Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
 высшего образования   
«Национальный исследовательский университет «МЭИ»

Кафедра «Релейная защита и автоматизация энергосистем»

Лабораторная работа №1

«Реализация алгоритма токовой отсечки»

Выполнил: Максимов Р.С.

Группа: Э-13м-19

Проверил: Холодов А. С.

Москва 2020

**Теоретическая справка:**

Токовая отсечка - мгновенно действующая токовая защита, селективность действия которой по отношении к защитам смежных участков достигается выбором тока срабатывания Iсз большим максимального тока внешнего трехфазного короткого замыкания Iкз.вн.мах.

Работа защиты на защищаемом участке обеспечивается тем, что ток в линии увеличивается по мере приближения места повреждения к источнику питания.

Для селективной работы токовой отсечки линии АБ ток срабатывания выбирается для трехфазного короткого замыкания следующим образом:

.

Особенность работы токовой отсечки: защищаемая зона, характеризующая чувствительность защиты, составляет только часть линии.

**Описание алгоритма:**

В классе Main прописан код для вывода графика. В данном классе вызываются методы класса Charts. Также создается объект класса inputData и вызывает метод start() для дальнейшей работы с данными:

**public class** Main {  
 **public static void** main(String[] args) **throws** FileNotFoundException {  
  
 **double** coef = 1.1; *//коэф. отстройки  
 //график № 1 для построения мгновенных токов при КЗ в конце линии* Charts.*createAnalogChart*(**"мгн. токи для расчета уставок"**,0);  
 Charts.*addSeries*(**"Фаза А"**, 0, 0);  
 Charts.*addSeries*(**"Фаза B"**, 0, 1);  
 Charts.*addSeries*(**"Фаза С"**, 0, 2);  
 *//график № 2 для построения действующих значений токов при КЗ в конце линии* Charts.*createAnalogChart*(**"rms токи для расчета уставок"**,1);  
 Charts.*addSeries*(**"Фаза А rms"**, 1, 0);  
 Charts.*addSeries*(**"Фаза B rms"**, 1, 1);  
 Charts.*addSeries*(**"Фаза С rms "**, 1, 2);  
 *//график № 3 для построения мгновенных токов при КЗ в начале линии* Charts.*createAnalogChart*(**"Токи КЗ"**,2);  
 Charts.*addSeries*(**"Фаза А"**, 2, 0);  
 Charts.*addSeries*(**"Фаза B"**, 2, 1);  
 Charts.*addSeries*(**"Фаза С"**, 2, 2);  
 *//график № 4 для построения действующих значений токов при КЗ в начале линии* Charts.*createAnalogChart*(**"Rms токи"**,3);  
 Charts.*addSeries*(**"Фаза А rms"**, 3, 0);  
 Charts.*addSeries*(**"Фаза B rms"**, 3, 1);  
 Charts.*addSeries*(**"Фаза С rms "**, 3, 2);  
 *//график № 5 для сигнала срабатывания защиты* Charts.*createDiscreteChart*(**"Trip"**,4);  
 *//найдем из комтрейд файла максимальное значение действующего тока при кз в конце линии для расчета уставки ТО* InputData inD1 = **new** InputData(**"Посчитать уставку"**,**"PhABC80"**);  
 **double** getMax = inD1.start(0.,0,1);  
 *//расчет уставки ТО* **double** setPoint = getMax\*coef;  
 *//добавляем для 4 графика сигнал - уставку* Charts.*addSeries*(**"Уставка"**,3,3);  
 System.***out***.println(**"Уставка = "** + setPoint); *//вывод уставки в консоль  
 //создаем объект класса inpudata и проверяем ТО с рассчитанной уставкой при кз в начале линии* InputData inD2 = **new** InputData(**"Проверить ТО"**,**"PhA80"**);  
 inD2.start(setPoint,2,3);  
 }  
}

Класс inputData необходим для считывания данных с предоставляемых файлов и работы с ними: преобразование сигнала фильтром Фурье и сравнение с уставкой по току. В данном случае, осуществляется считывание с файлов формата .cfg и .dat. В методе start() реализован алгоритм работы с файлами, а именно преобразование значений файла в мгновенные значения токов с помощью масштабных коэффициентов. Объекты filter и logic классов Filter и Logic непосредственно вызывают методы для преобразования сигнала и сравнения с уставкой.

