

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»
(МАИ)

ФАКУЛЬТЕТ №
КАФЕДРА

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ДИПЛОМНОМУ ПРОЕКТУ**

На тему: _____

Дипломант: _____
(фамилия, имя, отчество) (подпись)

Руководитель проекта: _____
(фамилия, имя, отчество) (подпись)

Консультанты:

по спец. части: _____
(фамилия, имя, отчество) (подпись)

по технологической части: _____
(фамилия, имя, отчество) (подпись)

по экономической части: _____
(фамилия, имя, отчество) (подпись)

по охране труда: _____
(фамилия, имя, отчество) (подпись)

Рецензент: _____
(фамилия, имя, отчество) (подпись)

Москва 200__год

«УТВЕРЖДАЮ»
Зав. кафедрой

Московский авиационный институт
(национальный исследовательский университет)»
(МАИ)

«__» _____ 20__ года.

Факультет _____ 3 _____
Кафедра _____ 308 _____

ЗАДАНИЕ
по подготовке дипломного проекта (работы)

Студенту _____ Панченко Владимиру Владимировичу _____

1. Тема проекта (работы) Разработка системы мониторинга состояния ЛА (Integrated System Health Management) на основе методов интеллектуального анализа данных (Data Mining)

2. Срок сдачи студентом законченного проекта (работы) 15 декабря _____ 2013__ года

3. Исходные данные к проекту (работе) данные телеметрии или их модель, алгоритмы интеллектуального анализа данных (Data Mining), модель распределённых вычислений MapReduce, алгоритмы выявления аномалий без учителя (Orca, GritBot, IMS, one-class SVM)

4. Перечень вопросов, подлежащих разработке в дипломном проекте, или краткое содержание дипломной работы:

№№ п/п	Разрабатываемый вопрос	Срок выполнения
	Обоснование актуальности разработки системы. Методы интеллектуального анализа данных как средство повышения эффективности систем мониторинга.	10.11.2013
1	Специальная часть	
1.1	Анализ существующих алгоритмов выявления аномалий без учителя	15.11.2013
1.2	Разработка метода мониторинга состояния ЛА на основе методов интеллектуального анализа данных	20.11.2013
1.3	Выбор программных средств реализации метода	25.11.2013
1.4	Разработка программной реализации метода	10.12.2013
1.5	Анализ результатов	12.12.2013
2	Экономическая часть	12.12.2013
3	Охрана труда и окружающей среды	12.12.2013

5. Перечень графического материала (с точным указанием обязательных чертежей):

[illegible]

6. Консультанты по проекту (работе)

по спец. части

(фамилия, инициалы)

(подпись)

по экономической части

(фамилия, инициалы)

(ПОДПИСЬ)

по охране труда

(фамилия, инициалы)

(подпись)

по технологической части

(фамилия, инициалы)

(подпись)

7. Дата выдачи задания _____

Руководитель

(подпись)

Задание принял к исполнению

(дата)

Подпись студента _____

РЕФЕРАТ

Панченко В.В. РАЗРАБОТКА СИСТЕМЫ МОНИТОРИНГА СОСТОЯНИЯ ЛА (INTEGRATED SYSTEM HEALTH MANAGEMENT) НА ОСНОВЕ МЕТОДОВ ИНТЕЛЛЕКТУАЛЬНОГО АНАЛИЗА ДАННЫХ (DATA MINING), дипломная работа: 182 с., 24 рис., 20 табл., 61 ист., 12 прил.

Ключевые слова: КОНТРОЛЬ И ДИАГНОСТИКА, ОБНАРУЖЕНИЕ АНОМАЛИЙ, ИНТЕЛЛЕКТУАЛЬНЫЙ АНАЛИЗ ДАННЫХ, DATA MINING, КЛАСТЕРИЗАЦИЯ, INTEGRATED SYSTEM HEALTH MONITORING

Содержание

Введение	7
1 Специальная часть	11
1.1 Постановка задачи	11
1.2 Анализ существующих методов выявления аномалий без учителя	11
1.2.1 Orca	12
1.2.2 GritBot	14
1.2.3 GMM (Gaussian Mixture Model)	17
1.2.4 DBN (Dynamic Bayesian Network)	20
1.2.5 One-Class SVM (Support Vector Machine)	25
1.2.6 Inductive Monitoring System (IMS)	28
1.3 Разработка метода мониторинга состояния ЛА на основе методов интеллектуального анализа данных	31
1.3.1 Формальная постановка задачи мониторинга системы	31
1.3.2 Описание метода	33
1.4 Выбор программных средств реализации метода	38
1.4.1 Предъявляемые требования	38
1.4.2 Анализ языков программирования	41
1.4.3 Выбор языка программирования	49
1.5 Разработка программной реализации метода	49
1.5.1 Архитектура системы	49
1.5.2 Руководство пользователя	54
1.6 Тестирование системы	58
2 Расчет экономической эффективности системы	59
2.1 Введение	59
2.2 Определение целесообразности разработки	59
2.3 Определение трудоемкости и затрат на создание ПП	60
2.4 Определение исполнителей	62
2.5 Расчет заработной платы исполнителей	62
2.6 Социальные отчисления	63

2.7 Накладные расходы	64
2.8 Прочие расходы	64
2.9 Расчет стоимости	64
2.10 Оценка экономической эффективности	65
2.11 Календарное планирование	66
2.12 Выводы	68
3 Охрана труда и окружающей среды	70
3.1 Анализ условий труда	70
3.1.1 Обеспечение условий труда в отделе разработки программного обеспечения	70
3.1.2 Характеристика помещения	70
3.1.3 Характеристика производственного процесса	71
3.1.4 Характеристика используемого оборудования	72
3.1.5 Санитарно-гигиенические факторы	72
3.1.6 Электроопасность	75
3.1.7 Пожароопасность	75
3.1.8 Эргономические факторы	76
3.1.9 Психофизиологические факторы	78
3.2 Мероприятия по обеспечению условий труда	80
3.3 Расчетная часть	81
3.3.1 Расчет уровня шума	81
3.4 Вывод	83
Заключение	85
Список использованных источников	86
Приложение А Блок-схема ЕМ-алгоритма для GMM	92
Приложение Б Блок-схема процесса обучения IMS	93
Приложение В Блок-схема процесса мониторинга IMS	94
Приложение Г Блок-схема процесса обучения разработанного метода	95

Приложение Д Блок-схема процесса создания базы кластеров для каждого режима работы системы (для разработанного метода)	96
Приложение Е Блок-схема процесса мониторинга для разработанного метода	97
Приложение Ж Диаграмма классов (операции с данными)	98
Приложение З Диаграмма классов (обнаружение аномалий в обучающей выборке)	99
Приложение И Диаграмма классов (фильтрация аномалий)	100
Приложение К Диаграмма классов (разработанный метод)	101
Приложение Л Параметры запуска программы	102
Приложение М Исходный код разработанного программного обеспечения .	104

Введение

Одной из ключевых проблем при эксплуатации летальных аппаратов (ЛА) является контроль и своевременная диагностика неисправностей. Подобный контроль выполняется на основе информации, поступающей с датчиков, контролирующих работу устройства. Для решения подобных задач используются системы ISHM (Integrated System Health Management) позволяющие оценить текущее и/или будущее состояние здоровья системы и интегрировать эту информацию в общую картину эксплуатационных потребностей с учётом имеющихся ресурсов [1]. В ISHM состояние системы контролируется по показаниям датчиков. Прогресс в развитии микроэлектроники за последние 10–15 лет привел к тому, что датчики стали существенно дешевле, легче и меньше по размерам. Это вызвало увеличение количества используемых датчиков и рост объемов телеметрической информации. Естественно, ручная обработка больших объемов информации слишком трудоемка — нужны средства автоматизации.

Традиционно системы ISHM используют одновременно несколько методов диагностики, в частности [2]:

- проверку выхода значения параметра за установленные пределы;
- экспертную систему, содержащую набор правил, описывающих нормальное поведение системы (rule-based);
- математическую модель, описывающую требуемое поведение системы (model-based).

Общий принцип у традиционных алгоритмов примерно один и тот же. Вначале эксперты задают модель поведения системы, представляющую набор правил, характеризующих поведение системы. В процессе работы системы поступающие телеметрические данные проверяются на соответствие модели. Если поведение данных начинает отклоняться от модели, то оператору, контролирующему работу системы, поступает тревожный сигнал о возможной неисправности.

У всех традиционных алгоритмов есть общий недостаток — они требуют интенсивной работы экспертов. Эксперты задают набор правил, конструируют математическую модель, устанавливают допустимые пределы значений параметров. Возрастает количество данных — возрастает количество работы, которую необходимо проделать экспертам, прежде чем система мониторинга сможет работать.

Данную задачу возможно автоматизировать средствами интеллектуального анализа данных — Data Mining. Это собирательное название, используемое для обозначения совокупности методов обнаружения в данных ранее неизвестных, нетривиальных, практически полезных и доступных интерпретации знаний, необходимых для принятия решений в различных сферах человеческой деятельности [3]. Фактически Data Mining — это набор технологий поиска скрытых закономерностей в больших необработанных объемах данных. Data Mining является частью процесса KDD (Knowledge Discovering in Databases), включающем, помимо поиска закономерностей, этапы сбора, подготовки данных и последующего анализа полученных результатов. К настоящему времени разработано множество алгоритмов и технологий Data Mining. Характерно, что универсального алгоритма для извлечения знаний из данных не существует. Каждое конкретное практическое приложение, обладающее специфическими характеристиками, требует либо адаптации существующих методик Data Mining, либо разработки новой технологии обработки данных.

Одним из ключевых направлений применения технологий Data Mining является автоматизация поиска аномалий. Поиск аномалий — это поиск шаблонов данных, не соответствующих ожидаемому поведению [4]. Хоукинс [5] определяет аномалию как «наблюдение, отличающееся от остальных настолько, что даёт основание полагать, что оно было сгенерировано с помощью другого метода или механизма». Поиск аномалий широко применяется в задачах мониторинга состояния технических систем [6]. Если в работе системы возникает неисправность, в данных, поступающих с датчиков, возникают аномалии, сигнализирующие об отклонении поведения системы от нормального поведения. Типичными задачами, решаемыми подобными системами мониторинга, являются определение факта возникновения аномалии, локализация ее местонахождения, диагностирование возникшей неисправности и прогнозирование возникновения неисправностей.

Методы диагностики аномалий, основанные на Data Mining (data-driven методы), свободны от недостатков традиционных методов и не требуют интенсивного участия экспертов для своей работы. Data-driven методы строят модель поведения системы автоматически на основе данных о нормальном поведении системы. Для обучения таким методам обычно достаточно несколько сотен точек нормальных данных.

Data-driven методы имеют ряд преимуществ по сравнению с традиционными:

- не требуют априорно заданных знаний о работе системы;
- не требуют системного анализа, чтобы определить соотношения между параметрами;
- способны обрабатывать телеметрические данные, поступающие от работающей системы, в режиме реального времени и быстро реагировать на появление аномалии, т.к. модель поведения системы очень компактна;
- позволяют устанавливать и отслеживать взаимосвязь между большим количеством параметров;
- способны обнаруживать коллективные и контекстные аномалии [4];
- дают возможность автоматически обрабатывать архивы накопленных данных и извлекать из них полезную информацию;
- позволяют легко учитывать новые данные о нормальном поведении системы и обновлять ранее построенную модель её поведения.

Разработки систем мониторинга неисправностей на основе методов Data Mining активно ведутся в Японии [2] и США [7, 8]. В последние годы за рубежом был разработан ряд data-driven методов и алгоритмов обнаружения аномалий, например, Orca, GritBot, IMS, GMM, LVS, одноклассовый SVM и др. Как показано в [9], результаты работы разных методов могут отличаться, поэтому целесообразно их комбинировать.

Наиболее весомым доказательством эффективности ISHM-систем на основе данных методов в аэрокосмической отрасли является их успешное применение в NASA для диагностики неисправностей в ЛА типа «Шаттл» и их преемниках — серии «Ares» [7]. Пробный пуск системы на архивных данных показал, что установка такой системы на аппарате «Колумбия» серии «Шаттл» позволила бы избежать взрыва ЛА при посадке, повлёкшего гибель всего экипажа. Как известно, «Колумбия» потерпела катастрофу из-за отрыва куска изоляционной обшивки, пробившей термоизоляцию на левом крыле. Отрыв произошел во время старта корабля, однако о проблемах с термоизоляцией стало известно лишь через 17 дней, во время приземления шатла [10]. База знаний ISHM строилась на основе анализа данных предыдущих 5 полетов «Колумбии». ISHM выдала сигнал о возникновении неисправности в течении двух минут с момента ее возникновения [6, 8]. В данной системе совместно используются методы Orca и IMS [11].

Подобные системы нашли применение на Международной Космической Стан-

ции (МКС) для контроля работоспособности и определения сроков ремонта и замены гироскопов (гироскоп, англ. control moment gyroscope, сокр. CMG — вращающееся инерциальное устройство, применяемое для высокоточной ориентации и стабилизации, как правило, космических аппаратов (КА), обеспечивающее правильную ориентацию в полете и предотвращающее беспорядочное вращение [12]). С 2008 года NASA ведёт работы по применению данных методов для контроля и диагностики других подсистем МКС [11].

Japan Aerospace Exploration Agency (Японское агентство аэрокосмических исследований) с 2011 года ведёт разработку систем мониторинга состояния спутников на основе данных телеметрии. Основным используемым методом в данной системе является SVM [13].

Таким образом, является перспективным разработать ISHM-систему на основе методов интеллектуального анализа данных, представляющую функционал, аналогичный зарубежным, но являющуюся открытой и доступной для использования в отечественных разработках.

1 Специальная часть

1.1 Постановка задачи

Разработать метод мониторинга состояния ЛА по данным телеметрии на основе методов интеллектуального анализа данных. Реализовать программную систему для ПЭВМ, использующую данный метод.

Система должна удовлетворять следующим требованиям:

- строить модель системы только на основе телеметрии при различных режимах её работы, без априорных данных о предметной области, назначении системы, её составе, конструкции (обучение без учителя);
- обладать способностью классифицировать аномалии в работе системы;
- в случае, если текущее поведение системы не было представлено в обучающей выборке, давать оператору численную характеристику отклонения системы от номинальных режимов;
- обрабатывать большие массивы входных данных (несколько десятков тысяч точек) за конечное время;
- учитывать как непрерывные, так и дискретные параметры системы;
- не иметь ограничений на закон распределения входных данных;
- быть устойчивой к аномалиям в обучающей выборке;
- быть устойчивой к отсутствию значений каких-либо параметров во входных данных;
- определять состояние системы в режиме реального времени.

1.2 Анализ существующих методов выявления аномалий без учителя

На данный момент существует несколько методов, для которых доказана возможность применения их в системах контроля и диагностики ЛА. Такими методами являются Orca, GritBot, GMM (Gaussian Mixture Model), DBN (Dynamic Bayesian Network), One-Class SVM (Support Vector Machine) и IMS (Inductive Monitoring System) [9].

1.2.1 Orca

Orca — метод поиска аномалий без учителя, использующий подход «ближайшего соседа» (nearest neighbor) для поиска аномалий [14]. Данный метод был разработан Стефеном Бэйем (Institute for the Study of Learning and Expertise) и Марком Швабахером (NASA Ames Research Center) и подробно описан в [15]. Orca относится к методам обнаружения аномалий, основанных на измерении расстояний между точками (distance-based).

Понятие аномалии для данного класса методов определено следующим образом: «объект O в выборке T является аномалией, если по крайней мере доля p из всех объектов в T лежит дальше от O , чем расстояние D » [16]. Distance-based методы являются обобщением некоторых статистических тестов на аномальность. Данный класс методов не требует априорных знаний о виде распределения для выборки. Кнорр и Нг предложили простейший алгоритм на вложенных циклах (Nested Loop, NL) [16], который находит аномалии путём вычисления расстояния между всеми точками в исходной выборке. Сложность данного алгоритма составляет $O(kN^2)$, где k — размерность пространства, а N — размер выборки.

Несмотря на то, что были разработаны более эффективные с т.з. вычислительной сложности алгоритмы ([17] и [18]), на практике наиболее сложным является определение расстояния D , по достижению которого точку следует считать аномалией. Может потребоваться непредсказуемо большое число итераций, чтобы найти подходящее значение D . Найти интервал $[D_{min}, D_{max}]$ возможно путём полного перебора, как показано в [17], но данный подход обладает слишком высокой вычислительной сложностью.

В качестве решения данной проблемы было предложено следующее определение аномалии, не требующее задания D : «объект считается аномалией, если это один из n объектов с наибольшим расстоянием до их k -ых ближайших соседей, где $k, n \in \mathbb{N}$ » [19]. Пользователю достаточно указать количество аномалий, которое должен вернуть алгоритм, без прямого указания дистанции D . Более того, возвращаемые алгоритмом аномалии будут ранжированы по степени аномальности, являющейся численной характеристикой.

Orca использует данный подход, развивая идею алгоритма на вложенных циклах (NL). Данный алгоритм на больших массивах данных показывает сложность, близ-

кую к линейной [15].

Псевдокод алгоритма приведён в листинге 1.

Листинг 1 — Псевдокод алгоритма Orca

Входные данные: k , количество ближайших соседей; n , количество аномалий; D , выборка.

Выходные данные: O , множество аномалий

- 1: Перемешать все объекты в выборке D .
 - 2: Инициализировать величину среза нулём.
 - 3: **до тех пор, пока** в выборке D остались необработанные объекты **выполнять**
 - 4: Загрузить фиксированное количество объектов B в буфер.
 - 5: **для** каждого объекта d в D **выполнять**
 - 6: **для** каждого объекта b в B **выполнять**
 - 7: Вычислить расстояние между b и d .
 - 8: **если** d ближе к b , чем k ближайших соседей b **то**
 - 9: Заменить соседа с наибольшим расстоянием на d .
 - 10: Вычислить степень аномальности b .
 - 11: **если** степень аномальности ниже величины среза **то**
 - 12: Удалить b из B .
 - 13: **конец**
 - 14: **конец**
 - 15: **конец цикла**
 - 16: **конец цикла**
 - 17: Поместить в O оставшиеся в B объекты.
 - 18: Отсортировать объекты в O по степени аномальности.
 - 19: Оставить в O только n объектов.
 - 20: Обновить величину среза степенью аномальности последнего объекта в O .
 - 21: **конец цикла**
 - 22: **вернуть как результат** O .
-

Ключевыми особенностями алгоритма являются:

- необходимость рандомизации исходных данных (строка 1). Для эффективной работы алгоритма требуется, чтобы объекты в выборке находились в случайном порядке. При обработке выборки на ПЗУ возможно рандомизировать выборку за линейное время и используя конечный объём памяти [15];
- использование вложенных циклов (строка 5). Основной идеей является отслеживание ближайших соседей для каждого объекта в D ;

- правило отсечения (строка 11). Когда для ближайших соседей объекта степень аномальности становится меньше, чем величина среза, алгоритм удаляет данный объект, так как больше нет оснований считать его аномальным. Чем больше объектов перебирает алгоритм, тем выше становится величина среза, улучшая таким образом эффективность алгоритма по времени.

В качестве метрики для определения расстояния может использоваться, к примеру, Евклидово расстояние для непрерывных и расстояние Хэмминга для дискретных переменных. Функция, определяющая степень аномальности, может быть любой монотонно убывающей функцией от расстояний до ближайших соседей [15], например, среднее расстояние до k ближайших соседей или расстояние до k -го ближайшего соседа.

Преимуществами метода являются:

- превосходная масштабируемость: на выборках большого объёма производительность алгоритма близка к линейной;
- низкие требования к памяти: не требуется загружать в память всю выборку;
- возможность задать любую метрику для расстояния и функцию для определения степени аномальности.

Недостатки следуют из природы метода. В качестве основных можно выделить следующие:

- в худшем случае (например, когда выборка не содержит аномалий) производительность алгоритма крайне низкая. Из-за вложенных циклов может потребоваться $O(N^2)$ операций вычисления расстояния и $O(N/l \cdot N)$ операций доступа к данным, где l — размер буфера;
- в качестве результата алгоритм возвращает фиксированное число аномалий, указанное перед началом работы;
- данный метод не способен определять аномальность объекта в реальном времени, так как для этого требуется вычислить расстояние до всех объектов в выборке.

1.2.2 GritBot

GritBot является коммерческим продуктом компании RuleQuest Research [20]. Вместо поиска точек, наиболее сильно отличающихся от остальной выборки, данный

метод ищет подмножества, аномальность которых очевидна [14]. Метод определяет границы для непрерывных и список возможных значений для дискретных переменных, формируя набор правил классификации. GritBot основан на использовании деревьев решений [9] и использует алгоритм C4.5 [21], разработанный Джоном Квинланом и описанный им в [22].

Для того, чтобы с помощью C4.5 построить дерево решений и применять его, входные данные должны удовлетворять нескольким условиям.

Информация об объектах, которые необходимо классифицировать, должна быть представлена в виде конечного набора признаков (атрибутов), каждый из которых имеет дискретное или непрерывное значение. Такой набор атрибутов назовём *примером*. Для всех примеров количество атрибутов и их состав должны быть постоянными.

Множество классов, на которые будут разбиваться примеры, должно иметь конечное число элементов, а каждый пример должен однозначно относиться к конкретному классу. Для случаев с нечёткой логикой, когда примеры принадлежат к классу с некоторой вероятностью, C4.5 неприменим.

В обучающей выборке количество примеров должно быть значительно больше количества классов, к тому же каждый пример должен быть заранее ассоциирован со своим классом. По этой причине C4.5 является вариантом машинного обучения с учителем.

Данный алгоритм рекурсивно разбивает множество объектов на подмножества так, чтобы энтропия полученных подмножеств была минимальна. Лучшее разбиение при этом выбирается перебором всех возможных вариантов.

Построение дерева решений в алгоритме C4.5 происходит следующим образом. Пусть имеется T — обучающая выборка примеров, а C — множество классов, состоящее из k элементов. Для каждого примера из T известна его принадлежность к какому-либо из классов $C_1 \dots C_k$.

На первом шаге имеется корень и ассоциированное с ним множество T , которое необходимо разбить на подмножества. Для этого необходимо выбрать один из атрибутов в качестве проверки. Выбранный атрибут A имеет n значение, что даёт разбиение на n подмножеств. Далее создаются n потомков корня, каждому из которых поставлено в соответствие своё подмножество, полученное при разбиении T . Процедура выбора атрибута и разбиения по нему рекурсивно применяется ко всем n потомкам и останав-

ливается в двух случаях:

- после очередного ветвления в вершине оказываются примеры из одного класса (тогда она становится *листом* дерева, а класс, которому принадлежат её примеры, будет решением листа);
- вершина оказалась ассоциированной с пустым множеством (тогда она становится листом, а в качестве решения выбирается наиболее часто встречающийся класс у непосредственного предка этой вершины).

Пример дерева решений, построенного алгоритмом C4.5, приведён на рисунке 1.

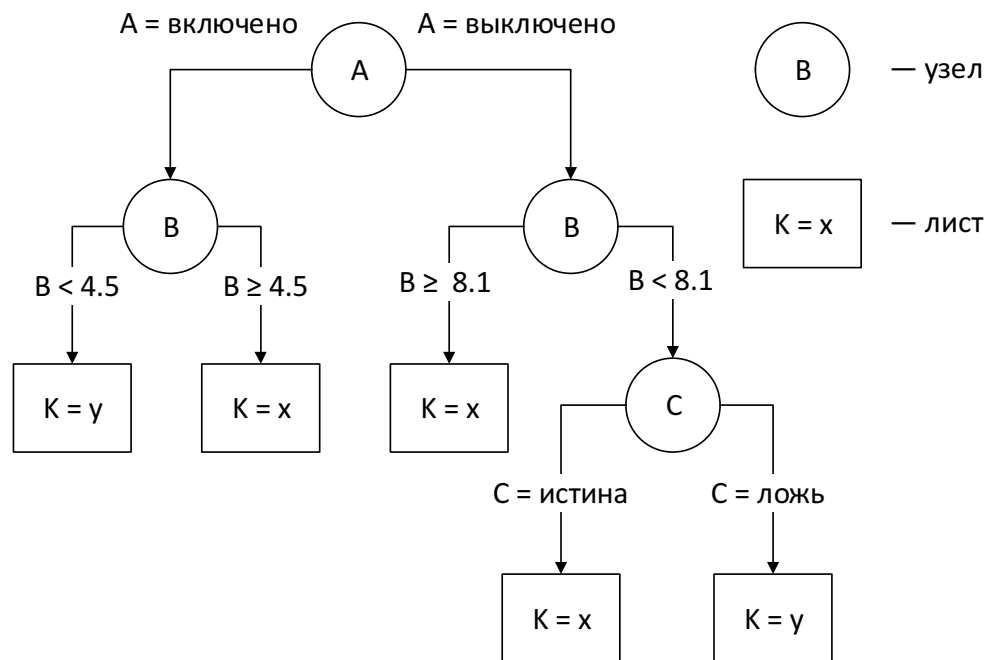


Рисунок 1 — Пример дерева решений

Так как алгоритм C4.5 относится к машинному обучению с учителем, GritBot дополняет его механизмом автоматического определения классов на основе вычисления статистических свойств выборки.

В исследовании [21] GritBot показал крайне низкую эффективность, не найдя ни одной добавленной в выборку аномалии. Это связано со статистическим подходом к определению аномальности объекта (метод ищет корреляцию между параметрами объектов в выборке).

К преимуществам можно отнести лёгкость интерпретации результатов человеком (из-за использования деревьев решений можно получить набор правил, по которым

пример был признан аномальным).

Недостатки:

- низкая эффективность при наличии в выборке объектов с большим числом аномальных параметров [21];
- данный метод загружает весь массив исходных данных в память [15]; таким образом, с его помощью невозможно обрабатывать сколь-либо большие выборки;
- нет численной оценки степени аномальности примера (метод сортирует аномалии по их статистической значимости) [9].

1.2.3 GMM (Gaussian Mixture Model)

GMM, или модель гауссовых смесей, наследует идеи байесовских сетей в том смысле, что она может быть легко представлена в рамках парадигмы графического моделирования.

Пример графической модели, представляющей гауссову смесь, показана на рисунке 2. Здесь $q_k \in \{1, \dots, M\}$, $\theta = (\pi_1, \dots, \pi_M, \mu_1, \dots, \mu_M, \Sigma_1, \dots, \Sigma_M)$. Закрашенные узлы представляют наблюдаемые непрерывные переменные, y_k для момента времени k . Незакрашенные узлы, q_k , представляют M ненаблюдаемых дискретных переменных, условная вероятность которых может быть вычислена на основе наблюдаемых данных. Параметры, содержащие θ , могут быть выражены как функция от этих условных вероятностей и от других похоже сформированных оценок для каждой из M гауссовых смесей, включая весовые коэффициенты смесей (π_i), математические ожидания (μ_i) и матрицы ковариации (Σ_i).

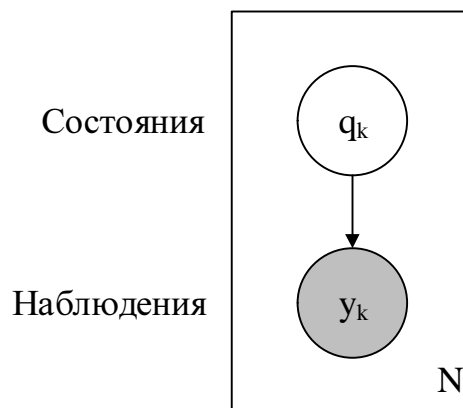


Рисунок 2 — Графическое представление GMM без учителя

Для использования данного метода требуется выполнение двух гипотез, представленных ниже.

Гипотеза о природе данных: тестовые примеры появляются случайно и независимо, согласно вероятностному распределению, равному смеси распределений кластеров. Данное условие отображено в формуле (1).

$$p(x) = \sum_{c \in C} w_c p_c(x), \sum_{c \in C} w_c = 1, \quad (1)$$

где w_c — вероятность появления объектов из кластера c ;

p_c — плотность распределения кластера c .

Гипотеза о форме кластеров: каждый кластер c описывается d -мерной гауссовской плотностью с центром $\mu_c = \{\mu_{c1}, \dots, \mu_{cd}\}$ и диагональной матрицей ковариации $\Sigma_c = \text{diag}(\sigma_{c1}^2, \dots, \sigma_{cd}^2)$ (т.е. по каждой координате своя дисперсия).

В этих предположениях для определения аномалий получается задача разделения смеси гауссовых распределений. Для этого обычно используется ЕМ-алгоритм (expectation-maximization) [9]. Подробное описание данного алгоритма представлено в [23].

ЕМ-алгоритм используется в математической статистике для нахождения оценок максимального правдоподобия параметров вероятностных моделей, в случае, когда модель зависит от некоторых скрытых переменных. Каждая итерация алгоритма состоит из двух шагов. На *E-шаге* (expectation) вычисляется ожидаемое значение функции правдоподобия, при этом скрытые переменные рассматриваются как наблюдаемые. На *M-шаге* (maximization) вычисляется оценка максимального правдоподобия, таким образом увеличивается ожидаемое правдоподобие, вычисляемое на Е-шаге. Затем это значение используется для Е-шага на следующей итерации. Алгоритм выполняется до сходимости.

Формальная постановка задачи разделения смеси гауссовых распределений выглядит следующим образом. Задана выборка X^l случайных и независимых наблюдений из смеси $p(x)$, в которой описание i -го элемента есть вектор $x_i \in \mathbb{R}^n$. Принята модель, в которой каждая компонента смеси есть гауссиана с параметрами μ и Σ , и известно число компонентов смеси — K . Смесь показана в формуле (2).

$$p(x) = \sum_{n=1}^K \pi_k N(x|\mu_k, \Sigma_k). \quad (2)$$

Требуется оценить вектор параметров $\theta = (\pi_1, \dots, \pi_M, \mu_1, \dots, \mu_M, \Sigma_1, \dots, \Sigma_M)$, доставляющий максимум функции правдоподобия (3).

$$\ln p(X|\pi, \mu, \Sigma) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k N(x_n|\mu_k, \Sigma_k) \right\} \quad (3)$$

Оптимальные параметры отыскиваются последовательно с помощью итерационного ЕМ-алгоритма. Основная идея — вводится вспомогательный вектор скрытых переменных. Это позволяет свести сложную оптимизационную задачу к последовательности итераций по пересчету коэффициентов (скрытых переменных по текущему приближению вектора параметров — Е-шаг) и максимизации правдоподобия (с целью найти следующее приближение вектора — М-шаг).

В начале работы алгоритма задаются параметры начального приближения θ_0 . Далее итеративно выполняется следующая пара процедур:

Е-шаг: используя текущее значение вектора параметров θ , вычисляется значение вектора скрытых переменных γ по формуле (4).

$$\gamma_{nk} = \frac{\pi_k N(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^K \pi_j N(x_n|\mu_j, \Sigma_j)} \quad (4)$$

М-шаг: переоценка вектора параметров по формулам (5), используя текущее значение вектора скрытых переменных.

$$\mu_k^{new} = \frac{1}{N_k} \sum_{n=1}^N \gamma_{nk} x_n, \quad (5a)$$

$$\Sigma_k^{new} = \frac{1}{N_k} \sum_{n=1}^N \gamma_{nk} (x_n - \mu_k^{new})(x_n - \mu_k^{new})^T, \quad (5b)$$

$$\pi_k^{new} = \frac{N_k}{N}, \quad (5b)$$

$$N_k = \sum_{n=1}^N \gamma_{nk}, \quad (5r)$$

Процедура останавливается после того, как норма разности векторов скрытых переменных на каждой итерации не будет превышать заданную константу Δ . Условие останова показано в (6).

$$\delta_{max} = \max \{ \delta_{max}, |\gamma_{nk} - \gamma_{nk}^0| \} \leq \Delta \quad (6)$$

Блок-схема алгоритма приведена в приложении А.

Для поиска аномалий могут быть использованы различные варианты моделей гауссовых смесей, например, модели для одного датчика системы (одномерный случай), либо для нескольких датчиков с учётом корреляции между ними (многомерный случай).

Оценка метода в применении к контролю и диагностике КА дана в [9] и [24].

Преимущества метода:

- возможность построения независимой модели для каждого датчика, что обеспечивает более точную диагностику;
- модель можно представить в графической форме.

Недостатки:

- строгие требования к исходным данным: если выборка не подчиняется нормальному распределению, то использование данного метода невозможно;
- метод не работает с выборками, имеющими коррелированные параметры;
- необходимость вручную задавать число кластеров, которое весьма трудно поддаётся определению в многомерных случаях и при больших объёмах данных [24].

1.2.4 DBN (Dynamic Bayesian Network)

Dynamic Bayesian Network, или динамическая байесовская сеть, является графической вероятностной моделью, представляющей собой множество переменных и их вероятностных зависимостей. Данный метод использует в своей работе формулу Байеса (7).

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \quad (7)$$

где $P(A)$ — априорная вероятность гипотезы A ;

$P(A|B)$ — вероятность гипотезы A при наступлении события B (апостериорная вероятность);

$P(B|A)$ — вероятность наступления события B при истинности гипотезы A ;

$P(B)$ — полная вероятность наступления события B .

