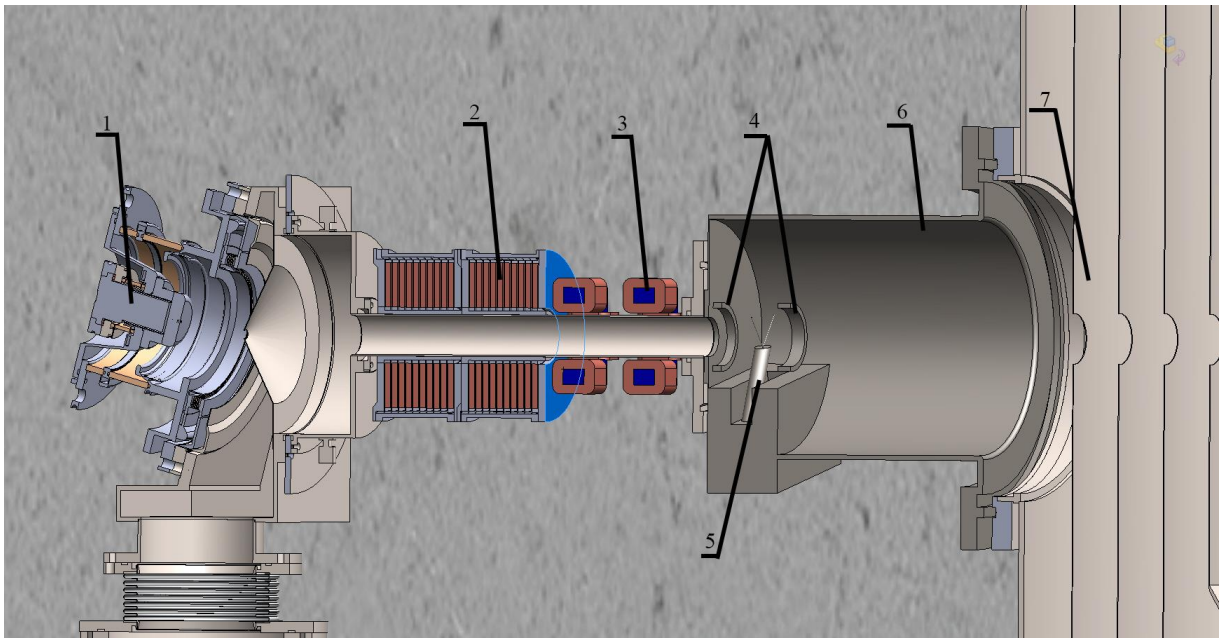

Рисунок 2 – Схема низкоэнергетичного тракта экспериментальной установки

Следует отметить, что профилометр установлен вертикально, ножками вверх. В исходном положении ножки профилометра расположены слева оси движения частиц пучка. При запуске измерения стержень начинает наклоняться направо, и считываемый угол профилометра начинает изменяться от 13 до минус 13 градусов.

Профилометр устанавливался таким образом, чтобы пучок, в нормальном режиме работы, проходил по измеряемой области профилометра. То есть профилометр измеряет область с центром не в оси вакуумной камеры, а в том месте, где проходит пучок. В дальнейшем калибровка позволила определить смещение центра измеряемой плоскости от оси камеры. Оно составило 6.25 мм влево и 0.6 мм вниз.

До и после профилометра расположены кольца, на которые подано напряжение минус 300 В. Они необходимы для запирания вторичной эмис-

сии электронов. Когда тяжелое ядро атома водорода ударяется о проволочку, оно выбивает несколько электронов. Электрическое поле колец не дает легким электронам с низкой начальной энергией вылететь из проволочки и не влияет на ток тяжелых ионов водорода. Таким образом, профилометр считывает только ток отрицательных ионов водорода. Подробнее о запирающих кольцах можно прочитать в публикации [3].

1.3 Управляющая программа

Управление профилометром осуществляется программой Wire Scanner V10.exe, которая поставлена вместе с профилометром. Она осуществляет управление двигателем и сбор данных с амперметра. Главная форма управляющей программы изображена на рисунке 3.

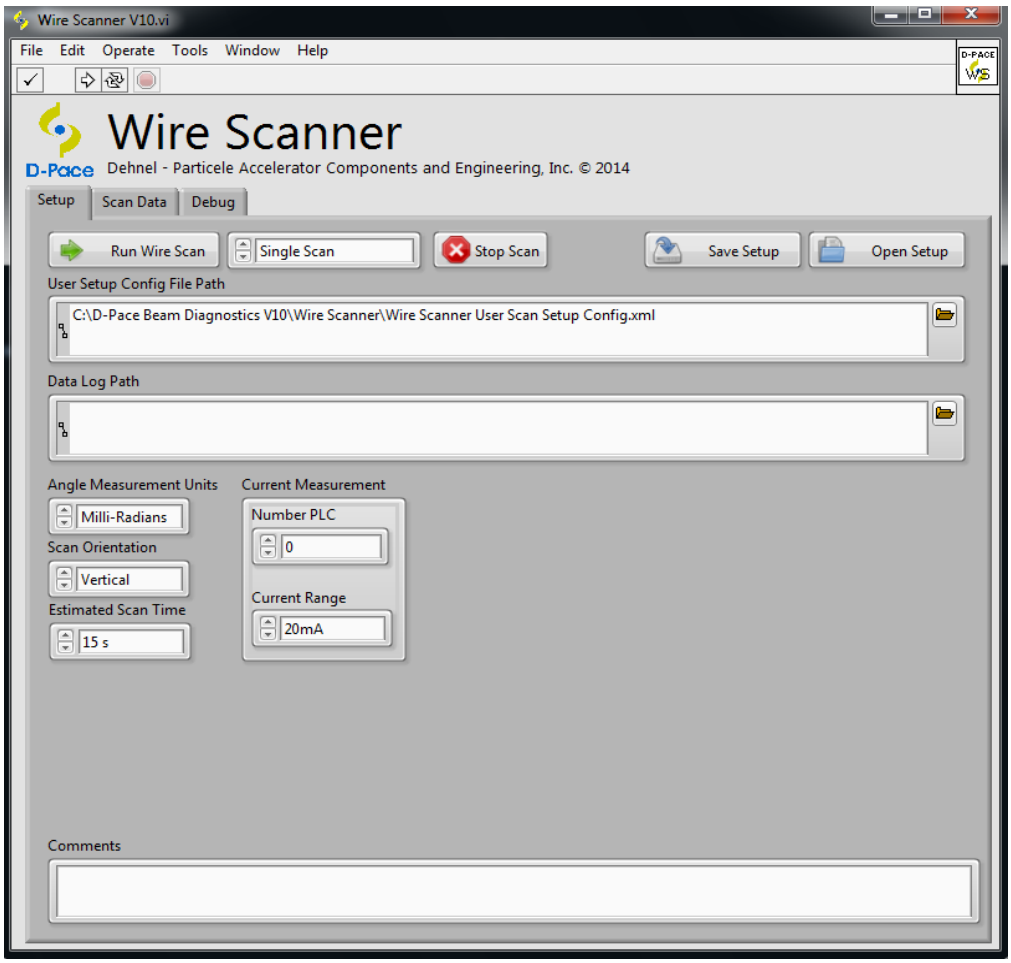

Рисунок 3 – Главная форма программы управления профилометром

Скорость движения профилометра задается временем прохода в поле ***Estimated Scan Time***. Скорость опроса амперметра постоянная и никак не ре-

гулируется. Для запуска сканирования необходимо нажать кнопку ***Run Wire Scan***.

После этого профилометр начнет плавно разгоняться из положения 13 градусов, в какой-то момент его скорость установится, он дойдет до положения минус 13 градусов и плавно остановится, и начнет движение в обратную сторону.

После окончания сканирования программа отображает данные на графике и сохраняет их в файл, указанный в поле ***Data Log Path***. График зависимости тока от положения профилометра изображен на рисунке 4. По оси абсцисс углы положения профилометра, по оси ординат ток, приходящий на проволочку.

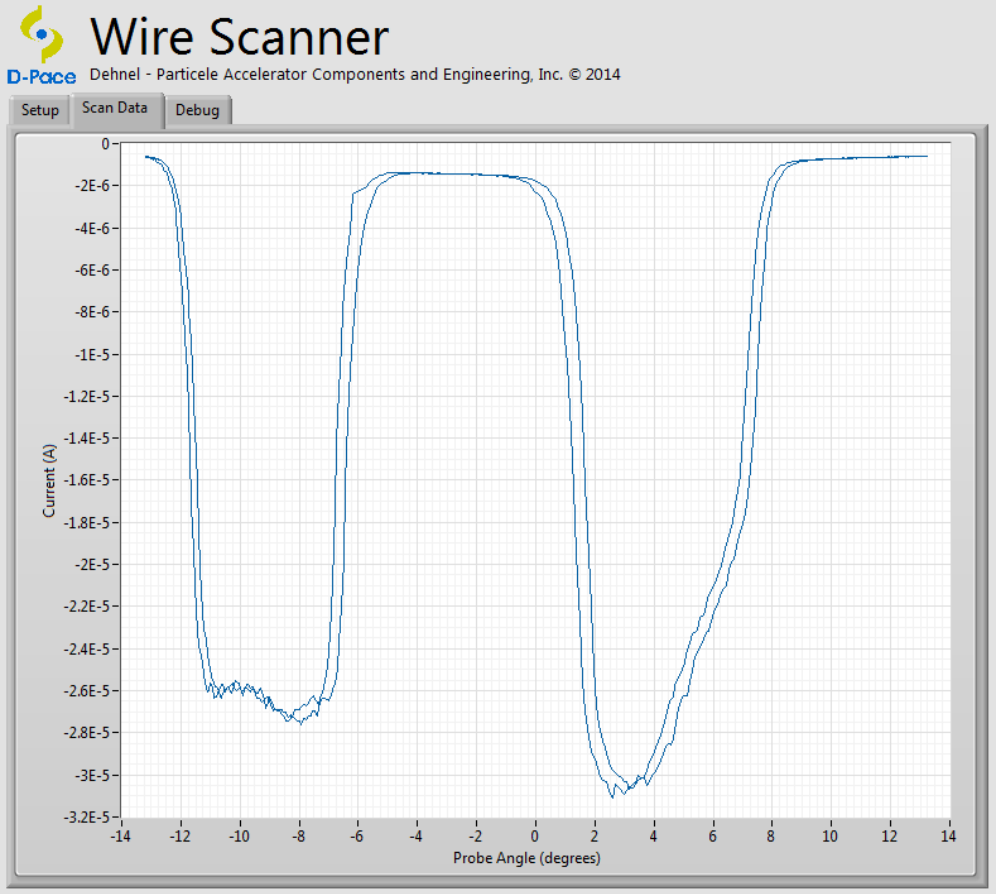

Рисунок 4 – График зависимости тока от положения профилометра после сканирования

В ходе эксплуатации было обнаружено, что профилометр считывает профиль пучка смещенным при проходе проволочек в разных направлениях. При этом форма пучка не изменялась. Было сделано предположение, что

сдвиг профилей был вызван физическим люфтом двигателя, а измерения углов осуществлялись по выставленному значению, а не по реальному. Этот люфт был измерен и составляет 0.4 градуса. Но направление люфта неизвестно. Этот сдвиг видно на рисунке 4.

Данные сканирования сохраняются в текстовый файл. Файл имеет следующую структуру: заголовок, где описываются имя файла, комментарий, время сканирования и единицы измерения, и раздел данных, который состоит из строк, где в каждой строке первое число – это угол в градусах, а второе число – это ток, приходящий на проволочку в амперах. При этом в одном файле может быть несколько кадров прохода профилометра от положения 13 градусов до минус 13 градусов. Кадры никак не разделены. Этот файл сохраняется только после окончания сканирования. На рисунке 5 представлен пример содержимого такого файла.

```
File name: C:\D-Pace Beam Diagnostics V10\Scan Data\Wire Scanner\26.07.17\12.csv,
Comments: ,
Scanner Type: Wire Scanner,
Wire Scanner Serial Number: WS002,
Scan Start Time: 1:03:48 PM 7/26/2017,
Scan Orientaion: Vertical,
Minimum Probe Angle (degrees): -14.0000,
Maximum Probe Angle (degrees): 14.0000,
,
Scan Angle (degrees),Current (A)
13.055168,-8.486211E-6
13.044801,-8.493662E-6
13.069127,-8.475035E-6
13.086656,-8.501112E-6
13.101878,-8.493662E-6
13.116007,-8.486211E-6
```

Рисунок 5 – Пример содержимого файла с данными сканирования

2 ПОСТАНОВКА ЗАДАЧИ

Необходимо разработать программное обеспечение, производящее дополнительную обработку данных из основной программы управления профилометром. Разработанная программа должна удовлетворять следующим функциональным и нефункциональным требованиям и ограничениям.

2.1 Функциональные требования

Разработанное программное обеспечение должно удовлетворять функциональным требованиям, которые описаны ниже.

- Расчет интегрального тока пучка с учетом геометрии движения и размеров проволочки.
- Перевод из углов положения профилометра в декартову систему координат.
- Возможность отобразить двумерный профиль пучка.
- Расчета размера пучка.
- Добавить алгоритм шумоподавления.
- Возможность отображать данные заданного кадра прохода профилометра.
- Отобразить данные сканирования на графике, с возможностью изменять стиль, цвет, ширину линии.
- Возможность сохранить изображение графика тока и двухмерного профиля.
- Все геометрические параметры проволочек должны храниться в файле настроек.
- Возможность изменять файл настроек из пользовательского интерфейса.

2.2 Нефункциональные требования

Разработанное программное обеспечение должно удовлетворять нефункциональным требованиям, которые описаны ниже.

– Реализовать графический интерфейс пользователя, в котором должны быть отображены все данные о пучке: положение максимума, ширина на полувысоте, общий ток, информация на заданном интервале, площадь занимаемая пучком.
– Спроектировать приложение таким образом, чтобы можно было отображать несколько файлов и их данных на одном графике, для возможности сравнения.
– Автоматическая загрузка данных в приложение после сканирования.

2.3 Ограничения

Ниже приведены основные факторы ограничивающие выбор возможных решений по реализации отдельных требований.

– Так как нет возможности получить данные о сканировании из основной программы напрямую в момент измерения, ограничиться получением данных из выходного файла программы. Таким образом, обработка будет происходить только после окончания сканирования.
– При построении двухмерного профиля пучка ограничиться произведением данных двух ножек. Это не будет являться корректным профилем пучка, т.к. каждый отсчет ножка собирает сумму токов, приходящих на нее.

3 ОПИСАНИЕ МЕТОДОВ

3.1 Метод расчета полного тока пучка

Чтобы рассчитать полный ток пучка необходимо просуммировать все токи, которые измеряет профилометр. Но для корректного расчета необходимо учесть геометрию движения и площадь, которую измеряет ножка профилометра.

В выходном файле значение тока – это суммарный ток, приходящий на проволочку, когда профилометр имеет соответствующее положение угла. Скорость изменения угла не постоянна и изменяется. В начале сканирования скорость изменения угла не велика т.к. профилометр разгоняется, в дальнейшем скорость увеличивается и устанавливается на постоянном значении. Скорость опроса амперметра постоянная. Таким образом, возможны следующие варианты:

а) Скорость изменения угла низкая, и профилометр принимает ток с одного и того же участка пучка несколько раз.

б) Скорость изменения угла слишком высока и профилометр пропускает некоторые участки пучка, где ток остается неучтенным.

На рисунке 6 схематично изображены оба этих случая: где Δa – угол на который изменилось положение профилометра за время t_2 минус t_1, S – площадь, которая в первом случае измеряется два раза, а во втором случае не измеряется.

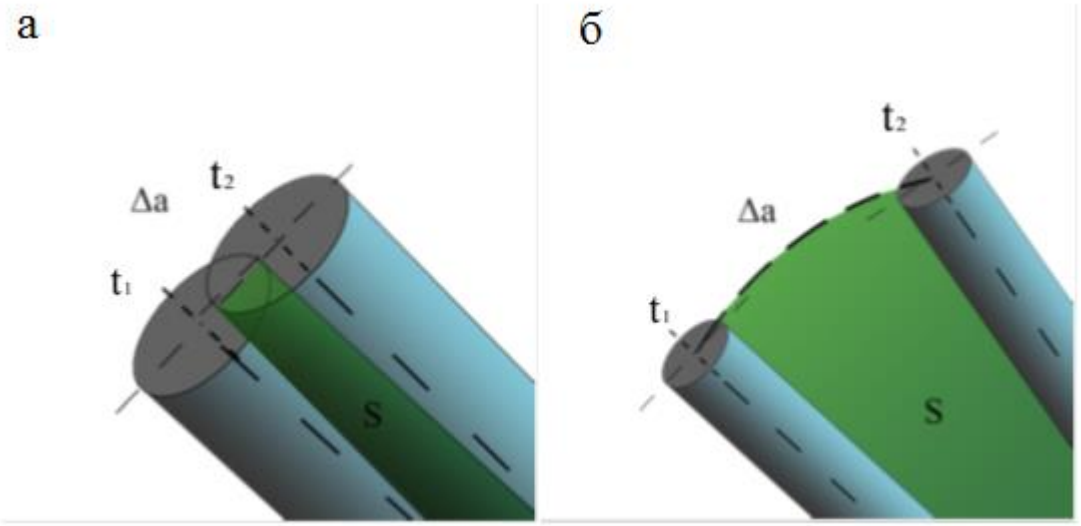

Рисунок 6 – Измеряемая площадь при разных скоростях профилометра: а – скорость низкая, б – скорость высокая

Для того чтобы учесть оба этих случая необходимо умножить текущее значение тока на коэффициент, который равен отношению площади S, которую прошла проволочка при изменении угла к площади, занимаемой самой проволочкой. Ниже представлена формула 1 для расчета текущего значения тока.

$$I = I_i \frac{S(\Delta a_i)}{S_{leg}} , \quad (1)$$

где I_i – это ток на проволочке, S_{leg} – площадь.

Эта площадь S_{leg} равна длине ножки умноженной на диаметр, что в свою очередь есть 49*0.5, и $S(\Delta a_i)$ – это площадь, которую прошла ножка за измененный угол Δa_i. Далее задача сводится к геометрической задаче нахождения этой площади $S(\Delta a_i)$.

На рисунке 7 изображена геометрия движения профилометра, где жирными линиями выделены положения профилометра с одной ножкой во время t_1 и t_2.

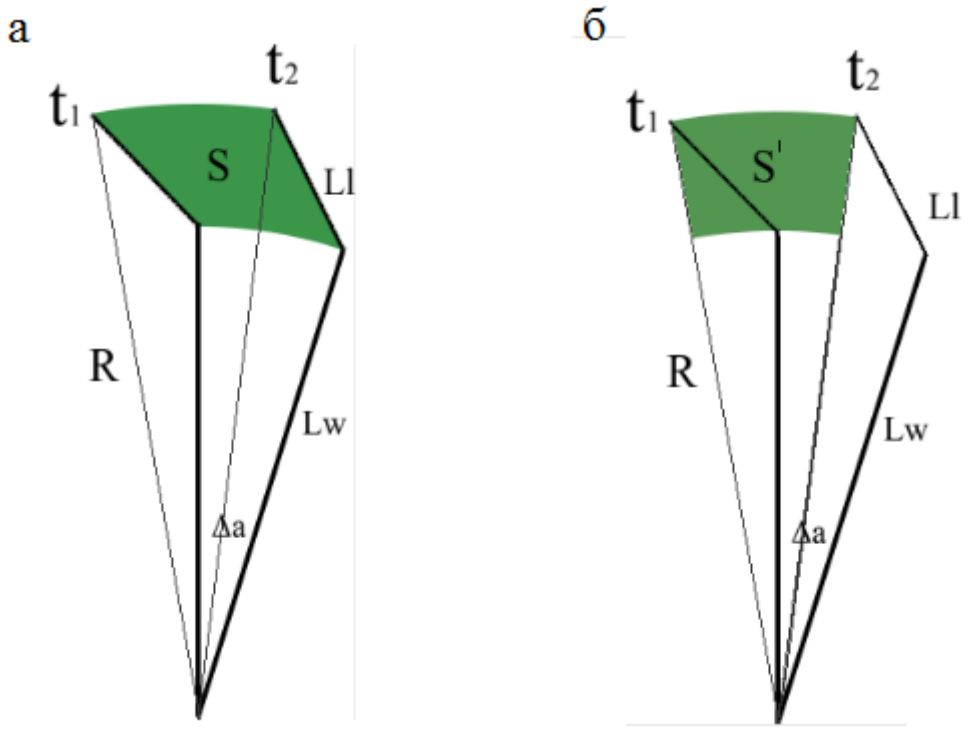

Рисунок 7 – Геометрия движения профилометра: а – площадь S, которую необходимо найти, б – эквивалентая площадь S’

Чтобы найти площадь S, можно перейти к нахождению площади сектора круга S’ с радиусом R, который описывает конец ножки. На рисунке 8 Δa – изменение угла за время t_2 минус t_1, Lw – длина основания, Ll – длина ножки.

По известным значениям Ll, Lw и угла между ними можно найти R по теореме косинусов:

$$R = \sqrt{Lw^2 + Ll^2 - 2 \times Lw \times Ll \times \cos(b)} \ . \quad (2)$$

В формуле 2 угол b это угол между основанием и ножкой. Так как угол между ножками известен и равен 90 градусов, угол b равен

$$b = \ 180 - \frac{90}{2} = 135 \ . \quad (3)$$

Чтобы найти площадь S′ необходимо вычесть площадь сектора круга с радиусом Lw из площади сектора круга с радиусом R. Сектор круга вычисляется по известной формуле. Итого получается:

$$S' = \pi \times R^2 \frac{\Delta a}{360} - \pi \times Lw^2 \frac{\Delta a}{360} \ . \quad (4)$$

В результате получается, что площадь зависит только от Δa, а полный ток пучка I рассчитывается по формуле 5

$$I = \sum I_i \frac{S'(\Delta a_i)}{S_{leg}} \ . \quad (5)$$

Таким образом, в тех местах, где скорость профилометра маленькая и Δa не велико, площадь, которую прошел профилометр, оказывается меньше, чем площадь, которую покрывает проволочка и этот коэффициент меньше 1. А в тех местах, где скорость движения профилометра большая и S′ больше S_{leg} коэффициент больше 1.

3.2 Метод пересчета из углов положения профилометра в координаты на плоскости

Каждые два угла профилометра, с первой ножки и второй, можно отобразить в координаты на плоскости. На рисунке 8 показаны два состояния профилометра, когда одна из ножек проходит по той же плоскости что и другая, но с другим углом. Зная эти углы можно вычислить координаты их пересечения.

