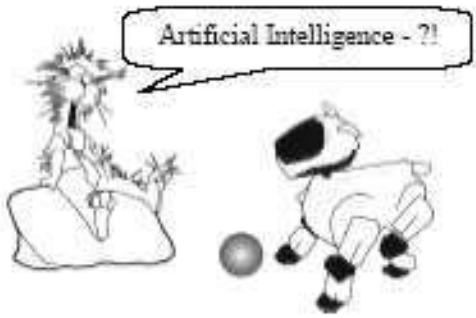
Родзин С.И.
Гребенюк Е.Ю.
Злыгостев А.С.
Шишков С.А.

СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

ЛАБОРАТОРНЫЙ ПРАКТИКУМ



МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Технологический институт
Федерального государственного образовательного учреждения высшего профессионального образования
«Южный федеральный университет»

Родзин С.И., Гребенюк Е.Ю., Злыгостев А.С., Шишков С.А.

Системы искусственного интеллекта
Лабораторный практикум

Учебное пособие

Для студентов направления «Информатика и вычислительная техника», специальностей «Математическое обеспечение и администрирование информационных систем». «Программное обеспечение вычислительной техники и автоматизированных систем»

Таганрог 2007

УДК 007. 52: 611.81 (075.8) + 519.7 (075.8)

Рецензенты:

Кафедра систем автоматизированного проектирования и поискового конструирования Волгоградского государственного технического университета, зав. кафедрой, доктор технических наук, профессор *Колаев В.А.*

Доктор технических наук, профессор Ростовского государственного университета путей сообщения *Ковалев С.М.*

Родзин С.И., Гребенюк Е.Ю., Злыгостев А.С., Шишков С.А. **Системы искусственного интеллекта. Лабораторный практикум.** Учебное пособие. – Таганрог: Изд-во ТТИ ЮФУ, 2007. – 128 с.

Интеллектуальные модели, методы и технологии, добавление которых в существующие системы различного назначения позволяет придавать им различные «разумные» свойства, являются перспективной междисциплинарной областью инженерии знаний. Цикл из 5 лабораторных работ охватывает вопросы продукционного и фреймового программирования на языке CLIPS, задачи «мягких вычислений» (нечеткие и нейронные модели представления объектов и ситуаций) в среде MatLab, а также проблемы проектирования интеллектуальных систем информационного поиска. Содержание учебного пособия основано на материалах и программных продуктах, используемых в образовательном процессе кафедры МОП ЭВМ в цикле лабораторных работ. В практикуме приведены как оригинальные задачи[1], так и заимствованные из специальной и учебной литературы.

Для студентов, изучающих дисциплину «Системы искусственного интеллекта», а также специализирующихся в области интеллектуальных информационных технологий.

Библиогр.: 12 назв.

© С.И. Родзин, Е.Ю. Гребенюк, А.С. Злыгостев, С.А. Шишков, 2007
© ТТИ ЮФУ, 2007

---

[1] Работа выполнена при поддержке гранта РФФИ 2007 г.

СОДЕРЖАНИЕ

Введение

В интеллектуализации сегодня нуждается практически любое рабочее место в составе автоматизированных информационных систем. Однако продуктивных прикладных разработок в области интеллектуализации без теоретической базы не бывает. Такой теоретической базой является работа со знаниями: модели, средства и системы представления знаний; методы поиска и обработки знаний. Именно на этой основе базируются прикладные разработки в области инженерии знаний, которые ведутся по следующим направлениям:

- решение отдельных интеллектуальных задач (поиск, семантический анализ и обработка информации на естественном языке, машинный перевод и реферирование, распознавание образов, моделирование поведения бионических систем, игры и т.д.);
- интеллектуальное программирование (языки представления знаний, семантической разметки, многоагентного взаимодействия и т.п.), инструментальные средства для автоматического синтеза программ; интеллектуальный интерфейс;
- проектирование и разработка интеллектуальных систем (экспертные системы, интеллектуальные АСУ, САПР, системы управления знаниями и поддержки принятия решений, обучающие и консультирующие системы, системы виртуальной реальности, нейропакеты и интеллектуальные роботы).

Эти направления исследований определили содержание данного лабораторного практикума, который основан на материалах, используемых в учебном процессе кафедры МОП ЭВМ в ходе профессиональной подготовки специалистов по профилю, связанному с математическим и программным обеспечением вычислительной техники, а также с администрированием информационных систем.

Ожидаемые результаты (задачи) изучения систем искусственного интеллекта в образовательных программах профильной подготовки состоят в следующем:

- систематизация и интеграция ранее приобретенных математических знаний, общепрофессиональных умений и навыков для автоматизации сложноформализуемых задач, считающихся прерогативой человека;
- способность применять знания математики и информатики для решения задач искусственного интеллекта, а также умение идентифицировать, формулировать задачи поиска, представления и вывода знаний, анализировать и объяснять полученные результаты.

Основными целями лабораторного цикла по дисциплине являются:

- изучение продукционной и фреймовой моделей представления знаний, механизмов прямого и обратного логического вывода, а также их практической реализации на языке программирования интеллектуальных систем CLIPS;
- умение построить интеллектуальную модель управления средствами нечеткой логики, а также изучение возможностей по моделированию искусственных нейронных сетей в интерактивной среде MatLab;
- ознакомление с основными методами поиска в коллекциях текстовой информации, практическая реализация поиска на базе подготовленного индекса поисковой машины, сравнительный анализ поисковых моделей.

Практикум включает цикл из 5 лабораторных работ. В соответствии с учебным планом на подготовку, выполнение и защиту каждой лабораторной работы выделяется от 6 до 8 часов аудиторных занятий и самостоятельной работы студента в зависимости от сложности изучаемой темы. Трудоемкость выполнения цикла работ равна одной зачетной единице, трудоемкость выполнения каждой работы – от 0,2 до 0,3 зачетных единиц. Содержание каждой работы состоит из краткого теоретического описания, которое

иллюстрируется рассмотрением конкретной задачи, а также технического задания на выполнение индивидуальных вариантов работы и вопросов для самопроверки.

Оценка результатов выполнения лабораторной работы осуществляется по трем критериям, которые отражают наиболее значимые аспекты контроля этого вида самостоятельной работы студентов. Все критерии имеют одинаковый «вес»: 2 балла – критерий полностью выполнен, 1 балл – имеются отступления от критерия, 0 баллов – критерий не выполнен. Ниже приводятся формулировки степени полного выполнения каждого критерия, что соответствует максимальному баллу 6.

Критерий 1. Цели и задачи лабораторной работы чётко сформулированы, структура отчета полностью соответствует установленным требованиям. Отчет оформлен аккуратно, в полном соответствии с ГОСТам и сдан в срок.

Критерий 2. Использованный стиль описания полностью соответствует характеру самостоятельно выполненной работы, чёткая логика рассуждений, даны правильные и точные ответы на вопросы в ходе защиты.

Критерий 3. При выполнении работы проявлен творческий подход и умение применять теоретические знания на практике, работать с информационными источниками. Выводы имеют характер нетривиальных утверждений, содержат обоснованную оценку степени достижения поставленных целей и задач, не подменяются перечислением сделанного, поддерживаются программной реализацией и экспериментальными данными, не содержат мотивации (нет фраз типа «анализ показал, что…,», «из изложенного следует…» и др.).

Итоговая оценка результата работы по выполнению индивидуального задания определяется как общая сумма баллов, в которые оценено соблюдение каждого из трех критериев.

В ходе самостоятельного выполнения цикла лабораторных работ рекомендуется использовать литературу, которая приводится в библиографическом списке. Основной литературой для выполнения первой и второй работ является [1, 2, 4, 9, 10], для третьей – [6, 8, 11, 12], для четвертой – [5, 6, 8, 11, 12], для пятой – [3, 7].

Самостоятельно изучив основные конструкции языка CLIPS, интерактивную систему MatLab, экспериментируя и создавая модели представления знаний, программируя и сравнивая их поведение, у студентов, выполнивших цикл работ, имеется возможность «контекстно» соотнести конкретные модели знаний со сферами их возможного применения в профессиональной области, ассоциировать свой собственный опыт с предметом искусственного интеллекта. Это позволяет сфокусировать внимание на анализе и разрешении многих конкретных проблемных ситуаций в области искусственного интеллекта, что становится отправной точкой в процессе курсового и дипломного проектирования.

Лабораторная работа №1

1. Программирование продукционных систем на языке CLIPS

Целью работы является изучение языка программирования CLIPS, как одного из распространенных инструментальных средств разработки экспертных систем, анализ продукционной модели представления знаний, механизмов прямого и обратного логического вывода, а также практическая реализация модели и механизмов вывода на языке CLIPS.

1.1. Краткое теоретическое описание

Декларативные модели знаний обычно подразделяются на продукционные, сетевые (семантические сети) и фреймовые. Одной из наиболее простых и эффективных моделей для представления знаний и описания интеллектуальных задач является продукционная модель или продукционная система. Продукционные системы привлекают разработчиков своей наглядностью, модульностью, легкостью внесения дополнений, простотой механизма логического вывода.

Согласно Посту продукционная (каноническая) система включает алфавит, из символов которого формируются строки, множество строк, которые рассматриваются как аксиомы, а также множество грамматических правил манипулирования строками символов (правила порождений). Система должна обладать разрешимостью, непротиворечивостью и полнотой. На первый взгляд продукционные системы довольно тривиальны, поскольку речь идет только о преобразовании одной строки символов в другую. Но если вдуматься, то любое математическое или логическое исчисление, в конце концов, сводится к набору правил манипулирования символами (гипотеза

Ньюэлла: необходимое и достаточное условие интеллектуальности системы – универсальность формальных манипуляций над конкретными символами). Иными словами, достаточно прочесть строку символов, разделить ее на компоненты и переупорядочить, добавив или удалив какие-то символы. Однако для нас важен смысл (семантика) логических и математических символов.

В продукционной системе знания представляются совокупностью правил вывода вида «ЕСЛИ (условия-посылки) ТО (действия-заключения)». Различают продукционные системы с прямым и обратным логическими выводами. Прямой вывод означает, что рассуждения строятся, отталкиваясь от условий (они должны удовлетворяться), к действиям (заключениям), вытекающим из этих условий. Обратный вывод означает, что рассуждения строятся, отталкиваясь от заданной цели, к условиям, при которых возможно ее достижение. Типичными представителями прямого вывода являются системы, используемые для решения задач диагностического характера, а типичными представителями систем обратного вывода – системы, используемые для решения задач проектирования.

В продукционных системах принято различать три компонента: *базу правил*, состоящую из набора правил вывода (продукций), *рабочую память* (база данных, которая определяет текущее состояние задачи и содержит множество фактов, описание цели и промежуточные результаты) и *интерпретатор правил*, который осуществляет логический вывод на основании фактов и решает, когда надлежит применить каждое из правил. Иными словами, база правил и база данных образуют *базу знаний*, а интерпретатор соответствует *механизму логического вывода*. Сильными сторонами продукционных систем является простота правил, возможность их модификации, а слабые стороны – неясность взаимных отношений правил, сложность оценки целостного образа знаний при большом числе правил; отсутствие гибкости в логическом выводе.

Продукционная система из трех основных компонентов схематично представлена на рис. 1.1.

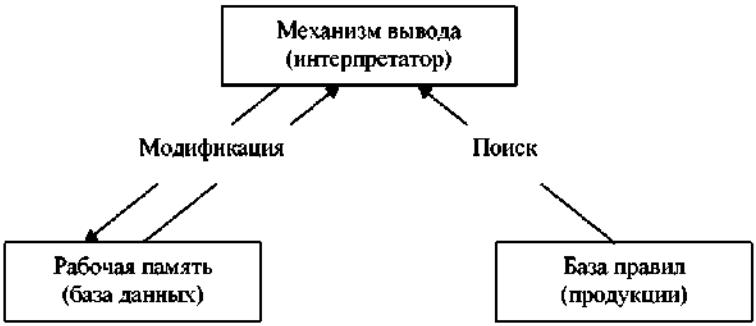


Рис. 1.1. Блок-схема продукционной системы

Рассмотрим компоненты продукционной системы более подробно. Чтобы правило было выполнимым, необходимо (но не достаточно), чтобы выполнялись все его условия. Условие выполняется, если соответствующий ему факт присутствует в списке фактов рабочей памяти. После определения, какие из правил являются выполнимыми, они помещаются в список активированных правил. Наконец, если какое-то одно правило из списка будет выполнено, действия задают изменения, которые должны быть внесены в состояние рабочей памяти. Функция рабочей памяти – хранить данные (факты) в формате векторов «объект – атрибут – значение атрибута». Эти данные используются интерпретатором, который активизирует те правила, условия которых на имеющихся данных выполняются. Наконец, процесс работы интерпретатора правил описывается в терминах сопоставления по образцу (цикл «распознавание – действие»):

1. Сопоставить условия правил и элементы рабочей памяти;
2. Если окажется, что можно активизировать более одного правила, то выбрать одно из них (*разрешить конфликт*);

3. Применить выбранное правило. Результатом, скорее всего, будет добавление нового элемента данных в рабочую память и/или удаление каких-либо данных из рабочей памяти. Затем перейти к п.1.

Отметим также, что при всем разнообразии способов разрешения конфликтов они используют следующие принципы:

- *разнообразие* (не следует применять правило, которое уже активировалось ранее, к одним и тем же данным, например, путем удаления его из конфликтующего множества правил; однако при необходимости повторения правила программист может воспользоваться функцией *refresh*);
- *новизна* (например, в CLIPS элементы рабочей памяти снабжаются атрибутом времени их порождения и приоритет отдается правилам, «реагирующим» на более «свежие» данные);
- *специфика* (правила считаются более специфичными, если включают большее число условий и поэтому их труднее удовлетворить).

В интерпретаторе CLIPS используются все три принципа, реализуемых с помощью следующих стратегий разрешения конфликтов:

- *стратегия глубины* (правила на основе данных, недавно включенных в рабочую память, имеют в конфликтующем множестве правил высший приоритет), которая реализована в CLIPS по умолчанию;
- *стратегия ширины* (правила на основе данных, давно включенных в рабочую память, имеют в конфликтующем множестве правил высший приоритет);
- *стратегия простоты* (определяется сложность каждого из конфликтующих правил по числу операций проверки их условных частей и приоритет отдается более простым правилам);
- *стратегия сложности* (определяется сложность каждого из конфликтующих правил по числу операций проверки их условных частей и приоритет отдается более сложным правилам);
- *LEX-стратегия* (из конфликтующего множества удаляются ранее использованные правила, остальные сортируются по «новизне» данных);

• *MEA-стратегия* (аналогична LEX-стратегии, но при анализе «новизны» принимаются во внимание только первые условия конкурирующих правил);

• свойство *выпуклости (salience)* (предпочтение отдается тому правилу, которое характеризуется большим значением выпуклости, по умолчанию любое правило имеет нулевое значение выпуклости, выпуклость может иметь и отрицательное значение).

Рассмотрим особенности продукционного программирования на CLIPS. Язык CLIPS (C Language Integrated Production System), являясь LIPS-подобным языком программирования, ориентированным на разработку экспертных систем, также поддерживает объектно-ориентированную и процедурную парадигмы программирования. Подробно с описанием языка можно ознакомиться в [2, 10], а также в электронном приложении к данному лабораторному практикуму. Версии языка могут эксплуатироваться на платформах UNIX, Windows, Macintosh, являются хорошо документированным общедоступным программным продуктом, доступным по сети FTP с множества университетских сайтов. Исходный код программного пакета CLIPS распространяется свободно и его можно установить на любой платформе, поддерживающей стандартный компилятор языка С. Поэтому здесь приведем лишь сведения, необходимые для общего понимания конструкций языка, используемых в данном кратком теоретическом описании лабораторной работы.

Для программирования язык использует простые типы данных (float, integer, symbol, string и др.), функции для манипулирования данными с префиксной формой их вызова (пользовательские фрагменты кода на языке С, стандартные арифметические и математические функции) и конструкции для пополнения базы знаний (defmodule, defrule, deffacts, deftemplate, deffunction, defclass и др.). Комментарии могут вставляться в код CLIPS при помощи точки с запятой «;». Факт является одной из основных форм представления информации, используемой правилами CLIPS. Каждый факт - это фрагмент информации, который помещается в текущий список фактов, называемый fact-list. Количество фактов и объем информации, который может быть сохранен в факте, ограничивается только размером памяти компьютера. Факт может описываться индексом или адресом и иметь позиционный (скобочная запись из выражений

символьного типа) и непозиционный (шаблон, который может использоваться при доступе к полям по именам) форматы представления. Шаблоны в CLIPS напоминают простые списки, но не имеют ничего общего с шаблонами в языке C++.

Как и в других языках программирования, в CLIPS для хранения значений используются переменные. В отличие от фактов, которые являются статическими и неизменными, содержание переменной может динамично изменяться путем присвоения ей новых значений. Идентификатор переменной всегда начинается с вопросительного знака, за которым следует ее имя. Кроме того, CLIPS предоставляет ряд дополнительных средств, необходимых при написании программ. Основными из них являются ограничения на значения полей, оператор проверки условия test (мощное средство для сравнения чисел, переменных и строк в левой части правила), использование функций в правилах, использование процедурных знаний.

Интерфейс CLIPS позволяет работать интерактивно с использованием GUI или простого текстового интерфейса командной строки, а также как система, интегрированная в другие приложения. После запуска файла clipswin.exe можно ввести программу непосредственно из диалогового окна, но в этом случае все подготовленные правила после закрытия CLIPS будут потеряны. Чтобы этого не происходило, необходимо сохранить текст программы в каком-либо файле. Для его модификации имеется встроенный редактор.

Итак, продукционные правила являются одной из основных моделей представления знаний в CLIPS. Правила служат для представления экспертных эвристик и определяют действия, которые необходимо выполнить в определенной ситуации. Разработчик интеллектуальной системы определяет совокупность правил, которые совместно используются для решения проблемы. Правило на CLIPS состоит из антецедента (условия) и записывается слева, а также консеквента (заключения), который записывается справа.

Левая часть правила представляет собой ряд условий (условных элементов), которые должны выполняться, чтобы правило было применимо. В CLIPS принято считать, что условие выполняется, если соответствующий ему факт присутствует в списке фактов. Одним из типов условных элементов может быть образец. Образцы состоят из набора ограничений, которые используются для описания того, какие

факты удовлетворяют условию, определяемому образцом. Процесс сопоставления фактов образцам выполняется блоком вывода CLIPS, который автоматически сопоставляет образцы, исходя из текущего состояния списка фактов, и определяет, какие из правил являются применимыми. Если все условия правила выполняются, то оно активируется и помещается в список активированных правил. Правила похожи на операторы типа if-then в процедурных языках программирования. Однако условие if-then оператора в процедурном языке проверяется только тогда, когда программа передает ему управление. С правилами на CLIPS ситуация иная. Интерпретатор постоянно отслеживает все правила, условия которых выполняются, и, таким образом, правило может быть выполнено в любой момент, как только оно становится активным.

В продукционном программировании правила реализуются в виде конструкций, манипулирующих структурами типа векторов, а не строк символов. Это влияние языков типа LISP, CLIPS и тех структур данных, которые они поддерживают. В языке CLIPS условия представляются в форме вектора *объект-атрибут-значение*. Например: (organism-1 (morphology rod) (aerobicity aerobic)). В данном случае условие состоит в том, что определенный микроорганизм имеет форму палочки и размножается в воздушной среде.

Продукционное правило, которое включает такую предпосылку, на языке CLIPS имеет вид:

```
(defrule diagnosis
      (patient (name Jones) (organism organism1))
      (organism (name organism1) (morphology rod)
      (aerobicity aerobic))
=>
      (assert (organism (name organism1) (identify
      enterobacteriaceae) (confidence 0.8))
)
```

Перечень предпосылок в таком правиле представляет собой образец вектора, которому должно соответствовать состояние рабочей памяти. Действия, такие как assert, modify, retract, задают изменения, которые должны быть внесены в состояние рабочей памяти. Например, специфицированное в приведенном выше правиле действие добавит в рабочую память новый вектор

(organism (name organism1) (identify
enterobacteriaceae) (confidence 0.8))

Другим компонентом продукционной системы является рабочая память, в которой хранятся исходные данные к задаче и выводы, полученные в ходе работы системы. Эти данные используются интерпретатором, который в случае присутствия (или отсутствия) определенных данных в рабочей памяти может активизировать те правила, предпосылки в которых соответствуют этим данным. Рассмотрим на примере, как это реализуется в CLIPS.

Пусть в рабочей памяти содержатся векторы:

(patient (name Jones) (age 40) (organism organism1))
(organism (name organism1) (morphology rod)
(aerobicity aerobic))

На очередном шаге цикла интерпретатор просмотрит имеющийся список правил и отыщет в нем то правило, которое содержит условия, соответствующие этим векторам. Если предпосылка в правиле не содержит переменных, она удовлетворяется при точном совпадении выражений в правиле и в рабочей памяти. Если же предпосылка в правиле содержит переменные, то есть является образцом, то она удовлетворяется, если в рабочей памяти содержится вектор, включающий такую пару *атрибут-значение*, которая остается постоянной при удовлетворении всех остальных условий в том же правиле.

В самом простом случае соответствие проверяется присвоением постоянных значений переменным, которые делают предпосылку, совпадающей с вектором в рабочей памяти. Так, вектор состояния в рабочей памяти

(patient (name Jones) (age 40) (organism organism1))

удовлетворяет предпосылку в правиле

(patient (name ?pat) (organism ?org))

подстановкой Jones вместо ?pat и organism1 вместо ?org

Обратите внимание на то, что при анализе соответствия не рассматриваются пары, которые отсутствуют в предпосылке правила. Поскольку другая предпосылка в этом же правиле также удовлетворяется при указанной подстановке, то новый вектор

(organism (name organism1) (identify
enterobacteriaceae) (confidence 0.8))

добавляется интерпретатором в рабочую память. Для заключения данного правила значение ?pat безразлично, поле имени пациента можно в условии вообще игнорировать.

Последний компонент продукционной системы – механизм логического вывода – использует правила в соответствии с содержимым рабочей памяти и действует путем «сопоставления по образцу», управляя перебором правил в прямом (от данных к поиску цели) или обратном (от цели для ее подтверждения – к данным) направлении. В процессе вывода в продукционной системе может возникнуть ситуация, когда на каком-либо шаге удовлетворяются условия применимости нескольких продукций. Тогда для выбора выполняемого правила привлекается информация о приоритетности, достоверности, значимости и прочих свойствах продукций.

Приведем примеры прямого и обратного логического вывода в конструкциях языка CLIPS, которые более подробно показывают работу продукционной системы через пошаговое изменение содержимого рабочей памяти.