Формула Байеса позволяет «переставить причину и следствие»: по известному факту события вычислить вероятность того, что оно было вызвано данной причиной. События, отражающие действие «причин», в данном случае называют *гипотезами*, так как они — предполагаемые события, повлекшие данное. Безусловную вероятность справедливости гипотезы называют *априорной* (насколько вероятна причина вообще), а условную — с учетом факта произошедшего события — *апостериорной* (насколько вероятна причина оказалась с учетом данных о событии).

Формально байесовская сеть — это направленный ациклический граф, каждой вершине которого соответствует случайная переменная, а дуги графа кодируют отношения условной независимости между этими переменными. Вершины могут представлять переменные любых типов, быть взвешенными параметрами, скрытыми переменными или гипотезами. Если переменные байесовской сети являются дискретными случайными величинами, то такая сеть называется дискретной байесовской сетью. Байесовские сети, которые моделируют последовательности переменных, называют *динамическими байесовскими сетями* [25].

Если дуга выходит из вершины A в вершину B , то A называют родителем B , а B называют потомком A . Если из вершины A существует ориентированный путь в другую вершину B , то B называется потомком A , а A называется предком B . Множество вершин-родителей вершины V_i обозначим как $parents(V_i) = PA_i$.

Направленный ациклический граф G называется байесовской сетью для вероятностного распределения $P(v)$, заданного над множеством случайных переменных V , если каждой вершине графа поставлена в соответствие случайная переменная из V , а дуги в графе удовлетворяют условию (марковское условие): любая переменная V_i из V должна быть условно независима от всех вершин, не являющихся ее потомками, если

заданы (получили означивание, обусловлены) все ее прямые родители PA_i в графе G , то есть выполняется выражение (8).

$$\forall V_i \in V : P(v_i|pa_i, s) = P(v_i|pa_i), \quad (8)$$

где v_i — значение V_i ;

S — множество всех вершин, не являющихся потомками V_i ;

s — конфигурация S ;

pa_i — конфигурация PA_i .

Тогда полное совместное распределение значений в вершинах можно удобно записать в виде декомпозиции (произведения) локальных распределений (9).

$$P(V_1, \dots, V_n) = \prod_{i=1}^n P(V_i|parents(V_i)) \quad (9)$$

Если у вершины V_i нет предков, то её локальное распределение вероятностей называют *безусловным*, иначе *условным*. Если вершина — случайная переменная получила означивание (например, в результате наблюдения), то такое означивание называют *свидетельством*. Если значение переменной было установлено извне (а не наблюдалось), то такое означивание называется *вмешательством* или *интервенцией* [25].

Условная независимость в байесовской сети представлена графическим свойством *d-разделённости*. Пусть X, Y, Z — непересекающиеся подмножества вершин в ациклическом ориентированном графе G . Говорят, что множество вершин Z *d-разделяет* X и Y тогда и только тогда, когда Z блокирует все пути из любой вершины, принадлежащей X , в любую вершину, принадлежащую Y . Под путём понимается последовательность следующих друг за другом рёбер (любого направления) в графе.

В соответствии с теоремой о *d-разделённости* для ациклично ориентированного графа G если вершины *d-разделены*, то они условно независимы; и если вершины условно-независимы во всех вероятностных распределениях, совместимых с графом G , то они *d-разделены* [25].

Для динамических байесовских сетей существуют две возможных стратегии для поиска аномалий [26]: байесовский доверительный интервал (Bayesian Credible Interval, BCI) и максимальная апостериорная оценка измерений (maximum a posteriori measurement status, MAP-ms).

1.2.4.1 Байесовский доверительный интервал (BCI)

Данная стратегия использует модель сети, представленную на рисунке 3. Вектор X представляет скрытые непрерывные переменные, вектор M — наблюдаемые непрерывные переменные. Нижние индексы обозначают моменты времени.



Рисунок 3 — Графическая модель сети для байесовского интервала правдоподобия (BCI)

Такая байесовская сеть отслеживает многомерные распределения линейных гауссовых переменных состояния и их наблюдаемые аналоги, которые измеряются с помощью датчиков. Скрытые переменные полагаются Марковскими процессами первого порядка, т.е. значение переменной в момент времени t зависит только от состояния в момент времени $t - 1$. Апостериорные вероятности скрытых и наблюдаемых переменных получаются с помощью фильтра Калмана, как только поступают новые измерения с датчиков. Данные вероятности используются для построения байесовского доверительного интервала $p\%$. Апостериорная вероятность p отражает тот факт, что наблюдаемая переменная находится внутри интервала. Таким образом, любое измерение, попадающее за пределы доверительного интервала $p\%$, может быть классифицировано как аномалия [26]. Параметры сети (распределения вероятностей $P(X_0)$, $P(X_t|X_{t-1})$, $P(M_t|X_t)$) могут быть получены из исходной выборки с помощью ЕМ-алгоритма [23].

1.2.4.2 Максимальная апостериорная оценка измерений (MAP-ms)

В стратегии MAP-ms используется более сложная модель сети, показанная на рисунке 4. Векторы X и Z представляют непрерывные и дискретные скрытые переменные, а вектор M — наблюдаемые непрерывные переменные. Нижние индексы обозначают моменты времени.



Рисунок 4 — Графическая модель сети для максимального апостериорного статуса измерений (MAP-ms)

Данная модель отслеживает многомерные многомерные распределения линейных гауссовых переменных состояния и их наблюдаемые аналоги, которые измеряются с помощью датчиков, как и вероятности скрытых дискретных переменных, показывающих статус каждого измерения (например, номинальный/аномальный). К примеру, если есть две измеряемых переменных состояния, то дискретная переменная статуса измерения будет иметь четыре возможных значения: (номинальный, номинальный), (аномальный, номинальный), (номинальный, аномальный) и (аномальный, аномальный). Апостериорные вероятности скрытых и наблюдаемых переменных получаются с помощью фильтра частиц Рао-Блэквелла, как только поступают новые измерения с датчиков. Максимальная апостериорная оценка измерения (например, наиболее вероятное значение, полученное из апостериорной вероятности) скрытой переменной состояния, показывающей статус измерения, может быть использована для классификации измерения как номинального или аномального.

MAP-ms для работы требует, во-первых, параметры байесовской сети, описывающие изменение во времени линейных гауссовых переменных состояния для каждого значения дискретного статуса измерения, и, во-вторых, параметры, описывающие изменение во времени дискретных переменных. Кроме того, необходимо вручную задать вероятности для дискретных переменных ($P(Z_0)$, $P(Z_t|Z_{t-1})$), опираясь на знание предметной области. Для случая, когда аномалий в исходной выборке нет, параметры сети совпадают с сетью для стратегии BCI, описанной в пункте 1.2.4.1. Если же од-

но или несколько измерений являются аномальными, параметры сети могут довольно сильно отличаться. [26]

Основные преимущества динамических байесовских сетей в применении к обнаружению аномалий:

- способность работать в режиме реального времени [26];
- возможность графически представить модель системы.

Недостатки:

- крайне высокая сложность метода;
- низкая эффективность на реальных данных [9].

1.2.5 One-Class SVM (Support Vector Machine)

Метод опорных векторов — набор схожих алгоритмов обучения с учителем, использующихся для задач классификации и регрессионного анализа. One-Class SVM, или одноклассовый метод опорных векторов, представляет собой специальный вариант данного семейства для поиска аномалий, позволяющий использовать его в задачах обучения без учителя. В случае, если новое измерение принадлежит классу, оно считается номинальным (система работает в номинальном режиме); если же нет, то это измерение является аномалией. Особым свойством метода опорных векторов является непрерывное уменьшение эмпирической ошибки классификации и увеличение зазора, поэтому метод также известен как *метод классификатора с максимальным зазором*.

Основная идея метода — перевод исходных векторов из низкой размерности, где они могут быть линейно неразделимы, в пространство более высокой размерности (вплоть до бесконечной) и поиск разделяющей гиперплоскости с максимальным зазором в этом пространстве. Две параллельных гиперплоскости строятся по обеим сторонам гиперплоскости, разделяющей классы. Разделяющей гиперплоскостью будет гиперплоскость, максимизирующая расстояние до двух параллельных гиперплоскостей. Алгоритм работает в предположении, что чем больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора [27].

Формальное описание задачи для случая с одним классом выглядит следующим образом: дана выборка $\Omega = \{x_1, x_2, \dots, x_n\}$, $x_i \in \mathbb{R}^m$, где x_i — точка в m -мерном пространстве, а $y_i \in \{0, 1\}$ определяет принадлежность точки классу. Строится раз-

деляющая гиперплоскость, которая имеет вид (10a), и гиперплоскость, параллельная ей (10б).

$$w^T x + b = 0; \quad (10a)$$

$$w^T x + b = 1, \quad (10б)$$

где $w \in \mathbb{F}$ — перпендикуляр к разделяющей гиперплоскости;

$b \in \mathbb{R}$ — расстояние по модулю от гиперплоскости до начала координат.

Если обучающая выборка линейно разделима, то возможно выбрать гиперплоскости таким образом, чтобы между ними не лежала ни одна точка обучающей выборки, и затем максимизировать расстояние между гиперплоскостями. Ширину полосы между ними легко найти из соображений геометрии, она равна $\frac{1}{\|w\|}$ [28], таким образом, задача сводится к минимизации $\|w\|$. Чтобы уберечь классификатор от переполнения зашумлёнными данными, вводятся ошибки ξ_i , позволяющие некоторым точкам лежать внутри зазора, и константа $C > 0$, определяющая соотношение между максимизацией зазора и количеством точек внутри него (и, соответственно, ошибкой обучения). Целевая функция с учётом этих условий имеет вид (11) [29].

$$\begin{aligned} \min_{w, b, \xi_i} \frac{\|w\|^2}{2} + C \sum_{i=1}^n \xi_i, \\ y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i \quad \text{для всех } i = 1, \dots, n, \\ \xi_i \geq 0 \quad \text{для всех } i = 1, \dots, n. \end{aligned} \quad (11)$$

Оригинальная версия метода опорных векторов относится к семейству алгоритмов линейной классификации, что не подходит для задачи поиска аномалий в реальных многомерных данных. Существует способ создания нелинейного классификатора, в основе которого лежит переход от скалярных произведений к произвольным ядрам, так называемый *kernel trick*, позволяющий строить нелинейные разделители. Результирующий алгоритм крайне похож на алгоритм линейной классификации, с той лишь разницей, что каждое скалярное произведение в приведённых выше формулах заменяется нелинейной функцией ядра (скалярным произведением в пространстве с большей размерностью). В этом пространстве уже может существовать оптимальная разделяющая гиперплоскость. Так как размерность получаемого пространства может быть больше

размерности исходного, то преобразование, сопоставляющее скалярные произведения, будет нелинейным, а значит функция, соответствующая в исходном пространстве оптимальной разделяющей гиперплоскости, будет также нелинейной [27]. Общий вид ядра показан в формуле (12).

$$K(x, x_i) = \phi(x)^T \phi(x_i) \quad (12)$$

Наиболее распространённые ядра [27]: полиномиальное однородное (13а), полиномиальное неоднородное (13б), радиальная базисная функция (13в), радиальная базисная функция Гаусса (13г).

$$K(x, x_i) = (x \cdot x_i)^d; \quad (13a)$$

$$K(x, x_i) = (x \cdot x_i + 1)^d; \quad (13б)$$

$$K(x, x_i) = \exp(-\gamma \|x - x_i\|^2), \gamma > 0; \quad (13в)$$

$$K(x, x_i) = \exp\left(-\frac{\|x - x_i\|^2}{2\sigma^2}\right). \quad (13г)$$

Преимущества метода:

- метод сводится к решению задачи квадратичного программирования, которая всегда имеет единственное решение;
- даёт численную оценку аномальности (расстояние от точки до гиперплоскости) [9].

Недостатки:

- метод неустойчив по отношению к шуму в исходных данных. Если обучающая выборка содержит шумовые выбросы, они будут существенным образом учтены при построении разделяющей гиперплоскости;
- необходимо вручную подбирать функцию ядра, исходя из знаний о предметной области и природе исходных данных;
- в общем случае, когда линейная разделимость не гарантируется, приходится подбирать управляющий параметр алгоритма C [28];
- в случае значительных изменений в режиме работы системы не способен обнаруживать аномалии [9].

1.2.6 Inductive Monitoring System (IMS)

IMS автоматически строит базу знаний для последующего мониторинга состояния системы на основе номинальных данных, собранных непосредственно во время работы системы либо её симуляций. IMS использует машинное обучение и методы интеллектуального анализа данных для описания типичного для системы поведения путём извлечения из исходных данных основных классов. В частности, IMS использует кластеризацию для группировки постоянно встречающихся последовательностей в обучающей выборке. Кластеризация — семейство алгоритмов, относящееся к обучению без учителя, упорядочивающих объекты в сравнительно однородные группы (*кластеры*). Идея метода IMS проистекает из двух алгоритмов кластеризации: алгоритма К-средних (K-means) [30] и DBSCAN [31] (кластеризация на основе плотности групп точек). Метод разработан Дэвидом Иверсоном (NASA Ames Research Center) и описан им в [8].

Базовой структурой данных для этого метода является вектор параметров системы. Пример подобного вектора приведён в таблице 1. Каждый вектор содержит упорядоченный набор параметров, полученный системой мониторинга в процессе сбора телеметрии. Он также может содержать производные от измерений параметры. Векторы параметров системы представляют собой точки в N -мерном пространстве, которые IMS группирует в кластеры. Значения вектора могут быть получены подсистемой сбора данных как одновременно, либо быть собранными в вектор из нескольких измерений в течение некоторого периода времени. Размер и конкретные параметры вектора определяются вручную в соответствии с поставленной задачей мониторинга.

Таблица 1 — Пример вектора данных IMS

Давление А	Позиция клапана 1	Давление В	Позиция клапана 2	Температура 1	Температура 2
2857.2	86.4%	1218.4	96.2%	49.8	37.6

IMS обрабатывает обучающую выборку, форматируя исходные данные в соответствии с заданной структурой вектора параметров, и строит базу знаний из кластеров. Пример подобного кластера приведён в таблице 2. Каждый кластер определяет диапазон допустимых значений всех переменных для входного вектора данных. Век-

тор верхней границы и вектор нижней границы в кластере могут быть представлены, как углы минимального ограничивающего гиперкуба в N -мерном пространстве. Точки, попадающие внутрь этого гиперкуба либо в его окрестность, классифицируются алгоритмом как относящиеся к номинальному режиму работы системы, так как гиперкуб определяется исходя из номинальных данных. Данный подход схож с интервальной диагностикой, когда строится математическая модель системы, и определяются допустимые диапазоны изменения каждой переменной, но не требует знаний о характере и устройстве системы.

Таблица 2 — Пример кластера IMS

	Давление А	Позиция клапана 1	Давление В	Позиция клапана 2	Температура 1	Температура 2
Верхняя граница	2857.2	86.4%	1218.4	96.2%	49.8	37.6
Нижняя граница	2857.2	86.4%	1218.4	96.2%	49.8	37.6

IMS начинает процесс обучения с пустой базы кластеров. Метод считывает элементы обучающей выборки и формирует из них векторы. Первый вектор добавляется в базу, как исходный кластер. Каждый последующий вектор сравнивается с содержимым базы кластеров с тем, чтобы найти ближайший к этому вектору кластер. Для определения расстояния между вектором и кластером могут быть использованы различные метрики, например, Евклидово расстояние. Для того, чтобы найти расстояние от вектора до кластера, необходимо выбрать точку в кластере, до которой оно будет измерено. Одним из вариантов, основанным на алгоритме К-средних [30], является выбор центроида кластера, который рассчитывается как среднее между нижней и верхней границами. Для каждого нового вектора из обучающей выборки IMS находит в базе ближайший к нему кластер. Далее определяется, попадает ли вектор внутрь ограничивающего гиперкуба данного кластера или в его окрестность. Как и в кластеризации на основе плотности точек [31], вводится пороговое значение ε , заданное пользователем, которое определяет максимальное допустимое расстояние между вектором и кластером. На основе этого расстояния алгоритм определяет, стоит ли добавлять вектор в кластер. Если

вектор достаточно близок (расстояние меньше или равно ε), границы кластера раздвигаются, чтобы вместить его. Если же расстояние превышает ε , создаётся новый кластер и добавляется в базу. Процесс обучения повторяется до тех пор, пока не будет обработана вся обучающая выборка. Малое значение ε приводит к созданию небольших по размеру кластеров, что обеспечивает более точный контроль, но значительно увеличивает размер базы знаний, делая невозможным мониторинг системы в реальном времени. Значение ε может быть подобрано таким образом, чтобы обеспечить баланс между скоростью работы и точностью контроля.

Блок-схема процесса обучения приведена в приложении Б.

Результатом обработки IMS обучающей выборки является модель системы в виде базы кластеров, характеризующая работу системы в номинальных режимах, отображённых в исходных данных. В дополнение к этому, каждый кластер определяет ограничения на значения каждого из параметров в векторе.

Для использования получившейся базы кластеров для мониторинга состояния системы IMS формирует векторы из приходящих данных и запрашивает базу на предмет ближайшего к каждому входному вектору кластера. Наиболее быстрый способ проверки требует, чтобы все векторы находились внутри по крайней мере одного из кластеров внутри базы (значения всех параметров должны находиться внутри диапазонов, определённых границами кластера). Такой способ устраняет необходимость вычисления расстояний. Более информативная техника мониторинга находит ближайший к входному вектору кластер и показывает пользователю расстояние между ними. Это даёт оператору представление, насколько поведение системы отклоняется от нормального, представленного в обучающей выборке. Метод может учитывать неполноту обучающей выборки и неточности измерений путём задания порогового значения ε .

Блок-схема процесса мониторинга приведена в приложении В.

Преимущества метода:

- не требует больших вычислительных ресурсов, способен работать в реальном времени;
- предоставляет оператору численную характеристику аномальности вектора измерений;
- не имеет ограничений на закон распределения входных данных;
- не требует никаких знаний о предметной области, природе системы, её внут-

реннем устройстве, характеристиках входных данных, что обеспечивает универсальность и возможность применения к широкому спектру технических систем.

Недостатки:

- метод неустойчив по отношению к шуму и аномалиям в обучающей выборке;
- вектор входных данных может содержать только непрерывные переменные;
- стандартный вариант метода рассчитывает расстояние до центра кластера, что снижает точность мониторинга.

1.3 Разработка метода мониторинга состояния ЛА на основе методов интеллектуального анализа данных

Все описанные выше методы имеют свои недостатки при их применении для диагностики сложных технических систем. Сравнение методов по критериям, указанным в разделе 1.1, приведено в таблице 3.

По результатам сравнения наиболее подходящим является метод IMS, описанный в подразделе 1.2.6. Так как он не удовлетворяет всем требованиям, указанным в разделе 1.1, необходимо создать на его основе метод, устойчивый к аномалиям в обучающей выборке, обеспечивающий возможность классификации аномалий и работающий не только с непрерывными, но и с дискретными параметрами.

1.3.1 Формальная постановка задачи мониторинга системы

Введём следующие обозначения:

M — дискретно-непрерывное пространство, имеющее p непрерывных и q дискретных измерений;

$R = \{R_1, R_2, \dots, R_n\} \in M$ — множество режимов работы системы;

$T = \{x_1, x_2, \dots, x_m\} \in M$ — обучающая выборка;

$x = \{a_1, a_2, \dots, a_p, b_1, b_2, \dots, b_q\} \in M, a_i \in \mathbb{R}, b_i \in \mathbb{N}$ — элемент обучающей выборки;

$\theta = \{\tilde{a}_1, \tilde{a}_2, \dots, \tilde{a}_p, \tilde{b}_1, \tilde{b}_2, \dots, \tilde{b}_q\} \in M, \tilde{a}_i \in \mathbb{R}, \tilde{b}_i \in \mathbb{N}$ — входной вектор;

$\Omega = \{\omega_1, \omega_2, \dots, \omega_{p+q}\}$ — вектор весов.

Необходимо отметить, что некоторые значения в векторах могут отсутствовать.

Таблица 3 — Сравнение методов выявления аномалий без учителя

Критерий \ Метод	Orca	GritBot	GMM	DBN	SVM	IMS
Построение модели системы без априорных данных о предметной области, назначении системы, её составе, конструкции	+	+	\pm	+	—	+
Классификация аномалий	—	—	—	+	—	—
Численная характеристика аномалии	+	—	—	—	+	+
Обработка больших выборок за конечное время	\pm	—	\pm	+	+	+
Работа с дискретными параметрами	+	+	—	\pm	—	—
Отсутствие ограничений на закон распределения входных данных	+	+	—	\pm	+	+
Устойчивость к аномалиям в обучающей выборке	+	+	—	—	—	—
Устойчивость к отсутствию значений параметров во входных данных	+	+	—	—	—	+
Работа в режиме реального времени	—	—	+	+	+	+

Даны n обучающих выборок $T_1 \in R_1, T_2 \in R_2, \dots, T_n \in R_n$ различной длины, содержащие m_i элементов каждая, где i — номер режима работы системы. Дан вектор Ω , содержащий веса, соответствующие важности каждого параметра системы. Дан входной вектор θ .

Необходимо определить, принадлежит ли θ к R ; если $\theta \in R$, то определить, в каком именно подмножестве $R_i \in R$ находится θ . Если же $\theta \notin R$, найти численную меру $\lambda \in \mathbb{R}_{\geq 0}$, показывающую отклонение θ от R .

1.3.2 Описание метода

Предложенный ниже метод относится к методам, основанным на измерении расстояний между точками (distance-based). Поэтому для работы метода необходимо определить взвешенную метрику $d(x, y, \Omega)$ на пространстве M . В качестве такой метрики может выступать комбинация Евклидова расстояния (14а) для непрерывных и функции отличия (14б) для дискретных переменных. Подобная метрика, определённая на пространстве M , показана в формуле (15). В случае отсутствия значения прибавляется только вес данного параметра.

$$D_E(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \quad (14a)$$

$$D_{diff}(x, y) = \begin{cases} 0, & x = y, \\ 1, & x \neq y \end{cases} \quad (14б)$$

$$d(x, y, \Omega) = \sqrt{\sum_{i=1}^p \left[\omega_i \left(a_i^{(x)} - a_i^{(y)} \right)^2 \right]} + \sum_{j=1}^q \left[\begin{cases} 0, & b_j^{(x)} = b_j^{(y)}, \\ \omega_{j+p}, & b_j^{(x)} \neq b_j^{(y)} \end{cases} \right] \quad (15)$$

Блок-схема процесса обучения метода представлена в приложении Г.

Обучение метода проходит в два этапа: подготовка данных и формирование базы режимов.

Пример единицы входных данных для метода представлен в таблице 4.

Таблица 4 — Пример единицы входных данных разрабатываемого метода

Давление А	Состояние клапана 1	Давление В	Состояние клапана 2	Температура 1	Температура 2
2857.2	Закрыт	1218.4	Открыт	49.8	37.6

1.3.2.1 Подготовка данных

Формат входных данных, показанный в таблице 4, может не соответствовать формату, описанному в подразделе 1.3.1. Для этого необходимо сгруппировать пара-

метры по своей природе (отдельно непрерывные и отдельно дискретные). Для каждого дискретного параметра составляется список возможных значений, в соответствие которым ставятся натуральные числа.

Так как элементы в векторах могут иметь различный масштаб, для корректного определения расстояния между двумя точками непрерывные значения векторов сначала необходимо нормализовать. Для этого может применяться как минимаксная нормализация (16а), так и нормализация с помощью стандартного отклонения (16б). Метод сохраняет характеристики, используемые при нормализации обучающих выборок (минимум, среднее значение и т.д.), для масштабирования входного вектора на стадии мониторинга.

$$\hat{x} = \frac{x - \min(X)}{\max(X) - \min(X)}, x \in X \quad (16a)$$

$$\hat{x} = \frac{x - \bar{x}}{\sigma_x}, \quad (16b)$$

где \bar{x} — среднее значение;

σ_x — стандартное отклонение.

Зачастую в обучающих выборках могут содержаться аномальные элементы (сбой в работе датчика, искажения в канале связи и т.д.). Для того, чтобы метод мог корректно обучаться, необходимо их исключить. Для этого на каждой выборке используется алгоритм Orca, описанный в подразделе 1.2.1. В качестве количества аномалий, которые необходимо найти, ему задаётся число элементов в выборке. С учётом этой особенности на выходе данный алгоритм выдаёт численную меру (степень) аномальности для каждого вектора. Далее применяется фильтр, который на основе этой численной меры для каждого вектора определяет, является ли вектор аномальным. Для этого могут использоваться как статистические (например, Гауссовый фильтр, считающий аномальными все векторы, численная мера для которых не укладывается в диапазон $[0, 3\sigma]$), так и пороговые фильтры (считающие аномальными векторы, численная мера которых выше определённой величины δ). Выбранные фильтром аномалии удаляются из выборки.

Результатом данного этапа являются нормализованные выборки $\hat{T}_i \in \hat{T}$, $i \in [1, n]$, не содержащие аномалий. Пример элемента такой выборки представлен в таблице 5.

Таблица 5 — Пример подготовленной единицы входных данных разрабатываемого метода

a_1	a_2	a_3	a_4	b_1	b_2
0.91	0.38	0.85	0.64	0	1

1.3.2.2 Формирование базы режимов

На данном этапе производится создание отдельной базы кластеров для каждого режима работы системы R_i , $i \in [1, n]$. Блок-схема процесса представлена в приложении Д.

Каждый кластер определяет диапазон допустимых значений для непрерывных переменных (a_1, a_2, \dots, a_p) и список допустимых значений для каждой дискретной переменной из (b_1, b_2, \dots, b_q) . Пример такого кластера представлен в таблице . Вектор верхней границы и вектор нижней границы в кластере могут быть представлены, как углы минимального ограничивающего прямоугольного параллелепипеда в p -мерном пространстве (p — число непрерывных переменных в векторе).

Таблица 6 — Пример кластера разрабатываемого метода

	a_1	a_2	a_3	a_4	b_1	b_2
Верхняя граница	0.91	0.38	0.85	0.64	—	—
Нижняя граница	0.45	0.45	0.55	0.14	—	—
Допустимые значения	—	—	—	—	0, 1	1

Метод последовательно обрабатывает все векторы из выборки $T_i \in R_i$. Процесс начинается с пустой базой кластеров. Первым вектором инициализируется начальный кластер (нижняя и верхняя граница этого кластера считаются равными вектору, в списки допустимых значений дискретных переменных добавляются значения дискретных переменных вектора). Каждый последующий вектор сравнивается со всеми кластерами в базе. Если вектор полностью находится в границах кластера, то метод переходит к следующему вектору в обучающей выборке. Если вектор не попал ни в один кластер, то измеряется расстояние $d_c(x, c, \Omega)$ между вектором и всеми кластерами в базе с

целью нахождения наиболее близкого кластера. Вводится пороговое значение ε , заданное пользователем, которое определяет максимальное допустимое расстояние между вектором и кластером (размер окрестности кластера). Если $d_c(x, c, \Omega) \leq \varepsilon$, то вектор находится в окрестности кластера и может быть включён в него. При этом границы кластера расширяются, чтобы включить в себя вектор, а в списки допустимых значений дискретных переменных добавляются соответствующие значения элементов вектора. Если же $d_c(x, c, \Omega) > \varepsilon$, то формируется новый кластер, инициализирующийся этим вектором, и добавляется в базу. Процесс обучения повторяется до тех пор, пока не будет обработана вся обучающая выборка. Малое значение ε приводит к созданию небольших по размеру кластеров, что обеспечивает более точный контроль, но значительно увеличивает размер базы знаний, делая невозможным мониторинг системы в реальном времени. Значение ε может быть подобрано таким образом, чтобы обеспечить баланс между скоростью работы и точностью контроля.

Функция расстояния $d_c(x, c, \Omega)$ может быть определена различными способами, зависящими от точки в кластере, до которой измеряется расстояние. В отличие от метода IMS, описанного в подразделе 1.2.6, использующего расстояние до центра кластера (рисунок 5а), предлагается использовать расстояние до ближайшей точки кластера (рисунок 5б), обеспечивающее более точную оценку удалённости вектора от границ кластера.

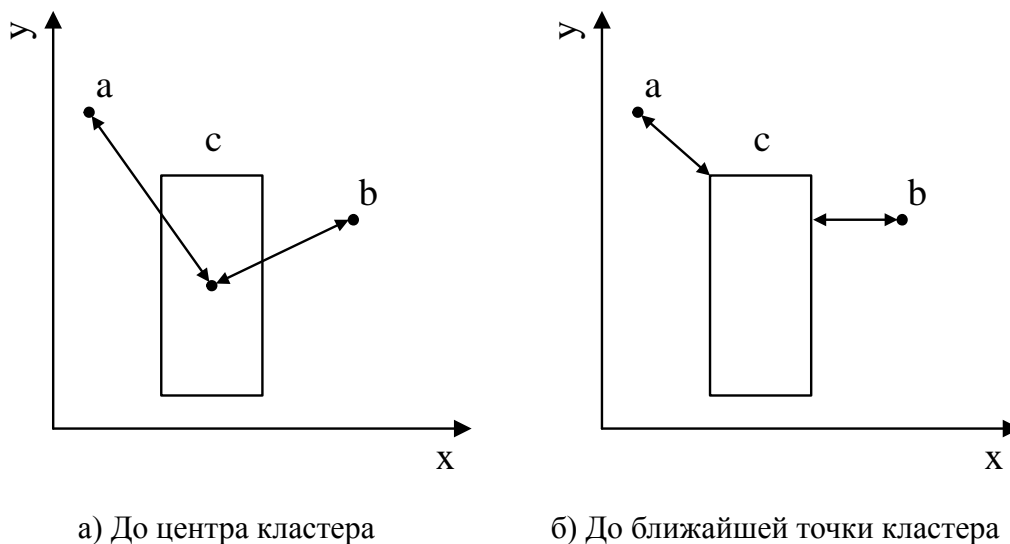


Рисунок 5 — Виды расстояний от вектора до кластера

Результатом работы данного этапа является модель системы, состоящая из ре-

жимов, каждый из которых содержит базу кластеров. Структура модели показана на рисунке 6.



Рисунок 6 — Структура модели системы для разрабатываемого метода

1.3.2.3 Мониторинг состояния системы

Для использования получившейся модели системы для мониторинга текущего состояния требуется получить из пришедшего вектора измерений входной вектор θ в формате, описанном в подразделе 1.3.1, и нормализовать его. Для этого над вектором измерений производятся операции, описанные в пункте 1.3.2.1. Нормализация производится с использованием характеристик, сохранённых на аналогичном этапе при подготовке данных обучающих выборок.

Метод по очереди перебирает режимы и проверяет, находится ли входной вектор внутри одного из кластеров в базе режима. Если входной вектор попал внутрь кластера, то система работает в том режиме, к которому относится данный кластер. Иначе путём измерения расстояний $d_c(x, c, \Omega)$ от входного вектора до каждого кластера находится ближайший кластер. Если $d_c(x, c, \Omega) \leq \varepsilon$, то вывод аналогичный с поправкой на то, что возможно отклонение от номинального поведения, поэтому метод даёт числен-

ную характеристику степени отклонения в виде расстояния $d_c(x, c, \Omega)$. В случае, когда $d_c(x, c, \Omega) > \varepsilon$ для всех кластеров во всех режимах, поведение системы считается аномальным; метод выводит ближайший режим и расстояние до него. Блок-схема данного процесса приведена в приложении Е.

1.4 Выбор программных средств реализации метода

К языку программирования и программной платформе в данной работе предъявлялись следующие требования:

- поддержка объектно-ориентированной парадигмы;
- удобство использования (скорость разработки);
- возможность повторного использования кода;
- быстроедействие.

Рассмотрим подробнее каждое из требований.

1.4.1 Предъявляемые требования

1.4.1.1 Поддержка объектно-ориентированной парадигмы

Объектно-ориентированное программирование (ООП) — парадигма программирования, в которой основными концепциями являются понятия объектов и классов. В случае языков с прототипированием вместо классов используются объекты-прототипы. В центре ООП находится понятие объекта. Объект — это сущность, которой можно посылать сообщения и которая может на них реагировать, используя свои данные. Объект — это экземпляр класса. Данные объекта скрыты от остальной программы (инкапсулированы). Наличие инкапсуляции достаточно для объектности языка программирования, но ещё не означает его объектной ориентированности — для этого требуется наличие наследования. Но даже наличие инкапсуляции и наследования не делает язык программирования в полной мере объектным с точки зрения ООП. Основные преимущества ООП проявляются только в том случае, когда в языке программирования реализован полиморфизм; то есть возможность объектов с одинаковой спецификацией иметь различную реализацию.

Основные концепции и понятия ООП представлены ниже.

Абстрагирование — это способ выделить набор значимых характеристик объекта, исключая из рассмотрения незначимые. Соответственно, абстракция — это набор

всех таких характеристик.

Инкапсуляция — это свойство системы, позволяющее объединить данные и методы, работающие с ними в классе, и скрыть детали реализации от пользователя.

Наследование — это свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом; новый класс — потомком, наследником или производным классом.

Полиморфизм — это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Класс — описываемая на языке терминологии (пространства имён) исходного кода модель ещё не существующей сущности (объекта). Фактически он описывает устройство объекта, являясь своего рода чертежом. Говорят, что объект — это экземпляр класса. При этом в некоторых исполняющих системах класс также может представляться некоторым объектом при выполнении программы посредством динамической идентификации типа данных. Обычно классы разрабатывают таким образом, чтобы их объекты соответствовали объектам предметной области.