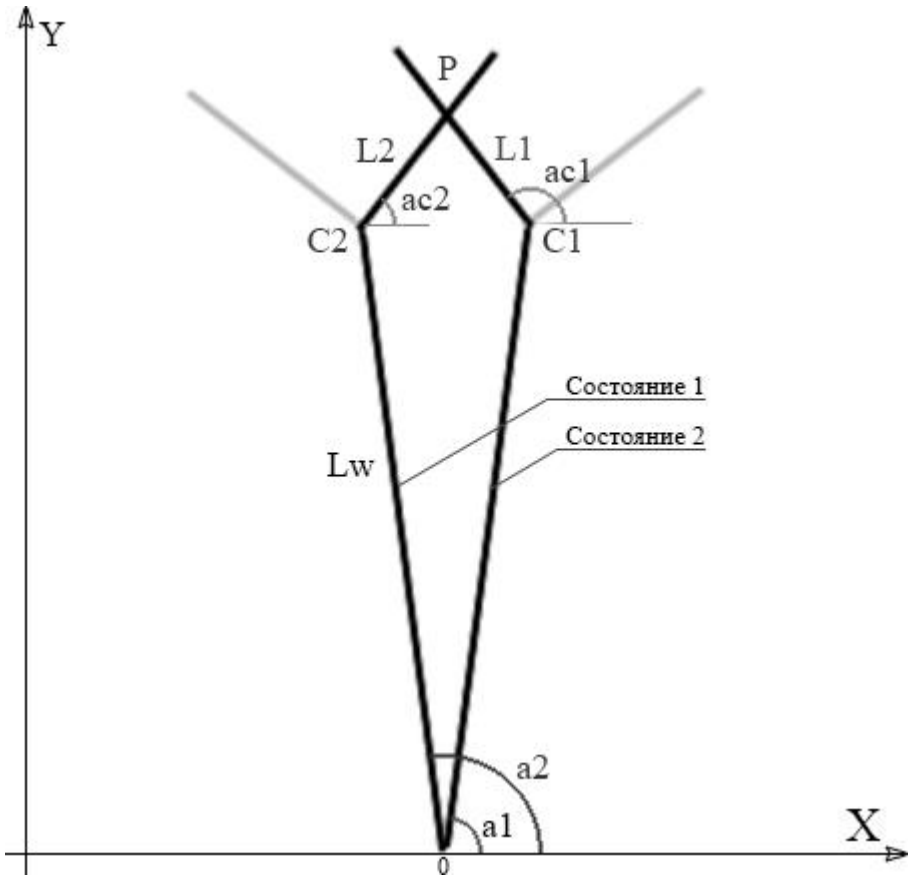

Рисунок 8 – Положения профилометра при угле a1 и угле a2

При определенных углах разные ножки профилометра проходят одну и ту же плоскость под разным углом. На рисунке 8 показаны такие углы a1 и a2. Lw – это длина основания, углы ac1 и ac2 – это углы наклона ножки к оси абсцисс, L1 и L2 – это неизвестная длина от основания до точки пересечения P пересечения с неизвестными координатами (X,Y). На самом профилометре углы отсчитываются от вертикального положения, и для удобства можно перевести в углы от оси абсцисс по формуле 6.

$$a1 = 90 - a1' \ ,$$
$$a2 = 90 - a2' \ , \qquad (6)$$

где $a1'$ и $a2'$ это углы профилометра от вертикального положения, которые записываются в файл.

Можно выразить координаты точки P(X, Y) через уравнение от разных углов a1 и a2, и составить систему этих уравнений. Для этого нужны коорди-

наты точки C(Xc,Yc) узла крепления ножек. Для состояния профилометра 1 и 2 координаты точек C1(Xc1,Yc1) C2(Xc2,Yc2) узла крепления проволочек будут выражаться по формулам 7:

$$\begin{aligned} Xc1 &= Lw \times \cos(a1) \ , \\ Yc1 &= Lw \times \sin(a1) \ , \\ Xc2 &= Lw \times \cos(a2) \ , \\ Yc2 &= Lw \times \sin(a2) \ . \end{aligned} \tag{7}$$

Далее координаты точки пересечения проволочек P(X, Y) можно выразить через эти координаты узла крепления и неизвестные длины L1 и L2, но вначале необходимо пересчитать углы для центра ac1 и ac2. Угол между ножками 90 градусов, но необходимо вынести этот параметр для настроек, поэтому угол между ножками обозначим La. Углы ac1 и ac2 выражаются из a1 и a2 по следующим формулам 8:

$$\begin{aligned} ac1 &= a1 + \frac{La}{2} \ , \\ ac2 &= a2 - \frac{La}{2} \ . \end{aligned} \tag{8}$$

Теперь можно записать систему уравнений 9 координат точки пересечения.

$$\begin{cases} X = L1 \times \cos(ac1) + Xc1 \\ Y = L1 \times \sin(ac1) + Yc1 \\ X = L2 \times \cos(ac2) + Xc2 \\ Y = L2 \times \sin(ac2) + Yc2 \ . \end{cases} \tag{9}$$

Для того, чтобы избавиться от L1 L2 выразим их, перенесем Xc Yc в левую часть и разделим на множитель при L1 L2. В результате получится система уравнений 10.

$$\begin{cases} \frac{X}{\cos(ac1)} - \frac{Xc1}{\cos(ac1)} = L1 \\ \frac{Y}{\sin(ac1)} - \frac{Yc1}{\sin(ac1)} = L1 \\ \frac{X}{\cos(ac2)} - \frac{Xc2}{\cos(ac2)} = L2 \\ \frac{Y}{\sin(ac2)} - \frac{Yc2}{\sin(ac2)} = L2 \end{cases} . \quad (10)$$

Теперь избавимся от L1 L2, приравняем уравнения, получается система 11.

$$\begin{cases} \frac{X}{\cos(ac1)} - \frac{Xc1}{\cos(ac1)} = \frac{Y}{\sin(ac1)} - \frac{Yc1}{\sin(ac1)} \\ \frac{X}{\cos(ac1)} - \frac{Xc1}{\cos(ac1)} = \frac{Y}{\sin(ac1)} - \frac{Yc1}{\sin(ac1)} \end{cases} . \quad (11)$$

Перенесем неизвестные члены в левую часть, а известные в правую, получается система 12.

$$\begin{cases} \frac{X}{\cos(ac1)} - \frac{Y}{\sin(ac1)} = \frac{Xc1}{\cos(ac1)} - \frac{Yc1}{\sin(ac1)} \\ \frac{X}{\cos(ac2)} - \frac{Y}{\sin(ac2)} = \frac{Xc2}{\cos(ac2)} - \frac{Yc2}{\sin(ac2)} \end{cases} . \quad (12)$$

Можно расписать систему относительно углов a1 a2. Вместо Xc Yc и ac1 и ac2 записать их уравнения, получится система 13.

$$\begin{cases} \frac{X}{\cos\left(a1+\frac{La}{2}\right)} - \frac{Y}{\sin\left(a1+\frac{La}{2}\right)} = \frac{Lw\times\cos(a1)}{\cos\left(a1+\frac{La}{2}\right)} - \frac{Lw\times\sin(a1)}{\sin\left(a1+\frac{La}{2}\right)} \\ \frac{X}{\cos\left(a2-\frac{La}{2}\right)} - \frac{Y}{\sin\left(a2-\frac{La}{2}\right)} = \frac{Lw\times\cos(a2)}{\cos\left(a2-\frac{La}{2}\right)} - \frac{Lw\times\sin(a2)}{\sin\left(a2-\frac{La}{2}\right)} \end{cases} . \quad (13)$$

Теперь получилась система уравнений с двумя уравнениями и двумя переменными. Ее можно решить любым известным способом.

Такая система уравнений не будет иметь решения, только когда не будет точки пересечения, т.е. когда ножки параллельны, во всех остальных случаях нет никакого ограничения на углы a1 и a2, и длину L1 и L2 и результат будет. Хотя предполагается что углы положения профилометра a1′ и a2′, из которых получаются a1 и a2 по формуле 6, принадлежат интервалам от 13 до 0 и от 0, до минус 13 соответственно. А длины L1 и L2 не могут быть отрицательными и бóльшими, чем длина ножки, которая равна 49 мм, хотя при решении этой системы будут возникать и такие решения.

Теперь, имея два угла a1 и a2, можно решить эту систему и найти координаты пересечения (X, Y). Решение этой системы для углов a1 и a2 оформляется в функцию, которая возвращает значения (X, Y).

3.3 Метод расчета размера пучка

Метод расчета размера пучка основывается на предыдущем алгоритме вычисления координат. Размер пучка вычисляется на полувысоте профиля. На рисунке 9 изображен профиль пучка.

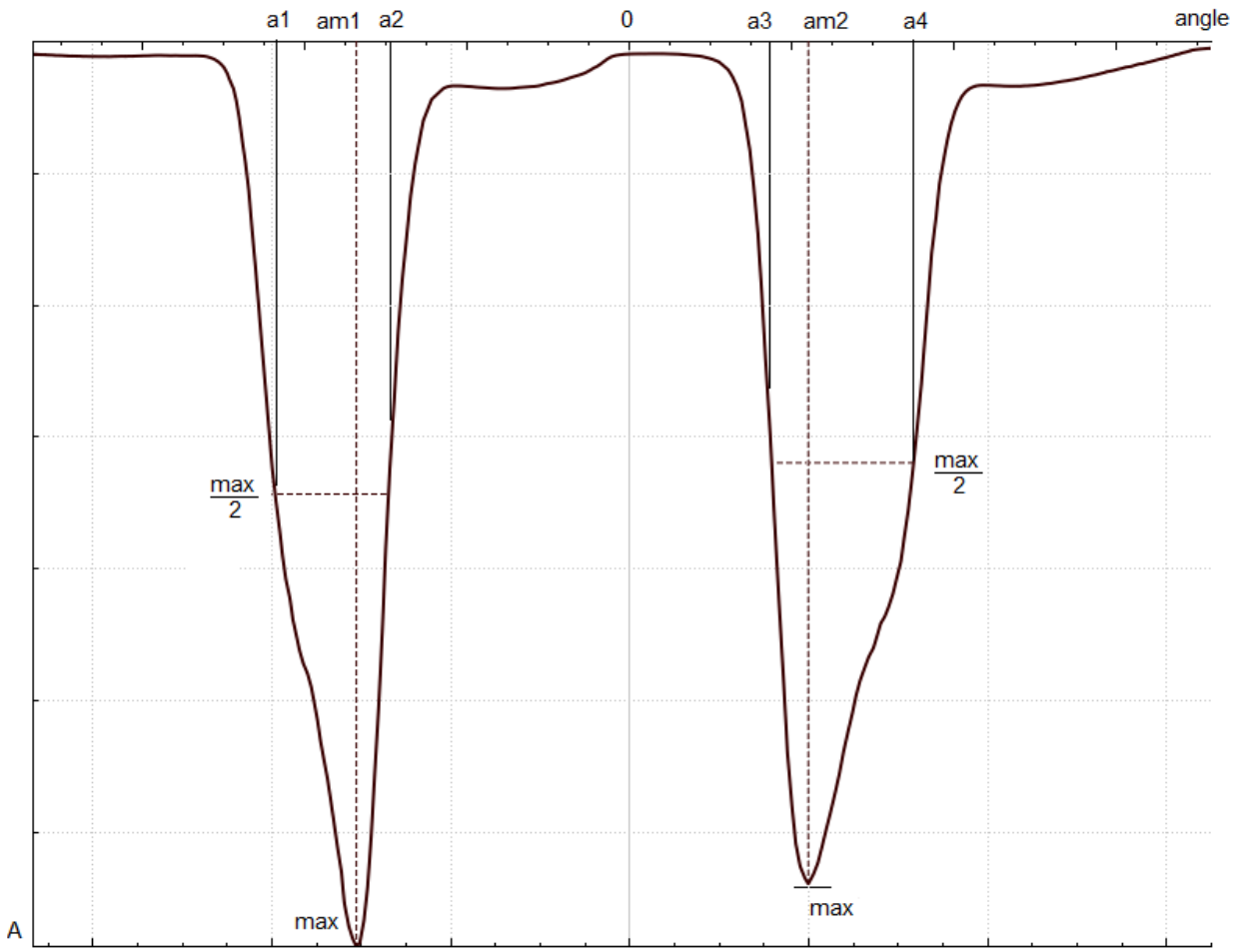

Рисунок 9 – Профиль пучка. На оси абсцисс откладываются углы профилометра, ось ординат - это ток

В начале определяются значения максимумов max и углы a1, a2 и a3, a4 для первой и второй ножки соответственно. Для каждой ножки эти значения свои. Интервалы углов для первой и второй ножки разделены нулем. Т.е. считается, что первая ножка имеет углы от 13 градусов до нуля, а вторая ножка от нуля до минус 13.

Далее находятся пересечения с графиком на уровне max/2. Бывает, что этих пересечений меньше или больше четырех. В таких случаях алгоритм возвращает ошибку. Если этих значений 4, то определяются углы при этих пересечениях.

Теперь, когда имеются углы на полувысоте и углы максимумов, можно найти координаты отрезков на плоскости. Длины этих отрезков являются шириной пучка в одной плоскости и в другой. Здесь нельзя употребить тер-

мин ширина и высота, потому что эти длины не являются таковыми. На рисунке 10 изображены положения профилометра при таких углах.

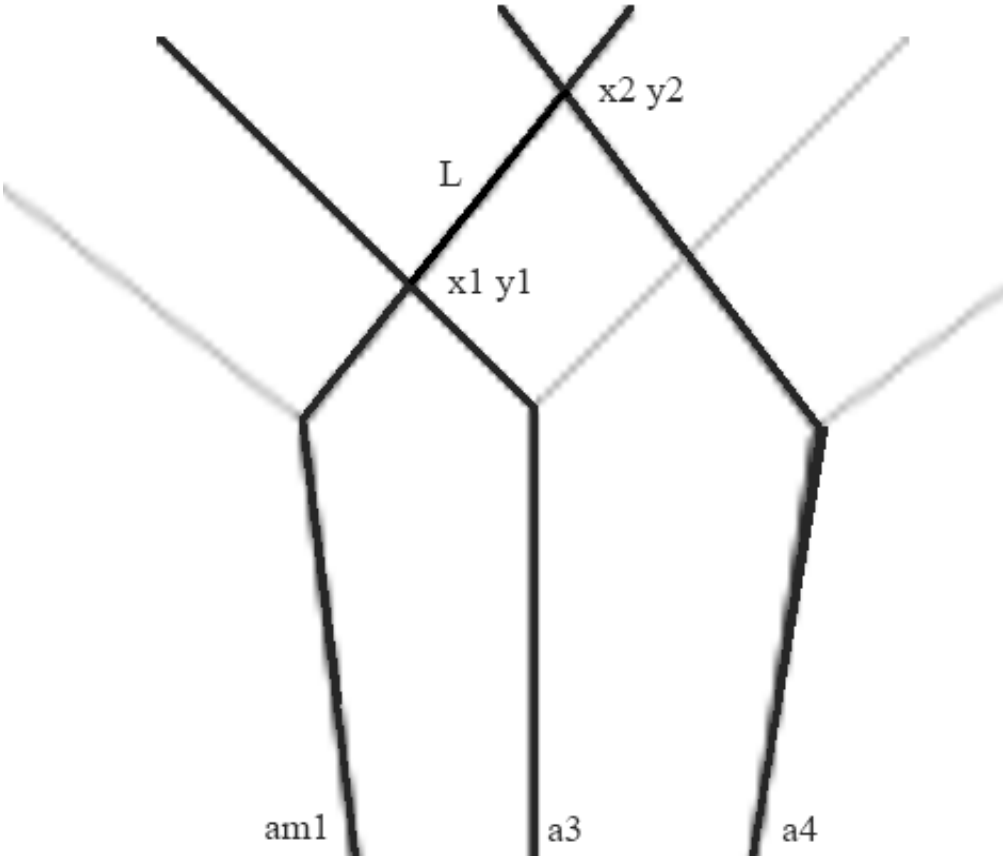

Рисунок 10 – Положения профилометра при углах на полувысоте и угле максимума

Для того чтобы найти (x1, y1) необходимо подставить углы am1 и a3 в функцию пересчета углов в координаты, которая описана в предыдущей главе. Аналогично для точки (x2, y2) – подставить углы am1 и a4. Имея координаты точек можно вычислить L – расстояние от (x1, y1) до (x2, y2) по известной формуле 10:

$$L = \sqrt{(x2 - x1)^2 + (y2 - y1)^2} \ . \qquad (14)$$

Таким образом, рассчитывается и другая ширина, для которой используется угол второго максимума am2 и углы a1 и a2.

В программе этот алгоритм расширяется, и пользователю предоставляется возможность выбрать, на какой высоте от максимума рассчитывать размер пучка.

3.4 Метод шумоподавления

Шумоподавление требуется для предварительной обработки данных, чтобы убрать незначительные всплески и упростить дальнейшую обработку. Для удаления шумов хорошо подходят алгоритмы частотного анализа данных. Существует множество алгоритмов частотного анализа. Так как данные, с которыми работает приложение, являются не периодичными, было выбрано дискретное вейвлет-преобразование [4].

Входные данные тока раскладываются на вейвлет-спектр, далее коэффициенты со значением ниже порогового удаляются, затем происходит обратное преобразование. Пороговое значение задается пользователем как процент от максимального значения. Таким образом из сигнала удаляются небольшие всплески и остаются все значащие данные.

В качестве материнского вейвлета для преобразования был выбран сплайн вейвлет, который, как показала практика, лучше всего сглаживает сигнал, т.к. сам является гладкой функцией.

На рисунке 11 представлены необработанные данные. Как видно, в сигнале есть небольшие шумовые всплески.

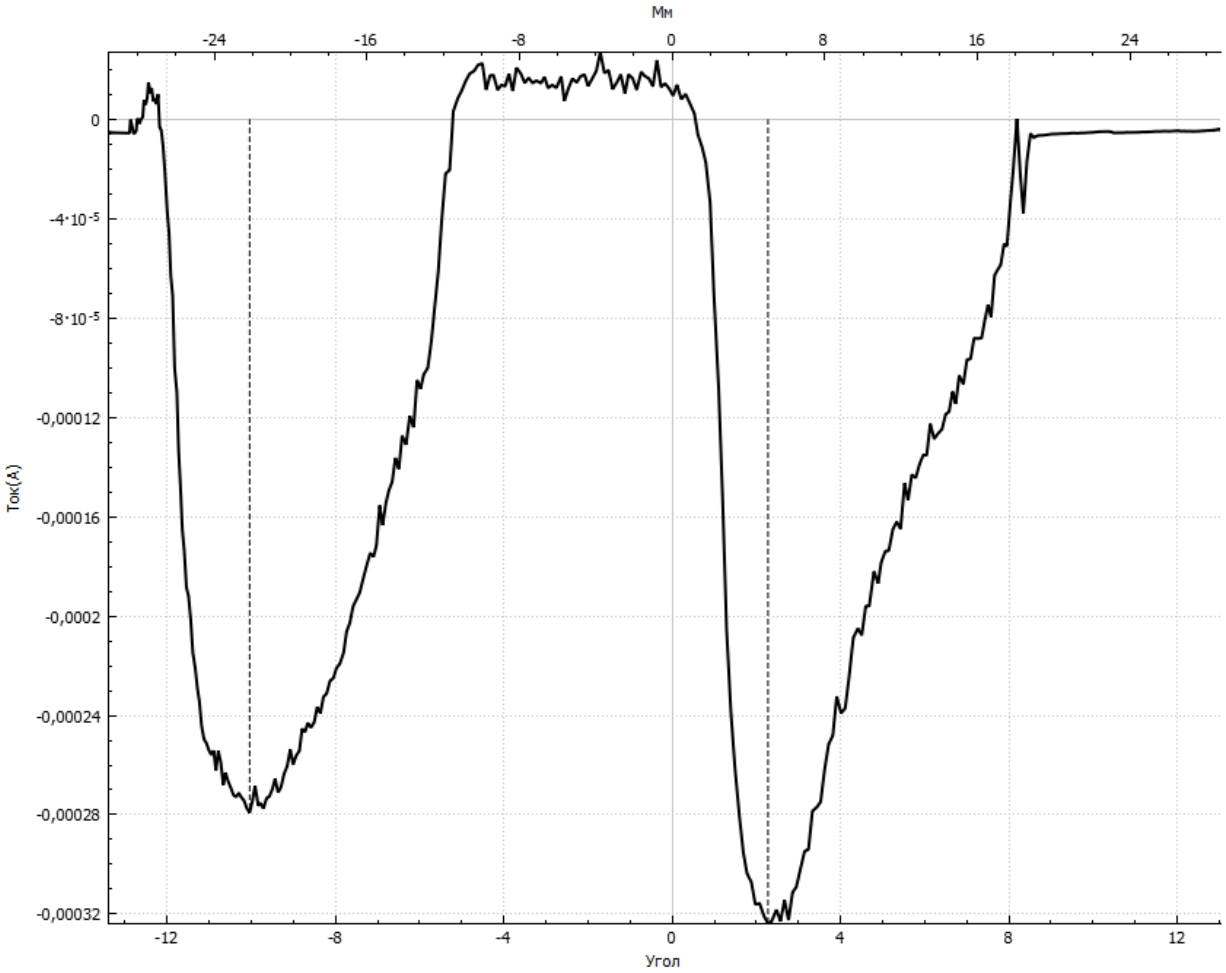

Рисунок 11 – Необработанные данные профиля пучка

Если применить вышеописанный алгоритм с 3 процентами порога на этот сигнал небольшие всплески сгладятся, результат показан на рисунке 12.