1.2. Пример выполнения прямого вывода

Задача: *управление роботом для перемещения объекта «ящик» из точки «B» в точку «A».*

Ниже представлен набор правил вместе с множеством векторов в рабочей памяти. CLIPS-программа состоит из выражений трех типов:

- *шаблонов* или *деклараций*, которые определяют формат векторов в рабочей памяти;
- *фактов*, которые задают начальное состояние задачи;
- *порождающих правил*, которые определяют возможные трансформации состояния задачи.

```
;; ШАБЛОНЫ:
;; Цель (goal) - вектор, состоящий из четырех
компонентов:
;; действие, которое нужно выполнить,
;; объект, над которым должно быть выполнено
действие;
;; исходное положение;
;; положение, в которое нужно перейти.
(deftemplate goal
```

```
        (field action (type SYMBOL))
        (field object (type SYMBOL))
        (field from (type SYMBOL))
        (field to (type SYMBOL))
)
;;Вектор 'in' указывает, где находится объект.
(deftemplate in
        (field object (type SYMBOL))
        (field location (type SYMBOL))
)
;; ФАКТЫ
;; Функция 'deffacts' вводит в рабочую память
описание
;;исходного положения и вызывается при перезапуске
системы.
;; Исходное состояние объектов следующее:
;; робот находится в комнате A;
;; ящик находится в комнате B.
;; Цель - переместить ящик в комнату A.
(deffacts world
        (in (object robot) (location RoomA))
        (in (object box) (location RoomB))
        (goal (action push) (object box) (from RoomB) (to
        RoomA))
)
;; ПРАВИЛА:
;; 1. Прекратить процесс, когда цель будет
достигнута.
(defrule stop
        (goal (object ?X) (to ?Y))
        (in (object ?X) (location ?Y))
=>
        (halt)
)
;; 2. Если робот отсутствует в той комнате, где
находится
;; ящик, который нужно передвинуть, то переместить
туда робот.
```

```
(defrule move
        (goal (object ?X) (from ?Y))
        (in (object ?X) (location ?Y))
        ?robot-position <- (in (object robot) (location
        ?Z&~?Y))
=>
        (modify ?robot-position (location ?Y))
)
;; 3. Если робот и объект находятся не в той комнате,
которая
;; указана в цели, то переместить туда робот вместе с
объектом
(defrule push
        (goal (object ?X) (from ?Y) (to ?Z))
        (in (object ?X) (location ?Y))
        ?object-position <- (in (object ?X) (location ?Y))
        ?robot-position <- (in (object robot) (location ?Y))
=>
        (modify ?robot-position (location ?Z))
        (modify ?object-position (location ?Z))
)
```

Приведем пошаговое описание программы для случая, когда робот находится в комнате A, а ящик – в комнате B:

```
- 0 ----------------------------------
(in (object robot) (location RoomA))
(in (object box) (location RoomB))
(goal (action push) (object box) (from RoomB) (to
RoomA))
```

Поскольку робот и ящик находятся в разных комнатах, то робот по правилу 2 (move) перемещается в комнату, где находится ящик. Выполнение правила move приводит к изменению состояния рабочей памяти:

```
- 1 ----------------------------------
(in (object robot) (location RoomB))
(in (object box) (location RoomB))
(goal (action push) (object box) (from RoomB) (to
RoomA))
```

Состояние рабочей памяти позволяет системе выбрать правило 3 (push), чтобы робот переместил ящик в целевое положение (комната A). Выполнение правила push приводит к изменению состояния рабочей памяти:

```
- 2 ----------------------------------
(in (object robot) (location RoomA))
(in (object box) (location RoomA))
(goal (action push) (object box) (from RoomB) (to
RoomA))
```

Факты в рабочей памяти соответствуют условной части правила 1 (stop), после выполнения которого процесс будет прекращен, так как цель достигнута.

Эту программу на языке CLIPS 6.1 можно выполнить, для чего необходимо ввести в систему текст этой программы и запустить ее на выполнение командами **reset** и **run**.

С помощью команды **watch** можно просмотреть, в каком порядке будут активизироваться специфицированные в программе правила.

1.3. Пример выполнения обратного вывода

Задача: построение пирамиды из блоков.

Приведем краткое описание спецификаций программы. Будем использовать стратегию разрешения конфликтов MEA (она более подходит для решения таких задач, как планирование). Программа должна чередовать выполнение двух целей: выбор очередного блока и установка блока на пирамиду. Шаблоны блоков представлены объектами, обладающими такими атрибутами: как цвет, размер и положение. Если положение блока не определено, предполагается, что он находится в куче блоков (*heap*), еще не установленных на пирамиду. Шаблон *on* описывает размещение одного блока (*upper*) на другом (*lower*). Информацию о текущем состоянии задачи (поиск блока или его установка) дает шаблон *goal*. Порядок перечисления правил в программе не имеет значения. Например, в предыдущей задаче управления роботом правило *stop* стояло в списке первым, а в этой программе оно будет стоять в самом конце списка правил.

```
;; СТРАТЕГИЯ РАЗРЕШЕНИЯ КОНФЛИКТОВ:
(declare (strategy mea))
;; ШАБЛОНЫ:
```

```
;; Объект block характеризуется цветом, размером,
положением
(deftemplate block
        (field color (type SYMBOL))
        (field size (type INTEGER))
        (field place (type SYMBOL) (default heap))
)
;; Вектор 'on' указывает, что блок <upper> находится
на <lower>
(deftemplate on
        (field upper (type SYMBOL))
        (field lower (type SYMBOL))
        (field place (type SYMBOL) (default heap))
)
;; Текущая цель (goal): найти (find) очередной блок
;; либо установить (build) блок на пирамиду
(deftemplate goal
        (field task (type SYMBOL))
)
;; ИНИЦИАЛИЗАЦИЯ:
;; В куче имеются три блока разных цветов и размеров
(deffacts the-facts
        (block (color red) (size 10))
        (block (color yellow) (size 20))
        (block (color blue) (size 30))
)
;; ПРАВИЛА:
;; 1. Задание цели: поиск первого блока
(defrule begin
        (initial-fact)
=>
        (assert (goal (task find)))
)
;; 2. Выбор самого большого блока в куче (heap)
(defrule pick-up
        ?my-goal <- (goal (task find))
        ?my-block <- (block (size ?S1) (place heap))
```

```
        (not (block (color ?C2) (size ?S2&:(> ?S2 ?S1))
(place heap)))
=>
        (modify ?my-block (place hand))
        (modify ?my-goal (task build))
)
;; 3. Установка первого блока в основание пирамиды
(tower)
;;    (этот блок не имеет под собой никакого другого
блока)
(defrule place-first
        ?my-goal <- (goal (task build))
        ?my-block <- (block (place hand))
        (not (block (place tower)))
=>
        (modify ?my-block (place tower))
        (modify ?my-goal (task find))
)
;; 4. Установка последующих блоков на основание
пирамиды
(defrule put-down
        ?my-goal <- (goal (task build))
        ?my-block <- (block (color ?C0) (place hand))
        (block (color ?C1) (place tower))
        (not (on (upper ?C2) (lower ?C1) (place tower)))
=>
        (modify ?my-block (place tower))
        (assert (on (upper ?C0) (lower ?C1) (place tower)))
        (modify ?my-goal (task find))
)
;; 5. Если в куче больше нет блоков, то пирамида
построена
(defrule stop
        ?my-goal <- (goal (task find))
        (not (block (place heap)))
=>
        (retract ?my-goal)
)
```

Приведем пошаговое описание состояния рабочей памяти в программе построения пирамиды из трех разных блоков, находящихся в куче:

```
- 0 ----------------------------------
(block (color red) (size 10) (place heap))
(block (color yellow) (size 20) (place heap))
(block (color blue) (size 30) (place heap))
```

Добавляем в рабочую память целевой вектор поиска первого блока:

```
- 1----------------------------------
(block (color red) (size 10) (place heap))
(block (color yellow) (size 20) (place heap))
(block (color blue) (size 30) (place heap))
(goal (task find))
```

Добавление этого факта позволяет активизировать правило выбора самого большого блока в куче:

```
- 2 ----------------------------------
(block (color red) (size 10) (place heap))
(block (color yellow) (size 20) (place heap))
(block (color blue) (size 30) (place hand))
(goal (task build))
```

Далее система, анализируя состояние рабочей памяти, выполняет правило установки найденного самого большого блока в основание пирамиды и начинает искать следующий по размерам блок:

```
- 3 ----------------------------------
(block (color red) (size 10) (place heap))
(block (color yellow) (size 20) (place heap))
(block (color blue) (size 30) (place tower))
(goal (task find))
- 4 ----------------------------------
(block (color red) (size 10) (place heap))
(block (color yellow) (size 20) (place hand))
(block (color blue) (size 30) (place tower))
(goal (task build))
```

Далее активизируется правило установки последующих блоков на уже сформированное основание пирамиды:

```
- 5 ----------------------------------
```

```
(block (color red) (size 10) (place heap))
(block (color yellow) (size 20) (place tower))
(block (color blue) (size 30) (place tower))
(goal (task find))
(on (upper yellow) (lower blue) (place tower))
- 6 ----------
(block (color red) (size 10) (place hand))
(block (color yellow) (size 20) (place tower))
(block (color blue) (size 30) (place tower))
(goal (task build))
(on (upper yellow) (lower blue) (place tower))
- 7 ----------
(block (color red) (size 10) (place tower))
(block (color yellow) (size 20) (place tower))
(block (color blue) (size 30) (place tower))
(goal (task find))
(on (upper red) (lower yellow) (place tower))
```

На следующем шаге цель поиска (find) очередного блока не может быть выполнена, так как в куче блоков больше нет. Поэтому пирамида построена и применяется правило прекращения процесса (stop).

Рассмотренные примеры показывают, что в прямом выводе рассуждения строятся, отталкиваясь от условий, которые удовлетворяются, к действиям, вытекающим из этих условий. Обратный вывод означает, что рассуждения строятся, отталкиваясь от заданной цели, к условиям, при которых возможно ее достижение. В CLIPS всегда сопоставляются состояние рабочей памяти и условные части правил, а затем выполняются действия, предусмотренные правой частью правил. Не существует способа, который бы позволял одному правилу вызвать другое (как в процедурном программировании). Одно правило может облегчить выполнение другого правила, но единственный способ его активизации – изменить состояние рабочей памяти. Иногда, чтобы решить, какое правило активизировать, желательно использовать конкретные знания, а не следовать стратегии разрешения конфликтов. Для этого в CLIPS имеется свойство выпуклости, отдающее при разрешении конфликтов предпочтение тому правилу, которое характеризуется большим значением этого свойства.

В целом, пространство поиска правил можно также представить И/ИЛИ-деревом, вершинами которого являются состояния рабочей памяти, а ребрами – правила. Если корень этого дерева считать целью, то листьями дерева являются данные или возможные варианты решения задачи. И-вершины соответствуют применению нескольких правил, а ИЛИ-вершины – наличию альтернативы при выборе правил. Продукционное программирование не снимает проблему комбинаторного взрыва, т.к. для многих задач И/ИЛИ-дерево может ветвиться по экспоненциальному закону. Поэтому алгоритмы вывода сравниваются на тестовых задачах логического программирования, таких как «Задача о ферзях», «Ханойские башни», «Стимроллер» и др.

1.4. Порядок выполнения работы

Порядок выполнения работы предусматривает изучение продукционной модели представления знаний, механизмов прямого и обратного логического вывода и их практической реализации на языке CLIPS.

Для этого необходимо:

- изучить представленный теоретический материал, рассмотреть примеры;
- четко представить себе, что необходимо сделать согласно своему варианту;
- определить, какие декларации или шаблоны целесообразно использовать;
- описать продукционные правила для решения заданной проблемы;
- задать начальные условия или сформулировать исходную проблему;
- практически реализовать на языке CLIPS программу, моделирующую заданную ситуацию;
- представить отчет о работе, включающий авторские выводы и оценку достижения поставленных целей.

1.5. Содержание отчета

1. Цель работы.
2. Постановка задачи согласно варианту.

3. Описание *самостоятельно* проделанной работы (изучение языка CLIPS, анализ продукционной модели представления знаний, механизмов прямого и обратного логического вывода; определение используемых деклараций и шаблонов; описание продукционных правил; задание начальных условий для исходной проблемы).

4. Выводы (они должны иметь характер нетривиальных утверждений, содержать обоснованную *оценку* степени достижения поставленных целей и задач, не подменяться перечислением наименований сделанного, поддерживаться программной реализацией и экспериментальными данными, не должны содержать мотивации, почему именно они такие).

Приложение: листинг программы на CLIPS с картой ее трассировки.

1.6. Варианты заданий

Представленные ниже варианты включают интеллектуальные задачи построения рассуждений на основе силлогизмов, прогнозирование погоды, медицинской и технической диагностики, выявления мотивации биологических особей, поиска маршрута по дорожной карте.

Каждый вариант предусматривает самостоятельную разработку продукционной модели представления знаний, реализацию механизмов прямого и обратного логического вывода в рамках заданной ситуации, а также их практическую реализацию на языке программирования CLIPS. Вариант задания выбирается студентом по согласованию с преподавателем.

1. Рассуждения на основе силлогизмов

На языке CLIPS напишите программу, которая будет выполнять рассуждения на основании силлогизмов. Силлогизм – это множество правил, состоящих из двух простых утверждений и одного заключения. Например, простыми силлогизмами являются:

все A_i являются B_i. Все B_i являются C_i. Все A_i являются C_i;

все A_i являются B_i. Некоторые A_i являются C_i. Все C_i являются B_i;

все A_i являются B_i. Ни один из C_i не является B_i. Ни один из C_i не является A_i.

Постройте также более сложные рассуждения на CLIPS в виде правил. С этой целью необходимо задать шаблоны, в которых определяется, что утверждение (*statement*) состоит из квантификатора (*quantifier*), принимающего, например, следующие *all (все), some (некоторые), no (ни один)* или некоторые из множеств *(set1,...)*.

Протестировать программу на некотором множестве фактов, например, щенки/собаки, собаки/млекопитающие, млекопитающие/животные, морские животные/собаки, морские животные/млекопитающие…

Реализовать механизмы прямого и обратного вывода на приведенной выше последовательности фактов или какой-нибудь другой.

2. Прогнозирование погоды

На языке CLIPS напишите программу прямого вывода, которая будет принимать решения на основе погодных данных и фактов, например наблюдений (солнечно, облачно, дождливо,..), температуры (жарко, умеренно, холодно), влажности (высокая, нормальная), ветрено (да, нет). В качестве возможных решений выбрать, например, «выйти на прогулку», «сидеть дома»,...

Другая программа на основе обратного вывода должна выявлять причины (погодные условия), по которым принимаются такие решения, как «выйти на прогулку» или «сидеть дома».

Смоделируйте данные и факты в виде правил. Проверить, насколько корректно работает программа на некотором множестве погодных наблюдений и фактов.

3. Диагностика заболевания

На языке CLIPS напишите программу, которая будет выполнять рассуждения для некоторого заболевания на основе набора диагностических правил, взятых из любого медицинского справочника. В исходном состоянии рабочей памяти необходимо представить множество симптомов заболевания. В интерфейсной части программы путем опроса пациента желательно учесть тот факт, что пациенты могут не упомянуть о некоторых неявных симптомах, в отличие от явных (боли). Самый простой способ – добавить правила, в предпосылках которых имеется только одно условие – жалобы на боль. Действие правила состоит в том, чтобы добавить в рабочую память вопрос, который нужно задать пациенту. Затем понадобится

правило, которое собственно задаст вопрос и поместит в рабочую память вектор, сформированный в соответствии с полученным ответом. Таким образом, и образец в левой части правила, и сопоставляемые с ним элементы в рабочей памяти должны соответствовать этим шаблонам.

Протестировать, как работает программа на некотором множестве симптомов, связанных с каким-либо органом (например, с сердцем, легкими или печенью). Сформировать вектор «диагностика», который классифицирует патологию органа. Указать некоторые факты, применив их к набору сформированных порождающих правил.

Реализовать механизмы прямого вывода (от симптомов к заболеванию) и обратного вывода (от заболевания к списку порождаемых и, возможно, зависимых симптомов).

4. Диагностика сбоя технического устройства

Ниже, в качестве примера, приводится модель диагностики неисправностей, взятая из инструкции по эксплуатации автомобиля (BMW 320).

Двигатель не заводится:

Если на стартер не подается ток, то причины могут быть следующие:

- разряжен аккумулятор;
- поврежден провод подключения к аккумуляторной батареи;
- поврежден соленоид стартера;
- плохой контакт с «массой».

Если на стартер подается ток, то причины могут быть следующие:

- заклинило шестерню стартера;
- поврежден двигатель стартера.

Двигатель проворачивается, но не запускается:

Если нет искры между электродами свечи, то причины могут быть следующие:

- загрязнены контакты прерывателя;
- наличие влаги в распределителе;
- неправильно подключены контакты прерывателя;

- поврежден конденсатор;
- поврежден ключ прерывателя;
- повреждена катушка.

Если нет топлива в жиклере карбюратора, то причинами может быть:

- отсутствие топлива в баке;
- паровая пробка в системе подачи топлива (жарким летом);
- засорен жиклер;
- неисправен бензонасос.

Двигатель заглох и вновь не заводится:

Если заливает карбюратор, то причины могут быть следующие:

- заедание игольчатого клапана;
- поврежден поплавок;
- неправильно установлен уровень поплавка.

Если нет топлива в жиклере карбюратора, то причинами может быть:

- отсутствие топлива в баке;
- попадание воды в систему подачи топлива.

Постройте на основе этой или аналогичной инструкции набор порождающих правил и разработайте CLIPS-программу диагностики автомобиля. Ввод причин неисправностей организуйте через ответы пользователя на запрос системы.

Реализовать механизм прямого многоэтапного вывода (от симптомов к искомой неисправности).

Реализовать механизм обратного многоэтапного вывода (от тех или иных причин неисправностей к их возможной симптоматике).

5. Выявление мотивации поведения биологической особи

Пусть имеется популяция рефлекторных агентов (искусственных, активных и относительно автономных организмов) с естественными потребностями энергии и размножения. Каждая потребность характеризуется определенной мотивацией, например, если энергетический ресурс агента небольшой, то у агента появляется мотивация найти пищу и пополнить энергетический ресурс. Каждый

агент эволюционирует в двумерной клеточной среде, в ячейках которой периодически вырастает трава (пища агентов). Каждый агент имеет внутренний энергетический ресурс, который пополняется при съедании им травы, либо уменьшается при выполнении каких-либо действий, например при размножении. Уменьшение ресурса до нуля приводит к смерти агента. Агенты могут скрещиваться, рождая новых агентов. Агент близорук и воспринимает состояние внешней среды только в четырех клетках: слева, справа, впереди и внутри той клетки, где он находится. Агент может выполнять следующие действия: отдыхать; перемещаться на одну клетку вправо, влево или вперед; принимать пищу; скрещиваться.

На языке CLIPS напишите программу, которая будет принимать решения на основе мотивации. Смоделируйте определенную мотивацию на CLIPS в виде правил поведения. Например, если возникает некоторая мотивация, то поведение агента изменяется с целью удовлетворения возникшей потребности.

Реализовать механизмы прямого многоэтапного вывода (от определенных мотиваций к потребностям агента), а также обратного многоэтапного вывода (от потребностей агента к породившим их мотивациям).

Протестировать корректность работы программы на некотором множестве агентов.

6. Поиск пути

Имеется абстрактная дорожная карта, представленная в виде графа на рис. 1.2.

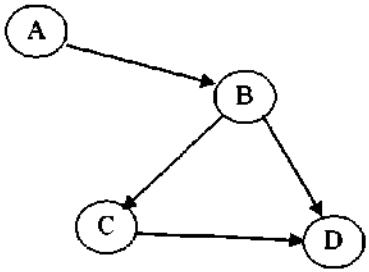


Рис. 1.2. Абстрактная карта дорог

Множество фактов составляют дороги между городами (A,B), (B,C), (B,D) и (C,D). База правил включает следующие три правила:

- прямая дорога между городами X и Y, если имеется факт (X,Y) или (Y,X);
- транзитная дорога между городами X и Y, если имеются факты (X,Z) и (Z,Y);
- дорога между городами X и Y, если имеется прямая дорога (X,Y) или транзитная дорога (X,Y).

Реализовать средствами CLIPS механизмы прямого вывода (найти все возможные маршруты дорог между городами A и D), а также обратного вывода (установить, есть ли дорога между пунктами A и D).

Контрольные вопросы

1. Согласно Посту каноническая система включает: а) алфавит, б) аксиомы, в) атрибуты, г) правила порождений, д) правила поведения.
2. Продукционная система включает: а) библиотеку, б) базу правил, в) рабочую память, г) память правил, д) компилятор правил, е) интерпретатор правил.
3. Необходимым условием выполнимости продукционных правил является: а) выполнение хотя бы одного из его условий, б) выполнение хотя бы одного из его заключений, в) выполнение всех его условий, г) выполнение всех его заключений.
4. Укажите правильную последовательность работы интерпретатора правил CLIPS: а) разрешить конфликт, б) применить выбранное правило, в) сопоставить условные части правил и элементы рабочей памяти.
5. Укажите соответствие между используемыми в CLIPS механизмами разрешения конфликтов: а) стратегия сложности, б) стратегия глубины, в) MEA-стратегия – и принципами их разрешения: 1) новизна, 2) разнообразие, 3) специфика.
6. CLIPS-программа может включать следующие элементы: а) аргументы, б) факты, в) шаблоны, г) шифры, д) комментарии, е) кванторы, ж) правила, з) порты.

7. Каждая из следующих последовательностей символов генерируется в соответствии с некоторым правилом. Опишите на CLIPS представление правила, необходимого для продолжения одной последовательности: а) 1, 2, 4, 8, 16,… б) 1, 1, 2, 3, 5, 8,… в) 1, a, 2, c, 3, f, 4,…

8. Ваша самооценка самостоятельно выполненной работы по критерию 1: 2_1_0, – по критерию 2: 2_1_0, – по критерию 3: 2_1_0.

Лабораторная работа №2

2. Программирование фреймовых моделей на языке CLIPS

Целью работы является изучение фреймовой модели представления знаний, а также ее практическое применение при решении интеллектуальных задач, используя язык CLIPS.

2.1. Краткое теоретическое описание

Продукционные модели знаний являются удобной моделью для представления связей между состоянием проблемы и действиями, которые необходимо предпринять для ее решения («что делать, если…?»). Иное дело, если необходимо представить знания о событиях, сложных объектах, связях между ними, их классификации или представить смысл выражений на естественном языке. Здесь практичнее использовать структуры в виде семантической сети (граф, вершины которого соответствуют понятиям или объектам, а дуги – отношениям между объектами). Однако у семантических сетей имеются очевидные недостатки, связанные с тем, что сеть одновременно является и средством представления знаний, и средством извлечения из нее нужной информации, и средством конструирования вывода на знаниях.

