**СОДЕРЖАНИЕ**

|  |  |  |
| --- | --- | --- |
| Введение ……………………………………………………………………… | | 3 |
| 1 Обоснование необходимости определения автокорреляционных и  взаимокорреляционных функций сигналов. Патентный поиск ......…….. | | 5 |
| 1.1 Обоснование необходимости определения автокорреляционных и  взаимокорреляционных функций сигналов…………………………… | | 5 |
| 1.2 Патентно-информационный поиск ……………………………………. | | 6 |
| 1.2.1 Способ корреляционной обработки широкополосных сигналов … | | 6 |
| 1.2.2 Корреляционный анализатор………………………………………... | | 8 |
| 1.2.3 Параллельный знаковый коррелометр …………………………….. | | 12 |
| 1.2.4 Многоканальный знаковый коррелометр ………………………….. | | 13 |
| 2 Анализ аппаратурных средств определения автокорреляционных и  взаимокорреляционых функций сигналов (аналоговых и цифровых  коррелометров) …………………………………………………………….. | | 16 |
| 2.1 Общие сведения об аппаратурных средствах определения  корреляционных характеристик сигналов …………………………….. | | 16 |
| 2.2 Аналоговые коррелометры …………………………………………….. | | 17 |
| 2.3 Аналоговые коррелометры с дискретизацией времени ………………. | | 21 |
| 2.4 Аналого-цифровые коррелометры …………………………………….. | | 24 |
| 2.5 Цифровые коррелометры ………………………………………………. | | 25 |
| 3 Разработка алгоритмических и технических требований к  программному анализатору корреляционных характеристик  радиотехнических сигналов ……………………………………………….. | | 28 |
| 3.1 Общие требования ……………………………………………………… | | 28 |
| 3.2 Специфические требования ……………………………………………. | | 30 |
| 3.3 Функциональные требования ………………………………………….. | | 31 |
| 4 Технико-экономическое обоснование себестоимости программного  обеспечения для анализа корреляционных характеристик  радиотехнических сигналов ..……………………………………………… | | 33 |
| 4.1 Характеристика разрабатываемого ПО ……………………………….. | | 33 |
| 4.2 Определение объема и трудоемкости ПО …………………………….. | | 33 |
| 4.3 Расчет себестоимости ПО ……………………………………………… | | 35 |
| 5 Функциональный анализ процессов определения корреляционных  функций сигналов ………………………………………………………......  6 Разработка общего алгоритма моделирования процессов определения  корреляционных функций сигналов. Выбор системы  программирования и разработка программы ……………………………..  6.1 Разработка общего алгоритма моделирования процессов  определения корреляционных функций сигналов …………………….  6.2 Выбор системы программирования и разработка программы ……….  7 Методика применения ПАК для определения и анализа  корреляционных функций …………………………………………………. | | 41  43  43  47  50 |
| 7.1 Методика работы с программой ……………………………………….. | | 50 |
| 7.1.1 Интерфейс программы и общие положения ……………………...... | | 50 |
| 7.1.2 Создание нового сигнала ……………………………………………. | | 51 |
| 7.1.3 Выбор сигнала для расчета автокорреляционной функции ……….  7.1.4 Построение автокорреляционной функции ……………….……….  7.1.5 Выбор сигнала для расчета взаимокорреляционной функции ……  7.1.6 Построение взаимокорреляционной функции ……………..……….  7.1.7 Справочная информация ……………………………………………. | | 53  53  55  56  58 |
| Заключение …………………………………………………………………… | 60 |
| Список использованных источников ……………………………………….. | 61 |
| Приложение А (обязательное) Алгоритм расчёта построенных корреляционных функций ................................................................................ | 62 |
| Приложение Б (обязательное) Алгоритм пошагового расчёта корреляционных функций …………………………………………………… | 64 |
| Приложение В (обязательное) Исходный код программного анализатора | 65 |
| Ведомость документов ………………………………………………………. | 111 |

**ВВЕДЕНИЕ**

Совершенствование образовательных технологий является важнейшим условием технологического прогресса и повышения квалификации специалистов во всех областях науки. Главными задачами улучшения образовательных технологий являются:

1 Уменьшение количества ресурсов, затрачиваемых на проведение исследовательских и научных работ.

2 Сокращение количества времени, затрачиваемого на освоение необходимой информации.

3 Предоставление полной и наглядной информации для понимания и изучения материала.

4 Обеспечение наиболее безопасных условий для изучения.

Для обеспечения всех этих требований следует разрабатывать специальное программное обеспечение, которое позволит многократно проводить исследования без всякого риска для здоровья человека. С помощью такого программного обеспечения сокращается количество времени, которое при выполнении реального исследования пришлось бы потратить на настройку или обеспечение необходимых условий для проведения эксперимента. Такое программное обеспечение можно снабдить необходимой литературой и замечаниями, возникающими в ходе исследования, помогающими наиболее полно осознать материал.

Задачей дипломной работы является разработка программного анализатора корреляционных характеристик радиотехнических сигналов, позволяющего создавать произвольные радиотехнические сигналы, рассчитывать и динамически отображать их корреляционные характеристики в зависимости от заданного шага. Целью разработки данного программного обеспечения является совершенствование процесса обучения и исследования корреляционных функций сигналов.

Для решения описанной выше задачи необходимо рассмотреть и решить следующие вопросы:

1 Обосновать необходимость определения автокорреляционных и взаимокорреляционных функций сигналов.

2 Проанализировать аппаратурные средства для определения авто-

корреляционных и взаимокорреляционных функций сигналов(аналоговых и цифровых коррелометров).

3 Разработать алгоритмические и технические требования к

программному анализатору

4 Провести функциональный анализ процессов определения корреля-

ционных функций сигналов.

5 Разработать общий алгоритм моделирования процессов определения корреляционных функций сигналов, выбрать систему программирования и разработать программу. Описать методику применения ПАК для определения и анализа корреляционных функций.

**1 ОБОСНОВАНИЕ НЕОБХОДИМОСТИ ОПРЕДЕЛЕНИЯ**

**АВТОКОРРЕЛЯЦИОННЫХ И ВЗАИМОКОРРЕЛЯЦИОННЫХ**

**ФУНКЦИЙ. ПАТЕНТНО-ИНФОРМАЦИОННЫЙ ПОИСК**

* 1. **Обоснование необходимости определения автокорреляционных**

**и взаимокорреляционных функций**

Решение многих задач радиоэлектроники, автоматики, связи, физики, акустики, гидроакустики, биологии, медицины и других областей науки и техники связано с широким применением корреляционного анализа. Особенно корреляционный анализ востребован в системах, предназначенных для решения задач обнаружения сигналов на фоне помех в радиолокации и радионавигации.

Так же при решении многих задач оптимальной обработки сигналов возникает потребность определять степень подобия различных сигналов или сигнала и его копии, сдвинутой на определенное время.

В результате решения этой задачи в рамках линейных систем синтезирован оптимальный обнаружитель сигналов, структура которого содержит согласованный фильтр или корреляционный приемник. Алгоритм работы подобного обнаружителя предполагает вычисление функции:

где s(t) – полезный сигнал;

Т – интервал времени, в пределах которого осуществляется обработка смеси сигнала и шума;

W0 – Энергетический спектр шума;

ε(t, τ) – отраженный от цели сигнал, представляющий собой сумму

задержанного на *τ* полезного сигнала и шума *n(t)*, т.е.

Здесь *θ* – случайная величина, причем θ равняется нулю, если полезный сигнал отсутствует, и *θ* равняется единице, если сигнал присутствует.

Задача обнаружителя – определить значение *θ*. Для этого результат вычисления функции сравнивается с порогом *h*. Если *q(T, τ)* имеет значение больше порогового, то *θ* равно единице, в противном случае нулю.

Как видно из рассмотренного алгоритма, оптимальный обнаружитель сигналов при отсутствии шумов предусматривает расчёт функции, которая в общем случае имеет вид:

Знание корреляционной функции позволяет определить степень подобия сигнала и его копии.

Для измерения корреляционных функций применяют специальные приборы, получившие название коррелометров или коррелографов. При строгом дифференцировании, под коррелометрами понимаются приборы, измеряющие отдельные ординаты корреляционной функции, а под коррелографами – устройства, регистраторы которых дают график всей функции корреляции. Однако чаще название коррелометр служит общим для всех измерителей корреляционных функций.

Таким образом, темами для патентно-информационного поиска будут являться: аппаратная реализация коррелометров (коррелографов) и метод измерения корреляционных функций.

**1.2 Патентно-информационный поиск**

Патентно-информационный поиск проводился в открытых базах патентов по темам, относящимся к корреляционному анализу радиотехнических сигналов. В данном разделе представлены некоторые из этих патентов.

1.2.1Способ корреляционной обработки широкополосных

сигналов

Грант RU2073954C1, 20-10-1992 г.

Авторы: Чугаева В.И.

Изобретение относится к области радиотехники и может быть использовано в системах связи. Техническим результатом является повышение помехоустойчивости. Способ основан на перемножении в каждом из *N* каналов входной смеси с соответствующим синхронным опорным сигналом, в каждом из *N* каналов выделенную огибающую сравнивают со вторым порогом, далее отфильтрованный сигнал перемножают с тем же синхронным опорным сигналом, за счет чего восстанавливают принятый данным каналом сигнал, который вычитают из входной смеси на входе каналов, в которых не обнаружено временное превышение двух порогов.

Предлагаемое изобретение относится к радиотехнике и может найти применение в системах связи с широкополосными сигналами с кодовым разделением каналов.

Устройство-прототип содержит *N*-каналов обработки, каждый из которых состоит из последовательно соединенных перемножителя (ПРМ), полосового фильтра (ПФ), амплитудного детектора (АД), блока сравнения с порогом (СхСр), выход которого соединен с соответствующим входом решающего блока (РБ), выход которого является выходом устройства. Один из выходов генератора копии сигнала (ГС) соединен с вторыми входами *N*-перемножителей, первые входы всех *N* каналов соединены между собой и являются входом устройства.

Структурная схема устройства, реализующего способ-прототип представлена на рисунке *1.1*.

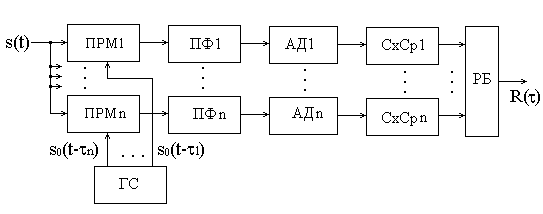


Рисунок 1.1 – Cхема устройства cтруктурная

Работает устройство-прототип следующим образом: входной широкополосный сигнал поступает одновременно на первые входы *N* перемножителей ПРМ1 – ПРМ*N*, на вторые входы которых от генератора сигналов подаются копии сигнала *s(t)*, отличающиеся между собой задержками, то есть сигналы ,…, . Результат перемножения (свертки) фильтруется в полосовом фильтре, детектируется в амплитудном детекторе и сравнивается с порогом в блоке сравнения. Решающий блок фиксирует канал, в котором превышен порог.

Способ-прототип, реализованный в устройстве, представленном на рисунке *1.1*, заключается в перемножении в каждом из *N* каналов входной смеси с опорным сигналом, фильтрации результата перемножения, детектировании, сравнении выделенной огибающей напряжения с порогом и фиксации канала, в котором превышен порог. При этом опорные сигналы *N* каналов отличаются задержками.

Недостатком способа-прототипа является низкая помехоустойчивость к структурным помехам.

Для устранения указанного недостатка в способе корреляционной обработки широкополосных сигналов, заключающемся в перемножении в каждом из *N* каналов входной смеси с соответствующим синхронным опорным сигналом, фильтрации результата перемножения, его детектировании и сравнении выделенной огибающей напряжения с первым порогом, в каждом из *N* каналов выделенную огибающую сравнивают со вторым порогом, далее отфильтрованный сигнал перемножают с тем же синхронным опорным сигналом. В результате восстанавливают принятый данным каналом сигнал, который вычитают из входной смеси на входе каналов, в которых не обнаружено одновременное превышение двух порогов.

Заявляемый способ основан на следующей последовательности действий над сигналом.

В предлагаемом способе представляется последовательность действий над входной смесью в корреляторе базовой станции системы связи с кодовым разделением каналов, в которой адресным признаком абонентов является код (структура) псевдослучайной последовательности.

Входную смесь, содержащую сигналы *N* абонентов сотовой системы связи с кодовым разделением перемножают в каждом из каналов с соответствующим синхронным опорным сигналом. Результат перемножения фильтруют, детектируют, выделенную огибающую напряжения одновременно сравнивают с первым (низким) и вторым (высоким) порогом.

Принятый в каждом из *N* каналов сигнал восстанавливают за счет перемножения отфильтрованного сигнала с тем же синхронным опорным сигналом. Восстановленные сигналы тех каналов, в которых зафиксировано одновременное превышение двух порогов, вычитают из входной смеси в тех каналах, в которых не обнаружено одновременное превышение двух порогов.

1.2.2 Корреляционный анализатор

Грант RU2227321C2, 17-07-2002 г.

Авторы: Аванесян Г.Р, Беспалов А.А.

Изобретение относится к области вычислительной техники и может быть использовано для анализа случайных процессов. Техническим результатом является расширение функциональных возможностей. Анализатор содержит коррелятор, блок вычитания, блок памяти, блок усреднения и блок управления.

Изобретение относится к специализированным вычислительным средствам и служит для сравнительного анализа случайных процессов, разнесенных во времени.

Известен корреляционный анализатор, построенный по классической схеме параллельного корреляционного анализа и содержащий первый и второй входные блоки, линию задержки, *n* перемножителей, *n* усреднителей и дисплей, причем первые входы *n* перемножителей объединены и подключены к выходу первого входного блока, а вторые входы *n* перемножителей подключены к соответствующим *n* отводам линии задержки, включая отвод с нулевой задержкой, вход которой соединен с выходом второго входного блока, входами устройства являются входы входных блоков, выходы *n* перемножителей подключены к соответствующим входам дисплея через *n* усреднителей.

Принцип действия коррелятора-аналога требует одновременного присутствия на двух его входах исследуемых сигналов. Таким образом, проводить сравнительный анализ сигналов, действующих в различные интервалы времени, при помощи данного устройства не представляется возможным, что и относится к недостатку устройства.

Наиболее близким по технической сущности к предлагаемому анализатору является корреляционный анализатор, содержащий два коррелятора, блок вычитания, блок регулируемой задержки, блок постоянной задержки, усилитель и два управляемых фильтра, причем выходы первого и второго корреляторов соединены с соответствующими входами блока вычитания, выход которого через усилитель соединен с управляющим входом блока регулируемой задержки, выход которого непосредственно и через блок постоянной задержки соответственно соединен с первыми входами первого и второго корреляторов, вторые входы которых соединены с выходом первого управляемого фильтра, выход второго управляемого фильтра соединен с информационным входом блока регулируемой задержки, информационные входы первого и второго управляемых фильтров являются соответственно первым и вторым входами устройства, управляющие входы первого и второго управляемых фильтров объединены и подключены к выходу усилителя.

Несмотря на то, что в состав анализатора-прототипа входят два независимых коррелятора, общая структура анализатора не позволяет проводить сравнительный анализ сигналов, действующих не одновременно, так как не предусмотрена возможность запоминания информации о ранее действующем сигнале.

Недостаток корреляционного анализатора-прототипа заключается в отсутствии функциональной возможности совместной обработки сигналов, действующих в различные моменты времени, с целью определения степени их подобия.

Технический результат, достигаемый при использовании настоящего изобретения, состоит в возможности количественного определения степени подобия сигналов, действующих в различные моменты времени.

Технический результат достигается тем, что в известный корреляционный анализатор, содержащий коррелятор и блок вычитания, первый вход которого соединен с выходом коррелятора, согласно изобретению введены блок памяти, блок усреднения и блок управления, причем второй вход блока вычитания соединен с выходом блока памяти, информационный вход которого подключен к выходу коррелятора, адресный вход которого объединен с адресным входом блока памяти и подключен к адресному выходу блока управления, стробирующий выход которого подключен к стробирующему входу коррелятора, информационный вход которого является информационным входом анализатора, выходом которого является выход блока усреднения, информационный вход которого соединен с выходом блока вычитания, а тактовый вход соединен с соответствующим выходом блока управления, обнуляющий вход которого объединен с обнуляющими входами коррелятора и блока усреднения и является обнуляющим входом анализатора, первым и вторым управляющими входами которого служат соответствующие входы блока управления, входы управления блока памяти соединены с соответствующими выходами блока управления.

Структурная схема блока управления устройства представлена на *рисунке* *1.2*.

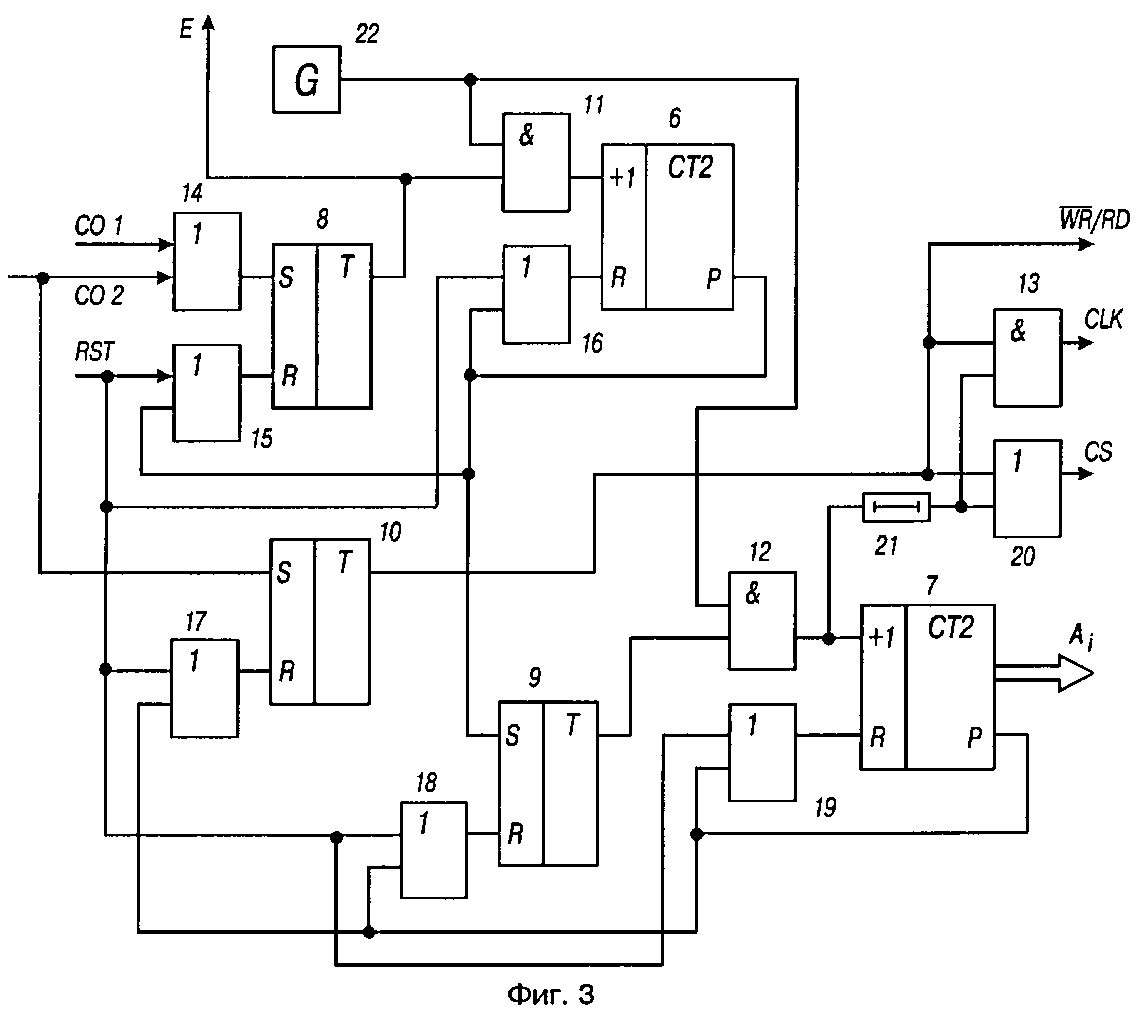


Рисунок 1.2 – Блок управления устройства

Блок управления (*рисунок* *1.2*) содержит счетчики *6* и *7*, триггеры *8*, *9* и *10*, элементы *2И 11*, *12* и *13*, элементы *2ИЛИ 14*, *15*, *16*, *17*, *18*, *19* и *20*, элемент *21* задержки и генератор *22* тактовых импульсов. Первый и второй входы элемента *2ИЛИ 14* являются соответственно первым *С01* и вторым *С02* управляющими входами блока *5*, обнуляющим входом *RST* которого служат объединенные первые входы элементов *2ИЛИ 15*, *16*, *17*, *18* и *19*, выходы которых подключены к обнуляющим входам триггера *8*, счетчика *6*, триггера *10*, триггера *9*, счетчика *7* соответственно, вторые входы элементов *2ИЛИ 15* и *16* объединены и подключены к выходу переполнения счетчика *6*, вторые входы элементов *2ИЛИ 17*, *18* и *19* объединены и подключены к выходу переполнения счетчика *7*, разрядные выходы которого являются адресным выходом *А* блока *5* управления, стробирующим выходом *Е* которого является выход триггера *8*, соединенный с первым входом элемента *2И 11*, второй вход которого объединен с первым входом элемента *2И 12* и подключен к выходу генератора *22* тактовых импульсов, выходы элементов *2И 11* и *12* подключены к суммирующим входам соответственно счетчиков *6* и *7*, выход переполнения счетчика *6* соединен с входом установки в единицу триггера *9*, выход которого соединен с вторым входом элемента *2ИЛИ 12*, к выходу которого подключен вход элемента *21* задержки, выход которого соединен с объединенными первыми входами элементов *2ИЛИ 20* и *2 И 13*, вторые входы которых также объединены и подключены к выходу триггера *10*, вход установки в единицу которого объединен с вторым входом элемента *2ИЛИ 14*, выход которого соединен с входом установки в единицу триггера *8*, выход триггера *10* и выход элемента *2ИЛИ 20* являются соответственно выходами и *CS* блока управления, тактовым выходом *CLK* которого является выход элемента *2И 13*. Принцип действия корреляционного анализатора основан на поочередном вычислении автокорреляционных функций *RXX(τi)* и *RYY(τi)* соответственно сигналов *X(t)* и *Y(t)*, стационарных и эргодических, действующих в различные моменты времени, запоминании и затем сравнении полученных значений. Первая из вычисленных функций запоминается в специально выделенном под нее блоке памяти и может храниться достаточно долго (в реальных пределах) до появления второго сигнала. Следовательно, временные ограничения на относительный сдвиг процессов *Х(t)* и *Y(t)* в практических задачах могут вообще отсутствовать, а теоретически определяются максимально возможным временем хранения информации без разрушения в конкретно используемых запоминающих структурах. Вторая автокорреляционная функция, в процессе анализа извлекается непосредственно из ячеек памяти (накапливающих сумматоров, например) самого коррелятора.