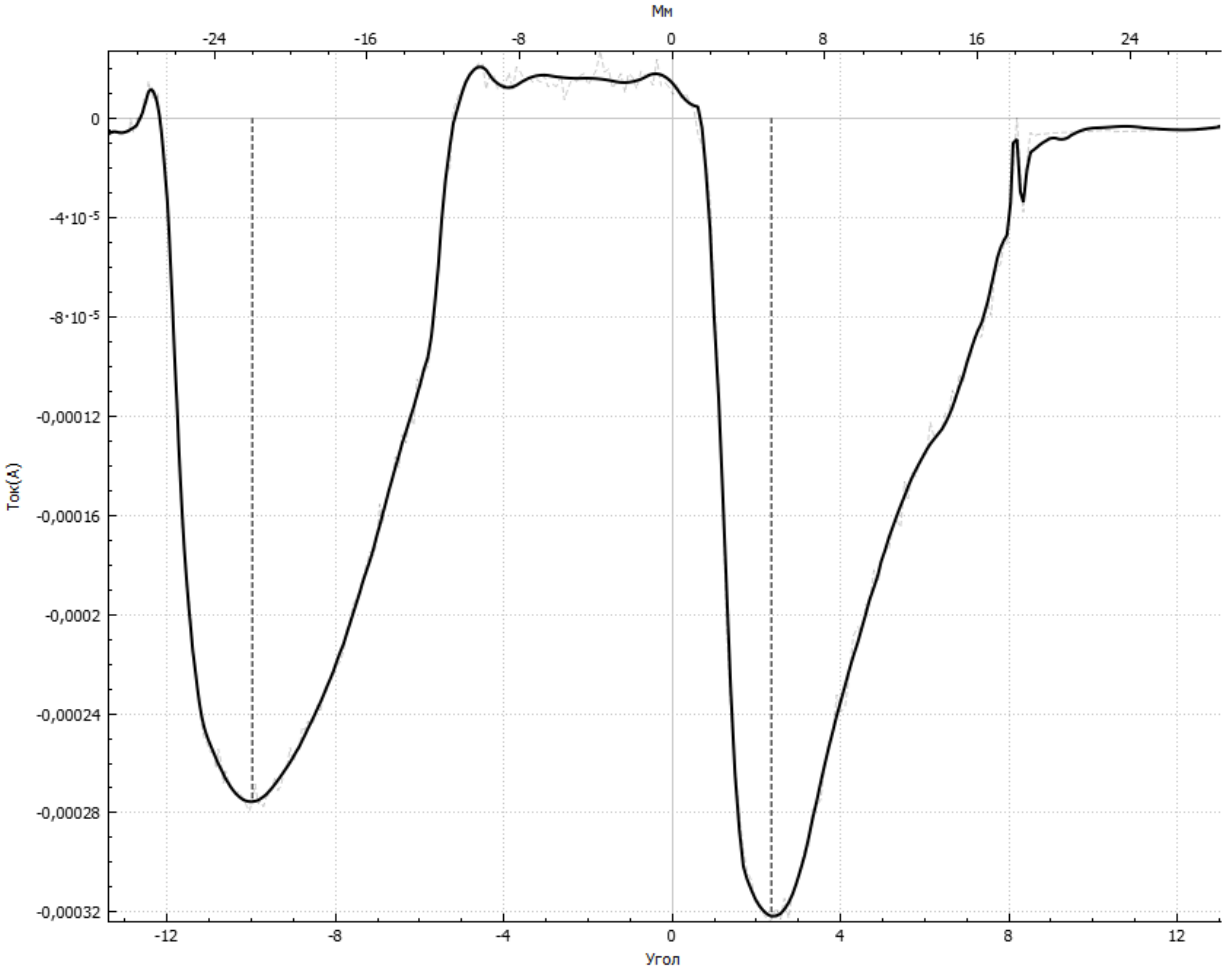

Рисунок 12 – Результат применения алгоритма шумоподавления с 3 процентами порога

Далее если выставить порог шума на 30 процентов, то останется только самый больший по амплитуде и низкий по частоте коэффициент в сигнале. Результат на рисунке 13.

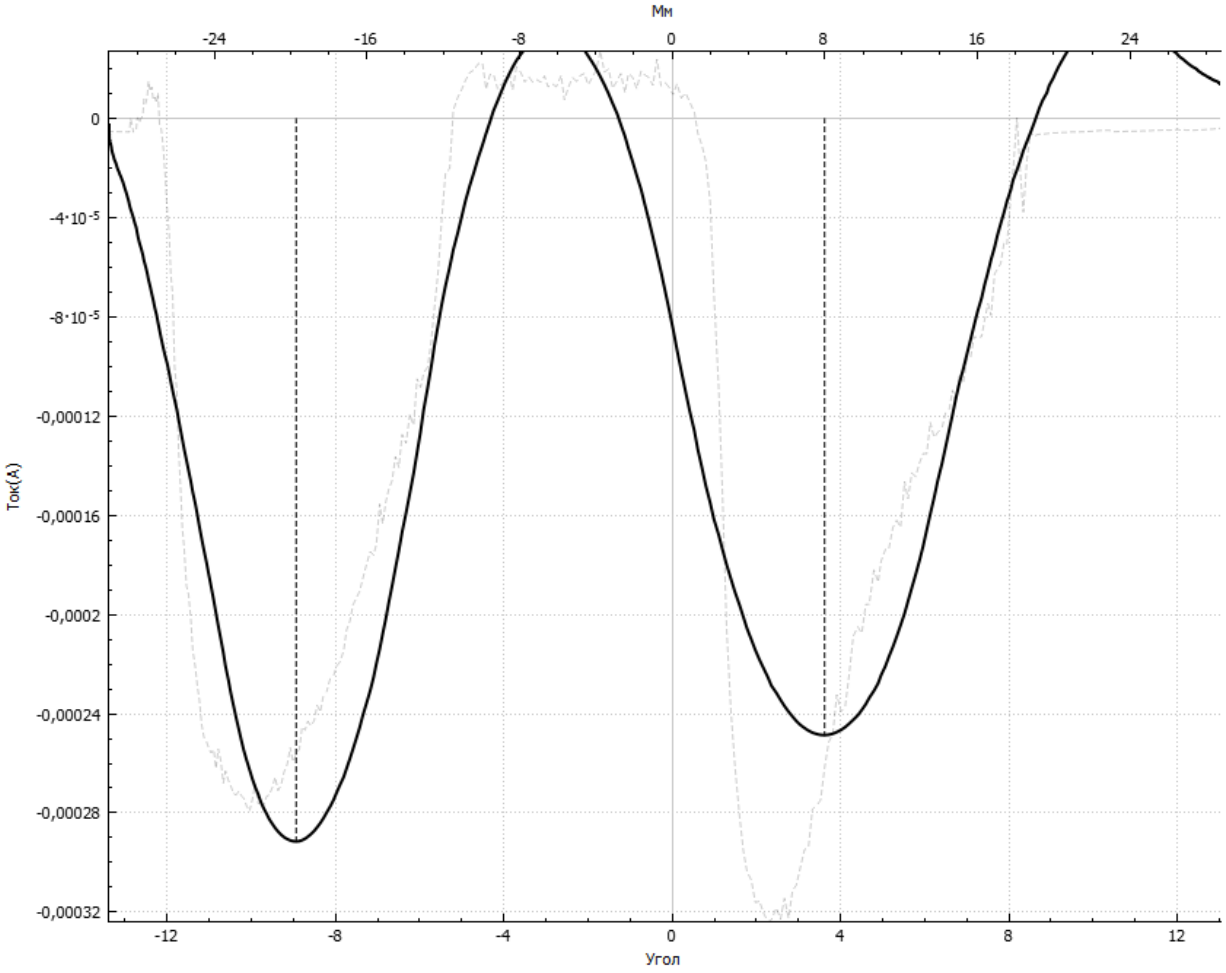

Рисунок 13 – Результат применения алгоритма шумоподавления с 30 процентами порога

4 ОПИСАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Для разработки приложения был выбран язык C++ и фреймворк Qt [5]. Qt – это фреймворк, позволяющий разрабатывать приложения с графическим интерфейсом пользователя на языке C++. При этом код, написанный на Qt, не привязывается к системе и является кроссплатформенным.

4.1 Описание пользовательского интерфейса

Главная форма разработанного приложения представлен на рисунке 14.

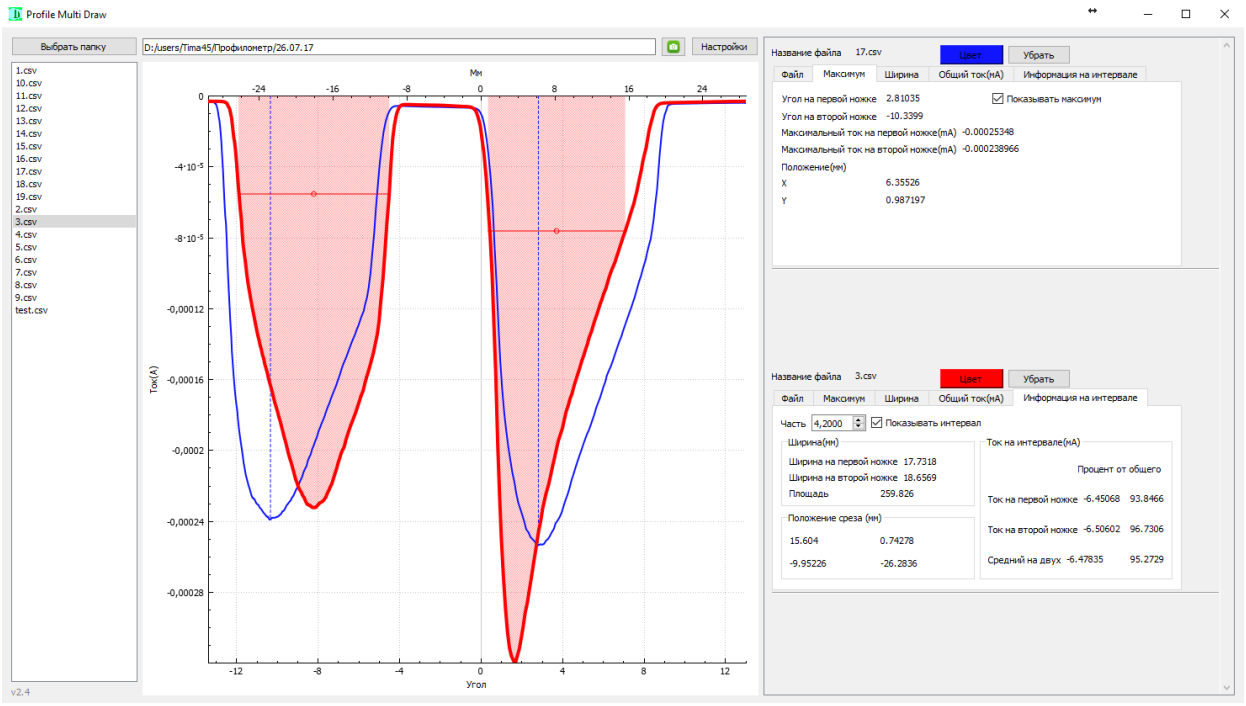

Рисунок 14 –Главная форма разработанного приложения

Для того, чтобы загрузить данные в приложение, необходимо нажать кнопку ***Выбрать папку***. Это вызовет стандартное диалоговое окно выбора дериктории. Необходимо выбрать папку, в которой содержатся файлы данных с расширением ***.csv***, которые сгенерировала программа управления профилометром. После того как папка выбрана, файлы с расширением ***.csv*** отображаются в списке слева. Для удобства выбранная директория сохраняется в текстовое поле сверху. Эта директория начинает просматриваться раз в секунду на наличие новых или измененных файлов, и если таковые есть, то они добавляются в список и отображаются на графике.

Для того, чтобы отобразить информацию в файле, необходимо кликнуть на него два раза в списке, тогда эти данные отобразятся на графике, и создастся новый блок формы справа, в котором будут отображены все вычисляемые данные о пучке и настройки графика. Если необходимо отобразить много файлов, то блоки формы будут добавляться снизу и появится слайдер для прокрутки.

Сам блок формы содержит все вычисляемые параметры пучка и настройки отображения графика. Но информации было настолько много, что пришлось разместить ее на разных вкладках. Над всеми вкладками отображено название файла, есть кнопка, позволяющая менять цвет графика, и кнопка ***Убрать***, которая удаляет график и данные о нем из отображения.

Вкладка ***Файл*** отображает общую информацию о файле и настройки графика, интерфейс изображен на рисунке 15. Так как в файле может быть несколько кадров, есть слайдер, который управляет отображением кадра. Пользователь может изменить размер и тип линии на графике. Кнопка ***Профиль*** вызывает новое окно, в котором отображается двумерный профиль пучка. Слайдер шумоподавления позволяет пользователю подобрать пороговое значение для алгоритма удаления шумов.

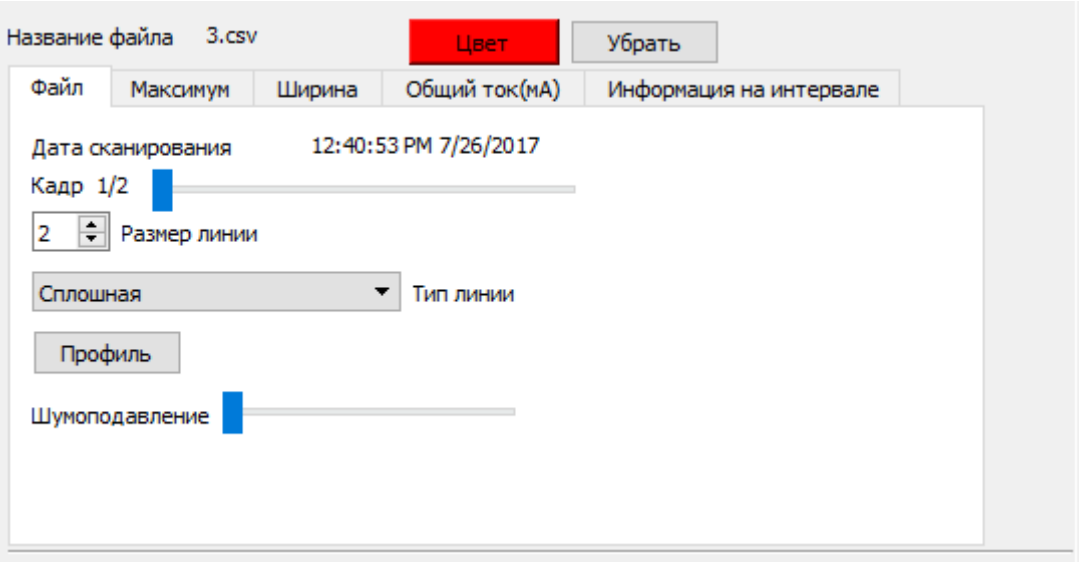

Рисунок 15 – Вкладка ***Файл*** в блоке данных

Вкладка ***Максимум*** отображает данные о максимуме пучка: угол максимума, максимальное значение тока и положение максимума в координатах

на плоскости. Также есть кнопка ***Показать максимум***, которая добавляет пунктирную линию на график в точке максимума. Вкладка представлена на рисунке 16.

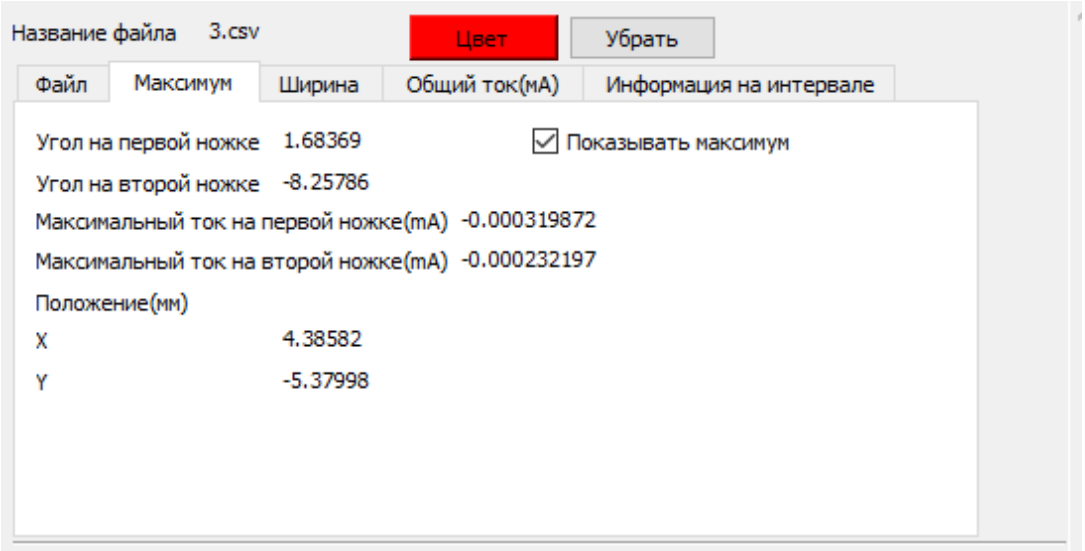

Рисунок 16 – Вкладка ***Максимум*** в блоке данных

На вкладке ***Ширина*** отображаются данные о ширине пучка на полувысоте. Данные вычисляются по алгоритму, описанному выше. Для удобства было добавлено поле ***Площадь***, которое отображает площадь эллипса, где ширина и высота эллипса есть ширины на двух ножках. Вкладка представлена на рисунке 17.

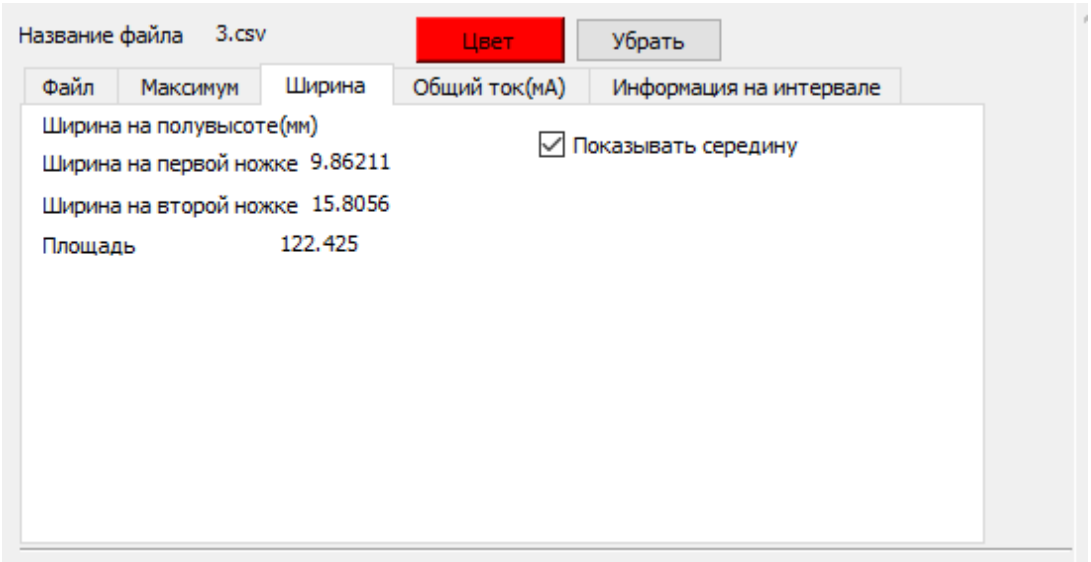

Рисунок 17 – Вкладка ***Ширина***

На вкладке ***Общий ток*** отображается полный ток пучка в миллиамперах. Интервал суммирования для первой ножки от 13 до 0 градусов, для вто-

рой ножки от 0 до минус 13. Также отображается средний ток на двух ножках. Вкладка изображена на рисунке 18.

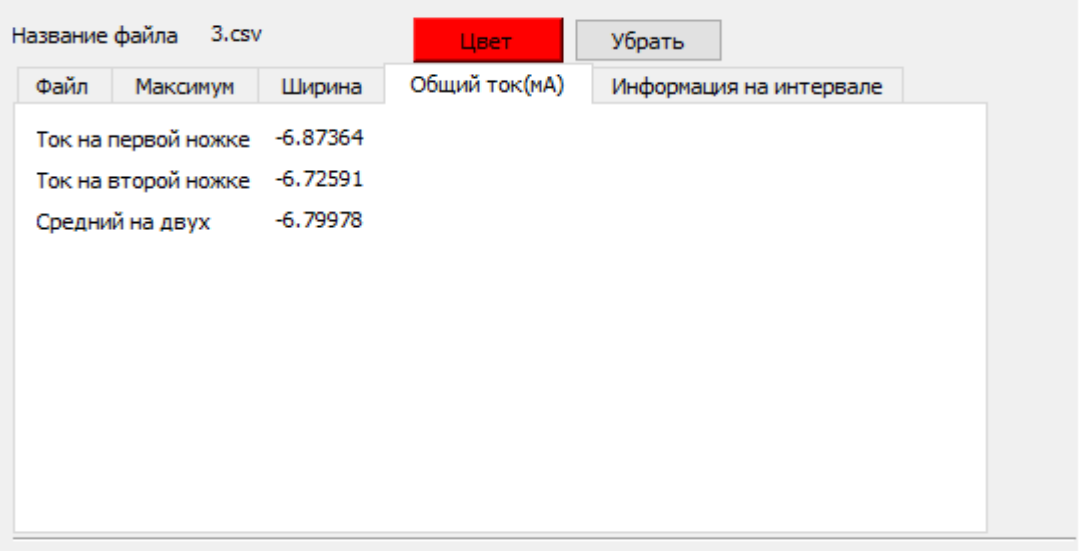

Рисунок 18 – Вкладка общий ток в блоке данных

На вкладке ***Информация на интервале*** отображаются данные в промежутке, который задается как максимум тока, деленный на число, введенное в поле ***Часть***. Пользователь может вводить произвольное число. Если ввести число 2, то интервал будет определен на полувысоте от максимума пучка. По умолчанию этот параметр подбирается таким образом, чтобы значение тока было 95 процентов от общего тока на всем интервале. Здесь отображается ширина в миллиметрах и ток в этом интервале. Форма представлена на рисунке 19. В будущем планируется добавить возможность задавать интервал просто четырьмя значениями угла.

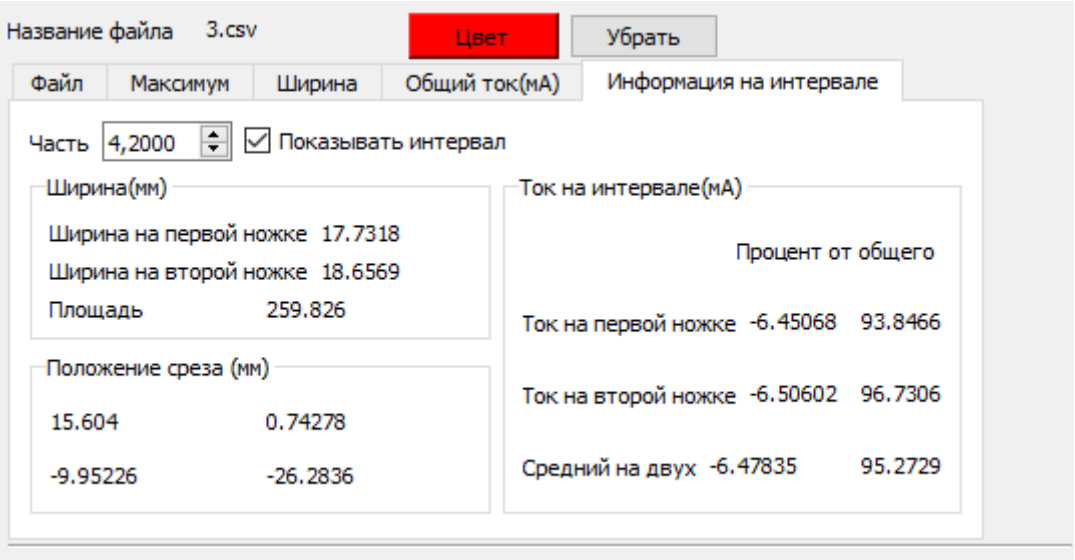

Рисунок 19 – Вкладка ***Информация на интервале***

Кнопка ***Профиль*** во вкладке ***Файл*** отображает двумерный профиль пучка. Окно отображения профиля изображено на рисунке 20.