**import** java.io.\*;  
  
**public class** InputData {  
  
 **public** File **comtrCfg**, **comtrDat**;  
 **private** BufferedReader **br**;  
 **private** String **line**;  
 **private** String[] **lineData**;  
 **private double**[] **k1**;  
 **private double**[] **k2**;  
 **private boolean t** = **false**;  
 **private** String **comtradeName**;  
 **private** String **function**;  
 **private** String **path**;  
 **private** String **cfgname** = **path**+**comtradeName**+**".cfg"**;  
 **private** String **datName** = **path**+**comtradeName**+**".dat"**;  
 **private** String **nameFile**;  
  
 **public** InputData(String function, String nameFile) {  
 **this**.**function** = function;  
 **this**.**nameFile** = nameFile;  
 }  
  
 **private** SampleValues **sv** = **new** SampleValues();  
 **private** RMSValues **rms** = **new** RMSValues();  
 **private** Filter **filter** = **new** Fourie(); *//фурье  
// private Filter filter = new MiddleValue(); //rms* **private** Logic **logic** = **new** Logic();  
  
  
 **public double** start(**double** setPoint, **int** numberFirstGraph, **int** numberSecondGraph) **throws** FileNotFoundException {  
 **if** (**function**.equals(**"Посчитать уставку"**)) {  
 **comtradeName** = **nameFile**;  
 **path** = **"C:\\Users\\maksimov\\Desktop\\Algoritms\\Опыты\\Конец линии\\"**;  
 **t** = **false**;  
 } **else** {  
 **comtradeName** = **nameFile**;  
 **path** = **"C:\\Users\\maksimov\\Desktop\\Algoritms\\Опыты\\Начало линии\\"**;  
  
 }  
  
 **cfgname** = **path**+**comtradeName**+**".cfg"**;  
 **datName** = **path**+**comtradeName**+**".dat"**;  
 **comtrCfg** = **new** File(**cfgname**);  
 **comtrDat** = **new** File(**datName**);  
 **filter**.setSv(**sv**); *//объект SV помещаем в объект filter,чтобы получать значения* **filter**.setRms(**rms**);*//объект rms помещаем в объект filter,чтобы устанавливать значения  
 //открываем cfg файл для получения коэф a и b для расчета y = ax+b* **br** = **new** BufferedReader(**new** FileReader(**comtrCfg**));  
 **int** lineNumber =0, count =0, numberData = 100;  
 **try** {  
 **while** ((**line**=**br**.readLine())!=**null**) {  
 System.***out***.println(**line**);  
 lineNumber++;  
 **if** (lineNumber ==2) {  
 *//получаем количество аналоговых сигналов во 2 строке cfg файла "4,3A,1D"* numberData = Integer.*parseInt*(**line**.split(**","**)[1].replaceAll(**"A"**,**""**));  
 *//создаем double " массивы " с размерностью равной количеству* **k1** = **new double**[numberData];  
 **k2** = **new double**[numberData];  
 }  
 *//коэф находятся на 3,4,5 строке это 5 и 6 элемент строки при парсинге* **if** (lineNumber>2 && lineNumber <numberData+3) {  
 **k1**[count] = Double.*parseDouble*(**line**.split(**","**)[5]);  
 **k2**[count] = Double.*parseDouble*(**line**.split(**","**)[6]);  
 count++;  
 };  
 }  
 count =0;  
 *//открываем data файл для получения значений* **br** = **new** BufferedReader(**new** FileReader(**comtrDat**));  
 **while** ((**line**=**br**.readLine())!=**null**) {  
 count++;  
 **if** ((count > 2000 && count < 5000)) {  
 **lineData** = **line**.split(**","**);  
 **if** (!**t**) {  
 **sv**.setPhA(Double.*parseDouble*(**lineData**[2]) \* **k1**[0] + **k2**[0]);  
 **sv**.setPhB(Double.*parseDouble*(**lineData**[3]) \* **k1**[1] + **k2**[1]);  
 **sv**.setPhC(Double.*parseDouble*(**lineData**[4]) \* **k1**[2] + **k2**[2]);  
 **sv**.setTime(Double.*parseDouble*(**lineData**[1]));  
 *//получаем значение уставки, отправляем функцию, чтобы рассчитать или не рассчитать уставку* setPoint = **filter**.calculate(**function**,setPoint);  
 *// ПЕРЕМЕННАЯ T = true при срабатывании, чтобы токи стали равными нулю, типо выключатель отключил их, если ток выше уставки* **t** = **logic**.process(**rms**,setPoint);  
 } **else** {  
 *// t = true, отключение КЗ* **sv**.setPhA(0.);  
 **sv**.setPhB(0.);  
 **sv**.setPhC(0.);  
 **sv**.setTime(Double.*parseDouble*(**lineData**[1]));  
 **rms**.setPhA(0);  
 **rms**.setPhB(0);  
 **rms**.setPhC(0);  
 Charts.*addDiscreteData*(0, **true**);  
 }  
 *//добавляем значения для графиков* Charts.*addAnalogData*(numberFirstGraph, 0, **sv**.getPhA());  
 Charts.*addAnalogData*(numberFirstGraph, 1, **sv**.getPhB());  
 Charts.*addAnalogData*(numberFirstGraph, 2, **sv**.getPhC());  
  