Объект — сущность в адресном пространстве вычислительной системы, появляющаяся при создании экземпляра класса или копирования прототипа (например, после запуска результатов компиляции и связывания исходного кода на выполнение).

Прототип — объект-образец, по образу и подобию которого создаются другие объекты. Объекты-копии могут сохранять связь с родительским объектом, автоматически наследуя изменения в прототипе; эта особенность определяется в рамках конкретного языка. [32]

Основными принципами объектно-ориентированного проектирования представлены ниже. Данный набор принципов принято обозначать аббревиатурой *SOLID*.

Принцип единственной обязанности (Single Responsibility Principle): каждый объект должен иметь одну обязанность и эта обязанность должна быть полностью инкапсулирована в класс. Все его сервисы должны быть направлены исключительно на обеспечение этой обязанности. Таким образом, класс или модуль должны иметь одну и только одну причину измениться.

Принцип открытости/закрытости (Open/Closed Principle): программные сущно-

сти (классы, модули, функции и т.п.) должны быть открыты для расширения, но закрыты для изменения. Это означает, что такие сущности могут позволять менять свое поведение без изменения их исходного кода.

Принцип подстановки Барбары Лисков (Liskov Substitution Principle): объекты в программе могут быть заменены их наследниками без изменения свойств программы. Функции, которые используют базовый тип, должны иметь возможность использовать подтипы (наследники) базового типа, не зная об этом.

Принцип разделения интерфейса (Interface Segregation Principle): клиенты не должны зависеть от методов, которые они не используют. Интерфейсы, содержащие слишком много полей и методов, необходимо разделять на более маленькие и специфические, чтобы клиенты маленьких интерфейсов знали только о методах, которые необходимы им в работе. В итоге, при изменении метода интерфейса не должны меняться клиенты, которые этот метод не используют.

Принцип инверсии зависимостей (Dependency Inversion Principle): зависимости внутри системы строятся на основе абстракций. Модули верхнего уровня не зависят от модулей нижнего уровня. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций. [33]

Применение объектно-ориентированной парадигмы позволяет улучшить понимание исходного кода, упростить его дальнейшую поддержку, использование и усовершенствование, так как модули и классы в коде представляют собой проекции реальных объектов предметной области. В частности, в проектируемой системе такими объектами являются модель системы, режимы работы, кластеры, входные векторы и т.д. Поэтому требуется, чтобы язык программирования в полной мере поддерживал все концепции ООП.

1.4.1.2 Удобство использования

Под удобством использования подразумевается как логичность, приспособленность конструкций конкретного языка к поставленным задачам, так и удобство средств разработки (уже независимо от выбранного языка). Кроме того, следует учесть качество стандартной библиотеки и количество готовых компонентов (фреймворков, библиотек и т.п.), которые возможно использовать для решения типовых задач, программируя на данном языке. Удобство использования языка непосредственно влияет на вре-

мя, затрачиваемое на написание программ на нём.

1.4.1.3 Возможность повторного использования кода

Повторное использование кода — методология проектирования компьютерных и других систем, заключающаяся в том, что система (компьютерная программа, программный модуль) частично либо полностью должна состояться из частей, написанных ранее компонентов и/или частей другой системы, и эти компоненты должны применяться более одного раза (если не в рамках одного проекта, то хотя бы разных). Повторное использование — основная методология, которая применяется для сокращения трудозатрат при разработке сложных систем.

Самый распространённый случай повторного использования кода — библиотеки программ. Библиотеки предоставляют общую достаточно универсальную функциональность, покрывающую избранную предметную область.

Данная система разрабатывается, как готовый программный продукт, однако должна быть возможность встраивания элементов системы (в частности, реализации метода диагностики аномалий) в другие программные продукты и системы. Также должна быть возможность доработки элементов системы другими специалистами, для которых специализированный или редко используемый язык может создать затруднения и увеличить затраты на использование модулей данной системы. Поэтому язык должен быть достаточно популярным в профессиональной среде и предоставлять средства для удобного создания модулей и отдельных библиотек.

1.4.1.4 Быстродействие

В зависимости от языка программирования быстродействие реализаций одного и того же алгоритма может значительно отличаться. Несмотря на то, что систему не предполагается использовать во встраиваемых ЭВМ, а ПЭВМ могут обеспечить мощности, более чем достаточные для подобной задачи, данный критерий всё равно представляется важным, так как система должна иметь возможность работать в режиме реального времени.

1.4.2 Анализ языков программирования

Наиболее распространёнными и набирающими популярность языками программирования, обеспечивающими полноценную поддержку ООП и позволяющими

разрабатывать кроссплатформенные приложения для настольных ПЭВМ, являются следующие [34]:

- C++;
- C#;
- D;
- Java;
- Python;
- Ruby;
- Visual Basic .NET.

Рассмотрим каждый из них более подробно.

1.4.2.1 C++

C++ — компилируемый статически типизированный язык программирования общего назначения. Поддерживает такие парадигмы программирования как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование, обеспечивает модульность, отдельную компиляцию, обработку исключений, абстракцию данных, объявление типов (классов) объектов, виртуальные функции. Стандартная библиотека включает, в том числе, общеупотребительные контейнеры и алгоритмы.

Язык возник в начале 1980-х годов как разработка сотрудника Bell Labs Бьёрна Страуструпа.

C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования [34]. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (игр). Существует множество реализаций языка C++, как бесплатных, так и коммерческих и для различных платформ. Например, на платформе x86 это GCC, Visual C++, Intel C++ Compiler, Embarcadero (Borland) C++ Builder и другие. C++ оказал огромное влияние на другие языки программирования, в первую очередь на Java и C#.

Преимущества:

- достаточно высокая производительность. Язык спроектирован так, чтобы

дать программисту максимальный контроль над всеми аспектами структуры и порядка исполнения программы. Имеется возможность работы с памятью на низком уровне.

- мультипарадигменность языка и широчайший спектр возможностей;
- огромное количество готовых библиотек и расширений.

Недостатки:

- плохо продуманный синтаксис;
- унаследованные от языка Си низкоуровневые свойства существенно тормозят и затрудняют прикладную разработку;
- язык содержит опасные возможности, существенно снижающие качество и надёжность программ сразу по всем показателям. [35]

1.4.2.2 C#

C# — мультипарадигменный язык программирования. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework и впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270.

C# относится к семье языков с C-подобным синтаксисом. Язык имеет поддержку статической и динамической типизации, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников — языков C++, Pascal, Модула, Smalltalk и в особенности Java — C#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C#, в отличие от C++, не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

Код на языке C# транслируется в код на промежуточном языке MSIL (Microsoft Intermediate Language), который выполняется в виртуальной машине. Существует несколько реализаций для разных платформ. Хотя он и предназначен для генерации кода, исполняемого в среде .NET, сам по себе он не является частью .NET. Однако по-

сколько язык C# предназначен для применения на платформе .NET, то разработчику, важно иметь представление о .NET Framework, если он хочет эффективно разрабатывать приложения на C#.

Центральной частью каркаса .NET является его Общезыковая исполняющая среда – Common Language Runtime (CLR), или .NET runtime. Код, исполняемый под управлением CLR, часто называют управляемым кодом. Однако перед тем как код сможет исполниться CLR, любой исходный текст (на C# или другом языке) должен быть скомпилирован. Компиляция в .NET состоит из двух шагов:

- компиляция исходного кода в IL;
- компиляция IL в специфический для платформы код с помощью CLR.

Этот двухшаговый процесс компиляции очень важен, потому что наличие IL (управляемого кода) — это ключ ко многим преимуществам .NET.

К преимуществам .NET следует отнести наличие промежуточного языка Microsoft (MSIL). MSIL разделяет с псевдокодом Java идею низкоуровневого языка с простым синтаксисом (базирующегося на числовых кодах вместо текста), который может быть очень быстро транслирован в родной машинный код. Наличие этого кода с четко определенным универсальным синтаксисом дает ряд существенных преимуществ. Это значит, что файл, содержащий инструкции псевдокода, может быть размещен на любой платформе; во время исполнения финальная стадия компиляции может быть легко осуществлена, что позволит выполнить код на конкретной платформе. Другими словами, компилируя в IL, вы получаете платформенную независимость .NET.

Другим преимуществом подхода является повышение производительности. IL всегда компилируется оперативно (Just-In-Time, или JIT-компиляция). Вместо компиляции всего приложения за один проход (что может привести к задержкам при запуске), JIT-компилятор просто компилирует каждую порцию кода при ее вызове (just-in-time — оперативно). Если промежуточный код однажды скомпилирован, то результирующий машинный исполняемый код сохраняется до момента завершения работы приложения, поэтому его перекомпиляция при повторных вызовах не требуется.

Компания Microsoft заявляет, что такой процесс более эффективен, чем компиляция всего приложения при запуске, поскольку высока вероятность того, что большие куски кода приложения на самом деле не будут выполняться при каждом запуске. При использовании JIT-компилятора такой код никогда не будет скомпилирован. Это

объясняет, почему можно рассчитывать на то, что выполнение родного управляемого кода IL будет почти настолько же быстрым, как и выполнение родного машинного кода. Финальная стадия компиляции проходит во время выполнения, JIT-компилятор на этот момент уже знает, на каком типе процессора будет запущена программа. Это значит, что он может оптимизировать финальный исполняемый код, используя инструкции конкретного машинного кода, предназначенные для конкретного процессора.

Преимущества:

- мультипарадигменность;
- удобный и понятный синтаксис, поддерживающий огромное количество возможностей;
- наличие большой стандартной библиотеки;
- возможность интегрировать написанные на C# модули в системы, написанные на других .NET-совместимых языках;
- наличие сборщика мусора;
- высокое быстродействие.

Недостатки: ограниченное использование во встраиваемых системах. [36]

1.4.2.3 D

D — компилируемый, статически и строго типизированный мультипарадигменный язык с Си-подобным синтаксисом, созданный Уолтером Брайтом из компании Digital Mars. Изначально был задуман как переосмысление языка C++, однако, несмотря на значительное влияние C++, не является его вариантом. В D были заново реализованы некоторые свойства C++, также язык испытал влияние концепций из других языков программирования, таких как Java, Python, Ruby, C# и Eiffel. При создании языка D была сделана попытка соединить производительность компилируемых языков программирования с безопасностью и выразительностью динамических. Он является языком более высокого уровня, нежели C++. Выделение памяти в языке D полностью контролируется методикой «сборки мусора».

Преимущества:

- высокая производительность;
- мультипарадигменность;
- наличие сборщика мусора;

- удобный и понятный синтаксис.

Недостатки:

- язык только набирает популярность, поэтому число пользователей не так велико, как у других языков;
- непродуманная и имеющая ограниченный функционал стандартная библиотека. [37]

1.4.2.4 Java

Java — объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой виртуальной Java-машине (JVM) вне зависимости от компьютерной архитектуры. Дата официального выпуска — 23 мая 1995 года.

Программы на Java транслируются в байт-код, выполняемый виртуальной машиной Java (JVM) — программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор. Дюк, талисман Java

Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии Java является гибкая система безопасности благодаря тому, что исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером) вызывают немедленное прерывание.

Разработка приложений с использованием языка Java производится быстрее, чем на C++, так как Java избавлена от низкоуровневых проблем (таких, как, например, выделение и освобождение памяти вручную).

Преимущества:

- высокая распространённость языка;
- большое количество готовых библиотек и компонентов;
- богатая встроенная библиотека.

Недостатки:

- низкая производительность, связанная с реализацией виртуальной машины (JVM);
- ограниченные возможности языка. [38]

1.4.2.5 Python

Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. Python поддерживает динамическую типизацию, то есть тип переменной определяется только во время исполнения. В то же время стандартная библиотека включает большой объём полезных функций. Разработка языка Python была начата в конце 1980-х годов сотрудником голландского института CWI Гвидо ван Россумом.

Python поддерживает несколько парадигм программирования, в том числе структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений и удобные высокоуровневые структуры данных. Код в Питоне организовывается в функции и классы, которые могут объединяться в модули (они в свою очередь могут быть объединены в пакеты).

Эталонной реализацией Python является интерпретатор CPython, поддерживающий большинство активно используемых платформ. Есть реализации интерпретаторов для JVM (с возможностью компиляции), MSIL (с возможностью компиляции), LLVM и других.

Преимущества:

- синтаксис поддерживает большое число функций;
- существуют реализации языка для многих популярных платформ и виртуальных машин;
- богатая встроенная библиотека.

Недостатки:

- низкая производительность в интерпретируемых реализациях;
- отсутствие концепции будущего развития языка. [39]

1.4.2.6 Ruby

Ruby — динамический, рефлексивный, интерпретируемый высокоуровневый язык программирования для быстрого и удобного объектно-ориентированного программирования. Язык обладает независимой от операционной системы реализацией многопоточности, строгой динамической типизацией, сборщиком мусора и многими другими возможностями. Ruby близок по особенностям синтаксиса к языкам Perl и Eiffel, по объектно-ориентированному подходу — к Smalltalk. Также некоторые черты языка взяты из Python, Lisp, Dylan и CLU. Создателем языка является Юкихиро Мацумото. Первая версия была опубликована в 1995 году.

Ruby — полностью объектно-ориентированный язык. В нём все данные являются объектами, в отличие от многих других языков, где существуют примитивные типы. Каждая функция — метод. Ruby является мультипарадигменным языком: он поддерживает процедурный стиль (определение функций и переменных вне классов), объектно-ориентированный (всё — объект), функциональный (анонимные функции, замыкания, возврат значения всеми инструкциями, возврат функцией последнего вычисленного значения). Он поддерживает отражение, метапрограммирование, информацию о типах переменных на стадии выполнения.

Для Ruby существуют несколько реализаций: официальный интерпретатор, написанный на Си, JRuby — реализация для Java, интерпретатор для платформы .NET IronRuby и др.

Преимущества:

- существуют реализации языка для многих популярных платформ и виртуальных машин;
- язык имеет лаконичный синтаксис.

Недостатки:

- низкая производительность в интерпретируемых реализациях;
- низкая популярность языка при создании технических систем. [40]

1.4.2.7 Visual Basic .NET

Visual Basic .NET — объектно-ориентированный язык программирования, реализация языка Visual Basic для платформы .NET.

Преимущества:

- простой синтаксис;
- возможность интеграции в любые системы на платформе .NET.

Недостатки:

- ограниченность синтаксиса;
- концепция языка является устаревшей и не подвергается серьёзным изменениям. [41]

1.4.3 Выбор языка программирования

Проведя анализ вышеописанных языков программирования, можно заключить, что язык C# является наиболее подходящим для разработки данной системы. Немаловажным обстоятельством является то, что именно этот язык является наиболее знакомым автору из всех рассмотренных, что служит дополнительным плюсом в пользу данного языка, так как на изучение всех особенностей другого языка может уйти большое количество времени. Зачастую именно грамотное использование всех особенностей языка и встроенной библиотеки и обуславливает написание качественного кода.

1.5 Разработка программной реализации метода

Программная реализация была разработана на языке программирования C# 5.0 с использованием фреймворка .NET 4.5 в среде Microsoft Visual Studio 2013. Система разработана таким образом, чтобы обеспечить универсальность и лёгкость интеграции в другие смежные системы, позволить повторное использование отдельных модулей. Она имеет текстовый (консольный) интерфейс в соответствии с философией UNIX для возможности использования из других программных систем.

1.5.1 Архитектура системы

Архитектура системы проектировалась с учётом возможного расширения и дополнения функционала. Система имеет модульную структуру. Основные функциональные модули:

- Thesis.DDMS — реализация разработанного метода;
- Thesis.Orca — реализация алгоритма Orca, описанного в пункте 1.2.1;
- Thesis.DataCleansing — реализация фильтров, используемых для очищения обучающих выборок от аномалий;

- Thesis.Miscellaneous — содержит общие классы предметной области, абстракции и функционал, используемый в остальных модулях;
- Thesis.App — программное приложение (исполняемый файл), связывающее воедино остальные модули и предоставляющее интерфейс пользователя.

1.5.1.1 Чтение и обработка входных данных

Реализовано чтение из текстовых и бинарных файлов специального формата. Для обеспечения возможности добавления поддержки других источников и форматов данных представлены абстракции чтения и записи обучающих выборок и синтаксического анализа данных телеметрии.

UML-диаграмма классов, отвечающих за операции над данными, приведена в приложении Ж. Описание элементов представлено ниже.

FieldType — тип параметра в векторах входных данных. Возможные значения: *IgnoreFeature* (игнорировать данный параметр при чтении), *Continuous* (непрерывный параметр), *Discrete* (дискретный с фиксированным множеством значений), *DiscreteDataDriven* (дискретный, возможные значения параметра получаются из входных данных).

Field — класс параметра. Имеет имя, тип, весовой коэффициент, список возможных значений (для дискретных параметров).

Record — вектор данных (запись). Содержит идентификатор, значения непрерывных и дискретных параметров.

IRecordParser<T> — абстракция над конвертером, преобразующим единицу входных данных (строку, значения датчиков, подключенных к системе, и т.п.) в вектор (запись) типа *Record*.

PlainTextParser — реализация конвертера *IRecordParser* для преобразования текстовой строки в вектор.

DataFormatException — исключение, создающееся при ошибках в текстовых входных данных.

StringHelper — класс с вспомогательными методами для обработки строк. Позволяет разбить строку на части, используя указанные символы-разделители.

IDataReader — абстракция над чтением из источника данных. Предоставляет интерфейс для чтения векторов, сброса источника на начальную позицию, получения

текущей позиции, описания параметров векторов и флага достижения конца данных.

IDataWriter — абстракция над записью в источник данных. Предоставляет интерфейс для записи вектора и получения количества уже записанных векторов.

PlainTextReader — реализация интерфейса *IDataReader* для чтения описания параметров и обучающих выборок из текстовых файлов.

BinaryDataReader — реализация интерфейса *IDataReader* для чтения описания параметров и обучающих выборок из бинарных файлов специального формата.

BinaryDataWriter — реализация интерфейса *IDataWriter* для записи описания параметров и обучающих выборок в бинарные файлы специального формата.

BatchDataReader — реализация интерфейса *IDataReader* для чтения записей блоками по указанному количеству штук. Представляет собой реализацию паттерна декоратор [42] над объектом типа *IDataReader*.

IScaling — абстракция над методом нормализации данных. Предоставляет интерфейс для прямого и обратного масштабирования вектора или только его непрерывных параметров.

MinmaxScaling — реализация интерфейса *IScaling* для минимаксной нормализации по формуле (16а).

StandardScaling — реализация интерфейса *IScaling* для нормализации с помощью стандартного отклонения по формуле (16б).

ScaleDataReader — реализация интерфейса *IDataReader* для нормализации данных при чтении из источника. Представляет собой реализацию паттерна декоратор [42] над объектом типа *IDataReader*.

1.5.1.2 Поиск аномалий в обучающих выборках с помощью алгоритма Orca

UML-диаграмма классов для поиска аномалий в обучающих выборках приведена в приложении 3. Описание элементов представлено ниже.

OrcaAD — класс, реализующий алгоритм Orca (см. подраздел 1.2.1).

Outlier — структура данных, хранящая информацию об аномалии. Содержит идентификатор записи и значение степени аномальности.

BinaryShuffle — реализует рандомизацию данных обучающей выборки (требуется для работы алгоритма Orca). Записывает рандомизированные данные в бинарный

файл. Рандомизация происходит в n итераций следующим образом: создаётся m временных файлов, в которые случайным образом распределяются элементы обучающей выборки. Затем файлы объединяются в порядке, определяемом с помощью перетасовки Дональда Кнута [43].

DataHelper — предоставляет удобный интерфейс для класса `BinaryShuffle`.

ScoreFunction — описывает тип функции оценки степени аномальности для алгоритма Orca. Класс ScoreFunctions содержит несколько вариантов таких функций.

BinaryHeap — реализация структуры данных «двоичная куча» [44]. Используется алгоритмом Orca для определения наиболее удалённых ближайших соседей.

1.5.1.3 Очистка обучающих выборок от аномалий

UML-диаграмма классов, используемых для очистки обучающих выборок от аномалий, приведена в приложении И. Описание элементов представлено ниже.

IAnomaliesFilter — абстракция над типом фильтра аномалий. Предоставляет интерфейс для метода *Filter*, принимающего на вход коллекцию объектов типа *Outlier* и возвращающего из неё только те элементы, которые прошли через фильтр (которые являются аномалиями).

DifferenceFilter — разностный фильтр. Псевдокод алгоритма фильтрации показан в листинге 2.

ThresholdFilter — пороговый фильтр. Псевдокод алгоритма фильтрации показан в листинге 3.

GaussianFilter — гауссовый фильтр. Псевдокод алгоритма фильтрации показан в листинге 4.

ScaleDataReader — реализация интерфейса *IDataReader* для пропуска аномальных записей при чтении из источника. Если текущая запись является аномальной, то объект данного класса переходит к следующей записи. Представляет собой реализацию паттерна декоратор [42] над объектом типа *IDataReader*.

1.5.1.4 Построение модели системы

UML-диаграмма классов для построения модели системы приведена в приложении К. Описание элементов представлено ниже.

DistanceMetric — описывает тип метрики пространства. Класс DistanceMetrics содержит несколько вариантов таких метрик, в частности, показанную в формуле (15),

и её квадрат.

Weights — класс, содержащий весовые коэффициенты параметров.

Cluster — представляет собой кластер для разработанного метода. Содержит значения непрерывных параметров для верхней и нижней границы и список допустимых значений для каждого дискретного параметра. Позволяет добавлять в себя вектор, расширяя свои границы и обновляя списки допустимых значений так, чтобы добавляемый вектор находился внутри кластера.

Листинг 2 — Псевдокод алгоритма фильтрации для разностного фильтра

Входные данные: значения степени аномальности для записей в обучающей выборке, отсортированные по убыванию; Δ , максимальная разность степеней аномальности для двух подряд идущих записей

Выходные данные: аномальные записи

- 1: **если** количество записей < 2 **то**
 - 2: **вернуть как результат** \emptyset
 - 3: **конец**
 - 4: **для** i от количества записей в выборке до 0 **выполнять**
 - 5: $\delta \leftarrow$ разность значений степени аномальности $(i - 1)$ -й и i -й записей
 - 6: **если** $\delta > \Delta$ **то**
 - 7: **вернуть как результат** i записей с самыми большими значениями степени аномальности
 - 8: **конец**
 - 9: **конец цикла**
 - 10: **вернуть как результат** \emptyset
-

Листинг 3 — Псевдокод алгоритма фильтрации для порогового фильтра

Входные данные: значения степени аномальности для записей в обучающей выборке, отсортированные по убыванию; ρ , максимальное значение степени аномальности

Выходные данные: аномальные записи

- 1: $\mu \leftarrow$ минимальное значение степени аномальности для всех записей в обучающей выборке
 - 2: **для всех** записей в выборке **выполнять**
 - 3: $x \leftarrow$ значение степени аномальности текущей записи
 - 4: **если** $x - \mu > \rho$ **то**
 - 5: **вернуть как результат** текущую запись
 - 6: **конец**
 - 7: **конец цикла**
-

Листинг 4 — Псевдокод алгоритма фильтрации для гауссового фильтра

Входные данные: значения степени аномальности для записей в обучающей выборке, отсортированные по убыванию

Выходные данные: аномальные записи

- 1: $M \leftarrow$ мат. ожидание степени аномальности
 - 2: $D \leftarrow$ дисперсия степени аномальности
 - 3: $\sigma \leftarrow \sqrt{D}$
 - 4: **для всех** записей в выборке **выполнять**
 - 5: $x \leftarrow$ значение степени аномальности текущей записи
 - 6: **если** $x > M + 3\sigma$ **то**
 - 7: **вернуть как результат** текущую запись
 - 8: **конец**
 - 9: **конец цикла**
-

ClusterDistance — описывает тип функции для определения расстояния между вектором и кластером. Класс ClusterDistances содержит несколько вариантов таких функций.

ClusterDatabase — база кластеров. Реализует всю логику по созданию новых кластеров и проверке попадания векторов в существующие.

Regime — представляет режим работы системы. Содержит имя и базу кластеров. Реализует логику проверки принадлежности вектора режиму и расчёта расстояния до него.

SystemModel — представляет модель системы. Содержит в себе список режимов работы, реализует логику по добавлению режимов и определению ближайшего к входному вектору режима.

1.5.2 Руководство пользователя

1.5.2.1 Установка программного обеспечения

Для работы программного обеспечения требуется ПЭВМ с установленной ОС Windows 7 и выше и наличием .NET Framework 4.5. Программное обеспечение представлено в виде нескольких файлов динамических библиотек (DLL) и файла приложения (*Thesis.App.exe*). Для установки программы на компьютер достаточно скопировать данные файлы в любую папку на ПЗУ. Имя и расположение папки не имеет значения. На этом установку системы можно считать законченной.

1.5.2.2 Формат входных данных

Для работы программы требуется минимум два файла: файл с описанием параметров векторов и файл с обучающей выборкой для номинального режима работы системы.

Во всех файлах существует возможность оставлять однострочные комментарии. Комментарий начинается с символа „%“ и продолжается до конца строки. При чтении файлов программой комментарии игнорируются.

Файл с описанием параметров должен заполняться следующим образом. Каждый параметр представляется файле отдельной строкой, которая имеет формат:

[весовой коэффициент] : имя : описание_параметра

Разделителями в данном файле служат двоеточие, точка с запятой и запятая. Весовой коэффициент является необязательным параметром.

Если параметр не должен учитываться программой, указывается ключевое слово *ignorefeature*.

Пример: time : ignorefeature

Поддерживаются следующие типы параметров:

- непрерывные (ключевое слово *continuous*).

Пример: pressure : continuous

- дискретные с фиксированным множеством значений. Для таких параметров все возможные значения указываются после имени.

Пример: state : opened, closed

- дискретные с открытым множеством значений (ключевое слово *discrete*). Для таких параметров системы сама определяет возможные значения на основе данных из обучающих выборок.

Пример: orbit : discrete

Файлы с данными обучающих выборок представляют собой набор строк (записей), каждая из которых представляет собой вектор с набором значений. Значения в строке разделяются запятыми или точками с запятой. Количество, состав и тип значений должны строго соответствовать указанным в файле с описанием параметров. Если значение какого-либо параметра в данной записи отсутствует, вместо него ставится знак вопроса (по умолчанию) либо строка, указанная при запуске программы в качестве соответствующего параметра.

Пример записи:

16:39:50, 45.5, opened, ?, high

Имя файла (без расширения) для каждой обучающей выборки программа воспринимает как название соответствующего режима. Например, файл *Nominal.txt* содержит обучающую выборку для режима «Nominal».

1.5.2.3 Запуск программного обеспечения

Исполняемым файлом программы является файл *Thesis.App.exe*. Программа при запуске принимает на вход параметры, указанные в таблице Л.1 приложения Л.

Пример команды запуска:

```
thesis.app fields.txt -r nominal1.txt nominal2.txt [-a regime1.txt regimeN.txt] [-f threshold 0.5] [-d kmeans] [-m sqreucld] [-n standard] [-v N/A]
```

Если программа запущена без параметров либо параметры указаны неверно, то будет выведена справочная информация, как показано на рисунке 7.

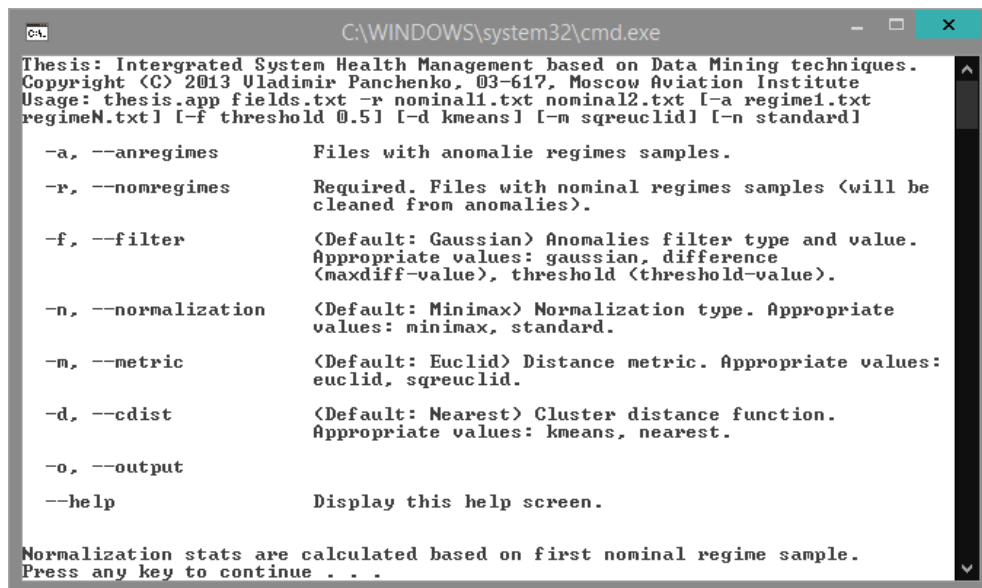


Рисунок 7 — Экран со справочной информацией

1.5.2.4 Работа с программным обеспечением

Если все параметры для запуска были указаны корректно, программа запрашивает у пользователя ввод порогового значения ε , как показано на рисунке 8.

После ввода программа отфильтровывает аномалии в указанных обучающих

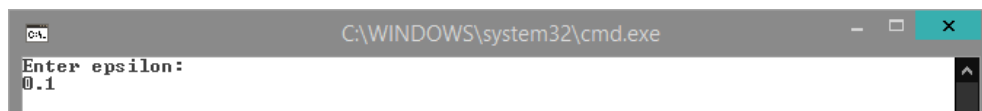


Рисунок 8 — Ввод порогового значения в программу

выборках и строит модель системы. Сначала выводятся найденные аномалии для указанных обучающих выборок номинальных режимов, а затем построенная база кластеров для каждого режима работы (рисунок 9).

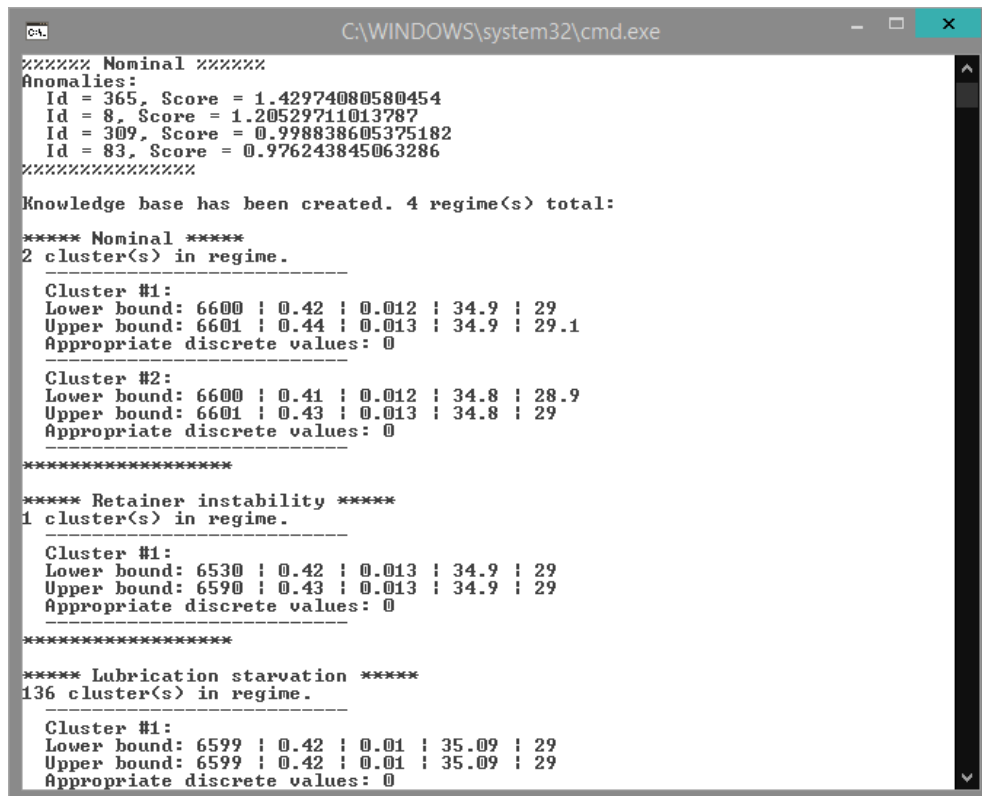


Рисунок 9 — Построенная программой модель системы

После чего программа готова к мониторингу поступающих телеметрических данных, о чём сообщает пользователю соответствующим сообщением (рисунок 10). Для выхода из режима мониторинга необходимо нажать *Enter*.