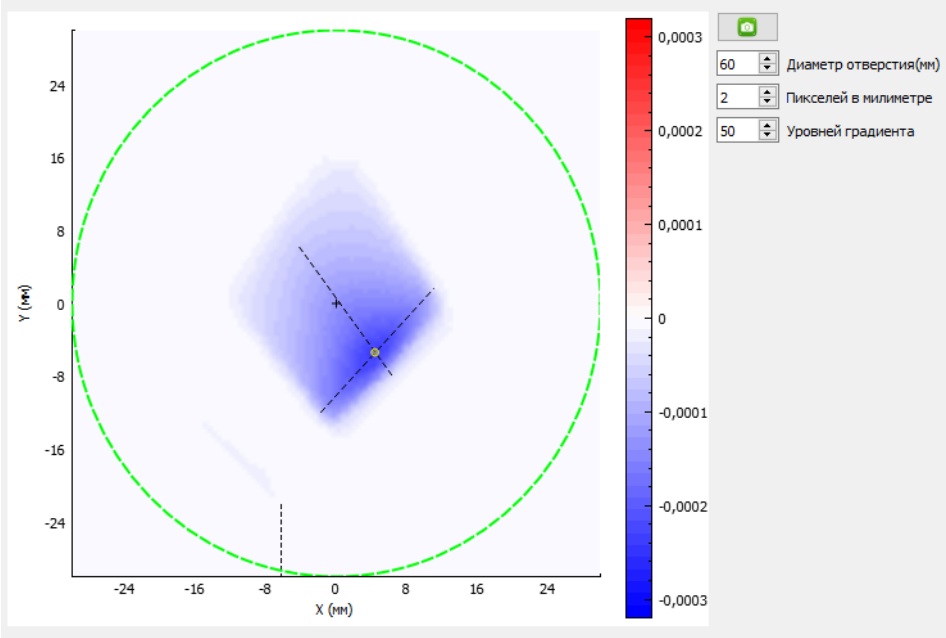

Рисунок 20 – Окно отображения двумерного профиля пучка

Каждый пиксель профиля строится как перемножение токов с двух ножек. При этом положение каждого пикселя вычисляется по алгоритму пересчета углов в координаты. Параметр ***Диаметр отверстия*** — это диаметр отображаемой области, он отображается пунктирной окружностью. Желтый кружок отображает максимум пучка. Пунктирные линии являются шириной на полувысоте. Штриховая линия внизу отображает вертикальное положение основания.

В приложении есть возможность задать параметры профилометра. Это делается нажатием на кнопку настройки в главном окне. На рисунке 21 изображено окно настроек.

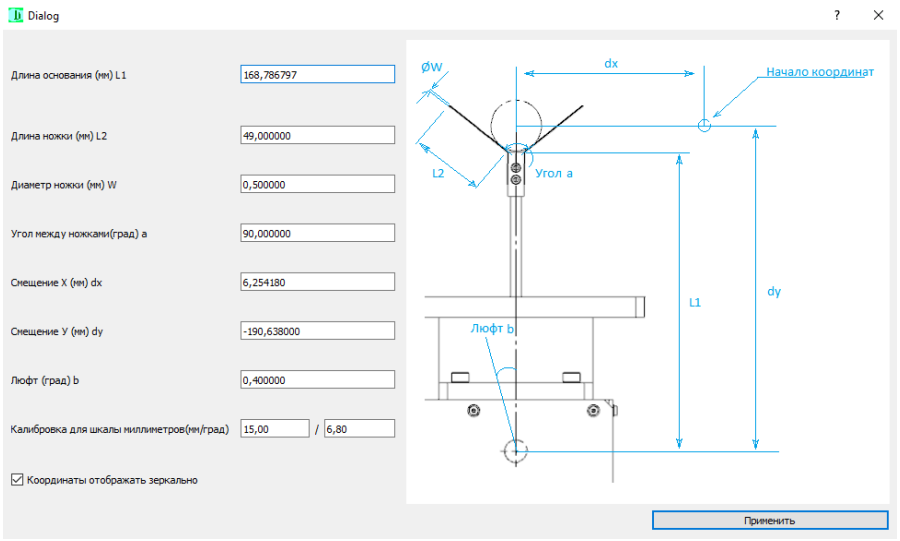

Рисунок 21 – Окно настроек

В этом окне задаются все параметры профилометра, от этих параметров зависят расчеты тока и положения. Есть возможность отображать координаты зеркально. Все эти настройки сохраняются в файл ***settings.ini*** в директории с приложением. На рисунке 22 изображен файл с настройками.

```
[General]
LastPath=D:/users/Tima45/\x41f\x440\x43e\x444\x438\x43b\x43e\x43c\x435\x442\x440/26.07.17
WireLength_mm=168.78679700000001
LegLength_mm=49
LegWidth_mm=0.5
LegsAngle_c=90
DeltaX_mm=6.2541799999999999
DeltaY_mm=-190.63800000000001
LuftAngle_c=0.40000000000000002
ShowMirror=true
mmAxisCalibNumerator_mm=15
mmAxisCalibDenominator_c=6.7999999999999998
SettingsVersion=2.4
```

Рисунок 22 – Содержимое файла с настройками

В этом файле хранится поле ***LastPath***, в котором записывается последняя выбранная папка. И она подгружается при открытии программы. Также указывается номер версии программы, чтобы файл настроек соответствовал версии программы. Если этого файла не будет в папке, то он сгенерируется программой автоматически, и в параметры профилометра будут проставлены значения по умолчанию.

4.2 Программная архитектура приложения

Для представления данных профилометра был разработан класс *ProfileData*. Этот класс содержит данные из одного файла. Также в нем содержатся объекты, отвечающие за отображение графиков, и виджет(визульный объект) отвечающий за отображение вычисляемых данных в графическом интерфейсе. На рисунке 23 изображена диаграмма классов на которой представлены основные классы, связанные с хранением и представлением данных.

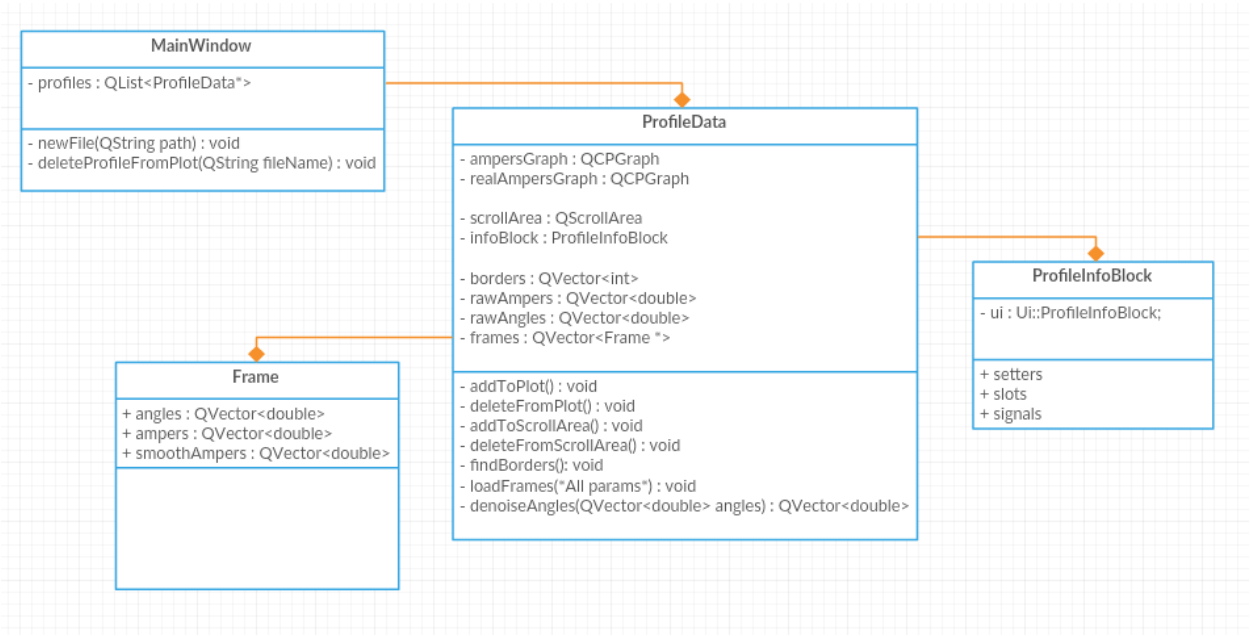

Рисунок 23 – UML диаграмма классов, отвечающих за представление данных

Класс *MainWindow* – это класс основной формы, он содержит список указателей на объекты *ProfileData*. В этом списке хранятся только добавленные на отображения данные, они подгружаются в момент добавления, т.е. когда пользователь решит их отобразить.

В конструкторе объект *ProfileData* разбивает данные на кадры и для каждого кадра создается объект *Frame*, который помещается в массив. Объект *Frame* содержит массивы углов, токов и сглаженных токов. В нем происходит вычисление всех данных о пучке.

В объекте *ProfileData* содержатся объекты управляющие графиками. Также в нем содержится указатель на виджет *ProfileInfoBlock*, в котором отображаются данные. Когда пользователь изменяет положение слайдера,

например, шумоподавления, то незамедлительно выполняется полный цикл обработки данных, включая отображение изменившихся данных на графике и в блоке интерфейса. Таким образом пользователь сразу видит результат своих действий.

4.3 Библиотека для работы с дискретным вейвлет-преобразованием

Для дискретного вейвлет-преобразования использовалась библиотека GSL (GNU Scientific Library), написанная на языке C [6]. Для более удобного использования библиотеки на языке C++ была разработана собственная библиотека, которая предоставляет программный интерфейс для работы с вейвлет-преобразованием.

Библиотека содержит класс *WaveletSpectrum*, который реализует преобразование. В конструкторе объекта передается массив данных произвольной длины, для которых выполняется преобразование, и тип вейвлета. Объект класса содержит публичный двумерный массив вейвлет-спектра. Пользователь кода может напрямую менять коэффициенты в спектре, но для удобства были разработаны методы, которые производят обработку спектра:

- void threshold(double percent) – удаляет коэффициенты из спектра которые меньше в процентах чем параметр *percent*.
- void highFilter(double value) – умножает верхнюю половину спектра на значение *value.*
- void lowFilter(double value) – аналогично предыдущему, только для нижней части спектра.
- void levelFilter(int level,double value) – умножает коэффициенты на заданном уровне на значение *value*.
- void intervalFilter(int startLevel,int stopLevel,double value) – умножает коэффициенты на параметр *value* на указанном интервале.
- QVector<double> toData() – возвращает массив данных после обратного преобразования.

Используя эту библиотеку, алгоритм шумоподавления описывается несколькими строками на языке C++ Qt. Пример кода представлен на рисунке 24.

```
void Frame::denoiseAmpers()
{
    if(!ampers.isEmpty()){
        smoothAmpers.clear();
        WaveletSpectrum *DWT = new WaveletSpectrum(ampers,WaveletSpectrum::BSPLINE_309);
        DWT->threshold(noiseReductionPercent/100.0);
        smoothAmpers = DWT->toData();
        delete DWT;
    }
}
```

Рисунок 24 – Пример использования библиотеки для шумоподавления

4.4 Эксплуатация программы

В процессе использования профилометра возникает большое количество данных, эти данные можно обработать в ручном режиме. Тогда после проведения эксперимента обработка займет от двух до пяти часов. В то время как с разработанным приложением необходимые данные о пучке можно видеть прямо во время эксперимента.

С помощью приложения было исследовано влияние пространственного заряда на пучок. В тракте намеренно ухудшался вакуум, при этом изменялся ток и размер пучка. Подробнее о исследовании пространственного заряда прочитать в статье [7]. Также эта программа использовалась в экспериментах по измерению эмиттанса и по вытягиванию большого тока из ионного источника. Обычно установка работает на токе от 2 до 5 миллиампер. Удалось получить ток в 8 миллиампер отрицательных ионов водорода. В дальнейшем будут попытки получить ток в 10 миллиампер.

ЗАКЛЮЧЕНИЕ

Виды проделанной работы:

1) Было разработано программное обеспечение с графическим интерфейсом пользователя.
2) Была разработана программная библиотека для работы с вейвлет-преобразованием на основе GSL.
3) Была изучена предметная область обработки цифровых сигналов.

В ходе этой работы были разработаны и реализованы следующие методы.

1) Метод расчета интегрального тока пучка.
2) Метод расчета положения максимума пучка.
3) Метод расчета размера пучка.

Разработанное приложение было протестировано и внедрено на действующей установке. Приложение успешно используется для измерения тока и других параметров пучка. К данной работе прилагается акт о внедрении.

Возможно, в дальнейшем разработанные методы и алгоритмы будут применяться для неразрушающей диагностики пучка в установке, предназначенной для клиники.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Таскаев С. Ю. Ускорительный источник эпитепловых нейтронов / С. Ю. Таскаев // Физика элементарных частиц и атомного ядра. — Объединенный институт ядерных исследований, 2015. — Т. 46. — № 6. — С.1770-1830.

2 WS 30 Wire Scanner System user manual [Электронный ресурс]. — URL: https://www.d-pace.com/, свободный. — Яз. англ. — (Дата обращ. 04.09.2017).

3 Suppression of an unwanted flow of charged particles in a tandem accelerator with vacuum insulation. / A. Ivanov, [at al] // JINST 11. — Interational School for Advanced Studies, Institute of Physics, 2016. — No P04018. — P.9.

4 Яковлев А. Н. Введение в вейвлет-преобразования / А. Н Яковлев. // Учеб. пособие. — Новосибирск: Изд-во НГТУ, 2003. — 104 с.

5 Официальный сайт Qt [Электронный ресурс]. — URL: https://www.qt.io/ru/, свободный. — Яз. англ. — (Дата обращ. 04.09.2017).

6 GSL – GNU Scientific Library [Электронный ресурс]. — URL: https://www.gnu.org/software/gsl/, свободный. — Яз. англ. — (Дата обращ. 04.09.2017).

7 Effect produced by a space charge and spherical aberration of lenses on a negative hydrogen ion beam injected into a vacuum-insulated tandem accelerator. / T. Bykov, [at al] // JINST. — Interational School for Advanced Studies, Institute of Physics, 2017. В печати.

ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД

```
#include "mainwindow.h"
#include <QApplication>

        int main(int argc, char *argv[])
        {
            QApplication a(argc, argv);
            QTime time = QTime::currentTime();
            qsrand((uint)time.msec());

            QFont f;
            f.setPixelSize(11);
            a.setFont(f);

            MainWindow w;
            w.show();
            return a.exec();
        }
```

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QDebug>
#include <QFileDialog>
#include <QListWidgetItem>
#include <QSettings>
#include "profiledata.h"
#include <directorymonitor.h>

#include "settingsdialog.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    //---------------------------------
    //Settings
    static QString settingsFileName;
    static QString lastPathFieldName;
    static QString wireLengthFieldName;
    static QString legLengthFieldName;
    static QString legWidthFieldName;
    static QString legsAngleFieldName;
    static QString deltaXFieldName;
    static QString deltaYFieldName;
    static QString luftAngleFieldName;
    static QString showMirrorFieldName;
    static QString angleToRKoefNumeratorFieldName;
    static QString angleToRKoefDenominatorFieldName;
private slots:

    void on_selectFolderBt_clicked();

    void on_profilesList_itemActivated(QListWidgetItem *item);

    void on_pathEdit_editingFinished();

    void deleteProfileFromPlot(QString fileName);

    void newFile(QString path);

    void watcher(int,int);
    void on_pushButton_clicked();
    void rescaleAxis(QCPRange r);

    void on_settingsButton_clicked();
```

```
    void reloadSettings(double _wireLength, double _legLength, double _legWidth, double
_legAngle, double _deltaX, double _deltaY, double _luftAngle, bool _showMirror, double
_angleToRKoefNumerator, double _angleToRKoefDenominator);
private:
    bool fromMonitor = false;
    Ui::MainWindow *ui;

    void loadingSettings();
    //---------------------------------
    //global info
    double wireLength = 169;
    double legLength = 50;
    double legWidth = 0.5;
    double deltaX = -190;
    double deltaY = 0;
    double luftAngle = 0.4;
    double angleToRKoefNumerator = 15;
    double angleToRKoefDenominator = 6.8;
    double angleToRKoef = angleToRKoefNumerator/angleToRKoefDenominator;
    double legsAngle = 103.6;
    bool showMirror = true;
    //---------------------------------
    void scanProfiles(QString folderPath);
    void addProfileToPlot(QString fileName);
    QList<ProfileData*> profiles;
    void initPlot();
    DirectoryMonitor *monitor = nullptr;
};

#endif // MAINWINDOW_H
```

```
        #include "mainwindow.h"
        #include "ui_mainwindow.h"


        QString MainWindow::settingsFileName = "settings.ini";
        QString MainWindow::lastPathFieldName = "LastPath";
        QString MainWindow::wireLengthFieldName = "WireLength_mm";
        QString MainWindow::legLengthFieldName = "LegLength_mm";
        QString MainWindow::legWidthFieldName = "LegWidth_mm";
        QString MainWindow::legsAngleFieldName = "LegsAngle_c";
        QString MainWindow::deltaXFieldName = "DeltaX_mm";
        QString MainWindow::deltaYFieldName = "DeltaY_mm";
        QString MainWindow::luftAngleFieldName = "LuftAngle_c";
        QString MainWindow::showMirrorFieldName = "ShowMirror";
        QString MainWindow::angleToRKoefNumeratorFieldName = "mmAxisCalibNumerator_mm";
        QString MainWindow::angleToRKoefDenominatorFieldName = "mmAxisCalibDenominator_c";


        //test
        MainWindow::MainWindow(QWidget *parent) :
            QMainWindow(parent),
            ui(new Ui::MainWindow)
        {
            ui->setupUi(this);
            QWidget *widget = new QWidget(ui->scrollArea);

            QBoxLayout *lay = new QBoxLayout(QBoxLayout::TopToBottom,widget);

            widget->setLayout(lay);

            ui->scrollArea->setWidget(widget);

            connect(ui->scrollArea-
>verticalScrollBar(),SIGNAL(rangeChanged(int,int)),this,SLOT(watcher(int,int)));
            loadingSettings();
            initPlot();
        }

        MainWindow::~MainWindow()
        {
            QSettings settings(settingsFileName,QSettings::IniFormat);
            settings.setValue(lastPathFieldName,ui->pathEdit->text());

            ui->profilesList->clear();
            for(int i = 0; i < profiles.count(); i++){
                profiles.at(i)->deleteLater();
            }
            profiles.clear();
```

```
            delete ui;
        }
        void MainWindow::on_selectFolderBt_clicked()
        {
            QString   folderPath   =   QFileDialog::getExistingDirectory(this,"Выбрать   папку",ui-
>pathEdit->text(),0);
            ui->pathEdit->setText(folderPath);
            scanProfiles(folderPath);
        }

        void MainWindow::scanProfiles(QString folderPath)
        {

            ui->profilesList->clear();
            for(int i = 0; i < profiles.count(); i++){
                profiles.at(i)->deleteLater();
            }
            profiles.clear();
            if(monitor != nullptr){
                monitor->stop();
                delete monitor;
            }
            monitor = new DirectoryMonitor(folderPath,1000,".csv",false);
            connect(monitor,SIGNAL(fileChanged(QString)),this,SLOT(newFile(QString)));
            monitor->start();

            QDir dir(folderPath);
            QFileInfoList fileInfoList = dir.entryInfoList();
            for(int i = 0; i < fileInfoList.count(); i++){
                if(fileInfoList.at(i).fileName().endsWith(".csv")){
                    ui->profilesList->addItem(fileInfoList.at(i).fileName());
                }
            }
        }

        void MainWindow::addProfileToPlot(QString fileName)
        {
            bool find = false;
            for(int i = 0; i < profiles.count(); i++){
                if(profiles.at(i)->fileName == fileName){
                    find = true;
                    break;
                }
            }
            if(!find){
                QColor randomColor(qrand()%256,qrand()%256,qrand()%256);
                ProfileData      *newProfile      =      new      ProfileData(fileName,ui->plot,ui-
>scrollArea,randomColor,wireLength,legLength,legWidth,legsAngle,deltaX,deltaY,luftAngle,showMirro
r,angleToRKoef,0);
                con-
nect(newProfile,SIGNAL(wantToBeDeleted(QString)),this,SLOT(deleteProfileFromPlot(QString)));
                profiles.append(newProfile);
                ui->plot->rescaleAxes();
                ui->plot->replot();
            }
        }

        void MainWindow::initPlot()
        {
            ui->plot->setInteraction(QCP::iRangeDrag, true);
            ui->plot->setInteraction(QCP::iRangeZoom, true);
            ui->plot->axisRect()->setRangeDrag(Qt::Horizontal);
            ui->plot->axisRect()->setRangeZoom(Qt::Horizontal);
            ui->plot->xAxis->setLabel("Угол");
            ui->plot->yAxis->setLabel("Ток(А)");
            ui->plot->xAxis2->setLabel("Мм");
            ui->plot->addLayer("down");
            ui->plot->addLayer("up");
            ui->plot->xAxis2->setVisible(true);
            ui->plot->xAxis2->setRange(QCPRange(-angleToRKoef*13.0,angleToRKoef*13.0));
            connect(ui->plot-
>xAxis,SIGNAL(rangeChanged(QCPRange)),this,SLOT(rescaleAxis(QCPRange)));
        }

        void MainWindow::on_profilesList_itemActivated(QListWidgetItem *item)
        {
            addProfileToPlot(ui->pathEdit->text()+"/"+item->text());
        }

        void MainWindow::on_pathEdit_editingFinished()
```

```
{
    scanProfiles(ui->pathEdit->text());
}

void MainWindow::deleteProfileFromPlot(QString fileName)
{
    int idToDelte = -1;
    for(int i = 0; i < profiles.count(); i++){
        if(profiles.at(i)->fileName == fileName){
            idToDelte = i;
        }
    }
    if(idToDelte != -1){
        //---
        profiles.at(idToDelte)->deleteLater();
        profiles.removeAt(idToDelte);
    }
}

void MainWindow::newFile(QString path)
{
    ui->profilesList->clear();
    QDir dir(ui->pathEdit->text());
    QFileInfoList fileInfoList = dir.entryInfoList();
    for(int i = 0; i < fileInfoList.count(); i++){
        if(fileInfoList.at(i).fileName().endsWith(".csv")){
            ui->profilesList->addItem(fileInfoList.at(i).fileName());
        }
    }
    addProfileToPlot(path);
    fromMonitor = true;
}

void MainWindow::watcher(int min, int max)
{
    if(fromMonitor){
        fromMonitor = false;
        ui->scrollArea->verticalScrollBar()->setValue(max);
        Q_UNUSED(min);
    }
}