 Charts.*addAnalogData*(numberSecondGraph, 0, **rms**.getPhA());  
 Charts.*addAnalogData*(numberSecondGraph, 1, **rms**.getPhB());  
 Charts.*addAnalogData*(numberSecondGraph, 2, **rms**.getPhC());  
  
 }  
  
  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
  
 **return** setPoint; // для расчета уставки  
  
 }  
  
}

В классе Logic прописана логика срабатывания/несрабатывания защиты. В случае, если действующее значение тока превышает уставку по ступени ТО, на графике отображается срабатывание защиты в виде перехода сигнала с нулевого уровня в высокий, что соответствует присвоению экземпляру класса OutputData значения True..

**public class** Logic {  
  
 **private** OutputData **od** = **new** OutputData(); *//объект класса, содержит в себе значение дискретного сигнала* **boolean triper**;  
  
 **public boolean** process(RMSValues rms, **double** tripPoint) {  
 *//если хотя бы один ток больше, чем уставка, то происходит отключение* **if** (rms.getPhA() > tripPoint | rms.getPhB() > tripPoint | rms.getPhC() > tripPoint) {  
 *//отключение* **triper** = **od**.trip(**true**);  
 } **else** {  
 **triper** = **od**.trip(**false**);  
 }  
 **return triper**;  
 }  
  
}

**public class** OutputData {  
 **private boolean tripper** = **false**;  
 **public boolean** trip(**boolean** triper) {  
 **tripper** = triper;  
 Charts.*addDiscreteData*(0, **tripper**);  
 **return** triper;  
 }  
}

В классе Fourie в методе calculate() непосредственно осуществлен алгоритм работы фильтра Фурье. Контроль выборок за период осуществляется переменной counter, что необходимо для сдвига окна, которое реализует buffer[counter].

**public class** Fourie **implements** Filter {  
 **private double**[] **bufferPhA** = **new double**[80];  
 **private double**[] **bufferPhB** = **new double**[80];  
 **private double**[] **bufferPhC** = **new double**[80];  
 **private** SampleValues **sv**;  
 **private** RMSValues **rms**;  
 **private double A0A**;  
 **private double Ak1A**;  
 **private double Bk1A**;  
 **private double A0B**;  
 **private double Ak1B**;  
 **private double Bk1B**;  
 **private double A0C**;  
 **private double Ak1C**;  
 **private double Bk1C**;  
 **private int count** = 0;  
  
 **public double** calculate(String function, **double** setPoint) {  
 **int** period = 80; *//количество точек за период  
 //Алгоритм фурье постоянная составляющая + 1 гармоника  
 //расчет разницы между двумя значениями* **double** sumPhA = **sv**.getPhA() - **bufferPhA**[**count**];  
 *//расчет постоянной составляющей, возникает при КЗ. В норм. режиме равна нулю -> интеграл синусоиды = 0* **A0A** += sumPhA / period;  
 *//расчет cos и sin составляющей для первой гармоники* **Ak1A** += 2 \* (Math.*cos*(**count** \* 2 \* Math.***PI*** / period) \* sumPhA) / period;  
 **Bk1A** += 2 \* (Math.*sin*(**count** \* 2 \* Math.***PI*** / period) \* sumPhA) / period;  
 *//расчет действующего значения для 1 гармоники по cos и sin составляющей* **double** Ck1A = Math.*sqrt*((Math.*pow*(**Ak1A**, 2) + Math.*pow*(**Bk1A**, 2)) / 2);  
 *//суммарная составляющая* **double** xA = Math.*sqrt*(Math.*pow*(Ck1A, 2) + Math.*pow*(**A0A**, 2));  
 **bufferPhA**[**count**] = **sv**.getPhA();  
  