Естественным желанием исследователей в области искусственного интеллекта были попытки объединить вместе модели представления знаний с помощью семантических сетей и продукционных правил. Кроме того, отличительным свойством знаний в сравнении с данными является их *активность* (возможность появления или удаления каких-то фактов или связей между ними), которая связывается с использованием структур, объединяющих декларативные и процедурные компоненты базы знаний. Это способствовало становлению концепции *фреймов* (frame — остов, скелет, костяк, каркас) как одной из перспективных моделей инженерии знаний, во многом обязанному ряду интуитивных предположений, касающихся механизмов психологической деятельности человека.

Идея фреймов основана на том, что представление понятий в мозге довольно расплывчато. Многие из тех сущностей, которыми мы пользуемся, не имеют четкого определения, а базируются на нечетких понятиях. Человек обращает внимание на свойства, которые у него ассоциируются с объектами-прототипами, наиболее ярко представляющими свой класс (птица – воробей, четырехугольник – прямоугольник и т.п.). Границы между разными классами всегда размыты, а в каждом правиле или классе встречаются исключения.

При использовании фреймов знания не «размазываются» по программному коду приложения, как в продукционных системах, но и не собираются воедино в виде метазнаний как в семантических сетях или в ассоциативных сетях (сети, используемые для представления семантики естественного языка). Используя идеи теории фреймов, в 70-х годах в MTI провели исследования, которые привели к разработке философии объектно-ориентированного программирования и созданию таких языков, как Smalltalk, FRL, C++, Java.

М. Минский определил фрейм как «структуру данных для представления стереотипных ситуаций» и как способ формализации знаний, используемый при решении таких интеллектуальных задач, как распознавание образов и понимание речи. Реализованная им идея заключалась в том, чтобы сконцентрировать все данные (знания) о конкретном объекте или событии в единой структуре, а не распределять ее между множеством более мелких структур. Иными словами, фреймовая модель позволяет судить о классе объектов, используя знание о его прототипах, которые хорошо представляют большинство разновидностей объектов данного класса, но могут корректироваться для того, чтобы представить всю сложность, присущую реальному миру. Такая ситуация типична для подавляющего большинства эвристических механизмов, используемых человеком при решении практических проблем. Поэтому фреймовая модель оказывается полезной, поскольку она позволяет структурировать эвристические знания.

Конструктивно фрейм представляет собой запись или сложный узел в сетевой гетерархии («запутанная» иерархия, в которой узлы могут иметь более одного предшественника) с наименованием сущности, которую он представляет. Фрейм идентифицируется уникальным *именем* и включает в себя множество *слотов*. В свою очередь, каждому слоту соответствует определенная структура. В

слотах описывается информация о фрейме: его свойства, характеристики, относящиеся к нему факты и т.д. Представление предметной области в виде иерархической системы фреймов хорошо отражает внутреннюю и внешнюю структуру объектов этой предметной области, так как фрейм, являясь формой описания знаний, очерчивает рамки рассматриваемого в задаче фрагмента предметной области. В процессе адаптации фрейма к конкретным условиям уточняются значения слотов. В случае поиска фрейма, релевантного некоторому образу, можно проверить совпадение слотов. Передать данные во фрейм, заполнив его слоты, можно несколькими способами: при конструировании фрейма, через вызов функции, указанной в слоте, через присоединенную к слоту процедуру (*демон*), из диалога с пользователем, через наследование свойств от других фреймов, из базы данных. Фреймовая модель также удобна для явного представления ситуационных знаний о предметной области, так как в конструкции фрейма предусматриваются управляющие слоты с данными для подтверждения или отклонения фрейма, для вывода сообщения-решения и т.п. Вместе с тем, для фреймовой модели характерны некоторая сложность управления выводом и относительно невысокая эффективность его присоединенных процедур.

Организация вывода во фреймовой системе базируется на обмене сообщениями между фреймами, активации и выполнении присоединенных процедур. Отражение в иерархии фреймов родовидовых отношений обеспечивает возможность реализации операции наследования, позволяющей приписывать фреймам нижних уровней иерархии свойства, присущие фреймам вышележащих уровней. Причем, данные, которые передаются в процессе функционирования системы от посторонних источников знаний во фреймы нижних уровней, имеют более высокий приоритет, чем данные, унаследованные от фреймов верхних уровней. Наследование позволяет описывать свойства и поведение класса в терминах других классов. Язык CLIPS поддерживает множественное наследование пользовательских и абстрактных классов (аналогичных виртуальным классам в C++). Следует учитывать, что такая организация связей между фреймами не влечет проблем, пока информация от разных источников наследования не является противоречивой. Поэтому

программисту лучше заранее подумать о том, как избежать конфликта в сети фреймов, например путем анализа ее топологии.

2.2. Пример выполнения

Использование объектно-ориентированных средств в CLIPS позволяет значительно упростить программирование правил, поскольку для обновления данных можно применять механизм передачи и обработки сообщений между классами. Продемонстрируем это на примере, который моделирует правила обращения с полуавтоматическим пистолетом и представлен П. Джексоном в [4].

Вначале определим класс pistol, в котором перечисляются свойства, необходимые для конструирования фреймовой модели:

```
(defclass pistol
        (is-a USER)
        (role concrete)
        (pattern-match reactive)
        (slot safety (type SYMBOL) (create-accessor read-write))
        (slot slide (type SYMBOL) (create-accessor read-write))
        (slot hammer (type SYMBOL) (create-accessor read-write))
        (slot chamber (type INTEGER) (create-accessor read-write))
        (slot magazine (type SYMBOL) (create-accessor read-write))
        (slot rounds (type INTEGER) (create-accessor read-write))
)
```

Прокомментируем данное описание. Первые три слота являются системными. Они нужны объектно-ориентированной надстройке CLIPS (COOL – CLIPS Object-Oriented Language) для спецификации пользовательского класса *pistol* и возможности создания экземпляров этого класса. Кроме того, экземпляры класса могут быть использованы в качестве объектов данных, которые сопоставляются с условиями в правилах CLIPS и используются в действиях, определенных этими правилами. Следующие шесть слотов представляют свойства и значения данных классов:

- слот safety (предохранитель) содержит информацию о положении предохранителя и может принимать одно из символьных значений - on или off;

• слот slide (затвор) содержит информацию о положении затвора и может принимать одно из символьных значений - forward или back;

• слот hammer (курок) содержит информацию о состоянии курка и может принимать одно из символьных значений - back или down;

• слот chamber (патронник) характеризует наличие или отсутствие патрона в патроннике и может принимать значения 1 или 0;

• слот magazine (обойма) характеризует наличие или отсутствие обоймы в пистолете и может принимать символьные значения in или out;

• слот rounds (патроны) содержит информацию о текущем количестве патронов в обойме и может принимать целочисленные значения.

Для записи в слот новых значений или считывания текущих используется фасет. Фасет – это однородная группа, связанная общностью признака (характеристики, основания деления). В приведенном выше описании класса pistol фасет реализуется через функцию create-accessor.

Сформируем *экземпляр* класса pistol с помощью следующего выражения:

```
(definstances pistols (PPK of pistol
        (safety on)
        (slide forward)
        (hammer down)
        (chamber 0)
        (magazine out)
        (rounds 6)
))
```

Данная запись означает, что экземпляр пистолета PPK находится в правильном походном состоянии: обойма вынута из рукоятки, пистолет установлен на предохранитель, затвор находится в переднем положении, курок опущен, а патронник пуст. В обойме имеется 6 патронов.

Теперь, определив программно класс pistol и сформировав экземпляр этого класса, разработаем правила и обработчики сообщений, с помощью которых можно описать отдельные операции обращения с пистолетом и стрельбы из него. С этой целью вначале

разработаем *шаблон* задачи, который позволяет отслеживать следующие состояния:

- есть ли патрон в патроннике;
- произведен ли выстрел из пистолета?

Для этого можно использовать следующий шаблон:

```
(deftemplate range-test
        (field check (type SYMBOL) (default no))
        (field fired (type SYMBOL) (default no))
)
```

Согласно шаблону по правилу start в рабочей памяти будет установлена задача range-test:

```
(defrule start
        (initial-fact)
=>
        (assert (range-test))
)
```

При активизации этого правила в рабочую память будет добавлен факт: (range-test (check no) (fired no)).

С помощью следующих трех правил (check, correct1, correct2) можно проверить, правильно ли снаряжен пистолет.

```
(defrule check
        (object (name [PPK]) (safety on) (magazine out))
        ?T <- (range-test (check no))
=>
        (send [PPK] clear)
        (modify ?T (check yes))
)
```

Правило check заключается в том, что если пистолет стоит на предохранителе (safety on), обойма вынута (magazine out) и пистолет не был проверен, то нужно очистить патронник и проверить, нет ли в нем патрона. При этом обработчик сообщения clear для класса pistol будет выглядеть следующим образом:

```
(defmessage-handler pistol clear ()
        (dynamic-put chamber 0)
        (ppinstance)
)
```

- где в первой строке объявляется, что clear является обработчиком сообщения для класса pistol, причем этот обработчик не

требует передачи аргументов. Оператор во второй строке «очищает» патронник. Присвоение выполняется независимо от того, какое текущее значение имеет слот chamber (0 или 1). Оператор в третьей строке требует, чтобы экземпляр распечатал информацию о текущем состоянии своих слотов.

В следующих двух правилах обрабатываются ситуации, когда пистолет снаряжен неправильно: не установлен на предохранитель или в него вставлена обойма. Правило correct1 устанавливает пистолет на предохранитель, а правило correct2 извлекает из него обойму.

```
(defrule correct1
        (object (name [PPK]) (safety off) )
        (range-test (check no))
=>
        (send [PPK] safety on)
)

(defrule correct2
        (object (name [PPK]) (safety on) (magazine in))
        (range-test (check no))
=>
        (send [PPK] drop)
)
```

Здесь, аналогично, как и при разработке правила check, рекомендуется использовать обработчики сообщений safety и drop.

```
(defmessage-handler pistol safety (?on-off)
        (dynamic-put safety ?on-off)
        (if (eq ?on-off on) then (dynamic-put hammer down)
))
```

Обработчик сообщения safety имеет единственный аргумент, который может принимать только два символьных значения (on или off). Естественно, при разработке фреймовой системы необходимо учитывать особенности предметной области. В данном случае особенность заключается в том, что в некоторых моделях пистолетов при установке его на предохранитель патронник автоматически очищается.

Обработчик сообщения drop фиксирует извлечение обоймы из пистолета:

```
(defmessage-handler pistol drop ()
```

```
        (dynamic-put magazine out)
)
```

Теперь, когда обеспечено правильное исходное снаряжение пистолета, можно приступить к описанию стрельбы. Следующее правило описывает подачу обоймы в пистолет перед стрельбой:

```
(defrule mag-in
        (object (name [PPK]) (safety on) (magazine out))
        (range-test (fired no) (check yes))
=>
        (send [PPK] seat)
)
```

При этом обработчик сообщения seat выполняет действия, противоположные тем, которые выполняет обработчик drop.

```
(defmessage-handler pistol seat ()
        (dynamic-put magazine in)
)
```

А вот включение в программу следующего правила mag-in:

```
(defrule mag-in
        ?gun <- (object (name [PPK]) (safety on) (magazine out))
        (range-test (fired no) (check yes))
=>
        (modify ?gun (magazine in))
)
```

- представляется не целесообразным, поскольку оно противоречит одному из принципов объектно-ориентированного программирования: объект должен самостоятельно обрабатывать содержащиеся в нем данные.

Следующее правило обеспечивает снаряжение обоймы патронами:

```
(defrule load
        (object (name [PPK]) (magazine in) (chamber 0))
=>
        (send [PPK] rack)
)
```

На примере обработчика сообщения rack можно убедиться в справедливости замечания о том, что обработку данных внутри объекта нужно поручать этому объекта, а не включать непосредственно в правило:

```
(defmessage-handler pistol rack ()
        (if (> (dynamic-get rounds) 0)
        then (dynamic-put chamber 1)
                (dynamic-put rounds (- (dynamic-get rounds) 1))
                (dynamic-put slide forward)
        else (dynamic-put chamber 0)
                (dynamic-put slide back)
))
```

В этом обработчике обеспечивается досылка патрона в патронник в том случае, если патроны имеются в обойме.

Следующее правило подготавливает пистолет к стрельбе, снимая его с предохранителя. В правиле повторно используется сообщение safety, но на этот раз с аргументом off:

```
(defrule ready
        (object (name [PPK]) (chamber 1)
=>
        (send [PPK] safety off)
)
```

Правило fire выполняет стрельбу.

```
(defrule fire
        (object (name [PPK]) (safety off)
        ?T <- (range-test (fired no))
=>
        (if (eq (send [PPK] fire) TRUE)
        then (modify ?T (fired yes)))
)
```

В данном правиле используется обработчик сообщения, которое позволяет выяснить, произведен ли выстрел, то есть выполнена ли в действительности та операция, которая «закреплена» в этом сообщении. Если в патроннике был патрон и пистолет был снят с предохранителя, то обработчик сообщения вернет значение TRUE (после того, как выведет на экран BANG!). В противном случае обработчик укажет значение FALSE (после того, как выведет на экран click):

```
(defmessage-handler pistol fire ()
        (if (and
        (eq (dynamic-get chamber) 1)
        (eq (dynamic-get safety) off)
```

```
        )
        then (printout t crlf "BANG!" t crlf)
                TRUE
        else (printout t crlf "click" t crlf)
                FALSE
))
```

В обработчике сообщения анализируется условие, которое уже было проанализировано правилом, отославшим сообщение (в данном случае речь идет об условии safety off). Дело в том, что одно и то же сообщение может отсылаться разными правилами и нет никакой гарантии, что в каждом из них будет проверяться это условие.

После завершения стрельбы пистолет нужно вновь вернуть в «походное» положение. Согласно инструкции, вначале пистолет необходимо установить на предохранитель. Соответствующее правило имеет следующий вид (в нем используется ранее разработанный обработчик сообщения safety):

```
(defrule unready
        (object (name [PPK]) (safety off))
        (range-test (fired yes))
=>
        (send [PPK] safety on)
)
```

Далее, по инструкции необходимо вынуть обойму. Соответствующее правило имеет следующий вид (в нем используется ранее разработанный обработчик сообщения drop):

```
(defrule drop
        (object (name [PPK]) (safety on))
        (range-test (fired yes))
=>
        (send [PPK] drop)
)
```

Наконец, согласно последнему правилу, патрон должен быть выброшен из патронника (в правиле используется ранее разработанный обработчик сообщения clear):

```
(defrule unload
        (object (name [PPK]) (safety on) (magazine out))
        (range-test (fired yes))
=>
```

```
    (send [PPK] clear)
)
```

Рассмотренный пример наглядно демонстрирует, как в рамках единой программы на CLIPS сочетаются правила и объекты. Правила управляют ходом вычислений, но некоторые операции объекты выполняют самостоятельно, получив соответствующее сообщение от правила. Объекты не являются резидентами рабочей памяти, но левые части правил (антецеденты-условия) могут быть сопоставлены с содержимым их слотов. Состояние объектов может быть также изменено после активизации того или иного правила, однако лучше предоставить объектам возможность самостоятельно выполнять манипуляции с хранящимися в них данными в ответ на поступающие от правил сообщения. Объекты не могут самостоятельно активизировать правила, но их обработчики сообщения могут возвращать определенную информацию о результатах, которая используется для управления логикой выполнения действий в правой части (консеквенты-действия) правила.

2.3. Порядок выполнения работы

Порядок выполнения работы предусматривает изучение фреймовой модели представления знаний и ее практическую реализацию на языке CLIPS.

Для этого необходимо:

- изучить представленный теоретический материал, рассмотреть пример;
- четко представить себе, что необходимо сделать согласно своему варианту;
- определить, какие декларации, шаблоны, классы целесообразно использовать;
- описать классы и правила для решения заданной проблемы;
- задать начальные условия или сформулировать исходную проблему;
- практически реализовать на языке CLIPS программу, моделирующую заданную ситуацию;
- представить отчет о работе, включающий авторские выводы и оценку достижения поставленных целей.

2.4. Содержание отчета

1. Цель работы.
2. Постановка задачи согласно варианту.
3. Описание *самостоятельно* проделанной работы (изучение фреймовой модели представления знаний, построение микромира при помощи фреймов и связей между ними; определение используемых деклараций и шаблонов; описание классов и необходимых правил; задание начальных условий для исходной проблемы).
4. Выводы (они должны иметь характер нетривиальных утверждений, содержать обоснованную *оценку* степени достижения поставленных целей и задач, не подменяться перечислением наименований сделанного, поддерживаться программной реализацией и экспериментальными данными, не должны содержать мотивации).

Приложение: листинг программы на CLIPS с картой ее трассировки.

2.5. Варианты заданий

Представленные ниже варианты включают интеллектуальные задачи-головоломки, разработку программы-помощника покупателя, составление таблиц спортивных соревнований. Каждый вариант предусматривает самостоятельную разработку фреймовой системы (желательно, чтобы фреймовая иерархия включала не менее трех уровней в заданной предметной области), моделирование поведения объектов. Конструировать типы фреймов можно по-разному: фреймы-структуры (например, описание процесса логического вывода или конструкции дома), фреймы-роли (например, компьютерная программа-помощник покупателя), фреймы-сценарии (движения по лабиринту и т.п.), особое внимание обратить на взаимодействие правил и объектов. Вариант задания выбирается студентом по согласованию с преподавателем.

1. Головоломка «Правдолюбцы и лжецы»

Остров населен обитателями двух категорий: одни всегда говорят правду (правдолюбцы), а другие всегда лгут (лжецы). Решить на выбор следующие головоломки:

• встречаются два человека (A и B), один из которых правдолюбец, а другой – лжец. Один говорит: «Либо я лжец, либо Вы правдолюбец». Кто из этих двоих является правдолюбцем, а кто - лжецом?

• встречаются три человека (A, B и C). Один говорит: «Все мы лжецы», другой отвечает: «Только один из нас правдолюбец». Кто из троих правдолюбец, а кто лжец?

Решение предполагает анализ ситуаций, когда персонаж, произносящий реплику, является либо правдолюбцем, либо лжецом.

1. Головоломка о миссионерах и каннибалах

На левом берегу реки находятся три миссионера и три каннибала. На этом же берегу причалена единственная лодка. На этой лодке нужно переправить всех миссионеров и всех каннибалов на правый берег при условии, что в лодке одновременно могут поместиться только двое, а в обратный путь к левому берегу должен отправиться хотя бы один человек. Есть одно обстоятельство, существенно влияющее на процесс решения задачи: если окажется, что каннибалов на любом из берегов больше, чем миссионеров, то несчастных просто съедят. Решение головоломки – это последовательность шагов, переводящая систему из исходного состояния в заданное конечное состояние. Для поиска решения можно воспользоваться эвристическим алгоритмом «первый лучший» с соответствующей оценочной функцией.

2. Криптографическая головоломка

Термин «криптографическая» означает использование цифр, зашифрованных буквами, и соответственно чисел, зашифрованных словами. Задача состоит в том, чтобы найти, какие цифры нужно подставить вместо букв, чтобы представленные арифметические операции над расшифрованными числами давали верный результат. Решить на выбор следующие головоломки:

а) B E S T	б) DONALD	в) CROSS
+MAD E	+GERALD	+ROADS
----------	----------	----------
MASTER	ROBERT	DANGER

Необходимо подумать, как представить слагаемые и сумму, какие возможны в решении задачи наборы операций для перехода из одного состояния в другое и какую эвристику применить для управления

поиском решения. Для решения можно использовать различные схемы. Например, рассмотреть первые разряды всех трех чисел (первый разряд – это единицы, второй – десятки, третий – сотни и т.д.). Очевидно, что множество пар буква-цифра, входящих в решение задачи, для каждого разряда должны удовлетворять определенным условиям. Анализируя эти условия можно получить решение, представляя комбинации из одной буквы и одной цифры в виде фактов и составляя правила для нахождения комбинаций, удовлетворяющих найденным условиям.

3. Программа-помощник покупателя компьютерной техники

Согласно экспертным оценкам директоров по продажам и маркетингу отечественных компьютерных фирм, некоторая сложность продажи компьютеров состоит в том, что розничного покупателя удерживает от приобретения незнание устройства, далекого от обычной бытовой техники. Прайс-лист, совет знакомых и продавца зачастую не помогают сделать правильный выбор. Необходимо разработать компьютерную программу, беспристрастно выясняющую запрос клиента по выбору некоторого узла персонального компьютера (например, видеокарты и т.п.) с учетом имеющегося предложения. При построении фреймовой системы учесть параметры узла, установить связи между параметрами выбора и отношением к ним клиента. Реализовать процесс приобретения знаний (формирование правил) путем заполнения полей слотов фрейма в ходе диалога с клиентом, а также используя пользовательские функции для формулирования вопросов и вариантов ответа на них. Результатом работы программы должны быть рекомендации клиенту по выбору той или иной компоненты компьютера.

4. Составление таблицы соревнований

Известны факты-результаты после нескольких игровых туров в группе футбольной Лиги чемпионов среди четырех команд. При построении фреймовой системы учесть количество игр, количество побед, ничьих и поражений, а также число забитых и пропущенных мячей, количество набранных командами очков. Место команд в турнирной таблице определяется на основании набранных очков, при равенстве очков учитываются результаты личных встреч между командами, разница забитых и пропущенных мячей и т.д. В процессе заполнения таблицы использовать правила и сообщения CLIPS.

5. Задача о замке́

Задано прямоугольное поле, разбитое на ячейки. Некоторые из ячеек закрыты, а некоторые открыты. При изменении состояния какой-либо ячейки, т.е. при ее закрытии или открытии, все ячейки, находящиеся на «кресте» (на общей с ней вертикали или на общей горизонтали), меняют свое состояние на противоположное. Задача состоит в том, чтобы путем переключения определенных ячеек открыть замок, т.е. сделать все его ячейки открытыми. Перебором данную задачу даже при относительно небольших размерах замка решить затруднительно. Однако она просто решается даже в уме по алгоритму, основанному на существовании инварианта в системе замка.

Контрольные вопросы

1. Укажите соответствие между понятиями: а) каноническая система, б) нечеткая логика, в) фрейм и их авторами: 1) Заде, 2) Минский, 3) Пост.

2. Передать данные во фрейм, заполнив его слоты, можно: а) из базы данных, б) через «демон», в) через наследование свойств от других фреймов, г) из диалога с пользователем, д) через вызов функции, указанной в слоте, е) через фасет. Какие способы передачи данных во фрейм использовали Вы?