void MainWindow::loadingSettings()
{
    if(QFileInfo(settingsFileName).isFile()){
        QSettings settings(settingsFileName,QSettings::IniFormat);
        ui->pathEdit->setText(settings.value(lastPathFieldName).toString());
        wireLength = settings.value(wireLengthFieldName).toDouble();
        legLength = settings.value(legLengthFieldName).toDouble();
        legWidth = settings.value(legWidthFieldName).toDouble();
        deltaX = settings.value(deltaXFieldName).toDouble();
        deltaY = settings.value(deltaYFieldName).toDouble();
        luftAngle = settings.value(luftAngleFieldName).toDouble();
        showMirror = settings.value(showMirrorFieldName).toBool();
        legsAngle = settings.value(legsAngleFieldName).toDouble();
        angleToRKoefNumerator = settings.value(angleToRKoefNumeratorFieldName).toDouble();
        angleToRKoefDenominator                                          =                  set-
tings.value(angleToRKoefDenominatorFieldName).toDouble();
        angleToRKoef = angleToRKoefNumerator/angleToRKoefDenominator;
    }else{
        QMessageBox::warning(this,"Внимание","Файл    настрок    не    найден.    Значения
дефолтные.");
        QSettings settings(settingsFileName,QSettings::IniFormat);
        settings.setValue(lastPathFieldName,"");
        settings.setValue(wireLengthFieldName,wireLength);
        settings.setValue(legLengthFieldName,legLength);
        settings.setValue(legWidthFieldName,legWidth);
        settings.setValue(deltaXFieldName,deltaX);
        settings.setValue(deltaYFieldName,deltaY);
        settings.setValue(luftAngleFieldName,luftAngle);
        settings.setValue(showMirrorFieldName,showMirror);
        settings.setValue(legsAngleFieldName,legsAngle);
        settings.setValue(angleToRKoefNumeratorFieldName,angleToRKoefNumerator);
        settings.setValue(angleToRKoefDenominatorFieldName,angleToRKoefDenominator);
    }
}

void MainWindow::on_pushButton_clicked()
{
```

```
            QString s = QFileDialog::getSaveFileName(this,"Сохранить скриншот",ui->pathEdit->text(),"png");
            ui->plot->grab().save(s+".png","png");
        }

        void MainWindow::rescaleAxis(QCPRange r)
        {
            ui->plot->xAxis2->setRange(QCPRange(angleToRKoef * r.lower,angleToRKoef * r.upper));
        }

        void MainWindow::on_settingsButton_clicked()
        {
            SettingsDialog *dialog = new SettingsDialog(wireLength,legLength,legWidth,legsAngle,deltaX,deltaY,luftAngle,showMirror,angleToRKoefNumerator,angleToRKoefDenominator,this);
            connect(dialog,SIGNAL(settingsUpdate(double,double,double,double,double,double,double,bool,double,double)),this,SLOT(reloadSettings(double,double,double,double,double,double,double,bool,double,double)));
            dialog->exec();
        }

        void MainWindow::reloadSettings(double _wireLength,double _legLength,double _legWidth,double _legAngle, double _deltaX, double _deltaY, double _luftAngle, bool _showMirror,double _angleToRKoefNumerator, double _angleToRKoefDenominator)
        {
            ui->profilesList->clear();
            for(int i = 0; i < profiles.count(); i++){
                profiles.at(i)->deleteLater();
            }
            profiles.clear();
            if(monitor != nullptr){
                monitor->stop();
                delete monitor;
                monitor = nullptr;
            }

            wireLength = _wireLength;
            legLength = _legLength;
            legWidth = _legWidth;
            legsAngle = _legAngle;
            deltaX = _deltaX;
            deltaY = _deltaY;
            luftAngle = _luftAngle;
            showMirror = _showMirror;
            angleToRKoefNumerator = _angleToRKoefNumerator;
            angleToRKoefDenominator = _angleToRKoefDenominator;
            angleToRKoef = angleToRKoefNumerator/angleToRKoefDenominator;
        }
```

```
#ifndef PROFILEDATA_H
#define PROFILEDATA_H

#include <QObject>
#include <QtCore>
#include <QMessageBox>
#include <QRegExp>
#include <waveletspectrum.h>
#include "plot/qcustomplot.h"
#include "frame.h"
#include "profileinfoblock.h"
#include "profilewindow.h"

class ProfileData : public QObject
{
    Q_OBJECT
public:
    ProfileData(QObject *parent = 0);
    ProfileData(QString inFileName, QCustomPlot *inPlot, QScrollArea *inScrollArea, QColor inColor, double wireLength, double legLength,double legWidth,double legsAngle, double deltaX, double deltaY, double luftAngle,bool showMirror,double inAngleToMMKoef, QObject *parent);
    ~ProfileData();
    QString fileName;
    QColor color;
signals:
    void wantToBeDeleted(QString fileName);
public slots:
    void deletePressed();
    void showFrame(int id);
```

```
    void changePart(double value);
    void showMaxCheck(bool);
    void showIntervalCheck(bool);
    void showMiddleCheck(bool);
    void redrawAll(QColor);
    void showProfileWindow();
    void unseeit();
    void changeLineSize(int value);
    void changeLineType(Qt::PenStyle);
    void denoiseCurrentFrame(int percent);
private:
    QCustomPlot *plot;
    QCPGraph *ampersGraph;
    QCPGraph *realAmpersGraph;
    QPen mainPen;

    QCPItemLine *maxCurrentLeg1Line;
    QCPItemLine *maxCurrentLeg2Line;
    QCPItemLine *middleLineLeg1;
    QCPItemLine *middleLineLeg2;
    QCPGraph *massZoneLeg1;
    QCPGraph *massZoneLeg2;
    QCPItemLine *massMiddleLeg1;
    QCPItemLine *massMiddleLeg2;
    QCPItemTracer *massTracerLeg1;
    QCPItemTracer *massTracerLeg2;
    void addToPlot();
    void deleteFromPlot();
    //-------------------------------------
    QScrollArea *scrollArea;
    ProfileInfoBlock *infoBlock;
    bool profileWindowIsShown = false;
    void addToScrollArea();
    void deleteFromScrollArea();
    //-------------------------------------
    QVector<int> borders;
    QVector<double> rawAmpers;
    QVector<double> rawAngles;
    QVector<Frame *> frames;
    int currentFrameId = -1;
    QString scanDate;

    void readData(QString fileName);
    void findBorders();
    void loadFrames(double wireLength, double legLength, double legWidth, double deltaX, double
deltaY, double luftAngle, double legsAngle, bool showMirror);
    QVector<double> denoiseAngles(QVector<double> angles);

    double angleToMMKoef;

};

#endif // PROFILEDATA_H
```

```
        #include "profiledata.h"


        ProfileData::ProfileData(QObject *parent) : QObject(parent)
        {

        }

        ProfileData::ProfileData(QString    inFileName,    QCustomPlot    *inPlot,    QScrollArea
*inScrollArea, QColor inColor, double wireLength, double legLength, double legWidth, double leg-
sAngle, double deltaX, double deltaY, double luftAngle, bool showMirror, double inAngleToMMKoef,
QObject *parent) : QObject(parent)
        {
            fileName = inFileName;
            plot = inPlot;
            scrollArea = inScrollArea;
            color = inColor;
            angleToMMKoef = inAngleToMMKoef;

            readData(fileName);
            findBorders();
            load-
Frames(wireLength,legLength,legWidth,deltaX,deltaY,luftAngle,legsAngle,showMirror);

            addToPlot();
```

```
            addToScrollArea();
            showFrame(0);
        }

        ProfileData::~ProfileData()
        {
            deleteFromPlot();
            deleteFromScrollArea();
        }

        void ProfileData::deletePressed()
        {
            emit wantToBeDeleted(this->fileName);
        }

        void ProfileData::showFrame(int id)
        {
            if(!frames.isEmpty() && id >= 0 && id < frames.count()){
                /*
                QCPRange xR = plot->xAxis->range();
                QCPRange yR = plot->yAxis->range();
        */
                currentFrameId = id;
                Frame* currentFrame = frames.at(currentFrameId);
                //--------------
                ampersGraph->clearData();
                ampersGraph->setData(currentFrame->angles,currentFrame->smoothAmpers);

                realAmpersGraph->clearData();
                realAmpersGraph->setData(currentFrame->angles,currentFrame->ampers);
                //----------------------------
                QVector<double> x1,y1,x2,y2;
                x1.append(currentFrame->massLeg1AngleStart);
                x1.append(currentFrame->massLeg1AngleStop);
                x2.append(currentFrame->massLeg2AngleStart);
                x2.append(currentFrame->massLeg2AngleStop);
                y1.append(0);
                y1.append(0);
                y2.append(0);
                y2.append(0);
                massZoneLeg1->setData(x1,y1);
                massZoneLeg2->setData(x2,y2);
                //----------------------------
                maxCurrentLeg1Line->start->setCoords(currentFrame->maxCurrentAngle1,0);
                maxCurrentLeg1Line->end->setCoords(currentFrame->maxCurrentAngle1,currentFrame-
>maxAmperLeg1);
                //--------------------------
                maxCurrentLeg2Line->start->setCoords(currentFrame->maxCurrentAngle2,0);
                maxCurrentLeg2Line->end->setCoords(currentFrame->maxCurrentAngle2,currentFrame-
>maxAmperLeg2);
                //-----------------------------
                middleLineLeg1->start->setCoords(currentFrame->Leg1AngleStart,currentFrame-
>maxAmperLeg1/2.0);
                middleLineLeg1->end->setCoords(currentFrame->Leg1AngleStop,currentFrame-
>maxAmperLeg1/2.0);
                //-----------------------------
                middleLineLeg2->start->setCoords(currentFrame->Leg2AngleStart,currentFrame-
>maxAmperLeg2/2.0);
                middleLineLeg2->end->setCoords(currentFrame->Leg2AngleStop,currentFrame-
>maxAmperLeg2/2.0);
                //------------------------
                massMiddleLeg1->start->setCoords(currentFrame->massLeg1AngleStart,currentFrame-
>massMiddleAmperLeg1);
                massMiddleLeg1->end->setCoords(currentFrame->massLeg1AngleStop,currentFrame-
>massMiddleAmperLeg1);

                massTracerLeg1->position->setCoords(                                (currentFrame-
>massLeg1AngleStart+currentFrame->massLeg1AngleStop)/2.0,currentFrame->massMiddleAmperLeg1);

                massMiddleLeg2->start->setCoords(currentFrame->massLeg2AngleStart,currentFrame-
>massMiddleAmperLeg2);
                massMiddleLeg2->end->setCoords(currentFrame->massLeg2AngleStop,currentFrame-
>massMiddleAmperLeg2);

                massTracerLeg2->position->setCoords(                                (currentFrame-
>massLeg2AngleStart+currentFrame->massLeg2AngleStop)/2.0,currentFrame->massMiddleAmperLeg2);
                //------------------------
                /*
                plot->xAxis->setRange(xR);
                plot->yAxis->setRange(yR);
```

```
                */
                plot->replot();
                //=======================================
                QStringList list = fileName.split("/");
                QString name = list.last();
                infoBlock->setFrame(currentFrameId);
                infoBlock->setFileName(name);
                infoBlock->setScanDate(scanDate);
                infoBlock->setMaxValueData(currentFrame->maxCurrentAngle1,currentFrame-
>maxCurrentAngle2,currentFrame->maxCurrentX,currentFrame->maxCurrentY,currentFrame-
>maxAmperLeg1,currentFrame->maxAmperLeg2);
                infoBlock->setWidth(currentFrame->widthLeg1,currentFrame->widthLeg2);
                infoBlock->setCurrent(currentFrame->currentLeg1,currentFrame-
>currentLeg2,currentFrame->current);
                infoBlock->setMassXY(currentFrame->massX,currentFrame->massY);
                infoBlock->setMassWidth(currentFrame->massWidthLeg1,currentFrame->massWidthLeg2);
                infoBlock->setMassCurrent(currentFrame->currentLeg1Mass,currentFrame-
>currentLeg2Mass,currentFrame->currentMass,currentFrame->difCurrentLeg1,currentFrame-
>difCurrentLeg2,currentFrame->difCurrent);
                infoBlock->setPart(currentFrame->part);
                infoBlock->setSection(currentFrame->massLeg1AngleStart,currentFrame-
>massLeg1AngleStop,currentFrame->massLeg2AngleStart,currentFrame-
>massLeg2AngleStop,angleToMMKoef);
            }
        }

        void ProfileData::changePart(double value)
        {
            for(int i = 0; i < frames.count();i++){
                frames.at(i)->findMass(value);
            }
            showFrame(currentFrameId);
        }

        void ProfileData::showMaxCheck(bool on)
        {
            maxCurrentLeg1Line->setVisible(on);
            maxCurrentLeg2Line->setVisible(on);
            plot->replot();
        }

        void ProfileData::showIntervalCheck(bool on)
        {
            massZoneLeg1->setVisible(on);
            massZoneLeg2->setVisible(on);
            massMiddleLeg1->setVisible(on);
            massMiddleLeg2->setVisible(on);
            massTracerLeg1->setVisible(on);
            massTracerLeg2->setVisible(on);
            plot->replot();
        }


        void ProfileData::showMiddleCheck(bool on)
        {
            middleLineLeg1->setVisible(on);
            middleLineLeg2->setVisible(on);
            plot->replot();
        }

        void ProfileData::redrawAll(QColor newColor)
        {
            color = newColor;

            mainPen.setColor(color);
            QPen linePen(color);
            linePen.setStyle(Qt::PenStyle::DashLine);
            QPen normalPen(color);
            QColor brushcolor(color);

            brushcolor.setAlpha(100);
            QBrush mainBrush(brushcolor);
            mainBrush.setStyle(Qt::Dense4Pattern);
            ampersGraph->setPen(mainPen);


            QColor c(color);
            c.setAlpha(50);
            QPen shadowPen(c);
            shadowPen.setStyle(Qt::DashLine);
```

```
            realAmpersGraph->setPen(shadowPen);

            massZoneLeg1->setBrush(mainBrush);
            massZoneLeg2->setBrush(mainBrush);
            maxCurrentLeg1Line->setPen(linePen);
            maxCurrentLeg2Line->setPen(linePen);
            middleLineLeg1->setPen(linePen);
            middleLineLeg2->setPen(linePen);
            massMiddleLeg1->setPen(normalPen);
            massMiddleLeg2->setPen(normalPen);
            massTracerLeg1->setPen(normalPen);
            massTracerLeg2->setPen(normalPen);
            plot->replot();
        }

        void ProfileData::showProfileWindow()
        {
            if(!profileWindowIsShown && currentFrameId != -1 && !frames.isEmpty()){
                profileWindowIsShown = true;
                ProfileWindow *p = new ProfileWindow(frames.at(currentFrameId)->angles,
                                                     frames.at(currentFrameId)->ampers,
                                                     frames.at(currentFrameId)->wireLength,
                                                     frames.at(currentFrameId)-
>legLength,frames.at(currentFrameId)->legsAngle,
                                                     frames.at(currentFrameId)->zeroAngleIndex,
                                                     frames.at(currentFrameId)->showMirror,
                                                     frames.at(currentFrameId)->deltaX,
                                                     frames.at(currentFrameId)->deltaY,
                                                     frames.at(currentFrameId)->maxCurrentX,
                                                     frames.at(currentFrameId)->maxCurrentY,
                                                     frames.at(currentFrameId)-
>massLineLeg1StartX,
                                                     frames.at(currentFrameId)-
>massLineLeg1StartY,
                                                     frames.at(currentFrameId)->massLineLeg1StopX,
                                                     frames.at(currentFrameId)->massLineLeg1StopY,
                                                     frames.at(currentFrameId)-
>massLineLeg2StartX,
                                                     frames.at(currentFrameId)-
>massLineLeg2StartY,
                                                     frames.at(currentFrameId)->massLineLeg2StopX,
                                                     frames.at(currentFrameId)->massLineLeg2StopY,
                                                     scrollArea);
                p->setWindowTitle(fileName);
                p->setAttribute(Qt::WA_DeleteOnClose);
                p->show();
                connect(p,SIGNAL(destroyed(QObject*)),this,SLOT(unseeit()));
            }
        }

        void ProfileData::unseeit()
        {
            profileWindowIsShown = false;
        }

        void ProfileData::changeLineSize(int value)
        {
            mainPen.setWidth(value);
            ampersGraph->setPen(mainPen);
            plot->replot();
        }

        void ProfileData::changeLineType(Qt::PenStyle type)
        {
            mainPen.setStyle(type);
            ampersGraph->setPen(mainPen);
            plot->replot();
        }

        void ProfileData::denoiseCurrentFrame(int percent)
        {
            if(currentFrameId != -1){
                frames.at(currentFrameId)->setNoiseReductionPercent(percent);
                showFrame(currentFrameId);
            }
        }


        void ProfileData::readData(QString fileName)
```

```
        {
            if(!QFileInfo(fileName).isFile()){
                QMessageBox::critical(scrollArea,"Ошибка",fileName+"\n Не файл.");
                return;
            }
            if(!fileName.endsWith(".csv")){
                QMessageBox::critical(scrollArea,"Ошибка","Не верное расширение\n Необходимый фор-
мат .csv");
                return;
            }
            QFile f(fileName);
            if(!f.open(QIODevice::ReadOnly | QIODevice::Text)){
                QMessageBox::critical(scrollArea,"Ошибка","Файл не открылся.");
                return;
            }
            //-------------------------------------------------------------------------------------
---
            rawAmpers.clear();
            rawAngles.clear();
            //-------------------------------------------------------------------------------------
---
            bool findData = false;
            QRegExp dateTampl("Scan Start Time*");
            dateTampl.setPatternSyntax(QRegExp::Wildcard);
            QRegExp dataStartTempl("Scan Angle (degrees),Current (A)*");
            dataStartTempl.setPatternSyntax(QRegExp::Wildcard);
            if(!f.atEnd()){
                while(!findData || !f.atEnd()){
                    QString rawLine = f.readLine();
                    if(dateTampl.exactMatch(rawLine)){
                        rawLine.remove(0,17);
                        rawLine.chop(2);
                        scanDate = rawLine;
                    }
                    if(dataStartTempl.exactMatch(rawLine)){
                        findData = true;
                        break;
                    }
                }
                //-------------------------------------------------------------------------------
-------
                if(findData){
                    while(!f.atEnd()){
                        QString rawLine = f.readLine();
                        QStringList slpittedLine = rawLine.split(",");
                        if(slpittedLine.count() == 2){
                            //-----------------------------------------------------------------
                            if(slpittedLine[0] != "NaN"){
                                bool good = false;
                                rawAngles.append(slpittedLine[0].toDouble(&good));
                                if(!good){
                                    f.close();
                                    QMessageBox::critical(scrollArea,"Ошибка","Файл поврежден.");
                                    return;
                                }
                            }else{
                                if(!rawAngles.isEmpty()){
                                    rawAngles.append(rawAngles.last());
                                }else{
                                    rawAngles.append(-14);
                                }
                            }
                            //-----------------------------------------------------------------
                            bool good = false;

                            rawAmpers.append(slpittedLine[1].toDouble(&good));
                            if(!good){
                                f.close();
                                QMessageBox::critical(scrollArea,"Ошибка","Файл поврежден.");
                                return;
                            }
                        } else{
                            f.close();
                            QMessageBox::critical(scrollArea,"Ошибка","Файл поврежден.");
                            return;
                        }
                    }
                }else{
                    QMessageBox::critical(scrollArea,"Ошибка","Данные не найдены.");
                }
```

```
            }
            f.close();

        }

        void ProfileData::findBorders()
        {
            borders.clear();
            if(!rawAngles.isEmpty()){
                QVector<double> smoothAngles = denoiseAngles(rawAngles);
                if(!smoothAngles.isEmpty()){
                    for(int i = 1; i < smoothAngles.count()-1; i++){
                        if( (smoothAngles.at(i-1) > smoothAngles.at(i) && smoothAngles.at(i+1) >
smoothAngles.at(i)) || (smoothAngles.at(i-1) < smoothAngles.at(i) && smoothAngles.at(i+1) <
smoothAngles.at(i))){
                            if(!borders.isEmpty()){
                                if(rawAngles.at(borders.last())*rawAngles.at(i) < 0){
                                    borders.append(i);
                                }
                            }else{
                                borders.append(i);
                            }
                        }
                    }
                }
            }
        }

        void ProfileData::loadFrames(double wireLength,double legLength,double legWidth,double
deltaX,double deltaY,double luftAngle,double legsAngle,bool showMirror)
        {
            if(!rawAngles.isEmpty() && !rawAmpers.isEmpty() && borders.count() > 1){
                for(int i = 0; i < frames.count(); i++){
                    frames.at(i)->deleteLater();
                }
                frames.clear();
                for(int i = 0; i < borders.count()-1; i++){
                    Frame::Direction dir;
                    if(rawAngles.at(borders.at(i)) > rawAngles.at(borders.at(i+1)))
                        dir = Frame::Direction::goDown;
                    else
                        dir = Frame::Direction::goUp;
                    Frame                    *newFrame                    =                    new
Frame(&rawAngles,&rawAmpers,dir,borders.at(i),borders.at(i+1),wireLength,legLength,legWidth,legsA
ngle,deltaX,deltaY,luftAngle,showMirror,this);
                    frames.append(newFrame);
                }
            }
        }

        QVector<double> ProfileData::denoiseAngles(QVector<double> angles)
        {
            WaveletSpectrum dwt(angles,WaveletSpectrum::BSPLINE_309);
            dwt.highFilter(0);
            return dwt.toData();
        }

        void ProfileData:: addToPlot()
        {
            mainPen.setColor(color);
            mainPen.setWidth(2);

            QPen linePen(color);
            linePen.setStyle(Qt::PenStyle::DashLine);
            QPen normalPen(color);
            QColor brushcolor(color);


            QColor c(color);
            c.setAlpha(50);
            QPen shadowPen(c);
            shadowPen.setStyle(Qt::DashLine);

            realAmpersGraph = plot->addGraph();
            realAmpersGraph->setPen(shadowPen);
            realAmpersGraph->setLayer("up");


            brushcolor.setAlpha(100);
            QBrush mainBrush(brushcolor);
```

```
    mainBrush.setStyle(Qt::Dense4Pattern);
    ampersGraph = plot->addGraph();
    ampersGraph->setPen(mainPen);
    ampersGraph->setLayer("up");



    QColor noColor;
    noColor.setAlpha(0);
    QPen noPen(noColor);
    massZoneLeg1 = plot->addGraph();
    massZoneLeg1->setPen(noPen);
    massZoneLeg1->setChannelFillGraph(ampersGraph);
    massZoneLeg1->setLayer("down");
    massZoneLeg1->setBrush(mainBrush);

    massZoneLeg2 = plot->addGraph();
    massZoneLeg2->setPen(noPen);
    massZoneLeg2->setLayer("down");
    massZoneLeg2->setChannelFillGraph(ampersGraph);
    massZoneLeg2->setBrush(mainBrush);

    massTracerLeg1 = new QCPItemTracer(plot);
    massTracerLeg1->setPen(normalPen);
    massTracerLeg1->setStyle(QCPItemTracer::tsCircle);
    plot->addItem(massTracerLeg1);

    massTracerLeg2 = new QCPItemTracer(plot);
    massTracerLeg2->setPen(normalPen);
    massTracerLeg2->setStyle(QCPItemTracer::tsCircle);
    plot->addItem(massTracerLeg2);