 **double** sumPhB = **sv**.getPhB() - **bufferPhB**[**count**];  
 **A0B** += sumPhB / period;  
 **Ak1B** += 2 \* (Math.*cos*(**count** \* 2 \* Math.***PI*** / period) \* sumPhB) / period;  
 **Bk1B** += 2 \* (Math.*sin*(**count** \* 2 \* Math.***PI*** / period) \* sumPhB) / period;  
 **double** Ck1B = Math.*sqrt*((Math.*pow*(**Ak1B**, 2) + Math.*pow*(**Bk1B**, 2)) / 2);  
 **double** xB = Math.*sqrt*(Math.*pow*(Ck1B, 2) + Math.*pow*(**A0B**, 2));  
 **bufferPhB**[**count**] = **sv**.getPhB();  
  
 **double** sumPhC = **sv**.getPhC() - **bufferPhC**[**count**];  
 **A0C** += sumPhB / period;  
 **Ak1C** += 2 \* (Math.*cos*(**count** \* 2 \* Math.***PI*** / period) \* sumPhC) / period;  
 **Bk1C** += 2 \* (Math.*sin*(**count** \* 2 \* Math.***PI*** / period) \* sumPhC) / period;  
 **double** Ck1C = Math.*sqrt*((Math.*pow*(**Ak1C**, 2) + Math.*pow*(**Bk1C**, 2)) / 2);  
 **double** xC = Math.*sqrt*(Math.*pow*(Ck1C, 2) + Math.*pow*(**A0C**, 2));  
 **bufferPhC**[**count**] = **sv**.getPhC();  
 *//устанавливаем действующие значения для логики* **rms**.setPhA(xA);  
 **rms**.setPhB(xB);  
 **rms**.setPhC(xC);  
  
  
 **if** (function.equals(**"Посчитать уставку"**)) {  
 **if** (**rms**.getPhA() >= setPoint) {  
 setPoint = **rms**.getPhA();  
 }  
 **if** (**rms**.getPhB() >= setPoint) {  
 setPoint = **rms**.getPhB();  
 }  
 **if** (**rms**.getPhC() >= setPoint) {  
 setPoint = **rms**.getPhC();  
 }  
 } **else** {  
 Charts.*addAnalogData*(3, 3, setPoint);  
 }  
 **rms**.setTime(**sv**.getTime());  
 **count**++;  
 **if** (**count** == period) {  
  
 **count** = 0;  
 }  
  
 **return** setPoint;  
  
 }  
  
  
 **public** SampleValues getSv() {  
 **return sv**;  
 }  
  
 **public void** setSv(SampleValues sv) {  
 **this**.**sv** = sv;  
 }  
  
 **public** RMSValues getRms() {  
 **return rms**;  
 }  
  
 **public void** setRms(RMSValues rms) {  
 **this**.**rms** = rms;  
 }  
  
}

Filter представляет собой интерфейс с отображением всех его методов.

**public interface** Filter {  
 **public double** calculate(String function, **double** setPoint);  
 **public** SampleValues getSv();  
 **public void** setSv(SampleValues sv);  
 **public** RMSValues getRms();  
 **public void** setRms(RMSValues rms);  
}