3. Укажите приоритет передаваемых во фрейм свойств в порядке убывания: а) из фреймов вышележащего уровня иерархии, б) от посторонних источников знаний, в) от своих слотов.

4. Укажите основные свойства фреймов, используемые в CLIPS: а) инкапсуляция, б) наследование, в) параллельность, г) полиморфизм, д) рекурсия.

5. Ваша самооценка самостоятельно выполненной работы по критерию 1: 2_1_0, по критерию 2: 2_1_0, по критерию 3: 2_1_0.

Лабораторная работа №3

3. Программирование нечетких моделей в среде MatLab

Целью работы является программирование нечеткой интеллектуальной модели управления средствами инструментария нечеткой логики в интерактивной среде MatLab.

3.1. Краткое теоретическое описание

Залогом успешного решения любой интеллектуальной задачи является правильное понимание ее сути, на основе чего можно выбрать адекватную математическую модель. Это особенно актуально для слабоструктурированных задач, к которым относятся, например, задачи интеллектуального управления в условиях неопределенности. Природа неопределенности может иметь как вероятностный, так и нечеткий субъективный характер. Для ее оценки Л. Заде была предложена нечеткая логика, которая является теорией, отличной от вероятностного подхода к представлению неопределенности в знаниях. В частности, в последних работах автора нечеткой логики указывается на ряд ограничений классической теории вероятности, которые не позволяют адекватно отразить присущее человеку частичное восприятие мира: частичную точность, истину, веру, определенность и т.п.

Нечеткая логика успешно моделирует частичную определенность. Например, нечеткие модели уже обеспечили себе коммерческий успех в самых разных приложениях: системы управления (от бытовой техники до автомобильных АБС-систем), автофокусирование видеокамер, фотоаппаратов и т.п. Они позволяют программам работать в диапазоне (0, 1) различных степеней истины. Их преимущества проявляются, когда система анализируется с помощью нечетких лингвистических переменных (низкий, высокий и т.п.). Нечеткая логика имеет свою аксиоматику и набор базовых операторов, действующих несколько иначе, чем аналогичные булевы операторы. Поскольку человеческая логика сама по себе является приблизительной, то и нечеткая логика имеет огромное значение при

создании программного обеспечения, где вместо детерминированных алгоритмов применяются экспертные знания. Подробно теоретические и прикладные аспекты теории нечетких множеств рассматриваются в [8, 10]. Ниже приводятся лишь основные понятия теории, необходимые для понимания существа данной лабораторной работы.

Нечетким множеством $\tilde{A}$ называется множество, определенное на произвольном непустом множестве X как множество пар вида:

$$\tilde{A} = \{\mu_{\tilde{A}}(x)/x\},$$

где $x \in X, \mu_{\tilde{A}}(x) \in [0,1]$. Множество X называется базовым множеством. Функция $\mu_{\tilde{A}}(x)$ называется *функцией принадлежности* множества $\tilde{A}$, она указывает на степень принадлежности элемента x нечеткому множеству $\tilde{A}$. Вид функции принадлежности определяется субъективно или экспертно. С возрастанием их числа и области неопределенности у эксперта могут возникнуть серьезные затруднения в процессе представления своих эвристических знаний в абстрактной форме. Поэтому на практике применяется иной подход. Он состоит в том, что после установки нечетких множеств, для характеристики взаимосвязей входных и выходных величин управления применяются те или иные простые формы фазирования функции принадлежности, как линейные (треугольники или трапеции), так и нелинейные (колокол). Примерный вид этих форм представлен на рис. 3.1 в виде функций принадлежности $\mu(x)$ на нечетком множестве элементов $\tilde{A}$ из базового множества X.

Например, если некоторое свойство нечеткого множества изменяется относительно некоторой центральной группы элементов, то функция $\mu(x)$ имеет как монотонно возрастающую, так и монотонно убывающую составляющие (трапеция). Если существует всего один элемент (m), удовлетворяющий условию $\sup_x \mu(x) = 1$, то трапеция преобразуется в треугольник. Нелинейное нечеткое множество, соответствующее этому случаю, имеет колоколообразный вид. Способ задания функции принадлежности может быть аксиоматическим (подбор на основе известной закономерности) или эмпирическим (по точкам на основе субъективных оценок экспертов). Детально функции принадлежности изучаются в рамках теории возможностей.

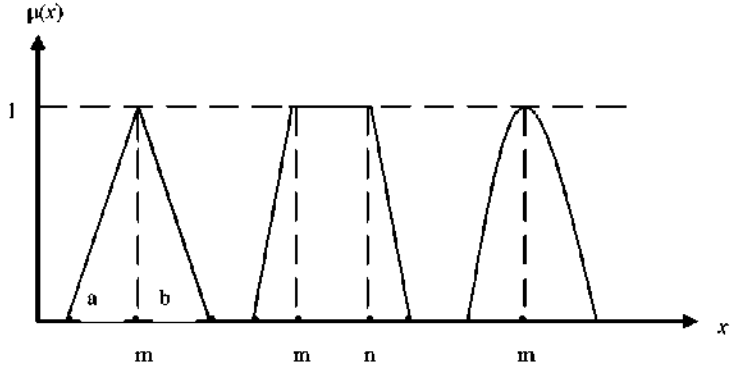


Рис. 3.1. Формы функции принадлежности

Нечеткая логика позволяет представлять ситуации, которые оперируют нечеткими понятиями или когда нет уверенности в самих фактах, описывающих ситуацию. Высказывание $\tilde{E}$ называется *нечетким высказыванием*, если допускается, что $\tilde{E}$ может быть одновременно истинным и ложным (в отличие от аристотелевской логики, где такая возможность исключается). Любое оценочное суждение, основанное на неполных или недостоверных данных, является нечетким и сопровождается обычно выражением степени уверенности (или сомнения) в его истинности. Мера истинности нечеткого высказывания $\tilde{E}$ также определяется функцией принадлежности, задаваемой обычно на множестве X= {true, false}. Нечеткие высказывания, характеризующиеся равной степенью уверенности и сомнения (0,5), называют нечетко-индифферентными.

Нечеткие высказывания могут быть простыми и составными. Составные высказывания образуются из простых с помощью логических операций НЕ (¬), И (&), ИЛИ (+) или импликации (→) вида «Если (условия), то (заключение)». В отличие от традиционной математической логики, в нечеткой логике этим операциям придается специфический смысл (минимаксная логика Заде).

В частности, *отрицанием* нечеткого высказывания $\tilde{E}$ называется нечеткое высказывание $\neg\tilde{E}$, степень истинности которого определяется выражением $1 - \mu(\tilde{E})$. *Конъюнкцией* нечетких высказываний $\tilde{E}_1 \& \tilde{E}_2$ называется нечеткое высказывание, степень истинности которого определяется выражением $\min(\mu(\tilde{E}_1), \mu(\tilde{E}_2))$, т.е. степень истинности

определяется наименее правдоподобным высказыванием. *Дизъюнкцией* нечетких высказываний $\tilde{E}_1+\tilde{E}_2$ называется нечеткое высказывание, степень истинности которого определяется выражением $\max(\mu(\tilde{E}_1), \mu(\tilde{E}_2))$, т.е. степень его истинности определяется наиболее правдоподобным высказыванием. *Импликацией* нечетких высказываний $\tilde{E}_1 \rightarrow \tilde{E}_2$ и называется нечеткое высказывание, степень истинности которого определяется выражением $\max(1 - \mu(\tilde{E}_1), \mu(\tilde{E}_2))$. Рассмотренные операции имеют удобную графическую интерпретацию с привлечением рассмотренных ранее форм представления функции принадлежности (по отношению к одномерным множествам). Кроме того, для двумерных нечетких множеств дополнительно применяются операции *проекции* и *композиции*. Композиция нечетких правил используется для организации *нечеткого вывода*. Переход от нечетких отношений к четким (*дефазификация*) основывается на принципе округления чисел (если $\mu(x) \le 0{,}5$, то $a = 0$, иначе $a = 1$, где a – элемент нечеткого отношения).

Согласно Заде, нечеткое высказывание, степень истинности которого может принимать произвольное значение из интервала [0,1], называется *нечеткой логической переменной*. Нечеткие логические формулы можно получить на основе операций над нечеткими логическими переменными и их значениями из интервала [0,1], т.е. понятие нечеткой логической формулы можно ввести индуктивно. Нечеткой логической формулой называется:

а) нечеткая логическая переменная или константа из интервала [0,1];

б) всякое выражение, построенное из нечетких логических формул применением любого конечного числа логических операций;

в) те и только те выражения, которые построены согласно пунктам а) и б).

Составные нечеткие высказывания являются нечеткими логическим формулами, если входящие в них простые нечеткие высказывания рассматривать как нечеткие логические (лингвистические) переменные.

Таким образом, при решении практических задач с использованием нечетких моделей объектов необходимо определить перечень признаков, характеризующих объект, диапазоны значений каждого признака, их значимость в общей оценке объекта, а также

перечень значений используемых лингвистических переменных. Последующие действия и шаги определяются характером предметной области и решаемой задачи и связаны с построением функций принадлежности, как наиболее сложной проблемой при построении систем с нечеткой логикой. Почему? - Функции принадлежности, на самом деле, имеют двоякую природу. С одной стороны, они призваны решать логическую задачу отнесения объекта к одному из классов. С другой стороны, они показывают не только факт, но и меру принадлежности определенному классу. Это позволяет рассматривать значения функций принадлежности как нормированные величины и выполнять над ними арифметические действия. Однако не следует забывать о том, что трудно рассчитывать на получение достоверного результата, используя субъективные исходные данные. Тем не менее, это не мешает человеку находить приемлемые решения задач, формулируемых на естественном языке в условиях неопределенности. А это означает, что нечеткие модели дают возможность автоматизировать и получать не худшие, чем принимаемые человеком, решения таких задач.

Еще раз отметим, что, приступая к самостоятельному выполнению задания к лабораторной работы в среде MatLab важно понять, в чем заключается источник неопределенности.

MatLab – это высокопроизводительный язык для решения аналитических и инженерных задач. Он включает в себя средства, позволяющие пользователю программировать в удобной среде математические задачи, создавать алгоритмы и модели вычислений, анализировать, исследовать и визуализировать символьные и графические данные, разрабатывать приложения, включая создание графического интерфейса.

Интерактивная система MatLab вобрала в себя не только опыт развития и компьютерной реализации численных методов, накопленный за последние три десятилетия, но и многие классические методы математики. MatLab применяют миллионы зарегистрированных пользователей, используют в своих научных проектах университеты и исследовательские центры. Популярности системы способствует наличие множества расширений, таких как Simulink, с удобными и простыми средствами визуального и объектно-ориентированного программирования, моделирования линейных и нелинейных динамических систем, что расширяет возможности и круг

пользователей данной системы, а также повлияло на выбор авторов лабораторного практикума.

Система MatLab состоит из пяти основных частей.

Язык MatLab служит для представления матриц и многомерных массивов, функций, структур данных, ввода-вывода, обладает возможностями объектно-ориентированного программирования. Язык удобен как для программирования небольших учебных задач, так и для создания сложных приложений.

Среда MatLab включает набор инструментов и средств, с которыми работает пользователь или программист MatLab. Эти средства служат для управления переменными, ввода и вывода данных, а также для создания, контроля и отладки М-файлов и приложений MatLab.

Графическая система MatLab включает в себя команды высокого уровня для визуализации двух- и трехмерных данных, обработки изображений, анимации и иллюстрации. Она также включает в себя команды низкого уровня, позволяющие полностью редактировать внешний вид графики при создании графического пользовательского интерфейса (GUI) для MatLab-приложений.

Библиотека математических функций состоит из обширной коллекции алгоритмов вычисления как элементарных функций (сумма, синус, косинус, арифметика комплексных чисел и др.), так и вычислительных алгоритмов обращения матриц, нахождения их собственных значений, функций Бесселя, быстрого преобразования Фурье и т.п.

Программный интерфейс организован в виде библиотеки, позволяющей разрабатывать взаимодействующие с MatLab программы на языках Си и Фортран. Библиотека включает средства для вызова вычислительных программ из MatLab (динамическая связь) и для чтения-записи файлов *.mat.

В системе MatLab важная роль отводится специализированным группам программ, называемым Toolbox (пакеты прикладных программ). Они очень важны для большинства пользователей, так как позволяют изучать и применять специализированные математические и инженерные методы. Toolboxes – это разносторонняя коллекция функций MatLab (M-файлов), которые позволяют решать такие классы задач, как обработка сигналов (Signal Processing Toolbox), управление (Control System Toolbox), линейное программирование (LMI Control

Toolbox), вейвлет-анализ (Wavelet Toolbox), проектирование нейронных сетей (Neural Networks Toolbox), разработка систем нечеткой логики (Fuzzy Logic Toolbox) и др.

В частности, пакет Fuzzy Logic используется при выполнении данной лабораторной работы. С его помощью обеспечивается поддержка методов нечеткой кластеризации и адаптивных нечетких нейронных сетей. Графические средства пакета позволяют интерактивно отслеживать особенности поведения проектируемой системы.

Пакет предоставляет следующие возможности:

- определение переменных, нечетких правил и функций принадлежности;
- интерактивный просмотр нечеткого логического вывода;
- гибридные методы адаптивного нечеткого вывода с использованием нейронных сетей и нечеткой кластеризации;
- интерактивное динамическое моделирование в Simulink;
- генерация переносимого Си-кода с помощью приложения Real-Time Workshop.

Одной из основных задач лабораторной работы является изучение различных схем вывода на базе нечетких правил. Пакет Fuzzy Logic позволяет строить нечеткие системы со схемами вывода по Мамдани и Суджено.

В системах с выводом по Мамдани база знаний состоит из правил вида *«Если x_1=низкий и x_2=средний, то y=высокий»*. В системах с выводом по Суджено база знаний состоит из правил вида *«Если x_1=низкий и x_2=средний, то $y=a_0+a_1x_1+a_2x_2$»*, где a_0 – константа, a_1, a_2 – весовые коэффициенты, определяющие степень влияния входных переменных на значение выходной переменной. Таким образом, основное отличие между выводами по Мамдани и Суджено заключается в способе определения значений выходной переменной в правилах, образующих базу знаний. Значения выходной переменной при выводе по Мамдани задаются нечеткими термами, в системах вывода по Суджено – как линейная комбинация входных переменных.

Рассмотрим необходимый инструментарий MatLab и пакета Fuzzy Logic. В состав Matlab входят следующие средства GUI, обеспечивающие доступ к инструментарию нечеткой логики: редакторы системы нечеткого вывода (СНВ), функции

принадлежности (ФП), правил вывода, а также средства просмотра правил и поверхности вывода. Эти средства связаны между собой динамически (изменения в одном из них влекут изменения в других).

Редактор СНВ предоставляет возможность формирования проектируемой системы вывода на уровне количества входных и выходных переменных, наименования переменных. Редактор ФП используется для определения формы функций принадлежности для каждой нечеткой переменной. Редактор правил вывода применяется для редактирования списка правил, которые определяют поведение проектируемой системы. Средства просмотра правил вывода используются в целях анализа и тестирования системы. С их помощью можно, например, оценить активность правил или форму влияния отдельных ФП на результат нечеткого вывода. Средства просмотра поверхности вывода используются для отображения зависимости выхода системы от одного или двух входов. Другими словами, они генерируют и выводят карту поверхности разработанной системы нечеткого вывода. Приведем основные сведения, необходимые для практического использования указанных инструментальных средств.

Редактор СНВ. Для создания нечеткой системы в командной строке основного окна Matlab необходимо набрать команду `fuzzy`. Окно редактора СНВ содержит входную переменную, обозначенную `input 1`, и выходную переменную, обозначенную `output 1`. По умолчанию инструментарий нечеткой логики использует СНВ по Мамдани.

Для того чтобы добавить новую переменную, необходимо выбрать в меню Edit–Add variable соответствующий пункт (для входной переменной – Input, для выходной переменной – Output). Чтобы изменить имя переменной, необходимо ее вначале отметить, а затем в поле редактирования изменить имя переменной, заданное по умолчанию на имя, предложенное пользователем.

Сохранение проектируемой системы в рабочем пространстве среды Matlab производится с помощью пункта меню File–Export–To Workspace. В этом случае данные сохраняются до окончания сеанса работы с Matlab. Для сохранения данных на диске после окончания сеанса работы применяется соответствующий пункт того же меню To Disk.

Редактор ФП. Следующим шагом в построении нечеткой модели вывода средствами инструментария нечеткой логики является определение функций принадлежности для каждой входной и выходной переменных. Данная операция производится в редакторе ФП путем выбора в меню Edit пункта Membership Functions или двойным щелчком мыши на изображении соответствующей переменной (входной или выходной) или набором в командной строке оператора `mfedit`.

Редактор ФП позволяет отображать и редактировать любые функции принадлежности всех входных и выходных переменных проектируемой системы нечеткого вывода. Чтобы связать функцию принадлежности с именем переменной, необходимо вначале выбрать нужное имя базовой переменной из набора графических объектов окна редактора ФП, далее указать диапазон изменения ее значений и диапазон для текущей переменной, наконец, в меню Edit выбрать пункт Add MFs и в появившемся окне - вид функций и их количество.

Редактировать функцию принадлежности текущей переменной можно, используя графическое окно редактора ФП или изменяя ее характеристики (имя, тип и числовые параметры). При выборе необходимой функции в графическом окне редактора ФП допускается плавное изменение функции с помощью мыши.

Таким образом, проектируя систему нечеткого вывода с помощью редактора ФП, можно определить соответствующие функции для каждой из входных и выходных переменных.

Редактор правил вывода. После того как указано количество входных и выходных переменных, определены их наименования и построены соответствующие функции принадлежности, необходимо вызвать редактор правил вывода. Для этого в меню Edit выбирается пункт Rules или в командной строке среды Matlab набирается команда `ruleedit`.

Основываясь на описаниях входных и выходных переменных, определенных в редакторе ФП, редактор правил вывода формирует структуру правил автоматически. От пользователя требуется установить связи между входными и выходными переменными, выбирая их из списка ранее заданных функций принадлежности и определяя логические операции (AND, OR) между ними. Также

допускается использование логического отрицания (NOT) и изменение «веса» правил в диапазоне от 0 до 1.

Правила вывода могут отображаться в окне редактора в различных форматах, которые выбираются из подменю Format меню Options. По умолчанию используется расширенный формат отображения правил вывода (Verbose) следующего вида:

```
If (input_1 is [not] mf_1j1) <and, or>…
(input_i is [not] mf_ijі)…<and, or>
(input_n is [not] mf_njn) then
(output_1 is [not] mf_1 + 1jn + 1) <and, or>…
(output_k is [not] mf_k + njk + n) <and, or>…
(output_m is [not] mf_m + njm + n) (w),
```

где i – номер входной переменной, j_i – номер ФП i-й переменной, k – номер выходной переменной, n – количество входных переменных, m – количество выходных переменных, w – вес правила. Круглые скобки заключают в себе обязательные параметры, квадратные – необязательные, а угловые – альтернативные параметры (один на выбор).

Кроме формата по умолчанию, существуют еще два вида форматов отображения правил: символьный (Symbolic) и индексный (Indexed). Символьный формат имеет следующий вид:

```
(input_1<~=,==>mf_1j1)<&,|>…
(input_i<~=,==>mf_iji) <&,|>…
(input_n<~=,==>mf_njn)=>
(output_1<~=,==>mf_n + 1jn+1) <&,|>…
(output_k<~=,==>mf_k + nkk+n)<&,|>…
(output_m<~=,==>mf_m + njm+n) (w).
```

Отличие символьного формата от расширенного состоит в том, что вместо словесной интерпретации логических операций используются символы «&» и «|», которые соответственно определяют логическое И и логическое ИЛИ, символ «~» – вместо логического отрицания, а символ «=>» является разделителем условной и заключительной частей правила (антецедент и консеквент).

Индексный формат имеет следующий вид:

```
[-] 1j1…[-] ijі…[-] njn[-] n + 1jn+1…[-] k + njk+1…[-] m
+ njm+n(w):<1,2>.
```

Здесь порядок следования чисел соответствует очередности вводимых переменных, причем символ «,» разделяет правило на условную и заключительную части. До двоеточия записывается порядковый номер соответствующей функции принадлежности, после двоеточия – вид логической связки («1» – логическое И, «2» – логическое ИЛИ). Логическое отрицание задается символом «-».

После определения правил вывода в одноименном редакторе можно утверждать, что система нечеткого вывода полностью специфицирована.

Средства просмотра правил вывода. Просмотр правил вывода позволяет отобразить процесс нечеткого вывода и визуализировать результат. Главное окно просмотра состоит из нескольких графических окон, располагаемых по строкам и столбцам. Количество строк соответствует числу правил нечеткого вывода, а количество столбцов – числу входных и выходных переменных, заданных в разрабатываемой СНВ. Дополнительное графическое окно служит для отображения результата нечеткого вывода и операции дефазификации. В каждом окне отображается соответствующая функция принадлежности, уровень ее среза (для входных переменных) и вклад отдельной функции в общий результат (для выходных переменных). В нижней части главного окна можно отобразить номера правил вывода в различных форматах вывода, отмечая их мышью.

Для изменения формата в меню Options выбирается пункт Rule display format.

Изменение значений входных переменных допустимо либо путем ввода в поле `Input` записи входного вектора, размерность которого равна количеству входных переменных, либо щелчком мыши в любом графическом окне, которое относится к изменяемой входной переменной. Входной вектор в каждом из этих вариантов изменения будет задаваться набором красных вертикальных прямых.

Средства просмотра поверхности вывода. Просмотр поверхности вывода позволяет строить трехмерную поверхность в виде зависимости одной из выходных переменных от двух входных. Выбор входных и выходных переменных осуществляется посредством ниспадающих меню главного окна рассматриваемого программного средства. Количество выводимых линий по осям X и Y определяется в полях ввода `X grids`, `Y grids`.

Построение систем нечеткого вывода по Суджено. Для построения СНВ по Суджено необходимо в меню File выбрать пункт New FIS – Sugeno. Количество входных и выходных переменных определяется так же, как и при построении СНВ по Мамдани.

Редактор функций принадлежности по Суджено отличается только схемой определения функций принадлежности для выходных переменных. Инструментарий нечеткой логики в среде Matlab позволяет разрабатывать два вида нечетких моделей Суджено.

Первая модель – это нечеткая модель Суджено нулевого порядка. Нечеткое правило вывода для нее имеет следующий вид:

```
if x is A and y is B then z = k,
```

где A и B – нечеткие множества антецедента; k – четко заданная константа консеквента. Для построения такой модели при редактировании функций принадлежности необходимо выбрать тип константы (constant) и задать в качестве параметра функции численное значение выбранной константы.