1.2.3 Параллельный знаковый коррелометр

Грант RU2252450C2, 20-08-2002 г.

Авторы: Якимов В.Н.

Изобретение относится к области вычислительной техники и может быть использовано в измерительных системах, предназначенных для анализа характеристик стохастической взаимосвязи случайных процессов.

Техническим результатом является упрощение структуры и повышение надежности коррелометра.

Указанный результат достигается за счет того, что коррелометр содержит первый *1* и второй *2* генераторы случайных равномерно распределенных сигналов, первый *3* и второй *4* компараторы, формирователь *5* коротких импульсов, первый *6* и второй *12* двоичные счетчики, дешифратор *7*, *D*-триггер *8*, первый *9* и второй *17 RS*-триггеры, элемент *И 10*, элемент *ИСКЛЮЧАЮЩЕЕ ИЛИ 11*, реверсивный счетчик *13*, генератор *14* тактовых импульсов, делитель *15* с перестраиваемым коэффициентом деления, распределитель импульсов *16*, группу, включающую *М* блоков *181*-*18М* синхронизации, группу, включающую *М* распределителей импульсов *191-19М*, *N* (*М*-1)-входовых элементов *ИЛИ 201-20N* и N блоков *211-21N* вычисления ординат корреляционной (взаимной корреляционной) функции.

На *рисунке* *1.3* представлена структурная схема коррелометра:

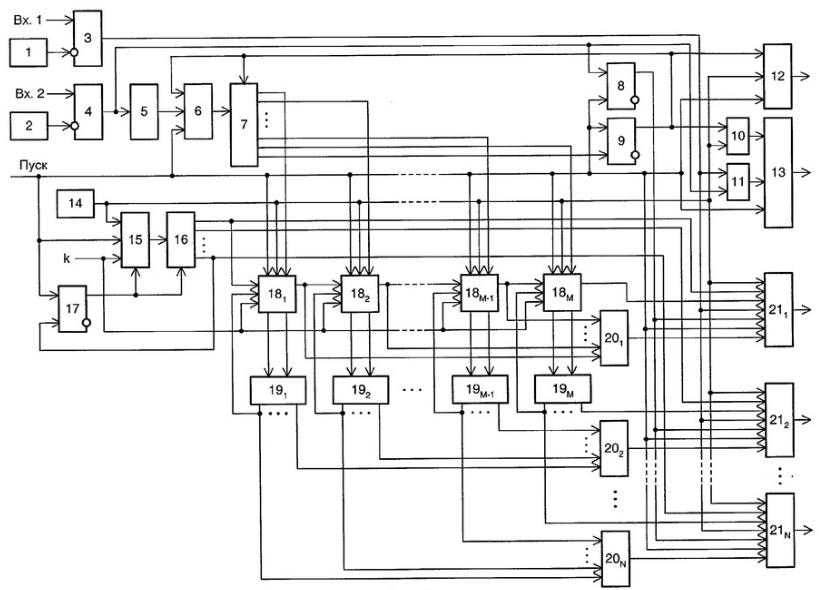


Рисунок 1.3 – Cхема коррелометра cтруктурная

Коррелометр может измерять корреляционную *Rхх(τ)*и взаимную корреляционную *Rxy(τ)* функции центрированных случайных сигналов. Его работа основана на знаковом методе измерения оценок.

1.2.4 Многоканальный знаковый коррелометр

Грант RU2177637C2, 18-01-1999 г.

Авторы: Якимов В.Н.

Изобретение относится к измерительной технике и может быть использовано в измерительных системах, предназначенных для анализа характеристик стохастической взаимосвязи случайных процессов. Технический результат заключается в повышении точности получения оценок корреляционной и взаимной корреляционной функций и расширении функциональных возможностей за счет анализа случайных процессов с любым законом распределения. Устройство содержит первый и второй компараторы, формирователь коротких импульсов, первый распределитель импульсов, группу из *N* двоичных счетчиков, генератор тактовых импульсов, группу из *N* *RS*-триггеров, первый и второй генераторы случайных равномерно распределенных сигналов, инвертор, первый, второй и третий элементы *И*, элемент *ИСКЛЮЧАЮЩЕЕ ИЛИ*, реверсивный счетчик, делитель с перестраиваемым коэффициентом деления, второй распределитель импульсов, *RS*-триггер и *М* блоков обработки временных интервалов. Структурная схема коррелометра представлена на *рисунке* *1.4*.

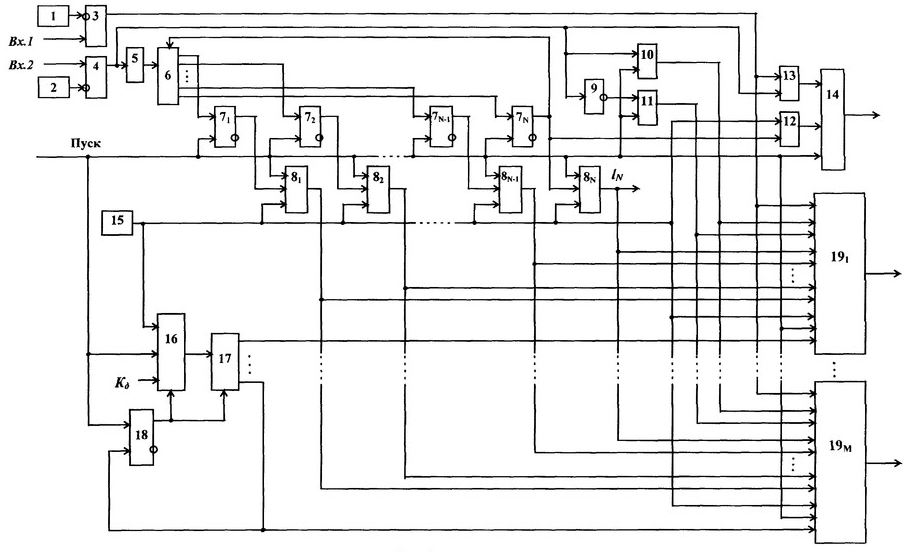


Рисунок 1.4 – Схема коррелометра cтруктурная

Проделанный патентно-информационный поиск позволяет проанализировать текущие решения по данной теме, определить задачи, которые следует решить при разработке программного анализатора корреляционных характеристик радиотехнических сигналов, и оптимальный путь их решения, исходя из стратегий их решения в рассмотренных патентах.

**2 АНАЛИЗ АППАРАТУРНЫХ СРЕДСТВ ОПРЕДЕЛЕНИЯ**

**АВТОКОРРЕЛЯЦИОННЫХ И ВЗАИМОКОРРЕЛЯЦИОННЫХ**

**ФУНКЦИЙ СИГНАЛОВ (АНАЛОГОВЫХ И ЦИФРОВЫХ**

**КОРРЕЛОМЕТРОВ)**

Анализ основных проблем, связанных с построением программного анализатора корреляционных характеристик радиотехнических сигналов, выполним на примере построения аппаратурных коррелометров. Цель анализа − получить сведения концептуального характера для решения основных вопросов дипломной работы.

**2.1 Общие сведения об аппаратурных средствах определения**

**корреляционных характеристик сигналов**

Основным аппаратурным средством определения корреляционных характеристик сигналов является коррелометр. Структурная и функциональная организация коррелометров зависит от множества факторов: принципа действия, формы представления и эффективной полосы частот спектров, исследуемых сигналов, вида регистрирующего устройства и др. Именно эти факторы положены в основу классификации коррелометров [1].

В соответствии с принципом действия различают коррелометры, основанные на методах умножения, суммирования (вычитания) и возведения в квадрат, аппроксимации корреляционной функции суммой членов разложения её в ряд по ортогональным функциям, знаковой корреляции, отображения диаграммы рассеивания и т.п.

В зависимости от формы представления исследуемых сигналов, схемотехнического решения и элементной базы коррелометры делят на аналоговые, аналого-цифровые и цифровые.

Коррелометры, представляющие собой аналоговые анализаторы, реализуют определение корреляционных функций по методу умножения. В связи с развитием цифровых методов обработки сигналов в настоящее время доминирующее положение занимают коррелометры, в которых используются элементы цифровой техники в сочетании с аналоговыми элементами. Это расширяет круг осуществляемых методов измерения, позволяет получать рациональные аппаратурные решения, делает возможным применение современной элементной базы (интегральных схем). В таких коррелометрах в качестве регистрирующего устройства используют цифровые индикаторы.

По диапазону граничных частот исследуемых процессов коррелометры делятся на инфранизкочастотные, низкочастотные, высокочастотные и сверхвысокочастотные. Исследуемая информация (процессы, сигналы), как правило, представляются в виде напряжений. Если информация записана на соответствующих носителях, в структуре коррелометра имеется специальное устройство, преобразующее различные виды записи информации в напряжение.

Рассмотрим более подробно структурные и функциональные особенности коррелометров.

**2.2 Аналоговые коррелометры**

Рассмотрим аналоговый коррелометр, работающий по методу умножения. Коррелометр, работающий по этому методу, выполняет действия, необходимые для получения оценок корреляционной и взаимной корреляционной функций в соответствии с выражением:

; (2.1)

; . (2.2)

Коррелометр осуществляет относительный сдвиг (задержку) сигнала на время , перемножает напряжения двух соответствующих сигналов (сдвинутого и несдвинутого), усредняет полученное произведение в течение определенного интервала времени.

Структурная схема двухканального коррелометра, предназначенного для вычисления АКФ и ВКФ двух сигналов представлена на *рисунке 2.1*.

При определённом временном сдвиге  операции, производимые коррелометром, дают одно значение  АКФ или одно значение  или  ВКФ.

Для получения всей кривой подобные операции повторяются при различных величинах задержки , причем изменение  может быть как

дискретным, так и непрерывным.

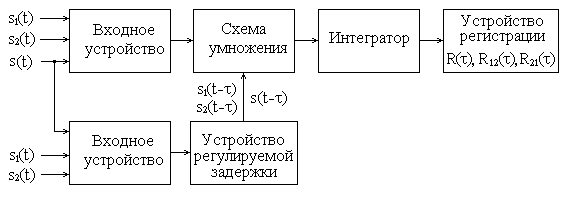


Рисунок 2.1 – Схема двухканального коррелометра cтруктурная

Таким образом, коррелометр, представленный на *рисунке 2.1*, осуществляет последовательный анализ и вычисление. При этом поочерёдно устанавливаются  величин *τ0* до *τmax = nτ0*и в каждой точке, то есть при каждой величине *τi = iτ0*, автоматически вычисляется соответствующая корреляционная функция.

Возможно реализовать одновременное (параллельное) вычисление всех  значений корреляционных функций при помощи многоканального коррелометра, представленного на *рисунке 2.2*.

Регистриру-

ющее устройство

*n*-й канал

Схема умножения

Усредняющее устройство

*i*-й канал

Усредняющее устройство

Схема умножения

2-й канал

Усредняющее устройство

Схема умножения

1-й канал

Усредняющее устройство

Схема умножения

0-й канал

Схема умножения

Усредняющее устройство

*s*0 (*t*)

Входное устройство

*sn* (*t*)

Входное устройство

τ0

τ0

τ0

τ0

Рисунок 2.2 – Коррелометр многоканальный

В данном устройстве выходной сигнал (оценка корреляционной функции) при непрерывном изменении задержки получается не в масштабе времени, а оказывается сильно растянутым во времени.

В настоящее время имеются разнообразные электронные аналоговые коррелометры, работающие по методу умножения. Они отличаются только принципом построения узла регулируемой задержки.

Рассмотрим предназначение всех блоков на функциональных схемах, представленных на *рисунке 2.1* и *рисунке 2.2*.

Входные устройства коррелометра предназначены для приема и соответствующего формирования принятых сигналов, которые поступают в виде электрических напряжений. Эти устройства обычно состоят из аттенюатора, эмиттерного повторителя и усилителя. Для того чтобы устройство измеряло АКФ и ВКФ, определяемые формулами (2.1) и (2.2), входные устройства должны обеспечить центрирование сигналов. Для этого во входных цепях коррелометра имеется фильтр верхних частот. При отсутствии центрирования коррелометр определяет моментные функции сигнала.

Устройство регулируемой задержки предназначено для запоминания на некоторое время напряжения исследуемого сигнала. Выходной сигнал повторяет форму входного через некоторый промежуток времени, называемый интервалом задержки.

Схемы умножения корреляционных анализаторов аналогичны классическим схемам, которые используются в вычислительной технике и радиоэлектронике. Всю совокупность схем умножения можно разделить на две большие группы: схемы прямого и непрямого (косвенного) действия.

Схемы прямого умножения представляют собой линейные системы с переменным параметром и работают согласно алгоритму *U = ks1s2*. Они основаны преимущественно на использовании принципа управления коэффициентом передачи четырехполюсника. На вход последнего подается напряжение сигнала *s*1, а коэффициент передачи изменяется пропорционально напряжению сигнала *s*2.

Следовательно, выходное напряжение *U* = *ks*1*s*2. К этому виду схем относятся различные модуляторы, например диодный кольцевой модулятор.

Другой разновидностью указанных множительных устройств являются время-импульсные устройства. Их действие связано с применением периодической последовательности прямоугольных импульсов, относительные длительности которых изменяются пропорционально одному из сомножителей (τ / *Т*с = *k*1 *s*1) и амплитуды – пропорционально второму сомножителю (Um = k2 *s*2). В результате усреднения такой последовательности получается напряжение [1]:

Множительные устройства многих коррелометров должны быть широкополосными и быстродействующими. Этим требованиям хорошо удовлетворяют устройства, основанные на использовании эффекта Холла в полупроводниках, практически безынерционные вплоть до очень высоких частот. Они отличаются также сравнительно простой конструкцией, малыми габаритами и высокой надежностью.

Схемы непрямого умножения, являющиеся устройствами с нелинейными преобразованиями, выполняют операции умножения в результате использования алгебраических или трансцендентных функциональных зависимостей [1]:

Наиболее широко известны устройства, выполняемые на квадраторах и осуществляющие первую зависимость. [1]

Следует отметить, что множительные устройства непрямого действия не всегда позволяют получить необходимую полосу пропускания и вносят заметные аппаратурные погрешности. Поэтому схемы умножения прямого действия, вообще говоря, более предпочтительны [1].

Усредняющие устройства в случае непрерывного изменения задержки выполняется на основе фильтра нижних частот (интегрирующего устройства). При использовании интегратора и скачкообразным изменением времени задержки коррелометр работает циклами. В начале цикла, соответствующего каждому новому значению *τ*, интегратор подключается к выходу схемы умножения и начинает процесс интегрирования. В конце интервала интегрирования вход интегратора отключается от схемы умножения. После фиксации результата измерения напряжение интегратора сбрасывается, путём замыкания конденсатора на малое сопротивление. Обычно эти операции выполняются автоматически.

Регистраторами служат графопостроители, осциллографы, вольтметры, запоминающие устройства с аналого-цифровыми преобразователями [1].

**2.3 Аналоговые коррелометры с дискретизацией времени**

Переход от непрерывных сигналов к дискретным выборкам открыл возможности существенного упрощения аппаратурных решений при построении многоканальных коррелометров. Это объясняется тем, что дискретизация времени позволяет применять даже в рамках аналоговой техники более простые устройства задержки и умножения по сравнению с приборами непрерывного действия.

Вычислительные операции в аналоговых коррелометрах с дискретизацией времени производятся согласно алгоритмам:

; (2.7)

;

. (2.8)

где , ,  – дискретные значения исследуемых сигналов;

** – интервал сдвига (*k* = 0,1,2,…, *n* – 1);

*n* – число измеряемых значений корреляционной функции;

 − интервал дискретизации, определяемый по теореме Котельникова.

В коррелометрах последовательного действия сначала вычисляется значение корреляционной функции соответствующая нулевому сдвигу, то есть энергия исследуемого сигнала, равная . При этом каждое значение  умножается само на себя. Затем вводится задержка  и определяется значение . Операции перемножения при этой задержке иллюстрируются *рисунке 2.3(а)*. На рисунке стрелками соединены перемножаемые значения  и  сигнала. Далее производятся вычисления при интервалах сдвига  (*k*=2),  (*k=3*) и т.д. путем перемножения значений сигнала согласно *рисункам 2.3(б)* и *2.3(в)*:

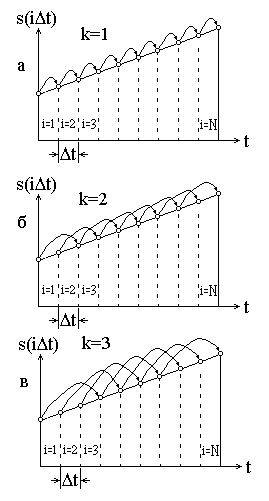


Рисунок 2.3 – Вычисление значений корреляционной функции

при различных интервалах сдвига

Коррелометр параллельного действия позволяет вычислять одновременно все *n* ординат функции корреляции, соответствующих интервалам сдвига от *0* до .

При многоканальном анализе можно определять среднее значение произведений некоррелированных пар выборок.

Упрощенная функциональная схема аналогового коррелятора с

дискретизацией времени представлена на *рисунке 2.4*.

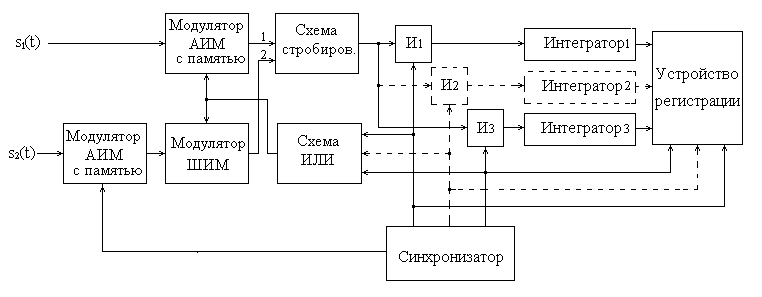


Рисунок 2.4 – Схема упрощенная функциональная аналогового

коррелятора дискретного действия

Работает схема следующим образом: напряжения реализаций *s1(t)* и *s*2(*t*) подводятся ко входным зажимам *1* и *2*. При этом напряжение *s*1(*t*) поступает на сигнальный вход модулятора АИМ, на управляющий вход которого подаются через интервалы *Т*ц = *nΔt* импульсы опроса от коммутатора. Выборка из реализации *s*1(*t*) полученная в момент *ti* опроса, запоминается на весь интервал *Тц*. В результате на сигнальном входе модулятора ШИМ в течение интервала *Т*ц имеется постоянное напряжение, пропорциональное значению *s*1(*ti*), т. е. *a1s*1(*ti*). Через промежутки *Δt* коммутатор посылает импульсы, запускающие модулятор ШИМ и второй модулятор АИМ с памятью. Последний осуществляет выборку из реализации *s2*(*t*) в момент *ti* + *Δt* и запоминает напряжение *a2s*2(*ti* + *Δt*), которое подводится ко входу *1* стробирующей схемы. Модулятор ШИМ преобразует входное напряжение *a1s*1(*ti*) в прямоугольные импульсы длительностью Δ*ti* = =*a3s*1(*ti*) с фиксированной амплитудой. Эти импульсы подаются на вход *2* стробирующей схемы. В моменты *ti* + *Δt*, *ti* + 2*Δt*, ..., *ti + kΔt*, ..., *ti*+*nΔt* стробирующая схема отпирается на время Δ*ti*. На её выходе образуются импульсы длительностью Δ*ti* с амплитудой, пропорциональной значению напряжения *a2s*2(*ti* + *Δt*).

Таким образом, площади выходных сигналов стробирующей схемы получаются пропорциональными произведениям *s*1(*ti*) *s*2(*ti* + *Δt*), *s*1(*ti*) *s*2(*ti* + +2*Δt*), ..., *s*1(*ti*) *s*2(*ti* + *nΔt*) Эти сигналы передаются через вентили *В1*, *B2*, ..., *Bn*, управляемые сигналами коммутатора, на соответствующие усреднители (*RС* – некоммутируемые цепочки), откуда на показывающие или регистрирующее устройство.

За всю продолжительность анализа *Т* проходят *N* циклов *Т*ц. В конце интервала *Т* напряжения на конденсаторах усреднителей пропорциональны оценкам ординат взаимной смешанной моментной функции второго порядка. С помощью показывающего или регистрирующего устройства осуществляется либо визуальное наблюдение, либо регистрация графика функции. [1]

**2.4 Аналого-цифровые коррелометры**

Сочетание аналоговых и цифровых элементов позволяет строить весьма экономичные схемы коррелометров, обладающих достаточной для практики точностью. [1]

Одно из лучших аппаратурных решений в этом плане воплощено в приборах, получивших в литературе название коррелометров Стильтьеса (упрощённая структурная схема представлена на *рисунке 2.5*). Их особенность заключается в том, что при определении взаимной корреляционной функции двух сигналов *s*1(*t*) и *s*2(*t*) квантуется напряжение реализации только одного сигнала, например *s*1(*t*), а напряжение реализации второго сигнала сохраняется в аналоговой форме. [1]

Выражения, определяющие алгоритмы аппаратурного определения временной функции взаимной корреляции, записываются в следующем виде:

При этом возможно квантование как по малому числу уровней, так и применение квантующего устройства с тремя двоичными разрядами (*8* уровней квантования) обеспечивает измерения с пренебрежимо малыми погрешностями.