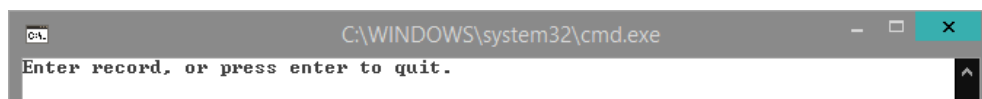
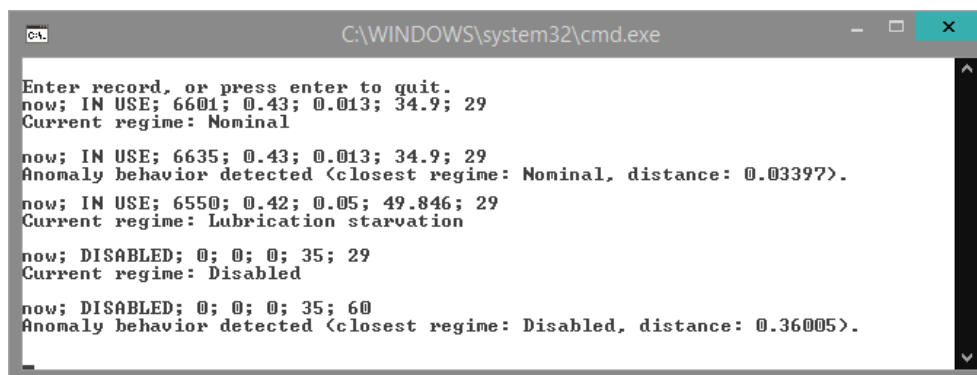


Рисунок 10 — Режим мониторинга: ввод записи

Записи вводятся в том же формате, что и во входных файлах. После ввода записи программа показывает результат мониторинга (текущий режим/текущий режим и

расстояние до него/факт наличия аномалии, ближайший режим и расстояние до него).
Пример работы программы в режиме мониторинга приведён на рисунке 11.



```
Enter record, or press enter to quit.  
now; IN USE; 6601; 0.43; 0.013; 34.9; 29  
Current regime: Nominal  
  
now; IN USE; 6635; 0.43; 0.013; 34.9; 29  
Anomaly behavior detected (closest regime: Nominal, distance: 0.033977).  
now; IN USE; 6550; 0.42; 0.05; 49.846; 29  
Current regime: Lubrication starvation  
  
now; DISABLED; 0; 0; 0; 35; 29  
Current regime: Disabled  
  
now; DISABLED; 0; 0; 0; 35; 60  
Anomaly behavior detected (closest regime: Disabled, distance: 0.36005).
```

Рисунок 11 — Режим мониторинга

1.6 Тестирование системы

2 Расчет экономической эффективности системы

2.1 Введение

Для оценки экономической эффективности программно-аппаратного продукта требуется:

- определить целесообразность разработки;
- определить трудоёмкость и затраты на создание;
- определить показатели экономической эффективности разработки.

Результатом выполнения данной части является обоснование технической, экономической и научной значимости и целесообразности продукта в соответствии с [45]. Объектом технико-экономического анализа является программная система мониторинга состояния ЛА на основе методов интеллектуального анализа данных.

2.2 Определение целесообразности разработки

Для обоснования целесообразности разработки продукта необходимо:

- выбрать аналог (если таковой имеется);
- сформулировать перечень функциональных характеристик по предлагаемому варианту разработки продукта;
- определить конкретные уровни характеристик и их значимость;
- определить индекс технического уровня программного продукта.

Функционально-технические характеристики разрабатываемого программного продукта представлены в таблице 7.

Индекс технического уровня разрабатываемого программного продукта определяется по формуле (17):

$$J_{TY} = \sum_{i=1}^n \frac{\alpha_i}{\alpha_{i0}} \mu_i, \quad (17)$$

где α_i — уровень i -й функционально-технической характеристики проектируемого алгоритма;

α_{i0} — уровень i -й функционально-технической характеристики базового алгоритма;

Таблица 7 — Функционально-технические характеристики

Функциональные характеристики	Единица измерения	Величина функциональных характеристик		Значимость характеристик
		Аналог	Новый вариант	
Простота использования	По 10-бальной шкале	2	9	0.05
Быстродействие	По 10-бальной шкале	5	10	0.2
Открытость	По 10-бальной шкале	3	10	0.15
Точность вычислений	По 10-бальной шкале	8	8	0.2
Надёжность	По 10-бальной шкале	9	7	0.2

μ_i — значимость i -го параметра;

n — количество рассматриваемых параметров.

$$J_{TY} = \frac{9}{2} \cdot 0.05 + \frac{10}{5} \cdot 0.2 + \frac{10}{3} \cdot 0.15 + \frac{8}{8} \cdot 0.2 + \frac{7}{9} \cdot 0.2 = 1.48.$$

Значение показателя технического уровня разрабатываемого программного продукта превышает 1 и равно 1.48. Полученный результат является подтверждением целесообразности разработки продукта.

2.3 Определение трудоемкости и затрат на создание ПП

Основой для определения затрат на создание ПП является показатель трудоемкости работ. В таблице 8 представлена структура затрат труда на создание ПП.

Таблица 8 — Структура затрат труда на создание ПП

№ п/п	Наименование (стадии) этапа работ	Доля работ на стадии (этапе) в общем объёме работ, %
1	Анализ предметной области и изучение средств разработки	3
2	Изучение программируемой задачи	5
3	Определение входных и выходных данных	3
4	Анализ методов решения задачи	5
5	Составление структуры ПП	4
6	Технико-экономическое обоснование выбора вариантов решения задачи	5
7	Уточнение и доработка выбранного варианта решения	3
8	Создание ПП	35
9	Отладка ПП	22
10	Испытание и анализ работы ПП в реальных условиях	10
11	Составление технической документации	5
	ИТОГО	100

Затраты труда определяются по формуле (18).

$$t_{ПРТ} = t_O + t_{II} + t_A + t_K + t_{OT} + t_d \quad (18)$$

$B = 2$ — увеличение затрат труда на изучение и постановку задачи вследствие их сложности и новизны;

$K = 0.8$ — коэффициент квалификации разработчика;

$$Q = q \cdot K_c \cdot (1 + \sum^n K_k) = 500 \cdot 1.5 \cdot (1 + (0.2 + 0.1)) = 975.$$

$t_O = 40$ — затраты труда на подготовку описания задачи;

$$t_{II} = \frac{Q \cdot B}{75 \cdot K} = 32.5 \text{ — затраты труда на изучение и постановку задачи;}$$

$$t_A = \frac{Q}{20 \cdot K} = 60.94 \text{ — затраты труда на проектирование системы;}$$

$t_K = \frac{Q}{10 \cdot K} = 121.88$ — затраты труда на программирование;

$t_{OT} = \frac{Q}{5 \cdot K} = 243.75$ — затраты труда на отладку программы;

$t_D = \frac{1.75 \cdot Q}{15 \cdot K} = 142.19$ — затраты труда на подготовку документации.

Таким образом, $t_{ПРТ} = 40 + 32.5 + 60.94 + 121.88 + 243.75 + 142.19 = 641.26$.

2.4 Определение исполнителей

Исполнители указаны в таблице 9.

Таблица 9 — Исполнители

Категория исполнителей	Число исполнителей	Зарплата с учетом премии (руб./мес.)	Часовые тарифные ставки, руб.
Инженер-программист	1	60000	360

2.5 Расчет заработной платы исполнителей

Оплата труда персонала определяется на основе общей трудоёмкости создания ПП по формуле (19).

$$ЗП_{ПП} = \sum_{i=1}^k T_i \cdot \bar{\tau}_i, \quad (19)$$

где k — количество этапов;

T_i — трудоёмкость i -го этапа;

$\bar{\tau}_i$ — средняя дневная тарифная ставка оплаты i -го этапа.

Результаты приведены в таблице 10.

Таблица 10 — Заработная плата исполнителей

№ п/п	Наименование этапов и работ	Трудоёмкость стадии (чел.-ч.)	Часовая ставка (руб/ч)	Зарплата за работу (руб)
1	Анализ предметной области и изучение средств разработки	40	360	6000
2	Изучение программируемой задачи	32.5	360	11700
3	Определение входных и выходных данных			
4	Анализ методов решения задачи			
5	Составление структуры ПП	60.94	360	21938.40
6	Технико-экономическое обоснование выбора вариантов решения задачи			
7	Уточнение и доработка выбранного варианта решения			
8	Создание ПП	121.88	360	43876.80
9	Отладка ПП	243.75	360	87750
10	Испытание и анализ работы ПП в реальных условиях			
11	Составление технической документации	142.19	360	51188.40
	ИТОГО	641.26	—	230853.60

2.6 Социальные отчисления

Социальные отчисления основных исполнителей составляет 30.2% от $ЗП_{ПП}$.

$$З_{CO} = 230853.60 \cdot 0.302 = 69717.79 \text{ (руб.)}$$

2.7 Накладные расходы

Под накладными расходами понимаются расходы на электроэнергию в период первого полугодия эксплуатации системы. Они рассчитываются по формуле (20).

$$K_{ЭЭ} = N_{\text{раб.дн.}} \cdot \sum P \cdot N_{\text{часов}} \cdot N_{\text{лет}} \cdot C_{ЭЭ}, \quad (20)$$

где $N_{\text{раб.дн.}}$ — количество рабочих дней в году;

$\sum P$ — суммарная потребляемая в час мощность оборудования (компьютер используется в течение всего рабочего дня, а принтеры — в среднем в течение половины рабочего дня);

$N_{\text{часов}}$ — количество рабочих часов в день;

$N_{\text{лет}}$ — количество лет разработки или использования системы;

$C_{ЭЭ}$ — стоимость одного киловатт/часа электроэнергии.

Таким образом, $K_{ЭЭ} = 224 \cdot 4.3 \cdot 8 \cdot 0.5 \cdot 4.5 = 17337.6 \text{ (руб.)}$

2.8 Прочие расходы

Прочие прямые расходы — это расходы на использование машинного времени. Система будет разрабатываться в течение 90 дней в среднем по 8 часов ежедневно. При стоимости машинного времени 13 руб./час, получаем:

$$З_{\text{м.в.}} = 90 \cdot 8 \cdot 13 = 9360.00 \text{ (руб.)}$$

Сведём все расходы на создание и эксплуатацию системы в таблицу 11.

2.9 Расчет стоимости

Цена программного продукта определяется исходя из принципа обеспечения безубыточности деятельности организации, получения прибыли, позволяющей выплачивать обязательные платежи в бюджет и инвестировать расширение деятельности.

Цена первоначальной продажи определяется по формуле (21).

$$Ц_{\text{НТПр}}^n = З_{\text{НТПр}} + \frac{ЗП_{\text{ПП}} \cdot \rho_{\text{ЗП}}}{100}, \quad (21)$$

Таблица 11 — Сводная таблица расходов

№ п/п	Виды расходов	Расходы, руб.	Удельный вес, %
1	Заработная плата разработчиков	230853.60	70.54
2	Социальные отчисления	69717.79	21.3
3	Накладные расходы	17337.6	5.3
4	Прочие расходы	9360	2.86
	ИТОГО	327268.99	100

где $З_{НТПр}$ — текущие затраты на создание, определяющиеся по формуле (22):

$$З_{НТПр} = З_{П_{mn}} + З_{М.В.}; \quad (22)$$

$З_{П_{mn}}$ — оплата труда основного персонала в общих текущих затратах на создание программного продукта;

$\rho_{ЗП}$ — уровень рентабельности (прибыли по отношению к оплате труда персонала), обеспечивающий безубыточность деятельности ($\rho_{ЗП} = 200\%$).

Таким образом,

$$З_{НТПр} = 230853.60 + 9360 = 240213.60 \text{ руб.};$$

$$Ц_{НТПр}^n = 240213.60 + \frac{230853.60 \cdot 200}{100} = 701920.80 \text{ руб.}$$

2.10 Оценка экономической эффективности

Так как данная дипломная работа связана с разработкой алгоритмов и программ, то $Э_{НТП}$ определяется по формуле (23).

$$Э_{НТП} = \sum_{i=1}^n \Delta T_{mi} \cdot C_{BT}, \quad (23)$$

где ΔT_{mi} — экономия машинного времени, ч.;

C_{BT} — стоимость одного машинного часа, руб.;

$n = 3000$ — количество задач, решаемых в год.

$$\mathcal{E}_{НТП} = 3000 \cdot 8 \cdot 50 = 1200000 \text{ (руб.)}$$

Уровень экономической эффективности ($E_{ПП}$) и срок окупаемости затрат на создание алгоритмов и ПП (T_{OK}) определяется по формулам (24) и (25).

$$E_{ПП} = \frac{\mathcal{E}_{НТП}}{C_{НТПр}} \quad (24)$$

$$T_{OK} = \frac{1}{E_{ПП}} \quad (25)$$

Таким образом,

$$E_{ПП} = \frac{1200000}{701920.80} = 1.71;$$

$$T_{OK} = \frac{1}{1.71} = 0.59 \text{ (года)}.$$

Так как уровень экономической эффективности составляет 1.71, то можно сделать вывод о том, что разработанный ПП выгоден с экономической точки зрения.

2.11 Календарное планирование

Календарный план работ представлен в таблице 12.

Таблица 12 — Календарный план работ

№ п/п	Наименование этапов (стадий, видов работ)	Удельный вес, %	Трудоемкость этапа, чел.-ч.	Кол-во ис- полнителей	Длительность этапа
1	Анализ предметной области и изучение средств разработки	6.24	40	1	6
2	Изучение программируемой задачи	2.31	14.8	1	1
3	Определение входных и выходных данных	1.21	7.76	1	5

Продолжение таблицы 12

№ п/п	Наименование этапов (стадий, видов работ)	Удельный вес, %	Трудоемкость этапа, чел.-ч.	Кол-во ис- полнителей	Длительность этапа
4	Анализ методов решения задачи	1.55	9.94	1	14
5	Составление структуры ПП	3.7	23.74	1	6
6	Технико- экономическое обоснование выбора вариантов решения задачи	4	25.66	1	6
7	Уточнение и доработка выбранного варианта решения	1.8	11.54	1	4
8	Составление ПП	19.01	121.88	1	39
9	Отладка ПП	22	141.08	1	16
10	Испытание и анализ работы ПП в реальных условиях	16.01	102.67	1	13
11	Составление технической документации	22.17	142.19	1	8
	ИТОГО	100	641.26	1	118

Производственный цикл каждого этапа определяется по формуле (26).

$$T_{uj} = \frac{T_j}{t_{pd} \cdot q_j}, \quad (26)$$

где T_j — трудоёмкость j -ой стадии (j -го этапа), чел.-час.;

t_{pd} — продолжительность рабочего дня, час.;

q_j — количество работников, одновременно участвующих в выполнении работ на j -ой стадии (j -м этапе), чел.

На основании данных таблицы 12 построен сетевой график (рисунок 12).

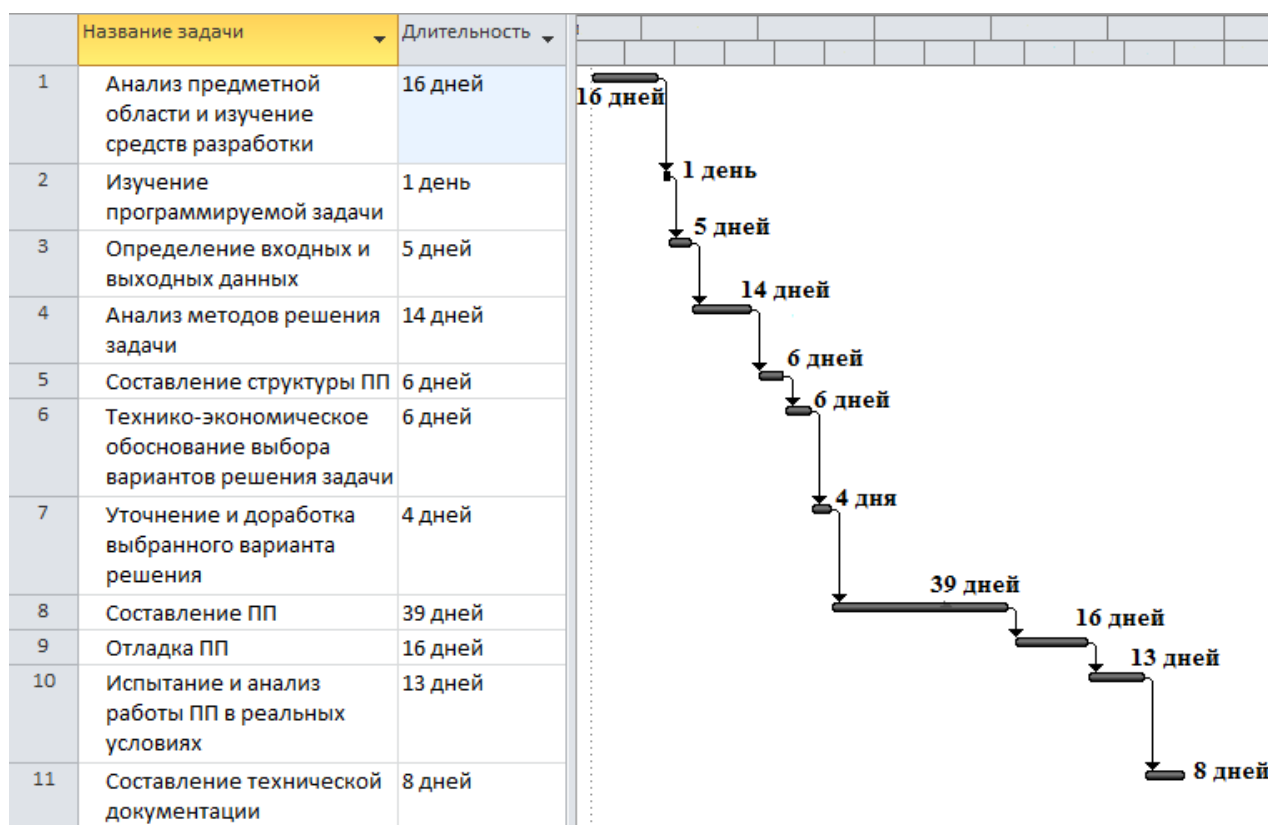


Рисунок 12 — Календарный план-график работ

2.12 Выводы

В экономической части дипломной работы получены следующие значения экономических показателей:

- Технический уровень $J_{TY} = 1.48$. Значение этого показателя должно быть больше 1. Полученное значение говорит о высоком техническом уровне разрабатываемого изделия.

– Уровень экономической эффективности $E_{III} = 1.71$.

На основании показателей экономической эффективности считаем разрабатываемую систему экономически эффективной и внедрение в производство целесообразным.

3 Охрана труда и окружающей среды

3.1 Анализ условий труда

3.1.1 Обеспечение условий труда в отделе разработки программного обеспечения

Дипломная работа посвящена разработке системы мониторинга состояния ЛА на основе алгоритмов интеллектуального анализа данных. Разработка производится на персональном компьютере и предполагает длительное пребывание за ним инженера.

Применение персонального компьютера освобождает человека от непроизводительной работы, связанной с обработкой информации, изменяет характер его труда. Однако при этом увеличивается доля умственного и нервно-напряженного труда, возрастает психоэмоциональная нагрузка. При значительной трудовой нагрузке, нерациональной организации работы и неблагоприятных факторах производственной среды быстро снижается работоспособность операторов, уменьшается производительность труда и ухудшается качество работы, может развиваться перенапряжение, а в отдельных случаях возникнуть срыв трудовой деятельности — дистресс.

В данном разделе проводится анализ условий труда в отделе разработки информационных систем с целью обеспечения безопасности и удобства, требуемых для работы инженера.

3.1.2 Характеристика помещения

Помещение находится в здании Московского Авиационного Института и представляет собой кафедральную лабораторию со следующими размерами:

- длина 6 м;
- ширина 4 м;
- высота 3.5 м.

Площадь: $6 \times 4 = 24 \text{ м}^2$.

Объём: $6 \times 4 \times 3.5 = 84 \text{ м}^3$.

Количество рабочих мест — 4.

Количество одновременно находящихся в помещении сотрудников не превышает 4 человек.

План помещения приведён на рисунке 13.

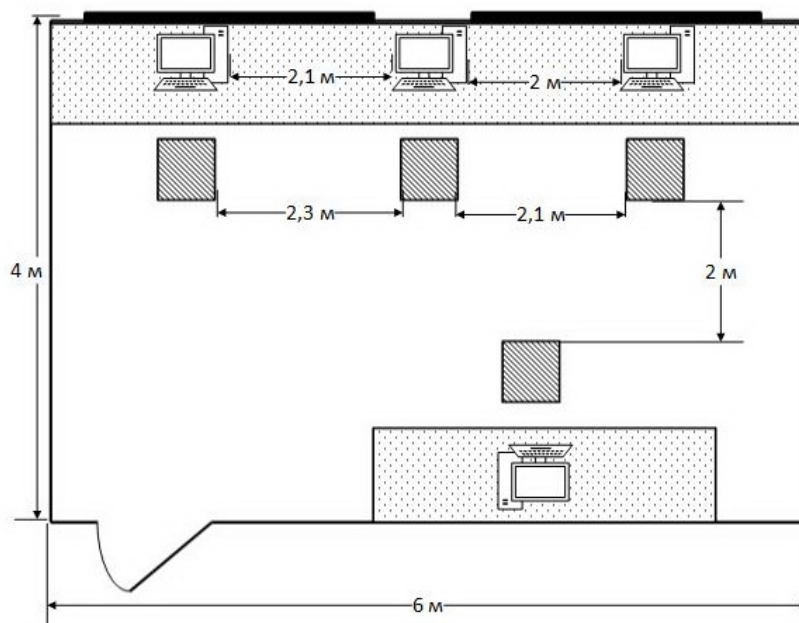


Рисунок 13 — План помещения

Нормативные требования к площади и объёму рабочих мест определены в [46]:

- площадь на одно рабочее место с ВДТ или ПЭВМ для взрослых пользователей должна составлять не менее 6 м^2 ;
- объём — не менее 20 м^3 .

Фактические значения на каждого сотрудника:

- площадь: $24/4 = 6 \text{ м}^2$;
- объём: $84/4 = 21 \text{ м}^3$.

Данные значения показывают, что кафедральная лаборатория полностью соответствует установленным нормам.

В помещении имеются 2 оконных проёма высотой 1,6 м и шириной 2,3 м, которые выходят на юго-запад.

Искусственное освещение представляет собой 6 светильников, расположенных параллельно окнам в 2 ряда.

3.1.3 Характеристика производственного процесса

Разработка программного обеспечения производится на ПЭВМ с подключенными к ней периферийными устройствами.

3.1.4 Характеристика используемого оборудования

В процессе разработки используется следующее оборудование:

а) ПЭВМ:

- 1) процессор Intel Core i5 3,60 ГГц;
- 2) оперативная память 8 Гб;
- 3) жёсткий диск 1 Тб;
- 4) напряжение питания 220 В.

б) ЖК монитор с диагональю 23 дюйма (58,42) ASUS VX239H:

- 1) частота 75 Гц;
- 2) яркость 250 кд/м²;
- 3) динамическая контрастность 8 000 000 : 1;
- 4) напряжение питания 220 В.

в) Клавиатура Logitech K330;

г) Мышь A7Tech X;

д) Принтер HP LaserJet 1005M:

- 1) напряжение питания 220 В.

3.1.5 Санитарно-гигиенические факторы

3.1.5.1 Микроклимат помещения

Микроклимат в рабочем помещении должен соответствовать [47].

Согласно [47], работа разработчика ПО относится к категории «Легкая – Ia», т.к. лёгкие физические работы — работы с расходом энергии не более 150 ккал (174 Вт), а категория Ia подразумевает энергозатраты до 120 ккал/ч (139 Вт).

Рабочее место разработчика ПО является постоянным, т.к. он находится на нём большую часть рабочего времени (более 50%).

Нормативные и фактические значения для категории работ «Легкая – Ia», лёгкие физические работы — работы с расходом энергии не более 150 ккал (174 Вт). Категория Ia подразумевает энергозатраты до 120 ккал/ч (139 Вт).

Рабочее место разработчика модели является постоянным, т.к. он находится на нём большую часть рабочего времени (более 50%).

Нормативные и фактические значения для категории работ «Легкая – Ia» и постоянного рабочего места приведены в таблице 13.

Таблица 13 — Значения характеристик микроклимата помещения

	Температура, °С	Относительная влажность, %	Скорость движения, м/с
Допустимые значения	22–24 — Холодный период 23–25 — Теплый период	40–60	0.1
Фактические значения	22–24 — Холодный период 25–30 — Теплый период	45–55	<0.1

Фактические значения параметров микроклимата данного помещения удовлетворяют допустимым значениям для холодного периода года. Во время теплого периода в помещении может преобладать повышенная температура из-за отсутствия кондиционера, который бы мог её регулировать.

3.1.5.2 Производственное освещение

Освещённость регламентируется [48].

Наименьший размер объекта различения в работе инженера составляет 0.3 мм. Объектом является символ, выводимый на экран монитора (наименьшим символом является точка). Зрительная работа относится к III разряду — высокая точность (наименьший размер объекта различения от 0.3 до 0.5 мм).

Контраст объекта с фоном средний, фон светлый, что соответствует подразряду б разряда III.

Требования к освещению помещений промышленных предприятий для подразряда б разряда III:

- При системе комбинированного освещения освещенность равна: всего — 1000 лк, в т.ч. от общего — 200 лк;
- При системе общего освещения освещённость равна 300 лк.

Система освещения в комнате общая, состоящая из 6 потолочных светильников ЛПО 46, в каждом из которых установлены 4 люминесцентные лампы ЛТБ мощностью 20 Вт и световым потоком 1100 лм. Светильники расположены в два ряда параллельно окнам. Фактическая освещенность составляет 275 лк, что полностью удовлетворяет нормативным значениям [48].

3.1.5.3 Шум

Источники шума в данном помещении: охлаждающие системы ПЭВМ (охлаждение процессоров).

Уровни шума на рабочих местах инженера ПЭВМ должны соответствовать [49].

Допустимые значения уровня шума при проектировании и программировании на рабочих местах в помещениях проектно-конструкторских бюро; расчётчиков, программистов вычислительных машин, в лабораториях для теоретических работ и обработки данных: не более 50 дБА.

Фактические значения уровня шума: не более 45 дБА, что подтверждается расчётами в разделе 3.3.

Согласно [49], значения уровня шума на рабочем месте удовлетворяют установленным требованиям.

3.1.5.4 Электромагнитное излучение

Во время работы ПЭВМ возникают электромагнитные поля, которые оказывают негативное влияние на организм человека.

Источниками электромагнитных полей на рабочем месте инженера являются системные блоки. Современные корпуса системных блоков ПЭВМ позволяют значительно ослабить излучения его элементов. Благодаря существующим достаточно строгим стандартам дозы рентгеновского излучения от современных мониторов и системных блоков не опасны для пользователей.

Документом, регламентирующим уровень электромагнитного излучения для ПЭВМ, является [46].

Согласно ему, напряжённости электрических и магнитных полей, энергетической нагрузки в течение рабочего дня не должны превышать значений, указанных в таблице 14.

Монитор ASUS VX239H соответствует стандарту [50], который устанавливает следующие предельные значения электромагнитного излучения:

- напряжённость электрического поля: в диапазоне 5Гц–2кГц не более 10 В/м, в диапазоне 2кГц–400кГц не более 1.0 В/м;
- напряжённость магнитного поля: в диапазоне 5Гц–2кГц не более 200 нТл, в диапазоне 2кГц–400кГц не более 25 нТл.

Таблица 14 — Предельные значения электромагнитного излучения.

Параметр	Предельные значения в диапазонах частот, МГц		
	от 0.06 до 3	св. 3 до 30	св. 30 до 300
$E_{\text{ПД}}, \text{В/м}$	500	300	80
$H_{\text{ПД}}, \text{А/м}$	50	—	—
$\text{ЭН}_{E_{\text{ПД}}}, (\text{В/м})^2 \cdot \text{ч}$	20000	7000	800
$\text{ЭН}_{H_{\text{ПД}}}, (\text{А/м})^2 \cdot \text{ч}$	200	—	—

Данные характеристики полностью соответствуют требованиям [46].

3.1.6 Электроопасность

В данном помещении используется оборудование, питающееся от сети переменного тока напряжением 220 В, частотой 50 Гц.

Согласно [51], помещение отдела разработки ИС относится к классу помещений без повышенной опасности поражения электрическим током: это сухое помещение с непроводящими полами, с нормальной температурой воздуха и влажностью, в нем отсутствует токопроводящая пыль.

Электрооборудование в помещении представлено мониторами и системными блоками ПЭВМ. Источником электрического поражения может быть металлический корпус системного блока при пробое изоляции, т.к. имеется напряжение 220 В, а в [52] допустимое напряжение и ток для аварийных режимов при времени воздействия более 1 секунды составляют 20 В и 6 мА.

3.1.7 Пожароопасность

В данном помещении имеются твердые горючие и трудногорючие вещества и материалы (книги, документы, деревянная мебель, оргтехника и т.д.), которые при взаимодействии с огнем будут гореть без взрыва. Также источником возгорания может быть электрическая проводка.

Согласно [53], данное помещение относится к классу Б и является пожароопасным.

3.1.8 Эргономические факторы

Требования к организации рабочих мест пользователей ПЭВМ изложены в [46].

Согласно [46], расстояние между рабочими столами с мониторами (в направлении тыла поверхности одного монитора и экрана другого монитора) должно быть не менее 2 м, а расстояние между боковыми поверхностями мониторов не менее 1.2 м.

Фактические значения (см. рисунок 13):

- расстояние между рабочими столами 2.1–2.5 м;
- расстояние между боковыми поверхностями мониторов 2.1-2.3 м.

Таким образом, размещение рабочих столов полностью соответствуют требованиям [46].

В помещении используется специальный стол — рабочая поверхность, изготовленная на заказ по выбранным заказчиком параметрам. Ее характеристики и нормативные значения указаны в таблице 15.

Таблица 15 — Характеристики используемого рабочего стола

Наименование параметра	Нормативное значение [46], мм	Фактическое значение, мм
Ширина рабочей поверхности	не менее 500	1200–2000
Глубина рабочей поверхности	не менее 800	800
Высота рабочей поверхности	не менее 725	800
Пространство для ног высотой	не менее 600	600
Глубина на уровне колен	не менее 450	450
Глубина на уровне вытянутых ног	не менее 650	650

Параметры стола полностью соответствуют требованиям [46].

В помещении используется офисное кресло БЮРОКРАТ Ch-G318AXN. Его характеристики и нормативные размеры указаны в таблице 16.

Таблица 16 — Характеристики используемого офисного кресла

Наименование параметра	Нормативное значение [46]	Фактическое значение
Ширина и глубина поверхности сиденья	не менее 400 мм	420 мм
Регулировка высоты поверхности сиденья	400–550 мм	440–570 мм
Регулировка углов наклона сиденья	вперед до 15° и назад до 5°	вперед до 15° и назад до 5°
Высота опорной поверхности спинки	300 ± 20 мм	310 мм
Ширина опорной поверхности спинки	не менее 380 мм	380 мм
Радиус кривизны горизонтальной плоскости опорной поверхности спинки	400 мм	400 мм
Угол наклона спинки в вертикальной плоскости	$\pm 30^\circ$	$\pm 30^\circ$
Регулировка расстояния спинки от переднего края сиденья	260–400 мм	260–450 мм
Стационарные или съёмные подлокотники	длина не менее 250 мм ширина 50–70 мм	длина 250 мм ширина 60 мм
Регулировка подлокотников по высоте над сиденьем	230 ± 30 мм	Нет
Регулировка внутреннего расстояния между подлокотниками	350–500 мм	420–500 мм

Параметры стула частично не соответствуют требованиям [46]: в данном рабо-

чем кресле отсутствует регулировка подлокотников по высоте над сидением.

3.1.9 Психофизиологические факторы

Факторами, оказывающими влияние на внимательность инженера и его производительность труда, в условиях его рабочего места являются:

- визуальные характеристики монитора (его яркость, контрастность, разрешение, частота обновления);
- напряженность работы;
- количество обрабатываемой информации — плотность воспринимаемых сигналов.

Используется ЖК монитор ASUS VX239H. Его фактические характеристики и нормативные значения [46] приведены в таблице 17.

Таблица 17 — Характеристики используемого ЖК монитора

Наименование фактора	Действительное значение	Нормативное значение [46]
Размер экрана по диагонали	58,42 см	не менее 31 см
Удалённость экрана	60 см	не менее 50 см
Частота обновления изображения	75 Гц	не менее 75 Гц
Контрастность	8 000 000:1	не менее 3:1
Яркость знака	250 кд/м ²	не менее 35 кд/м ²

Характеристики монитора полностью соответствуют требованиям [46].

Напряженность работы на основании данных таблицы классов условий труда по показателям напряженности трудового процесса [54] представлена в таблице 18.

Таблица 18 — Напряженность работы

Критерий	Характеристика	Напряженность трудового процесса
Содержание работы	Решение сложных задач по известным алгоритмам (работа по серии инструкций)	Напряженный труд 1 степени
Восприятие сигналов (информации и их оценка)	Восприятие сигналов с последующей комплексной оценкой взаимосвязанных параметров	Напряженный труд 2 степени
Степень сложности задания	Обработка, проверка и контроль за выполнением задания	Напряженность труда средней степени
Характер выполняемой работы	График с возможной корректировкой	То же
Длительность сосредоточенного наблюдения (в % от времени смены)	26–50%	—//—
Плотность сигнала за 1 час работы	176–300	Напряженный труд 1 степени
Число объектов одновременного наблюдения	6–10	Напряженность труда средней степени

Критерий	Характеристика	Напряженность трудового процесса
Размер объекта различения, мм, при длительном сосредоточенном наблюдении (% времени смены)	3–10 мм до 50% времени	То же
Степень ответственности	Ответственность за качество конечного результата	Напряженный труд 2 степени
Значимость ошибки	Влечет за собой дополнительные усилия в работе со стороны работника	Напряженность труда лёгкой степени
Продолжительность рабочего дня	8–9 часов	Напряженность труда средней степени

Среднее значение по данным критериям соответствует средней степени напряженности труда.