Вторая модель – это нечеткая модель Суджено первого порядка. Для нее нечеткое правило вывода записывается следующим образом:

```
if x as A and y is B then z = p x + q y + r.
```

где p, q и r – константы. В данном случае функции принадлежности являются линейными (linear). Для определения параметров отдельной функции необходимо ввести вектор, элементы которого соответствуют численным значениям констант консеквента.

Редактирование правил вывода, а также просмотр правил и поверхности вывода выполняется аналогично построению систем нечеткого вывода по Мамдани.

3.2. Примеры проектирования системы нечеткого вывода

Проектирование вывода по Мамдани. Рассмотрим основные этапы проектирования вывода по Мамдани на примере создания системы нечеткого логического вывода, моделирующей зависимость

$$y = x_1^2 \cdot \sin(x_2 - 1), \qquad \text{где} \qquad x_1 \in [-7;3], x_2 \in [-4.4;1.7].$$

Проектирование системы нечеткого логического вывода будем проводить на основе графического изображения указанной зависимости.

Для построения эталонного трехмерного изображения функции $y = x_1^2 \cdot \sin(x_2 - 1)$ в области $x_1 \in [-7;3], x_2 \in [-4.4;1.7]$ составляется следующая программа на языке MatLab:

```
    n=15; %количество точек разбиения интервала
x1=-7:10/(n-1):3; %область значений x1
x2=-4.4:6.1/(n-1):1.7; %область значений x2
y=zeros(n,n); %обнуление y
for j=1:n
y(j,:)=x1.^2*sin(x2(j)-1); %вычисление значений y
end
surf(x1,x2,y) %построение изображения функции
xlabel('x1') %подпись оси x
ylabel('x2') %подпись оси y
zlabel('y')  %подпись оси z
title('Target'); %подпись графика
```

В результате выполнения программы в среде MatLab получим графическое изображение, приведенное на рис. 3.2. Для проектирования соответствующей системы нечеткого вывода необходимо выполнить следующую последовательность шагов.

Шаг 1. Для загрузки редактора СНВ напечатаем команду `fuzzy` в командной строке. После этого откроется нового графическое окно, показанное на рис. 3.3.

Шаг 2. Добавим вторую входную переменную.

Шаг 3. Переименуем первую входную переменную в `x1`, вторую входную переменную – в `x2`, а выходную переменную – в `y`.

Шаг 4. Зададим имя системы. Для этого в меню File выбираем в подменю Export и команду To Disk. Вводим имя файла, например, first.

Шаг 5. Перейдем в редактор ФП.

Шаг 6. Зададим диапазон изменения переменной `x1`. Для этого вводим `[ -7  3]` в поле `Range` (рис. 3.4).

Шаг 7. Зададим функцию принадлежности переменной `x1`. Для лингвистической оценки этой переменной будем использовать 3 терма с треугольными функциями принадлежности. Для этого в меню Edit выберем команду Add MFs. В результате появится диалоговое окно выбора типа и количества функций принадлежностей. По умолчанию это 3 терма с треугольными функциями принадлежности.

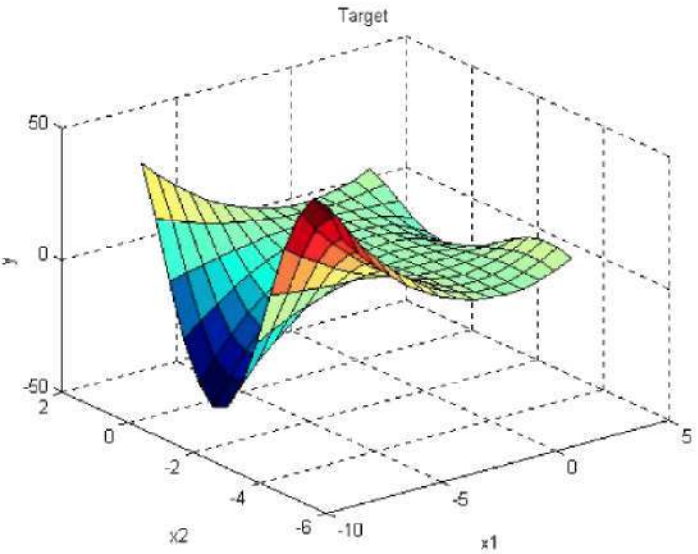


Рис.3.2. Эталонная поверхность

Шаг 8. Зададим наименования термов переменной `x1`. Для этого делаем один щелчок левой кнопкой мыши по графику первой функции принадлежности (см. рис. 3.4) и задаем, например, наименования термов: низкий, средний, высокий. В результате получим графическое окно, изображенное на рис. 3.4.

Шаг 9. Зададим функцию принадлежности переменной `x2`. Для лингвистической оценки этой переменной будем использовать 5 термов с колоколообразными (гауссовскими) функциями принадлежности. Для этого активизируем переменную `x2` с помощью щелчка левой кнопки мыши на блоке `x2`. Зададим диапазон изменения переменной `x2`. Для этого вводим `[-4.4 1.7]` в поле `Range` (рис. 3.5). Затем в меню Edit выберем команду Add MFs. В появившемся диалоговом окне выбираем тип функции принадлежности `gaussmf` в поле `MF type` и 5 термов в поле `Number of MFs`.

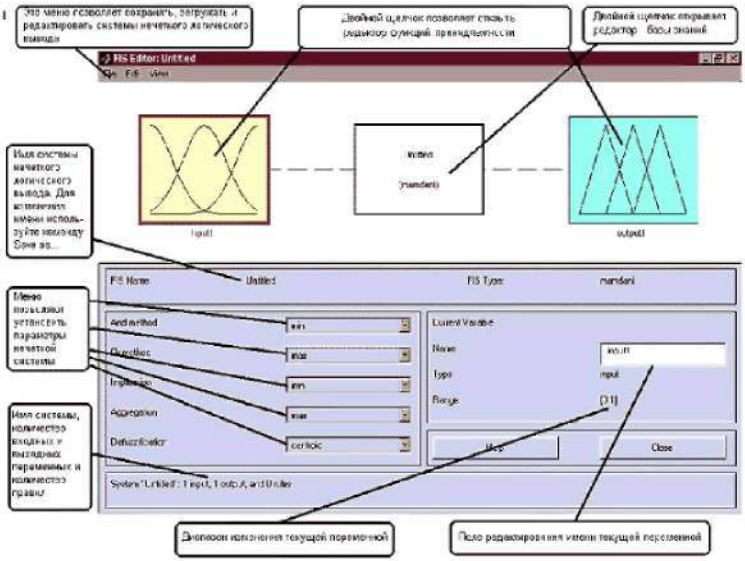


Рис.3.3. Окно редактора FIS-Editor

Шаг 10. Зададим следующие наименования термов переменной x2: низкий, нижесреднего, средний, вышесреднего, высокий. В результате получим графическое окно, изображенное на рис. 3.5.

Шаг 11. Зададим функции принадлежности переменной y. Для лингвистической оценки этой переменной будем использовать 5 термов с треугольными функциями принадлежности. Для этого активизируем переменную y с помощью щелчка левой кнопки мыши на блоке y. Зададим диапазон изменения переменной y. Для этого вводим [-50 50] в поле Range (рис. 3.6). Затем в меню Edit выберем команду Add MFs. В появившимся диалоговом окне выбираем 5 термов в поле Number of MFs.

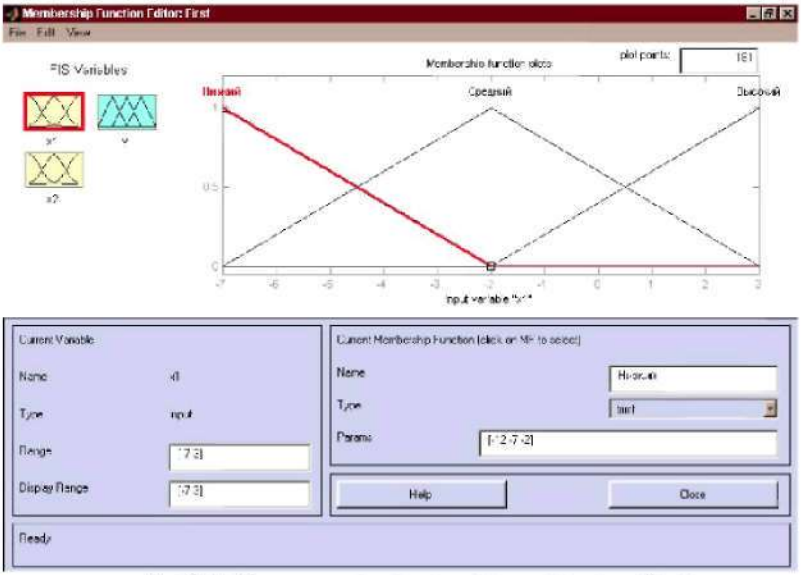


Рис.3.4. Функции принадлежности переменной x1

Шаг 12. Зададим следующие наименования термов переменной y: низкий, нижесреднего, средний, вышесреднего, высокий. В результате получим графическое окно, изображенное на рис. 3.6.

Шаг 13. Перейдем в редактор базы знаний RuleEditor. Для этого выберем в меню Edit команду Edit rules.

Шаг 14. На основе визуального наблюдения за графиком, изображенным на рис. 3.2, сформулируем следующие девять правил.

1. Если x1=средний, то y=средний.
2. Если x1=низкий и x2=низкий, то y=высокий.
3. Если x1=низкий и x2=высокий, то y=высокий.
4. Если x1=высокий и x2=высокий, то y=вышесреднего.
5. Если x1=высокий и x2=низкий, то y=вышесреднего.
6. Если x1=высокий и x2=средний, то y=средний.
7. Если x1=низкий и x2=средний, то y=низкий.
8. Если x1=высокий и x2=вышесреднего, то y=средний.
9. Если x1=высокий и x2=нижесреднего, то y=средний.

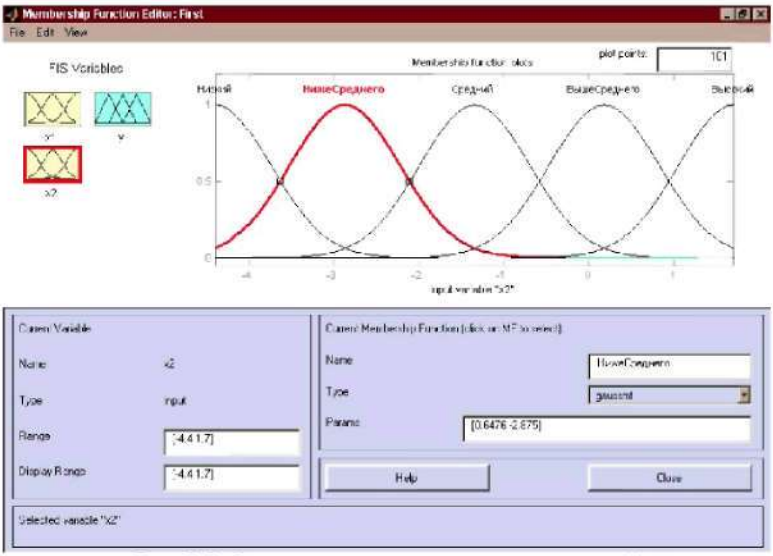


Рис. 3.5. Функции принадлежности переменной x2

Для ввода правила необходимо выбрать в меню соответствующую комбинацию термов и нажать кнопку Add rule. На рис. 3.7 изображено окно редактора базы знаний после ввода всех девяти правил. Число, приведенное в скобках в конце каждого правила, представляет собой весовой коэффициент соответствующего правила.

Шаг 17. Сохраним созданную систему на диске(To Disk).

На рис. 3.8 приведено окно визуализации нечеткого логического вывода. Это окно активизируется командой View rules меню View. В поле Input указываются значения входных переменных, для которых выполняется логический вывод.

На рис. 3.9 приведена поверхность «входы-выход», соответствующая синтезированной нечеткой системе. Для вывода этого окна необходимо использовать команду View surface меню View. Визуально сравним поверхности на рис. 3.2 и на рис. 3.9.

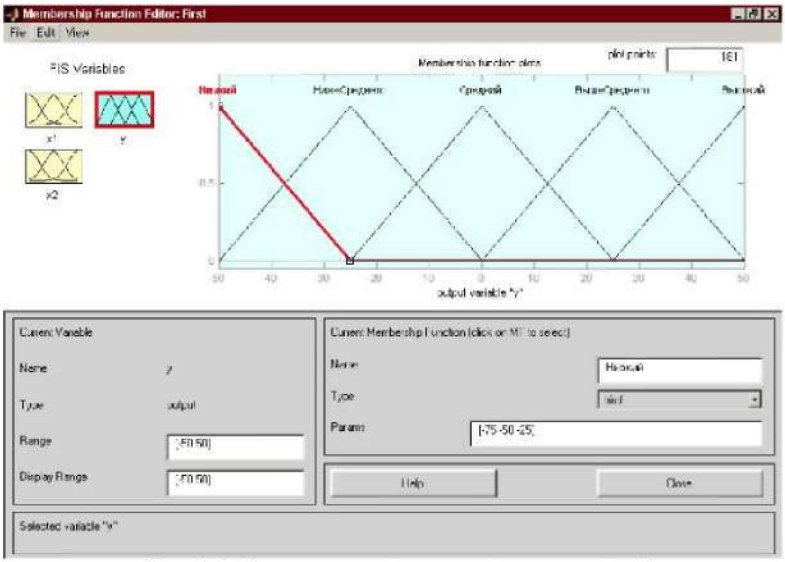


Рис. 3.6. Функции принадлежности переменной y

Анализ значений в разных точках показывает их сходство, т.е. сформулированные нечеткие правила достаточно точно описывают исходную нелинейную зависимость, а полученная трехмерная модель почти идентична эталонной поверхности.

Проектирование вывода по Суджено. Рассмотрим основные этапы проектирования вывода по Суджено на примере создания системы нечеткого логического вывода, моделирующей зависимость $y = x_1^2 \cdot \sin(x_2 - 1)$, $x_1 \in [-7;3], x_2 \in [-4.4;1.7]$ (рис. 3.1). Моделирование этой зависимости будем осуществлять с помощью следующей базы из 6 правил.

1. Если x1=средний, то y=0.
2. Если x1=высокий и x2=высокий, то y=2x1+2x2+1.
3. Если x1=высокий и x2=низкий, то y=4x1-x2.
4. Если x1=низкий и x2=средний, то y=8x1+2x2+8.
5. Если x1=низкий и x2=низкий, то y=50.
6. Если x1=низкий и x2=высокий, то y=50.

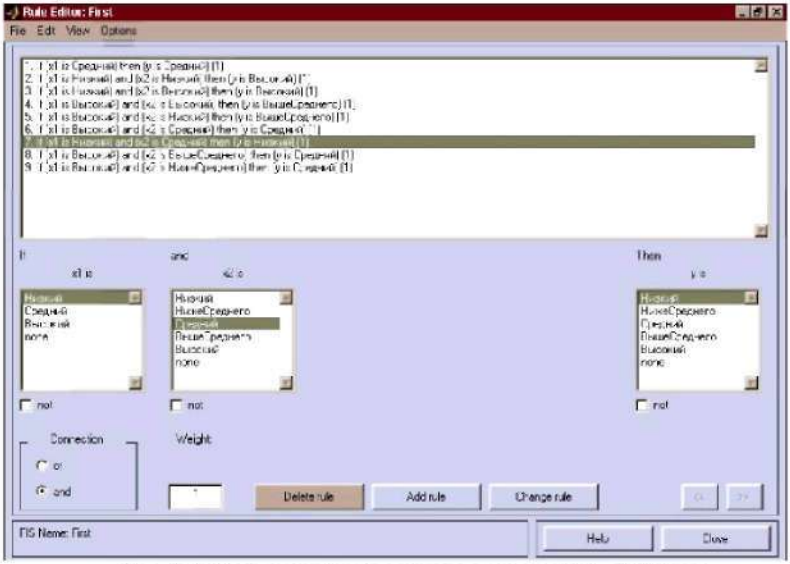


Рис. 3.7. База знаний в редакторе правил `RuleEditor`

Проектирование системы нечеткого вывода по Сугено состоит в выполнении следующей последовательности шагов.

Шаг 1. Для загрузки редактора СНВ напечатаем команду `fuzzy` в командной строке. После этого откроется графическое окно (рис. 3.3).

Шаг 2. Выберем тип системы. Для этого в меню File выбираем в подменю New fis и команду Sugeno.

Шаг 3. Добавим вторую входную переменную.

Шаг 4. Переименуем первую входную переменную в `x1`, вторую входную переменную в `x2`, а выходную переменную – в `y`.

Шаг 5. Зададим имя системы. Для этого в меню File выбираем в подменю Export команду To Disk и введем имя файла, например, `FirstSugeno`.

Шаг 8. Перейдем в редактор ФП.

Шаг 9. Зададим диапазон изменения переменной `x1`. Для этого вводим `|-7  3 |` в поле `Range` (рис. 3.10).

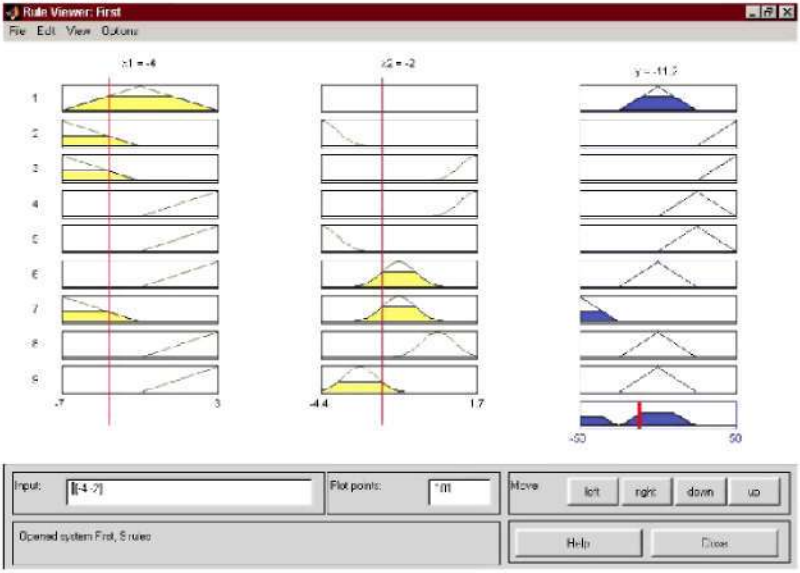


Рис 3.8. Визуализация нечеткого логического вывода в `RuleViewer`

Шаг 10. Зададим функции принадлежности переменной `x1`. Для лингвистической оценки этой переменной будем использовать 3 терма по умолчанию с треугольными функциями принадлежности. Установим наименования термов переменной `x1`: низкий, средний, высокий. В результате получим графическое окно, изображенное на рис. 3.10.

Шаг 11. Зададим ФП переменной `x2`. Для лингвистической оценки этой переменной будем использовать 3 терма с треугольными функциями принадлежности, которые установлены по умолчанию. Для этого активизируем переменную `x2` с помощью щелчка левой кнопки мыши на блоке `x2`. Зададим диапазон изменения переменной `x2`. Для этого введем `|-4.4 1.7|` в поле `Range` (рис. 3.11). По аналогии с предыдущим шагом зададим следующие наименования термов переменной `x2`: низкий, средний, высокий. В результате получим графическое окно, изображенное на рис. 3.11.

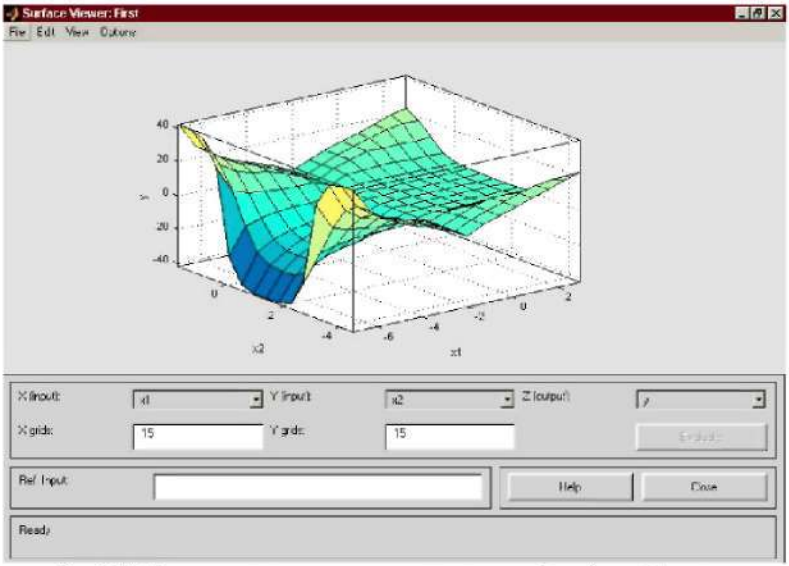


Рис 3.9. Поверхность «входы-выход» в окне `SurfaceViwer`

Шаг 12. Зададим линейные зависимости между входами и выходом, приведенные в базе правил. Для этого активизируем переменную `y` с помощью щелчка левой кнопки мыши на блоке `y`. В правом верхнем углу появилось обозначение трех функций принадлежности, каждая из которых соответствует одной линейной зависимости между входами и выходом. В приведенной базе из 6 правил указаны 5 различных значений `y`: `50`; `4x1-x2`; `2x1+2x2+1`; `8x1+2x2+8`; `0`. Поэтому добавим еще две зависимости путем выбора команды **Add Mfs** меню **Edit**. В появившимся диалоговом окне в поле **Number of MFs** выбираем 2.

Шаг 13. Зададим наименования и параметры линейных зависимостей. Для этого делаем один щелчок левой кнопкой мыши по наименованию первой зависимости `mf1`. Затем вводим наименование зависимости, например `50`, в поле `Name`, и устанавливаем тип зависимости – константа путем выбора опции `Constant` в меню `Type`. После этого вводим значение параметра `50` в поле `Params`.