Достоинство подобных приборов заключается в сравнительной простоте устройства (по отношению к полностью цифровым коррелометрам) вследствие применения ряда простых устройств: сдвигающего регистра в качестве узла задержки, гибридных перемножителей и т. д. [1]

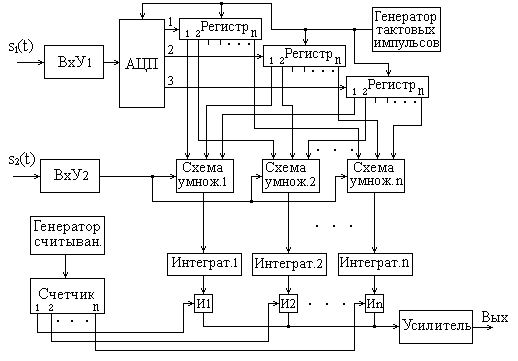


Рисунок 2.5 – Коррелометр аналого-цифровой

**2.5 Цифровые коррелометры**

Устройства этого типа, предполагающие дискретизацию времени и квантование (по многим уровням), осуществляют вычисления согласно следующим алгоритмам

где *s*1кв и *s*2кв – квантованные значения сигналов *s1* и *s*2в дискретные моменты времени;

*kT*0 – интервал сдвига.

Подобные устройства, позволяющие получить наиболее высокую точность, представляют собой специализированные электронные вычислительные установки, в которых используются многие блоки и узлы, аналогичные применяемым в универсальных цифровых машинах.

Цифровые коррелометры, основанные на методе умножения, строятся по общей функциональной схеме, изображенной на *рисунке 2.6*. На этом рисунке показаны лишь основные узлы, имеющиеся в большинстве установок. Разумеется, что в различных коррелометрах возможны свои специфические схемные решения, обусловливающие применение дополнительных узлов.

*s*(*t*)

Входное

устройство

Запоминающее

устройство

Арифметическое

устройство

Регистрирующее

устройство

Управляющее

устройство

Рисунок 2.6 – Схема общая функциональная цифрового коррелометра

Входное устройство может выполнять самые различные функции ослабления или усиления напряжений; считывания реализаций, записанных на бумажной ленте или фотопленке, с преобразованием их в электрические напряжения; воспроизведения сигналов, зафиксированных на магнитных носителях информации; считывания данных с перфоленты; преобразования напряжений в цифровой код. В конечном итоге информация вводится в машину в цифровой форме. Устройства ввода весьма разнообразны. Наиболее сложные устройства применяются в коррелометрах, предназначенных для анализа процессов, реализации которых представлены в виде графика на бумажной ленте или фотопленке. [1]

Запоминающие устройства цифровых коррелометров обладают большой емкостью, позволяют запоминать многоразрядные числа (количество разрядов определяется необходимой точностью вычисления), осуществляют временной сдвиг между перемножаемыми значениями, позволяют многократно извлекать информацию (допускают регенерацию), имеют высокое быстродействие.

В качестве запоминающих устройств могут быть применены различные типы блоков памяти, встречающиеся в цифровых вычислительных машинах.

Арифметическое устройство выполняет вычислительные операции с заданным алгоритмом умножения, накопления произведений, деления результата суммирования *N* произведений на число *N*.

В зависимости от того, каким способом производится умножение, возможны различные функциональные схемы построения арифметических устройств.

Обычно они содержат регистры множимого и множителя, а также сумматор-регистр частичных произведений.

Показывающее или регистрирующее устройство обычно состоит из выходного узла (устройства вывода данных), системы цифровой индикации, цифропечатающей машины. Иногда предусматривают цифро-аналоговый преобразователь, устройство визуального наблюдения коррелограммы или самопишущий прибор.

Управляющее устройство вырабатывает сигналы управления – команды передачи информации из запоминающего устройства в арифметическое, проведения вычислительных операций, вывода результатов измерений и выдачи их на печать. [1]

Анализ аппаратурных средств определения корреляционных функций показал, что практически во всех приборах, измеряющих корреляционную функцию можно выделить следующих четыре блока: формирователь сигналов, умножитель, интегратор и регистрирующее устройство.

На основании проведенного анализа можно разработать алгоритмические и технические требования к программному анализатору корреляционных характеристик радиотехнических сигналов.

**3 РАЗРАБОТКА АЛГОРИТМИЧЕСКИХ И ТЕХНИЧЕСКИХ**

**ТРЕБОВАНИЙ К ПРОГРАММНОМУ АНАЛИЗАТОРУ**

**КОРРЕЛЯЦИОННЫХ ХАРАКТЕРИСТИК**

**РАДИОТЕХНИЧЕСКИХ СИГНАЛОВ**

**3.1 Общие требования**

Программный анализатор корреляционных характеристик радиотехнических сигналов – это функционально законченный программный продукт, применение которого позволит усовершенствовать исследование и анализ корреляционных функций сигналов, повысить эффективность учебного процесса и научных работ.

Целью разработки программного анализатора корреляционных характеристик радиотехнических сигналов является – создание программы, обеспечивающей наиболее эффективный способ анализа корреляционных характеристик, наиболее емкое понимание сути работы и предоставляющей интуитивно понятный пользовательский интерфейс. Так же программа должна отвечать всем требованиям, установленными преподавателем.

Разработка программного анализатора корреляционных характеристик должна осуществляться с использованием современных языков программирования и сред разработке, а так же графических библиотек способных наиболее точно отразить сигналы и их корреляционные характеристики.

В современной разработке программного обеспечения часто используется принцип разделения программы на отдельные слабосвязанные модули, реализацию которых можно легко изменять без изменения связанных с ним модулей. Такой подход к разработке программного обеспечения позволяет значительно повысить эффективность как самой программы так и ее разработки, значительно сокращая количество времени, затрачиваемое на внесение необходимых функциональных изменений. Для реализации такого подхода используется объектно-ориентированное программирование, когда каждый модуль системы, представляется в виде отдельного компонента, содержащего в себе общий для многих частей программы код, который легко исправлять, отлаживать или изменять в зависимости от установленных технических требований. Тот же принцип касается и пользовательского интерфейса, когда определенный функционал программы разбивается на несколько отдельных шагов, которые необходимо

выполнить пользователю. Это позволяет создать простой и понятный интерфейс, который позволяет наиболее быстро получить необходимые сведения, не тратя время на осознание принципов работы программы.

Общие положения, которым должен удовлетворять программный анализатор корреляционных характеристик радиотехнических сигналов:

Адаптированность – программа должна быть спроектирована таким образом, чтобы обеспечить наибольшую гибкость в ее использовании, возможность изменения ее функционала в зависимости от заданных условий.

Эффективность – программа должна сокращать количество ресурсов и времени, затрачиваемого на обучение и исследование корреляционных характеристик радиотехнических сигналов, не теряя при этом в качестве и полноте предоставляемой информации

Простота – пользователь не должен обладать специальными знаниями и умениями, чтобы использовать данное программное обеспечение. Время затрачиваемое на освоение программы должно быть минимальным.

Скорость – программа должна быть оптимизирована, чтобы обеспечить минимальное время ожидания по нажатию пользователем клавиши, а также обеспечить необходимую скорость вычисления корреляционной функции для ее последующей визуализации.

Модульность – программа должна быть спроектирована таким образом, чтобы все ее модули были слабосвязаны и легко могли быть изменены. Общий код для нескольких частей программы должен быть вынесен в отдельный компонент для оптимизации программы и повышения эффективности разработки.

Сложность разработки такого программного обеспечения обусловлена четырьмя основными причинами:

– сложностью предметной области (временной анализ детерминированных сигналов различной формы и параметров), из которой исходит задание на разработку;

– необходимостью динамической визуализации корреляционных характеристик;

– необходимостью обеспечить достаточную гибкость и расширяемость программного продукта;

– необходимостью определить методику создания и обработки произвольных радиотехнических сигналов.

При создании программы следует также учесть общие требования:

Программа должна быть способной работать на современных персональных компьютерах, используемых в учебных заведениях.

Программа должна быть простой в понимании, не требовать изучения дополнительной литературы или получения специальных навыков обращения с программой.

Программный анализатор должен быть *Windows* приложением, иметь простой, удобный, интуитивно понятный интерфейс.

При структурном программировании большое значение имеют также специфические требования.

**3.2 Специфические требования**

Программный анализатор корреляционных характеристик радиотехнических сигналов должен удовлетворять следующим требованиям, которые сформулированы с учетом интересов и возможностей пользователя и содержательного смысла основных вопросов, связанных с анализом корреляционных характеристик радиотехнических сигналов, а также возможностей технических средств ПЭВМ.

Программный анализатор должен представлять собой программу моделирующую создание радиотехнических сигналов и вычисление их корреляционных функций.

Компьютерное моделирование должно осуществляться путем выполнения программы на ПЭВМ. При этом необходимо обеспечить:

– возможность работы в диалоговом режиме независимо от навыков и умений пользователя в обращении с компьютером;

– оптимальность и полноту отображаемой информации;

– оперативное управление процессом изменения исходных данных, формированием, выбором и содержанием отображаемой информации;

– создание динамической визуальной информации;

– современный дизайн отображаемой информации;

– динамическое изменение отображаемой информации в ответ на возникновение исключительных ситуаций в программе.

Программный анализатор должен обеспечивать формирование и вывод динамической визуальной информации с использованием современных технологий программирования и моделирования.

Программный анализатор должен быть оформлен в виде программного продукта в общепринятом смысле. Размер конечного файла программы ограничен объемом свободного пространства на разделе жесткого диска применяемой ПЭВМ.

Программа должны удовлетворять классическому перечню требований: отсутствие ошибок, устойчивость работы программы при неправильных или случайных нажатиях клавиш, простота пользования программой, обработка возникающих исключительных ситуаций, минимальное время реакции на действия пользователей и др.

Программный анализатор корреляционных характеристик радиотехнических сигналов должен обеспечить воспроизведение наиболее существенной информации, в оптимальном количестве для ее осознания и в форме наиболее легкой для понимания.

Основные требования к средствам методического обеспечения не отличаются от известных требований к учебно-методической литературе и технической документации программного продукта. В методических рекомендациях следует указать:

– инструкцию по переносу, загрузке, запуску и пользованию программой;

– название учебной дисциплины и темы, изучая которые рекомендуется использовать программу;

– назначение программы, методическую цель, приобретаемые пользователем знания, умения, навыки;

– основные теоретические сведения по тематике программного анализатора корреляционных характеристик радиотехнических сигналов;

– методические указания по пользованию программой, необходимые для самостоятельной работы.

**3.3 Функциональные требования**

Программный анализатор должен обеспечивать:

– формирование произвольного радиотехнического сигнала;

– выбор созданного сигнала из файла;

– моделирование процессов вычисления автокорреляционной функции и взаимокорреляционной функции;

– создание динамической визуальной информации по результатам расчета.

Для формирования произвольного радиотехнического сигнала используется ряд задаваемых параметров:

– начало сигнала;

– конец сигнала;

– амплитуда сигнала;

– количество импульсов;

– частота сигнала;

– тип сигнала;

Когда тип сигнала был выбран, программа должна генерировать сигнал, форма которого зависит от заданных параметров. Программный анализатор корреляционных характеристик радиотехнических сигналов должен обеспечивать задание следующих сигналов:

– синусоидального сигнала со случайной начальной фазой;

– узкополосного случайного сигнала;

– пилообразного сигнала со случайной начальной фазой (случайным параметром сдвига).

**4 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ**

**СЕБЕСТОИМОСТИ ПРОГРАММНОГО ОЕСПЕЧЕНИЯ ДЛЯ**

**АНАЛИЗА КОРРЕЛЯЦИОННЫХ ХАРАКТЕРИСТИК**

**РАДИОТЕХНИЧЕСКИХ СИГНАЛОВ**

**4.1 Характеристика разрабатываемого ПО**

Разрабатываемое программное обеспечение позволяет проводить динамический анализ корреляционных характеристик радиотехнических сигналов в режиме реального времени. Результат вычислений отображается графически с возможностью изменения шага построения графика.

**4.2 Определение объёма и трудоёмкости ПО**

Объём программного средства определяется исходя из количества функций, реализуемых программой, которые представлены в *таблице 4.1*, и рассчитываются по формуле [1]:

|  |  |
| --- | --- |
|  | (4.1) |

где – общий объём ПС;  
 - объём функции ПС;  
 n – общее число функций.

Таблица 4.1 – Перечень и объем функций программного модуля

|  |  |  |  |
| --- | --- | --- | --- |
| Номер | Наименование (содержание) функции | Объём функ-  ции | Уточнённый объём функции |
| 102 | Контроль, предварительная обработка и ввода информации | 550 | 520 |
| 305 | Обработка файлов | 800 | 800 |
| 309 | Формирование файла | 1080 | 1050 |
| 502 | Монитор системы(управление комплекса ПО) | 3880 | 3900 |
| 506 | Обработка ошибочных и сбойных ситуаций | 410 | 410 |
| 604 | Справка и обучение | 850 | 850 |

По формуле (4.1) находим общий объём ПО

На основании общего объёма ПО определяется нормативная трудоёмкость ( с учётом сложности ПО. Так как разрабатываемый продукт относится к *1*-ой группе сложности из-за наличия машинной графики, то нормативная трудоёмкость составляет *252* человеко-дней.

Общая трудоёмкость ПО рассчитывается на основе нормативной с учётом дополнительного коэффициента сложности (4.3), дополнительный коэффициент сложности рассчитывается по формуле 4.2:

|  |  |
| --- | --- |
|  | (4.2) |
|  |  |
| где – коэффициент, соответствующий повышению сложности ПО ы за счет конкретной характеристики;  где – количество учитываемых характеристик. |  |
|  |  |

|  |  |
| --- | --- |
|  | (4.3) |

где – общая трудоёмкость ПС;

где – нормативная трудоёмкость ПС;

где – дополнительный коэффициент сложности ПО.

Таким образом, общая трудоёмкость с учётом округления составила *283* человеко-дней.

На основании общей трудоёмкости разработки ПС и установленного периода разработки устанавливается общая плановая численность разработчиков:

|  |  |
| --- | --- |
|  | (4.4) |

где - плановая численность разработчиков (чел.);

где - годовой эффективный фонд времени работы одного работника в течение года (дней в год);

где - плановая продолжительность разработки ПО (лет). Устанавливается календарным графиком разработки, равна *0,4* года.

Эффективный фонд времени одного работника ( рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (4.5) |

где - количество дней в году (365);

где - количество праздничных дней в году (8);

где - количество выходных дней в году (104);

где - количество дней отпуска (25 дней).

Таким образом, получаем 228 дней.

Рассчитаем общую плановую численность работников:

Общая плановая численность разработчиков и трудоёмкость служат базой для расчёта основной заработной платы.

**4.3 Расчёт себестоимости ПО**

Исходные данные для расчёта основной заработной платы исполнителя приведены в *таблице 4.2*.

Таблица 4.2 – Разработчик ПС.

|  |  |  |  |
| --- | --- | --- | --- |
| Разработчик | Тарифный разряд | Тарифный коэффициент | Эффективный фонд времени (дней) |
| Ведущий программист | 15 | 3.48 | 103 |
| Программист I категории | 14 | 3.25 | 60 |
| Программист I категории | 14 | 3.25 | 60 |
| Программист I категории | 14 | 3.25 | 60 |

Месячная тарифная ставка исполнителя определяется путём умножения действующей тарифной ставки первого разряда предприятия на тарифный коэффициент , соответствующий установленному тарифному разряду:

|  |  |
| --- | --- |
|  | (4.6) |

При действующей тарифной ставке первого разряда, равной *34* белорусским рублям, учитывая, что у трех программистов первой категории одинаковый тарифный разряд, получаем:

Часовая тарифная ставка рассчитывается путём деления тарифной ставки на установленный фонд рабочего времени.

|  |  |
| --- | --- |
|  | (4.7) |

где - часовая тарифная ставка руб.;

где МЧФ – месячный часовой фонд

Основная заработная плата разработчиков рассчитывается по формуле:

|  |  |
| --- | --- |
|  | (4.8) |

где n – количество разработчиков, занятых разработкой ПО;

где - тарифная ставка i-го исполнителя в руб.;

где - эффективный фонд рабочего времени i-го исполнителя (дней);

где - количество часов работы в день (ч);

где - коэффициент премирования (1,5).

Так как количество разработчиков данной ПС равно *4*, то можно следующим образом рассчитать основную заработную плату разработчиков:

Дополнительная заработная плата на конкретное ПО включает выплаты, предусмотренные законодательством о труде (оплата отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей), и определяется по нормативу в процентах к основной заработной плате

|  |  |
| --- | --- |
|  | (4.9) |

где - дополнительная заработная плата исполнителей на конкретное ПС (руб.);

где - норматив дополнительной заработной платы в целом по организации (20%).

458.16 руб.

Следующей статьёй сметы затрат являются отчисления на социальные нужды. Отчисления на социальные нужды включают в предусмотренные законодательством отчисления в фонд социальной защиты и фонд обязательного страхования (34,6%) в процентах от основной и дополнительной заработной платы. Отчисления в фонд социальной защиты населения и обязательного страхования от несчастных случаев на производстве и профзаболеваний рассчитываются следующим образом:

|  |  |
| --- | --- |
|  | (4.10) |

где – норматив отчислений на социальные нужды.

Норма расхода материалов в суммарном выражении определяется либо в расчёте на *100* строк исходного кода, либо в процентах к основной заработной плате разработчиков (3-5%). Возьмём 4% от основной зарплаты.

Следовательно, расходы на материалы рассчитываем следующим образом:

|  |  |
| --- | --- |
|  | (4.11) |

где - норматив отчислений на материалы.

Расходы на спецоборудование (дополнительное ПО) отсутствуют.

Расходы на оплату машинного времени определяются по формуле:

|  |  |
| --- | --- |
|  | (4.12) |

где - цена одного машино-часа (0,05 руб.);

где - общий объём ПО (строк исходного кода);

где - норматив расхода машинного времени на отладку *100* строк исходного кода (машино-часов).

Средний расход машинного времени *12* часов на *100* строк кода для ПО.

Расходы по статье «Научные командировки» определяются по нормативу, разрабатываемому в целом по организации, в процентах к основной заработной плате:

|  |  |
| --- | --- |
|  | (4.13) |

где - норматив расходов на научные командировки в целом по организации (10%).

Расходы по статье «Прочие затраты» определяются по формуле 4.14:

|  |  |
| --- | --- |
|  | (4.14) |

где - норматив прочих затрат (10%).

Расчёт накладных расходов ведётся по формуле 4.15:

|  |  |
| --- | --- |
|  | (4.15) |

где - накладные расходы на конкретную ПС (руб.);

где - норматив накладных расходов в целом по организации (100%).

Общая сумма расходов по всем статьям сметы рассчитывается по формуле 4.16:

|  |  |
| --- | --- |
|  | (4.16) |

руб.

Организация-разработчик участвует в освоении ПО и несёт соответствующие затраты, на которые составляется смета, оплачиваемая заказчиком по договору.

Смета затрат на разработку ПО сведена в *таблицу 4.3*.

Таблица 4.3 – Смета затрат на разработку ПС.

|  |  |  |
| --- | --- | --- |
| Наименование статей затрат | Условное обозначение | Значение, руб. |
| Основная заработная плата |  |  |
| Дополнительная заработная плата |  |  |
| Отчисления в фонд социальной защиты населения и обязательного страхования от несчастных случаев на производстве и профзаболеваний |  |  |
| Материалы |  |  |
| Машинное время |  |  |
| Расходы на научные командировки |  |  |
| Прочие затраты |  |  |
| Накладные расходы |  |  |
| Полная себестоимость |  |  |

В данном разделе была рассчитана смета на разработку ПО для анализа корреляционных характеристик радиотехнических сигналов. Система относится к первой группе сложности. Численность исполнителей составляет четыре человека на каждой стадии, в числе которых три программиста первой категории и один ведущий программист.

В результате расчётов полная себестоимость готового продукта составляет *6585.87* руб.

**5 ФУНКЦИОНАЛЬНЫЙ АНАЛИЗ ПРОЦЕССОВ**

**ОПРЕДЕЛЕНИЯ КОРРЕЛЯЦИОННЫХ ФУНКЦИЙ**

**СИГНАЛОВ**

Для того чтобы рассчитать корреляционную функцию, необходимо сформировать необходимые радиотехнические сигналы. Одним из путей реализации такого формирователя является создание последовательности временных отсчетов соответствующих значениям сигнала в эти временные отсчеты с последующей записью и чтением из файла.

Как говорилось выше, корреляционная функция двух сигналов рассчитывается по следующим формулам:

В случае автокорреляционной функции, оба сигнала одинаковы, но один из них сдвинут на некоторое время (τ). В случае взаимокорреляционной сигналы эти сигналы различны.

Для определения корреляционной функции один из сигналов необходимо постепенно сдвигать относительно другого на время *τ*.

Далее необходимо найти площадь пересечения двух сигналов: оригинального и сдвинутого на время *τ*. Чтобы найти произведение двух радиотехнических сигналов необходимо перемножить между собой значения, которые принимают сигналы в соответствующие отсчеты времени.

После того как произведение оригинального сигнала и сигнала, сдвинутого на время *τ*, будет найдено необходимо проинтегрировать полученное произведения. Пределы интегрирования будут зависеть от шага, с которым брались отсчеты сигнала.

Таким образом, проинтегрировав произведение двух сигналов, мы найдем площадь их пересечения.

По мере возрастания области пересечения сигналов, площадь этой области будет так же возрастать, что при заданном пороге обнаружения может означать присутствие полезного сигнала.

Для того, чтобы корреляционные характеристики радиотехнических сигналов были рассчитаны наиболее точно следует разбивать сигналы на временные отсчеты с минимальным допустимым шагом. Тогда, проинтегрировав произведение сигналов, мы получим значение корреляционной функции максимально приближенное к реальности.

Таким образом, как видно из функционального анализа процессов определения корреляционных функций сигналов, для реализации модуля расчёта корреляционных функций в программном анализаторе необходимо и достаточно реализовать расчёт формул (5.1) и (5.2).