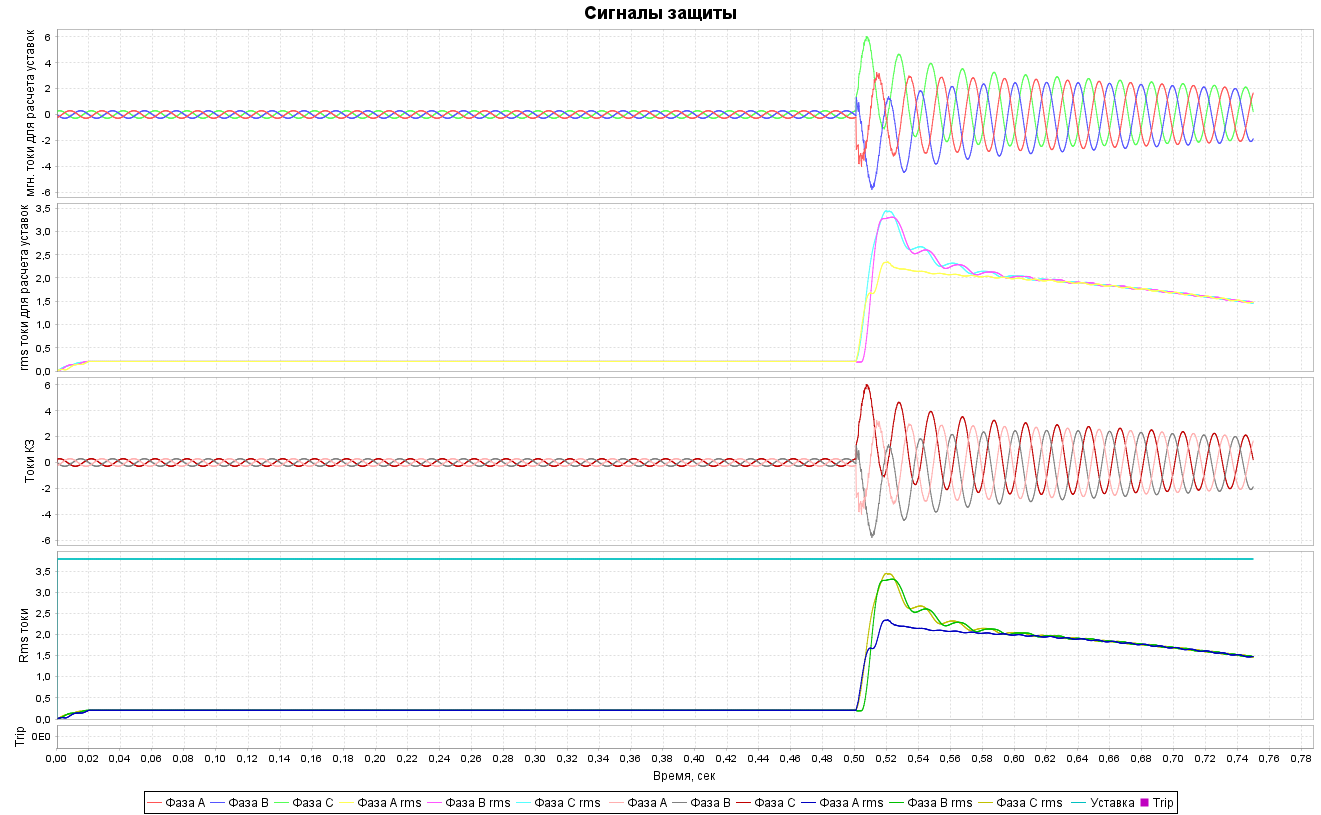
Классы SampleValues и RMSValues идентичны и содержат в себе методы геттеры и сеттеры для обращения и записи значений.

**public class** SampleValues {  
 **private double phA**;  
 **private double phB**;  
 **private double phC**;  
 **private double time**;  
 *//alt+insert* **public double** getPhA() {  
 **return phA**;  
 }  
  