3.2 Мероприятия по обеспечению условий труда

Микроклимат в помещении в холодное время года обеспечивается с помощью системы центрального отопления. В летний же период времени в помещении не происходит регулирования температуры и влажности из-за отсутствия кондиционера. Рекомендуется установить, например, потолочный кондиционер Panasonic CS-A18BTP/CU-A18BVP5 с циркуляцией воздуха 840 м³/час.

Мероприятия по обеспечению требуемых условий по освещенности можно отнести к выбору ламп в источниках света. Часть цвет предметов освещенных люминес-

центными лампами может быть несколько искажён и быть неприятен человеку. Это, в свою очередь, вызывает усталость и напряженность глаз. Для предотвращения этого целесообразно использовать лампы с «трехполосным» и «пятиполосным» люминофором — веществом, способным преобразовывать поглощаемую им энергию в световое излучение. Это позволяет добиться более равномерного распределения излучения по видимому спектру, что приводит к более натуральному воспроизведению света. Данным параметрам соответствует лампа Philips Master TL-D De Luxe 36W/D65.

Уровень шума при использовании описанных ПЭВМ не превосходит норм, поэтому дополнительных мер для предотвращения излишнего шума не требуется. При покупке нового оборудования следует учитывать уровень шума от каждой новой единицы.

Обеспечение электробезопасности основано на применении устройств защитного отключения (УЗО). Данное устройство реагирует на ухудшение изоляции электропроводки: когда ток утечки повысится до предельной величины 30 мА, происходит отключение напряжения в течение 30 микросекунд. Целесообразно применять УЗО Legrand DX 06576.

Пожаробезопасность должна быть обеспечена при помощи обработки жидкостью от возгорания предметов мебели, инструктажа персонала на предмет мер предотвращения пожара или эвакуационных действий при пожаре, тщательной проверки оборудования на предмет повреждения проводов или оборудования.

3.3 Расчетная часть

3.3.1 Расчет уровня шума

Цель расчета — определить, соответствует ли фактический уровень шума в помещении нормативным значениям. Основными источниками шума в рассматриваемом помещении является аппаратура системных блоков ПЭВМ (кулер процессора).

Расчет производится в соответствии с [55].

План помещения с указанием рабочих мест и рабочей точки показан на рисунке 14

Будем считать, что оператор, находящийся в расчётной точке, располагается в зоне действия прямого звука. Тогда фактические значения уровней звукового давления



Рисунок 14 — План помещения с указанием рабочих мест и рабочей точки

в октановых полосах частот на расстоянии r от источника и суммарных значений от нескольких источников рассчитываются по формулам (27) и (28).

$$L_i = L_{W_i} - 20 \cdot \log r_i - 11; \quad (27)$$

$$L_{\Sigma} = 10 \cdot \log \sum_{i=1}^n 10^{0.1 \cdot L_i}, \quad (28)$$

где L_{W_i} — уровень звуковой мощности i -го источника.

Допустимые уровни звукового давления в расчётной точке представлены в соответствии с [46].

Расстояния от рабочих мест до рабочей точки:

$$r_1 = 1.8 \text{ м}, r_2 = 1.8 \text{ м}, r_3 = 3.1 \text{ м}, r_4 = 1.2 \text{ м}.$$

Уровни звукового давления для источников шума 1-4 (PM1-PM4) для уровня звуковой мощности кулера $L = 38$ дБ и среднегеометрической частоты 500 Гц:

$$L_1 = 38 - 20 \cdot \log 1.8 - 11 = 31.9 \text{ дБ};$$

$$L_2 = 38 - 20 \cdot \log 1.8 - 11 = 31.9 \text{ дБ};$$

$$L_3 = 38 - 20 \cdot \log 3.1 - 11 = 27.2 \text{ дБ};$$

$$L_4 = 38 - 20 \cdot \log 1.2 - 11 = 38.5 \text{ дБ}.$$

В этом случае уровень звукового давления от 4 источников:

$$L_{\Sigma} = 10 \cdot \log(10^{0.1 \cdot 31.9} + 10^{0.1 \cdot 31.9} + 10^{0.1 \cdot 27.2} + 10^{0.1 \cdot 38.5}) = 38.5 \text{ дБ}.$$

Уровни звукового давления для источников шума 1-4 (PM1-PM4) для уровня звуковой мощности кулера $L = 37$ дБ и среднегеометрической частоты 1000 Гц:

$$L_1 = 37 - 20 \cdot \log 1.8 - 11 = 30.9 \text{ дБ};$$

$$L_2 = 37 - 20 \cdot \log 1.8 - 11 = 30.9 \text{ дБ};$$

$$L_3 = 37 - 20 \cdot \log 3.1 - 11 = 26.2 \text{ дБ};$$

$$L_4 = 37 - 20 \cdot \log 1.2 - 11 = 37.5 \text{ дБ}.$$

В этом случае уровень звукового давления от 4 источников:

$$L_{\Sigma} = 10 \cdot \log(10^{0.1 \cdot 30.9} + 10^{0.1 \cdot 30.9} + 10^{0.1 \cdot 26.2} + 10^{0.1 \cdot 37.5}) = 37.5 \text{ дБ}.$$

Уровни звукового давления источников шума для остальных частот рассчитываются аналогично. Результаты расчётов сведены в таблицу 19.

Уровень шума в расчетной точке не превышает допустимые значения, таким образом, не требуются никакие мероприятия по снижению шума в помещении.

3.4 Вывод

В разделе «Охрана труда и окружающей среды» был проведен анализ условий труда инженера-программиста по следующим факторам: санитарно-гигиеническим, эргономическим, психофизическим; была проведена оценка помещения по электроопасности и пожароопасности. Также были предложены мероприятия по обеспечению требований предъявляемых к эргономическим характеристикам рабочего места. В расчетной части был осуществлен расчет уровня шума в помещении, в результате которого были получены значения, не превышающие допустимые.

По всем перечисленным факторам было выявлено соответствие нормам и требованиям ГОСТов, СанПиНов и СНиПу.

Таблица 19 — Сводная таблица значений уровней шума

Среднегеометрическая частота, Гц	63	125	250	500	1000	2000	4000	8000
Уровень звуковой мощности кулера блока питания, дБ	35	37,5	37,5	38	37	36,5	36,5	35,5
Уровень звукового давления от источника шума 1 (PM1) при $r = 1.8$ м, дБ	28.9	31.4	31.4	31.9	30.9	30.4	30.4	29.4
Уровень звукового давления от источника шума 2 (PM2) при $r = 1.8$ м, дБ	28.9	31.4	31.4	31.9	30.9	30.4	30.4	29.4
Уровень звукового давления от источника шума 3 (PM3) при $r = 3.1$ м, дБ	24.2	26.7	26.7	27.2	26.2	25.7	25.7	24.7
Уровень звукового давления от источника шума 4 (PM4) при $r = 1.2$ м, дБ	32.4	34.9	34.9	35.4	34.4	33.9	33.9	32.9
Уровень звукового давления от 4 источников шума, дБ	35.5	38	38	38.5	37.5	37	37	36
Допустимый уровень звукового давления в расчетной точке (РТ), дБ	71	61	54	49	45	42	40	38

Заключение

здесь должно быть заключение

Список использованных источников

1. Jennions I. K. Integrated Vehicle Health Management: Perspectives on an Emerging Field. SAE International, 2011. 188 p.
2. Yairi T., Kato Y., Hori K. Fault Detection by Mining Association Rules from House-keeping Data // Proceedings of the 6th International Symposium on Artificial Intelligence and Robotics & Automation in Space (i-SAIRAS 2001). Quebec, Canada: Canadian Space Agency, 2001.
3. Fayyad U., Piatetsky-Shapiro G., Smyth P. From Data Mining to Knowledge Discovery in Databases // AI Magazine. 1996. Vol. 17, no. 3. P. 37–54.
4. Chandola V., Banerjee A., Kumar V. Anomaly Detection: A Survey // ACM Computing Surveys. 2009. Vol. 41, no. 3. P. 15:1–15:58.
5. Hawkins D. M. Identification of Outliers. New York, NY, USA: Chapman and Hall, 1980. 188 p.
6. Деревяненко В.В. Применение Data Mining в космических приложениях // Исследования Наукограда. 2012. № 1. С. 47–51.
7. General Purpose Data-Driven System Monitoring for Space Operations / D. Iverson, R. Martin, M. Schwabacher et al. // AIAA Infotech@Aerospace Conference. 2009.
8. Iverson D. Inductive System Health Monitoring // Proceedings of The 2004 International Conference on Artificial Intelligence (IC-AI'04). Las Vegas, NV, USA: CSREA Press, 2004.
9. Comparison of Unsupervised Anomaly Detection Methods for Systems Health Management Using Space Shuttle Main Engine Data / R. Martin, M. Shwabacher, N. Oza et al. // Proceedings of the 54th Joint Army-Navy-NASA-Air Force Propulsion Meeting. Denver, CO, USA: 2007.
10. Columbia Accident Investigation Board: Report: Washington, D.C., USA: NASA, 2003.

11. Iverson D. System Health Monitoring for Space Mission Operations // Proceedings of The SpaceOps 2008 Conference (European Space Agency, AIAA). Heidelberg, Germany: 2008.
12. Гиродин. Материал из Википедии — свободной энциклопедии. URL: <http://ru.wikipedia.org/wiki/Гиродин> (дата обращения: 20.11.2013).
13. Fukushima Y. Telemetry Data Mining with SVM for Satellite Monitoring // Modern Telemetry / Ed. by O. Krejcar. InTech, 2011. P. 95–114.
14. Schwabacher M. Machine Learning for Rocket Propulsion Health Monitoring // Proceedings of the SAE World Aerospace Congress. Vol. 114-1. Dallas, TX, USA: Society of Automotive Engineers, 2005. P. 1192–1197.
15. Bay S. D., Schwabacher M. Mining Distance-Based Outliers in Near Linear Time with Randomization and a Simple Pruning Rule // Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '03). New York, NY, USA: ACM Press, 2003. P. 29–38.
16. Knorr E. M., Ng R. T. Algorithms for Mining Distance-Based Outliers in Large Datasets // Proceedings of the 24rd International Conference on Very Large Data Bases (VLDB '98). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998. P. 392–403.
17. Tao Y., Xiao X., Zhou S. Mining Distance-Based Outliers from Large Databases in Any Metric Space // Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '06). Philadelphia, PA, USA: ACM Press, 2006. P. 394–403.
18. Angiulli F., Fasseti F. Very Efficient Mining of Distance-Based Outliers // Proceedings of the sixteenth ACM conference on Conference on information and knowledge management (CIKM '07). New York, NY, USA: ACM Press, 2007. P. 791–800.
19. Ramaswamy S., Rastogi R., Shim K. Efficient Algorithms for Mining Outliers from Large Data Sets // Proceedings of the 2000 ACM SIGMOD international conference

- on Management of data (SIGMOD '00). New York, NY, USA: ACM Press, 2000. P. 427–438.
20. RuleQuest Research. Checking Data Quality with GritBot. 2005. URL: <http://http://www.rulequest.com/gritbot-info.html> (дата обращения: 19.11.2013).
 21. Eberle W., Holder L., Cook D. Identifying Threats Using Graph-based Anomaly Detection // Machine Learning in Cyber Trust: Security, Privacy, and Reliability / Ed. by J. J. Tsai, P. S. Yu. Springer, 2009. P. 97–98.
 22. Quinlan J. R. C4.5: Programs for Machine learning. San Mateo, CA, USA: Morgan Kaufmann Publishers Inc., 1993. 302 p.
 23. Королев В.Ю. ЕМ-алгоритм, его модификации и их применение к задаче разделения смесей вероятностных распределений. Теоретический обзор. М.: ИПИ РАН, 2007. С. 102.
 24. Martin R. A. Unsupervised Anomaly Detection and Diagnosis for Liquid Rocket Engine Propulsion // Proceedings of the IEEE Aerospace Conference. Big Sky, MT, USA: 2007.
 25. Pearl J. Causality: Models, Reasoning, and Inference. 2nd edition. Cambridge, UK: Cambridge University Press, 2009. 464 p.
 26. Hill D., Minsker B., Amir E. Real-time Bayesian Anomaly Detection for Environmental Sensor Data // Proceedings of the 32nd Conference of the International Association of Hydraulic Engineering & Research (IAHR '07). 2007.
 27. Лифшиц Юрий. Алгоритмы для интернета. Курс лекций. СПб.: ПОМИ РАН — СПбГУ ИТМО, 2006.
 28. Воронцов К.В. Машинное обучение. Курс лекций. М.: Школа анализа данных Яндекса, 2009.
 29. Vlasveld R. Introduction to One-class Support Vector Machines. 2013. URL: <http://rvlasveld.github.io/blog/2013/07/12/introduction-to-one-class-support-vector-machines/> (дата обращения: 25.11.2013).

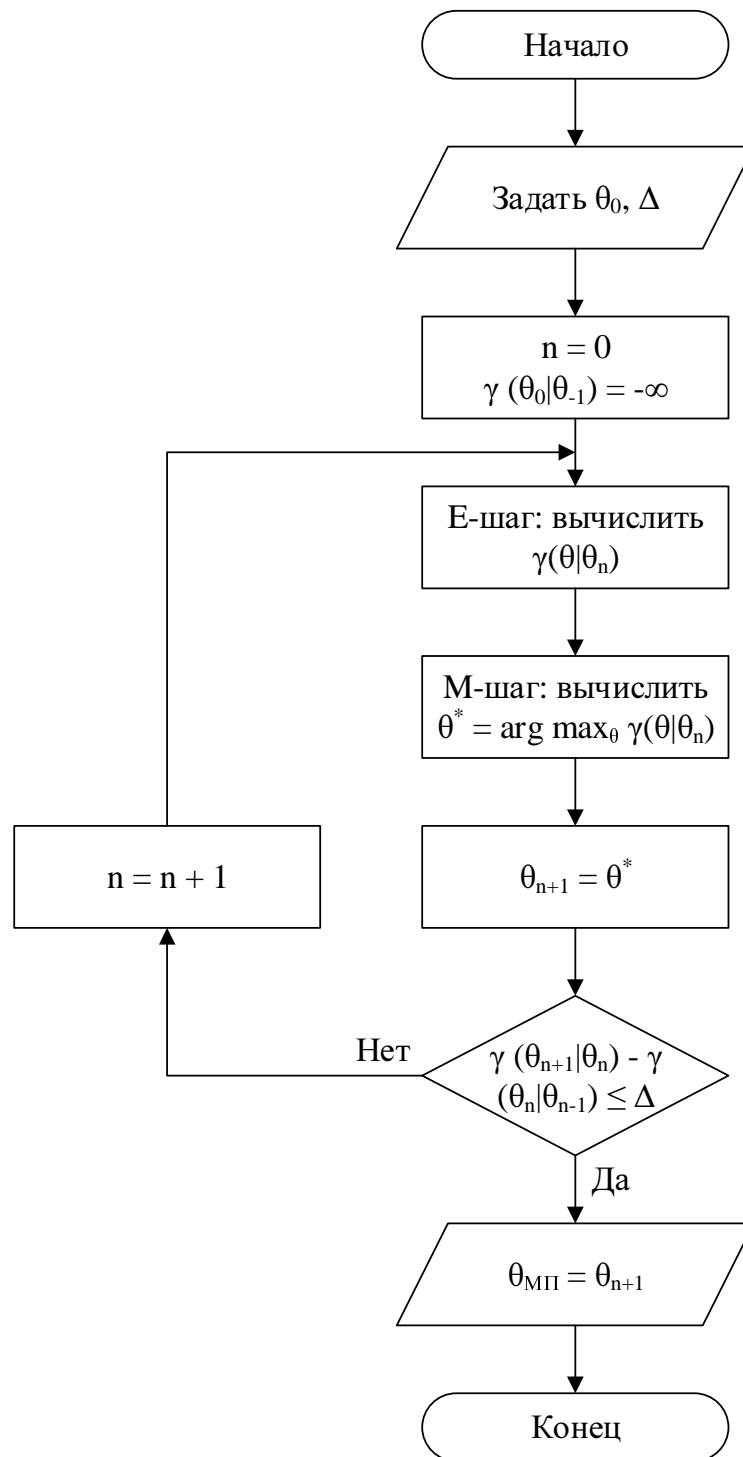
30. Bradley P. S., Fayyad U. M. Refining Initial Points of K-means Clustering // Proceedings of the International Conference on Machine Learning (ICML-98). San Mateo, CA, USA: Morgan Kaufmann Publishers Inc., 1998. P. 91–99.
31. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise / M. Ester, H.-P. Kriegel, J. Sander et al. // Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96). Menlo Park, CA, USA: AAAI Press, 1996. P. 226–231.
32. Пайсон Михаил. ООП с примерами. Курс лекций. Барнаул: АлтГУ, 2010.
33. Роберт Мартин Мика Мартин. Принципы, паттерны и методики гибкой разработки на языке C#. СПб.: Символ-Плюс, 2011. 768 с.
34. TIOBE Software. TIOBE Programming Community Index (2013). URL: <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html> (дата обращения: 05.12.2013).
35. C++. Материал из Википедии — свободной энциклопедии. URL: <http://ru.wikipedia.org/wiki/C++> (дата обращения: 25.10.2013).
36. C#. Материал из Википедии — свободной энциклопедии. URL: http://ru.wikipedia.org/wiki/C_sharp (дата обращения: 25.10.2013).
37. D (язык программирования). Материал из Википедии — свободной энциклопедии. URL: [http://ru.wikipedia.org/wiki/D_\(язык_программирования\)](http://ru.wikipedia.org/wiki/D_(язык_программирования)) (дата обращения: 25.10.2013).
38. Java. Материал из Википедии — свободной энциклопедии. URL: <http://ru.wikipedia.org/wiki/Java> (дата обращения: 25.10.2013).
39. Python. Материал из Википедии — свободной энциклопедии. URL: <http://ru.wikipedia.org/wiki/Python> (дата обращения: 25.10.2013).
40. Ruby. Материал из Википедии — свободной энциклопедии. URL: <http://ru.wikipedia.org/wiki/Ruby> (дата обращения: 25.10.2013).

41. C++. Материал из Википедии — свободной энциклопедии.
URL: http://ru.wikipedia.org/wiki/Visual_Basic_.NET (дата обращения: 25.10.2013).
42. Design Patterns: Elements of Reusable Object-Oriented Software / E. Gamma, R. Helm, R. Johnson et al. Boston, MA, USA: Addison-Wesley, 1994. 416 p.
43. Knuth D. E. The Art of Computer Programming. Volume 2, Seminumerical Algorithms. MA, USA: Addison-Wesley, 1969. P. 124–125.
44. Introduction to Algorithms / T. H. Cormen, C. E. Leiserson, R. L. Rivest et al. 3rd edition. Cambridge, MA, USA: The MIT Press, 2009. P. 151–169.
45. Экономическое обоснование дипломных проектов (работ) по приборо- и радио-приборостроению / Панагушин В.П., Ковалева Т.С., Малютина О.А. [и др.]; под ред. Панагушина В.П. М.: ИВАКО Аналитик, 2008. 44 с.
46. СанПиН 2.2.2/2.4.1340–03 (с изменениями от 25 апреля 2007 г.) «Гигиенические требования к персональным электронно-вычислительным машинам и организации работы». М.: Минздрав России, 2003. 32 с.
47. ГОСТ 12.1.005–88 ССБТ «Общие санитарно-гигиенические требования к воздуху рабочей зоны». М.: ИПК Издательство стандартов, 2002. 71 с.
48. СНиП 23-05-95 «Естественное и искусственное освещение». М.: Минстрой России, 1995. 59 с.
49. ГОСТ 12.1.003–83 «Шум. Общие требования безопасности». М.: ИПК Издательство стандартов, 1983. 13 с.
50. The Swedish Confederation Of Professional Employees. TCO '03. 2005.
URL: <http://www.dtic.ua.es/ibis/recursos/normativas/TCO03CRT.pdf> (дата обращения: 11.11.2013).
51. Правила устройства электроустановок. Издание седьмое. М.: Минэнерго России, 2002. 222 с.

52. ГОСТ 12.1.038–82 «Электробезопасность. Предельно допустимые значения напряжений прикосновения и токов». М.: ИПК Издательство стандартов, 1988. 5 с.
53. ГОСТ 12.1.004–91 «Пожарная безопасность. Общие требования». М.: ИПК Издательство стандартов, 1996. 83 с.
54. Р 2.2.2006–05 «Руководство по гигиенической оценке факторов рабочей среды и трудового процесса. Критерии и классификация условий труда». М.: Минздрав России, 2005. 156 с.
55. Бобков Н.И., Голованова Т.В. Охрана труда на ВЦ. Методическими указаниями к дипломному проектированию. М.: МАИ, 1991. 16 с.
56. ISSLive! ADCO Control Moment Gyroscopes Display.
URL: <http://spacestationlive.nasa.gov/displays/adcoDisplay3.html> (дата обращения: 25.11.2013).
57. Space Station Control Moment Gyroscope Lessons Learned / C. Gurrisi, R. Seidel, S. Dickerson et al. // Proceedings of the 40th Aerospace Mechanisms Symposium (AMS '10). 2010. P. 161–175.
58. Burt R. R., Loffi R. W. Failure Analysis of International Space Station Control Moment Gyro // Proceedings of the 10th European Space Mechanisms and Tribology Symposium (ESMATS '03). 2003. P. 13–25.
59. ГОСТ 2.105–95 «Единая система конструкторской документации. Общие требования к текстовым документам». М.: Стандартинформ, 2005. 29 с.
60. ГОСТ 7.1–2003 «Библиографическая запись. Библиографическое описание. Общие требования и правила составления». М.: ИПК Издательство стандартов, 2004. 166 с.
61. ГОСТ 7.32–2001 «Отчет о научно-исследовательской работе. Структура и правила оформления». М.: ИПК Издательство стандартов, 2001. 22 с.

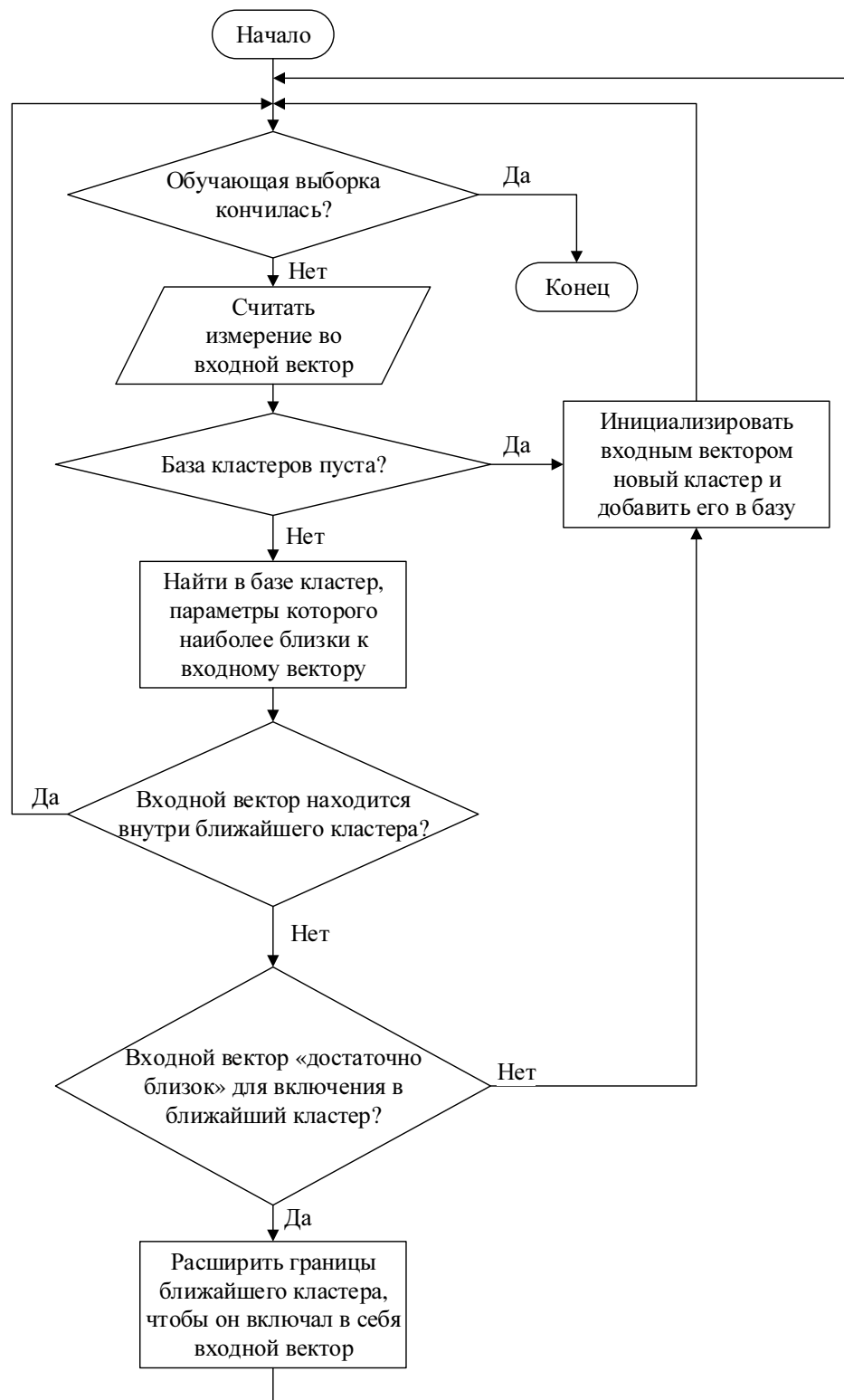
ПРИЛОЖЕНИЕ А

Блок-схема ЕМ-алгоритма для GMM



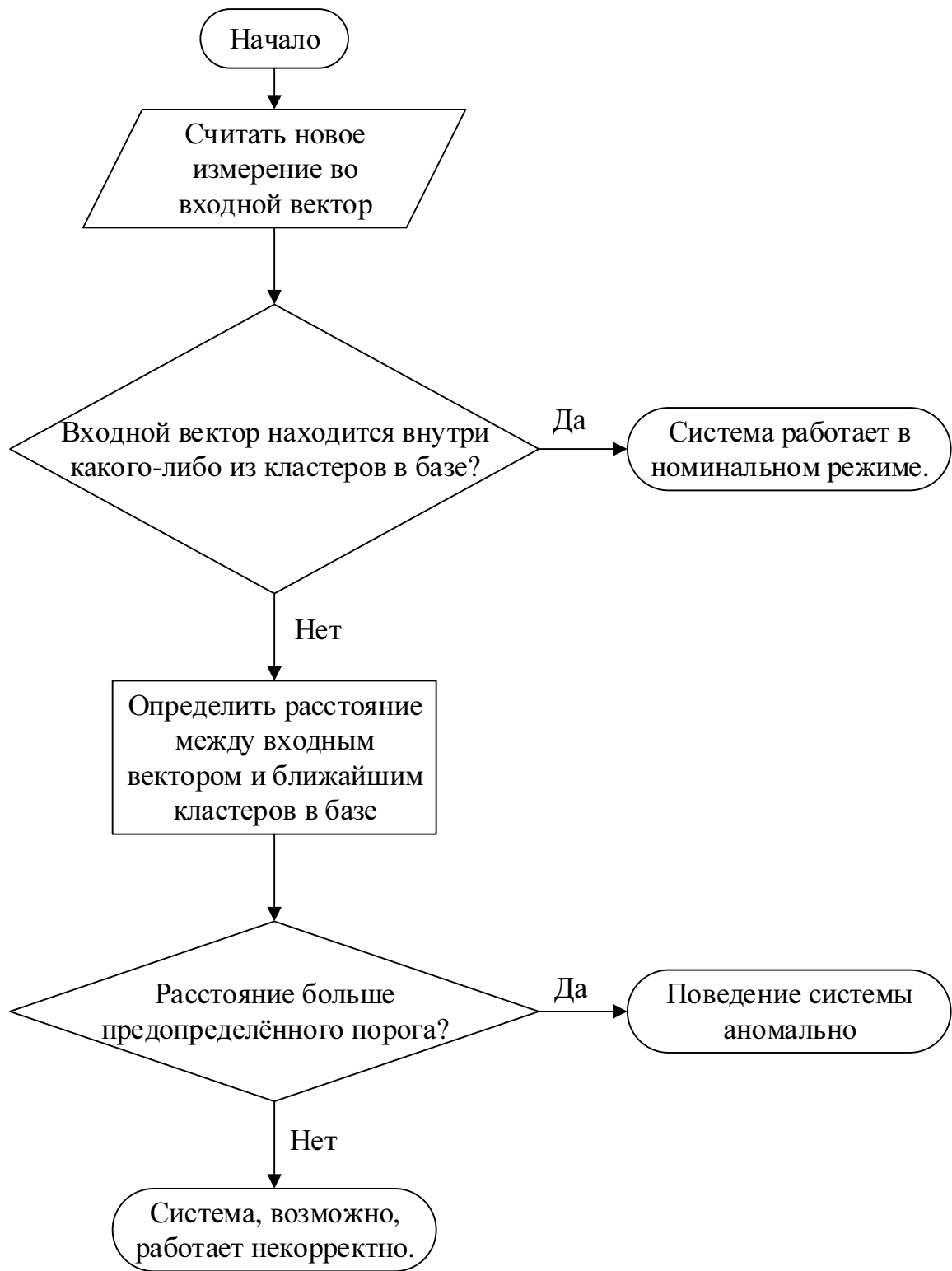
ПРИЛОЖЕНИЕ Б

Блок-схема процесса обучения IMS



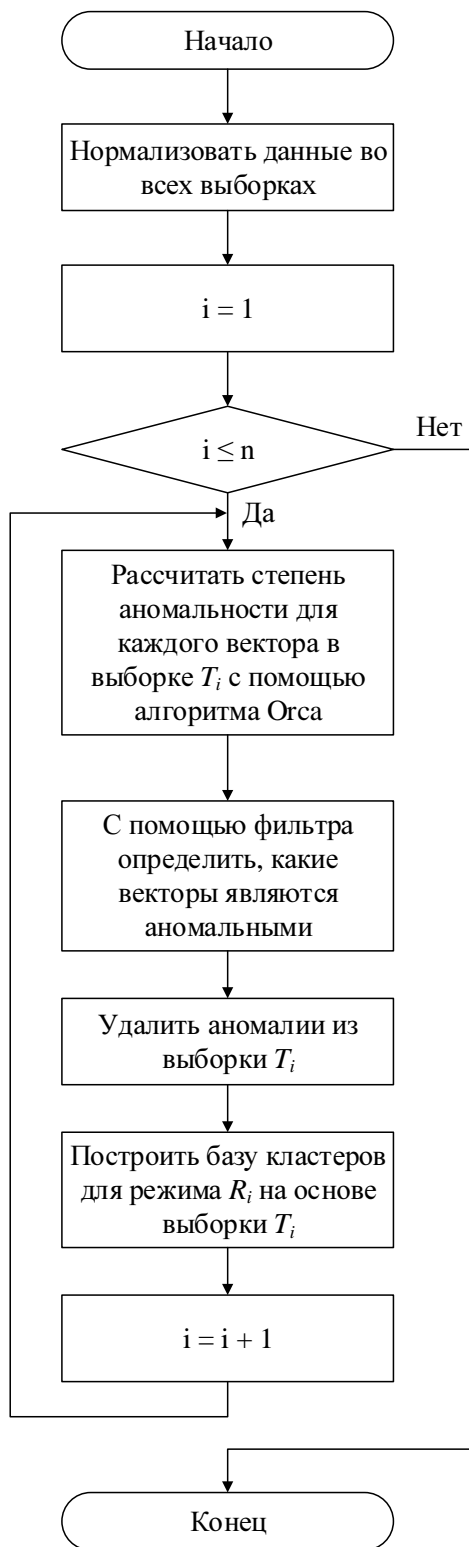
ПРИЛОЖЕНИЕ В

Блок-схема процесса мониторинга IMS



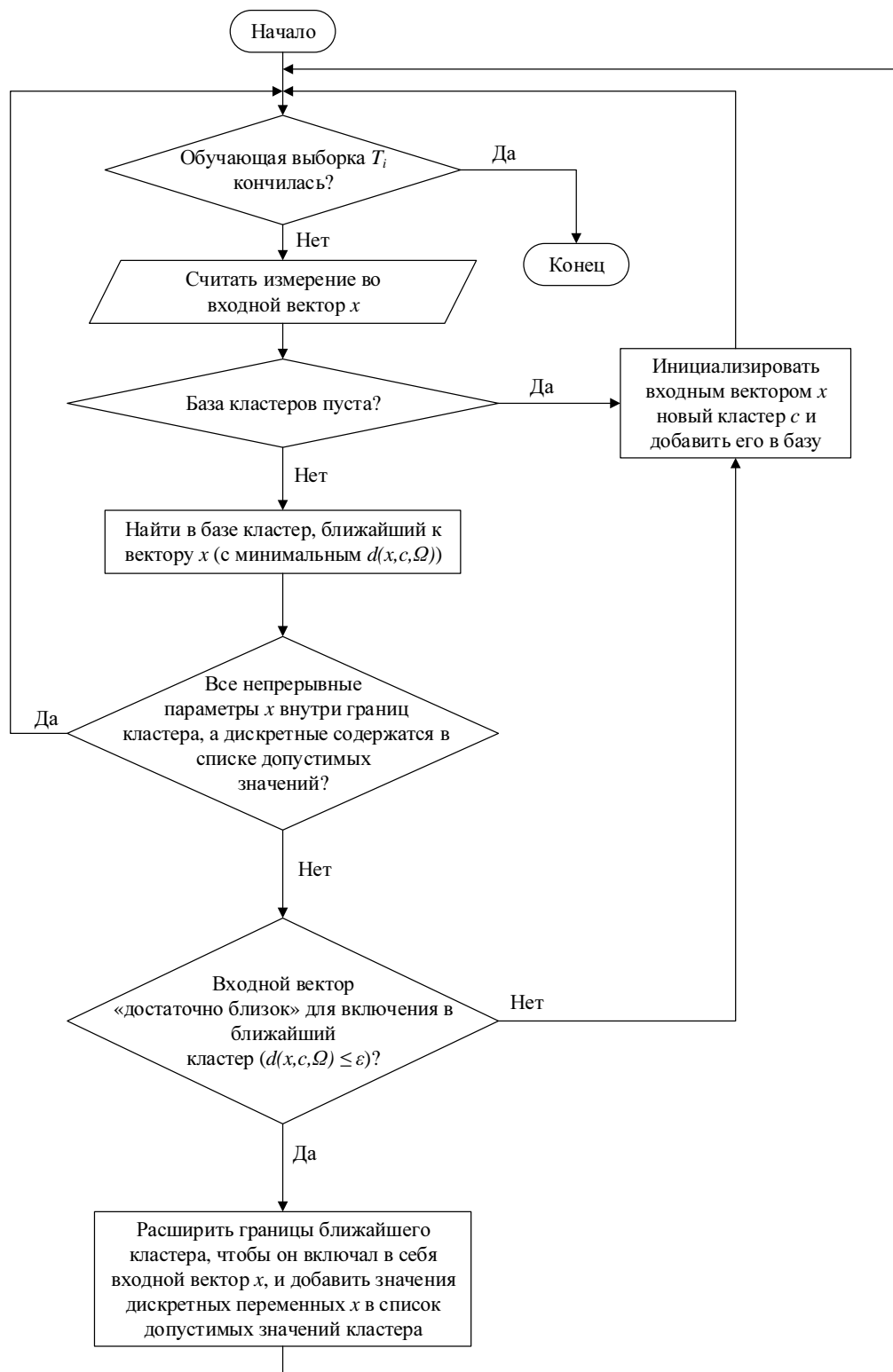
ПРИЛОЖЕНИЕ Г

Блок-схема процесса обучения разработанного метода



ПРИЛОЖЕНИЕ Д

Блок-схема процесса создания базы кластеров для каждого режима работы системы (для разработанного метода)