    //----------------------------
    maxCurrentLeg1Line = new QCPItemLine(plot);
    maxCurrentLeg1Line->setLayer("up");
    maxCurrentLeg1Line->setPen(linePen);
    plot->addItem(maxCurrentLeg1Line);
    //----------------------------
    maxCurrentLeg2Line = new QCPItemLine(plot);
    maxCurrentLeg2Line->setLayer("up");
    maxCurrentLeg2Line->setPen(linePen);
    plot->addItem(maxCurrentLeg2Line);
    //----------------------------
    middleLineLeg1 = new QCPItemLine(plot);
    middleLineLeg1->setLayer("up");
    middleLineLeg1->setPen(linePen);
    plot->addItem(middleLineLeg1);
    //---------------------------------
    middleLineLeg2 = new QCPItemLine(plot);
    middleLineLeg2->setLayer("up");
    middleLineLeg2->setPen(linePen);
    plot->addItem(middleLineLeg2);
    //---------------------------------
    massMiddleLeg1 = new QCPItemLine(plot);
    massMiddleLeg1->setLayer("up");
    massMiddleLeg1->setPen(normalPen);
    plot->addItem(massMiddleLeg1);
    //---------------------------
    massMiddleLeg2 = new QCPItemLine(plot);
    massMiddleLeg2->setLayer("up");
    massMiddleLeg2->setPen(normalPen);
    plot->addItem(massMiddleLeg2);


    middleLineLeg1->setVisible(false);
    middleLineLeg2->setVisible(false);
    maxCurrentLeg1Line->setVisible(false);
    maxCurrentLeg2Line->setVisible(false);
    massZoneLeg1->setVisible(false);
    massZoneLeg2->setVisible(false);
    massMiddleLeg1->setVisible(false);
    massMiddleLeg2->setVisible(false);
    massTracerLeg1->setVisible(false);
    massTracerLeg2->setVisible(false);

}

void ProfileData::deleteFromPlot()
{
    plot->removeGraph(ampersGraph);
```

```
            plot->removeGraph(massZoneLeg1);
            plot->removeGraph(massZoneLeg2);
            plot->removeItem(maxCurrentLeg1Line);
            plot->removeItem(maxCurrentLeg2Line);
            plot->removeItem(middleLineLeg1);
            plot->removeItem(middleLineLeg2);
            plot->removeItem(massMiddleLeg1);
            plot->removeItem(massMiddleLeg2);
            plot->removeItem(massTracerLeg1);
            plot->removeItem(massTracerLeg2);
            plot->removeGraph(realAmpersGraph);
            plot->replot();
        }

        void ProfileData::addToScrollArea()
        {
            infoBlock = new ProfileInfoBlock(0);



            connect(infoBlock,SIGNAL(removeProfile()),this,SLOT(deletePressed()));
            connect(infoBlock,SIGNAL(showFrame(int)),this,SLOT(showFrame(int)));
            connect(infoBlock,SIGNAL(changePart(double)),this,SLOT(changePart(double)));

            connect(infoBlock,SIGNAL(showMaxCheck(bool)),this,SLOT(showMaxCheck(bool)));
            connect(infoBlock,SIGNAL(showIntervalCheck(bool)),this,SLOT(showIntervalCheck(bool)));
            connect(infoBlock,SIGNAL(showMiddleCheck(bool)),this,SLOT(showMiddleCheck(bool)));
            connect(infoBlock,SIGNAL(redrawAll(QColor)),this,SLOT(redrawAll(QColor)));
            connect(infoBlock,SIGNAL(showProfileWindow()),this,SLOT(showProfileWindow()));
            connect(infoBlock,SIGNAL(changeLineSize(int)),this,SLOT(changeLineSize(int)));
            con-
nect(infoBlock,SIGNAL(changeLineType(Qt::PenStyle)),this,SLOT(changeLineType(Qt::PenStyle)));
            con-
nect(infoBlock,SIGNAL(changeNoiseReduction(int)),this,SLOT(denoiseCurrentFrame(int)));

            scrollArea->widget()->layout()->addWidget(infoBlock);

            currentFrameId = 0;
            infoBlock->setFrameMax(frames.count());
            infoBlock->setFrame(currentFrameId);
            infoBlock->setColor(color);
        }

        void ProfileData::deleteFromScrollArea()
        {
            infoBlock->deleteLater();
        }

#ifndef FRAME_H
#define FRAME_H

#include <QObject>
#include <QtCore>
#include "waveletspectrum.h"

class Frame : public QObject
{
    Q_OBJECT
public:
    enum Direction{
        goDown,
        goUp,
    };
    static void anglesToXY(double inA1,double inA2,double wireLength,double legLength,double leg-
sAngle,double &outX,double &outY,bool &ok);
    explicit Frame(QVector<double> *inRawAngles, QVector<double> *inRawAmpers, Direction inDir,
int startKey, int stopKey, double inWireLength, double inLegLength, double inLegWidth, double
inLegsAngle, double inDeltaX, double inDeltaY, double inLuftAngle, bool inshowMirror, QObject
*parent = 0);
    QVector<double> angles;
    QVector<double> ampers;
    QVector<double> smoothAmpers;
    int noiseReductionPercent = 0;

    void loadData(QVector<double> *inRawAngles,QVector<double> *inRawAmpers);
    int startIndex;
    int stopIndex;
    int zeroAngleIndex;
    void findZeroAngleIndex();
```

```
//-------------------------------------
Direction dir;
double wireLength;
double legLength;
double legWidth;
double deltaX;
double deltaY;
double luftAngle;
double legsAngle;
bool showMirror;
//-------------------------------------
//Максимум
void findMaxCurrentPosition();
bool ok;
double maxCurrentAngle1; double maxAmperLeg1;
double maxCurrentAngle2; double maxAmperLeg2;
int maxCurrentIndex1;
int maxCurrentIndex2;
double maxCurrentX;
double maxCurrentY;
//-------------------------------------
//Ширина на полувысоте
void findWidth();
double widthLeg1;
double widthLeg2;

double Leg1AngleStart = 0;
double Leg1AngleStop = 0;
double Leg2AngleStart = 0;
double Leg2AngleStop = 0;

double LineLeg1StartX;
double LineLeg1StartY;
double LineLeg1StopX;
double LineLeg1StopY;
double LineLeg2StartX;
double LineLeg2StartY;
double LineLeg2StopX;
double LineLeg2StopY;
//-------------------------------------
//Общий ток
void findCurrent();
double currentLeg1;
double currentLeg2;
double current;
//-------------------------------------
//Положение центра масс и ток
void findMass(double partValue);
void findPart(double percentOfBeam);
double part = 2.7182;
double massX;
double massY;

double massWidthLeg1;
double massWidthLeg2;

double massLineLeg1StartX;
double massLineLeg1StartY;
double massLineLeg1StopX;
double massLineLeg1StopY;
double massLineLeg2StartX;
double massLineLeg2StartY;
double massLineLeg2StopX;
double massLineLeg2StopY;


double massLeg1AngleStart = 0;
double massLeg1AngleStop = 0;
double massMiddleAmperLeg1 = 0;

double massLeg2AngleStart = 0;
double massLeg2AngleStop = 0;
double massMiddleAmperLeg2 = 0;

double currentLeg1Mass;
double currentLeg2Mass;
double currentMass;

double difCurrentLeg1;
double difCurrentLeg2;
```

```
    double difCurrent = 0;

private:
    double countCurrent(int start,int stop);
signals:

public slots:
    void changePart(double value);
    void setNoiseReductionPercent(int percent);
    void denoiseAmpers();
};

#endif // FRAME_H
```

```
        #include "frame.h"


        void Frame::anglesToXY(double inA1,double inA2,double wireLength,double legLength,double
legsAngle,double &outX,double &outY,bool &ok){
            ok = true;

            double a1 = inA1*M_PI/180.0;
            double a2 = inA2*M_PI/180.0;

            double as1 = M_PI_2-a1;
            double as2 = M_PI_2-a2;

            double Xc1 = wireLength*cos(as1);
            double Yc1 = wireLength*sin(as1);

            double Xc2 = wireLength*cos(as2);
            double Yc2 = wireLength*sin(as2);

            double ac1 = M_PI - ((1-legsAngle/360.0)*M_PI - as1);
            double ac2 = (1-legsAngle/360.0)*M_PI - (M_PI - as2);

            double cosC1 = cos(ac1);
            double sinC1 = sin(ac1);

            double cosC2 = cos(ac2);
            double sinC2 = sin(ac2);

            double                        Lc1                        =                        (Yc1-Yc2-
((Xc1*sinC2)/(cosC2))+((Xc2*sinC2)/(cosC2)))/(((cosC1*sinC2)/(cosC2))-sinC1);
            double Lc2 = (Xc1-Xc2+Lc1*cosC1)/(cosC2);
            if(Lc1 <= legLength && Lc2 <= legLength){
                outX = Xc1 + Lc1*cosC1;
                outY = Yc1 + Lc1*sinC1;
            }else{
                ok = false;
                outX = 0;
                outY = 0;
            }
        }


        Frame::Frame(QVector<double> *inRawAngles, QVector<double> *inRawAmpers, Direction inDir,
int startKey, int stopKey, double inWireLength, double inLegLength, double inLegWidth,double
inLegsAngle, double inDeltaX, double inDeltaY, double inLuftAngle, bool inshowMirror, QObject
*parent) : QObject(parent)
        {
            dir = inDir;
            startIndex = startKey;
            stopIndex = stopKey;
            luftAngle = inLuftAngle;
            showMirror = inshowMirror;
            legsAngle = inLegsAngle;

            loadData(inRawAngles,inRawAmpers);
            findZeroAngleIndex();

            wireLength = inWireLength;
            legLength = inLegLength;
            legWidth = inLegWidth;
            deltaX = inDeltaX;
            deltaY = inDeltaY;

            denoiseAmpers();
            findMaxCurrentPosition();
            findWidth();
```

```
            findCurrent();
            findPart(95);
        }

        void Frame::loadData(QVector<double> *inRawAngles,QVector<double> *inRawAmpers)
        {
            ampers.clear();
            angles.clear();
            if(!inRawAngles->isEmpty() && !inRawAmpers->isEmpty()){
                if(dir == goDown){
                    for(int i = startIndex; i < stopIndex; i++){
                        ampers.append(inRawAmpers->at(i));
                        angles.append(inRawAngles->at(i)-(luftAngle/2.0));
                    }
                }else if(dir == goUp){
                    for(int i = stopIndex; i >= startIndex; i--){
                        ampers.append(inRawAmpers->at(i));
                        angles.append(inRawAngles->at(i)+(luftAngle/2.0));
                    }
                }
            }
        }

        void Frame::findZeroAngleIndex()
        {
            if(!angles.isEmpty()){
                double minAbs = fabs(angles.at(0));
                zeroAngleIndex = 0;
                for(int i = 1; i < angles.count(); i++){
                    if(fabs(angles.at(i)) < minAbs){
                        minAbs = fabs(angles.at(i));
                        zeroAngleIndex = i;
                    }
                }
            }
        }

        void Frame::findMaxCurrentPosition()
        {
            if(!angles.isEmpty() && !smoothAmpers.isEmpty()){
                double maxAbsValue1 = 0;
                maxCurrentAngle1 = -1;
                for(int i = 0; i < zeroAngleIndex; i++){
                    if(fabs(smoothAmpers.at(i)) > maxAbsValue1){
                        maxAbsValue1 = fabs(smoothAmpers.at(i));
                        maxCurrentAngle1 = angles.at(i);
                        maxCurrentIndex1 = i;
                    }
                }
                double maxAbsValue2 = 0;
                for(int i = zeroAngleIndex; i < smoothAmpers.count(); i++){
                    if(fabs(smoothAmpers.at(i)) > maxAbsValue2){
                        maxAbsValue2 = fabs(smoothAmpers.at(i));
                        maxCurrentAngle2 = angles.at(i);
                        maxCurrentIndex2 = i;
                    }
                }
                maxAmperLeg1 = smoothAmpers.at(maxCurrentIndex1);
                maxAmperLeg2 = smoothAmpers.at(maxCurrentIndex2);
                an-
glesToXY(maxCurrentAngle1,maxCurrentAngle2,wireLength,legLength,legsAngle,maxCurrentX,maxCurrentY
,ok);
                maxCurrentX += deltaX;
                maxCurrentY += deltaY;

                if(showMirror){
                    maxCurrentX = -1.0*maxCurrentX;
                }

            }
        }

        void Frame::findWidth()
        {
            if(ok && !smoothAmpers.isEmpty() && !angles.isEmpty()){
                double halfLeg1 = smoothAmpers.at(maxCurrentIndex1)/2.0;
                double halfLeg2 = smoothAmpers.at(maxCurrentIndex2)/2.0;

                QVector<double> anglesOfMiddleLeg1;
                QVector<double> anglesOfMiddleLeg2;
```

```
                for(int i = 1; i < zeroAngleIndex; i++){
                    if((smoothAmpers.at(i-1) >= halfLeg1 && smoothAmpers.at(i) <= halfLeg1) ||
(smoothAmpers.at(i) >= halfLeg1 && smoothAmpers.at(i-1) <= halfLeg1)){
                        anglesOfMiddleLeg1.append(angles.at(i));
                    }
                }
                for(int i = zeroAngleIndex; i < smoothAmpers.count(); i++){
                    if((smoothAmpers.at(i-1) >= halfLeg2 && smoothAmpers.at(i) <= halfLeg2) ||
(smoothAmpers.at(i) >= halfLeg2 && smoothAmpers.at(i-1) <= halfLeg2)){
                        anglesOfMiddleLeg2.append(angles.at(i));
                    }
                }
                //-----------------------------------------------------
                if(anglesOfMiddleLeg1.count() >= 2 && anglesOfMiddleLeg2.count() >= 2){

                    Leg1AngleStart = anglesOfMiddleLeg1.first();
                    Leg1AngleStop = anglesOfMiddleLeg1.last();
                    Leg2AngleStart = anglesOfMiddleLeg2.first();
                    Leg2AngleStop = anglesOfMiddleLeg2.last();

                    bool ook;
                    an-
glesToXY(anglesOfMiddleLeg1.first(),maxCurrentAngle2,wireLength,legLength,legsAngle,LineLeg1Start
X,LineLeg1StartY,ook);
                    LineLeg1StartX += deltaX;
                    LineLeg1StartY += deltaY;

                    an-
glesToXY(anglesOfMiddleLeg1.last(),maxCurrentAngle2,wireLength,legLength,legsAngle,LineLeg1StopX,
LineLeg1StopY,ook);
                    LineLeg1StopX += deltaX;
                    LineLeg1StopY += deltaY;

                    an-
glesToXY(maxCurrentAngle1,anglesOfMiddleLeg2.first(),wireLength,legLength,legsAngle,LineLeg2Start
X,LineLeg2StartY,ook);
                    LineLeg2StartX += deltaX;
                    LineLeg2StartY += deltaY;

                    an-
glesToXY(maxCurrentAngle1,anglesOfMiddleLeg2.last(),wireLength,legLength,legsAngle,LineLeg2StopX,
LineLeg2StopY,ook);
                    LineLeg2StopX += deltaX;
                    LineLeg2StopY += deltaY;

                    widthLeg1  =  sqrt(powf((LineLeg1StartX-LineLeg1StopX),2)+powf((LineLeg1StartY-
LineLeg1StopY),2));
                    widthLeg2  =  sqrt(powf((LineLeg2StartX-LineLeg2StopX),2)+powf((LineLeg2StartY-
LineLeg2StopY),2));
                }
            }
        }

        void Frame::findCurrent()
        {
            if(!ampers.isEmpty() && !angles.isEmpty()){
                currentLeg1 = countCurrent(0,zeroAngleIndex);
                currentLeg2 = countCurrent(zeroAngleIndex,ampers.count());
                current = (currentLeg1+currentLeg2)/2.0;
            }
        }

        void Frame::findMass(double partValue)
        {
            if(!angles.isEmpty() && !smoothAmpers.isEmpty()){
                if(partValue > 1){
                    part = partValue;
                    double halfLeg1 = smoothAmpers.at(maxCurrentIndex1)/partValue;
                    double halfLeg2 = smoothAmpers.at(maxCurrentIndex2)/partValue;
                    massMiddleAmperLeg1 = halfLeg1;
                    massMiddleAmperLeg2 = halfLeg2;


                    QVector<int> anglesIndexOfMiddleLeg1;
                    QVector<int> anglesIndexOfMiddleLeg2;
                    for(int i = 1; i < zeroAngleIndex; i++){
                        if((smoothAmpers.at(i-1) >= halfLeg1 && smoothAmpers.at(i) <= halfLeg1) ||
(smoothAmpers.at(i) >= halfLeg1 && smoothAmpers.at(i-1) <= halfLeg1)){
                            anglesIndexOfMiddleLeg1.append(i);
                        }
```

```
                    }
                    for(int i = zeroAngleIndex; i < smoothAmpers.count(); i++){
                        if((smoothAmpers.at(i-1) >= halfLeg2 && smoothAmpers.at(i) <= halfLeg2) ||
(smoothAmpers.at(i) >= halfLeg2 && smoothAmpers.at(i-1) <= halfLeg2)){
                            anglesIndexOfMiddleLeg2.append(i);
                        }
                    }
                    if(anglesIndexOfMiddleLeg1.count() >= 2 && anglesIndexOfMiddleLeg2.count() >=
2){
                        massLeg1AngleStart = angles.at(anglesIndexOfMiddleLeg1.first());
                        massLeg1AngleStop = angles.at(anglesIndexOfMiddleLeg1.last());
                        massLeg2AngleStart = angles.at(anglesIndexOfMiddleLeg2.first());
                        massLeg2AngleStop = angles.at(anglesIndexOfMiddleLeg2.last());

                        double angle1 = (massLeg1AngleStart+massLeg1AngleStop)/2.0;
                        double angle2 = (massLeg2AngleStart+massLeg2AngleStop)/2.0;
                        bool ook;
                        anglesToXY(angle1,angle2,wireLength,legLength,legsAngle,massX,massY,ook);
                        massX += deltaX;
                        massY += deltaY;
                        //----------------------------
                        an-
glesToXY(massLeg1AngleStart,maxCurrentAngle2,wireLength,legLength,legsAngle,massLineLeg1StartX,ma
ssLineLeg1StartY,ook);
                        massLineLeg1StartX += deltaX;
                        massLineLeg1StartY += deltaY;

                        an-
glesToXY(massLeg1AngleStop,maxCurrentAngle2,wireLength,legLength,legsAngle,massLineLeg1StopX,mass
LineLeg1StopY,ook);
                        massLineLeg1StopX += deltaX;
                        massLineLeg1StopY += deltaY;

                        an-
glesToXY(maxCurrentAngle1,massLeg2AngleStart,wireLength,legLength,legsAngle,massLineLeg2StartX,ma
ssLineLeg2StartY,ook);
                        massLineLeg2StartX += deltaX;
                        massLineLeg2StartY += deltaY;

                        an-
glesToXY(maxCurrentAngle1,massLeg2AngleStop,wireLength,legLength,legsAngle,massLineLeg2StopX,mass
LineLeg2StopY,ook);
                        massLineLeg2StopX += deltaX;
                        massLineLeg2StopY += deltaY;

                        massWidthLeg1               =               sqrt(powf((massLineLeg1StartX-
massLineLeg1StopX),2)+powf((massLineLeg1StartY-massLineLeg1StopY),2));
                        massWidthLeg2               =               sqrt(powf((massLineLeg2StartX-
massLineLeg2StopX),2)+powf((massLineLeg2StartY-massLineLeg2StopY),2));
                        //----------------------------
                        if(showMirror){
                            massX = -1.0*massX;
                        }
                        currentLeg1Mass                         =                       countCur-
rent(anglesIndexOfMiddleLeg1.first(),anglesIndexOfMiddleLeg1.last());
                        currentLeg2Mass                         =                       countCur-
rent(anglesIndexOfMiddleLeg2.first(),anglesIndexOfMiddleLeg2.last());
                        currentMass = (currentLeg1Mass+currentLeg2Mass)/2.0;

                        difCurrentLeg1 = currentLeg1Mass*100.0/currentLeg1;
                        difCurrentLeg2 = currentLeg2Mass*100.0/currentLeg2;
                        difCurrent = currentMass*100.0/current;
                    }
                }
            }
        }

        void Frame::findPart(double percentOfBeam)
        {
            part = 1.0;
            while(difCurrent < percentOfBeam && part < 100){
                part += 0.1;
                findMass(part);
            }
        }

        double Frame::countCurrent(int start, int stop)
        {
            //=================================================
            double result = 0;
```

```
            if(!ampers.isEmpty()){
                double     r1     =     sqrt(wireLength*wireLength     +     legLength*legLength     -
2*wireLength*legLength*cos((1-legsAngle/360.0)*M_PI));
                result += 1000*ampers.at(start);
                for(int i = start+1;i < stop; i++){
                    double deltaFi = fabs(angles.at(i-1) - angles.at(i));
                    double              squWay              =              (M_PI*r1*r1*deltaFi/360.0)-
(M_PI*wireLength*wireLength*deltaFi/360.0);
                    double squLeg = legLength*legWidth;
                    result += 1000*ampers.at(i)*squWay/squLeg;
                }
            }
            return result;
        }

        void Frame::changePart(double value)
        {
            findMass(value);
        }

        void Frame::setNoiseReductionPercent(int percent)
        {
            noiseReductionPercent = percent;
            denoiseAmpers();

            findMaxCurrentPosition();
            findWidth();
            findCurrent();
            findMass(part);
        }

        void Frame::denoiseAmpers()
        {
            if(!ampers.isEmpty()){
                smoothAmpers.clear();
                WaveletSpectrum *DWT = new WaveletSpectrum(ampers,WaveletSpectrum::BSPLINE_309);
                DWT->threshold(noiseReductionPercent/100.0);
                smoothAmpers = DWT->toData();
                delete DWT;
            }
        }
#ifndef PROFILEINFOBLOCK_H
#define PROFILEINFOBLOCK_H

#include <QWidget>
#include <QtCore>
#include <QColorDialog>

namespace Ui {
class ProfileInfoBlock;
}

class ProfileInfoBlock : public QWidget
{
    Q_OBJECT

public:
    explicit ProfileInfoBlock(QWidget *parent = 0);
    ~ProfileInfoBlock();
    void setFrame(int id);
    void setFrameMax(int value);
    void setScanDate(QString);
    void setFileName(QString);
    void setMaxValueData(double maxAngleLeg1,double maxAngleLeg2,double maxX,double maxY,double
maxAmpLeg1,double maxAmpLeg2);
    void setWidth(double width1, double width2);
    void setCurrent(double currentLeg1,double currentLeg2,double current);
    void setMassXY(double massX, double massY);
    void setMassWidth(double massWidthLeg1,double massWidthLeg2);
    void setMassCurrent(double currentLeg1,double currentLeg2, double current, double
difLeg1,double difLeg2, double dif);
    void setSection(double leg1StartAngle,double leg1StopAngle,double leg2StartAngle, double
leg2StopAngle,double angleToMMHof);
    void setColor(QColor color);
    void setPart(double value);
signals:
    void removeProfile();
    void showFrame(int id);
    void changePart(double value);
    void showMaxCheck(bool);
```