**6 РАЗРАБОТКА ОБЩЕГО АЛГОРИТМА МОДЕЛИРОВАНИЯ**

**ПРОЦЕССОВ ОПРЕДЕЛЕНИЯ КОРРЕЛЯЦИОННЫХ**

**ФУНКЦИЙ СИГНАЛОВ. ВЫБОР СИСТЕМЫ**

**ПРОГРАММИРОВАНИЯ И РАЗРАБОТКА ПРОГРАММЫ**

**6.1 Разработка общего алгоритма моделирования процессов**

**определения корреляционных функций сигналов**

Проведенный функциональный анализ процессов определения корреляционных функций сигнала показал, что для выполнения технического задания необходимо и достаточно смоделировать работу следующих блоков: формирователь сигналов, умножитель, интегратор, устройство отображения динамической визуальной информации.

На основании результатов проведенного анализа можно составить структурную схему программного анализатора. Структурная схема программного анализатора приведена на *рисунке 6.1*.

Формирователь сигнала

Интегратор

Умножитель

Формирование и отображение динамической визуальной информации

Формирователь сигнала

Рисунок 6.1 – Схема программного анализатора cтруктурная

Формирователь сигнала должен создавать сигнал на основании ряда задаваемых параметров и типа радиотехнического сигнала. Возможные типы задаваемых сигналов должны быть внесены на основании условий, заданных техническим руководителем.

Всеми вычислительными процессами в ЭВМ занимается процессор. Так как процессор работает на основе двоичной логики, задавать аналоговый сигнал, соответствующий внесенным параметрам, невозможно. Поэтому при расчете корреляционных характеристик радиотехнических сигналов необходимо использовать исключительно цифровой сигнал. Элементом, служащим для преобразования аналогового сигнала в цифровой, является аналогово-цифровой преобразователь(АЦП). Поэтому в основу работы формирователя сигнала будет положен тот же принцип, на основании которого работает АЦП. Для этого сигнал нужно представить в виде дискретных временных отсчетов, которые нужно брать с определенным шагом дискретизации. При прохождении аналогового сигнала через АЦП часть сигнала теряется в зависимости от выбранного шага дискретизации. Поэтому для формирования цифрового сигнала наиболее точно соответствующего аналоговому необходимо принять шаг дискретизации минимально возможным. Для программного анализатора корреляционных характеристик радиотехнических сигналов примем этот шаг равным единице как минимально возможному целочисленному числу, с которым можно осуществлять выборки сигнала.

Сформированный формирователем сигнал сохраняется в файл, информация из которого затем поступает на умножитель, принцип работы которого был рассмотрен в функциональном анализе процессов определения корреляционных функций сигналов. Таким образом, на умножитель может поступать сигнал как из файла, сформированного формирователем сигнала, так и из заранее сформированных файлов.

При моделировании работы интегратора необходимо учитывать, что интегрировать произведение двух сигналов в пределах бесконечности не представляется возможным. Поэтому, в соответствии с техническим заданием, целесообразно формировать сигналы длительностью не более *120* мкс, чтобы конечная последовательность дискретных отсчетов сигнала не получилась излишне большой и пределами интегрирования были бы значения от -*120* мкс до *120* мкс.

Блок формирования сигнала следует вынести в отдельный структурный модуль, представляющий собой независимый программный компонент, не связанный с модулем расчета и отображения.

При создании сигнала необходимо добавить возможность задания различных параметров, необходимых для формирования сигналов разной формы и длительности: начало и конец сигнала, амплитуда, количество импульсов, частота и тип сигнала.

Для создания сигнала следует добавить кнопку задания сигнала, по нажатию которой будет генерироваться сигнал на основании описанных выше параметров. Значения сигнала в отдельные моменты времени должны быть записаны в таблицу, где их можно будет просмотреть, удалить, изменить или добавить новые значения вручную.

Для того, чтобы можно было просмотреть форму сгенерированного сигнала на этапе формирования сигнала, следует добавить кнопку отображения сигнала. Значения из таблицы, хранящую значения сигнала в отдельные моменты времени, будут отображаться на соответствующий график. Это позволит просмотреть форму сигнала и, визуально проанализировав, изменить ее на этапе формирования сигнала.

Также должна быть добавлена возможность сохранения сгенерированного сигнала в файл путем переноса дискретных значений сигнала в файл, который создаст пользователь.

Такой способ формирования сигнала позволяет избежать возможных ошибок и увеличить скорость создания сигнала при ручном формировании.

Так как программный анализатор корреляционных характеристик радиотехнических сигналов должен обеспечивать создание динамической визуальной информации, то расчет корреляционных характеристик будет проводиться двумя способами.

В первом случае необходимо произвести расчет и отображение полностью построенной корреляционной функции сигнала. Для того чтобы осуществить данные операции следует вычислить границы сдвига копии сигнала относительно оригинального сигнала, в пределах которых сигналы будут пересекаться. Затем, последовательно увеличивая значение нижней границы и соответствующее значение сдвига копии сигнала, рассчитывать значение корреляционной функции сигнала по методу, описанном в пункте *6.1*, в зависимости от сдвига сигнала и отображать данное значение сигнала на графике.

Во втором случае необходимо задать отсчет в момент времени, предшествующий началу пересечения двух сигналов, и, обрабатывая нажатие клавиш ← →, сдвигать копию сигнала по временной оси. Затем необходимо рассчитать значение корреляционной функции для соответствующего сдвига и отобразить его на графике.

С учётом всего вышеизложенного, главное страницу приложения имеет смысл сформировать таким образом, как показано на *рисунке 6.2*. Как видно из *рисунка 6.2*, на главной странице доступно четыре варианта выбора.

Динамическое построение корреляционных функций включается путем выбора соответствующего флага

.

Создание сигнала

Главное страница

Построение и анализ автокорреляционных функций

Задать сигнал

Отобразить сигнал

Сохранить сигнал

Внести значение

Удалить значение

Выбор сигнала

Отображение построенной автокорреляционной функции

Выбор сигнала

Справка

Построение и анализ взаимо

корреляционных функций

Отображение построенной взаимо

корреляционной функции

Динамическое построение автокорреляционной функции

Динамическое построение взаимо

корреляционной функции

Рисунок 6.2 – Структура главного меню

На странице справки можно просмотреть форму со справочными данными.

На основании приведенного выше алгоритма работы будет разработана и реализована программа в выбранной среде разработки.

**6.2 Выбор системы программирования и разработка программы**

Для реализации приведенного выше алгоритма необходимо выбрать язык программирования, среду разработки и необходимые библиотеки.

В первую очередь стоит определиться язык программирования, на котором будет разрабатываться приложение.

Так как одним из требований к программному анализатору являлось возможность работы в операционной системе *Windows XP*, то нам понадобится язык программирования, на котором возможно разрабатывать *Windows* приложения.

Вместе с тем следует думать о возможности запуска приложения на других операционных системах, поэтому необходимо выбрать язык программирования, компилирующий код программы в байт-код способный запускаться на любой операционной системе.

Одним из языков программирования способных транслировать код программы в исполняемый байт-код является язык *Java*. *Java* – это строго типизированный объектно-ориентированный язык широко использующийся в разработке приложений.

Таким образом, на основании технических требований к разработке программного анализатора корреляционных характеристик радиотехнических сигналов был выбран язык программирования *Java*. Выбор языка программирования *Java* оправдан ориентированностью на объектно-ориентированное программирование, обеспечивающей выполнения требования о модульности приложения, простотой разработки из-за наличия простого и интуитивно понятного синтаксиса, множества готовых решений и сборщика мусора.

Сборщик мусора – это специальный процесс, управляющий жизненным циклом объектов в языке *Java*, которые в свою очередь являются именованными сущностями, хранящими некоторые значения. Так как при создании объекта Виртуальная машина *Java*(*Virtual Java Machine* – *JVM*) выделяет для него некоторую область памяти, то сборщик мусора по окончанию жизненного цикла объекта освобождает эту память, что позволяет разработчику сосредоточиться исключительно на логике работы программы, не занимаясь конфигурацией внутренних процессов работы программы, что значительно повышает скорость работы программы и сокращает количество рабочего кода приложения.

Следующим этапом является выбор среды разработки, в которой будет разрабатываться приложение. Для языка *Java* создан ряд программных продуктов, позволяющих эффективно осуществлять разработку приложения: *NetBeans*, *IntelliJ IDEA*, *Eclipse*, *JDeveloper*, *Android Studio* и др. На основании анализа приведенных выше сред разработки была выбрана среды *IntelliJ IDEA* как наиболее подходящая для разработки программного анализатора корреляционных характеристик радиотехнических сигналов. *IntelliJ IDEA* – мощная среда разработки, позволяющая эффективно создавать приложения на языке *Java*. Среда разработки оптимизирована и предоставляет большое количество готовых решений и плагинов разработки, позволяющих упростить создание программы, быстро находить и исправлять допущенные синтаксические и логические ошибки.

Финальным этапом выбора системы программирования является выбор необходимых библиотек.

Так как программный анализатор корреляционных характеристик радиотехнических сигналов должен иметь возможность создания динамической визуальной информации, то для реализации этой возможности необходимо выбрать графическую библиотеку, написанную на языке *Java*.

Самым мощным и простым графическим средством разработки для языка *Java* на сегодняшний день является библиотека *JavaFX*.

Используя средства разработки, которые предоставляет библиотека *JavaFX*, можно легко создавать динамически изменяющиеся графические формы и добавлять на них различные графические компоненты: кнопки, таблицы, графики и анимацию.

Вывод динамической визуальной информации будет основан на использовании компонента *LineChart*. Выбор компонента *LineChart* обусловлен наличием удобного интерфейса для отображения формы корреляционной функции сигнала и динамического добавления и удаления значений.

Для отображения текста будет использоваться компонент *Label*, позволяющий настраивать тип шрифта, размер и другие параметры.

Для осуществления операций и перехода между формами будут добавлены компоненты *Button*.

Чтобы задать некоторые флаги, которые впоследствии будет обрабатывать программа были использованы компоненты *CheckBox*.

Для задания параметров при создании нового сигнала на форму были добавлены компоненты *TextField*, данные из которых по нажатию на компонент *Button* будут передаваться на форму.

Для вывода списка возможных генерируемых типов сигналов на форму был добавлен компонент *ChoiceBox*, данные из которого также передаются в приложение по нажатию компоненты *Button*.

Для отображения дискретных значений сигнала на этапе формирования сигнала на форму был добавлен компонент *TableView* и сконфигурирован таким образом, чтобы была возможность изменения добавленных значений.

На странице справки был добавлен компонент *Pagination* для отображения нескольких страниц справочной информации.

Для обеспечения нормальной работы программы на ЭВМ с различным разрешением экрана были добавлены компоненты *StackPane*, *GridPane* и *BorderPane*, чтобы обеспечить динамическое изменение размера компонентов при изменении размера формы.

Алгоритм расчёта построенных корреляционных функций представлен блок-схемой (приложение А). Блок-схема алгоритма пошагового расчета корреляционных функций представлена в приложении Б.

Исходный код программного анализатора корреляционных характеристик радиотехнических сигналов приведен в листинге приложении Д.

С целью практического применения разработанного анализатора необходимо создать методику использования разработанного программного анализатора.

**7 МЕТОДИКА ПРИМЕНЕНИЯ ПАК ДЛЯ ОПРЕДЕЛЕНИЯ И**

**АНАЛИЗА КОРРЕЛЯЦИОННЫХ ФУНКЦИЙ**

Любое программное обеспечение (ПО) требует пояснения методики его практического применения (методики использования). Разработанный программный анализатор корреляционных характеристик радиотехнических сигналов так же не является исключением.

Программный анализатор предназначен для учебных и исследовательских целей по дисциплинам, связанным с радиотехникой, а так же для выполнения лабораторных работ по тематике «Корреляционный анализ радиотехнических сигналов» и решения некоторых вопросов исследовательского характера.

При работе с программным анализатором пользователь имеет возможность создать базу различных радиотехнических сигналов, закрепить навыки использования ПЭВМ, приобрести умение определять приблизительный вид корреляционных функций всевозможных сигналов вследствие понимания процесса расчёта корреляционных функций в динамике.

Программный анализатор корреляционных характеристик радиотехнических сигналов для своей работы требует всего лишь 2 Мб свободного места на жёстком диске (в базовой комплектации: программа и файлы сигналов), операционную систему *Microsoft Windows XP* (так же проверена работа на операционных системах: *Microsoft Windows Vista*, *Microsoft Windows 7* и *Microsoft Windows 8*), что полностью удовлетворяет предъявленным требованиям к программному анализатору в части совместимости.

**7.1 Методика работы с программой**

7.1.1 Интерфейс программы и общие положения

Для запуска программного анализатора, как и любого *Windows* приложения, достаточно дважды кликнуть левой кнопкой мыши по его исполняемому *jar* файлу (в данном случае этой файл “KorrelFunction.jar”).

Перед пользователем сразу же откроется главная страница программы,

вид которого представлен на *рисунке 7.1*.

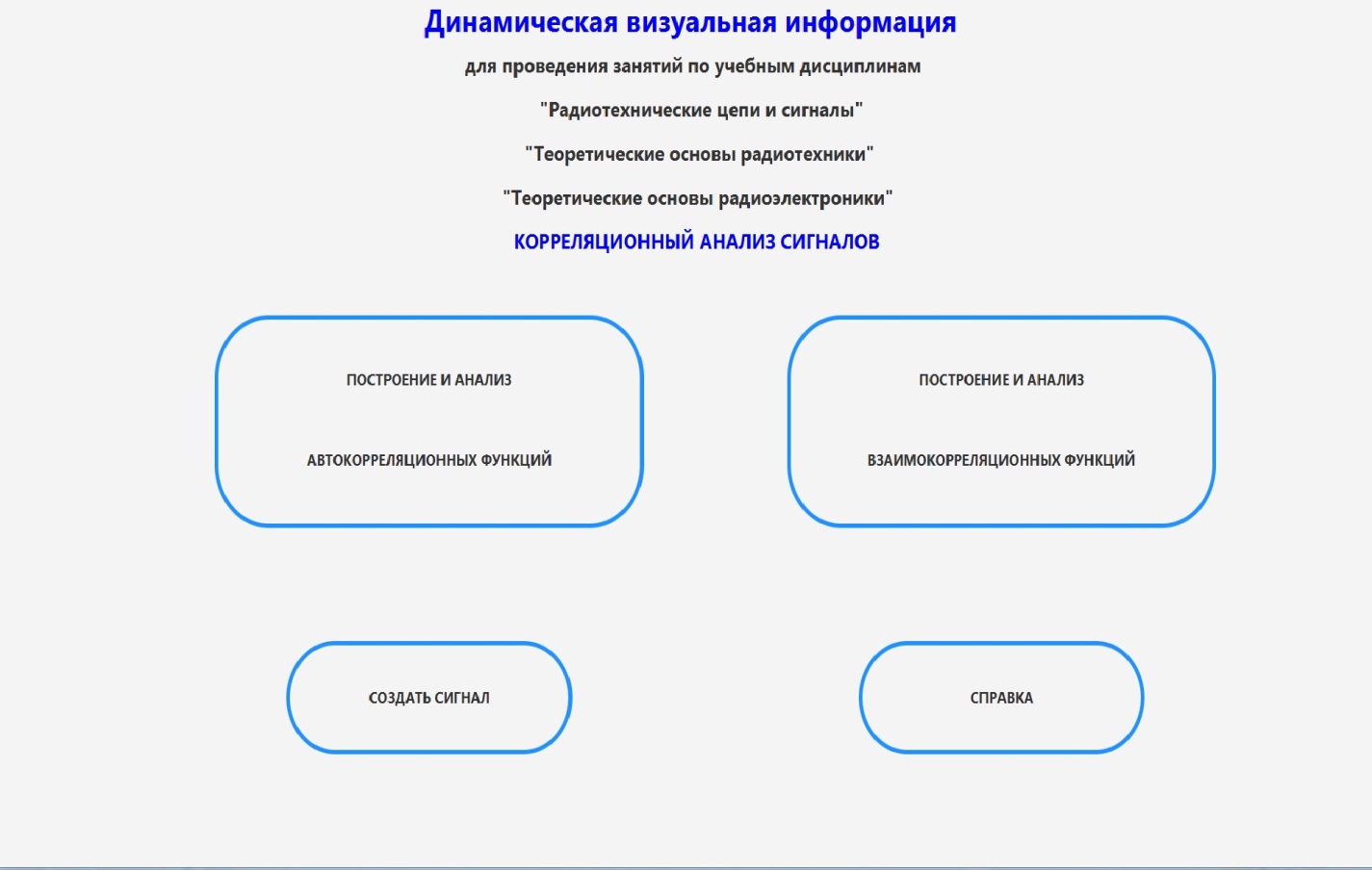


Рисунок 7.1 – Страница главная

На главной странице можно увидеть четыре доступных варианта выбора: “Построение и анализ автокорреляционных функций”, “Построение и анализ взаимокорреляционных функций”, “Создать сигнал” и “Справка”.

При наведении указателя на один из вариантов текст внутри блока загорается красным. При нажатии на один из блоков пользователь попадает на соответствующую страницу, откуда при желании сможет вернуться обратно.

7.1.2 Создание нового сигнала

На странице создания нового сигнала можно создать произвольный радиотехнический сигнал, изменить его, просмотреть его форму и сохранить полученный сигнал в файл.

На *рисунке 7.2* показан созданный в программе треугольный видеоимпульс. Для того чтобы сгенерировать сигнал, необходимо задать ряд параметров: начало и конец сигнала, амплитуда, количество импульсов, частота сигнала и тип сигнала. Так же для создания последовательности видеоимпульсов или периодического сигнала можно задать параметр количество импульсов или частота сигнала соответственно. После того как параметры будут заданы следует нажать на кнопку “Задать”.

После того, как сигнал будут сгенерирован, данные о значениях сигнала в отдельные отсчеты времени будут внесены в таблицу, где их можно изменять нажав два раза на значение в нужной колонке. После этого необходимо ввести новое значение и нажать на клавишу “Enter”.

Для того, чтобы добавить или удалить значение из таблицы стоит внести данные о моменте времени и значение сигнала в этот момент времени в соответствующие поля, после чего нажать кнопку “Внести” или “Удалить” соответственно.

Для того чтобы просмотреть и проанализировать форму сгенерированного сигнала, необходимо нажать на кнопку “Отобразить” после чего на графике будет отображен сгенерированный сигнал.

Чтобы сохранить сигнал в файл необходимо нажать на кнопку “Сохранить”.



Рисунок 7.2 – Создание нового сигнала

7.1.3 Выбор сигнала для расчета автокорреляционной функции

При нажатии на блок “Построение и анализ автокорреляционных функций” появится страница выбора сигнала для расчета автокорреляционной функции. По нажатию на кнопку “Выбрать” откроется диалоговое окно, где можно будет выбрать заранее созданный сигнал или созданный пользователем на странице создания нового сигнала. После того как сигнал будет выбран, его форма отобразится на графике. Результат выбора треугольного видеоимпульса показан на *рисунке 7.3*.

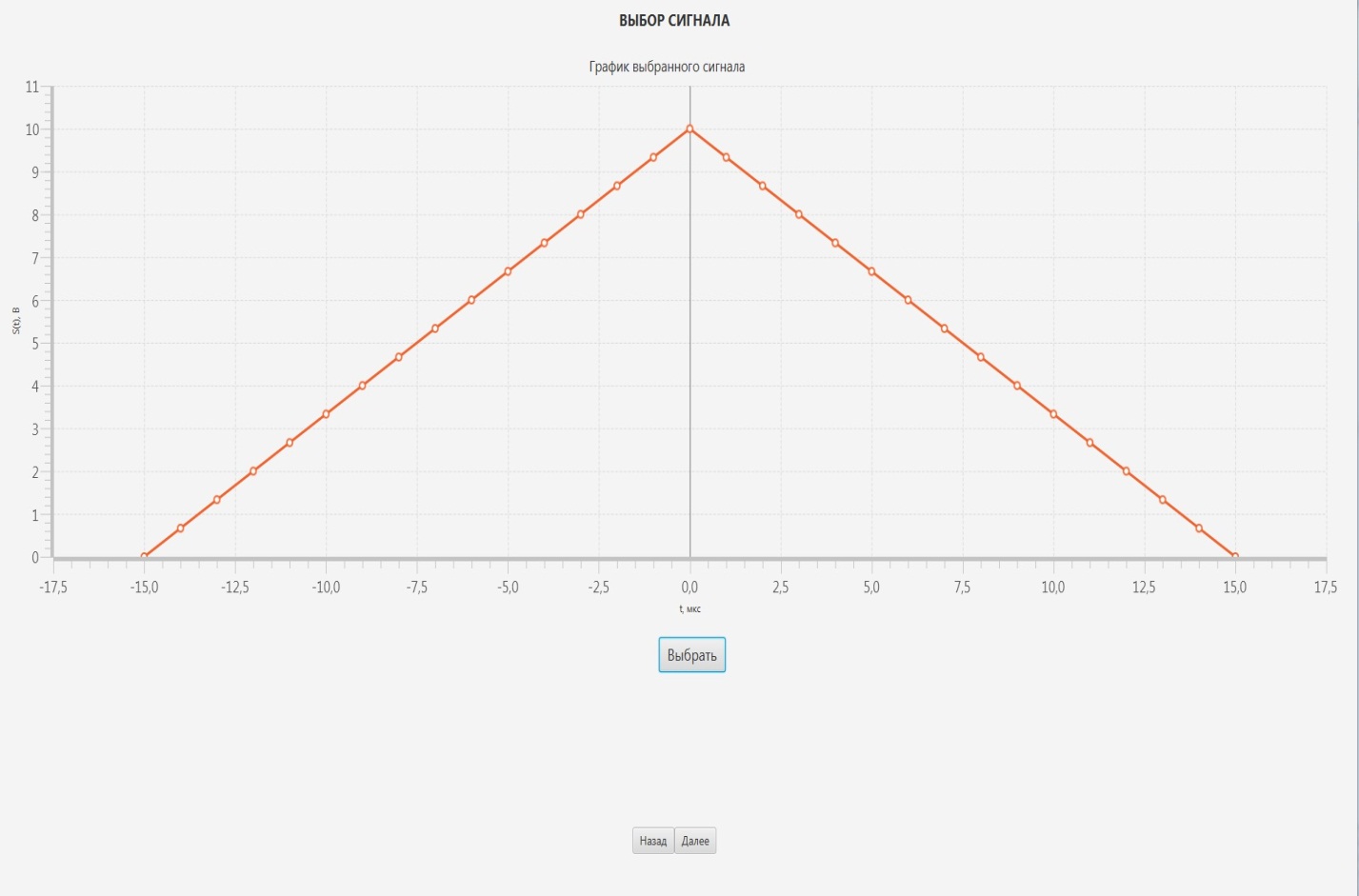


Рисунок 7.3 – Выбор сигнала для расчета автокорреляционной функции

По нажатию кнопки “Далее” пользователь попадает на страницу построения автокорреляционной функции.