ПРИЛОЖЕНИЕ Е

Блок-схема процесса мониторинга для разработанного метода

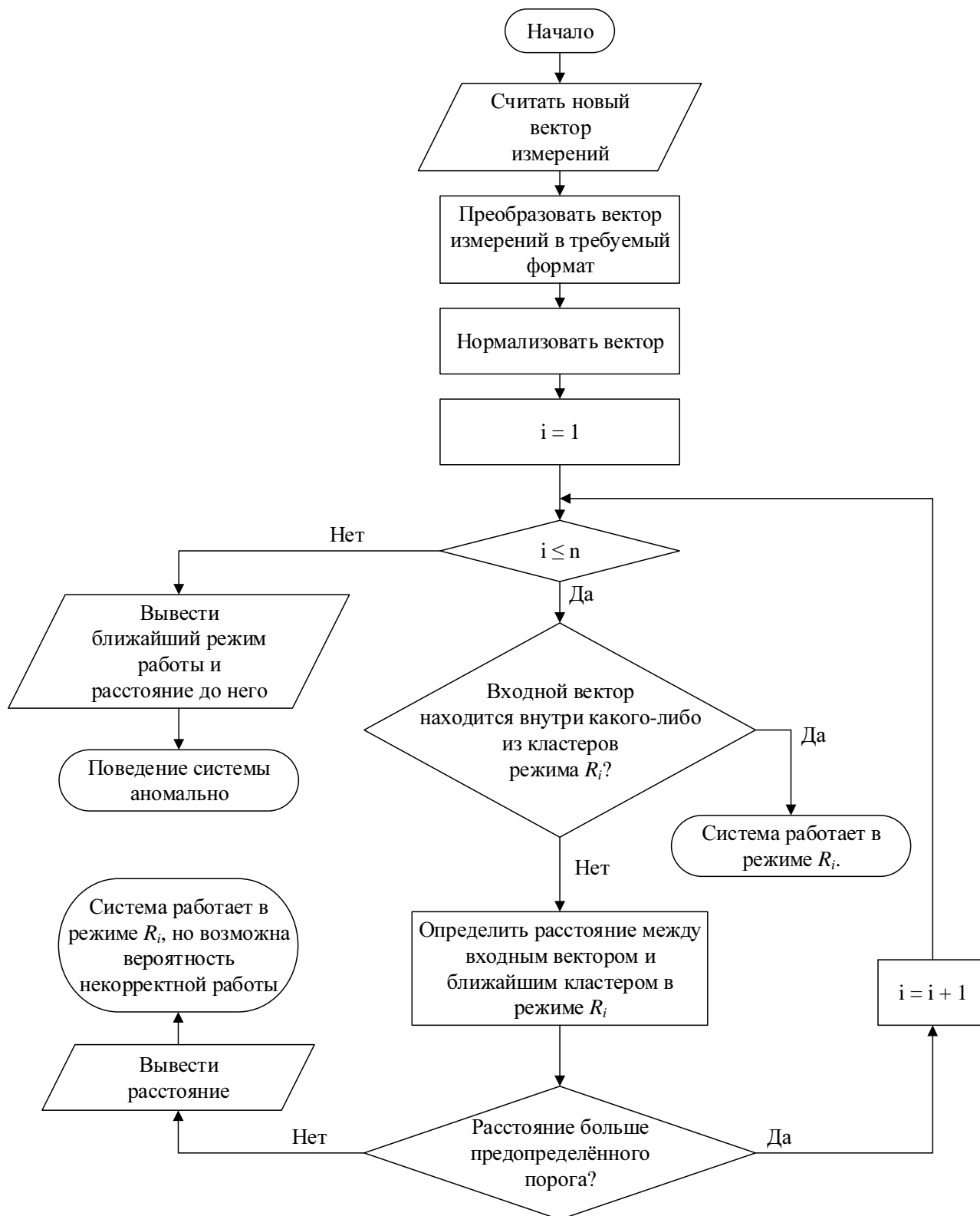
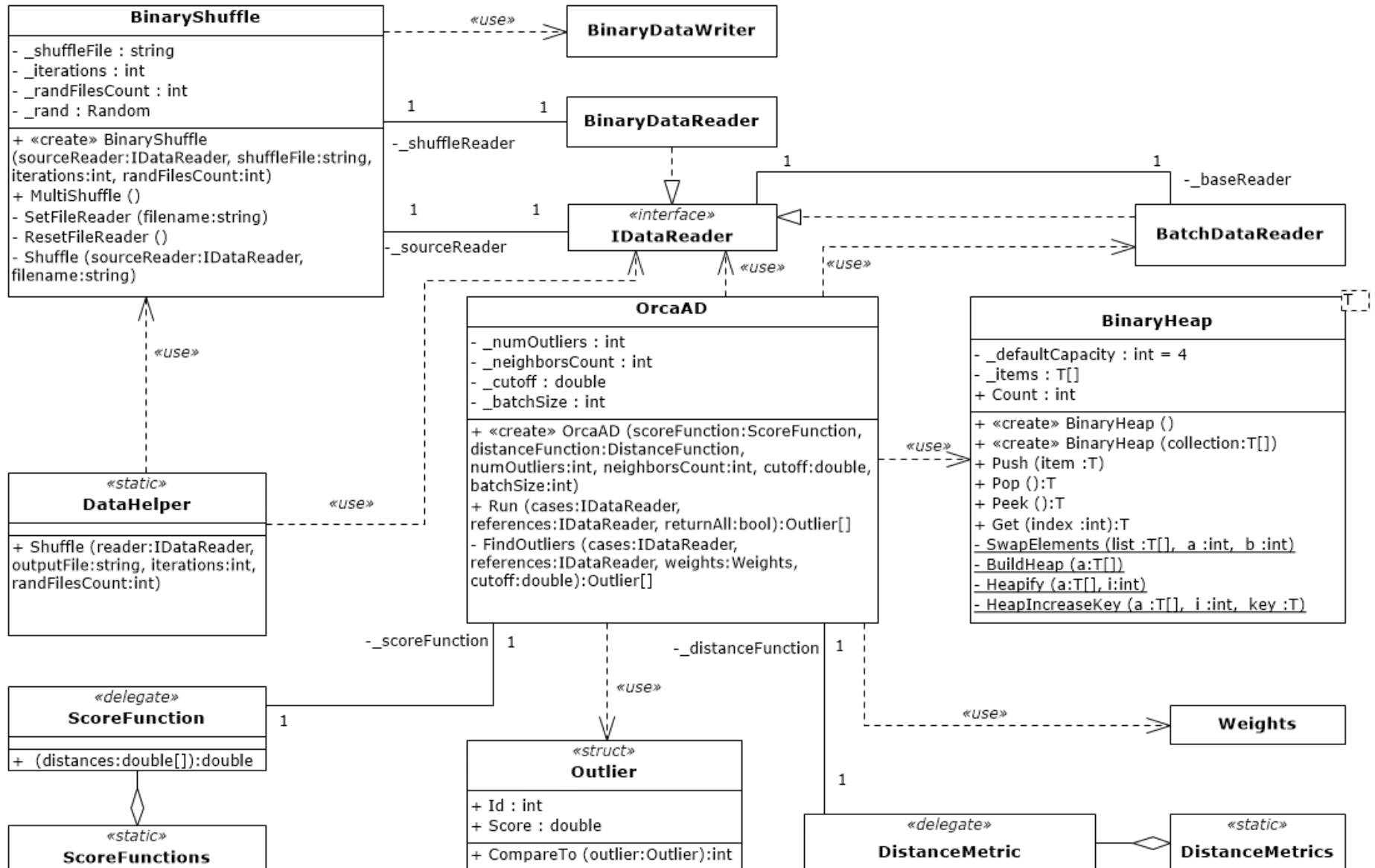


Диаграмма классов (операции с данными)



ПРИЛОЖЕНИЕ 3

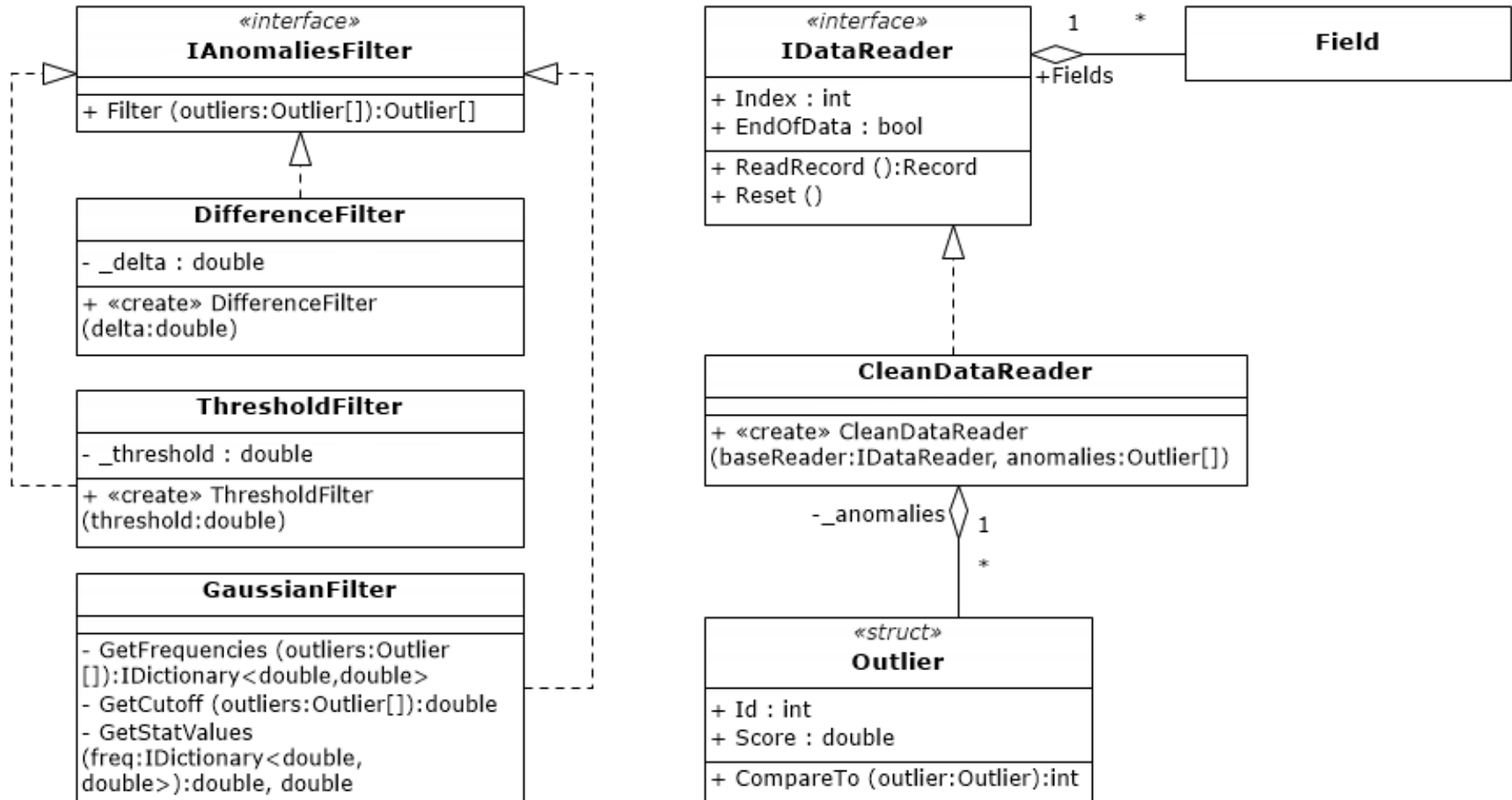
Диаграмма классов (обнаружение аномалий в обучающей выборке)



ПРИЛОЖЕНИЕ И

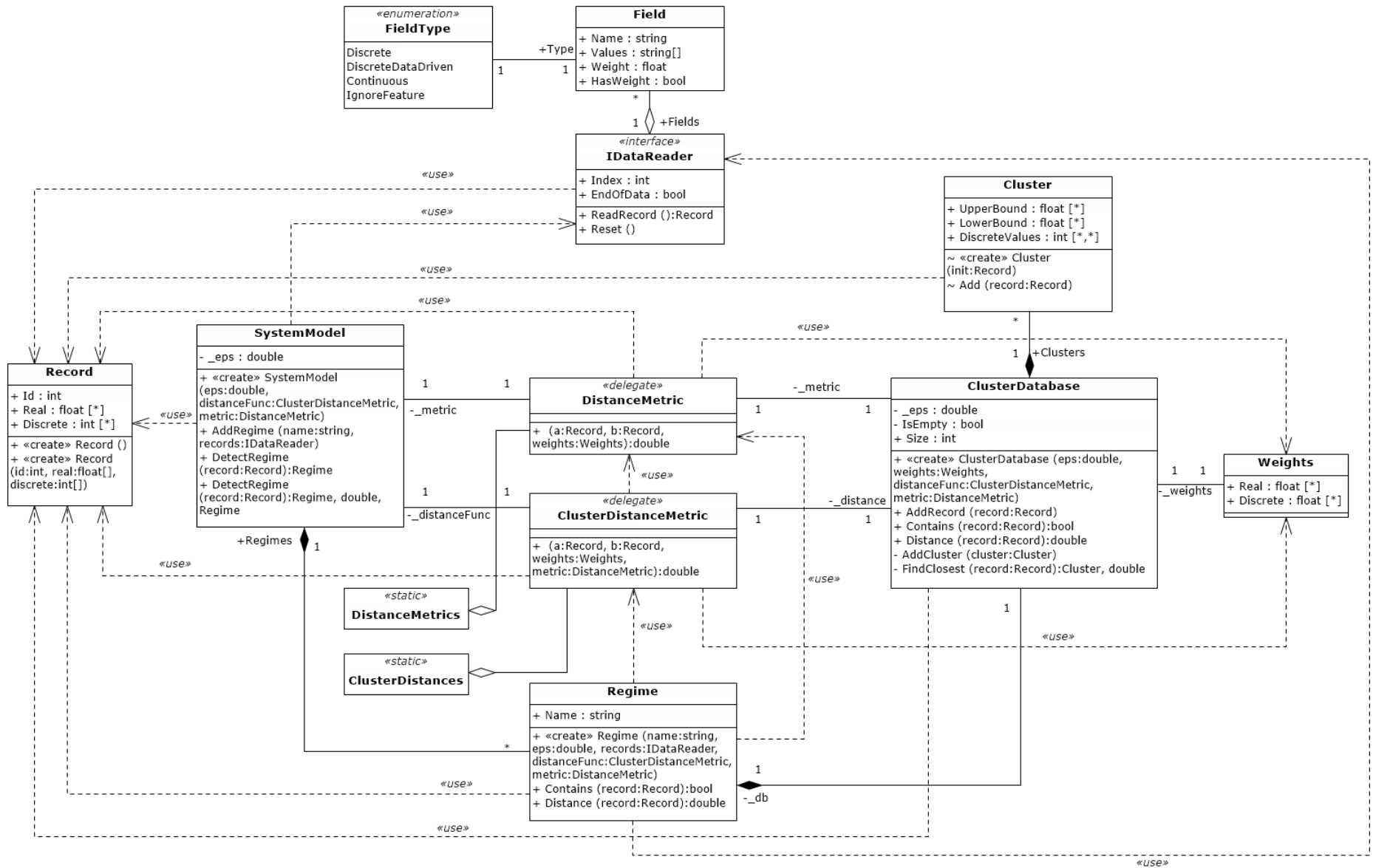
Диаграмма классов (фильтрация аномалий)

100



ПРИЛОЖЕНИЕ К

Диаграмма классов (разработанный метод)



ПРИЛОЖЕНИЕ Л

Параметры запуска программы

Таблица Л.1 — Параметры запуска программы

Формат	Описание	Обязательный
имя_файла	Файл с описанием формата параметров векторов.	Да
-г имя_файла1... имя_файлаN	Файлы с обучающими выборками для номинальных режимов работы системы	Да
-а имя_файла1... имя_файлаN	Файлы с обучающими выборками для внештатных (аномальных) режимов работы системы	Нет
-f тип_фильтра [параметр_фильтра]	Тип фильтра, использующегося при удалении аномалий из обучающих выборок. Допустимые значения: gaussian (гауссовый), difference (разностный), threshold (пороговый). Для разностного фильтра необходимо указать максимальную разность, для порогового - величину порога. Значение по умолчанию: gaussian.	Нет
-о имя_файла	Имя файла для сохранения вывода программы.	Нет

Формат	Описание	Обязательный
-n вид_нормализации	Вид нормализации (масштабирования) векторов с данными. Допустимые значения: minimax (минимаксная), standard (по стандартному отклонению). Значение по умолчанию: minimax.	Нет
-m тип_метрики	Тип метрики пространства. Допустимые значения: euclid (евклидова), sqeuclid (квадрат евклидовой). Значение по умолчанию: euclid.	Нет
-d тип_расстояния	Тип функции для измерения расстояния между кластером и вектором. Допустимые значения: kmeans (до центра кластера), nearest (до ближайшей точки кластера). Значение по умолчанию: nearest.	Нет
-v строка	Строка, замещающая отсутствующие значения в векторах в файлах с обучающими выборками. Значение по умолчанию: «?».	Нет

ПРИЛОЖЕНИЕ М

Исходный код разработанного программного обеспечения

```
/* **** */
* Options.cs
* **** */

using CommandLine;
using CommandLine.Text;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis.App
{
    enum Filter
    {
        Gaussian, Difference, Threshold
    }

    enum Normalization
    {
        Minimax, Standard
    }

    enum Metric
    {
        Euclid, SqrEuclid
    }

    enum ClusterDistanceType
    {
        KMeans, Nearest
    }

    class Options
    {
        [ValueOption(0)]
        public string FieldsDescription { get; set; }

        [OptionArray('a', "anregimes", HelpText = "Files_with_anomalie_regimes_samples."
        )]
        public string[] Samples { get; set; }
    }
}
```

```

[OptionArray('r', "nomregimes", HelpText="Files with nominal regimes samples (
    will be cleaned from anomalies).", Required = true)]
public string[] NominalSamples { get; set; }

[Option('f', "filter", DefaultValue = Filter.Gaussian,
    HelpText = "Anomalies filter type and value. Appropriate values: gaussian,
    difference (maxdiff-value), threshold (threshold-value).")]
public Filter Filter { get; set; }

[Option('n', "normalization", DefaultValue = Normalization.Minimax,
    HelpText = "Normalization type. Appropriate values: minimax, standard.")]
public Normalization Normalization { get; set; }

[Option('m', "metric", DefaultValue = Metric.Euclid,
    HelpText = "Distance metric. Appropriate values: euclid, sqreucld.")]
public Metric Metric { get; set; }

[Option('d', "cdist", DefaultValue = ClusterDistanceType.Nearest,
    HelpText = "Cluster distance function. Appropriate values: kmeans, nearest."
    )]
public ClusterDistanceType ClusterDistanceType { get; set; }

[Option('v', "novalue", DefaultValue = "?",
    HelpText = "Non-existing values replacement string.")]
public string NoValueReplacement { get; set; }

[Option('o', "output")]
public string OutputFile { get; set; }

[ValueList(typeof(List<string>))]
public List<string> Args { get; set; }

[HelpOption]
public string GetUsage()
{
    var help = new HelpText
    {
        Heading = new HeadingInfo("Thesis: Intergrated System Health Management
            based on Data Mining techniques."),
        Copyright = new CopyrightInfo("Vladimir Panchenko, 03-617, Moscow
            Aviation Institute", 2013),
        AdditionalNewLineAfterOption = true,
        AddDashesToOption = true
    };
    help.AddPreOptionsLine("Usage: thesis.app_fields.txt -r nominal1.txt

```

```

        nominal2.txt[-a_regime1.txt_regimeN.txt][-f_threshold0.5][-d_kmeans]
        [-m_sqreucld][-n_standard][-v_N/A]);
    help.AddPostOptionsLine("Normalization_stats_are_calculated_based_on_first_
        nominal_regime_sample.\n");
    help.AddOptions(this);
    return help;
}

}

class FilterOptions
{
    //[Option('v', "fvalue", HelpText = "Maximum difference for difference filter |
    Threshold value for threshold filter")]
    [ValueOption(0)]
    public double? Value { get; set; }
}

}

/*****
* Program.cs
*****/

using System;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Thesis.DataCleansing;
using Thesis.DDMS;
using Thesis.Orca;

namespace Thesis.App
{
    class Program
    {
        static string _fieldsFile;
        static IAnomaliesFilter _filter;
        static IRecordParser<string> _parser;
        static IScaling _scaling;
        static SystemModel _model;
        static DistanceMetric _metric;
        static ClusterDistanceMetric _clusterDistMetric;
        static IList<Field> _fields;

```

```

static bool _writeToFile;

static string GetBinName(string name)
{
    return String.Concat(name, ".bin");
}

static void ShowUsage()
{
    Console.WriteLine(new Options().GetUsage());
    Console.WriteLine();
    Environment.Exit(-1);
}

static void AddNominalSample(string filename)
{
    if (!File.Exists(filename))
        throw new ApplicationException("File not found.");

    using (IDataReader reader = new PlainTextReader(filename, _fieldsFile,
        _parser)) // read input data
    using (IDataReader binReader = new BinaryDataReader(reader, GetBinName(
        filename))) // convert input data to binary format
    using (IDataReader scaleReader = new ScaleDataReader(binReader, _scaling))
        // scale input data
    {
        if (_fields == null)
            _fields = reader.Fields;

        string shuffleFile = String.Concat(filename, ".shuffle");
        scaleReader.Shuffle(shuffleFile);
        IEnumerable<Outlier> outliers = Enumerable.Empty<Outlier>();

        string regimeName = Path.GetFileNameWithoutExtension(filename);

        using (IDataReader cases = new BinaryDataReader(shuffleFile))
        using (IDataReader references = new BinaryDataReader(shuffleFile))
        {
            var orca = new OrcaAD(DistanceMetrics.Euclid, neighborsCount: 10);
            outliers = orca.Run(cases, references, true);
        }

        File.Delete(shuffleFile);

        var anomalies = _filter.Filter(outliers);
    }
}

```

```

        Console.WriteLine("\n%%%%%%%%_{0}_%%%%%%%%", regimeName);
        Console.WriteLine("Anomalies:");
        foreach (var anomaly in anomalies)
            Console.WriteLine("_Id=_{0},_Score=_{1}", anomaly.Id, anomaly.
                Score);

        Console.WriteLine("%%%%%%%%%\n");

        using (IDataReader cleanReader = new CleanDataReader(scaleReader,
            anomalies)) // clean input data from anomalies
        {
            _model.AddRegime(regimeName, cleanReader);
        }
    }
}

static void AddSample(string filename)
{
    if (!File.Exists(filename))
        throw new ApplicationException("File_not_found.");

    string regimeName = Path.GetFileNameWithoutExtension(filename);

    using (IDataReader tempReader = new PlainTextReader(filename, _fieldsFile,
        _parser))
    using (IDataReader tempBinReader = new BinaryDataReader(tempReader,
        GetBinName(filename)))
    {
        using (IDataReader tempScaleReader = new ScaleDataReader(tempBinReader,
            _scaling))
        {
            _model.AddRegime(regimeName, tempScaleReader);
        }
    }
}

static void Main(string[] args)
{
    Contract.ContractFailed += (s, e) =>
    {
        Console.WriteLine("Something_went_wrong._Please_contact_the_developer.");
        ;
    };
}

```

```

var argsParser = CommandLine.Parser.Default;
Options options = new Options();
if (argsParser.ParseArgumentsStrict(args, options))
{
    _writeToFile = !String.IsNullOrEmpty(options.OutputFile);

    // Filter type
    if (options.Filter == Filter.Gaussian)
        _filter = new GaussianFilter();
    else
    {
        FilterOptions filterOpts = new FilterOptions();
        if (argsParser.ParseArgumentsStrict(options.Args.ToArray(),
            filterOpts))
        {
            if (filterOpts.Value == null)
                ShowUsage();
            switch (options.Filter)
            {
                case Filter.Difference:
                    _filter = new DifferenceFilter((double)filterOpts.Value);
                    break;
                case Filter.Threshold:
                    _filter = new ThresholdFilter((double)filterOpts.Value);
                    break;
                default:
                    _filter = new GaussianFilter();
                    break;
            }
        }
    }
}

try
{
    _parser = new PlainTextParser(options.NoValueReplacement);
    _fieldsFile = options.FieldsDescription;

    IDataReader mainReader = new PlainTextReader(options.NominalSamples
        [0], options.FieldsDescription, _parser);
    switch (options.Normalization)
    {
        case Normalization.Standard:
            _scaling = new StandardScaling(mainReader);
            break;
    }
}

```

```

        default:
            _scaling = new MinmaxScaling(mainReader);
            break;
    }
    mainReader.Dispose();

    switch (options.Metric)
    {
        case Metric.SqrEuclid:
            _metric = DistanceMetrics.cSqrEulid;
            break;
        default:
            _metric = DistanceMetrics.Euclid;
            break;
    }

    switch (options.ClusterDistanceType)
    {
        case ClusterDistanceType.KMeans:
            _clusterDistMetric = ClusterDistances.CenterDistance;
            break;
        default:
            _clusterDistMetric = ClusterDistances.NearestBoundDistance;
            break;
    }

    Console.WriteLine("Enter_epsilon:");
    double eps;
    while (!double.TryParse(Console.ReadLine(), out eps))
        Console.WriteLine("Wrong_format. Please_enter_epsilon_again.");

    _model = new SystemModel(eps);

    foreach (var nominalSample in options.NominalSamples)
        AddNominalSample(nominalSample);

    foreach (var sample in options.Samples)
        AddSample(sample);

    Console.WriteLine("Knowledge_base_has_been_created. {0} regime(s) total:", _model.Regimes.Count);
    foreach (var regime in _model.Regimes)
    {
        Console.WriteLine("\n***** {0} *****", regime.Name);
    }

```

```

        Console.WriteLine("{0}_cluster(s)_in_regime.", regime.Clusters.
            Count);
        int i = 0;
        foreach (var cluster in regime.Clusters)
        {
            Console.SetBufferSize(Console.BufferWidth, Console.
                BufferHeight + 10);
            Console.WriteLine("_");
            Console.WriteLine("_Cluster_{0}:", ++i);
            Console.WriteLine("_Lower_bound:{0}", String.Join("_|_",
                _scaling.Unscale(cluster.LowerBound)));
            Console.WriteLine("_Upper_bound:{0}", String.Join("_|_",
                _scaling.Unscale(cluster.UpperBound)));
            Console.WriteLine("_Appropriate_discrete_values:{0}",
                String.Join("_|_", cluster.DiscreteValues.Select(f =>
                    String.Join(";", f))));
        }
        Console.WriteLine("_");
        Console.WriteLine("*****", regime.Name);
    }

    Console.WriteLine("\nEnter_record,_or_press_enter_to_quit.");

    string line = String.Empty;
    do
    {
        line = Console.ReadLine();
        if (String.IsNullOrEmpty(line)) break;

        var record = _parser.TryParse(line, _fields);
        if (record == null)
        {
            Console.WriteLine("Wrong_record_format._Please_enter_record_
                again.");
            continue;
        }

        _scaling.Scale(record);
        double distance;
        Regime closest;
        Regime currentRegime = _model.DetectRegime(record, out distance,
            out closest);
        if (currentRegime == null)
            Console.WriteLine("Anomaly_behavior_detected_(closest_regime
                :_{0},_distance:_{1:0.00000}).\n",

```



```

        closest.Name, distance);
    }
    else
        Console.WriteLine("Current regime: {0}\n", currentRegime.
            Name);
    } while (true);
}
catch (DataFormatException dfex)
{
    Console.WriteLine("Wrong data format. {0}", dfex.Message);
    Console.ReadLine();
}
catch (Exception ex)
{
    Console.WriteLine("Error: {0} Please contact the developer.", ex.
        Message);
    Console.ReadLine();
}
}
}
}

/*****
* CleanDataReader.cs
*****/
using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis.DataCleansing
{
    /// <summary>
    /// Decorator for IDataReader: filters all specified anomalies.
    /// </summary>
    public class CleanDataReader : IDataReader
    {
        private IDataReader _baseReader;
        private HashSet<int> _anomalies;

        public CleanDataReader(IDataReader baseReader, IEnumerable<Outlier> anomalies)
        {

```

```

        Contract.Requires<ArgumentNullException>(baseReader != null);
        Contract.Requires<ArgumentNullException>(anomalies != null);

        _baseReader = baseReader;
        _anomalies = new HashSet<int>(anomalies.Select(a => a.Id));
    }

    public IList<Field> Fields
    {
        get { return _baseReader.Fields; }
    }

    public Record ReadRecord()
    {
        var record = _baseReader.ReadRecord();
        if (EndOfData)
            return null;

        if (_anomalies.Contains(record.Id)) // if this record is anomaly
            return ReadRecord();           // go to next record
        else
            return record;
    }

    public void Reset()
    {
        _baseReader.Reset();
    }

    public bool EndOfData
    {
        get { return _baseReader.EndOfData; }
    }

    public int Index
    {
        get { return _baseReader.Index; }
    }

    public IEnumerator<Record> GetEnumerator()
    {
        Reset();
        var record = ReadRecord();
        while (record != null)
        {

```

```

        yield return record;
        record = ReadRecord();
    }
}

IEnumerator IEnumerable.GetEnumerator()
{
    return this.GetEnumerator();
}

#region IDisposable
public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}

private bool m_Disposed = false;

protected virtual void Dispose(bool disposing)
{
    if (!m_Disposed)
    {
        if (disposing)
        {
            // Managed resources are released here.
            _baseReader.Dispose();
        }

        // Unmanaged resources are released here.
        m_Disposed = true;
    }
}

~CleanDataReader()
{
    Dispose(false);
}
#endregion
}

}

/*****
 * DifferenceFilter.cs
 *****/