```
    void showIntervalCheck(bool);
    void showMiddleCheck(bool);
    void redrawAll(QColor);
    void showProfileWindow();
    void changeLineSize(int value);
    void changeLineType(Qt::PenStyle);
    void changeNoiseReduction(int value);
private slots:


    void on_closeButton_clicked();

    void on_frameSlider_valueChanged(int value);

    void on_partBox_valueChanged(double arg1);

    void on_showMaxCheck_clicked(bool checked);

    void on_showIntervalCheck_clicked(bool checked);

    void on_showMiddleCheck_clicked(bool checked);

    void on_pushButton_clicked();

    void on_pushButton_2_clicked();

    void on_sizeBox_valueChanged(int arg1);

    void on_lineTypeBox_currentIndexChanged(int index);

    void on_noiseReductionSlider_valueChanged(int value);

private:
    Ui::ProfileInfoBlock *ui;
};

#endif // PROFILEINFOBLOCK_H
#include "profileinfoblock.h"
#include "ui_profileinfoblock.h"

ProfileInfoBlock::ProfileInfoBlock(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::ProfileInfoBlock)
{
    ui->setupUi(this);
    ui->lineTypeBox->addItem("Сплошная");           //SolidLine
    ui->lineTypeBox->addItem("Штриховая");          //DashLine
    ui->lineTypeBox->addItem("Пунктирная");         //DotLine
    ui->lineTypeBox->addItem("Штрихпунктирная");        //DashDotLine
    ui->lineTypeBox->addItem("Штрихпунктирная с двумя точками");    //DashDotDotLine
}

ProfileInfoBlock::~ProfileInfoBlock()
{
    delete ui; //sfdad
}

void ProfileInfoBlock::setFrame(int id)
{
    ui->frameSlider->setValue(id);
    ui->frameNumberLabel->setText(QString::number(id+1)+"/"+QString::number(ui->frameSlider-
>maximum()+1));
}

void ProfileInfoBlock::setFrameMax(int value)
{
    ui->frameSlider->setMaximum(value-1);
}

void ProfileInfoBlock::setScanDate(QString date)
{
    ui->scanDateLabel->setText(date);
}

void ProfileInfoBlock::setFileName(QString fileName)
{
    ui->fileNameLabel->setText(fileName);
}
```

```
void ProfileInfoBlock::setMaxValueData(double maxAngleLeg1, double maxAngleLeg2, double maxX,
double maxY, double maxAmpLeg1, double maxAmpLeg2)
{
    ui->maxAngleLeg1Label->setText(QString::number(maxAngleLeg1));
    ui->maxAngleLeg2Label->setText(QString::number(maxAngleLeg2));
    ui->maxXLabel->setText(QString::number(maxX));
    ui->maxYLabel->setText(QString::number(maxY));
    ui->maxAmpLeg1Label->setText(QString::number(maxAmpLeg1));
    ui->maxAmpLeg2Label->setText(QString::number(maxAmpLeg2));
}

void ProfileInfoBlock::setWidth(double width1, double width2)
{
    ui->width1Label->setText(QString::number(width1));
    ui->width2Label->setText(QString::number(width2));
    ui->sqLabelMid->setText(QString::number(M_PI*width1*width2/4.0));
}

void ProfileInfoBlock::setCurrent(double currentLeg1, double currentLeg2, double current)
{
    ui->currentLeg1Label->setText(QString::number(currentLeg1));
    ui->currentLeg2Label->setText(QString::number(currentLeg2));
    ui->currentLabel->setText(QString::number(current));
}

void ProfileInfoBlock::setMassXY(double massX, double massY)
{
    ui->massXLabel->setText(QString::number(massX));
    ui->massYLabel->setText(QString::number(massY));
}

void ProfileInfoBlock::setMassWidth(double massWidthLeg1, double massWidthLeg2)
{
    ui->massWidthLeg1Label->setText(QString::number(massWidthLeg1));
    ui->massWidthLeg2Label->setText(QString::number(massWidthLeg2));
    ui->sqLabel->setText(QString::number(M_PI*massWidthLeg1*massWidthLeg2/4.0));
}

void ProfileInfoBlock::setMassCurrent(double currentLeg1, double currentLeg2, double current,
double difLeg1, double difLeg2, double dif)
{
    ui->massCurrentLeg1Label->setText(QString::number(currentLeg1));
    ui->massCurrentLeg2Label->setText(QString::number(currentLeg2));
    ui->massCurrentLabel->setText(QString::number(current));
    ui->diffLeg1Label->setText(QString::number(difLeg1));
    ui->diffLeg2Label->setText(QString::number(difLeg2));
    ui->diffLabel->setText(QString::number(dif));
}

void ProfileInfoBlock::setSection(double leg1StartAngle, double leg1StopAngle, double
leg2StartAngle, double leg2StopAngle, double angleToMMHof)
{
    ui->sectionLeg1StartLabel->setText(QString::number(leg1StartAngle*angleToMMHof));
    ui->sectionLeg1StopLabel->setText(QString::number(leg1StopAngle*angleToMMHof));
    ui->sectionLeg2StartLabel->setText(QString::number(leg2StartAngle*angleToMMHof));
    ui->sectionLeg2StopLabel->setText(QString::number(leg2StopAngle*angleToMMHof));
}

void ProfileInfoBlock::setColor(QColor color)
{
    ui->pushButton-
>setStyleSheet("background:rgb("+QString::number(color.red())+","+QString::number(color.green())+
","+QString::number(color.blue())+")");
}

void ProfileInfoBlock::setPart(double value)
{
    ui->partBox->setValue(value);
}

void ProfileInfoBlock::on_closeButton_clicked()
{
    emit removeProfile();
}

void ProfileInfoBlock::on_frameSlider_valueChanged(int value)
{
    emit showFrame(value);
}
```

```
void ProfileInfoBlock::on_partBox_valueChanged(double arg1)
{
    emit changePart(arg1);
}

void ProfileInfoBlock::on_showMaxCheck_clicked(bool checked)
{
    emit showMaxCheck(checked);
}

void ProfileInfoBlock::on_showIntervalCheck_clicked(bool checked)
{
    emit showIntervalCheck(checked);
}

void ProfileInfoBlock::on_showMiddleCheck_clicked(bool checked)
{
    emit showMiddleCheck(checked);
}

void ProfileInfoBlock::on_pushButton_clicked()
{
    QColor color = QColorDialog::getColor();

    ui->pushButton-
>setStyleSheet("background:rgb("+QString::number(color.red())+","+QString::number(color.green())+
","+QString::number(color.blue())+")");
    emit redrawAll(color);
}

void ProfileInfoBlock::on_pushButton_2_clicked()
{
    emit showProfileWindow();
}

void ProfileInfoBlock::on_sizeBox_valueChanged(int arg1)
{
    emit changeLineSize(arg1);
}

void ProfileInfoBlock::on_lineTypeBox_currentIndexChanged(int index)
{
    Qt::PenStyle type;
    type = (Qt::PenStyle)(index + 1);
    emit changeLineType(type);
}

void ProfileInfoBlock::on_noiseReductionSlider_valueChanged(int value)
{
    emit changeNoiseReduction(value);
}
#ifndef PROFILEWINDOW_H
#define PROFILEWINDOW_H

#include <QDialog>
#include <QDebug>
#include "plot/qcustomplot.h"
#include "frame.h"
namespace Ui {
class ProfileWindow;
}

class ProfileWindow : public QDialog
{
    Q_OBJECT

public:
    explicit ProfileWindow(QVector<double> inAngles,
                           QVector<double> inAmpers,
                           double inWireLength,
                           double inLegLength, double inLegsAngle,
                           int inZeroAngleId,
                           bool inShowMirror,
                           double inDeltaX,
                           double inDeltaY,
                           double inPeakX, double inPeakY,
                           double inMiddleLeg1StartX, double inMiddleLeg1StartY, double inMid-
dleLeg1StopX, double inMiddleLeg1StopY,
                           double inMiddleLeg2StartX, double inMiddleLeg2StartY, double inMid-
dleLeg2StopX, double inMiddleLeg2StopY,
```

```
                            QWidget *parent = 0);
    ~ProfileWindow();

    double wireLength;
    double legLength;
    bool showMirror;
    int zeroAngleId;
    double deltaX;
    double deltaY;
    double legsAngle;

    double peakX,peakY;
    double middleLeg1StartX,middleLeg1StartY,middleLeg1StopX,middleLeg1StopY;
    double middleLeg2StartX,middleLeg2StartY,middleLeg2StopX,middleLeg2StopY;

    QVector<double> angles;
    QVector<double> ampers;
private slots:
    void on_holeBox_valueChanged(int arg1);

    void on_pixelsPerMmBox_valueChanged(int arg1);

    void on_spinBox_valueChanged(int arg1);

    void on_pushButton_clicked();

private:
    Ui::ProfileWindow *ui;

    QCPColorGradient g;
    QCPColorScale *colorScale;
    QCPColorMap *colorMap;

    QCPItemEllipse *hole;
    QCPItemTracer *center;
    QCPItemTracer *peak;
    QCPItemLine *profilometrAxis;
    QCPItemLine *middleLeg1;
    QCPItemLine *middleLeg2;

    void reDraw();
    void initPlot();
};

#endif // PROFILEWINDOW_H
#include "profilewindow.h"
#include "ui_profilewindow.h"

ProfileWindow::ProfileWindow(   QVector<double> inAngles,
                                QVector<double> inAmpers,
                                double inWireLength,
                                double inLegLength,double inLegsAngle,
                                int inZeroAngleId,
                                bool inShowMirror,
                                double inDeltaX,
                                double inDeltaY,
                                double inPeakX, double inPeakY,
                                double inMiddleLeg1StartX, double inMiddleLeg1StartY,double
inMiddleLeg1StopX, double inMiddleLeg1StopY,
                                double inMiddleLeg2StartX,double inMiddleLeg2StartY,double inMid-
dleLeg2StopX,double inMiddleLeg2StopY,
                                QWidget *parent) :
    QDialog(parent),
    ui(new Ui::ProfileWindow)
{
    ui->setupUi(this);
    angles = inAngles;
    ampers = inAmpers;
    wireLength = inWireLength;
    legLength = inLegLength;
    legsAngle = inLegsAngle;
    showMirror = inShowMirror;
    zeroAngleId = inZeroAngleId;
    deltaX = inDeltaX;
    deltaY = inDeltaY;

    peakX = inPeakX;
    peakY = inPeakY;

    middleLeg1StartX = inMiddleLeg1StartX;
```

```
    middleLeg1StartY = inMiddleLeg1StartY;
    middleLeg1StopX = inMiddleLeg1StopX;
    middleLeg1StopY = inMiddleLeg1StopY;

    middleLeg2StartX = inMiddleLeg2StartX;
    middleLeg2StartY = inMiddleLeg2StartY;
    middleLeg2StopX = inMiddleLeg2StopX;
    middleLeg2StopY = inMiddleLeg2StopY;

    initPlot();
    reDraw();
}

ProfileWindow::~ProfileWindow()
{
    delete ui;
}

void ProfileWindow::reDraw()
{
    if(!ampers.isEmpty()){
        int holeDiameter = ui->holeBox->value();
        int pixelsPerMm = ui->pixelsPerMmBox->value();
        //-------------
        double currentMaxAbsValue = 0;
        int maxId = 0;
        for(int i = 0; i < ampers.count(); i++){
            if(fabs(ampers.at(i)) > currentMaxAbsValue){
                currentMaxAbsValue = fabs(ampers.at(i));
                maxId = i;
            }
        }
        double sign = 1;
        if(ampers.at(maxId) < 0){
            sign = -1;
        }
        if(currentMaxAbsValue != 0){
            //-------------------------------------------------------
            colorMap->data()->clear();
            colorMap->data()->clear();
            colorMap->data()->setSize(holeDiameter*pixelsPerMm,holeDiameter*pixelsPerMm);

            colorMap->data()->setRange(QCPRange(-holeDiameter/2.0,holeDiameter/2.0),QCPRange(-
holeDiameter/2.0,holeDiameter/2.0));
            colorMap->data()->setSize(holeDiameter*pixelsPerMm,holeDiameter*pixelsPerMm);
            colorMap->data()->setRange(QCPRange(-holeDiameter/2.0,holeDiameter/2.0),QCPRange(-
holeDiameter/2.0,holeDiameter/2.0));

            for(int i = 0; i < zeroAngleId; i++){
                for(int j = zeroAngleId; j < angles.count(); j++){
                    double x = 0;
                    double y = 0;
                    double a1 = angles.at(i);
                    double a2 = angles.at(j);
                    bool ok = true;
                    Frame::anglesToXY(a1,a2,wireLength,legLength,legsAngle,x,y,ok);
                    if(ok){
                        x += deltaX;
                        y += deltaY;
                        if(showMirror){
                            x *= (-1);
                        }
                        int Xid = 0;
                        int Yid = 0;
                        colorMap->data()->coordToCell(x,y,&Xid,&Yid);
                        double mull = (sign*ampers.at(j) * ampers.at(i))/currentMaxAbsValue;
                        colorMap->data()->setCell(Xid,Yid,mull);
                    }
                }
            }
            colorMap->rescaleDataRange();
            colorScale->setDataRange(QCPRange(-currentMaxAbsValue,currentMaxAbsValue));
            ui->plot->replot();
        }
    }
}

void ProfileWindow::initPlot()
{
    //-----------------------------------------------------------------------
```

```
    ui->plot->setInteraction(QCP::iRangeDrag, true);
    ui->plot->setInteraction(QCP::iRangeZoom, true);
    ui->plot->axisRect()->setRangeDrag(Qt::Horizontal | Qt::Vertical);
    ui->plot->axisRect()->setRangeZoom(Qt::Horizontal | Qt::Vertical);
    ui->plot->addLayer("low");
    ui->plot->addLayer("high");
    ui->plot->xAxis->setLabel("X (мм)");
    ui->plot->yAxis->setLabel("Y (мм)");

    //-------------------------------------------------------------------------
    colorScale = new QCPColorScale(ui->plot);
    colorScale->setType(QCPAxis::atRight);
    g.clearColorStops();
    g.setColorInterpolation(QCPColorGradient::ColorInterpolation::ciRGB);
    g.setColorStopAt(0,QColor(0,0,255));
    g.setColorStopAt(0.5,QColor(255,255,255));
    g.setColorStopAt(1,QColor(255,0,0));
    g.setLevelCount(50);
    colorScale->setGradient(g);
    colorScale->setDataRange(QCPRange(-1,1));
    ui->plot->plotLayout()->addElement(0,1,colorScale);
    //-------------------------------------------------------------------------
    colorMap = new QCPColorMap(ui->plot->xAxis,ui->plot->yAxis);
    colorMap->setColorScale(colorScale);
    colorMap->setTightBoundary(true);
    ui->plot->addPlottable(colorMap);
    colorMap->setLayer("low");
    ui->plot->xAxis->setRange(-30,30);
    ui->plot->yAxis->setRange(-30,30);
    //-------------------------------------------------------------------------
    profilometrAxis = new QCPItemLine(ui->plot);
    QPen axP;
    axP.setColor(QColor(Qt::black));
    axP.setStyle(Qt::PenStyle::DashLine);
    profilometrAxis->setPen(axP);
    profilometrAxis->start->setCoords((showMirror)?(-deltaX):(deltaX),deltaY);
    profilometrAxis->end->setCoords((showMirror)?(-deltaX):(deltaX),deltaY+wireLength);
    ui->plot->addItem(profilometrAxis);
    profilometrAxis->setLayer("high");
    //-------------------------------------------------------------------------
    middleLeg1 = new QCPItemLine(ui->plot);
    middleLeg2 = new QCPItemLine(ui->plot);
    middleLeg1->setPen(axP);
    middleLeg2->setPen(axP);
    ui->plot->addItem(middleLeg1);
    ui->plot->addItem(middleLeg2);
    middleLeg1->setLayer("high");
    middleLeg2->setLayer("high");

    middleLeg1->start->setCoords((showMirror)?(-
middleLeg1StartX):(middleLeg1StartX),middleLeg1StartY);
    middleLeg1->end->setCoords((showMirror)?(-
middleLeg1StopX):(middleLeg1StopX),middleLeg1StopY);
    middleLeg2->start->setCoords((showMirror)?(-
middleLeg2StartX):(middleLeg2StartX),middleLeg2StartY);
    middleLeg2->end->setCoords((showMirror)?(-
middleLeg2StopX):(middleLeg2StopX),middleLeg2StopY);
    //-------------------------------------------------------------------------
    hole = new QCPItemEllipse(ui->plot);
    QPen p1;
    p1.setWidth(2);
    p1.setColor(QColor(Qt::green));
    p1.setStyle(Qt::PenStyle::DashLine);
    hole->setPen(p1);

    hole->topLeft->setCoords(-ui->holeBox->value()/2.0,ui->holeBox->value()/2.0);
    hole->bottomRight->setCoords(ui->holeBox->value()/2.0,-ui->holeBox->value()/2.0);
    ui->plot->addItem(hole);
    hole->setLayer("high");
    //-------------------------------------------------------------------------
    center = new QCPItemTracer(ui->plot);
    center->position->setCoords(0,0);
    center->setStyle(QCPItemTracer::tsPlus);
    ui->plot->addItem(center);
    center->setLayer("high");
    //-------------------------------------------------------------------------
    peak = new QCPItemTracer(ui->plot);
    peak->position->setCoords(0,0);
    peak->setStyle(QCPItemTracer::tsCircle);
    peak->setPen(QPen(QColor(255,255,0,200)));
```

```
    peak->setBrush(QBrush(QColor(255,255,0,100)));
    peak->position->setCoords(peakX,peakY);
    ui->plot->addItem(peak);
    peak->setLayer("high");
}

void ProfileWindow::on_holeBox_valueChanged(int arg1)
{
    Q_UNUSED(arg1);
    hole->topLeft->setCoords(-ui->holeBox->value()/2.0,ui->holeBox->value()/2.0);
    hole->bottomRight->setCoords(ui->holeBox->value()/2.0,-ui->holeBox->value()/2.0);
    reDraw();
}

void ProfileWindow::on_pixelsPerMmBox_valueChanged(int arg1)
{
    Q_UNUSED(arg1);
    reDraw();
}

void ProfileWindow::on_spinBox_valueChanged(int arg1)
{
    g.setLevelCount(arg1);
    colorScale->setGradient(g);
    ui->plot->replot();
}

void ProfileWindow::on_pushButton_clicked()
{
    QString s = QFileDialog::getSaveFileName(this,"Сохранить скриншот","","png");
    ui->plot->grab().save(s+".png","png");
}
#ifndef SETTINGSDIALOG_H
#define SETTINGSDIALOG_H

#include <QDialog>
#include <QSettings>
#include "mainwindow.h"
namespace Ui {
class SettingsDialog;
}

class SettingsDialog : public QDialog
{
    Q_OBJECT
signals:
    void settingsUpdate(double wireLength,double legLength,double legWidth,double legAngle, dou-
ble deltaX, double deltaY, double luftAngle, bool showMirror,double angleToRKoefNumerator, double
angleToRKoefDenominator);
public:
    explicit SettingsDialog(double wireLength,double legLength,double legWidth,double legAngle,
double deltaX, double deltaY, double luftAngle, bool showMirror,double angleToRKoefNumerator,
double angleToRKoefDenominator,QWidget *parent = 0);

    ~SettingsDialog();

private slots:
    void on_acceptButton_clicked();

private:
    Ui::SettingsDialog *ui;

};

#endif // SETTINGSDIALOG_H
#include "settingsdialog.h"
#include "ui_settingsdialog.h"

SettingsDialog::SettingsDialog(double wireLength, double legLength, double legWidth,double legAn-
gle, double deltaX, double deltaY, double luftAngle, bool showMirror,double angleToRKoefNumera-
tor, double angleToRKoefDenominator,QWidget *parent) :
    QDialog(parent),
    ui(new Ui::SettingsDialog)
{
    ui->setupUi(this);
    ui->wireLengthBox->setValue(wireLength);
    ui->legLengthBox->setValue(legLength);
    ui->legWidthBox->setValue(legWidth);
    ui->legsAngleBox->setValue(legAngle);
    ui->deltaXBox->setValue(deltaX);
```

```
    ui->deltaYBox->setValue(deltaY);
    ui->luftBox->setValue(luftAngle);
    ui->checkBox->setChecked(showMirror);
    ui->angleToRKoefDenominatorBox->setValue(angleToRKoefDenominator);
    ui->angleToRKoefNumeratorBox->setValue(angleToRKoefNumerator);
    ui->picLabel->setPixmap(QPixmap("settingsPic.png"));
}
SettingsDialog::~SettingsDialog()
{
    delete ui;
}

void SettingsDialog::on_acceptButton_clicked()
{
    QSettings settings(MainWindow::settingsFileName,QSettings::IniFormat);
    settings.setValue(MainWindow::wireLengthFieldName,ui->wireLengthBox->value());
    settings.setValue(MainWindow::legLengthFieldName,ui->legLengthBox->value());
    settings.setValue(MainWindow::legWidthFieldName,ui->legWidthBox->value());
    settings.setValue(MainWindow::deltaXFieldName,ui->deltaXBox->value());
    settings.setValue(MainWindow::deltaYFieldName,ui->deltaYBox->value());
    settings.setValue(MainWindow::luftAngleFieldName,ui->luftBox->value());
    settings.setValue(MainWindow::showMirrorFieldName,ui->checkBox->isChecked());
    settings.setValue(MainWindow::legsAngleFieldName,ui->legsAngleBox->value());
    settings.setValue(MainWindow::angleToRKoefNumeratorFieldName,ui->angleToRKoefNumeratorBox-
>value());
    settings.setValue(MainWindow::angleToRKoefDenominatorFieldName,ui-
>angleToRKoefDenominatorBox->value());
    emit settingsUpdate(ui->wireLengthBox->value(),ui->legLengthBox->value(),ui->legWidthBox-
>value(),ui->legsAngleBox->value(),ui->deltaXBox->value(),ui->deltaYBox->value(),ui->luftBox-
>value(),ui->checkBox->isChecked(),ui->angleToRKoefNumeratorBox->value(),ui-
>angleToRKoefDenominatorBox->value());
    this->close();
}