7.1.4 Построение автокорреляционной функции

На странице построения автокорреляционной функции пользователь может ознакомиться с результатом построения автокорреляционной функции выбранного сигнала. Результат работы программы для треугольного видеоимпульса показан на *рисунке 7.4*

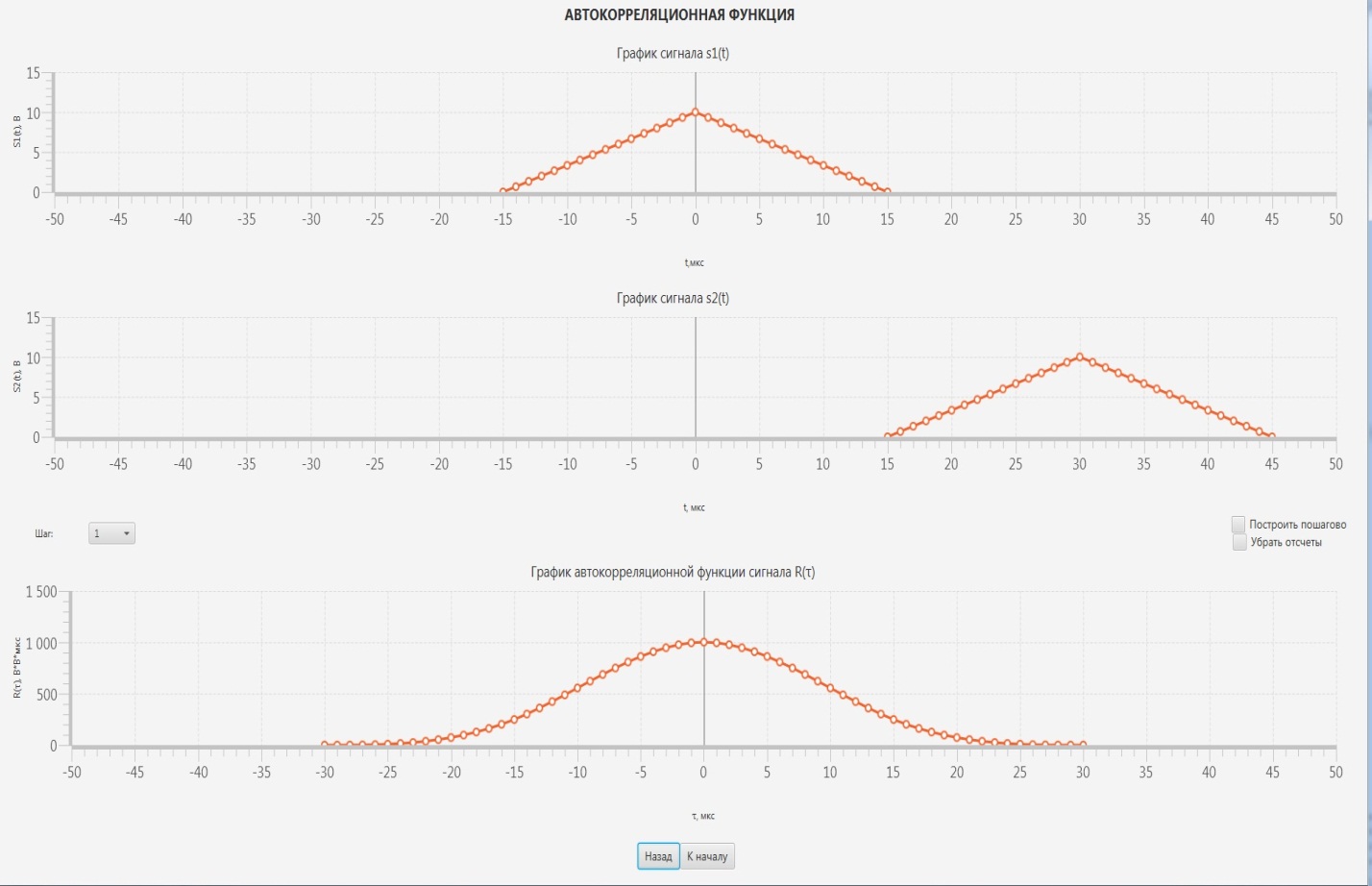


Рисунок 7.4 – Функция построенная автокорреляционная

На странице построения автокорреляционной функции можно задать шаг, с которым будет сдвигаться копия сигнала относительно оригинального сигнала.

Выбрав флаг “Построить пошагово” на странице построения автокорреляционной функции пользователь может сдвигать копию сигнала по временной оси с выбранным шагом в любую сторону путем нажатия клавиш ← →. Для каждого значения сдвига копии сигнала будет вычисляться автокорреляционная функция и динамически отображаться на графике. Результат работы программы для пошагового построения

автокорреляционной функции треугольного видеоимпульса показан на *рисунке 7.5*.

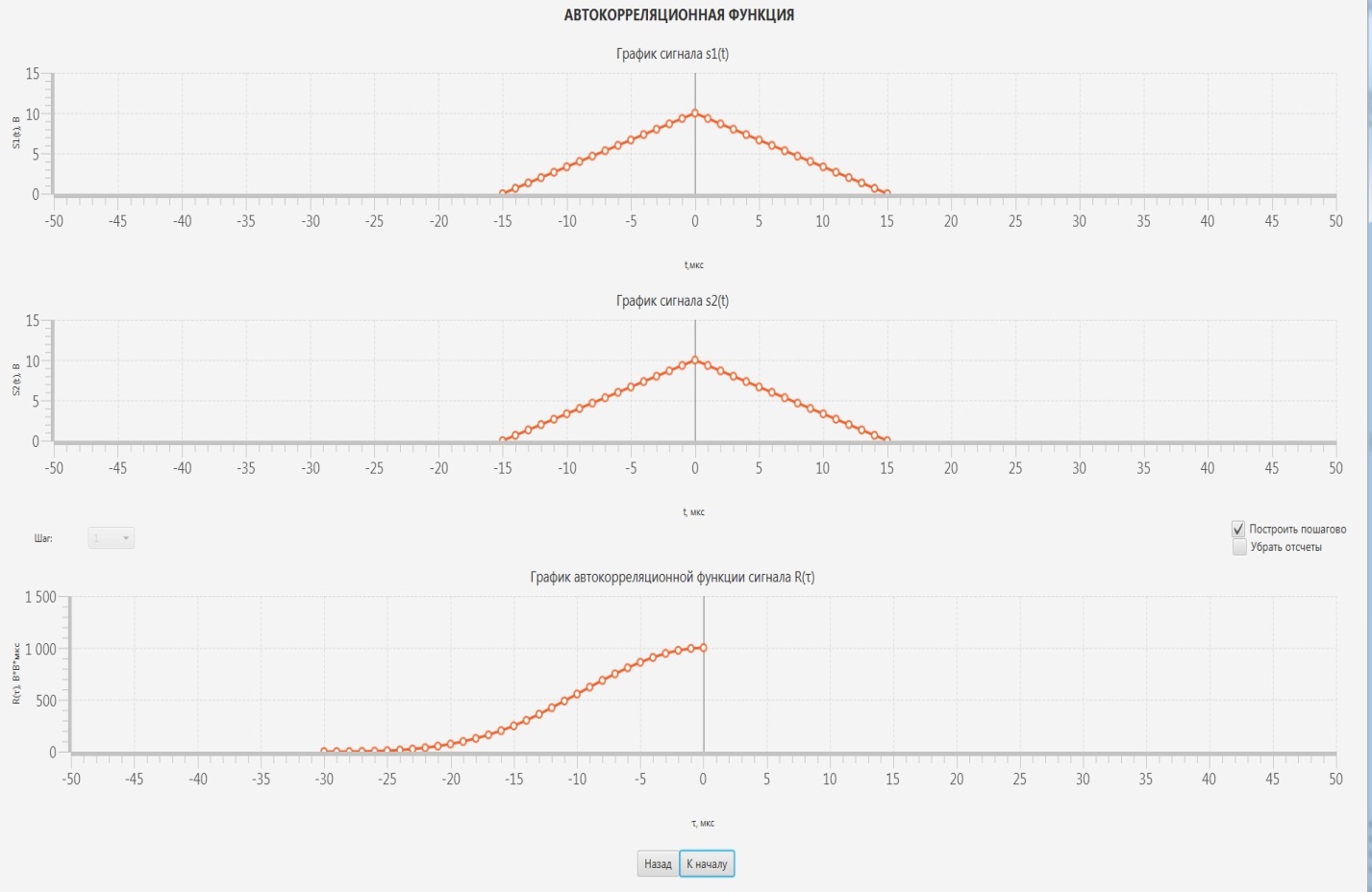


Рисунок 7.5 – Построение автокорреляционной функции пошаговое

Проанализировав форму автокорреляционной функции пользователь может вернуться на страницу выбора сигнала путем нажатия кнопки “Назад” или же вернуться на главную страницу с помощью нажатия кнопки “К началу”.

7.1.5 Выбор сигнала для расчета взаимокорреляционной функции

При нажатии на блок “Построение и анализ взаимокорреляционных функций” появится страница выбора сигналов для расчета взаимокорреляционной функции. По нажатию на кнопку “Выбрать” откроется диалоговое окно, где можно будет выбрать заранее созданный сигнал или созданный пользователем на странице создания нового сигнала. После того как сигнал будет выбран, его форма отобразится на графике. Для того, чтобы перейти на страницу построения взаимокорреляционной функции необходимо выбрать оба сигнала. Результат выбора треугольного видеоимпульса и прямоугольного видеоимпульса показан на *рисунке 7.6*.

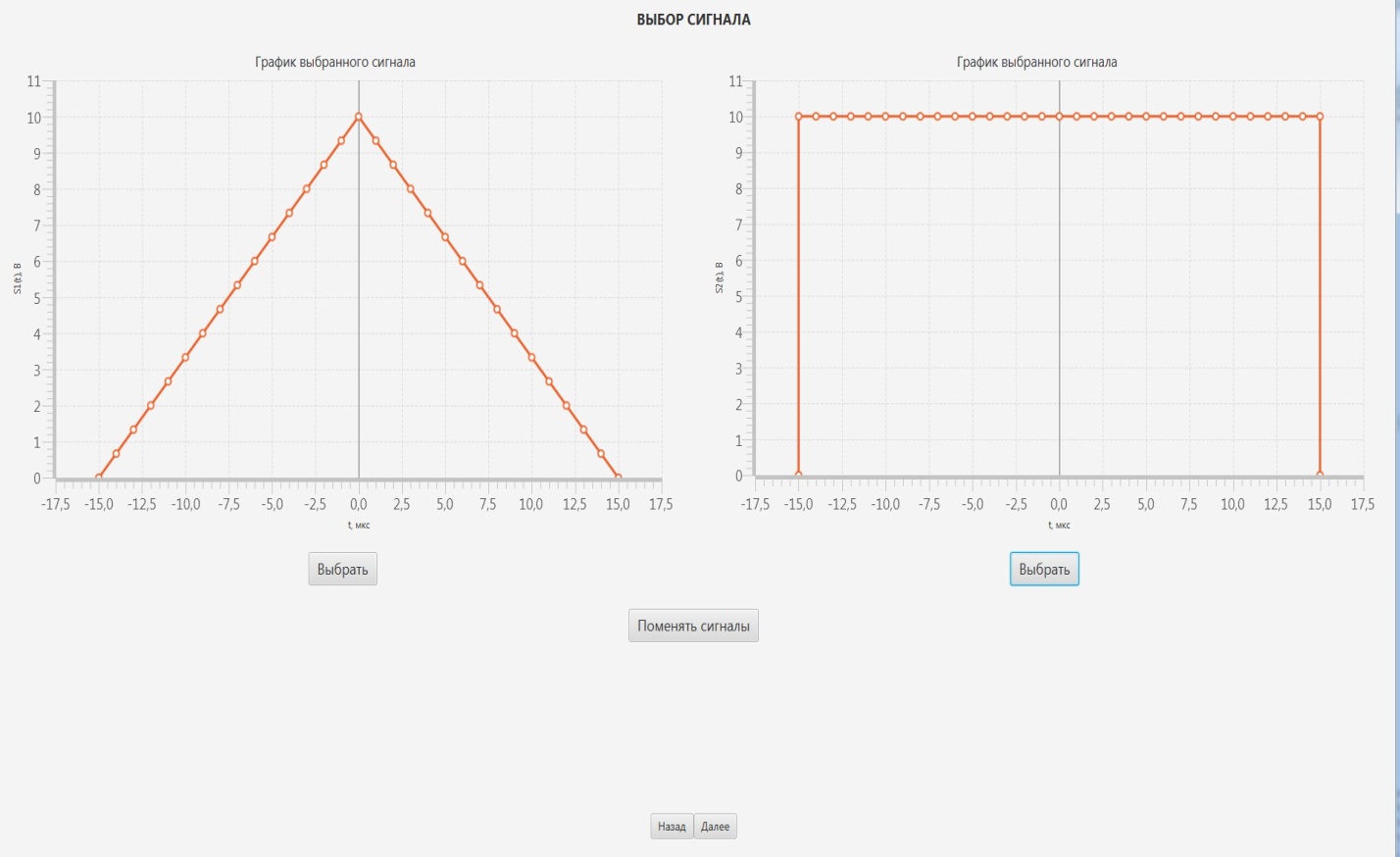


Рисунок 7.6 – Выбор сигналов для расчета взаимокорреляционной функции

Для того чтобы поменять сигналы местами необходимо нажать на кнопку "Поменять сигналы”.

По нажатию кнопки “Далее” пользователь попадает на страницу построения взаимокорреляционной функции.

7.1.6 Построение взаимокорреляционной функции

На странице построения взаимокорреляционной функции пользователь может ознакомиться с результатом построения взаимокорреляционной функции выбранных сигналов. Результат работы программы для

треугольного видеоимпульса и прямоугольного видеоимпульса показан на *рисунке 7.7*.

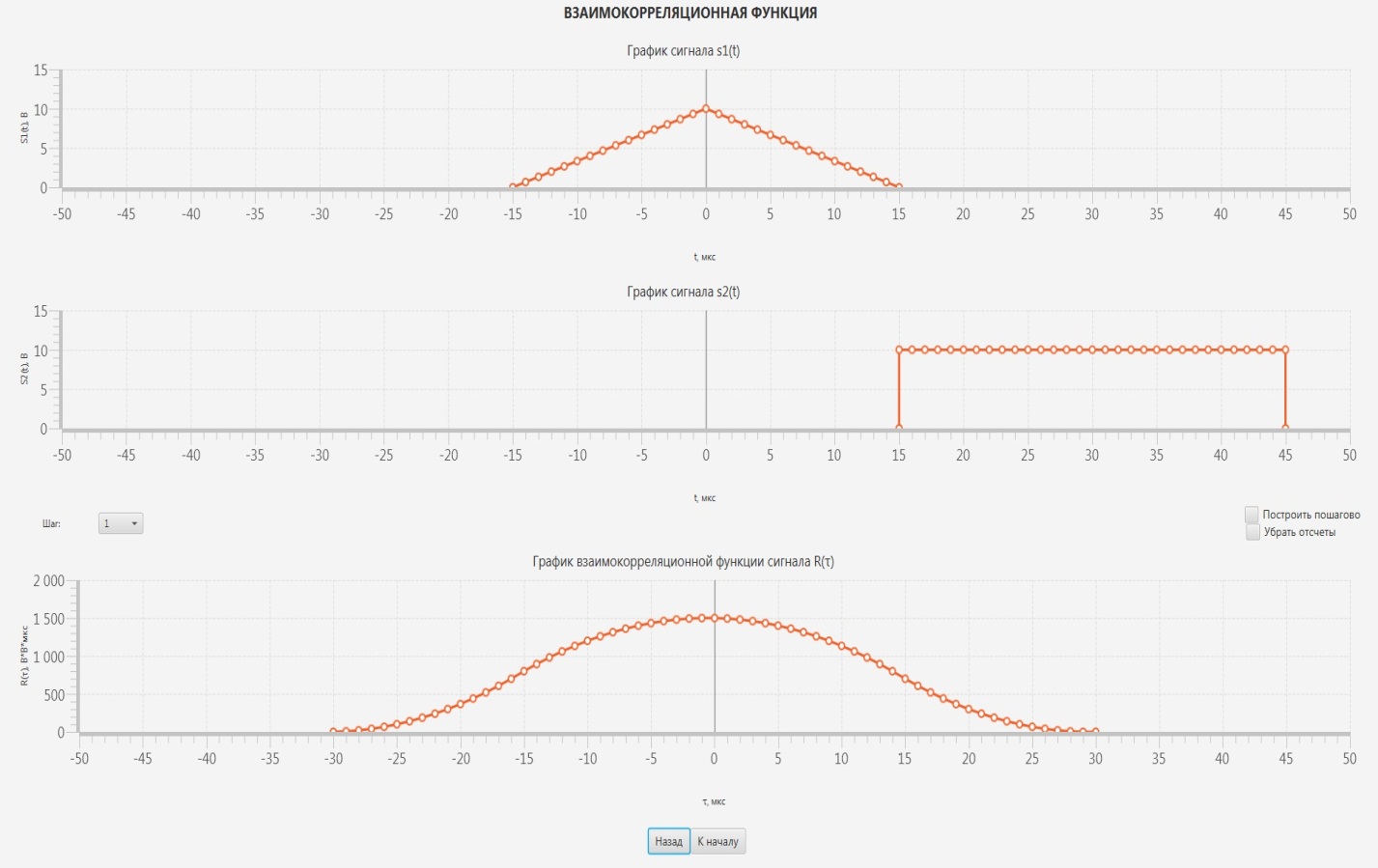


Рисунок 7.7 – Функция построенная взаимокорреляционная

На странице построения взаимокорреляционной функции можно задать шаг, с которым будет сдвигаться сигнал s2(t) относительно сигнала s1(t).

Выбрав флаг “Построить пошагово” на странице построения взаимокорреляционной функции пользователь может сдвигать сигнал s2(t) по временной оси с выбранным шагом в любую сторону путем нажатия клавиш ← →. Для каждого значения сдвига сигнала s2(t) будет вычисляться взаимокорреляционная функция и динамически отображаться на графике. Результат работы программы для пошагового построения

взаимокорреляционной функции треугольного и прямоугольного видеоимпульсов показан на *рисунке 7.8*.

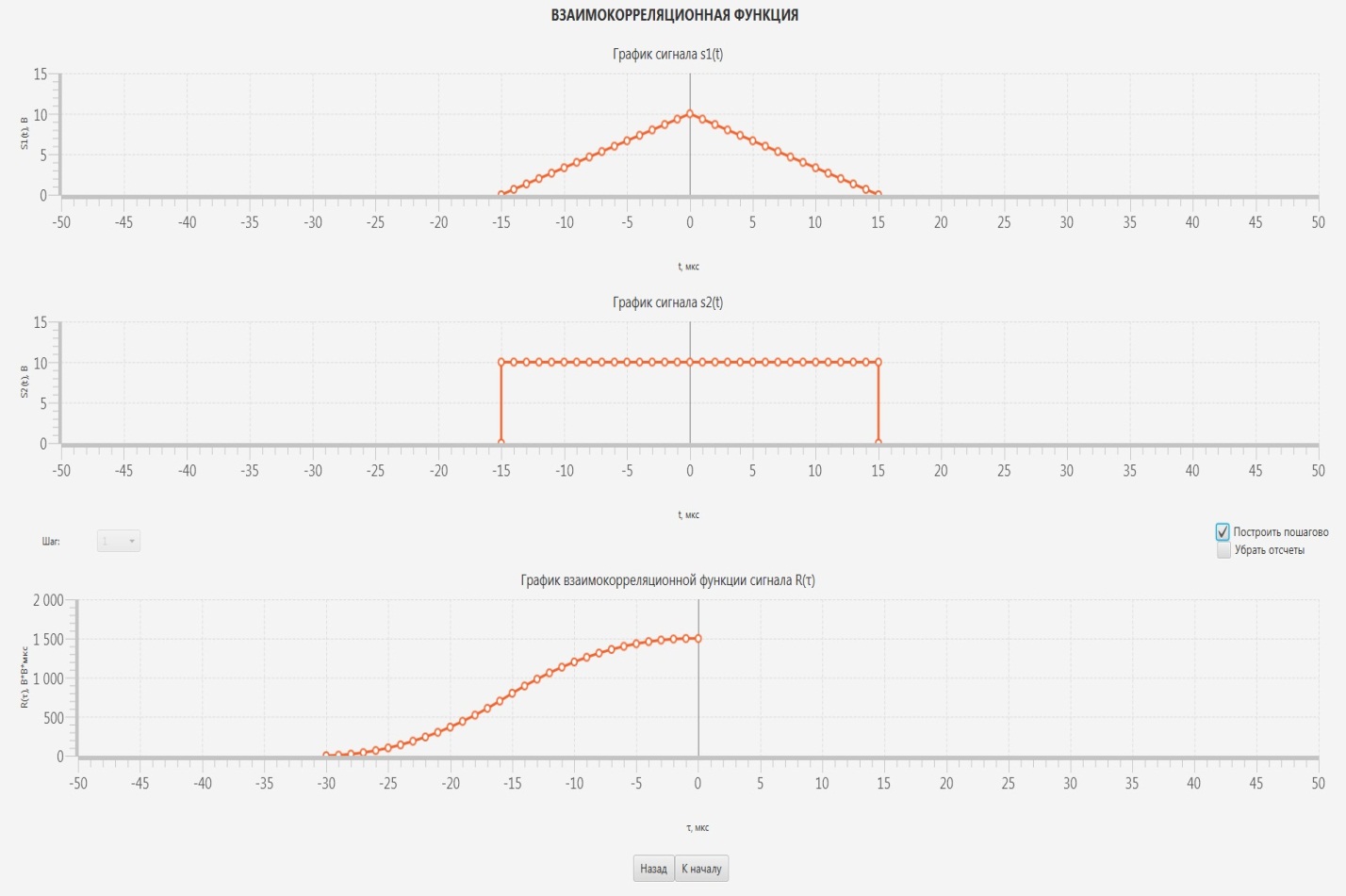


Рисунок 7.8 – Построение взаимокорреляционной функции пошаговое

Проанализировав форму взаимокорреляционной функции пользователь может вернуться на страницу выбора сигналов путем нажатия кнопки “Назад” или же вернуться на главную страницу с помощью нажатия кнопки “К началу”.