```

```

using System;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis.DataCleansing
{
    public class DifferenceFilter : IAnomaliesFilter
    {
        private double _delta;

        public DifferenceFilter(double delta)
        {
            Contract.Requires<ArgumentOutOfRangeException>(delta >= 0);

            _delta = delta;
        }

        public IEnumerable<Outlier> Filter(IEnumerable<Outlier> outliers)
        {
            Contract.Requires<ArgumentNullException>(outliers != null);

            var o = outliers.ToArray();
            if (o.Length < 2)
                return Enumerable.Empty<Outlier>();

            for (int i = o.Length - 1; i > 0; i--)
            {
                double diff = o[i-1].Score - o[i].Score;
                if (diff > _delta)
                    return o.Take(i);
            }

            return Enumerable.Empty<Outlier>();
        }
    }
}

/*****
 * GaussianFilter.cs
 *****/

using System;
using System.Collections.Generic;

```

```

using System.Diagnostics.Contracts;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Thesis.Orca;

namespace Thesis.DataCleansing
{
    /// <summary>
    /// Filters anomalies which score is greater than  $Mo+3$ .
    /// </summary>
    public class GaussianFilter : IAnomaliesFilter
    {
        public IEnumerable<Outlier> Filter(IEnumerable<Outlier> outliers)
        {
            Contract.Requires<ArgumentNullException>(outliers != null);

            var anomalies = new List<Outlier>();
            double cutoff = GetCutoff(outliers);
            foreach (var outlier in outliers)
            {
                if (outlier.Score > cutoff)
                    anomalies.Add(outlier);
            }
            return anomalies;
        }

        private IDictionary<double, double> GetFrequencies(IList<Outlier> outliers)
        {
            return outliers.GroupBy(o => o.Score)
                .ToDictionary(gr => gr.Key,
                    gr => (double)gr.Count() / (double)outliers.Count);
        }

        private double GetCutoff(IEnumerable<Outlier> outliers)
        {
            var freq = GetFrequencies(outliers.ToList());
            double mean, disp;
            GetStatValues(freq, out mean, out disp);
            double cutoff = mean + 3 * Math.Sqrt(disp);
            return cutoff;
        }

        /// <summary>

```

```

    /// Calculates mean value and dispersion by one pass through frequencies
    /// dictionary.
    /// </summary>
    private void GetStatValues(IDictionary<double, double> freq, out double mean,
        out double dispersion)
    {
        mean = 0;
        dispersion = 0;
        foreach (var f in freq)
        {
            double x = f.Key * f.Value;
            mean += x;
            dispersion += x * f.Key;
        }

        dispersion -= mean * mean;
    }

    // Slow
    //private double Mean(IDictionary<double, double> freq)
    //{
    //    return freq.Select(v => v.Key * v.Value).Sum();
    //}

    //private double Dispersion(IDictionary<double, double> freq, double mean)
    //{
    //    return freq.Select(v => v.Key * v.Key * v.Value).Sum()
    //        - Math.Pow(Mean(freq), 2);
    //}
}

/*****
 * IAnomaliesFilter.cs
 *****/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Thesis.Orca;

namespace Thesis.DataCleansing
{
    public interface IAnomaliesFilter

```

```

    {
        /// <summary>
        /// Return anomalies in all outliers.
        /// </summary>
        IEnumerable<Outlier> Filter(IEnumerable<Outlier> outliers);
    }
}

/*****
* ThresholdFilter.cs
*****/

using System;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis.DataCleansing
{
    /// <summary>
    /// Filters anomalies which score (reduced by min score) is greater then threshold.
    /// </summary>
    public class ThresholdFilter : IAnomaliesFilter
    {
        private double _threshold;

        public ThresholdFilter(double threshold)
        {
            Contract.Requires<ArgumentOutOfRangeException>(threshold >= 0);

            _threshold = threshold;
        }

        public IEnumerable<Outlier> Filter(IEnumerable<Outlier> outliers)
        {
            double min = outliers.Min().Score;
            foreach (var outlier in outliers)
            {
                if (outlier.Score - min > _threshold)
                    yield return outlier;
            }
        }
    }
}

```

```

/*****
* Cluster.cs
*****/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    public class Cluster
    {
        public float[] UpperBound { get; set; }
        public float[] LowerBound { get; set; }

        /// <summary>
        /// Appropriate discrete values for this cluster.
        /// </summary>
        public HashSet<int>[] DiscreteValues { get; set; }

        internal Cluster(Record init)
        {
            LowerBound = (float[])init.Real.Clone();
            UpperBound = (float[])init.Real.Clone();
            DiscreteValues = init.Discrete.Select(i =>
            {
                var hs = new HashSet<int>();
                hs.Add(i);
                return hs;
            }).ToArray();
        }

        /// <summary>
        /// Determines if record lies inside cluster boundaries.
        /// </summary>
        internal bool Contains(Record record)
        {
            int realFieldsCount = UpperBound.Length;
            for (int i = 0; i < realFieldsCount; i++)
            {
                if (record.Real[i] > UpperBound[i])
                    return false;
                if (record.Real[i] < LowerBound[i])

```



```

        return false;
    }

    int discreteFieldsCount = DiscreteValues.Length;
    for (int i = 0; i < discreteFieldsCount; i++)
    {
        if (!DiscreteValues[i].Contains(record.Discrete[i]))
            return false;
    }

    return true;
}

/// <summary>
/// Adds record to the cluster (expands cluster bounds, if necessary).
/// </summary>
internal void Add(Record record)
{
    // Expand cluster bounds
    int realFieldsCount = UpperBound.Length;
    for (int i = 0; i < realFieldsCount; i++)
    {
        if (record.Real[i] > UpperBound[i])
            UpperBound[i] = record.Real[i];
        else if (record.Real[i] < LowerBound[i])
            LowerBound[i] = record.Real[i];
    }

    // Add discrete values to cluster appropriate values
    int discreteFieldsCount = DiscreteValues.Length;
    for (int i = 0; i < discreteFieldsCount; i++)
        DiscreteValues[i].Add(record.Discrete[i]);
}

}

/*****
 * ClusterDatabase.cs
 *****/

using System;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

```

```

namespace Thesis.DDMS
{
    class ClusterDatabase
    {
        private List<Cluster> _clusters = new List<Cluster>();
        private double _eps;
        private Weights _weights;
        private ClusterDistanceMetric _distance;
        private DistanceMetric _metric;

        private bool IsEmpty
        {
            get { return _clusters.Count == 0; }
        }

        public int Size
        {
            get { return _clusters.Count; }
        }

        public IReadOnlyCollection<Cluster> Clusters
        {
            get { return _clusters.AsReadOnly(); }
        }

        public ClusterDatabase(double eps, Weights weights, ClusterDistanceMetric
            distanceFunc, DistanceMetric metric)
        {
            Contract.Requires<ArgumentOutOfRangeException>(eps >= 0);
            Contract.Requires<ArgumentNullException>(weights != null);
            Contract.Requires<ArgumentNullException>(distanceFunc != null);
            Contract.Requires<ArgumentNullException>(metric != null);

            _eps = eps;
            _weights = weights;
            _distance = distanceFunc;
            _metric = metric;
        }

        public void AddRecord(Record record)
        {
            if (IsEmpty) // if cluster database empty
            {
                // form input vector into cluster and insert into cluster database
            }
        }
    }
}

```

```

        AddCluster(new Cluster(record));
    }
    else
    {
        // if record is inside at least one cluster, do nothing;
        // else:
        if (!_clusters.Any(c => c.Contains(record)))
        {
            double dist;
            Cluster closest = FindClosest(record, out dist);
            // if distance to the closest cluster is greater than epsilon
            if (dist > _eps)
                // add new cluster initialized by this record
                AddCluster(new Cluster(record));
            else
                // add record to the closest cluster
                closest.Add(record);
        }
    }
}

///<summary>
///Determines if record is inside or close enough
///to at least one cluster in database.
///</summary>
public bool Contains(Record record)
{
    // if distance to the closest cluster is less than epsilon, return true
    return Distance(record) <= _eps;
}

///<summary>
///Calculates distance from record to the closest cluster.
///</summary>
public double Distance(Record record)
{
    if (record == null)
        return double.PositiveInfinity;

    // if record is inside at least one cluster
    if (_clusters.Any(c => c.Contains(record)))
        return 0;

    double dist;
    Cluster closest = FindClosest(record, out dist);

```

```

        return dist;
    }

    private void AddCluster(Cluster cluster)
    {
        if (cluster != null)
            _clusters.Add(cluster);
    }

    /// <summary>
    /// Finds closest cluster from specified record.
    /// </summary>
    private Cluster FindClosest(Record record, out double dist)
    {
        dist = double.PositiveInfinity;
        Cluster closest = null;
        foreach (var cluster in _clusters)
        {
            double d = _distance(cluster, record, _weights, _metric);
            if (d < dist)
            {
                dist = d;
                closest = cluster;
            }
        }

        return closest;
    }
}

/*****
* ClusterDistanceMetric.cs
*****/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis.DDMS
{
    public delegate double ClusterDistanceMetric(Cluster c, Record x, Weights weights,
        DistanceMetric metric);
}

```

```

/*****
* ClusterDistances.cs
*****/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis.DDMS
{
    public static class ClusterDistances
    {
        /// <summary>
        /// Distance from record to the center of cluster.
        /// </summary>
        public static double CenterDistance(Cluster cluster, Record record, Weights
            weights, DistanceMetric metric)
        {
            int realFieldsCount = record.Real.Length;
            int discreteFieldsCount = record.Discrete.Length;

            // Calculate center of the cluster
            Record center = new Record();
            center.Real = new float[realFieldsCount];
            center.Discrete = new int[discreteFieldsCount];
            for (int i = 0; i < realFieldsCount; i++)
                center.Real[i] = (cluster.LowerBound[i] + cluster.UpperBound[i]) / 2;
            for (int i = 0; i < discreteFieldsCount; i++)
                // if cluster contains value, keep it the same as in record
                center.Discrete[i] = cluster.DiscreteValues[i].Contains(record.Discrete[
                    i]) ? record.Discrete[i] : -1;

            return metric(center, record, weights);
        }

        /// <summary>
        /// Distance from record to the nearest cluster boundary line.
        /// </summary>
        public static double NearestBoundDistance(Cluster cluster, Record record,
            Weights weights, DistanceMetric metric)
        {
            int realFieldsCount = record.Real.Length;
            int discreteFieldsCount = record.Discrete.Length;

```

```

        // Calculate nearest boundary of the cluster
        Record nearestBound = new Record();
        nearestBound.Real = new float[realFieldsCount];
        nearestBound.Discrete = new int[discreteFieldsCount];
        for (int i = 0; i < realFieldsCount; i++)
        {
            if (record.Real[i] >= cluster.LowerBound[i])
            {
                if (record.Real[i] <= cluster.UpperBound[i])
                    nearestBound.Real[i] = record.Real[i];
                else
                    nearestBound.Real[i] = cluster.UpperBound[i];
            }
            else
            {
                nearestBound.Real[i] = cluster.LowerBound[i];
            }
        }
        for (int i = 0; i < discreteFieldsCount; i++)
            // if cluster contains value, keep it the same as in record
            nearestBound.Discrete[i] = cluster.DiscreteValues[i].Contains(record.
                Discrete[i]) ? record.Discrete[i] : -1;

        return metric(nearestBound, record, weights);
    }
}

/*****
* Regime.cs
*****/
using System;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis.DDMS
{
    /// <summary>
    /// Regime of the system.
    /// </summary>
    public class Regime

```

```

{
    private string _name;
    private ClusterDatabase _db;

    public IReadOnlyCollection<Cluster> Clusters
    {
        get { return _db.Clusters; }
    }

    public string Name
    {
        get { return _name; }
    }

    public Regime(string name, double eps, IDataReader records,
        ClusterDistanceMetric distanceFunc, DistanceMetric metric)
    {
        Contract.Requires<ArgumentOutOfRangeException>(eps >= 0);
        Contract.Requires<ArgumentNullException>(records != null);
        Contract.Requires<ArgumentNullException>(distanceFunc != null);
        Contract.Requires<ArgumentNullException>(metric != null);

        _name = name;
        _db = new ClusterDatabase(eps, records.Fields.Weights(), distanceFunc,
            metric);
        foreach (var record in records)
            _db.AddRecord(record);
    }

    /// <summary>
    /// Determines if record is inside or close enough
    /// to at least one cluster in the regime.
    /// </summary>
    public bool Contains(Record record)
    {
        return _db.Contains(record);
    }

    /// <summary>
    /// Calculates distance from record to the regime.
    /// </summary>
    public double Distance(Record record)
    {
        return _db.Distance(record);
    }
}

```

```

    }
}

/*****
* SystemModel.cs
*****/

using System;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis.DDMS
{
    /// <summary>
    /// Represents all regimes of work for the system.
    /// </summary>
    public class SystemModel
    {
        private ClusterDistanceMetric _distanceFunc;
        private DistanceMetric _metric;
        private List<Regime> _regimes = new List<Regime>();
        private double _eps;

        public IReadOnlyCollection<Regime> Regimes
        {
            get { return _regimes.AsReadOnly(); }
        }

        public SystemModel(double eps)
            : this(eps, ClusterDistances.NearestBoundDistance, DistanceMetrics.Euclid)
        { }

        public SystemModel(double eps, ClusterDistanceMetric distanceFunc,
            DistanceMetric metric)
        {
            Contract.Requires<ArgumentOutOfRangeException>(eps >= 0);
            Contract.Requires<ArgumentNullException>(distanceFunc != null);
            Contract.Requires<ArgumentNullException>(metric != null);

            _distanceFunc = distanceFunc;
            _metric = metric;
            _eps = eps;
        }
    }
}

```



```

public void AddRegime(string name, IDataReader records)
{
    Contract.Requires<ArgumentNullException>(records != null);

    Regime regime = new Regime(name, _eps, records, _distanceFunc, _metric);
    _regimes.Add(regime);
}

/// <summary>
/// Detects closest regime to the record.
/// Returns null, if record is further then epsilon from all regimes.
/// </summary>
public Regime DetectRegime(Record record)
{
    double distance;
    Regime closest;
    return DetectRegime(record, out distance, out closest);
}

/// <summary>
/// Detects closest regime to the record and calculates distance between them.
/// Returns null, if record is further then epsilon from all regimes.
/// </summary>
public Regime DetectRegime(Record record, out double distance, out Regime
    closestRegime)
{
    double mind = double.PositiveInfinity;
    closestRegime = null;

    foreach (var regime in _regimes)
    {
        double d = regime.Distance(record);
        if (d < mind)
        {
            mind = d;
            closestRegime = regime;
        }
    }

    distance = mind;
    return mind > _eps ? null : closestRegime;
}
}
}

```

```

/*****
* BinaryShuffle.cs
*****/

using System;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    class BinaryShuffle
    {
        private IDataReader _sourceReader;
        private IDataReader _shuffleReader;

        private string _shuffleFile;

        private int _iterations;
        private int _randFilesCount;
        private Random _rand;

        /// <param name="sourceReader">Reader for input data.</param>
        /// <param name="shuffleFile">Name of shuffle binary file.</param>
        /// <param name="iterations">Number of shuffle iterations.</param>
        /// <param name="randFilesCount">Number of random part-files.</param>
        public BinaryShuffle(IDataReader sourceReader, string shuffleFile,
            int iterations = 5, int randFilesCount = 10)
        {
            Contract.Requires<ArgumentNullException>(sourceReader != null);
            Contract.Requires<ArgumentException>(!String.IsNullOrEmpty(shuffleFile));
            Contract.Requires<ArgumentOutOfRangeException>(iterations >= 1);
            Contract.Requires<ArgumentOutOfRangeException>(randFilesCount >= 1);

            _sourceReader = sourceReader;
            _shuffleFile = shuffleFile;
            _iterations = iterations;
            _randFilesCount = randFilesCount;

            _rand = new Random((int)DateTime.Now.Ticks);

            MultiShuffle();
        }
    }
}

```

```

}

private void SetFileReader(string filename)
{
    Contract.Requires(!String.IsNullOrEmpty(filename));

    // if infile_ already points to a file, close it
    if (_shuffleReader != null)
        _shuffleReader.Dispose();

    _shuffleReader = new BinaryDataReader(filename);
}

private void ResetFileReader()
{
    if (_shuffleReader != null)
        _shuffleReader.Dispose();
}

public void MultiShuffle()
{
    Shuffle(_sourceReader, _shuffleFile);
    _shuffleReader = new BinaryDataReader(_shuffleFile);

    for (int i = 1; i < _iterations; i++)
    {
        Shuffle(_shuffleReader, _shuffleFile);
        SetFileReader(_shuffleFile);
    }

    ResetFileReader();
}

private void Shuffle(IDataReader sourceReader, string filename)
{
    //-----
    // set up tmp file names
    //
    string[] tmpFileNames = new string[_randFilesCount];
    for (int i = 0; i < _randFilesCount; i++)
        tmpFileNames[i] = filename + ".tmp." + i.ToString();

    //-----

```

```

// open files for writing
//
IDataWriter[] tmpFilesOut = new BinaryDataWriter[_randFilesCount];
try
{
    for (int i = 0; i < tmpFileNames.Length; i++)
        tmpFilesOut[i] = new BinaryDataWriter(tmpFileNames[i], _sourceReader
            .Fields);

    //-----
    // read in data file and randomly shuffle examples to
    // temporary files
    //
    foreach (var rec in _sourceReader)
    {
        int index = _rand.Next(tmpFilesOut.Length);
        tmpFilesOut[index].WriteRecord(rec);
    }
}
finally
{
    // close temporary files
    for (int i = 0; i < tmpFilesOut.Length; i++)
        if (tmpFilesOut[i] != null)
            tmpFilesOut[i].Dispose();
}

//-----
// open tmpfiles for reading
//

IDataReader[] tmpFilesIn = new BinaryDataReader[_randFilesCount];
try
{
    for (int i = 0; i < tmpFilesIn.Length; i++)
        tmpFilesIn[i] = new BinaryDataReader(tmpFileNames[i]);

    //-----
    // open final destination file
    //

    ResetFileReader(); // closes original file

    using (IDataWriter outfile = new BinaryDataWriter(filename,
        _sourceReader.Fields))

```

```

{
    //-----
    // concatenate tmp files in random order
    //
    int[] order = new int[_randFilesCount];
    for (int i = 0; i < _randFilesCount; i++)
        order[i] = i;

    // The modern version of the -FisherYates shuffle (the Knuth shuffle
    )
    for (int i = order.Length - 1; i >= 0; i--)
    {
        int j = _rand.Next(i + 1);
        int temp = order[i];
        order[i] = order[j];
        order[j] = temp;
    }

    for (int i = 0; i < order.Length; i++)
    {
        IDataReader infile = tmpFilesIn[order[i]];
        foreach (var rec in infile)
            outfile.WriteRecord(rec);
    }
}

finally
{
    // close temporary files
    for (int i = 0; i < tmpFilesIn.Length; i++)
        if (tmpFilesIn[i] != null)
            tmpFilesIn[i].Dispose();
}

//-----
// delete tmpfiles
//
foreach (var fileName in tmpFileNames)
    File.Delete(fileName);

ResetFileReader();
}
}
}

```

```

/*****
* DistanceMetric.cs
*****/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    public delegate double DistanceMetric(Record a, Record b, Weights weights);
}

/*****
* DistanceMetrics.cs
*****/

using System;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    public static class DistanceMetrics
    {
        /// <summary>
        /// Squared Euclid distance.
        /// </summary>
        public static double cSqrEulid(Record a, Record b, Weights weights)
        {
            Contract.Requires(a.Real.Length == b.Real.Length);
            Contract.Requires(a.Discrete.Length == b.Discrete.Length);
            Contract.Requires(a.Real.Length == weights.Real.Length);
            Contract.Requires(a.Discrete.Length == weights.Discrete.Length);

            int realFieldsCount = a.Real.Length;
            int discreteFieldsCount = a.Discrete.Length;

            double d = 0;

            // real
            for (int i = 0; i < realFieldsCount; i++)

```

```

    {
        // check for missing values
        int missingCount = 0;
        if (float.IsNaN(a.Real[i]))
            missingCount++;
        if (float.IsNaN(b.Real[i]))
            missingCount++;

        if (missingCount == 0)
        {
            double diff = a.Real[i] - b.Real[i];
            d += diff * diff * weights.Real[i];
        }
        // one value is missing
        else if (missingCount == 1)
        {
            d += weights.Real[i];
        }
    }

    // discrete
    for (int i = 0; i < discreteFieldsCount; i++)
    {
        if (a.Discrete[i] != b.Discrete[i])
            d += weights.Discrete[i];
    }

    return d;
}

public static double Euclid(Record a, Record b, Weights weights)
{
    return Math.Sqrt(cSqrEulid(a, b, weights));
}
}

/*****
* Field.cs
*****/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

```

```

using System.Diagnostics.Contracts;

namespace Thesis
{
    public class Field
    {
        public enum FieldType
        {
            Discrete = 0,
            /// <summary>
            /// Discrete Data Driven
            /// </summary>
            DiscreteDataDriven = 1,
            Continuous = 2,
            IgnoreFeature = 3
        }

        public string Name { get; set; }

        public FieldType Type { get; set; }

        /// <summary>
        /// List of values for discrete field.
        /// </summary>
        public IList<string> Values { get; set; }

        private float _weight = float.NaN;
        public float Weight
        {
            get { return _weight; }
            set { _weight = value; }
        }

        public bool HasWeight
        {
            get { return !float.IsNaN(Weight); }
        }
    }
}

/*****
 * FieldsHelper.cs
 *****/

using System;
using System.Collections.Generic;

```



```

using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    public static class FieldsHelper
    {
        public static int RealCount(this IEnumerable<Field> fields)
        {
            return fields.Count(f => f.Type == Field.FieldType.Continuous);
        }

        public static int DiscreteCount(this IEnumerable<Field> fields)
        {
            return fields.Count(f => f.Type == Field.FieldType.Discrete ||
                                   f.Type == Field.FieldType.DiscreteDataDriven);
        }

        public static Weights Weights(this IEnumerable<Field> fields)
        {
            var real = fields.Where(f => f.Type == Field.FieldType.Continuous).Select(f
                => f.Weight).ToArray();
            var discrete = fields.Where(f => f.Type == Field.FieldType.Discrete)
                .Concat(fields.Where(f => f.Type == Field.FieldType
                    .DiscreteDataDriven))
                .Select(f => f.Weight).ToArray();
            return new Weights() { Real = real, Discrete = discrete };
        }
    }
}

/*****
* Outlier.cs
*****/

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    [DebuggerDisplay("Id={Id}; Score={Score}")]

```

```

public struct Outlier : IComparable<Outlier>
{
    //public Record Record { get; set; }
    public int Id { get; set; }
    public double Score { get; set; }
    //public IEnumerable<int> Neighbors { get; set; }

    public int CompareTo(Outlier other)
    {
        return Score.CompareTo(other.Score);
    }
}

}

/*****
* Record.cs
*****/

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    [DebuggerDisplay("Id={Id}")]
    public class Record : ICloneable
    {
        public int Id { get; set; }
        public float[] Real { get; set; }
        public int[] Discrete { get; set; }

        public Record() { }

        public Record(int id, float[] real, int[] discrete)
        {
            Id = id;
            Real = real;
            Discrete = discrete;
        }

        object ICloneable.Clone()
        {
            return this.Clone();
        }
    }
}

```

```

    }

    public Record Clone()
    {
        return new Record(Id, (float[])Real.Clone(), (int[])Discrete.Clone());
    }
}

/*****
* Weights.cs
*****/

using System;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    public class Weights
    {
        public float[] Real { get; set; }
        public float[] Discrete { get; set; }

        public static Weights Identity(int realFieldsCount, int discreteFieldsCount)
        {
            Contract.Requires<ArgumentOutOfRangeException>(realFieldsCount >= 0);
            Contract.Requires<ArgumentOutOfRangeException>(discreteFieldsCount >= 0);

            float[] real = Enumerable.Repeat(1f, realFieldsCount).ToArray();
            float[] discrete = Enumerable.Repeat(1f, discreteFieldsCount).ToArray();

            return new Weights()
            {
                Real = real,
                Discrete = discrete
            };
        }
    }
}

/*****
* BinaryHeap.cs
*****/

```

```

*****/
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis.Collections
{
    public class BinaryHeap<T> : ICollection, IEnumerable<T> where T : IComparable
    {
        private const int _defaultCapacity = 4;

        private List<T> _items;

        public BinaryHeap()
            : this(_defaultCapacity) { }

        public BinaryHeap(int capacity)
        {
            if (capacity < 0)
                throw new ArgumentOutOfRangeException("capacity");
            _items = new List<T>(capacity);
        }

        public BinaryHeap(IEnumerable<T> collection)
        {
            _items = new List<T>(collection);
            BuildHeap(_items);
        }

        private static void SwapElements(IList<T> list, int a, int b)
        {
            T temp = list[a];
            list[a] = list[b];
            list[b] = temp;
        }

        private static void BuildHeap(IList<T> a)
        {
            for (int i = a.Count / 2; i > 0; i--)
                Heapify(a, i);
        }
    }
}

```

```

private static void Heapify(IList<T> a, int i)
{
    int left = 2 * i;
    int right = 2 * i + 1;
    int largest = i;

    if (left <= a.Count && a[left - 1].CompareTo(a[i - 1]) > 0)
        largest = left;
    if (right <= a.Count && a[right - 1].CompareTo(a[largest - 1]) > 0)
        largest = right;

    if (largest != i)
    {
        SwapElements(a, i - 1, largest - 1);
        Heapify(a, largest);
    }
}

private static void HeapIncreaseKey(IList<T> a, int i, T key)
{
    a[i - 1] = key;

    for (int j = i; j > 1 && a[j / 2 - 1].CompareTo(a[j - 1]) < 0; j = j / 2)
        SwapElements(a, j - 1, j / 2 - 1);

    //for (int j = i; j > 0 && a[j].CompareTo(a[j - 1]) > 0; j--)
    //    SwapElements(a, j, j - 1);
}

public int IndexOf(T item)
{
    return _items.IndexOf(item);
}

public T this[int index]
{
    get
    {
        return _items[index];
    }
    set
    {
        HeapIncreaseKey(_items, index + 1, value);
    }
}

```

```

public void Push(T item)
{
    _items.Add(item);
    HeapIncreaseKey(_items, _items.Count, item);
}

public void Clear()
{
    _items.Clear();
}

public bool Contains(T item)
{
    return _items.Contains(item);
}

public void CopyTo(T[] array, int arrayIndex)
{
    _items.CopyTo(array, arrayIndex);
}

public IEnumerator<T> GetEnumerator()
{
    return _items.GetEnumerator();
}

IEnumerator IEnumerable.GetEnumerator()
{
    return _items.GetEnumerator();
}

/// <summary>
/// Removes and returns max element of the heap.
/// </summary>
/// <returns>Max element of the heap</returns>
public T Pop()
{
    if (Count == 0)
        throw new InvalidOperationException("Heap is empty.");

    T max = this[0];
    if (Count == 1)
    {
        _items.Remove(max);
    }
}

```

```

        return max;
    }

    T last = _items[_items.Count - 1];
    _items.Remove(last);
    _items[0] = last;
    Heapify(_items, 1);
    return max;
}

public T Peek()
{
    if (Count == 0)
        throw new InvalidOperationException("Heap is empty.");
    return this[0];
}

#region ICollection
public int Count
{
    get { return _items.Count; }
}

public void CopyTo(Array array, int index)
{
    ((ICollection)_items).CopyTo(array, index);
}

public bool IsSynchronized
{
    get { return false; }
}

public object SyncRoot
{
    get { return this; }
}
#endregion
}

}

/*****
* BatchDataReader.cs
*****/
using System;

```

```

using System.Collections;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    public class BatchDataReader : IDataReader
    {
        IDataReader _baseReader;

        int _batchSize;
        int _lastOffset;

        public int Offset { get; private set; }

        public IList<Record> CurrentBatch { get; private set; }

        public BatchDataReader(IDataReader baseReader, int batchSize)
        {
            Contract.Requires<ArgumentNullException>(baseReader != null);
            Contract.Requires<ArgumentOutOfRangeException>(batchSize > 0);

            _baseReader = baseReader;
            _baseReader.Reset();
            _batchSize = batchSize;
        }

        /// <summary>
        /// Reads next batch of records.
        /// </summary>
        public bool GetNextBatch()
        {
            CurrentBatch = new List<Record>(_batchSize);

            int nr = 0;
            for (int i = 0; i < _batchSize; i++)
            {
                if (_baseReader.EndOfData)
                    break;
                CurrentBatch.Add(_baseReader.ReadRecord());
                nr++;
            }
        }
    }
}

```



```

        Offset += _lastOffset;
        _lastOffset = nr;

        return nr > 0;
    }

    #region IDataReader
    public IList<Field> Fields
    {
        get { return _baseReader.Fields; }
    }

    public Record ReadRecord()
    {
        return _baseReader.ReadRecord();
    }

    public void Reset()
    {
        _baseReader.Reset();
        Offset = 0;
        _lastOffset = 0;
        CurrentBatch = null;
    }

    public bool EndOfData
    {
        get { return _baseReader.EndOfData; }
    }

    public int Index
    {
        get { return _baseReader.Index; }
    }

    public IEnumerator<Record> GetEnumerator()
    {
        Reset();
        var record = ReadRecord();
        while (record != null)
        {
            yield return record;
            record = ReadRecord();
        }
    }

```

```

    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return this.GetEnumerator();
    }
#endregion

#region IDisposable
public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}

private bool m_Disposed = false;

protected virtual void Dispose(bool disposing)
{
    if (!m_Disposed)
    {
        if (disposing)
        {
            // Managed resources are released here.
            _baseReader.Dispose();
        }

        // Unmanaged resources are released here.
        m_Disposed = true;
    }
}

~BatchDataReader()
{
    Dispose(false);
}
#endregion
}
}

/*****
 * DataFormatException.cs
 *****/
using System;
using System.Collections.Generic;

```

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    public class DataFormatException : FormatException
    {
        public DataFormatException()
            : base() { }

        public DataFormatException(string message)
            : base(message) { }

        public DataFormatException(string message, Exception innerException)
            : base(message, innerException) { }
    }
}

/*****
* DataHelper.cs
*****/

using System;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    public static class DataHelper
    {
        public static void CopyTo(this IDataReader reader, IDataWriter destination)
        {
            foreach (var record in reader)
                destination.WriteRecord(record);
        }

        public static void Shuffle(this IDataReader reader, string outputFile,
                                   int iterations = 5, int randFilesCount = 10)
        {
            Contract.Requires<ArgumentException>(!String.IsNullOrEmpty(outputFile));
            Contract.Requires<ArgumentOutOfRangeException>(iterations >= 1);
            Contract.Requires<ArgumentOutOfRangeException>(randFilesCount >= 1);
        }
    }
}

```

```

        BinaryShuffle shuffle = new BinaryShuffle(reader, outputFile, iterations,
            randFilesCount);
        shuffle.MultiShuffle();
    }
}

/*****
* IDataReader.cs
*****/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    public interface IDataReader: IEnumerable<Record>, IDisposable
    {
        IList<Field> Fields { get; }

        Record ReadRecord();

        void Reset();

        bool EndOfData { get; }

        int Index { get; }
    }
}

/*****
* IDataWriter.cs
*****/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    public interface IDataWriter : IDisposable

```

```

    {
        void WriteRecord(Record record);
        int Count { get; }
    }
}

/*****
* IRecordParser.cs
*****/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    public interface IRecordParser<T>
    {
        Record Parse(T input, IList<Field> fields);

        /// <summary>
        /// Returns null if parsing failed.
        /// </summary>
        Record TryParse(T input, IList<Field> fields);
    }
}

/*****
* ScaleDataReader.cs
*****/

using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    /// <summary>
    /// Scales every record on reading with defined scaling method.
    /// </summary>
    public class ScaleDataReader : IDataReader

```

```

{
    private IDataReader _baseReader;
    private IScaling _scaling;

    public ScaleDataReader(IDataReader baseReader, IScaling scaling)
    {
        Contract.Requires<ArgumentNullException>(baseReader != null);
        Contract.Requires<ArgumentNullException>(scaling != null);

        _baseReader = baseReader;
        _scaling = scaling;
    }

    #region IDataReader
    public IList<Field> Fields
    {
        get { return _baseReader.Fields; }
    }

    public Record ReadRecord()
    {
        var record = _baseReader.ReadRecord();
        if (record != null)
            _scaling.Scale(record);
        return record;
    }

    public void Reset()
    {
        _baseReader.Reset();
    }

    public bool EndOfData
    {
        get { return _baseReader.EndOfData; }
    }

    public int Index
    {
        get { return _baseReader.Index; }
    }

    public IEnumerator<Record> GetEnumerator()
    {
        Reset();

```

```

        var record = ReadRecord();
        while (record != null)
        {
            yield return record;
            record = ReadRecord();
        }
    }

IEnumerator IEnumerable.GetEnumerator()
{
    return this.GetEnumerator();
}
#endregion

#region IDisposable
public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}

private bool m_Disposed = false;

protected virtual void Dispose(bool disposing)
{
    if (!m_Disposed)
    {
        if (disposing)
        {
            // Managed resources are released here.
            _baseReader.Dispose();
        }

        // Unmanaged resources are released here.
        m_Disposed = true;
    }
}

~ScaleDataReader()
{
    Dispose(false);
}
#endregion
}
}

```

```

/*****
* BinaryDataReader.cs
*****/

using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    /// <summary>
    /// Represents binary record format file reader.
    /// </summary>
    public class BinaryDataReader : IDataReader
    {
        BinaryReader _infile;

        public int RecordsCount { get; private set; }
        public int RealFieldsCount { get; private set; }
        public int DiscreteFieldsCount { get; private set; }

        private long _dataOffset; // 0 + size of header

        private IList<Field> _fields;

        private readonly string _tempFile;

        public BinaryDataReader(string filename)
        {
            InitReader(filename);
        }