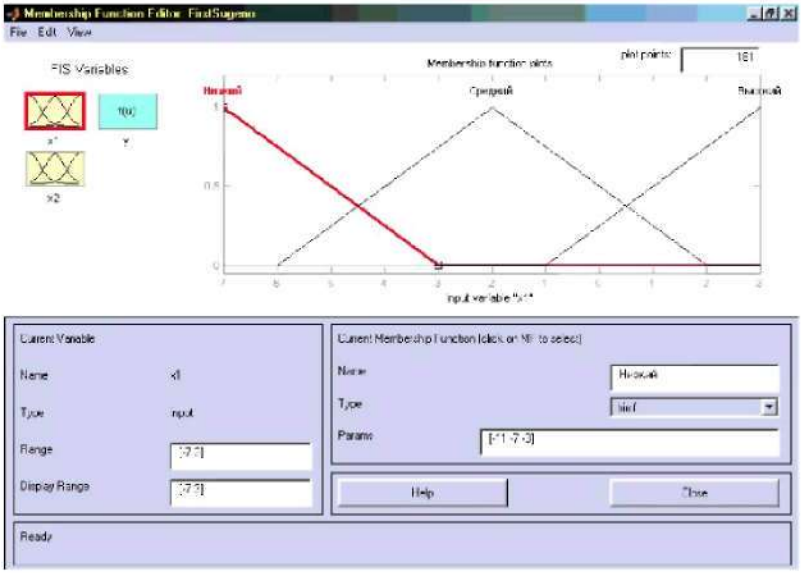


Рис 3.10. Функции принадлежности переменной `x1`

Аналогично для второй зависимости `mf2` введем наименование зависимости, например `8+8x1+2x2`. Затем укажем линейный тип зависимости путем выбора опции `Linear` в меню `Type` и введем параметры зависимости `8 2 8` в поле `Params`. Для линейной зависимости порядок параметров следующий: первый параметр – коэффициент при первой переменной, второй – при второй и т.д., и последний параметр – свободный член зависимости. Для третьей зависимости `mf3` введем наименование зависимости, например `1+2x1+2x2`, укажем линейный тип зависимости и введем параметры зависимости `2 2 1`. Для четвертой зависимости `mf4` введем наименование зависимости, например `4x1-x2`, укажем линейный тип зависимости и введем параметры зависимости `4 -1 0`. Для пятой зависимости `mf5` введем наименование зависимости, например `0`, укажем тип зависимости (константа) и введем параметр зависимости `0`.

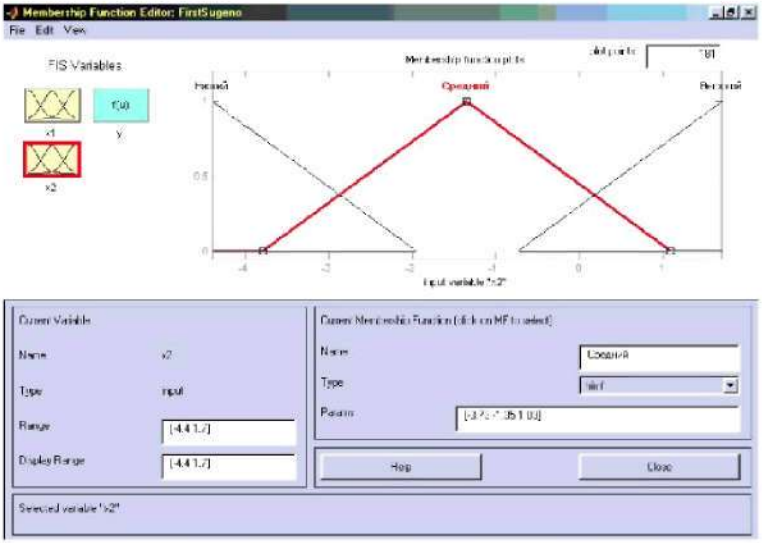


Рис. 3.11. Функции принадлежности переменной x2

В результате получим графическое окно, изображенное на рис. 3.12.

Шаг 14. Перейдем в редактор базы знаний RuleEditor. Для этого выберем в меню Edit команду Edit rules и введем правила из базы знаний. Для ввода правила необходимо выбрать соответствующую комбинацию термов и зависимостей и нажать кнопку Add rule. На рис. 3.13 изображено окно редактора базы знаний после ввода всех шести правил.

На рис. 3.14 приведено окно визуализации нечеткого логического вывода. Это окно активизируется командой View rules меню View. В поле Input указываются значения входных переменных, для которых выполняется логический вывод. Как видно из этого рисунка, значение выходной переменной рассчитывается как среднее взвешенное значение результатов вывода по каждому правилу.

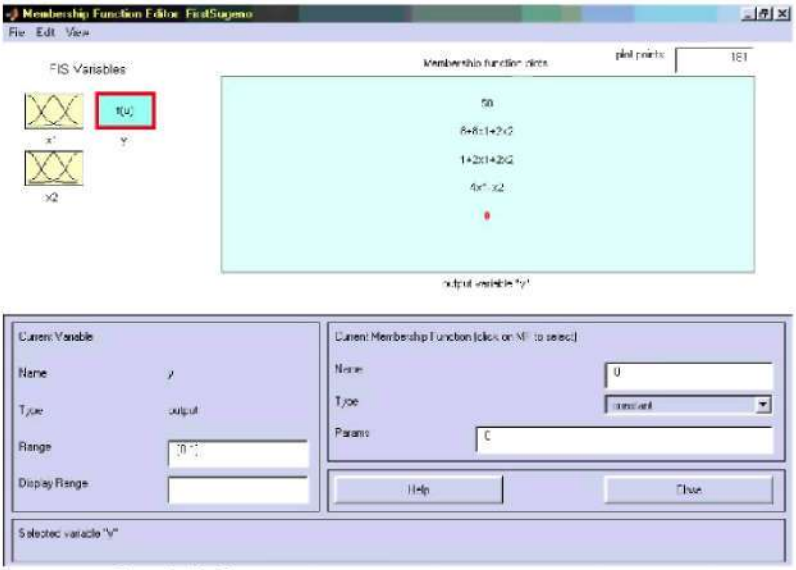


Рис 3.12. Окно линейных зависимостей «входы-выход»

На рис. 3.15 также приведена поверхность «входы-выход», соответствующая синтезированной нечеткой системе. Для вывода этого окна необходимо использовать команду View surface меню View. Можно отметить, что сформулированные нечеткие правила достаточно точно описывают сложную эталонную нелинейную зависимость, а полученная трехмерная модель с высокой степенью достоверности может быть признана идентичной эталонной поверхности.

Визуально сравним поверхности на рис. 3.9 и на рис. 3.15.

Анализ разных точек показывает сходство этих поверхностей. При этом модель Суджено для примера более точно соответствует эталонной модели, нежели модель по Мамдани. Однако в данном случае преимущество модели Мамдани состоит в том, что правила базы знаний являются более понятными, в отличие от модели Суджено, в которой не ясно, какие линейные зависимости «входы-выход» необходимо использовать.

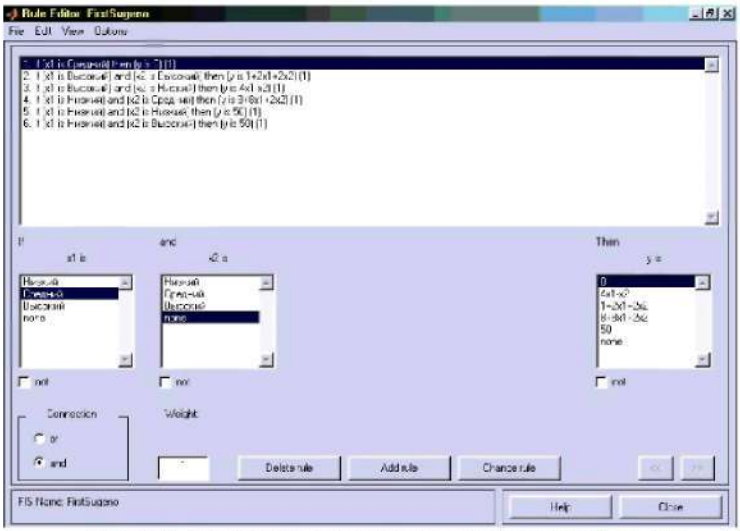


Рис 3.13. Нечеткая база знаний для системы типа Суджено

3.3. Порядок выполнения работы

Сущность задания лабораторной работы состоит в том, чтобы с помощью инструментария нечеткой логики интерактивной системы MatLab реализовать нечеткую систему управления. Для этого необходимо построить базу знаний, описывающую объект управления, произвести ее отладку, проверку работоспособности на выбранном варианте задания, проанализировать результаты, подготовить и защитить отчет.

Этапы выполнения лабораторной работы:

- изучить теоретический материал, рассмотреть примеры;
- четко представить себе, что необходимо сделать согласно своему варианту;

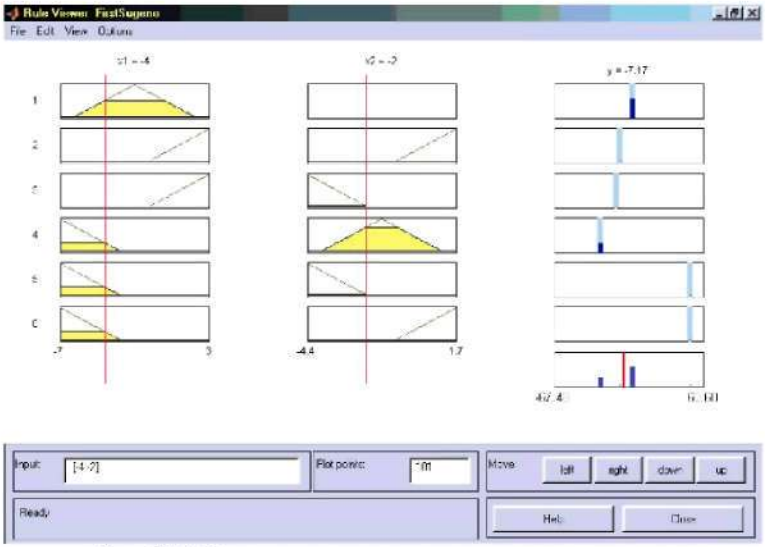


Рис. 3.14. Визуализация нечеткого логического вывода по Сугено

- описать выбранный объект управления в терминах нечеткой логики;
- реализовать с помощью инструментария нечеткой логики интерактивной системы для технических расчетов MatLab нечеткую систему управления, моделирующую заданную ситуацию;
- реализовать в построенной базе знаний нечеткий логический вывод по Мамдани и Сугено при заданных значениях входных признаков, описывающих объект управления и сравнить полученные результаты;
- представить отчет о работе, включающий авторские выводы и оценку достижения поставленных целей.

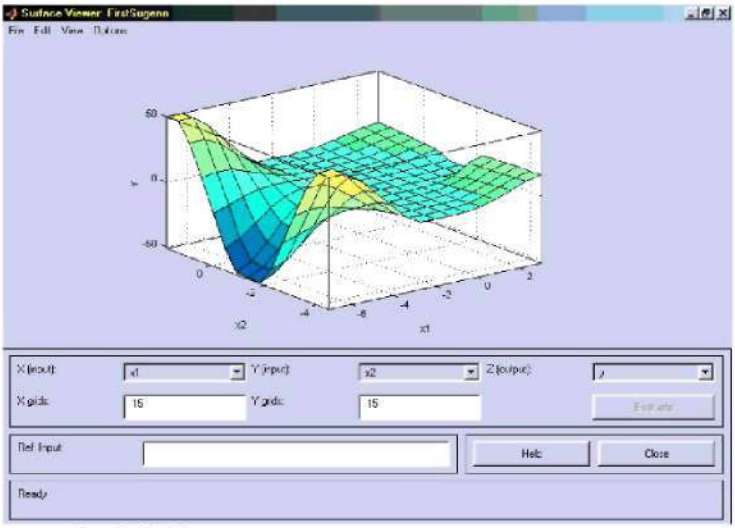


Рис. 3.15. Поверхность «входы-выход» для системы вывода по Суджено

3.4. Содержание отчета

1. Цель работы.
2. Постановка задачи согласно варианту.
3. Описание самостоятельно проделанной работы (анализ нечетких моделей и объекта управления; задание функций принадлежности; описание работы в системе MatLab; результаты вывода по Мамдани и Суджено, их сравнительная характеристика).
4. Выводы.

Приложение: необходимые для защиты лабораторной работы «скриншоты» экрана.

3.5. Варианты заданий

Представленные ниже варианты включают задачи проектирования базы знаний для систем регулирования температуры в помещении, оценки уровня опасности движущегося автомобиля, а также управления светофором. Каждый вариант предусматривает самостоятельную разработку нечетких моделей управления, а также их практическую реализацию в системе MatLab.

Вариант задания выбирается студентом по согласованию с преподавателем.

Конкретные варианты заданий для указанных выше задач проектирования систем нечеткого управления представлены ниже и включают конкретные технические условия и ограничения.

1. Модель регулирования температуры в помещении:

- объект управления – регулятор температуры;
- цель управления – обеспечение комфортной температуры в помещении;
- признаками оценки ситуации являются температура воздуха в помещении (T), скорость изменения температуры в помещении (TC), объем помещения (V);
- управление состоит в изменении температуры воздуха (TD);
- диапазоны четких значений и наборы термов признаков и управляющего воздействия приведены в табл. 3.1 – 3.4.

Таблица 3.1

№ \ T(C°)	Диапазон	Набор термов
1	5-45	«малая», «средняя», «высокая»
2	10-50	«очень малая», «малая», «средняя», «высокая»
3	0-40	«малая», «средняя», «высокая», «очень высокая»
4	15-60	«малая», «небольшая», «средняя», «высокая»
5	0-60	«малая», «средняя», «довольно высокая», «высокая»

Таблица 3.2

T(С/мин) / №	Диапазон	Набор термов
1	0-50	«малая», «средняя», «высокая»
2	0-60	«очень малая», «малая», «средняя», «высокая»
3	5-40	«малая», «средняя», «довольно высокая», «высокая»
4	0-40	«малая», «средняя», «высокая», «очень высокая»
5	10-50	«малая», «небольшая», «средняя», «высокая»

Таблица 3.3

V($м^3$) / №	Диапазон	Набор термов
1	10-100	«малая», «средняя», «высокая», «очень высокая»
2	30-150	«малая», «средняя», «довольно высокая», «высокая»
3	25-75	«очень малая», «малая», «средняя», «высокая»
4	15-80	«малая», «небольшая», «средняя», «высокая»
5	30-200	«малая», «средняя», «высокая», «очень высокая»

Таблица 3.4

T(C^0) / №	Диапазон	Набор термов
1	-5 – 5	«небольшое», «среднее», «большое»
2	-2 – 2	«небольшое», «среднее», «очень большое»
3	-1 – 1	«малое», «среднее», «большое»
4	-3 – 3	«малое», «среднее», «довольно большое»
5	-4 – 4	«небольшое», «среднее», «большое»

2. Модель оценки уровня опасности движущегося автомобиля:

- объект оценки – движущийся по автодороге легковой автомобиль;
- цель управления – обеспечение безопасности движения. В случае повышения уровня опасности движения система должна сигнализировать водителю о возможности возникновения аварийной ситуации на дороге;
- признаками оценки ситуации являются скорость движения автомобиля (S), расстояние до находящего впереди препятствия (DR), расстояние до правой обочины (DT);
- оценивается уровень опасности (L);
- диапазоны четких значений, наборы термов признаков и самой оценки приведены в табл. 3.5 – 3.8.

Таблица 3.5

S(км/ч) / №	Диапазон	Набор термов
1	0-90	«малая», «средняя», «большая»
2	0-150	«очень малая», «малая», «большая», «очень большая»
3	0-100	«малая», «средняя», «большая», «очень большая»
4	0-120	«очень малая», «малая», «средняя», «большая»
5	0-111	«малая», «средняя», «довольно большая»

Таблица 3.6

DP(м) / №	Диапазон	Набор термов
1	0-80	«очень малое», «малое», «небольшое»
2	0-100	«критическое», «малое», «небольшое»
3	0-120	«малое», «среднее», «большое»
4	0-150	«очень малое», «малое», «высокая», «очень высокая»
5	0-110	«малая», «небольшое», «большое»

Таблица 3.7

DT(м) / №	Диапазон	Набор термов
1	0-5	«малое», «среднее», «большое»
2	0-8	«очень малое», «малое», «небольшое»
3	0-6	«критическое», «малое», «большое»
4	0-10	«малое», «небольшое», «большое»
5	0-5	«малое», «среднее», «большое»

Таблица 3.8

L(ед) / №	Диапазон	Набор термов
1	0-1	«низкий», «небольшой», «высокий», «критический»
2	0-0,9	«довольно малый», «малый», «высокий»
3	0-0,8	«малый», «средний», «довольно большой»
4	0-0,7	«предельно малый», «малый», «невысокий», «высокий»
5	0-1	«низкий», «средний», «довольно высокий», «критический»

3. Модель управления светофором:

- объект управления – светофор;
- цель управления – обеспечить переключение светофора в зависимости от ситуации, наблюдаемой на перекрестке. Управление светофором осуществляется при помощи решений «открыть движение» (зеленый цвет), «не открывать движение» (красный цвет);
- признаками оценки ситуации являются количество автомобилей на закрытом направлении (CA), на открытом направлении (OA), пешеходов на закрытом (CP) и на открытом направлениях (OP);
- управление состоит в переключении светофора (U);
- диапазоны четких значений и наборы термов признаков и управляющего воздействия приведены в табл. 3.9 – 3.13.

Таблица 3.9

CA(шт) / №	Диапазон	Набор термов
1	0-20	«малое», «небольшое», «большая»
2	0-30	«довольно малое», «среднее», «большое»
3	0-15	«малое», «среднее», «довольно большое»
4	0-25	«очень малое», «малое», «довольно большое»
5	0-40	«предельно малое», «малое», «большое»

Таблица 3.10

QA(шт) / №	Диапазон	Набор термов
1	0-40	«небольшое», «среднее», «большое»
2	0-20	«малое», «небольшое», «большое»
3	0-30	«очень малое», «среднее», «большое»
4	0-15	«предельно малое», «малое», «большое»
5	0-25	«малое», «среднее», «большое»

Таблица 3.11

CP(шт) / №	Диапазон	Набор термов
1	0-5	«малое», «среднее», «большое»
2	0-8	«очень малое», «малое», «довольно большое»
3	0-6	«предельно малое», «малое», «большое»
4	0-10	«довольно малое», «малое», «большое»
5	0-5	«малое», «небольшое», «большое»

Таблица 3.12

QP(шт) / №	Диапазон	Набор термов
1	0-60	«очень малое», «малое», «большое»
2	0-40	«малое», «среднее», «большое»
3	0-30	«малое», «небольшое», «большое»
4	0-100	«малое», «небольшое», «очень большое»
5	0-50	«малое», «среднее», «очень большое»

Таблица 3.13

U(ед) / №	Диапазон	Набор термов
1	0-5	«изменить цвет», «не менять цвет»
2	0-10	«переключить», «не переключать»
3	0-2	«открыть движение», «не открыть движение»
4	0-6	«изменить цвет», «не менять цвет»
5	0-8	«переключить», «не переключить»

Контрольные вопросы

1. Основные формы фазирования функции принадлежности нечетких множеств: а) круг, б) колокол, в) трапеция, г) треугольник д) квадрат.
2. В нечеткой логике степень истинности конъюнкции нескольких высказываний определяется: а) наиболее правдоподобным, б) наименее правдоподобным, в) средним значением.
3. Основными частями интерактивной системы Matlab являются: а) база данных, б) библиотека математических функций, в) программный интерфейс, г) рабочая память, д) среда MatLab, е) управляемая графика, ж) язык MatLab.
4. Пакет Fuzzy Logic Toolbox позволяет: а) определять нечеткие переменные, б) выполнять интерактивное динамическое моделирование, в) визуализировать результаты нечеткого вывода, г) строить фреймы и продукционные системы, д) определять функции принадлежности.
5. Из каких правил состоит база знаний с выводом по Мамдани?
6. Из каких правил состоит база знаний с выводом по Суджено?
7. Значения выходной переменной при выводе по Суджено задаются: а) нечеткими термами, б) как линейная комбинация входов, в) как нелинейная комбинация входов.
8. Редактор системы нечеткого вывода пакета Fuzzy Logic Toolbox запускается командой: а) fuzzy, б) fuzzy_logic, в) neural.
9. Редактор функций принадлежности пакета Fuzzy Logic Toolbox запускается: а) выбором в меню Edit пункта Membership Functions, б) набор в командной строке оператора mfedit

в) выбор в меню Edit, пункта Rules, г) двойным щелчком мыши на изображении соответствующей переменной (входной или выходной).

10. Ваша сравнительная характеристика результатов нечеткого логического вывода по Мамдани и Суджено.

11. Ваша самооценка самостоятельно выполненной работы по критерию 1: 2_1_0 по критерию 2: 2_1_0 по критерию 3: 2_1_0.

Лабораторная работа №4

4. Программирование искусственной нейронной сети в среде MatLab

Целью работы является программирование модели искусственной нейронной сети для решения задач распознавания образов и аппроксимации в интерактивной среде MatLab.

4.1. Краткое теоретическое описание

К феноменам мозга относят кодирование информации о внешнем мире, оперативную и долговременную память, ассоциативный поиск и самоорганизацию памяти, оперирование информацией в процессе решения интеллектуальных задач, практически мгновенное распознавание образов, инсайт и др. Конструктивного научного объяснения этим феноменам пока не найдено. Мозг обладает огромным запасом «прочности», что позволяет ему парировать отказы и приспосабливаться к изменению внешних условий.

С момента появления одной из первых обучаемых искусственных нейронных сетей (персептрон Розенблатта – электромеханическое устройство, моделирующее глаз улитки и его взаимодействие с мозгом, позволяющее различать буквы латинского алфавита, но очень чувствительное к их написанию) прошло примерно полстолетия. Главным содержанием технологий искусственных нейронных сетей (ИНС) является создание электронных и программных аналогов естественных нейронных сетей и использование этих аналогов для имитации функций человеческого интеллекта. Потенциальными сферами применения нейротехнологий и нейрокомпьютинга являются все плохо формализуемые предметные области, в которых классические математические модели и алгоритмы оказываются мало эффективными по сравнению с человеком. Значимые практические результаты применения ИНС уже получены в области обработки изображений, реализации ассоциативной памяти, систем управления реального времени, распознавания образов, систем безопасности, выявления профилей интересов пользователей Internet, систем анализа финансового рынка и др.

Сегодняшняя парадигма нейрокомпьютинга состоит в разработке алгоритмов обучения, специализированных для операций с различными образами и адаптированных под конкретные информационные задачи. Структура работ в области технологий нейрокомпьютинга весьма обширна. Она подробно представлена в [3] и других многочисленных публикациях. Ниже приводятся лишь основные понятия теории ИНС, необходимые для понимания существа лабораторной работы.