7.1.7 Справочная информация

На странице справки пользователь может просмотреть некоторую справочную информацию по тематике корреляционного анализа. Для того чтобы перейти на страницу справки необходимо нажать на блок “Справка” на главной странице программы. Результат работы программы при выборе

страницы “Справка” показан на *рисунке 7.9*

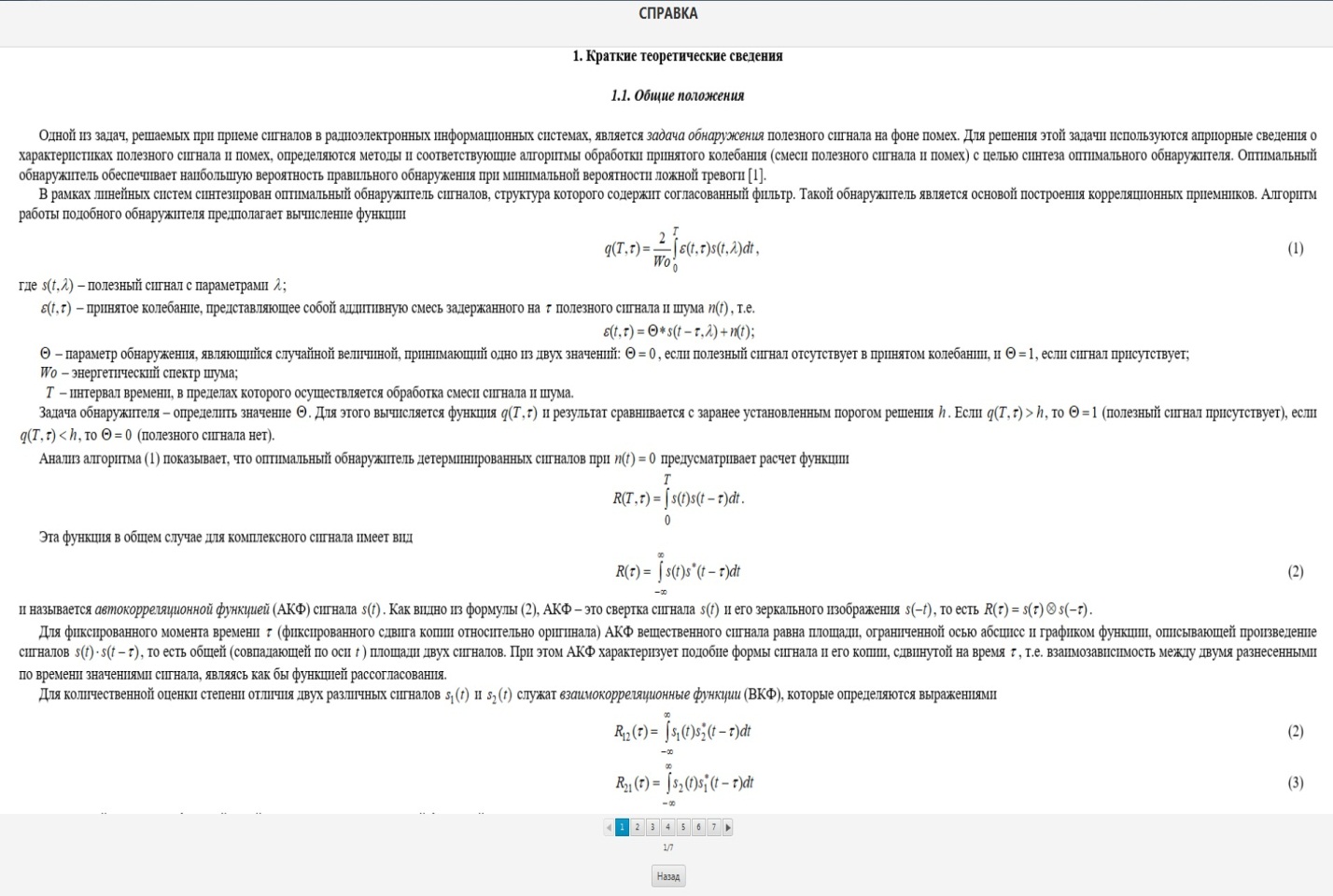


Рисунок 7.9 – Информация справочная

Для того чтобы переключить слайд со справочной информацией необходимо нажать на соответствующий номер страницы в меню пагинации.

Для того чтобы вернуться на главную страницу программы необходимо нажать на кнопку “Назад”.

Приведенную выше методику использования рекомендуется использовать в качестве методического пособия по работе с программой, как для студентов, так и для преподавателей (лаборантов).

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения дипломной работы был разработан программный анализатор корреляционных характеристик радиотехнических сигналов, создание которого было направлено на совершенствование процесса обучения и исследования корреляционных функций сигналов.

Для достижения поставленной задачи была обоснована необходимость определения автокорреляционной и взаимокорреляционной функций сигналов, произведён патентно-информационный поиск, выполнен анализ аппаратурных средств определения автокорреляционных и взаимокорреляционных функций сигналов (аналоговых и цифровых коррелометров), а так же функциональный анализ процессов определения корреляционных функций сигналов.

Были установлены алгоритмические и функциональные требования к программному анализатору корреляционных характеристик радиотехнических сигналов и на их основании был разработан общий алгоритм моделирования процессов определения корреляционных функций сигналов. Программа была написана на языке *Java*.

Также была описана методика применения программного анализатора корреляционных характеристик радиотехнических сигналов для обучения и исследования корреляционных функций.

В качестве технико-экономического обоснования дипломной работы был произведён расчёт экономических показателей разработки программного средства, где была рассчитана себестоимость разработки программного анализатора корреляционных характеристик радиотехнических сигналов на основе количества строк исходного текста программы.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Мирский Г. Я. Аппаратурное определение характеристик случайных процессов / Г. Я. Мирский. Изд. 2-е перераб. и доп., М., «Энергия», 1972. – 456 с. с ил.

[2] Надольский А. Н. Теоретические основы радиотехники: учеб. пособие для студ. спец. «Радиотехника», «Радиоинформатика» и «Радиотехнические системы» всех форм обуч. / А. Н. Надольский. – Мн.: БГУИР, 2005. – 232 с.: ил.

[3] Михнюк Т.Ф. Охрана труда: учеб. пособие для студентов учреждений, обеспечивающих получение высшего образования по специальностям в области радиоэлектроники и информатики / Т.Ф. Михнюк. – Минск: ИВЦ Минфина, 2007. – 320 с.

[4] ТКП 474-2013 (02300) Категорирование помещений, зданий и наружных установок по взрывопожарной и пожарной опасности (постановление Министерства по чрезвычайным ситуациям Республики Беларусь от 29.01.2013 № 4)

[5] Знаки категорийности помещений [Электронный ресурс]. – Электронные данные. – Режим доступа : <http://znaktb.by/products/znaki_8_1.html>

[6] Палицын В.А. Технико-экономическое обоснование дипломных проектов: Метод. пособие для студ. всех спец. БГУИР. В 4-х ч. Ч. 4: Проекты программного обеспечения / В.А. Палицын. – Минск: БГУИР, 2006. – 76 с.

[7] Чердынцев В.А. Оптимизация информационных систем. Уч. пособие / В.А. Чердынцев, В.В. Дубровский – Минск: БГУИР, 2005. – 180 с.

[8] Тихонов В.Н. Статистический анализ и синтез радиотехнических устройств и систем / В.Н. Тихонов – М.: Радио и связь, 1991. – 608 с.

[9] Ширман Я.Д. Основы теории обнаружения радиолокационных сигналов и измерения их параметров / Я.Д. Ширман, В.Н. Голиков – М.: Сов. радио, 1963. – 277 с.

[10] Системы программирования [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://inf.gym5cheb.ru/p20aa1.html>

[11] Системы программирования [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://pascal-programm.narod.ru/DelphiHarakteristika.html>

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Алгоритм расчёта построенных корреляционных функций**

да

Начало

Очистить предыдущий результат. Проверить сигналы на наличие

*i* = *begin*; *i* = *i* + 1

Сместить сдвигаемый сигнал вправо

*i*-й отсчёт больше отсчёта *end* ?

Определение границ сдвига сигнала s2(t) *begin* и *end,* количества отсчетов *length* сигнала s1(t) и значение сдвига *shift*. Задание сдвигаемого сигнала

нет

*с* = 1*+shift*; *с* = *с* + 1

Рассчитать пошагово сумму результатов перемножения сигналов

*с* <  *length*

Б

А

Продолжение приложения А

А

Б

*i* ≤ end

Умножение результата на шаг выборки

Конец

**ПРИЛОЖЕНИЕ Б**

**(обязательное)**

**Алгоритм пошагового расчёта корреляционных функций**

Начало

Проверить сигналы на наличие

Обработать нажатие клавиш и сдвинуть сигнал s2(t). Определить количество отсчетов *length* сигнала s1(t) и значение сдвига *shift*.

*с* = 1*+shift*; *с* = *с* + 1

Рассчитать пошагово сумму результатов перемножения сигналов

*с* <  *length*

Умножение результата на шаг выборки

Конец

**ПРИЛОЖЕНИЕ В**

**(обязательное)**

**Исходный код программного анализатора**

**Модуль создания сигналов**

public class CreateController implements Initializable{

private ObservableList<Point> pointsData = FXCollections.observableArrayList();

private double[] signalY;

private double[] signalX;

private int begin;

private int end;

private double amplitude;

private int freq;

private String currentSignal;

@FXML

public TableView<Point> points;

@FXML

private TableColumn<Point, String> coordX;

@FXML

private TableColumn<Point, String> coordY;

@FXML

public TextField endTextField;

@FXML

public TextField beginTextField;

@FXML

public TextField amplitudeTextField;

@FXML

public LineChart constructedSignalChart;

@FXML

public TextField xAddField;

@FXML

public TextField yAddField;

@FXML

public ChoiceBox signals;

@FXML

public TextField freqTextField;

public void createTable(ActionEvent actionEvent) {

pointsData.clear();

freq=1;

double amplitude=0;

String beginStr=beginTextField.getCharacters().toString();

String endStr=endTextField.getCharacters().toString();

String amplitudeStr=amplitudeTextField.getCharacters().toString();

String freqStr=freqTextField.getCharacters().toString();

String[] freqSignalsStr={"Косинусоидальный сигнал","Синусоидальный сигнал","Последовательность прямоугольных видеоимпульсов при s(t)>0",

"Последовательность прямоугольных видеоимпульсов при s(t)>0 и s(t)<0"};

List<String> freqSignalsList=Arrays.asList(freqSignalsStr);

if(freqSignalsList.contains(signals.getSelectionModel().getSelectedItem().toString()) && !freqStr.matches("-?[0-9]+([0-9]+)?")){

return;

}

else if(freqSignalsList.contains(signals.getSelectionModel().getSelectedItem().toString())){

freq=Integer.valueOf(freqTextField.getCharacters().toString());;

}

if(!beginStr.matches("-?[0-9]+")||!endStr.matches("-?[0-9]+")||!amplitudeStr.matches("-?[0-9]+([0-9]+)?")){

return;

}

begin=Integer.valueOf(beginTextField.getCharacters().toString());

end=Integer.valueOf(endTextField.getCharacters().toString());

amplitude=Double.valueOf(amplitudeTextField.getCharacters().toString());

if(begin>end || freq<0){

return;

}

pointsData.addAll(SignalBuilder.buildSignal(signals.getSelectionModel().getSelectedItem().toString(),amplitude,begin,end, freq));

points.setItems(pointsData);

}

@Override

public void initialize(URL location, ResourceBundle resources) {

coordX.setCellValueFactory(new PropertyValueFactory<Point, String>("x"));

coordY.setCellValueFactory(new PropertyValueFactory<Point, String>("y"));

coordY.setCellFactory(TextFieldTableCell.forTableColumn());

coordY.setOnEditCommit(

new EventHandler<TableColumn.CellEditEvent<Point, String>>() {

@Override

public void handle(TableColumn.CellEditEvent<Point, String> t) {

((Point) t.getTableView().getItems().get(

t.getTablePosition().getRow())

).setY(t.getNewValue());

}

}

);

String[] signalsStr={"Прямоугольный видеоимпульс","Треугольный видеоимпульс","Прямой треугольный видеоимпульс",

"Косинусоидальный сигнал","Синусоидальный сигнал","Последовательность прямоугольных видеоимпульсов при s(t)>0",

"Последовательность прямоугольных видеоимпульсов при s(t)>0 и s(t)<0"};

signals.getItems().addAll(signalsStr);

signals.getSelectionModel().select(0);

}

public void saveFile(ActionEvent actionEvent) {

if(begin==end){

return;

}

try {

FileOpener.saveFile(pointsData, begin, end, 1);

}

catch (IOException exception){

}

}

public void constructSignal(ActionEvent actionEvent) {

if(begin==end){

return;

}

signalY=new double[pointsData.size()];

signalX=new double[pointsData.size()];

int currentDiscret=0;

for(Point point:pointsData){

signalY[currentDiscret]=Double.valueOf(point.getY());

signalX[currentDiscret]=Double.valueOf(point.getX());

currentDiscret++;

}

ChartDrawer.drawSignal(constructedSignalChart,begin,end,signalY, signalX,1);

}

public void moveToMain(ActionEvent actionEvent) {

SceneSelector.chooseScene("MAIN\_SCENE");

}

public void addPoint(ActionEvent actionEvent) {

int index=0;

int prevIndex=0;

String xCoord=xAddField.getCharacters().toString();

String yCoords=yAddField.getCharacters().toString();

if(!xCoord.matches("-?[0-9]+")||!yCoords.matches("-?[0-9]+([0-9]+)?")){

return;

}

ListIterator<Point> iterator=pointsData.listIterator();

if(Integer.valueOf(xCoord)>end){

end++;

pointsData.add(new Point(String.valueOf(end),yCoords));

return;

}

if(Integer.valueOf(xCoord)<begin){

begin--;

pointsData.add(0,new Point(String.valueOf(begin),yCoords));

return;

}

while(iterator.hasNext()){

Point point=iterator.next();

if(Integer.valueOf(point.getX())<Integer.valueOf(xCoord)){

prevIndex=iterator.previousIndex();

}

}

while(iterator.hasPrevious()){

Point point=iterator.previous();

if(point.getX().equals(xCoord)){

index=iterator.nextIndex();

break;

}

}

if(index!=0) {

pointsData.add(index + 1, new Point(xCoord, yCoords));

}

else{

pointsData.add(prevIndex+1, new Point(xCoord, yCoords));

}

}

public void deletePoint(ActionEvent actionEvent) {

String xCoord=xAddField.getCharacters().toString();

String yCoords=yAddField.getCharacters().toString();

ListIterator<Point> iterator=pointsData.listIterator();

while(iterator.hasNext()){

Point point=iterator.next();

if(point.getX().equals(xCoord) && point.getY().equals(yCoords)){

iterator.remove();

}

}

}

}

public class SignalBuilder {

public static List<Point> buildSignal(String signalType, double amplitude, int begin, int end, int freq){

List<Point> points=new ArrayList<>();

double value=0;

int pointZero=end-(end-begin)/2;

int impulsPeriod=(Math.abs(end-begin)/(freq\*2));

if(signalType.equals("Прямоугольный видеоимпульс")){

points.add(new Point(""+begin+"",""+0+""));

for(int i=begin;i<end+1;i++){

points.add(new Point(""+i+"",""+amplitude+""));

}

points.add(new Point(""+end+"",""+0+""));

}

if(signalType.equals("Треугольный видеоимпульс")){

for(int i=begin;i<end+1;i++){

if(i<pointZero){

value=(1+(2\*(double)(i-pointZero)/(end-begin)))\*amplitude;

}

else{

value=(1-(2\*(double)(i-pointZero)/(end-begin)))\*amplitude;

}

points.add(new Point(""+i+"",""+value+""));

}

}

if(signalType.equals("Прямой треугольный видеоимпульс")){

for(int i=begin;i<end+1;i++){

value=(1+(2\*(double)i/(end-begin)))\*amplitude/2;

points.add(new Point(""+i+"",""+value+""));

}

points.add(new Point(""+end+"",""+0+""));

}

if(signalType.equals("Косинусоидальный сигнал")){

for(int i=begin;i<end+1;i++){

value=amplitude\*cos(2\*Math.PI\*i\*freq/(end-begin));

points.add(new Point(""+i+"",""+value+""));

}

}

if(signalType.equals("Синусоидальный сигнал")){

for(int i=begin;i<end+1;i++){

value=amplitude\*sin(2\*Math.PI\*i\*freq/(end-begin));

points.add(new Point(""+i+"",""+value+""));

}

}

if(signalType.equals("Последовательность прямоугольных видеоимпульсов при s(t)>0")){

int impulsCounter=0;

for(int i=0;i<(impulsPeriod/2);i++){

points.add(new Point("" + (i+begin) + "", "" + value + ""));

}

impulsCounter=impulsPeriod-1;

for(int i=begin+impulsPeriod/2;i<end+1;i++){

points.add(new Point(""+i+"",""+value+""));

impulsCounter++;

if(impulsCounter==impulsPeriod){

if(value>amplitude-1){

value=0;

}

else{

value=amplitude;

}

points.add(new Point(""+i+"",""+value+""));

impulsCounter=0;

}

}

}

if(signalType.equals("Последовательность прямоугольных видеоимпульсов при s(t)>0 и s(t)<0")){

int impulsCounter=0;

value=-amplitude;

for(int i=0;i<(impulsPeriod/2);i++){

points.add(new Point("" + (i+begin) + "", "" + value + ""));

}

impulsCounter=impulsPeriod-1;

for(int i=begin+impulsPeriod/2;i<end+1;i++){

points.add(new Point(""+i+"",""+value+""));

impulsCounter++;

if(impulsCounter==impulsPeriod){

if(value>amplitude-1){

value=-amplitude;

}

else{

value=amplitude;

}

points.add(new Point(""+i+"",""+value+""));

impulsCounter=0;

}

}

}

return points;

}

}

public class Point {

String x;

String y;

public Point(String x, String y) {

this.x = x;

this.y = y;

}

public Point() {

}

public String getX() {

return x;

}

public void setX(String x) {

this.x = x;

}

public String getY() {

return y;

}

public void setY(String y) {

this.y = y;

}

}

**Главный модуль**

public class Main extends Application {

private Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();

@Override

public void start(Stage primaryStage) throws Exception{

primaryStage.setTitle("Анализатор корреляционных характеристик");

SceneSelector.setPrimaryStage(primaryStage);

loadScene("autokorrel.fxml","AUTO\_KORREL\_SCENE");

loadScene("crosskorrel.fxml","CROSS\_KORREL\_SCENE");

loadScene("chooseAuto.fxml","CHOOSE\_AUTO\_SCENE");

loadScene("chooseCross.fxml","CHOOSE\_CROSS\_SCENE");

loadScene("main.fxml","MAIN\_SCENE");

loadScene("creator.fxml","CREATE\_SCENE");

loadScene("help.fxml","HELP\_SCENE");

SceneSelector.chooseScene("MAIN\_SCENE");

}

private void loadScene(String resource, String key) throws Exception{

FXMLLoader fxmlLoader=new FXMLLoader(getClass().getResource(resource));

Parent root = fxmlLoader.load();

if(fxmlLoader.getController() instanceof Controller) {

SceneSelector.addController(key, fxmlLoader.getController());

}

SceneSelector.addScene(key,new Scene(root, dim.getWidth(), dim.getHeight()-75));

}

public static void main(String[] args) {

launch(args);

}

}

public class SignalKeeper {

private String type;

private int amplitude;

private int begin;

private int end;

private double[] signal;

private int duration;

private int step;

private double[] originSignal;

private boolean changed;

private boolean moved;

private double[] signalX;

private double[] signalY;

private int shiftBegin;

private int shiftEnd;

public String getType() {

return type;

}

public void setType(String type) {

this.type = type;

}

public int getAmplitude() {

return amplitude;

}

public void setAmplitude(int amplitude) {

this.amplitude = amplitude;

}

public int getBegin() {

return begin;

}

public void setBegin(int begin) {

this.begin = begin;

}

public int getEnd() {

return end;

}

public void setEnd(int end) {

this.end = end;

}

public double[] getSignal() {

return signal;

}

public void setSignal(double[] signal) {

this.signal = signal;

}

public int getDuration() {

return duration;

}

public void setDuration(int duration) {

this.duration = duration;

}

public int getStep() {

return step;

}

public void setStep(int step) {

this.step = step;

}

public double[] getOriginSignal() {

return originSignal;

}

public void setOriginSignal(double[] originSignal) {

this.originSignal = originSignal;

}

public boolean isChanged() {

return changed;

}

public void setChanged(boolean changed) {

this.changed = changed;

}

public boolean isMoved() {

return moved;

}

public void setMoved(boolean moved) {

this.moved = moved;

}

public double[] getSignalX() {

return signalX;

}

public void setSignalX(double[] signalX) {

this.signalX = signalX;

}

public double[] getSignalY() {

return signalY;

}

public void setSignalY(double[] signalY) {

this.signalY = signalY;

}

public int getShiftBegin() {

return shiftBegin;

}

public void setShiftBegin(int shiftBegin) {

this.shiftBegin = shiftBegin;

}

public int getShiftEnd() {

return shiftEnd;

}

public void setShiftEnd(int shiftEnd) {

this.shiftEnd = shiftEnd;

}

}

public class StrategyKeeper {

private boolean stepStrategy;

private int begin;

private int end;

private int defaultBegin;

private int defaultEnd;

private double[] signal;

private int stepStrategyBegin;

public boolean isStepStrategy() {

return stepStrategy;

}

public void setStepStrategy(boolean stepStrategy) {

this.stepStrategy = stepStrategy;

}

public int getBegin() {

return begin;

}

public void setBegin(int begin) {

this.begin = begin;

}

public int getEnd() {

return end;

}

public void setEnd(int end) {

this.end = end;