        /// <summary>
        /// Copies all record from source to binary file using BinaryDataWriter
        /// and opens data reader on that file.
        /// </summary>
        /// <param name="source">Source data reader.</param>
        /// <param name="tempFile">Temporary binary file name.</param>
        /// <param name="keepBinary">true, if binary file should be deleted hereafter.</
        param>

```



```

public BinaryDataReader(IDataReader source, string tempFile, bool keepBinary =
    false)
{
    var writer = new BinaryDataWriter(source, tempFile);
    writer.Dispose();
    if (!keepBinary)
        _tempFile = tempFile;
    InitReader(tempFile);
}

/// <summary>
/// Initializes reader.
/// </summary>
/// <param name="filename">Input data file name.</param>
private void InitReader(string filename)
{
    _infile = new BinaryReader(File.OpenRead(filename));
    _fields = new List<Field>();
    ReadHeader();
    SeekPosition(0);
}

private void ReadHeader()
{
    long oldPos = _infile.BaseStream.Position;

    _infile.BaseStream.Position = 0;
    RecordsCount = _infile.ReadInt32();
    RealFieldsCount = _infile.ReadInt32();
    DiscreteFieldsCount = _infile.ReadInt32();

    int fieldsCount = _infile.ReadInt32();
    for (int i = 0; i < fieldsCount; i++)
        _fields.Add(ReadField());

    _dataOffset = _infile.BaseStream.Position;
    _infile.BaseStream.Position = oldPos;
}

private Field ReadField()
{
    string name = _infile.ReadString();
    Field.FieldType type = (Field.FieldType)_infile.ReadInt32();
    float weight = _infile.ReadSingle();
}

```

```

    bool hasValues = _infile.ReadBoolean();
    List<string> values = null;
    if (hasValues)
    {
        int valuesCount = _infile.ReadInt32();
        values = new List<string>();
        for (int i = 0; i < valuesCount; i++)
            values.Add(_infile.ReadString());
    }

    return new Field() { Name = name, Type = type, Weight = weight, Values =
        values };
}

private void SeekPosition(int pos)
{
    Contract.Requires<ArgumentOutOfRangeException>(pos >= 0);
    Contract.Requires<ArgumentOutOfRangeException>(pos <= RecordsCount);

    long filepos = _dataOffset + pos *
        (sizeof(int) + RealFieldsCount * sizeof(float) + DiscreteFieldsCount *
            sizeof(int));

    _infile.BaseStream.Position = filepos;
    Index = pos;
}

#region IDataReader
public int Index { get; set; }

public IList<Field> Fields
{
    get { return _fields; }
}

public Record ReadRecord()
{
    if (EndOfData)
        return null;

    var id = _infile.ReadInt32();
    var real = _infile.ReadFloatArray(RealFieldsCount);
    var discrete = _infile.ReadIntArray(DiscreteFieldsCount);

    Index++;
}

```

```

        return new Record(id, real, discrete);
    }

    public void Reset()
    {
        SeekPosition(0);
    }

    public bool EndOfData
    {
        get { return Index == RecordsCount; }
    }

    public IEnumerator<Record> GetEnumerator()
    {
        Reset();
        while (!EndOfData)
            yield return ReadRecord();
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return this.GetEnumerator();
    }
#endregion

#region IDisposable
public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}

private bool m_Disposed = false;

protected virtual void Dispose(bool disposing)
{
    if (!m_Disposed)
    {
        if (disposing)
        {
            // Managed resources are released here.
            _infile.Close();
            if (!String.IsNullOrEmpty(_tempFile))

```

```

        File.Delete(_tempFile);
    }

    // Unmanaged resources are released here.
    m_Disposed = true;
}

~BinaryDataReader()
{
    Dispose(false);
}
#endregion
}
}

```

```

/*****
* BinaryDataWriter.cs
*****/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Diagnostics.Contracts;

namespace Thesis
{
    /// <summary>
    /// Represents Orca format binary file writer.
    /// </summary>
    public class BinaryDataWriter : IDataWriter
    {
        BinaryWriter _outfile;

        bool _headerWritten = false;

        IList<Field> _fields;

        private int _realFieldsCount;
        private int _discreteFieldsCount;

        private int _count = 0; // number of records
    }
}

```

```

public BinaryDataWriter(string filename, IList<Field> fields)
{
    Contract.Requires<ArgumentException>(!String.IsNullOrEmpty(filename));
    Contract.Requires<ArgumentNullException>(fields != null);

    _outfile = new BinaryWriter(File.Create(filename));

    _fields = fields;
    _realFieldsCount = fields.RealCount();
    _discreteFieldsCount = fields.DiscreteCount();

    WriteHeader();
}

/// <summary>
/// Creates new BinaryDataWriter and copies all records from IDataReader source.
/// </summary>
public BinaryDataWriter(IDataReader source, string filename)
    : this(filename, source.Fields)
{
    foreach (var record in source)
        WriteRecord(record);

    WriteHeader(Count);
}

private void WriteHeader()
{
    //long oldPos = _outfile.BaseStream.Position;

    _outfile.Seek(0, SeekOrigin.Begin);
    _outfile.Write(_count); // number of records
    _outfile.Write(_realFieldsCount);
    _outfile.Write(_discreteFieldsCount);

    _outfile.Write(_fields.Count);
    foreach (var field in _fields)
        WriteField(field);

    //_outfile.BaseStream.Position = oldPos;
    _headerWritten = true;
}

```

```

private void WriteHeader(int numRecords)
{
    if (!_headerWritten)
        WriteHeader();

    long oldPos = _outfile.BaseStream.Position;
    _outfile.Seek(0, SeekOrigin.Begin);
    _outfile.Write(numRecords);
    _outfile.BaseStream.Position = oldPos;
}

private void WriteField(Field field)
{
    _outfile.Write(field.Name);
    _outfile.Write((int) field.Type);
    _outfile.Write(field.Weight);

    bool hasValues = field.Values != null;
    _outfile.Write(hasValues);
    if (hasValues)
    {
        _outfile.Write(field.Values.Count);
        foreach (var value in field.Values)
            _outfile.Write(value);
    }
}

public void WriteRecord(Record record)
{
    if (record == null)
        return;

    if (record.Real.Length != _realFieldsCount ||
        record.Discrete.Length != _discreteFieldsCount)
        throw new ArgumentException("Wrong number of values in record.");

    if (!_headerWritten)
        WriteHeader();

    _outfile.Write(record.Id);
    if (_realFieldsCount > 0)
        _outfile.Write(record.Real);
    if (_discreteFieldsCount > 0)
        _outfile.Write(record.Discrete);
}

```

```

        _count++;
    }

    public int Count
    {
        get { return _count; }
    }

    #region IDisposable
    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this);
    }

    private bool m_Disposed = false;

    protected virtual void Dispose(bool disposing)
    {
        if (!m_Disposed)
        {
            if (disposing)
            {
                // Managed resources are released here.
                WriteHeader(_count);
                _outfile.Close();
            }

            // Unmanaged resources are released here.
            m_Disposed = true;
        }
    }

    ~BinaryDataWriter()
    {
        Dispose(false);
    }
    #endregion
}

/*****
* BinaryHelper.cs
*****/
using System;

```

```

using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    internal static class BinaryHelper
    {
        public static void Write(this BinaryWriter writer, float[] data)
        {
            Contract.Requires<ArgumentNullException>(data != null);

            byte[] binaryData = new byte[data.Length * sizeof(float)];
            System.Buffer.BlockCopy(data, 0, binaryData, 0, binaryData.Length);
            writer.Write(binaryData);
        }

        public static void Write(this BinaryWriter writer, int[] data)
        {
            Contract.Requires<ArgumentNullException>(data != null);

            byte[] binaryData = new byte[data.Length * sizeof(int)];
            System.Buffer.BlockCopy(data, 0, binaryData, 0, binaryData.Length);
            writer.Write(binaryData);
        }

        public static float[] ReadFloatArray(this BinaryReader reader, int count)
        {
            Contract.Requires<ArgumentOutOfRangeException>(count >= 0);

            if (count == 0)
                return new float[0];

            float[] result = new float[count];
            byte[] binaryData = reader.ReadBytes(sizeof(float) * count);
            Buffer.BlockCopy(binaryData, 0, result, 0, binaryData.Length);

            return result;
        }

        public static int[] ReadIntArray(this BinaryReader reader, int count)
        {

```



```

        Contract.Requires<ArgumentOutOfRangeException>(count >= 0);

        if (count == 0)
            return new int[0];

        int[] result = new int[count];
        byte[] binaryData = reader.ReadBytes(sizeof(int) * count);
        Buffer.BlockCopy(binaryData, 0, result, 0, binaryData.Length);

        return result;
    }
}

/*****
 * PlainTextParser.cs
 *****/
using System;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Globalization;

namespace Thesis
{
    public class PlainTextParser : IRecordParser<string>
    {
        private char[] _recordDelimiters;
        private string _noValueReplacement;

        public PlainTextParser(char[] recordDelimiters,
                               string noValueReplacement = "?")
        {
            Contract.Requires<ArgumentNullException>(recordDelimiters != null);
            Contract.Requires<ArgumentException>(recordDelimiters.Length > 0);
            Contract.Requires<ArgumentException>(!String.IsNullOrEmpty(
                noValueReplacement));

            _recordDelimiters = recordDelimiters;
            _noValueReplacement = noValueReplacement;
        }

        public PlainTextParser(string noValueReplacement = "?")

```

```

        : this(new char[] { ',', ';' }, noValueReplacement)
    { }

    /// <summary>
    /// Parses string and returns record. Returns null if string is a comment.
    /// </summary>
    /// <exception cref="Thesis.DataFormatException"/>
    public Record Parse(string input, IList<Field> fields)
    {
        Contract.Requires<ArgumentNullException>(fields != null);

        var tokens = StringHelper.Tokenize(input, _recordDelimiters);

        if (tokens.Length == 0) // if comment
            return null;

        // check to make sure there are the correct number of tokens
        if (tokens.Length != fields.Count)
            throw new DataFormatException("Wrong number of tokens.");

        int realFieldsCount = fields.Count(f => f.Type == Field.FieldType.Continuous);
        int discreteFieldsCount = fields.Count(f => f.Type == Field.FieldType.
            Discrete ||
                f.Type == Field.FieldType.
                    DiscreteDataDriven);

        var real = new float[realFieldsCount];
        var discrete = new int[discreteFieldsCount];
        int iReal = 0;
        int iDiscrete = 0;

        for (int i = 0; i < fields.Count; i++)
        {
            if (fields[i].Type == Field.FieldType.IgnoreFeature)
                continue;
            if (tokens[i] == _noValueReplacement)
            {
                switch (fields[i].Type)
                {
                    case Field.FieldType.Continuous:
                        real[iReal++] = float.NaN; break;
                    case Field.FieldType.Discrete:
                    case Field.FieldType.DiscreteDataDriven:

```

```

        discrete[iDiscrete++] = -1; break;
    }
}
else
{
    switch (fields[i].Type)
    {
        case Field.FieldType.Continuous:
            real[iReal++] = float.Parse(tokens[i], CultureInfo.
                InvariantCulture); break;
        case Field.FieldType.Discrete:
            int value = fields[i].Values.IndexOf(tokens[i]);
            if (value != -1)
                discrete[iDiscrete++] = value;
            else
                throw new DataFormatException(String.Format(
                    "Discrete_{value}_{0}'_{for}_{field}_{1}'_{doesn't}_{exist}.
                    ", tokens[i], fields[i]));
            break;
        case Field.FieldType.DiscreteDataDriven:
            int valuec = fields[i].Values.IndexOf(tokens[i]);
            if (valuec != -1)
                discrete[iDiscrete++] = valuec;
            else
            {
                // Add new value to the field description.
                fields[i].Values.Add(tokens[i]);
                discrete[iDiscrete++] = fields[i].Values.Count - 1;
            }
            break;
    }
}

return new Record(0, real, discrete);
}

public Record TryParse(string input, IList<Field> fields)
{
    Contract.Requires<ArgumentNullException>(fields != null);

    try
    {
        return Parse(input, fields);
    }
}

```

```

        catch (DataFormatException)
        {
            return null;
        }
    }
}

/*****
* PlainTextReader.cs
*****/
using System;
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    public class PlainTextReader : IDataReader
    {
        private IRecordParser<string> _parser;

        private StreamReader _infile;
        private List<Field> _fields = new List<Field>();

        private char[] _fieldsDelimiters;

        private float _realWeight;
        private float _discreteWeight;

        public int _realFieldsCount;
        public int _discreteFieldsCount;

        public PlainTextReader(string dataFile, string fieldsFile, IRecordParser<string>
            parser,
                                float realWeight = 1.0f, float discreteWeight = 0.4f)
            : this(dataFile, fieldsFile, parser,
                    new char[] { ',', ':', ';' }, realWeight, discreteWeight)
        { }

        public PlainTextReader(string dataFile, string fieldsFile, IRecordParser<string>

```

```

        parser,

        char[] fieldsDelimiters,
        float realWeight = 1.0f, float discreteWeight = 0.4f)
{
    Contract.Requires<ArgumentException>(!String.IsNullOrEmpty(dataFile));
    Contract.Requires<ArgumentException>(!String.IsNullOrEmpty(fieldsFile));
    Contract.Requires<ArgumentNullException>(parser != null);
    Contract.Requires<ArgumentNullException>(fieldsDelimiters != null);
    Contract.Requires<ArgumentException>(fieldsDelimiters.Length > 0);

    _parser = parser;

    _realWeight = realWeight;
    _discreteWeight = discreteWeight;
    _fieldsDelimiters = fieldsDelimiters;

    LoadFields(fieldsFile);
    _infile = new StreamReader(dataFile);
    Index = 0;
}

private void LoadFields(string filename)
{
    Contract.Requires(!String.IsNullOrEmpty(filename));

    using (var infile = new StreamReader(filename))
    {
        while (!infile.EndOfStream)
        {
            string line = infile.ReadLine();
            var tokens = StringHelper.Tokenize(line, _fieldsDelimiters);
            if (tokens.Length > 0)
            {
                Field newField = CreateField(tokens);
                _fields.Add(newField);
            }
        }
    }

    _realFieldsCount = _fields.Count(f => f.Type == Field.FieldType.Continuous);
    _discreteFieldsCount = _fields.Count(f => f.Type == Field.FieldType.Discrete
        ||
        f.Type == Field.FieldType.
            DiscreteDataDriven);
}

```

```

}

private Field CreateField(string[] tokens)
{
    Contract.Requires<ArgumentNullException>(tokens != null);
    Contract.Requires<ArgumentException>(tokens.Length > 0);

    Field field = new Field();

    field.Weight = float.NaN; // no weight

    int i = 0; // start token
    float weight;
    if (float.TryParse(tokens[0], out weight)) // if weight is defined
    {
        field.Weight = weight;
        i++;
    }

    field.Name = tokens[i++];
    string sType = tokens[i];

    if (tokens.Length == i)
        field.Type = Field.FieldType.IgnoreFeature;
    else
    {
        switch (sType)
        {
            case "ignore":
                field.Type = Field.FieldType.IgnoreFeature;
                break;
            case "continuous":
                field.Type = Field.FieldType.Continuous;
                if (!field.HasWeight)
                    field.Weight = _realWeight;
                break;
            case "discrete":
                field.Type = Field.FieldType.DiscreteDataDriven;
                field.Values = new List<string>();
                if (!field.HasWeight)
                    field.Weight = _discreteWeight;
                break;
            default:
                //Discrete type of field: adding all of it's values
                field.Type = Field.FieldType.Discrete;

```

```

        field.Values = new List<string>(tokens.Length - 1);
        for (int j = 1; j < tokens.Length; j++)
            field.Values.Add(tokens[j]);
        if (!field.HasWeight)
            field.Weight = _discreteWeight;
        break;
    }
}

return field;
}

#region IDataReader
public IList<Field> Fields
{
    get { return _fields.AsReadOnly(); }
}

public Record ReadRecord()
{
    if (!EndOfData)
    {
        string line = _infile.ReadLine();
        var record = _parser.Parse(line, _fields);
        if (record == null) // if comment
            return ReadRecord(); // go to next line
        else
        {
            Index++;
            record.Id = Index;
            return record;
        }
    }
    else
    {
        return null;
    }
}

public void Reset()
{
    Index = 0;
    _infile.BaseStream.Position = 0;
}

```

```

public bool EndOfData
{
    get { return _infile.EndOfStream; }
}

public int Index { get; private set; }

IEnumerator IEnumerable.GetEnumerator()
{
    return this.GetEnumerator();
}

public IEnumerator<Record> GetEnumerator()
{
    Reset();
    var record = ReadRecord();
    while (record != null)
    {
        yield return record;
        record = ReadRecord();
    }
}
#endregion

#region IDisposable
public void Dispose()
{
    Dispose(true);
    GC.SuppressFinalize(this);
}

private bool m_Disposed = false;

protected virtual void Dispose(bool disposing)
{
    if (!m_Disposed)
    {
        if (disposing)
        {
            // Managed resources are released here.
            _infile.Close();
        }

        // Unmanaged resources are released here.
        m_Disposed = true;
    }
}

```



```

    }
}

~PlainTextReader()
{
    Dispose(false);
}
#endregion
}
}

/*****
* StringHelper.cs
*****/

using System;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    static class StringHelper
    {
        public static string[] Tokenize(string line, char[] delimiters)
        {
            Contract.Requires<ArgumentNullException>(line != null);
            Contract.Requires<ArgumentNullException>(delimiters != null);
            Contract.Requires<ArgumentException>(delimiters.Length > 0, "No delimiters
                specified.");

            // Strip comments (original; remove comments in this version)
            int index = line.IndexOf('%');
            if (index >= 0)
            {
                line = line.Remove(index);
            }
            // Replace tab characters
            line = line.Replace('\t', ' ');
            // Split string into tokens
            var tokens = line.Split(delimiters, StringSplitOptions.RemoveEmptyEntries);
            // Trim whitespaces in tokens
            tokens = tokens.Select(s => s.Trim()).ToArray();

            return tokens;
        }
    }
}

```

```

    }
}

/*****
* IScaling.cs
*****/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    public interface IScaling
    {
        void Scale(Record record);
        void Unscale(Record record);

        float[] Scale(float[] real);
        float[] Unscale(float[] real);
    }
}

/*****
* MinmaxScaling.cs
*****/

using System;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    /// <summary>
    /// Scales values by min and max value (from zero to one).
    /// </summary>
    public class MinmaxScaling : IScaling
    {
        private int _realFieldsCount;

        private float[] _min;
        private float[] _range;
    }
}

```

```

private IDataReader _data;

public MinmaxScaling(IDataReader data)
{
    _data = data;
    _realFieldsCount = data.Fields.RealCount();

    GetDataProperties();
}

/// <summary>
/// Calculates data properties (min and range).
/// </summary>
private void GetDataProperties()
{
    // initialize vectors
    _min = new float[_realFieldsCount];
    float[] max = new float[_realFieldsCount];
    _range = new float[_realFieldsCount];
    for (int i = 0; i < _realFieldsCount; i++)
    {
        _min[i] = float.MaxValue;
        max[i] = float.MinValue;
    }

    foreach (var record in _data)
    {
        for (int i = 0; i < _realFieldsCount; i++)
        {
            float value = record.Real[i];
            if (!float.IsNaN(value))
            {
                if (value < _min[i])
                    _min[i] = value;
                else if (value > max[i])
                    max[i] = value;
            }
        }
    }

    // calculate range
    for (int i = 0; i < _realFieldsCount; i++)
        _range[i] = max[i] - _min[i];
}

```

```

        _data.Reset();
    }

    public void Scale(Record record)
    {
        for (int i = 0; i < _realFieldsCount; i++)
            record.Real[i] = _range[i] == 0 ? 0 : (record.Real[i] - _min[i]) /
                _range[i];
    }

    public void Unscale(Record record)
    {
        for (int i = 0; i < _realFieldsCount; i++)
            record.Real[i] = record.Real[i] * _range[i] + _min[i];
    }

    public float[] Scale(float[] real)
    {
        Contract.Requires<ArgumentNullException>(real != null);
        Contract.Assert(real.Length == _realFieldsCount);

        float[] result = new float[_realFieldsCount];
        for (int i = 0; i < _realFieldsCount; i++)
            result[i] = _range[i] == 0 ? 0 : (real[i] - _min[i]) / _range[i];

        return result;
    }

    public float[] Unscale(float[] real)
    {
        Contract.Requires<ArgumentNullException>(real != null);
        Contract.Assert(real.Length == _realFieldsCount);

        float[] result = new float[_realFieldsCount];
        for (int i = 0; i < _realFieldsCount; i++)
            result[i] = real[i] * _range[i] + _min[i];

        return result;
    }
}

/*****
* StandardScaling.cs
*****/

```

```

using System;
using System.Collections.Generic;
using System.Diagnostics.Contracts;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis
{
    /// <summary>
    /// Standardize real values (scales to mean and standard deviations).
    /// </summary>
    public class StandardScaling : IScaling
    {
        private int _realFieldsCount;

        private float[] _mean;
        private float[] _std;

        private IDataReader _data;

        public StandardScaling(IDataReader data)
        {
            _data = data;
            _realFieldsCount = data.Fields.RealCount();

            GetDataProperties();
        }

        /// <summary>
        /// Calculates data properties (min and range).
        /// </summary>
        private void GetDataProperties()
        {
            // initialize vectors
            _mean = new float[_realFieldsCount];
            _std = new float[_realFieldsCount];
            double[] sumv = new double[_realFieldsCount];
            double[] sumsqv = new double[_realFieldsCount];
            int[] num = new int[_realFieldsCount];

            foreach (var record in _data)
            {
                for (int i = 0; i < _realFieldsCount; i++)
                {

```

```

        if (!float.IsNaN(record.Real[i]))
        {
            double r = ((double)record.Real[i]);
            sumv[i] += r;
            sumsqv[i] += r * r;
            num[i]++;
        }
    }

    for (int i = 0; i < _realFieldsCount; i++)
    {
        if (num[i] > 1)
        {
            double meanValue = sumv[i] / num[i];
            _mean[i] = double.IsNaN(meanValue) ? 0 : ((float)meanValue);

            double stdValue = Math.Sqrt((sumsqv[i] - sumv[i] * sumv[i] / num[i]) / (num[i] - 1));
            _std[i] = double.IsNaN(stdValue) ? 0 : ((float)stdValue);
        }
        else
        {
            _mean[i] = 0;
            _std[i] = 0;
        }
    }

    _data.Reset();
}

public void Scale(Record record)
{
    for (int i = 0; i < _realFieldsCount; i++)
        record.Real[i] = _std[i] == 0 ? 0 : (record.Real[i] - _mean[i]) / _std[i];
}

public void Unscale(Record record)
{
    for (int i = 0; i < _realFieldsCount; i++)
        record.Real[i] = record.Real[i] * _std[i] + _mean[i];
}

public float[] Scale(float[] real)

```

```

    {
        Contract.Requires<ArgumentNullException>(real != null);
        Contract.Assert(real.Length == _realFieldsCount);

        float[] result = new float[_realFieldsCount];
        for (int i = 0; i < _realFieldsCount; i++)
            result[i] = _std[i] == 0 ? 0 : (real[i] - _mean[i]) / _std[i];

        return result;
    }

    public float[] Unscale(float[] real)
    {
        Contract.Requires<ArgumentNullException>(real != null);
        Contract.Assert(real.Length == _realFieldsCount);

        float[] result = new float[_realFieldsCount];
        for (int i = 0; i < _realFieldsCount; i++)
            result[i] = real[i] * _std[i] + _mean[i];

        return result;
    }
}

/*****
 * NeighborsDistance.cs
 *****/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Thesis.Collections;

namespace Thesis.Orca
{
    struct NeighborsDistance
    {
        public Record Record { get; set; }
        public BinaryHeap<double> Distances { get; set; }
    }
}

/*****/

```

```

* OrcaAD.cs
*****/

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics.Contracts;
using Thesis.Collections;
using System.Diagnostics;

namespace Thesis.Orca
{
    public class OrcaAD
    {
        private ScoreFunction _scoreFunction;
        private DistanceMetric _distanceFunction;
        private int _numOutliers;
        private int _neighborsCount;
        private double _cutoff;
        private int _batchSize;

        #region Constructors
        /// <param name="distanceFunction">Distance function for calculating the
        /// distance between two examples with weights.</param>
        /// <param name="numOutliers">Number of outliers.</param>
        /// <param name="neighborsCount"></param>
        /// <param name="cutoff"></param>
        /// <param name="batchSize"></param>
        public OrcaAD(DistanceMetric distanceFunction,
            int numOutliers = 30, int neighborsCount = 5,
            double cutoff = 0, int batchSize = 1000)
            : this(ScoreFunctions.Average, distanceFunction, numOutliers,
                neighborsCount, cutoff, batchSize)
        { }

        /// <param name="scoreFunction">Distance score function.</param>
        /// <param name="numOutliers">Number of outliers.</param>
        /// <param name="neighborsCount"></param>
        /// <param name="cutoff"></param>
        /// <param name="batchSize"></param>
        public OrcaAD(ScoreFunction scoreFunction,
            int numOutliers = 30, int neighborsCount = 5,
            double cutoff = 0, int batchSize = 1000)
            : this(scoreFunction, DistanceMetrics.cSqrEulid, numOutliers,

```



```

        neighborsCount, cutoff, batchSize)
    { }

    /// <param name="numOutliers">Number of outliers.</param>
    /// <param name="neighborsCount"></param>
    /// <param name="cutoff"></param>
    /// <param name="batchSize"></param>
    public OrcaAD(int numOutliers = 30, int neighborsCount = 5,
        double cutoff = 0, int batchSize = 1000)
        : this(ScoreFunctions.Average, DistanceMetrics.cSqrEulid, numOutliers,
            neighborsCount, cutoff, batchSize)
    { }

    /// <param name="scoreFunction">Distance score function.</param>
    /// <param name="distanceFunction">Distance function for calculating the
        distance between two examples with weights.</param>
    /// <param name="numOutliers">Number of outliers.</param>
    /// <param name="neighborsCount"></param>
    /// <param name="cutoff"></param>
    /// <param name="batchSize"></param>
    public OrcaAD(ScoreFunction scoreFunction,
        DistanceMetric distanceFunction,
        int numOutliers = 30, int neighborsCount = 5,
        double cutoff = 0, int batchSize = 1000)
    {
        Contract.Requires<ArgumentNullException>(scoreFunction != null);
        Contract.Requires<ArgumentNullException>(distanceFunction != null);
        Contract.Requires<ArgumentOutOfRangeException>(numOutliers > 0);
        Contract.Requires<ArgumentOutOfRangeException>(neighborsCount > 0);
        Contract.Requires<ArgumentOutOfRangeException>(cutoff >= 0);
        Contract.Requires<ArgumentOutOfRangeException>(batchSize > 0);

        _scoreFunction = scoreFunction;
        _distanceFunction = distanceFunction;
        _numOutliers = numOutliers;
        _neighborsCount = neighborsCount;
        _cutoff = cutoff;
        _batchSize = batchSize;
    }
    #endregion

    /// <param name="cases">Data reader for input data.</param>
    /// <param name="references">Data reader for reference data (can't be the same
        reader object).</param>
    /// <param name="returnAll">If true, returns score info for all records in input

```

```

        data.</param>
    /// <returns></returns>
    public IEnumerable<Outlier> Run(IDataReader cases, IDataReader references, bool
        returnAll = false)
    {
        Contract.Requires<ArgumentNullException>(cases != null);
        Contract.Requires<ArgumentNullException>(references != null);
        Contract.Requires<ArgumentException>(!object.ReferenceEquals(cases,
            references));

        // Test cases
        using (BatchDataReader batchInFile = new BatchDataReader(cases, _batchSize))
        // Reference database
        {
            List<Outlier> outliers = new List<Outlier>();
            bool done = false;
            double cutoff = _cutoff;
            Weights weights = cases.Fields.Weights();

            //-----
            // run the outlier search
            //
            done = !batchInFile.GetNextBatch(); //start batch
            while (!done)
            {
                Trace.PrintRecords(batchInFile.CurrentBatch);

                var o = FindOutliers(batchInFile, references, weights, cutoff);
                outliers.AddRange(o);

                references.Reset();

                //-----
                // sort the current best outliers
                // and keep the best
                //
                outliers.Sort();
                outliers.Reverse(); // sorting in descending order
                int numOutliers = _numOutliers;
                if (outliers.Count > numOutliers &&
                    outliers[numOutliers - 1].Score > cutoff)
                {
                    // New cutoff
                    cutoff = outliers[numOutliers - 1].Score;
                }
            }
        }
    }

```

```

        done = !batchInFile.GetNextBatch();
    }

    return returnAll ? outliers : outliers.Take(_numOutliers);
}

}

private IList<Outlier> FindOutliers(BatchDataReader cases, IDataReader
    references, Weights weights, double cutoff)
{
    Contract.Requires<ArgumentNullException>(cases != null);
    Contract.Requires<ArgumentNullException>(references != null);

    int k = _neighborsCount; // number of neighbors

    var records = new List<Record>(cases.CurrentBatch);
    int batchRecCount = records.Count;

    // distance to neighbors – Neighbors(b) in original description
    var neighborsDist = new List<NeighborsDistance>(batchRecCount);
    // initialize distance score with max distance
    for (int i = 0; i < batchRecCount; i++)
    {
        var kDistDim = new NeighborsDistance()
        {
            Record = records[i],
            Distances = new BinaryHeap<double>(k)
        };
        for (int j = 0; j < k; j++)
            kDistDim.Distances.Push(double.MaxValue);
        neighborsDist.Add(kDistDim);
    }

    // vector to store furthest nearest neighbour
    var minkDist = new List<double>(batchRecCount);
    for (int i = 0; i < neighborsDist.Count; i++)
        minkDist.Add(double.MaxValue);

    // candidates stores the integer index
    var candidates = Enumerable.Range(0, batchRecCount).ToList();

    int neighborsDist_i;
    int minkDist_i;
    int candidates_i;

```

```

// loop over objects in reference table
foreach (var descRecord in references)
{
    neighborsDist_i = 0;
    minkDist_i = 0;
    candidates_i = 0;

    for (int j = 0; j < batchRecCount; j++)
    {
        double dist = _distanceFunction(records[j], descRecord, weights);

        if (dist < minkDist[minkDist_i])
        {
            if (cases.Offset + candidates[candidates_i] != references.Index
                - 1)
            {
                BinaryHeap<double> kvec = neighborsDist[neighborsDist_i].
                    Distances;
                kvec.Push(dist);
                kvec.Pop();
                minkDist[minkDist_i] = kvec.Peek();

                double score = _scoreFunction(kvec);

                if (score <= cutoff)
                {
                    candidates.RemoveAt(candidates_i--);
                    records.RemoveAt(j--); batchRecCount--;
                    neighborsDist.RemoveAt(neighborsDist_i--);
                    minkDist.RemoveAt(minkDist_i--);

                    if (candidates.Count == 0)
                        break;
                }
            }
        }

        neighborsDist_i++;
        minkDist_i++;
        candidates_i++;
    }

    if (candidates.Count == 0)
        break;
}

```



```

using System.Text;
using System.Threading.Tasks;

namespace Thesis.Orca
{
    public static class ScoreFunctions
    {
        private static readonly ScoreFunction _average = new ScoreFunction(dist => dist.
            Sum() / dist.Count());
        private static readonly ScoreFunction _sum = new ScoreFunction(dist => dist.Sum
            ());
        private static readonly ScoreFunction _kthNeighbor = new ScoreFunction(dist =>
            dist.FirstOrDefault());

        /// <summary>
        /// Average distance to k neighbors.
        /// </summary>
        public static ScoreFunction Average
        {
            get { return _average; }
        }

        /// <summary>
        /// Total distance to k neighbors (sum).
        /// </summary>
        public static ScoreFunction Sum
        {
            get { return _sum; }
        }

        /// <summary>
        /// Distance to kth neighbor.
        /// </summary>
        public static ScoreFunction KthNeighbor
        {
            get { return _kthNeighbor; }
        }
    }
}

/*****
 * Trace.cs
 *****/
using System;
using System.Collections.Generic;

```

```

using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Thesis.Orca
{
    static class Trace
    {
        [Conditional("DEBUG")]
        public static void PrintRecords(IEnumerable<Record> records)
        {
            Console.WriteLine("—————TRACE—————");
            IEnumerable<Record> pRecords = records.Count() <= 10 ? records :
                records.Take(10);
            foreach (var record in pRecords)
            {
                Console.Write("#{0}:␣", record.Id);
                foreach (var real in record.Real)
                    Console.Write("{0}␣", real);
                Console.Write("|␣");
                foreach (var discrete in record.Discrete)
                    Console.Write("{0}␣", discrete);
                Console.WriteLine();
            }
            Console.WriteLine("—————END␣TRACE—————");
            Console.WriteLine();
        }

        [Conditional("DEBUG")]
        public static void Message(string message)
        {
            Console.WriteLine("TRACE:␣" + message);
        }
    }
}

```