Нейрон – базовый нелинейный элемент сети. *Синапс* – элемент, обеспечивающий связь между нейронами; может обладать весом и «задержкой». *Сумматор* – компонент нейрона (или специализированный нейрон), выполняющий сложение сигналов, поступающих по синаптическим связям от других нейронов и внешних входных сигналов. *Обучающие примеры* – наборы вход-выходных значений и целевой функции, определяющей величину отклонения реального выхода от эталонных значений при данных входах. *Алгоритм обучения ИНС* – процесс нахождения экстремума некоторой функции, отображающей взаимодействие входов и выходов ИНС. В ИНС, как правило, используются модели простых нейронов, реализующих элементарные нелинейные функции (пороговые, сигмоидальные и т.д.). Например, классический *персептрон* представляет собой сеть с пороговыми нейронами и входными сигналами, равными нулю или единице. Персептрон должен решать задачу классификации на два класса по бинарным входным сигналам. Набор входных сигналов обозначим n-мерным вектором φ. Все элементы вектора являются булевыми переменными. Согласно Ф. Розенблатту, персептроном называется нелинейный преобразователь, на вход которого подаются некоторые сигналы, а с выхода снимается значение, связанное с входными через следующую функцию:

$$\psi = \left[\sum_{i=1}^{n} \alpha_i \omega_i > \theta \right],$$

- где α_i – веса персептрона, θ – порог, φ_i – значения входных сигналов, квадратные скобки [ ] означают возможный переход от булевых значений к числовым. В качестве входных сигналов отдельного нейрона могут выступать как внешние входные сигналы

сети, так и выходные значения других нейронов. Если в функцию Ψ добавить постоянный единичный входной сигнал $\varphi_0 = 1$ и положить $\alpha_0 = -\theta$, то персептрон можно переписать в следующем виде:

$$\psi = \left[\sum_{i=0}^{m} \alpha_i \omega_i > 0\right].$$

Очевидно, что Ψ вычисляется одним нейроном с пороговым нелинейным преобразователем. Каскад из нескольких слоев таких нейронов называют *многослойным персептроном*. При объединении множества нейронов в сеть различают входные, скрытые и выходные «слои», связь между которыми может быть однонаправленной или с «обратным распространением» (рекуррентные сети, например, сеть Хопфилда). В рекуррентных сетях, в отличие от однонаправленных сетей прямого распространения, в которых при заданной комбинации входных сигналов генерируются выходные значения, не зависящие от предыдущего состояния сети; модифицируются входы нейронов, что влечет изменение в сети до тех пор, пока не наступит устойчивое состояние.

Характерной чертой нейротехнологий является *обучение нейросети на примерах*. Например, персептрон можно обучать по *правилам Хебба*. Для этого предъявляют на вход персептрона один пример. Если выходной сигнал персептрона совпадает с правильным ответом, то никаких действий предпринимать не надо. В случае ошибки необходимо обучить персептрон правильно решать данный пример. Ошибки могут быть двух типов. Рассмотрим каждый из них.

Первый тип ошибки – на выходе персептрона 0, а правильный ответ – 1. Для того чтобы персептрон выдавал правильный ответ необходимо, чтобы сумма в правой части формулы Ψ стала больше порога. Поскольку входные переменные принимают значения 0 или 1, то увеличение суммы может быть достигнуто за счет увеличения весов α_i. Однако нет смысла увеличивать веса связей для тех переменных φ_i, которые равны нулю. Необходимо увеличивать веса α_i для переменных φ_i=1 и закрепить эти единичные сигналы, распространив их до входов персептрона по всем его слоям. Таким образом, для рассматриваемого типа ошибки *первое правило Хебба* можно сформулировать следующим образом: если на выходе персептрона получен 0, а

правильный ответ равен 1, то необходимо увеличить веса связей между одновременно активными нейронами от выхода до входов.

Второй тип ошибки – на выходе персептрона 1, а правильным ответом является 0. Для обучения персептрона правильному решению на примере следует уменьшить сумму в правой части формулы Ψ. Для этого необходимо уменьшить веса связей α_i при тех переменных φ_i, которые равны 1. Необходимо также провести эту процедуру для всех активных нейронов предыдущих слоев. В результате получаем *второе правило Хебба*: если на выходе персептрона получена 1, а правильный ответ равен 0, то необходимо уменьшить веса связей между одновременно активными нейронами от выхода до входов. На примере правил Хебба, которые, кстати, являются далеко не единственными правилами обучения ИНС, видно, что нейронная сеть является в некотором смысле «числовой записью» ранее рассмотренных продукционных правил «Если (условия), то (действия)». С логической точки зрения возможно однозначное преобразование ИНС – продукционные правила.

Процедура обучения сводится к последовательному перебору всех примеров из обучающей выборки с применением правил Хебба для обучения ошибочно решенных примеров. Если после очередного цикла предъявления всех примеров окажется, что все они решены правильно, то процедура обучения завершается. Некоторое время открытыми оставались два теоретических вопроса: о сходимости процедуры обучения и о том, насколько надо увеличивать или уменьшать веса связей при применении правил Хебба. Удалось доказать, что

- если существует вектор весов α, при котором персептрон правильно решает все примеры обучающей выборки, то при обучении персептрона по правилам Хебба решение будет найдено за конечное число шагов (теорема о сходимости);
- если не существует вектора весов α, при котором персептрон правильно решает все примеры обучающей выборки, то при обучении персептрона по правилам Хебба через конечное число шагов вектор весов начнет повторяться (теорема о «зацикливании»).

В нейротехнологиях обучается не отдельный нейрон, а вся сеть в целом. Алгоритмы обучения ИНС напоминают процессы

программирования и оптимизации. Базовыми подходами к обучению ИНС являются:

- модификация весов синаптических связей без изменения параметров нейронов (*коннекционизм*);
- модификация параметров нейронов без изменения синаптических связей (*гетерогенные ИНС*).

Иногда применяется комплексный подход, а также существуют разновидности алгоритмов обучения, которые не требуют обучения на примерах, а обучаются прямо в процессе решения реальной задачи (самоорганизующиеся карты Кохонена). Классическим считается алгоритм обучения методом обратного распространения ошибок (*error back propagation*) для архитектуры многослойного персептрона с аналоговыми синапсами и сигмоидальной активационной функцией нейронов. Многочисленные публикации о промышленных применениях многослойных сетей с этим алгоритмом обучения подтвердили его принципиальную работоспособность на практике.

Основная идея алгоритма обратного распространения состоит в том, как получить оценку ошибки для нейронов скрытых слоев, поскольку ошибки нейронов выходного слоя возникают вследствие ошибок нейронов скрытых слоев. Чем сильнее синаптические связи между нейроном скрытого слоя и выходным нейроном, тем больше ошибка. Следовательно, оценку ошибки нейронов скрытых слоев можно получить как взвешенную сумму ошибок последующих слоев. При обучении информация распространяется от входных нейронов через скрытые слои к выходам, а оценки ошибки сети – в обратном направлении, что отражено в названии алгоритма.

Подробное описание этого алгоритма представлено в [2, 5, 12]. Здесь отметим, что начальные значения весов всех нейронов всех слоев обычно полагаются случайными числами. Сети предъявляется входной образ из обучающей выборки, в результате формируется выходной образ. Функционал квадратичной ошибки, подлежащий минимизации и отражающий зависимость ошибки от весовых значений нейронов, для данного входного образа имеет вид псевдопараболоида:

$$E = \frac{1}{2}\sum_{i=1}^{n}(\psi_s - \psi_p)^2,$$

где n – число входов сети, $\Psi_э$ – эталонное значение выходных сигналов сети, $\Psi_р$ – реальное значение выходов. Классический градиентный метод оптимизации указанного функционала состоит в итерационной коррекции весовых значений нейронов. Метод учитывает полезное свойство сигмоидальной функции $f(x) = \frac{1}{1+\exp(-x)}$: ее производная выражается только через само значение функции, т.е. $f'(x) = f(1-f)$. При этом, если в нейронной сети имеется несколько скрытых слоев, то процедура обратного распространения применяется последовательно для каждого из них, начиная со слоя, предшествующего выходному, и далее до слоя, следующего за входным. При этом все используемые формулы сохраняют свой вид с заменой элементов выходного слоя на элементы соответствующего скрытого слоя. Обучение завершается по достижении заданного значения функционала или максимально допустимого числа итераций. Кроме того, в итерационные формулы обычно вводится коэффициент скорости обучения η, с помощью которого можно управлять величиной коррекции весов и который обычно выбирается достаточно малым для сходимости алгоритма (невысокий темп сходимости является слабостью всех градиентных методов, т.к. локальное направление градиента отнюдь не обязательно совпадает с направлением к минимуму). Отметим также, что теоретические исследования показали: для представления произвольного функционального отображения, задаваемого обучающей выборкой, достаточно всего два слоя нейронов (входной и выходной). Однако на практике, в случае сложных функций, использование скрытых слоев может давать экономию полного числа нейронов в ИНС.

Перед тем, как привести пример программирования ИНС, приведем необходимые сведения из программного приложения MatLab – пакета Neural Networks Toolbox, учитывая что определенные навыки работы с MatLab были получены в ходе выполнения предыдущей лабораторной работы: «Программирование нечетких моделей в среде MatLab».

Поскольку интерактивная система MatLab представляет собой интерпретатор, то освоение инструментария нейронных сетей заключается в основном в изучении их функций и их параметров

нейронов. Например, понять возможности нейрона как классификатора простых линейно сепарабельных задач можно путем проведения экспериментов с моделью одного нейрона. Для того чтобы создать в MatLab нейрон, используют функцию newp, имеющую следующий синтаксис:

```
net = newp (PR, S, TF, LF),
```

где PR – матрица минимальных и максимальных значений входных элементов; S – количество нейронов (при создании одного нейрона S=1); TF – функция активации (transfer function); LF – имя функции обучения нейрона.

Если параметры функции newp не заданы, их значения можно ввести через соответствующие диалоговые окна. Построенный нейрон характеризуется функциями весов (weight function), входов сети (net input function) и определенной функцией активации. Веса инициализируется нулями. Для обучения используются следующие функции.

Функция learnp настраивает веса нейрона. Синтаксис этой функции имеет следующий вид:

```
dW, LS] = learnp (W, P, Z, N, A, T, E, gW, gA,
                  D, LP, LS),
```

где вектор W – вектор весов; P – вектор входов; Z – вектор взвешенных входов; N – вектор сети; A – вектор выхода; T - вектор желаемых выходов; E – вектор ошибок; gW – вектор изменения весов; gA – изменения выходов. Функция DW возвращает значения изменения матрицы весов в процессе обучения; LS – новый уровень обученности нейрона.

Функция learnp может быть использована с параметрами по умолчанию:

```
dW = learnp ([], P, [], [], [], [], E, [], [],
                 [], [], []),
```

где использование пустого списка [] означает параметр по умолчанию.

Функция adapt адаптирует нейрон к условиям задачи:

```
[net, Y, E, Pf, Af] = adapt (net, P, T, Pi,
                             Ai),
```

где net – идентификатор нейронной сети, P – входы сети, T – желаемый выход, P_i - исходные условия задержки входов сети, A_i – исходные условия задержки для слоя. Функция возвращает параметры адаптированной сети net.adaptParam: net – измененная сеть, Y – выход сети, E – ошибки сети, Pf – условия задержки входов, Af – условия задержки слоя. Параметры Pi и Pf являются не обязательными, они необходимы только для сетей, имеющих задержки на входах и между слоями.

Функция train также обучает нейронную сеть и использует следующий синтаксис:

```
[net, tr] = train (net, P, T, Pi, Ai),
```

где net - сеть, P – входы сети, T – эталонный выход, P_i – исходные условия задержки входа, A_i – исходные условия задержки слоя. Функция моделирует нейронную сеть:

```
[Y, Pf, Af] = sim (net, P, Pi, Ai),
```

где net – идентификатор моделируемой сети; P – входы сети; P_i – исходные условия задержки входов сети; A_i – исходные условия задержки слоя. Данная функция возвращает: Y – выходы сети; Pf – окончательные условия задержки входов; Af – окончательные условия задержки слоя.

Функции активации и их назначение в Neural Networks Toolbox следующие:

hardlim (N)– возвращает 1, если N положительное и 0 в противном случае;

tansig (N) – вычисляет гиперболический тангенс от входа;

purelin – вычисляет выход слоя от входов ИНС.

Структура данных сети net – это описание обученной ИНС. Обучение осуществляется в соответствии со следующими параметрами, значения которых либо устанавливаются пользователем, либо по умолчанию:

net.trainParam.epochs 100 – максимальное количество итераций (эпох) обучения;

net.trainParam.goal 0 – целевое значение ошибки;

net.trainParam.max_fail 5 – максимальное значение ошибки;

net.trainParam.mem_reduc 1 – фактор оптимизации процесса обучения (памяти или времени, необходимого для ускорения процесса обучения;

net.trainParam.min_grad 1e-10 – минимальное значение градиента;

net.trainParam.show 25 – количество эпох между показами;

net.trainParam.time inf – максимальное время обучения (с);

TR – структура данных, содержащая значения об обученности нейронной сети в текущую эпоху;

TR.epoch – номер эпохи;

TR.perf – уровень обученности (*training performance*);

TR.vperf – достигнутая степень качества (*validation performance*) после окончания процесса обучения;

TR.tperf – результативность обработки теста (*test performance*);

TR.mu – значение адаптивности.

Структура данных описания адаптированной нейронной сети net.adaptfcn включает в себя поля net, adapt и param:

net – идентификатор адаптированной нейронной сети;

Y – выходы нейронной сети;

E – ошибки нейронной сети;

Pf – окончательные входные значения задержек;

Af – окончательные выходные задержки;

TR – результат обучения (эпохи и квадратичная ошибка).

4.2. Примеры построения нейронных сетей

Проведем в интерактивной среде MatLab эксперименты, используя рассмотренные функции.

Пример 1. Создание нейрона, выполняющего функцию логического И. Создадим нейрон с одним двухэлементным входом (интервалы первого и второго элементов [0;1]). Определим два первых параметра функции newp, а в качестве значений третьего и четвертого параметра (типа функции активации и имени процедуры обучения) воспользуемся значениями по умолчанию:

\\ создание нейрона с одним двухэлементным входом (интервал первого элемента [0;1] и интервал второго элемента [0;1]

```
net = newp ([0 1; 0 1], 1);
```

Для того чтобы исследовать поведение нейрона, необходимо моделировать его работу с помощью функции `sim`. Для определения последовательности значений входа создадим последовательность `P1`:

\\ создание последовательности значений входа

```
P1 = {[0; 0] [0; 1] [1; 0] [1; 1]};
```

\\ имитация работы нейрона `net` на последовательности входов `P1` желаемых выходов – `T`, которая позволяет нам провести адаптацию нейрона (обучить его) через 20 проходов

```
Y = sim (net, P1);
```

\\ создание последовательности выходов

```
T1 = {0, 0, 0, 1};
```

\\ установка количества проходов (циклов) адаптации

```
net.adaptParam.passes = 20;
```

\\ адаптация нейрона на `net` для обучающей выборки <`P1; T`>

```
net = adapt (net, P1, T1);
```

\\ симуляция работы нейрона `net` на последовательности входов `P1`

```
Y = sim (net, P1);
```

В результате получим нейрон, выполняющий функцию логического И.

Пример 2. Обучение нейрона классификации векторов на две категории. Начнем с классификации векторов на основе двухвходового нейрона. Будем использовать функцию `newp` для создания нейрона, `sim` для имитации его работы, `adapt` для адаптации (обучения) нейронной сети. Обучим двухвходовой нейрон классифицировать входные векторы на две категории:

\\ определим четырех двухэлементных входа

```
P = [ -0,5 -0,5 +0,4 -0,2; -0,5 +0,5 -0,4 +1,0];
```

\\ зададим желаемые выходы нейрона как реакции на входы `P`

```
T = [1 1 0 0];
```

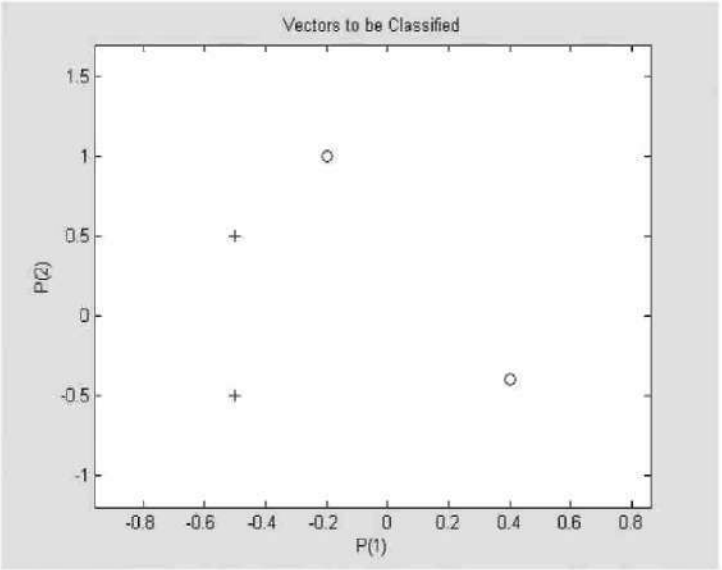


Рис.4.3. Исходные векторы, предназначенные для классификации нейроном

\\ изобразим входные векторы (точки) на плоскости (рис. 4.3), где каждый из четырех входных секторов на плоскости `P` определяется двумя координатами, представленными как двухэлементные столбцы в матрице `P`

```
plotpv (P, T);
```

\\ создадим один нейрон с двумя входами, значения которых лежат в интервале [-1, 1] (нейрон по умолчанию имеет функцию активации `hardlim` и такой нейрон разделяет входные векторы прямой линией

```
net = newp ([ -1 1; -1 1], 1);
```

\\ определим координаты линии классификации: веса (IW) и порог срабатывания нейрона (b), т.е. получим управляющую структуру `linehandle` для изображения разделяющей линии в координатах весов `(IW)` и порога срабатывания нейрона `(b)`

```
linehandle = plotpc (net.IW {1}, net.b {1});
plotpc (net.IW {1}, net.b {1});
```

Далее, если исходным весам задать нулевые значения, то любые входы дадут одинаковые выходы, и линия классификации не будет видна на плоскости. Проведем обучение нейрона:

```
\\ очистка координатных осей
cla;
\\ изображение входных векторов двух категорий (категория задается элементами вектора T)
plotpv (P, T);
\\ получение управляющей структуры linehandle
linehandle = plotpc (net.IW {1}, net.b {1});
\\ присвоение начального значения ошибки
E = 1;
\\ инициализация нейрона
net = init (net);
\\ получение управляющей структуры linehandle
linehandle = plotpc (net.IW {1}, net.b {1});
\\ повторение цикла до тех пор, пока ошибка не станет равной 0
while (mse (E))',
\\ адаптация нейрона net на обучающей выборке <P, T>, функция возвращает адаптированный нейрон net, выход Y и ошибку E
[net, Y, E] = adapt (net, P, T);
\\ изображение разделяющей прямой нейрона после адаптации
linehandle = plotpc (net.IW {1}, net.b {1}, linehandle);
\\ очистка окна графиков
drawnow;
\\ конец цикла while
end;
```

Функция adapt возвращает новый объект – сеть, которая выполняет классификацию, выход сети и ошибку (рис. 4.4).

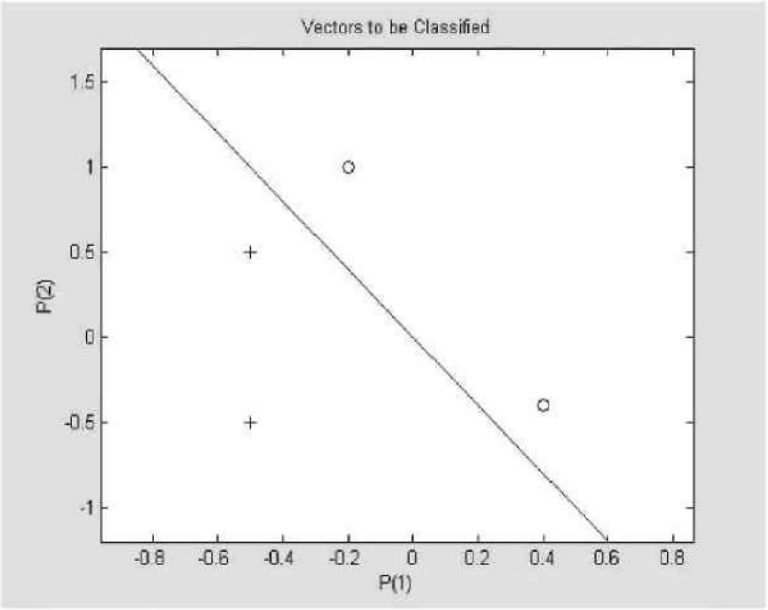


Рис. 4.4. Прямая, разделяющая исходные векторы на классы

Проведем классификацию нового вектора с помощью обученного нейрона на основе функции `sim`:

```
\\ определение нового вектора P
P = [0,6; 1,1];
\\ имитация работы нейрона net, получение отклика нейрона a
a = sim (net, p);
\\ изображение входа p, отнесенного нейроном к категории a
plotpv (p, a).
```

Наконец, полученный в ходе обучения нейрон можно использовать для классификации любого вектора:

```
\\ включить режим добавления графиков в графическом окне
hold on;
\\ получить изображение входных точек в соответствии с категориями T
plotpv (P, T);
\\ получить изображение разделяющей поверхности
```

```
plotpc (net.IW {1}, net.b {1});
```
\\ отключить режим добавления графиков
```
hold off;
```

В результате нейрон классифицирует новую точку, как принадлежащую категории «0» (она на рис. 4.4 представлена кружком), а не категории «1» (представлена символом «+»).

Пример 3. Процесс обучения нейрона. Для выполнения примера будем использовать функции: `newlin` – для создания нейрона, `train` – для обучения, `sim` – для имитации поведения нейрона:

\\ определим два входа нейрона вектором P
```
P = [1.0 -1.4];
```
\\ определим эталонные (желаемые) выходы нейрона
```
T = [0.5 1.0];
```

Для решения задачи используем линейный нейрон с одним входом.

\\ установим интервала изменения весов w и порога b
```
w = -1 : 0,1 : 1;
b = -1 : 0,1 : 1;
```

\\ вычислим и изобразим поверхность функции ошибки нейрона в координатах весов и пороговых значений (рис. 4.5), причем наилучший вес и значение порога являются самыми нижними точками на поверхности, в которых величина ошибки (расхождение эталонного и реального выходных значений) минимальна
```
ES = errsurf (P, T, w, b, 'purelin');
plotes (w, b, ES);
```
\\ переопределим уровень обученности для слоя с bias с вектором P
```
maxlr = 0.20 * maxlinlr (P, 'bias');
net = newlin ([-2 2], 1, [0], maxlr);
```

Здесь функция `maxlinlr` находит значение, соответствующее самому высокому уровню обученности нейросети. Возьмем 20% от вычисленного уровня. Функция `newlin` содержит четыре параметра: матрицу интервалов входов размерностью 2×2, количество элементов в выходном векторе, вектор задержек входов, а также уровень обучения.