}

public int getDefaultBegin() {

return defaultBegin;

}

public void setDefaultBegin(int defaultBegin) {

this.defaultBegin = defaultBegin;

}

public int getDefaultEnd() {

return defaultEnd;

}

public void setDefaultEnd(int defaultEnd) {

this.defaultEnd = defaultEnd;

}

public double[] getSignal() {

return signal;

}

public void setSignal(double[] signal) {

this.signal = signal;

}

public int getStepStrategyBegin() {

return stepStrategyBegin;

}

public void setStepStrategyBegin(int stepStrategyBegin) {

this.stepStrategyBegin = stepStrategyBegin;

}

public void copySignal(SignalKeeper signalKeeper1, SignalKeeper signalKeeper2) {

double signal2[]=new double[signalKeeper1.getSignal().length+(signalKeeper2.getSignal().length-1)\*2];

for(int i=0;i<signalKeeper2.getSignal().length;i++){

signal2[i]=signalKeeper2.getSignal()[i];

}

this.signal=signal2;

}

}

public class SceneSelector {

private static Stage primaryStage;

private static Map<String, Scene> scenes = new HashMap<String, Scene>();

private static Map<String, Controller> controllers=new HashMap<>();

private static SignalKeeper autoSignalKeeper=new SignalKeeper();

private static SignalKeeper crossSignalKeeper1=new SignalKeeper();

private static SignalKeeper crossSignalKeeper2=new SignalKeeper();

private static StrategyKeeper autoStrategyKeeper=new StrategyKeeper();

private static StrategyKeeper crossStrategyKeeper=new StrategyKeeper();

public static Stage getPrimaryStage() {

return primaryStage;

}

public static void setPrimaryStage(Stage primaryStageArg) {

primaryStage = primaryStageArg;

}

public static Map<String, Scene> getScenes() {

return scenes;

}

public static void addScene(String key, Scene value) {

scenes.put(key, value);

}

public static void chooseScene(String key) {

primaryStage.setScene(scenes.get(key));

primaryStage.show();

}

public static void addController(String name, Controller controller){

controllers.put(name,controller);

}

public static Controller getController(String name){

return controllers.get(name);

}

public static SignalKeeper getAutoSignalKeeper() {

return autoSignalKeeper;

}

public static SignalKeeper getCrossSignalKeeper1() {

return crossSignalKeeper1;

}

public static SignalKeeper getCrossSignalKeeper2() {

return crossSignalKeeper2;

}

public static StrategyKeeper getAutoStrategyKeeper() {

return autoStrategyKeeper;

}

public static StrategyKeeper getCrossStrategyKeeper() {

return crossStrategyKeeper;

}

public static void setCrossSignalKeeper1(SignalKeeper crossSignalKeeper1) {

SceneSelector.crossSignalKeeper1 = crossSignalKeeper1;

}

public static void setCrossSignalKeeper2(SignalKeeper crossSignalKeeper2) {

SceneSelector.crossSignalKeeper2 = crossSignalKeeper2;

}

}

public class Controller{

@FXML

public LineChart chooseAutoChart;

@FXML

public LineChart autoKorrelChart1;

@FXML

public LineChart autoKorrelChart3;

@FXML

public LineChart autoKorrelChart2;

@FXML

public LineChart chooseCrossChart1;

@FXML

public LineChart chooseCrossChart2;

@FXML

public LineChart crossKorrelChart1;

@FXML

public LineChart crossKorrelChart2;

@FXML

public LineChart crossKorrelChart3;

@FXML

public ComboBox autoComboBox;

@FXML

public CheckBox autoCheckBox;

@FXML

public ComboBox crossComboBox;

@FXML

public CheckBox crossCheckBox;

@FXML

public NumberAxis autoChart3AxisY;

@FXML

public NumberAxis crossChart3AxisY;

@FXML

public NumberAxis autoChart1AxisX;

@FXML

public NumberAxis autoChart2AxisX;

@FXML

public NumberAxis autoChart3AxisX;

@FXML

public NumberAxis crossChart1AxisX;

@FXML

public NumberAxis crossChart2AxisX;

@FXML

public NumberAxis crossChart3AxisX;

@FXML

public CheckBox autoKorrelDeleteDiscretsCheckBox;

@FXML

public CheckBox crossKorrelDeleteDiscretsCheckBox;

public void moveFromMainToChooseAuto(MouseEvent mouseEvent) {

SceneSelector.chooseScene("CHOOSE\_AUTO\_SCENE");

}

public void moveFromMainToChooseCross(MouseEvent mouseEvent) {

SceneSelector.chooseScene("CHOOSE\_CROSS\_SCENE");

}

public void moveFromAutoKorrelToChooseAuto(ActionEvent actionEvent) {

SceneSelector.getController("AUTO\_KORREL\_SCENE").autoComboBox.getSelectionModel().select(0);

if(SceneSelector.getAutoStrategyKeeper().isStepStrategy()) {

SceneSelector.getController("AUTO\_KORREL\_SCENE").autoChart3AxisY.setAutoRanging(true);

SceneSelector.getAutoSignalKeeper().setBegin(SceneSelector.getAutoStrategyKeeper().getDefaultBegin());

SceneSelector.getAutoSignalKeeper().setEnd(SceneSelector.getAutoStrategyKeeper().getDefaultEnd());

ChartBuilder.returnChart(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoChart2AxisX,SceneSelector.getAutoSignalKeeper());

SceneSelector.getController("AUTO\_KORREL\_SCENE").autoComboBox.setDisable(false);

SceneSelector.getAutoStrategyKeeper().setStepStrategy(false);

SceneSelector.getController("AUTO\_KORREL\_SCENE").autoCheckBox.setSelected(false);

}

SceneSelector.chooseScene("CHOOSE\_AUTO\_SCENE");

}

public void moveFromCrossKorrelToChooseCross(ActionEvent actionEvent) {

SceneSelector.getController("CROSS\_KORREL\_SCENE").crossComboBox.getSelectionModel().select(0);

if(SceneSelector.getCrossStrategyKeeper().isStepStrategy()) {

SceneSelector.getController("CROSS\_KORREL\_SCENE").crossChart3AxisY.setAutoRanging(true);

SceneSelector.getCrossSignalKeeper2().setBegin(SceneSelector.getCrossStrategyKeeper().getDefaultBegin());

SceneSelector.getCrossSignalKeeper2().setEnd(SceneSelector.getCrossStrategyKeeper().getDefaultEnd());

ChartBuilder.returnChart(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossChart2AxisX,SceneSelector.getCrossSignalKeeper2());

SceneSelector.getController("CROSS\_KORREL\_SCENE").crossComboBox.setDisable(false);

SceneSelector.getCrossStrategyKeeper().setStepStrategy(false);

SceneSelector.getController("CROSS\_KORREL\_SCENE").crossCheckBox.setSelected(false);

}

SceneSelector.chooseScene("CHOOSE\_CROSS\_SCENE");

}

public void moveFromChooseAutoToMain(ActionEvent actionEvent) {

SceneSelector.chooseScene("MAIN\_SCENE");

}

public void moveFromChooseAutoToAutoKorrel(ActionEvent actionEvent) {

if(SceneSelector.getAutoSignalKeeper().isChanged()) {

int step=(int)SceneSelector.getController("AUTO\_KORREL\_SCENE").autoComboBox.getSelectionModel().getSelectedItem();

ChartDrawer.drawSignal(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoKorrelChart1, SceneSelector.getAutoSignalKeeper());

ChartDrawer.drawShiftSignal(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoKorrelChart2, SceneSelector.getAutoSignalKeeper());

ChartDrawer.drawKorrelSignal(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoKorrelChart3, SceneSelector.getAutoSignalKeeper(), SceneSelector.getAutoSignalKeeper(), step, SceneSelector.getAutoStrategyKeeper());

SceneSelector.getAutoSignalKeeper().setChanged(false);

TooltipSetter.addTooltips(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoKorrelChart3);

}

SceneSelector.chooseScene("AUTO\_KORREL\_SCENE");

}

public void moveFromChooseCrossToMain(ActionEvent actionEvent) {

SceneSelector.chooseScene("MAIN\_SCENE");

}

public void moveFromChooseCrossToCrossKorrel(ActionEvent actionEvent) {

if(SceneSelector.getCrossSignalKeeper2().getSignal()==null || SceneSelector.getCrossSignalKeeper1().getSignal()==null){

return;

}

if(SceneSelector.getCrossSignalKeeper1().isChanged() || SceneSelector.getCrossSignalKeeper2().isChanged()) {

int step = (int) SceneSelector.getController("CROSS\_KORREL\_SCENE").crossComboBox.getSelectionModel().getSelectedItem();

ChartDrawer.drawSignal(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossKorrelChart1, SceneSelector.getCrossSignalKeeper1());

ChartDrawer.drawShiftSignal(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossKorrelChart2, SceneSelector.getCrossSignalKeeper2());

ChartDrawer.drawKorrelSignal(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossKorrelChart3, SceneSelector.getCrossSignalKeeper1(), SceneSelector.getCrossSignalKeeper2(), step, SceneSelector.getCrossStrategyKeeper());

SceneSelector.getCrossSignalKeeper1().setChanged(false);

SceneSelector.getCrossSignalKeeper2().setChanged(false);

TooltipSetter.addTooltips(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossKorrelChart3);

}

SceneSelector.chooseScene("CROSS\_KORREL\_SCENE");

}

public void moveFromAutoKorrelToMain(ActionEvent actionEvent) {

SceneSelector.getController("AUTO\_KORREL\_SCENE").autoComboBox.getSelectionModel().select(0);

if(SceneSelector.getAutoStrategyKeeper().isStepStrategy()) {

SceneSelector.getController("AUTO\_KORREL\_SCENE").autoChart3AxisY.setAutoRanging(true);

SceneSelector.getAutoSignalKeeper().setBegin(SceneSelector.getAutoStrategyKeeper().getDefaultBegin());

SceneSelector.getAutoSignalKeeper().setEnd(SceneSelector.getAutoStrategyKeeper().getDefaultEnd());

ChartBuilder.returnChart(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoChart2AxisX,SceneSelector.getAutoSignalKeeper());

SceneSelector.getController("AUTO\_KORREL\_SCENE").autoComboBox.setDisable(false);

SceneSelector.getAutoStrategyKeeper().setStepStrategy(false);

SceneSelector.getController("AUTO\_KORREL\_SCENE").autoCheckBox.setSelected(false);

}

SceneSelector.chooseScene("MAIN\_SCENE");

}

public void moveFromCrossKorrelToMain(ActionEvent actionEvent) {

SceneSelector.getController("CROSS\_KORREL\_SCENE").crossComboBox.getSelectionModel().select(0);

if(SceneSelector.getCrossStrategyKeeper().isStepStrategy()) {

SceneSelector.getController("CROSS\_KORREL\_SCENE").crossChart3AxisY.setAutoRanging(true);

SceneSelector.getCrossSignalKeeper2().setBegin(SceneSelector.getCrossStrategyKeeper().getDefaultBegin());

SceneSelector.getCrossSignalKeeper2().setEnd(SceneSelector.getCrossStrategyKeeper().getDefaultEnd());

ChartBuilder.returnChart(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossChart2AxisX,SceneSelector.getCrossSignalKeeper2());

SceneSelector.getController("CROSS\_KORREL\_SCENE").crossComboBox.setDisable(false);

SceneSelector.getCrossStrategyKeeper().setStepStrategy(false);

SceneSelector.getController("CROSS\_KORREL\_SCENE").crossCheckBox.setSelected(false);

}

SceneSelector.chooseScene("MAIN\_SCENE");

}

public void chooseAutoChart(ActionEvent actionEvent) {

File file= FileOpener.openFile();

if(file==null){

return;

}

Parser.parseFile(file,SceneSelector.getAutoSignalKeeper());

SceneSelector.getAutoSignalKeeper().setChanged(true);

int difference=SceneSelector.getAutoSignalKeeper().getDuration();

SceneSelector.getAutoSignalKeeper().setShiftBegin(SceneSelector.getAutoSignalKeeper().getBegin()+SceneSelector.getAutoSignalKeeper().getDuration());

SceneSelector.getAutoSignalKeeper().setShiftEnd(SceneSelector.getAutoSignalKeeper().getEnd()+SceneSelector.getAutoSignalKeeper().getDuration());

ComboBoxCreator.createComboBox(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoComboBox, SceneSelector.getAutoSignalKeeper().getStep(),difference);

ChartBuilder.buildChart(SceneSelector.getAutoSignalKeeper(),SceneSelector.getAutoSignalKeeper(),true);

ChartDrawer.drawSignal(SceneSelector.getController("CHOOSE\_AUTO\_SCENE").chooseAutoChart,SceneSelector.getAutoSignalKeeper());

}

public void chooseCrossChart1(ActionEvent actionEvent){

File file= FileOpener.openFile();

if(file==null){

return;

}

Parser.parseFile(file,SceneSelector.getCrossSignalKeeper1());

SceneSelector.getCrossSignalKeeper1().setChanged(true);

int difference1=SceneSelector.getCrossSignalKeeper1().getDuration();

int difference2=SceneSelector.getCrossSignalKeeper2().getDuration();

SceneSelector.getCrossSignalKeeper1().setShiftBegin(SceneSelector.getCrossSignalKeeper1().getBegin());

SceneSelector.getCrossSignalKeeper1().setShiftEnd(SceneSelector.getCrossSignalKeeper1().getEnd());

if(SceneSelector.getCrossSignalKeeper2()!=null){

int shiftBegin=-(SceneSelector.getCrossSignalKeeper2().getBegin()-SceneSelector.getCrossSignalKeeper1().getEnd());

shiftBegin=(shiftBegin>0)?shiftBegin:0;

int shiftEnd=-(SceneSelector.getCrossSignalKeeper2().getBegin()-SceneSelector.getCrossSignalKeeper1().getEnd());

shiftEnd=(shiftEnd>0)?shiftEnd:0;

SceneSelector.getCrossSignalKeeper2().setShiftBegin(SceneSelector.getCrossSignalKeeper2().getBegin()+shiftBegin);

SceneSelector.getCrossSignalKeeper2().setShiftEnd(SceneSelector.getCrossSignalKeeper2().getEnd()+shiftEnd);

shiftBegin=-(SceneSelector.getCrossSignalKeeper1().getBegin()-SceneSelector.getCrossSignalKeeper2().getEnd());

shiftBegin=(shiftBegin>0)?shiftBegin:0;

shiftEnd=-(SceneSelector.getCrossSignalKeeper1().getBegin()-SceneSelector.getCrossSignalKeeper2().getEnd());

shiftEnd=(shiftEnd>0)?shiftEnd:0;

SceneSelector.getCrossSignalKeeper1().setShiftBegin(SceneSelector.getCrossSignalKeeper1().getBegin()+shiftBegin);

SceneSelector.getCrossSignalKeeper1().setShiftEnd(SceneSelector.getCrossSignalKeeper1().getEnd()+shiftEnd);

}

ComboBoxCreator.createComboBox(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossComboBox, SceneSelector.getCrossSignalKeeper1().getStep(),SceneSelector.getCrossSignalKeeper2().getStep(),difference1,difference2);

ChartBuilder.buildChart(SceneSelector.getCrossSignalKeeper1(),SceneSelector.getCrossSignalKeeper2(),false);

ChartDrawer.drawSignal(SceneSelector.getController("CHOOSE\_CROSS\_SCENE").chooseCrossChart1,SceneSelector.getCrossSignalKeeper1());

}

public void chooseCrossChart2(ActionEvent actionEvent){

File file= FileOpener.openFile();

if(file==null){

return;

}

Parser.parseFile(file,SceneSelector.getCrossSignalKeeper2());

SceneSelector.getCrossSignalKeeper2().setChanged(true);

int difference1=SceneSelector.getCrossSignalKeeper1().getDuration();

int difference2=SceneSelector.getCrossSignalKeeper2().getDuration();

if(SceneSelector.getCrossSignalKeeper1()!=null){

int shiftBegin=-(SceneSelector.getCrossSignalKeeper2().getBegin()-SceneSelector.getCrossSignalKeeper1().getEnd());

shiftBegin=(shiftBegin>0)?shiftBegin:0;

int shiftEnd=-(SceneSelector.getCrossSignalKeeper2().getBegin()-SceneSelector.getCrossSignalKeeper1().getEnd());

shiftEnd=(shiftEnd>0)?shiftEnd:0;

SceneSelector.getCrossSignalKeeper2().setShiftBegin(SceneSelector.getCrossSignalKeeper2().getBegin()+shiftBegin);

SceneSelector.getCrossSignalKeeper2().setShiftEnd(SceneSelector.getCrossSignalKeeper2().getEnd()+shiftEnd);

shiftBegin=-(SceneSelector.getCrossSignalKeeper1().getBegin()-SceneSelector.getCrossSignalKeeper2().getEnd());

shiftBegin=(shiftBegin>0)?shiftBegin:0;

shiftEnd=-(SceneSelector.getCrossSignalKeeper1().getBegin()-SceneSelector.getCrossSignalKeeper2().getEnd());

shiftEnd=(shiftEnd>0)?shiftEnd:0;

SceneSelector.getCrossSignalKeeper1().setShiftBegin(SceneSelector.getCrossSignalKeeper1().getBegin()+shiftBegin);

SceneSelector.getCrossSignalKeeper1().setShiftEnd(SceneSelector.getCrossSignalKeeper1().getEnd()+shiftEnd);

}

ComboBoxCreator.createComboBox(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossComboBox, SceneSelector.getCrossSignalKeeper1().getStep(),SceneSelector.getCrossSignalKeeper2().getStep(),difference1, difference2);

ChartBuilder.buildChart(SceneSelector.getCrossSignalKeeper1(),SceneSelector.getCrossSignalKeeper2(),false);

ChartDrawer.drawSignal(SceneSelector.getController("CHOOSE\_CROSS\_SCENE").chooseCrossChart2,SceneSelector.getCrossSignalKeeper2());

}

public void autoChangeStrategy(ActionEvent actionEvent) {

int step=(int)SceneSelector.getController("AUTO\_KORREL\_SCENE").autoComboBox.getSelectionModel().getSelectedItem();

SceneSelector.getAutoStrategyKeeper().setStepStrategy(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoCheckBox.isSelected());

if(SceneSelector.getAutoStrategyKeeper().isStepStrategy()){

SceneSelector.getController("AUTO\_KORREL\_SCENE").autoChart3AxisY.setAutoRanging(false);

SceneSelector.getAutoStrategyKeeper().copySignal(SceneSelector.getAutoSignalKeeper(),SceneSelector.getAutoSignalKeeper());

SceneSelector.getAutoStrategyKeeper().setStepStrategyBegin(-SceneSelector.getAutoSignalKeeper().getDuration());

SceneSelector.getAutoStrategyKeeper().setDefaultBegin(SceneSelector.getAutoSignalKeeper().getBegin());

SceneSelector.getAutoStrategyKeeper().setDefaultEnd(SceneSelector.getAutoSignalKeeper().getEnd());

SceneSelector.getAutoSignalKeeper().setEnd(SceneSelector.getAutoSignalKeeper().getBegin());

SceneSelector.getAutoSignalKeeper().setBegin(SceneSelector.getAutoSignalKeeper().getBegin()-SceneSelector.getAutoSignalKeeper().getDuration());

SceneSelector.getController("AUTO\_KORREL\_SCENE").autoComboBox.setDisable(true);

ChartBuilder.moveChart(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoChart2AxisX, SceneSelector.getAutoSignalKeeper(), SceneSelector.getAutoStrategyKeeper());

SceneSelector.getAutoSignalKeeper().setChanged(true);

ChartDrawer.drawSignal(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoKorrelChart2,SceneSelector.getAutoSignalKeeper());

}

else{

SceneSelector.getController("AUTO\_KORREL\_SCENE").autoChart3AxisY.setAutoRanging(true);

SceneSelector.getAutoSignalKeeper().setBegin(SceneSelector.getAutoStrategyKeeper().getDefaultBegin());

SceneSelector.getAutoSignalKeeper().setEnd(SceneSelector.getAutoStrategyKeeper().getDefaultEnd());

ChartBuilder.returnChart(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoChart2AxisX,SceneSelector.getAutoSignalKeeper());

SceneSelector.getController("AUTO\_KORREL\_SCENE").autoComboBox.setDisable(false);

SceneSelector.getAutoSignalKeeper().setChanged(false);

ChartDrawer.drawShiftSignal(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoKorrelChart2,SceneSelector.getAutoSignalKeeper());

}

ChartDrawer.drawKorrelSignal(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoKorrelChart3,SceneSelector.getAutoSignalKeeper(),SceneSelector.getAutoSignalKeeper(),step, SceneSelector.getAutoStrategyKeeper());

TooltipSetter.addTooltips(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoKorrelChart3);

}

public void crossChangeStrategy(ActionEvent actionEvent) {

int step=(int)SceneSelector.getController("CROSS\_KORREL\_SCENE").crossComboBox.getSelectionModel().getSelectedItem();

SceneSelector.getCrossStrategyKeeper().setStepStrategy(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossCheckBox.isSelected());

if(SceneSelector.getCrossStrategyKeeper().isStepStrategy()){

SceneSelector.getController("CROSS\_KORREL\_SCENE").crossChart3AxisY.setAutoRanging(false);

SceneSelector.getCrossStrategyKeeper().copySignal(SceneSelector.getCrossSignalKeeper1(), SceneSelector.getCrossSignalKeeper2());

SceneSelector.getCrossStrategyKeeper().setStepStrategyBegin(-(SceneSelector.getCrossSignalKeeper2().getEnd()-SceneSelector.getCrossSignalKeeper1().getBegin()));

SceneSelector.getCrossStrategyKeeper().setDefaultBegin(SceneSelector.getCrossSignalKeeper2().getBegin());

SceneSelector.getCrossStrategyKeeper().setDefaultEnd(SceneSelector.getCrossSignalKeeper2().getEnd());

SceneSelector.getCrossSignalKeeper2().setEnd(SceneSelector.getCrossSignalKeeper1().getBegin());

SceneSelector.getCrossSignalKeeper2().setBegin(SceneSelector.getCrossSignalKeeper1().getBegin()-SceneSelector.getCrossSignalKeeper2().getDuration());

SceneSelector.getController("CROSS\_KORREL\_SCENE").crossComboBox.setDisable(true);