 **public double** getPhB() {  
 **return phB**;  
 }  
  
 **public double** getPhC() {  
 **return phC**;  
 }  
  
 **public void** setPhA(**double** phA) {  
 **this**.**phA** = phA;  
 }  
  
 **public void** setPhB(**double** phB) {  
 **this**.**phB** = phB;  
 }  
  
 **public void** setPhC(**double** phC) {  
 **this**.**phC** = phC;  
 }  
  
  
 **public double** getTime() {  
 **return time**;  
 }  
  
 **public void** setTime(**double** time) {  
 **this**.**time** = time;  
 }  
}

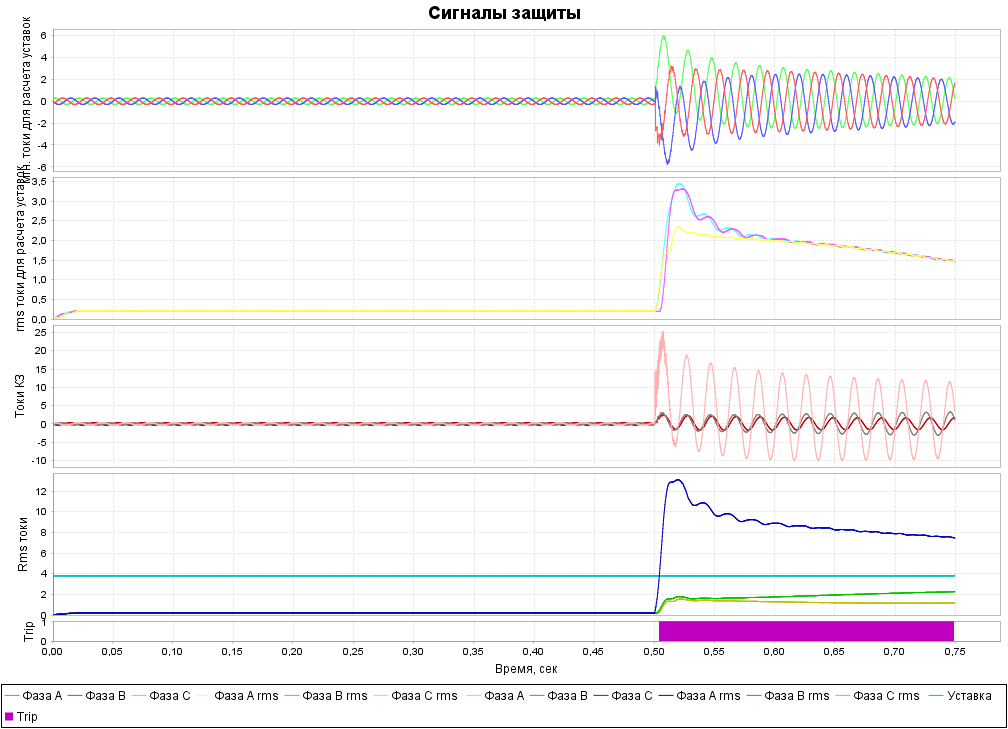
**public class** RMSValues {  
 **private double phA**;  
 **private double phB**;  
 **private double phC**;  
 **private double time**;  
 **public double** getPhA() {  
 **return phA**;  
 }  
  