#ifndef DIRECTORYMONITOR_H
#define DIRECTORYMONITOR_H

#include <QObject>
#include <QtCore>
#include <QDebug>
#include <QDir>
#include <QTimer>

class DirectoryMonitor : public QObject
{
    Q_OBJECT
public:
    DirectoryMonitor(QObject *parent = 0);
    DirectoryMonitor(QString initDirectiryPath = "",int refreshTimeMs = 1000,QString lookupFormat
= "",bool lookInSubFolder = true,QObject *parent = 0);

    void setDirectory(QString initDirectiryPath);
    void setRefreshTime(int refreshTimeMs);
    void setLoopupFormat(QString lookupFormat);
    void setLookInSubFolder(bool lookInSubFolder);


    void start();
    void stop();
signals:
    void fileChanged(QString fileName);
private slots:
    void refresh();
private:
    QDir dir;
    QMap<QString,QDateTime> *list;
    QTimer timer;
    QString format = "";
    bool subFolder = true;
    int refreshTime = 1000;
    void fileScan(QDir dir,QMap<QString,QDateTime> * newList);
};

#endif // DIRECTORYMONITOR_H
#include "directorymonitor.h"

DirectoryMonitor::DirectoryMonitor(QObject *parent) : QObject(parent)
{
    list = new QMap<QString,QDateTime>;
    fileScan(dir,list);
```

```
    timer.setInterval(refreshTime);
    connect(&timer,SIGNAL(timeout()),this,SLOT(refresh()));

}

DirectoryMonitor::DirectoryMonitor(QString initDirectiryPath, int refreshTimeMs, QString
lookupFormat, bool lookInSubFolder, QObject *parent) : QObject(parent)
{
    format = lookupFormat;
    dir.setPath(initDirectiryPath);
    subFolder = lookInSubFolder;
    list = new QMap<QString,QDateTime>;
    refreshTime = refreshTimeMs;


    fileScan(dir,list);
    timer.setInterval(refreshTime);
    connect(&timer,SIGNAL(timeout()),this,SLOT(refresh()));
}

void DirectoryMonitor::setDirectory(QString initDirectiryPath)
{
    dir.setPath(initDirectiryPath);
}

void DirectoryMonitor::setRefreshTime(int refreshTimeMs)
{
    refreshTime = refreshTimeMs;
    timer.setInterval(refreshTime);
}

void DirectoryMonitor::setLoopupFormat(QString lookupFormat)
{
    format = lookupFormat;
}

void DirectoryMonitor::setLookInSubFolder(bool lookInSubFolder)
{
    subFolder = lookInSubFolder;
}

void DirectoryMonitor::start()
{
    timer.start();
}

void DirectoryMonitor::stop()
{
    timer.stop();
}

void DirectoryMonitor::refresh()
{
    dir.refresh();
    QMap<QString,QDateTime> *newList = new QMap<QString,QDateTime>;
    fileScan(dir,newList);

    //QMapIterator<QString,QDateTime> oldIterator(*list);
    QMapIterator<QString,QDateTime> newIterator(*newList);
    while(newIterator.hasNext()){
        newIterator.next();
        if(list->contains(newIterator.key())){
            if(newIterator.value() != list->value(newIterator.key())){
                emit fileChanged(newIterator.key());
            }
        }else{
            emit fileChanged(newIterator.key());
        }

    }
    list->clear();
    delete list;
    list = newList;
}
void DirectoryMonitor::fileScan(QDir dir, QMap<QString, QDateTime> *newList)
{
    QFileInfoList entryList = dir.entryInfoList();
    for(int i = 0; i < entryList.count(); i++){
        if(entryList[i].isFile() && entryList[i].fileName().endsWith(format)){
            newList->insert(entryList[i].filePath(),entryList[i].lastModified());
```

```
        }
        if(subFolder){
            if(entryList[i].isDir() && entryList[i].fileName()!="." && entryL-
ist[i].fileName()!=".."){
                fileScan(entryList[i].filePath(),newList);
            }
        }
    }
}
/* WaveletSpectrum
 * Обертка вокруг gsl для удобного использования вейвлет преобразований с типами данных Qt
 * Tima45 14.02.2017
 */
#ifndef WAVELETSPECTRUM_H
#define WAVELETSPECTRUM_H

#include <QObject>
#include <QtCore>
class WaveletSpectrum : public QObject
{
    Q_OBJECT
public:
    enum WaveTypes{
        HAAR_2,
        HAAR_CENTERED_2,
        DAUBECHIES_4,
        DAUBECHIES_6,
        DAUBECHIES_8,
        DAUBECHIES_10,
        DAUBECHIES_12,
        DAUBECHIES_14,
        DAUBECHIES_16,
        DAUBECHIES_18,
        DAUBECHIES_20,
        DAUBECHIES_CENTERED_4,
        DAUBECHIES_CENTERED_6,
        DAUBECHIES_CENTERED_8,
        DAUBECHIES_CENTERED_10,
        DAUBECHIES_CENTERED_12,
        DAUBECHIES_CENTERED_14,
        DAUBECHIES_CENTERED_16,
        DAUBECHIES_CENTERED_18,
        DAUBECHIES_CENTERED_20,
        BSPLINE_103,
        BSPLINE_105,
        BSPLINE_202,
        BSPLINE_204,
        BSPLINE_206,
        BSPLINE_208,
        BSPLINE_301,
        BSPLINE_303,
        BSPLINE_305,
        BSPLINE_307,
        BSPLINE_309,
        BSPLINE_CENTERED_103,
        BSPLINE_CENTERED_105,
        BSPLINE_CENTERED_202,
        BSPLINE_CENTERED_204,
        BSPLINE_CENTERED_206,
        BSPLINE_CENTERED_208,
        BSPLINE_CENTERED_301,
        BSPLINE_CENTERED_303,
        BSPLINE_CENTERED_305,
        BSPLINE_CENTERED_307,
        BSPLINE_CENTERED_309
    };
    WaveletSpectrum(QObject *parent = 0);
    WaveletSpectrum(QVector<double> data,WaveTypes waveType,QObject *parent = 0);
    int getLevels();
    WaveTypes getType();
    //-------------------------------------
    /* Двумерный спектр в ваших руках.
     * Не меняйте количество элементов, иначе обратное преобразование может вернуть чупуху или
сломаться.
     * Мне лень писать проверки.
     */
    double constantAmplitude;
    QVector<QVector<double>> spectrum;
    //-------------------------------------
    void threshold(double percent);
```

```
    /* threshold
     * Зануляет все амплитуды которые меньше(в процентах) чем значение percent.
     * Таким образом можно удалить небольшие всплески, избавив сигнал от шума.
     * Праметр: percent
     * percent == 100 удалит все амплитуды, обнулит спектр заисключением constantAmplitude, оста-
вит прямую.
     * percent > 100 так-же удалит все.
     * percent < 100 удалит мелкие всплески оставив большие.
     * В зависимости от амплитуды шума можно выбирать значение percent от 0.1(если шума мало) до
10(Если шум по амплитуде состовляет примерно 10 процентов от всего сигнала)
     */
    //-------------------------------------
    void highFilter(double value);
    /* highFilter
     * Умножает верхнюю половину(levels/2) амплитуд на значение value
     * Праметр: value
     * value == 1 то частоты остаются прежними.
     * value < 1 уменьшает амплитуду верхних частот.
     * value > 1 увеличивает амплитуду верхних частот.
     * value == 0 срезает верхнюю половину.
     */
    //-------------------------------------
    void lowFilter(double value);
    /* lowFilter
     * Умножает нижнюю половину(levels/2) амплитуд на значение value
     * Праметр: value
     * value == 1 то частоты остаются прежними.
     * value < 1 уменьшает амплитуду нижних частот.
     * value > 1 увеличивает амплитуду нижних частот.
     * value == 0 срезает нижних половину.
     */
    //-------------------------------------
    void levelFilter(int level,double value);
    /* levelFilter
     * Умножает умплитуду на заданном уровене(частоте)level на значения value.
     * level должно быть в интервале от 0 до levels иначе ничего не произойдет.
     */
    //-------------------------------------
    void intervalFilter(int startLevel,int stopLevel,double value);
    /* intervalFilter
     * Умнажает амплитуды на значение value для заданного интервала частот
     * Параметры:
     * startLevel нижнея граница фильтра.
     * startLevel < 0 интервал от нижних частот.
     *
     * stopLevel верхняя граница фильтра.
     * stopLevel > levels максимальная верхняя частота
     *
     * startLevel > stopLevel ничего не произойдет.
     * startLevel == stopLevel эффект для одного уровня. равносильно вызову levelFilter();
     * value коеффициент умножения.
     */
    QVector<double> toData();
private:
    int startDataCount;
    int roundToPowerOfTwo(int value);
    WaveTypes type;
    int levels = 0;
    double max();
};

#endif // WAVELETSPECTRUM_H
        #include "waveletspectrum.h"
        #include <gsl/gsl_wavelet.h>

        int WaveletSpectrum::roundToPowerOfTwo(int value){
            double tmp = log2(value);
            return pow(2,ceil(tmp));
        }

        int WaveletSpectrum::getLevels()
        {
            return levels;
        }

        WaveletSpectrum::WaveTypes WaveletSpectrum::getType()
        {
            return type;
        }
        double WaveletSpectrum::max(){
```

```
    double maxValue = 0;
    if(!spectrum.isEmpty()){
        for(int l = 0; l < spectrum.count(); l++){
            for(int i = 0; i < spectrum.at(l).count(); i++){
                if(fabs(spectrum.at(l).at(i)) > maxValue){
                    maxValue = fabs(spectrum.at(l).at(i));
                }
            }
        }
    }
    return maxValue;
}
void WaveletSpectrum::threshold(double percent)
{
    if(!spectrum.isEmpty()){
        double thresholdValue = max()*percent/100.0;
        for(int l = 0; l < spectrum.count(); l++){
            for(int i = 0; i < spectrum.at(l).count(); i++){
                if(fabs(spectrum.at(l).at(i)) < thresholdValue){
                    spectrum[l][i] = 0;
                }
            }
        }
    }
}

void WaveletSpectrum::highFilter(double value)
{
    if(!spectrum.isEmpty()){
        for(int l = levels/2.0; l < spectrum.count(); l++){
            for(int i = 0; i < spectrum.at(l).count(); i++){
                spectrum[l][i] *= value;
            }
        }
    }
}

void WaveletSpectrum::lowFilter(double value)
{
    if(!spectrum.isEmpty()){
        for(int l = 0; l < spectrum.count()/2.0; l++){
            for(int i = 0; i < spectrum.at(l).count(); i++){
                spectrum[l][i] *= value;
            }
        }
    }
}

void WaveletSpectrum::levelFilter(int level, double value)
{
    if(!spectrum.isEmpty()){
        if(level >= 0 && level < levels){
            for(int i = 0; i < spectrum.at(level).count(); i++){
                spectrum[level][i] *= value;
            }
        }
    }
}

void WaveletSpectrum::intervalFilter(int startLevel, int stopLevel, double value)
{
    if(!spectrum.isEmpty()){
        if(stopLevel > levels){
            stopLevel = levels;
        }
        if(startLevel < 0){
            startLevel = 0;
        }
        if(startLevel < stopLevel){
            for(int l = startLevel; l < stopLevel; l++){
                for(int i = 0; i < spectrum.at(l).count(); i++){
                    spectrum[l][i] *= value;
                }
            }
        }
    }
}

QVector<double> WaveletSpectrum::toData()
{
```

```
QVector<double> result;
if(spectrum.isEmpty()){
    return result;
}else{
    gsl_wavelet_workspace *work = gsl_wavelet_workspace_alloc (pow(2,levels));
    gsl_wavelet *w;
    switch(type){
    case HAAR_2:{
        w = gsl_wavelet_alloc (gsl_wavelet_haar,2);
        break;
    }
    case HAAR_CENTERED_2:{
        w = gsl_wavelet_alloc (gsl_wavelet_haar_centered,2);
        break;
    }
    case DAUBECHIES_4:{
        w = gsl_wavelet_alloc (gsl_wavelet_daubechies,4);
        break;
    }
    case DAUBECHIES_6:{
        w = gsl_wavelet_alloc (gsl_wavelet_daubechies,6);
        break;
    }
    case DAUBECHIES_8:{
        w = gsl_wavelet_alloc (gsl_wavelet_daubechies,8);
        break;
    }
    case DAUBECHIES_10:{
        w = gsl_wavelet_alloc (gsl_wavelet_daubechies,10);
        break;
    }
    case DAUBECHIES_12:{
        w = gsl_wavelet_alloc (gsl_wavelet_daubechies,12);
        break;
    }
    case DAUBECHIES_14:{
        w = gsl_wavelet_alloc (gsl_wavelet_daubechies,14);
        break;
    }
    case DAUBECHIES_16:{
        w = gsl_wavelet_alloc (gsl_wavelet_daubechies,16);
        break;
    }
    case DAUBECHIES_18:{
        w = gsl_wavelet_alloc (gsl_wavelet_daubechies,18);
        break;
    }
    case DAUBECHIES_20:{
        w = gsl_wavelet_alloc (gsl_wavelet_daubechies,20);
        break;
    }
    case DAUBECHIES_CENTERED_4:{
        w = gsl_wavelet_alloc (gsl_wavelet_daubechies_centered,4);
        break;
    }
    case DAUBECHIES_CENTERED_6:{
        w = gsl_wavelet_alloc (gsl_wavelet_daubechies_centered,6);
        break;
    }
    case DAUBECHIES_CENTERED_8:{
        w = gsl_wavelet_alloc (gsl_wavelet_daubechies_centered,8);
        break;
    }
    case DAUBECHIES_CENTERED_10:{
        w = gsl_wavelet_alloc (gsl_wavelet_daubechies_centered,10);
        break;
    }
    case DAUBECHIES_CENTERED_12:{
        w = gsl_wavelet_alloc (gsl_wavelet_daubechies_centered,12);
        break;
    }
    case DAUBECHIES_CENTERED_14:{
        w = gsl_wavelet_alloc (gsl_wavelet_daubechies_centered,14);
        break;
    }
    case DAUBECHIES_CENTERED_16:{
        w = gsl_wavelet_alloc (gsl_wavelet_daubechies_centered,16);
        break;
    }
    case DAUBECHIES_CENTERED_18:{
```

```
    w = gsl_wavelet_alloc (gsl_wavelet_daubechies_centered,18);
    break;
}
case DAUBECHIES_CENTERED_20:{
    w = gsl_wavelet_alloc (gsl_wavelet_daubechies_centered,20);
    break;
}
case BSPLINE_103:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,103);
    break;
}
case BSPLINE_105:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,105);
    break;
}
case BSPLINE_202:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,202);
    break;
}
case BSPLINE_204:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,204);
    break;
}
case BSPLINE_206:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,206);
    break;
}
case BSPLINE_208:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,208);
    break;
}
case BSPLINE_301:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,301);
    break;
}
case BSPLINE_303:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,303);
    break;
}
case BSPLINE_305:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,305);
    break;
}
case BSPLINE_307:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,307);
    break;
}
case BSPLINE_309:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,309);
    break;
}
case BSPLINE_CENTERED_103:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline_centered,103);
    break;
}
case BSPLINE_CENTERED_105:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,105);
    break;
}
case BSPLINE_CENTERED_202:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,202);
    break;
}
case BSPLINE_CENTERED_204:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,204);
    break;
}
case BSPLINE_CENTERED_206:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,206);
    break;
}
case BSPLINE_CENTERED_208:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,208);
    break;
}
case BSPLINE_CENTERED_301:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,301);
    break;
}
case BSPLINE_CENTERED_303:{
```

```
                w = gsl_wavelet_alloc (gsl_wavelet_bspline,303);
                break;
            }
            case BSPLINE_CENTERED_305:{
                w = gsl_wavelet_alloc (gsl_wavelet_bspline,305);
                break;
            }
            case BSPLINE_CENTERED_307:{
                w = gsl_wavelet_alloc (gsl_wavelet_bspline,307);
                break;
            }
            case BSPLINE_CENTERED_309:{
                w = gsl_wavelet_alloc (gsl_wavelet_bspline,309);
                break;
            }
            }
            result.append(constantAmplitude);
            for(int l = 0; l < spectrum.count(); l++){
                for(int i = 0; i < spectrum.at(l).count(); i++){
                    result.append(spectrum.at(l).at(i));
                }
            }
            gsl_wavelet_transform_inverse(w,result.data(),1,pow(2,levels),work);
            free(w);
            free(work);
            int needToDelete = result.count()-startDataCount;
            for(int i = 0; i < needToDelete; i++){
                result.removeLast();
            }
            return result;
        }
    }
    WaveletSpectrum::WaveletSpectrum(QObject *parent) : QObject(parent)
    {

    }

    WaveletSpectrum::WaveletSpectrum(QVector<double>  data,  WaveletSpectrum::WaveTypes  wa-
veType, QObject *parent) : QObject(parent)
    {
        if(data.isEmpty()){
            return;
        }
        type = waveType;
        //-------------------------------------------------------------------
        startDataCount = data.count();
        int needToAppend = roundToPowerOfTwo(data.count())-data.count();
        for(int i = 0; i < needToAppend; i++){
            data.append(0);
        }
        levels = log2(data.count());
        //-------------------------------------------------------------------
        gsl_wavelet_workspace *work = gsl_wavelet_workspace_alloc (data.count());
        gsl_wavelet *w;
        switch(type){
        case HAAR_2:{
            w = gsl_wavelet_alloc (gsl_wavelet_haar,2);
            break;
        }
        case HAAR_CENTERED_2:{
            w = gsl_wavelet_alloc (gsl_wavelet_haar_centered,2);
            break;
        }
        case DAUBECHIES_4:{
            w = gsl_wavelet_alloc (gsl_wavelet_daubechies,4);
            break;
        }
        case DAUBECHIES_6:{
            w = gsl_wavelet_alloc (gsl_wavelet_daubechies,6);
            break;
        }
        case DAUBECHIES_8:{
            w = gsl_wavelet_alloc (gsl_wavelet_daubechies,8);
            break;
        }
        case DAUBECHIES_10:{
            w = gsl_wavelet_alloc (gsl_wavelet_daubechies,10);
            break;
        }
        case DAUBECHIES_12:{
```

```
    w = gsl_wavelet_alloc (gsl_wavelet_daubechies,12);
    break;
}
case DAUBECHIES_14:{
    w = gsl_wavelet_alloc (gsl_wavelet_daubechies,14);
    break;
}
case DAUBECHIES_16:{
    w = gsl_wavelet_alloc (gsl_wavelet_daubechies,16);
    break;
}
case DAUBECHIES_18:{
    w = gsl_wavelet_alloc (gsl_wavelet_daubechies,18);
    break;
}
case DAUBECHIES_20:{
    w = gsl_wavelet_alloc (gsl_wavelet_daubechies,20);
    break;
}
case DAUBECHIES_CENTERED_4:{
    w = gsl_wavelet_alloc (gsl_wavelet_daubechies_centered,4);
    break;
}
case DAUBECHIES_CENTERED_6:{
    w = gsl_wavelet_alloc (gsl_wavelet_daubechies_centered,6);
    break;
}
case DAUBECHIES_CENTERED_8:{
    w = gsl_wavelet_alloc (gsl_wavelet_daubechies_centered,8);
    break;
}
case DAUBECHIES_CENTERED_10:{
    w = gsl_wavelet_alloc (gsl_wavelet_daubechies_centered,10);
    break;
}
case DAUBECHIES_CENTERED_12:{
    w = gsl_wavelet_alloc (gsl_wavelet_daubechies_centered,12);
    break;
}
case DAUBECHIES_CENTERED_14:{
    w = gsl_wavelet_alloc (gsl_wavelet_daubechies_centered,14);
    break;
}
case DAUBECHIES_CENTERED_16:{
    w = gsl_wavelet_alloc (gsl_wavelet_daubechies_centered,16);
    break;
}
case DAUBECHIES_CENTERED_18:{
    w = gsl_wavelet_alloc (gsl_wavelet_daubechies_centered,18);
    break;
}
case DAUBECHIES_CENTERED_20:{
    w = gsl_wavelet_alloc (gsl_wavelet_daubechies_centered,20);
    break;
}
case BSPLINE_103:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,103);
    break;
}
case BSPLINE_105:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,105);
    break;
}
case BSPLINE_202:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,202);
    break;
}
case BSPLINE_204:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,204);
    break;
}
case BSPLINE_206:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,206);
    break;
}
case BSPLINE_208:{
    w = gsl_wavelet_alloc (gsl_wavelet_bspline,208);
    break;
}
case BSPLINE_301:{
```

```
            w = gsl_wavelet_alloc (gsl_wavelet_bspline,301);
            break;
        }
        case BSPLINE_303:{
            w = gsl_wavelet_alloc (gsl_wavelet_bspline,303);
            break;
        }
        case BSPLINE_305:{
            w = gsl_wavelet_alloc (gsl_wavelet_bspline,305);
            break;
        }
        case BSPLINE_307:{
            w = gsl_wavelet_alloc (gsl_wavelet_bspline,307);
            break;
        }
        case BSPLINE_309:{
            w = gsl_wavelet_alloc (gsl_wavelet_bspline,309);
            break;
        }
        case BSPLINE_CENTERED_103:{
            w = gsl_wavelet_alloc (gsl_wavelet_bspline_centered,103);
            break;
        }
        case BSPLINE_CENTERED_105:{
            w = gsl_wavelet_alloc (gsl_wavelet_bspline,105);
            break;
        }
        case BSPLINE_CENTERED_202:{
            w = gsl_wavelet_alloc (gsl_wavelet_bspline,202);
            break;
        }
        case BSPLINE_CENTERED_204:{
            w = gsl_wavelet_alloc (gsl_wavelet_bspline,204);
            break;
        }
        case BSPLINE_CENTERED_206:{
            w = gsl_wavelet_alloc (gsl_wavelet_bspline,206);
            break;
        }
        case BSPLINE_CENTERED_208:{
            w = gsl_wavelet_alloc (gsl_wavelet_bspline,208);
            break;
        }
        case BSPLINE_CENTERED_301:{
            w = gsl_wavelet_alloc (gsl_wavelet_bspline,301);
            break;
        }
        case BSPLINE_CENTERED_303:{
            w = gsl_wavelet_alloc (gsl_wavelet_bspline,303);
            break;
        }
        case BSPLINE_CENTERED_305:{
            w = gsl_wavelet_alloc (gsl_wavelet_bspline,305);
            break;
        }
        case BSPLINE_CENTERED_307:{
            w = gsl_wavelet_alloc (gsl_wavelet_bspline,307);
            break;
        }
        case BSPLINE_CENTERED_309:{
            w = gsl_wavelet_alloc (gsl_wavelet_bspline,309);
            break;
        }
        }
        //-----------------------------------------------
        gsl_wavelet_transform_forward (w,data.data(),1,data.count(),work); //<-
        free(w);
        free(work);
        constantAmplitude = data.at(0);
        int j = 1; //итератор для данных
        for(int l = 0; l < levels; l++){
            double maxK = pow(2,l);
            spectrum.append(QVector<double>());
            for(int k = 0; k < maxK; k++){
                spectrum.last().append(data.at(j));
                j++;
            }
        }
    }
```