SceneSelector.getCrossSignalKeeper1().setChanged(true);

SceneSelector.getCrossSignalKeeper2().setChanged(true);

ChartBuilder.moveChart(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossChart2AxisX, SceneSelector.getCrossSignalKeeper2(), SceneSelector.getCrossStrategyKeeper());

ChartDrawer.drawSignal(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossKorrelChart2,SceneSelector.getCrossSignalKeeper2());

}

else{

SceneSelector.getController("CROSS\_KORREL\_SCENE").crossChart3AxisY.setAutoRanging(true);

SceneSelector.getCrossSignalKeeper2().setBegin(SceneSelector.getCrossStrategyKeeper().getDefaultBegin());

SceneSelector.getCrossSignalKeeper2().setEnd(SceneSelector.getCrossStrategyKeeper().getDefaultEnd());

ChartBuilder.returnChart(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossChart2AxisX,SceneSelector.getCrossSignalKeeper2());

SceneSelector.getController("CROSS\_KORREL\_SCENE").crossComboBox.setDisable(false);

SceneSelector.getCrossSignalKeeper1().setChanged(false);

SceneSelector.getCrossSignalKeeper2().setChanged(false);

ChartDrawer.drawShiftSignal(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossKorrelChart2,SceneSelector.getCrossSignalKeeper2());

}

ChartDrawer.drawKorrelSignal(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossKorrelChart3,SceneSelector.getCrossSignalKeeper1(),SceneSelector.getCrossSignalKeeper2(),step, SceneSelector.getCrossStrategyKeeper());

TooltipSetter.addTooltips(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossKorrelChart3);

}

public void moveAutoSignal(KeyEvent keyEvent) {

int step=(int)SceneSelector.getController("AUTO\_KORREL\_SCENE").autoComboBox.getSelectionModel().getSelectedItem();

if(!SceneSelector.getAutoStrategyKeeper().isStepStrategy()){

return;

}

ChartBuilder.moveChart(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoChart2AxisX,SceneSelector.getAutoSignalKeeper());

if(keyEvent.getCode().getName().equalsIgnoreCase("RIGHT")){

SceneSelector.getAutoSignalKeeper().setBegin(SceneSelector.getAutoSignalKeeper().getBegin()+step);

SceneSelector.getAutoSignalKeeper().setEnd(SceneSelector.getAutoSignalKeeper().getEnd()+step);

ChartDrawer.incrementKorrelSignal(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoKorrelChart3,SceneSelector.getAutoSignalKeeper(),SceneSelector.getAutoSignalKeeper(), SceneSelector.getAutoStrategyKeeper(), step, true, true);

TooltipSetter.addTooltip(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoKorrelChart3, step);

}

if(keyEvent.getCode().getName().equalsIgnoreCase("LEFT")){

SceneSelector.getAutoSignalKeeper().setBegin(SceneSelector.getAutoSignalKeeper().getBegin()-step);

SceneSelector.getAutoSignalKeeper().setEnd(SceneSelector.getAutoSignalKeeper().getEnd()-step);

ChartDrawer.incrementKorrelSignal(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoKorrelChart3,SceneSelector.getAutoSignalKeeper(),SceneSelector.getAutoSignalKeeper(), SceneSelector.getAutoStrategyKeeper(), step, false, true);

}

ChartDrawer.drawSignal(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoKorrelChart2,SceneSelector.getAutoSignalKeeper());

}

public void moveCrossSignal(KeyEvent keyEvent) {

int step=(int)SceneSelector.getController("CROSS\_KORREL\_SCENE").crossComboBox.getSelectionModel().getSelectedItem();

if(!SceneSelector.getCrossStrategyKeeper().isStepStrategy()){

return;

}

ChartBuilder.moveChart(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossChart2AxisX,SceneSelector.getCrossSignalKeeper2());

if(keyEvent.getCode().getName().equalsIgnoreCase("RIGHT")){

SceneSelector.getCrossSignalKeeper2().setBegin(SceneSelector.getCrossSignalKeeper2().getBegin()+step);

SceneSelector.getCrossSignalKeeper2().setEnd(SceneSelector.getCrossSignalKeeper2().getEnd()+step);

ChartDrawer.incrementKorrelSignal(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossKorrelChart3,SceneSelector.getCrossSignalKeeper1(),SceneSelector.getCrossSignalKeeper2(), SceneSelector.getCrossStrategyKeeper(), step, true, false);

TooltipSetter.addTooltip(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossKorrelChart3, step);

}

if(keyEvent.getCode().getName().equalsIgnoreCase("LEFT")){

SceneSelector.getCrossSignalKeeper2().setBegin(SceneSelector.getCrossSignalKeeper2().getBegin()-step);

SceneSelector.getCrossSignalKeeper2().setEnd(SceneSelector.getCrossSignalKeeper2().getEnd()-step);

ChartDrawer.incrementKorrelSignal(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossKorrelChart3,SceneSelector.getCrossSignalKeeper1(),SceneSelector.getCrossSignalKeeper2(), SceneSelector.getCrossStrategyKeeper(), step, false, false);

}

ChartDrawer.drawSignal(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossKorrelChart2,SceneSelector.getCrossSignalKeeper2());

}

public void deleteAutoKorrelDiscrets(ActionEvent actionEvent) {

boolean createSymbols=true;

if(SceneSelector.getController("AUTO\_KORREL\_SCENE").autoKorrelDeleteDiscretsCheckBox.isSelected()){

createSymbols=false;

}

SceneSelector.getController("AUTO\_KORREL\_SCENE").autoKorrelChart1.setCreateSymbols(createSymbols);

SceneSelector.getController("AUTO\_KORREL\_SCENE").autoKorrelChart2.setCreateSymbols(createSymbols);

SceneSelector.getController("AUTO\_KORREL\_SCENE").autoKorrelChart3.setCreateSymbols(createSymbols);

}

public void deleteCrossKorrelDiscrets(ActionEvent actionEvent) {

boolean createSymbols=true;

if(SceneSelector.getController("CROSS\_KORREL\_SCENE").crossKorrelDeleteDiscretsCheckBox.isSelected()){

createSymbols=false;

}

SceneSelector.getController("CROSS\_KORREL\_SCENE").crossKorrelChart1.setCreateSymbols(createSymbols);

SceneSelector.getController("CROSS\_KORREL\_SCENE").crossKorrelChart2.setCreateSymbols(createSymbols);

SceneSelector.getController("CROSS\_KORREL\_SCENE").crossKorrelChart3.setCreateSymbols(createSymbols);

}

public void createSignal(MouseEvent actionEvent) {

SceneSelector.chooseScene("CREATE\_SCENE");

}

public void changeSignals(ActionEvent actionEvent) {

SignalKeeper signalKeeper1=SceneSelector.getCrossSignalKeeper1();

SignalKeeper signalKeeper2=SceneSelector.getCrossSignalKeeper2();

SceneSelector.setCrossSignalKeeper1(signalKeeper2);

SceneSelector.setCrossSignalKeeper2(signalKeeper1);

ChartBuilder.buildChart(SceneSelector.getCrossSignalKeeper1(),SceneSelector.getCrossSignalKeeper2(),false);

SceneSelector.getCrossSignalKeeper1().setChanged(true);

SceneSelector.getCrossSignalKeeper2().setChanged(true);

ChartDrawer.drawSignal(SceneSelector.getController("CHOOSE\_CROSS\_SCENE").chooseCrossChart1,SceneSelector.getCrossSignalKeeper1());

ChartDrawer.drawSignal(SceneSelector.getController("CHOOSE\_CROSS\_SCENE").chooseCrossChart2,SceneSelector.getCrossSignalKeeper2());

}

public void moveFromMainToHelpScene(MouseEvent mouseEvent) {

SceneSelector.chooseScene("HELP\_SCENE");

}

}

**Модуль выбора сигналов и отображения корреляционных характеристик**

public class Parser {

public static void parseFile(File file, SignalKeeper signalKeeper){

String path=file.getAbsolutePath();

Properties properties=new Properties();

try(FileInputStream fileInputStream=new FileInputStream(path)){

properties.load(fileInputStream);

signalKeeper.setBegin(Integer.valueOf(properties.getProperty("BEGIN")));

signalKeeper.setEnd(Integer.valueOf(properties.getProperty("END")));

signalKeeper.setStep(Integer.valueOf(properties.getProperty("STEP")));

signalKeeper.setDuration(signalKeeper.getEnd()-signalKeeper.getBegin());

String signalString=properties.getProperty("SIGNAL");

double signal[][]=parseSignal(signalString);

signalKeeper.setSignalY(copyArray(signal[1]));

double[] realSignal=removeDublicats(signal[1], signal[0]);

signalKeeper.setSignal(realSignal);

signalKeeper.setOriginSignal(realSignal);

signalKeeper.setSignalX(signal[0]);

}

catch (IOException e) {

e.printStackTrace();

}

}

private static double[][] parseSignal(String signalString){

String[] signalArrayString=signalString.split(",");

double signalY[]=new double[signalArrayString.length];

double signalX[]=new double[signalArrayString.length];

for(int i=0;i<signalArrayString.length;i++){

String currentString=signalArrayString[i];

currentString=currentString.substring(1,currentString.length()-1);

String[] stringValues=currentString.split(";");

signalY[i]=Double.valueOf(stringValues[1]);

signalX[i]=Integer.valueOf(stringValues[0]);

}

double[][] totalSignal={signalX,signalY};

return totalSignal;

}

private static double[] removeDublicats(double[] dublicateSignal, double[] signalX){

int dublicateCounts=0;

for(int i=0;i<signalX.length;i++){

if(i==0){

continue;

}

if(signalX[i]==signalX[i-1]){

if(dublicateSignal[i-dublicateCounts]>dublicateSignal[i-dublicateCounts-1] && dublicateSignal[i-dublicateCounts-1]!=0){

dublicateSignal[i-dublicateCounts-1]=dublicateSignal[i-dublicateCounts];

}

shiftArray(dublicateSignal, i-dublicateCounts);

dublicateCounts++;

}

}

double[] realSignal=new double[dublicateSignal.length-dublicateCounts];

for(int i=0;i<realSignal.length;i++){

realSignal[i]=dublicateSignal[i];

}

return realSignal;

}

private static void shiftArray(double[] dublicateSignal, int index){

for(int i=index;i<dublicateSignal.length;i++){

if(i==dublicateSignal.length-1){

continue;

}

dublicateSignal[i]=dublicateSignal[i+1];

}

}

private static double[] copyArray(double[] signal){

double[] newSignal=new double[signal.length];

for(int i=0;i<signal.length;i++){

newSignal[i]=signal[i];

}

return newSignal;

}

}

public class FileOpener {

private static final FileChooser fileChooser = new FileChooser();

private static final String absolutePath=new File("").getAbsolutePath();

private static final File propertyFile=new File(absolutePath);

static{

fileChooser.setTitle("Выбор файла");

fileChooser.setInitialDirectory(propertyFile);

FileChooser.ExtensionFilter extFilter = new FileChooser.ExtensionFilter("Properties files (\*.properties)", "\*.properties");

fileChooser.getExtensionFilters().add(extFilter);

}

public static File openFile(){

File file = fileChooser.showOpenDialog(SceneSelector.getPrimaryStage());

return file;

}

public static void saveFile(List<Point> points,int begin,int end,int step) throws IOException{

File file = fileChooser.showSaveDialog(SceneSelector.getPrimaryStage());

if(file!=null){

FileWriter writer = null;

writer = new FileWriter(file);

writer.write("BEGIN="+begin+"\n");

writer.write("END="+end+"\n");

writer.write("STEP="+step+"\n");

writer.write("SIGNAL=");

for(Point point : points) {

String X = point.getX();

String Y = point.getY();

writer.write("("+X+";"+Y+"),");

}

writer.close();

}

}

}

public class ChartDrawer {

public static void drawSignal(LineChart lineChart, SignalKeeper signalKeeper){

if(signalKeeper.getSignal()==null){

return;

}

lineChart.getData().clear();

XYChart.Series series = new XYChart.Series();

int currentDiscret=0;

int begin=signalKeeper.getBegin();

int end=signalKeeper.getEnd();

double[] signalX=signalKeeper.getSignalX();

double[] signalY=signalKeeper.getSignalY();

for(int i=begin;i<end+1;i+=signalKeeper.getStep()) {

if(!(signalX.length-1==currentDiscret)&&currentDiscret!=0&&signalX[currentDiscret]==signalX[currentDiscret-1]){

i--;

}

series.getData().add(new XYChart.Data(i, signalY[currentDiscret]));

currentDiscret++;

if(i==end&&!(signalX.length-1<currentDiscret)&&signalX[currentDiscret]==signalX[currentDiscret-1]){

i--;

}

}

lineChart.getData().add(series);

}

public static void drawShiftSignal(LineChart lineChart, SignalKeeper signalKeeper){

if(signalKeeper.getSignal()==null){

return;

}

lineChart.getData().clear();

XYChart.Series series = new XYChart.Series();

int currentDiscret=0;

int begin=signalKeeper.getShiftBegin();

int end=signalKeeper.getShiftEnd();

double[] signalX=signalKeeper.getSignalX();

double[] signalY=signalKeeper.getSignalY();

for(int i=begin;i<end+1;i+=signalKeeper.getStep()) {

if(!(signalX.length-1==currentDiscret)&&currentDiscret!=0&&signalX[currentDiscret]==signalX[currentDiscret-1]){

i--;

}

series.getData().add(new XYChart.Data(i, signalY[currentDiscret]));

currentDiscret++;

if(i==end&&!(signalX.length-1<currentDiscret)&&signalX[currentDiscret]==signalX[currentDiscret-1]){

i--;

}

}

lineChart.getData().add(series);

}

public static void drawSignal(LineChart lineChart, int begin, int end, double[] signalY, double[] signalX, int step){

if(signalY==null){

return;

}

lineChart.getData().clear();

XYChart.Series series = new XYChart.Series();

int currentDiscret=0;

for(int i=begin;i<end+1;i+=step) {

if(!(signalX.length-1==currentDiscret)&&currentDiscret!=0&&signalX[currentDiscret]==signalX[currentDiscret-1]){

i--;

}

series.getData().add(new XYChart.Data(i, signalY[currentDiscret]));

currentDiscret++;

if(i==end&&!(signalX.length-1<currentDiscret)&&signalX[currentDiscret]==signalX[currentDiscret-1]){

i--;

}

}

lineChart.getData().add(series);

}

public static void drawKorrelSignal(LineChart lineChart, SignalKeeper signalKeeper1, SignalKeeper signalKeeper2, int step, StrategyKeeper strategyKeeper){

if(signalKeeper1.getSignal()==null || signalKeeper2.getSignal()==null){

return;

}

lineChart.getData().clear();

int leftShift=(signalKeeper2.getEnd()-signalKeeper1.getBegin())/signalKeeper2.getStep();

int begin=-(signalKeeper2.getEnd()-signalKeeper1.getBegin());

int end=signalKeeper1.getEnd()-signalKeeper2.getBegin();

double signal1[]=signalKeeper1.getSignal();

double signal2[]=new double[signal1.length+(signalKeeper2.getSignal().length-1)\*2];

for(int i=0;i<signalKeeper2.getSignal().length;i++){

signal2[i]=signalKeeper2.getSignal()[i];

}

if(strategyKeeper.isStepStrategy()){

begin=end=strategyKeeper.getStepStrategyBegin();

}

XYChart.Series series = new XYChart.Series();

double amplitude=0;

int shift=-leftShift;

for(int i=begin;i<end+1;i+=signalKeeper2.getStep()){

if(shift==-leftShift){

amplitude=0;

}

else {

amplitude = sumSignal(signal1, signal2, shift, signalKeeper2.getSignal().length, (signalKeeper2.getBegin()-signalKeeper1.getBegin())/signalKeeper2.getStep());

}

shift++;

series.getData().add(new XYChart.Data(i,amplitude\*signalKeeper2.getStep()));

}

lineChart.getData().add(series);

}

public static void incrementKorrelSignal(LineChart lineChart, SignalKeeper signalKeeper1, SignalKeeper signalKeeper2, StrategyKeeper strategyKeeper, int step, boolean increment, boolean isAuto){

if(signalKeeper1.getSignal()==null || signalKeeper2.getSignal()==null){

return;

}

int signalKeeper1Begin=(isAuto)?SceneSelector.getAutoStrategyKeeper().getDefaultBegin():signalKeeper1.getBegin();

int signalKeeper1End=(isAuto)?SceneSelector.getAutoStrategyKeeper().getDefaultEnd():signalKeeper1.getEnd();

if(increment==true && !(signalKeeper2.getEnd()<signalKeeper1Begin+signalKeeper1.getStep()) && !(signalKeeper1End<signalKeeper2.getBegin())) {

XYChart.Series series = (XYChart.Series) lineChart.getData().get(0);

for(int i=0;i<step;i++) {

XYChart.Data data = (XYChart.Data) series.getData().get(series.getData().size() - 1);

int lastValue = (Integer) data.getXValue();

int beginDifference = (isAuto) ? 0 : (strategyKeeper.getDefaultBegin() - signalKeeper1.getBegin()) / signalKeeper1.getStep();

double amplitude = sumSignal(signalKeeper1.getSignal(), strategyKeeper.getSignal(), (lastValue + signalKeeper1.getStep()) / signalKeeper1.getStep(), signalKeeper2.getSignal().length, beginDifference);

series.getData().add(new XYChart.Data(lastValue + signalKeeper1.getStep(), amplitude \* signalKeeper1.getStep()));

}

}

else if(increment==false && !(signalKeeper1End-signalKeeper1.getStep()<signalKeeper2.getBegin())){

XYChart.Series series = (XYChart.Series) lineChart.getData().get(0);

if(series.getData().size()!=1) {

for(int i=0;i<step;i++) {

moveSignalBack(strategyKeeper.getSignal());

}

series.getData().remove(series.getData().size()-step,series.getData().size());

}

}

}

private static double sumSignal(double[] signal1, double[] signal2, int shift, int originLength, int beginDifference){

for(int i=signal2.length-2;i>-1;i--){

signal2[i+1]=signal2[i];

}

signal2[0]=0;

double result=0;

shift += beginDifference;

if (shift < 0) {

shift = 0;

}

if(shift>signal1.length-1){

shift=signal1.length-1;

}

for (int i = 1+shift; i < signal1.length; i++) {

result += signal1[i] \* signal2[i + originLength-1];

}

return result;

}

private static void moveSignalBack(double[] signal){

for(int i=0;i<signal.length-1;i++){

signal[i]=signal[i+1];

}

signal[signal.length-1]=0;

}

}

public class ComboBoxCreator {

private static final int[] options={1,5,10};

public static void createComboBox(ComboBox comboBox, int step, int difference){

comboBox.getItems().clear();

int length=(difference%10==0)?options.length:options.length-1;

length=(difference%5==0)?length:length-1;

comboBox.getItems().add(step);

int visibleRowCount=1;

for(int i=0;i<length;i++){

if(options[i]>step){

comboBox.getItems().add(options[i]);

visibleRowCount++;

}

}

comboBox.setVisibleRowCount(visibleRowCount);

comboBox.getSelectionModel().select(0);

}

public static void createComboBox(ComboBox comboBox, int step1, int step2, int difference1,int difference2){

comboBox.getItems().clear();

int length=(difference1%10==0 && difference2%10==0)?options.length:options.length-1;

length=(difference1%5==0 && difference2%5==0)?length:length-1;

int step=step1>step2?step1:step2;

comboBox.getItems().add(step);

int visibleRowCount=1;

for(int i=0;i<length;i++){

if(options[i]>step){

comboBox.getItems().add(options[i]);

visibleRowCount++;

}

}

comboBox.setVisibleRowCount(visibleRowCount);

comboBox.getSelectionModel().select(0);

}

}

public class TooltipSetter {

public static void addTooltips(LineChart lineChart){

XYChart.Series series = (XYChart.Series) lineChart.getData().get(0);

List<XYChart.Data> dataList=series.getData();

for(XYChart.Data data:dataList){

Node node = data.getNode();

Tooltip tooltip = new Tooltip('(' + data.getXValue().toString() + " ; " + String.format("%.2f",data.getYValue()) + ')');

Tooltip.install(node, tooltip);

}

}

public static void addTooltip(LineChart lineChart, int step){

XYChart.Series series = (XYChart.Series) lineChart.getData().get(0);

if(series.getData().size()<step){

return;

}

List<XYChart.Data> dataList=series.getData().subList(series.getData().size()-step,series.getData().size());

for(XYChart.Data data:dataList) {

Node node = data.getNode();

Tooltip tooltip = new Tooltip('(' + data.getXValue().toString() + " ; " + String.format("%.2f",data.getYValue()) + ')');

Tooltip.install(node, tooltip);

}

}

}

**Справочный модуль**

public class HelpController implements Initializable{

private static final String absolutePath=new File("").getAbsolutePath();

private File[] files;

@FXML

Pagination pagination;

@Override

public void initialize(URL location, ResourceBundle resources) {

File selectedDirectory = new File(absolutePath+"/help/");

if (selectedDirectory != null) {

FilenameFilter filterPng = new FilenameFilter() {

@Override

public boolean accept(File dir, String name) {

return name.toLowerCase().endsWith(".png");

}

};

files = selectedDirectory.listFiles(filterPng);

pagination.setPageFactory(new Callback<Integer, Node>() {

@Override

public Node call(Integer pageIndex) {

return createPage(pageIndex);

}

});

}

}

private VBox createPage(int index){

ImageView imageView = new ImageView();

File file = files[index];

try {

BufferedImage bufferedImage = ImageIO.read(file);

Image image = SwingFXUtils.toFXImage(bufferedImage, null);

imageView.setImage(image);

imageView.setFitWidth(SceneSelector.getScenes().get("HELP\_SCENE").getWidth());

imageView.setFitHeight(SceneSelector.getScenes().get("HELP\_SCENE").getHeight());

imageView.setPreserveRatio(true);

imageView.setSmooth(true);

imageView.setCache(true);

}

catch (IOException ex) {

ex.printStackTrace();

}

VBox pageBox = new VBox();

pageBox.getChildren().add(imageView);

return pageBox;

}

public void moveFromHelpToMainScene(ActionEvent actionEvent) {

SceneSelector.chooseScene("MAIN\_SCENE");

}