 **public double** getPhB() {  
 **return phB**;  
 }  
  
 **public double** getPhC() {  
 **return phC**;  
 }  
  
 **public void** setPhA(**double** phA) {  
 **this**.**phA** = phA;  
 }  
  
 **public void** setPhB(**double** phB) {  
 **this**.**phB** = phB;  
 }  
  
 **public void** setPhC(**double** phC) {  
 **this**.**phC** = phC;  
 }  
  
  
 **public double** getTime() {  
 **return time**;  
 }  
  
 **public void** setTime(**double** time) {  
 **this**.**time** = time;  
 }  
}

**Результаты:**

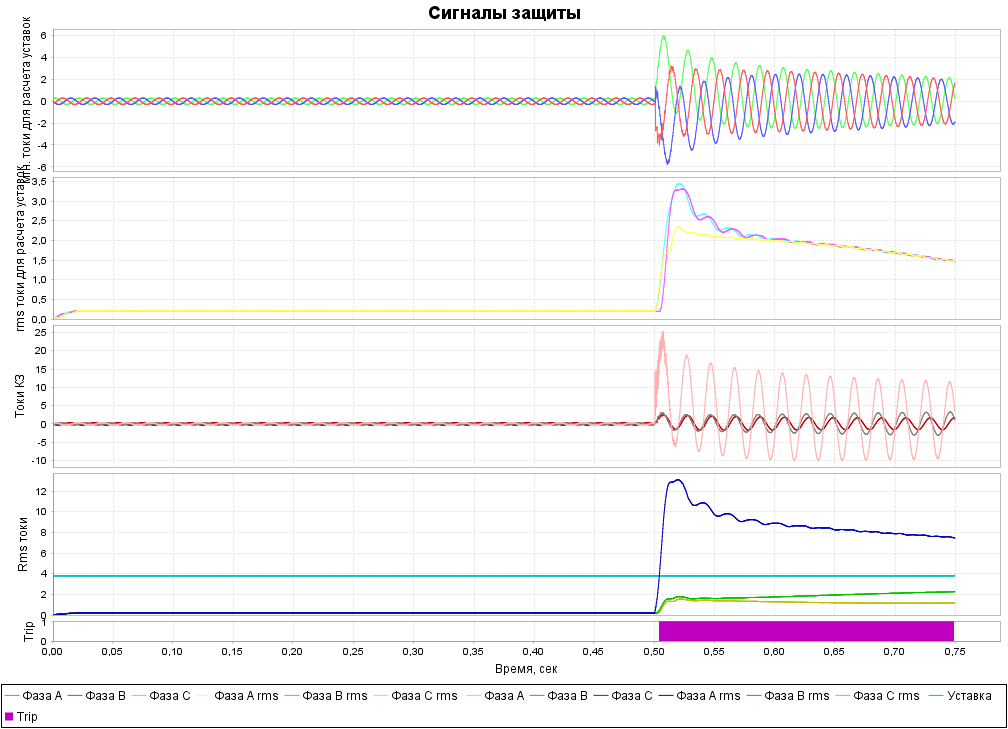
* Несрабатывание защиты при трехфазном КЗ в конце защищаемого участка.

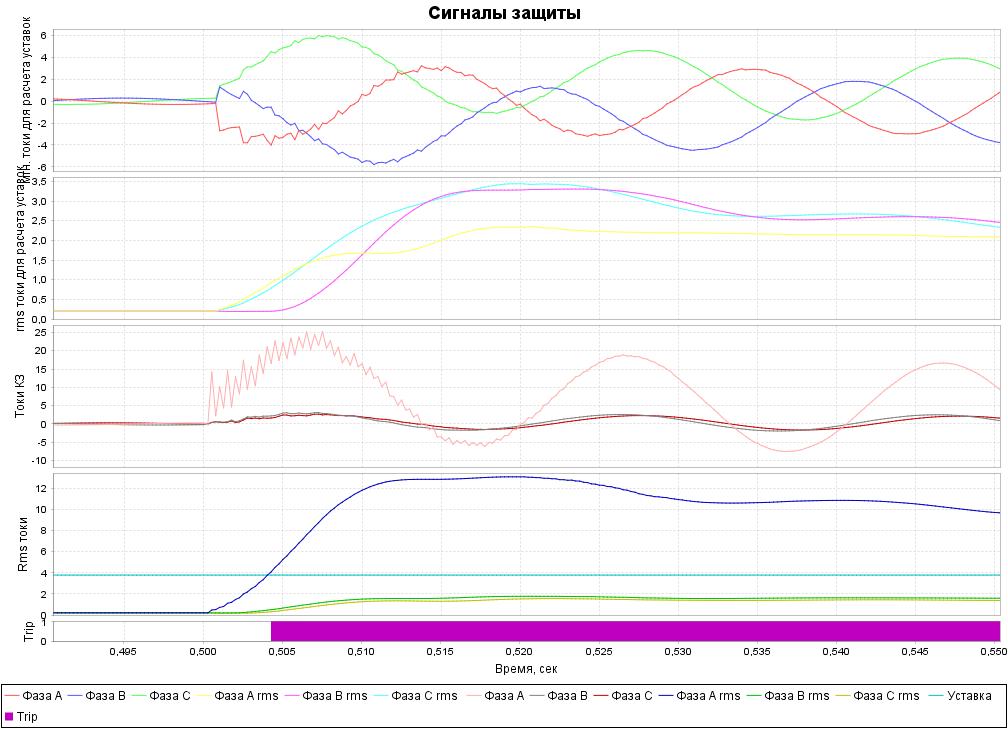


* Cрабатывание защиты при двухвазном КЗ ВС в начале защищаемого участка с отключением.



* Cрабатывание защиты при двухвазном КЗ ВС в начале защищаемого участка без отключения.





* Диаграмма взаимодействия объектов.



**Вывод по работе:** в ходе выполнения данной лабораторной работы разработали алгоритм для считывания данных с файлов, их обработки. Осуществили алгоритм фильтра Фурье для обработки сигнала и работы ТО.