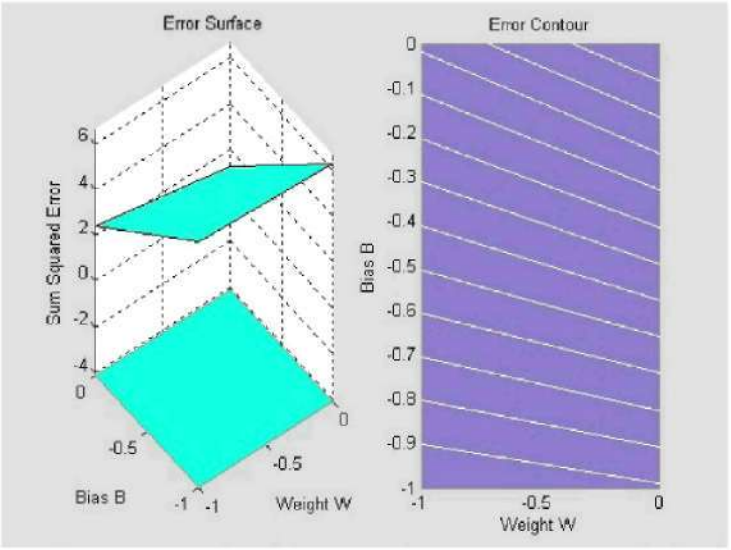


Рис.4.5. Поверхность функции ошибки

\\ установим целевое значение ошибки, сформировав поверхность ошибки путем создания двух активных координатных осей в графическом окне

```
ES = errsurf (P, T, w, b, 'purelin' );
plotes (w, b, ES);
subplot (1, 2, 2);
```

\\ произвольно изменим веса нейрона в установленных интервалах, установим эпоху обучения нейрона и сформируем историю обучения `tr`

```
net.IW { 1, 1} = 0;
net.b { 1} = 0,1;
[ net, tr] = train (net, P, T);
```

В частности, функция `train` выводит на экран историю обучения сети (`tr`). Ее график (рис. 4.6) показывает уменьшение ошибки в ходе обучения, что свидетельствует о приближении получаемых значений к эталонным. Используем функцию `sim`, чтобы

протестировать нейрон на входных значениях -1 и 1, где выход должен быть равен 1. Результат очень близок к 1 и равен 0,9167:

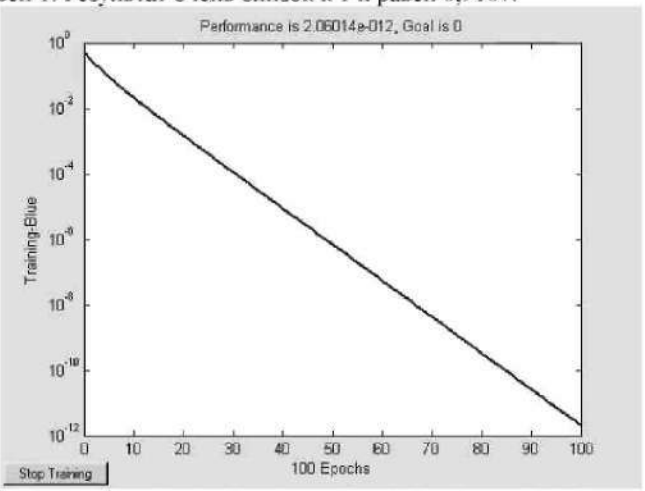


Рис.4.6. Изменение среднеквадратичной ошибки в ходе обучения

```
plotperf (tr, net.trainParam.goal);
p = -1,1;
a = sim (net, p)
```

4.3. Порядок выполнения работы

Сущность задания к лабораторной работе состоит в том, чтобы с помощью инструментария нейронных сетей в среде MatLab реализовать простейшую нейронную сеть. Для этого необходимо построить нейронную сеть, реализующую поставленные в варианте задачи, произвести ее настройку, обучение, проверку работоспособности на тестовых примерах, а также подготовить и защитить отчет о самостоятельно проделанной работе.

Этапы выполнения лабораторной работы:

- изучить представленный теоретический материал, рассмотреть примеры;
- четко представить себе, что необходимо сделать согласно выбранному варианту задания;

- реализовать с помощью соответствующего инструментария в среде MatLab нейронную сеть, реализующую поставленные задачи;
- обучить построенную нейронную сеть на наборе тестовых эталонных значений;
- построить график изменения ошибки в процессе обучения нейронной сети, сделать выводы;
- проверить работоспособность построенной нейронной сети на различных примерах и сравнить полученные результаты с ожидаемыми;
- представить отчет о работе, включающий авторские выводы и оценку достижения поставленных целей.

4.4. Содержание отчета

1. Цель работы.
2. Постановка задачи согласно выбранному варианту.
3. Описание самостоятельно проделанной работы (анализ этапов проектирования нейросети; задание обучающей выборки; график изменения ошибки в процессе обучения нейросети; результаты работы сети на примерах, их сравнительная характеристика).
4. Выводы.

Приложение: необходимые для защиты лабораторной работы "скриншоты" экрана.

4.5. Варианты заданий

Представленные ниже варианты включают различные задачи распознавания, а также аппроксимации. Каждый вариант предусматривает самостоятельную разработку модели нейросети, ее обучение и практическую реализацию в среде MatLab. Вариант задания выбирается студентом по согласованию с преподавателем.

Конкретные варианты заданий представлены ниже.

1. Распознавание чисел

Необходимо реализовать модель искусственной нейронной сети, распознающей десятичные числа в битовом представлении. Закрашенные ячейки битового поля соответствуют 1 на входе соответствующего нейрона, пустые ячейки – 0. Размер битового поля для представления десятичных чисел выбирается самостоятельно, при

этом необходимо, чтобы представления различных десятичных чисел не были идентичны.

Пример битового представления числа «5» на битовом поле размером 4×8 представлен на рис. 4.7.

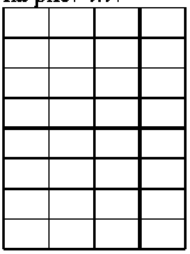


Рис. 4.7. Пример битового представления числа «5»

2. Аппроксимация функции

Необходимо реализовать модель искусственной нейронной сети, аппроксимирующую функцию $y = 2x^2$ на интервале $x \in [-2; 2]$. При обучении нейронной сети в качестве обучающей выборки необходимо самостоятельно выбрать некоторое количество точек из интервала *[-2; 2]*, для которых необходимо рассчитать ожидаемый в соответствии с заданной функцией выход. Остальные значения функции на данном интервале необходимо получить при помощи обученной нейронной сети, шаг изменения аргумента x взять равным *0,01*. В отчете необходимо представить графическое представление полученной аппроксимированной функции и ее эталонного представления, построенного средствами MatLab.

3. Аппроксимация функции

Необходимо реализовать модель искусственной нейронной сети, аппроксимирующую функцию $y = 3x^3-1$ на интервале $x \in [-1; 1]$. При обучении нейронной сети в качестве обучающей выборки необходимо самостоятельно выбрать некоторое количество точек из этого интервала, для которых необходимо рассчитать ожидаемый в соответствии с заданной функцией выход. Остальные значения функции на данном интервале необходимо получить при помощи обученной нейронной сети, шаг изменения аргумента x взять равным *0,005*. В отчете необходимо представить графическое представление

полученной аппроксимированной функции и ее эталонного представления, построенного средствами MatLab.

4. Классификация состояния больных с ишемией

Необходимо реализовать модель искусственной нейронной сети, выполняющей задачу классификации состояния больных с ишемической болезнью сердца. Ишемическая болезнь сердца - сердечно-сосудистое заболевание, характеризующееся нарушениями функций сердца в связи с недостаточностью кровоснабжения. По набору номинальных (например, уровень боли при стенокардии) и непрерывных показателей (например, артериальное давление и возраст), классифицируется состояние пациентов с ишемической болезнью сердца. Показатели, по которым необходимо провести оценку состояния пациента: содержание холестерина в крови, вес, артериальное давление, подвижность, курение, эмоциональный стресс, возраст, пол, наличие стенокардии, аритмии, сердечной недостаточности (список может быть дополнен). Значения этих показателей выбираются самостоятельно с учетом специфики номинальных и непрерывных показателей. Необходимо представить несколько различных диагнозов. Обучение нейронной сети желательно провести для трех случаях: человек абсолютно здоров (все показатели с положительной оценкой), человек болен ишемической болезнью (все показатели с отрицательной оценкой), «средний» случай. При работе нейронной сети желательно в качестве примеров сымитировать больничные карты нескольких пациентов с широким разбросом показателей.

5. Распознавание букв номерных знаков

Необходимо реализовать модель искусственной нейронной сети, распознающей буквы на автомобильных номерных знаках Российской Федерации в битовом представлении (на этих знаках используются буквы русского алфавита, совпадающие по написанию с соответствующими буквами английского алфавита). Закрашенные ячейки битового поля соответствуют 1 на входе соответствующего нейрона, пустые ячейки – 0. Размер битового поля для представления букв выбирается самостоятельно, при этом необходимо, чтобы представления различных букв не были идентичны.

Пример битового представления буквы «А» на битовом поле размером 4×8 представлен на рис. 4.8.

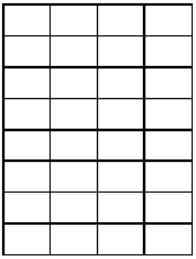


Рис. 4.8. Пример битового представления буквы «А»

Контрольные вопросы

1. Если существует множество значений весов, которые обеспечивают конкретное различение образов, то в конечном итоге алгоритм обучения персептрона приводит либо к этому множеству, либо к эквивалентному ему множеству, такому, что данное различение образов будет достигнуто: а) верно, б) неверно.
2. Классами нейронных сетей прямого распространения являются: а) многослойный персептрон, б) однослойная сеть Хопфилда, в) однослойный персептрон, г) сеть Кохонена, д) сеть радиальных базисных функций, е) соревновательные сети.
3. Базовый нелинейный элемент нейронной сети – это: а) синапс, б) нейрон, в) сумматор, г) персептрон.
4. Составьте из предлагаемых слов и словосочетаний современную парадигму нейрокомпьютинга: а) адаптируются, б) алгоритмы, в) интеллектуальные задачи, г) класс операций с образами, д) порождаемые данными, е) процесс обучения, ж) специализируются.
5. Обучение ИНС – это: а) поиск экстремума функции, отображающей вход-выходное взаимодействие, б) ответы на вопросы и решение задач, в) двусторонний процесс передачи понятной информации.
6. Укажите соответствие между названиями двух подходов к обучению ИНС: 1) гетерогенность, 2) коннекционизм и сущностью этих подходов: а) изменяются веса синаптических связей без изменения параметров нейронов, б) модифицируются параметры нейронов, связи не меняются.

7. Создание нейрона с помощью пакета Neural Networks Toolbox осуществляется по команде: а) neural, б) newp, в) newlog.
8. Допустимыми функциями активации в пакете Neural Networks Toolbox являются: а) logsig, б) softmax, в) trainoss, г) newlin, д) satlin, е) gensim.
9. Обучение нейронной сети осуществляется при помощи команды: а) trainoss, б) train, в) sim.
10. Имитация нейронной сети осуществляется при помощи команды: а) trainoss, б) train, в) sim.
11. Ваша самооценка самостоятельно выполненной работы по критерию 1: 2_1_0 по критерию 2: 2_1_0 по критерию 3: 2_1_0.

Лабораторная работа №5

5. Проектирование интеллектуальных систем информационного поиска

Целью работы является ознакомление с основными моделями и технологиями информационного поиска в коллекциях текстовой информации, практическая реализация поиска на базе подготовленного индекса поисковой машины, сравнительный анализ поисковых моделей.

5.1. Краткое теоретическое описание

Исследования в области информационного поиска начались более сорока лет назад. За это время из узкоспециализированной тематики информационный поиск превратился в одну из ключевых областей теоретической информатики и искусственного интеллекта. Необходимость поиска и обработки нужной текстовой информации для миллионов пользователей Интернета, баз данных, систем электронного документооборота, разнообразных картотек и справочников обусловливает актуальность дальнейшего совершенствования информационно-поисковых систем, исследования их архитектурно-функциональных особенностей, а также повышения качества их функционирования. По мере увеличения объема информационных ресурсов все острее стоит вопрос об эффективных способах поиска необходимых пользователю сведений. Современная поисковая система должна осуществлять поиск документов, имеющих прямое отношение к указанному поисковому запросу, а не просто содержащих информационный объект. Это требует научной и прикладной проработки вопросов, связанных с тем, что элементы знаний, в отличие от данных, не могут произвольно переноситься из одной базы в другую без потери свойств, т.к. они контекстно связаны. Именно здесь возникает основная теоретическая проблема, связанная с пониманием текстов на естественном языке как сложного многоуровневого процесса обработки информационных объектов.

Любой информационный объект включает три аспекта: структуру, контент и контекст. Структура объекта характеризуется

набором атрибутов и информационных связей. Контент описывает информационное содержание объекта с помощью понятий и отношений между ними. Контекст в отличие от контента рассматривает информационный объект как единое целое и не зависит (явно) от его содержания. Раскрытие контекста в ходе автоматизированного полнотекстового семантического поиска означает учет смыслового содержания слов, словосочетаний запроса пользователя и предложений текстов проиндексированных информационных ресурсов. Недостаточно одних формальных критериев классификации по количеству слов в запросе, применяемых в известных методиках. Более существенными являются семантические признаки запросов. Разные запросы требуют различных условий релевантности, мерой которой в реальных поисковых системах является степень удовлетворенности пользователя качеством результатов поиска. Этот критерий остается малоисследованным и не поддается точному формальному определению, в отличие от критериев, используемых в экспериментах по информационному поиску. Развитие систем семантического поиска и анализа больших объемов текстов в последние годы идет быстрыми темпами, они постоянно пополняются и становятся незаменимы, когда человек не точно представляет цель поиска. Однако пока такие системы не дают достаточно адекватных результатов.

Библиография работ в области технологий информационного поиска весьма обширна. Она подробно представлена в [3, 7] и других публикациях. Ниже приводятся лишь основные понятия из теории интеллектуальных систем информационного поиска, необходимые для понимания существа данной лабораторной работы.

Информационный поиск – это совокупность операций и процедур, направленных на отбор данных, хранящихся в информационной системе. Информационно-поисковые системы (ИПС) бывают трех типов: *документальные*, *фактографические* и *гипертекстовые*.

Документальные ИПС хранят и выдают сведения о документах. Признаки документа называют *поисковым образом*, а признаки запроса – *поисковым предписанием*. Перевод документа и запроса на него в форму представления, принятую в ИПС, называется *индексированием*. Чтобы сопоставить поисковый образ и его предписание используют критерий смыслового соответствия (*релевантность*). Запросы обычно состоят из ключевых слов или

дескрипторов. В целях ускорения поиска для каждого дескриптора указывается список рефератов документов, в которых он встречается. Такая информационная структура ИПС называется *индексом*. Развитием поиска по дескрипторам является полнотекстовый поиск, реализуемый в поисковых машинах Интернет.

В фактографических ИПС хранятся не документы, а собственно факты об объектах предметной области. Они реализуются обычно на реляционных БД. С точки зрения релевантности, поиск в них, в отличие от документального, является полным и точным.

Гипертекстовые ИПС, кроме документов, включают семантическую структуру. Это означает, что на множестве документов в хранилище задана оценка смысловой близости каждой пары документов. Если она равна 0, то документы эквивалентны по смыслу. Если оценка не определена, то документы считаются семантически несопоставимыми. Кроме того, могут вводиться оценки отдельных свойств документов. Эти оценки обычно выражаются действительными числами из некоторого интервала (чем больше число, тем важнее для пользователя документ). Тогда любой поисковый запрос можно рассматривать как некоторый виртуальный документ. В зависимости от цели поиска можно ожидать получения различных результатов. Например, среди множества документов не найдено документов, релевантных запросу; найден единственный документ или несколько документов; найденные документы имеют допустимые расхождения со смыслом запроса; найденные документы имеют наибольшую оценку важности некоторого свойства (векторный поиск). Указанные результаты зависят от способов задания смысловой близости документов в ИПС.

Эффективность информационного поиска принято оценивать по коэффициентам информационной *полноты* и информационного *шума* на интервале [0, 1]. Идеальный вариант – полнота максимальна, а шум нулевой. В литературе вместо информационного шума часто используют обратный ему показатель – коэффициент *точности*. В теории информационного поиска предложен обобщенный комплексный показатель эффективности информационного поиска (*мера Ван Ризбергена*), позволяющий учитывать предпочтение, отдаваемое пользователем точности или полноте.

Обозначим через $|C|$ общее число документов в некоторой коллекции C; через r_q – число полученных документов, релевантных

запросу q; через n_q – общее число документов, полученных из коллекции C в ответ на запрос q, через R_q – число документов коллекции C, релевантных запросу q. Тогда точность поиска в C при запросе q равна r_q / n_q, а полнота поиска при запросе q равна r_q / R_q.

Сравнение документальных, фактографических и гипертекстовых ИПС по таким характеристикам, как полнота и шум, показывает, что для документальных полнота поиска равна 0,5, а шум – 1. Для фактографических ИПС полнота равна 1, а шум – 0, для гипертекстовых ИПС полнота находится на интервале [0,9; 1,0], а шум – на интервале [0,1; 0,2].

Объектом исследования в данной лабораторной работе является анализ результатов поиска в текстовых коллекциях документов. При анализе статических коллекций возникают три основные проблемы: качество поиска, скорость выдачи результата на запрос пользователя, а также объём вычислительных ресурсов, необходимых для работы поисковой системы. Для решения этих проблем применяются различные подходы и методы, при этом используются следующие термины:

- *терм* обозначает элементы поискового образа документа (слова, основы слов, фразы и т.п.);

- *поисковый индекс* – это образ коллекции документов внутри поисковой системы для обслуживания пользовательских запросов без реального доступа к документам коллекции;

- *релевантность* означает степень близости документа поисковому запросу пользователя;

- *ранжирование* – это процесс оценки релевантности страниц запросу пользователя.

Поисковые машины или *автоматические индексы*, размещаемые на серверах свободного доступа, состоят из трех компонент: программы *сканирования* (*crawler*), *индексирования* и *поиска*.

Crawler или spider (паук) – это программа, которая на основе заданных алгоритмов автоматически сканирует различные Web-сайты и обеспечивает полную или частичную индексацию ресурсов, URL, ключевых слов, ссылок и документов. В последующем на основе созданных индексных баз данных поисковые машины предоставляют пользователю доступ к распределенной на узлах сети информации. Это реализуется через выполнение *поисковых запросов* в рамках соответствующего интерфейса. Программа crawler может также

переходить по расположенным на сайте ссылкам на другие, близкие по содержанию, страницы. При этом она периодически возвращается к исходным сайтам, чтобы проверить, не произошли ли какие-нибудь изменения и снова считывает страницы. Когда пользователь делает поисковой машине запрос, ее программное обеспечение проходит по созданному индексу в поиске Web-страниц с заданными ключевыми словами и классифицирует эти страницы по степени близости к предмету поиска. Все, что находит и считывает робот, попадает в индексы поисковой машины.

Индексы представляют собой гигантское хранилище информации, где хранятся копии текстовой составляющей всех посещенных и проиндексированных Web- страниц. Для профессиональной работы с автоматическим индексом необходимо учитывать два определяющих аспекта его работы:

- индексирование программой-роботом содержимого Web-страниц (адрес очередного документа робот узнает либо от автора ресурса, который представил его в систему, либо из гиперссылки, найденной им на уже пройденной странице);
- обработка запросов пользователей по ключевым словам на основе синтаксиса поискового языка системы.

Обе эти фазы работы поисковой машины тесно связаны: чем больше информации о ресурсе извлечено при сканировании, тем потенциально шире возможности поиска. Тот факт, что каждая система в обоих случаях имеет свою специфику, может быть использован для тонкой настройки на решение поисковой задачи. Поскольку индексы сканируют единое информационное поле (WWW), то в них может находиться информация об одних и тех же ресурсах. Однако время, затраченное на получение результата при поиске, может существенно зависеть от выбранной поисковой машины. Кроме того, использование всего одной поисковой системы не дает никаких гарантий по полноте охваченных ресурсов (реальная полнота охвата Web-ресурсов отдельными ИПС не превышает 30%).

В ходе индексирования выполняется «компрессия» текста и перевод его содержания с естественного языка на язык ИПС. Чтобы включить документ в коллекцию, необходимо его проиндексировать путем разбора текста документа и записи информации в базу данных. Структура данных для представления индекса поисковой системы

зависит от выбранной модели поиска. Основными моделями полнотекстового поиска, основанного на анализе статистических данных, являются булева модель, векторная модель, вероятностная модель (относится к многоитерационным видам поиска), а также модель латентного семантического индексирования. Главная задача индексирования состоит в быстром получении информации о релевантности документа.

Наконец, получив запрос пользователя, программное обеспечение ИПС использует индекс для *поиска* документов, наиболее близких запросу. Найденные документы сортируются по релевантности, и список их адресов, с некоторой дополнительной информацией (традиционно это заголовок, краткое описание документа, цитата из документа с ключевыми словами из запроса и т.п.), возвращается пользователю. Запрос пользователя к поисковой системе обычно формируется в виде набора ключевых слов. Система анализирует ключевые слова и приводит их к некоторому нормализованному виду (термы поисковой системы). Нормализованным видом может быть основная форма слова (например, именительный падеж и единственное число для существительного), либо основа слова, полученная отсечением окончания слова и некоторых суффиксов (именно этот вид используется в данной работе). Информация о виде нормализованных термов извлекается из поискового индекса.

Бывают разные типы поиска: нахождение документов, содержащих слова из пользовательского запроса (используется в данной работе), поиск по цитате, поиск «похожих» документов и т.п. Для разных типов поиска удобно использовать разное представление информации в индексе поисковой системы.

В связи с тем, что коллекции документов могут иметь значительный объём и релевантных документов для ручного перебора может оказаться слишком много, поставим вопрос о критериях оценки уровня релевантности документов в процессе их ранжирования и сортировки по результатам поиска.

Современные поисковые системы значительно расширили критерии, используемые при ранжировании найденных документов. Согласно некоторым из них учитывается *положение ключевых слов* в документе. Вес ключевого слова в документе может быть увеличен, если оно встречается в заглавии гипертекстового документа, или входит в метатеги списка ключевых слов документа, или находится в

заголовках документа, или выделено шрифтом. Согласно другому критерию необходимо оценить *авторитетность* документа (Google использует понятие «PageRank» - категория или класс страницы, Yandex – понятие индекса цитирования, Webalta – показатель уровня доверия и т.д.). Иными словами, результаты работы поисковой системы зависят от количества ссылающихся на сайт ресурсов сети, от цитируемости ссылок, времени регистрации доменного имени, репутации компаний, на чьих серверах физически размещёна страница и др. Чем выше показатель страницы, тем выше его место в результатах поиска. Алгоритмы подсчёта таких показателей довольно запутаны и компании не очень стремятся их разъяснять.

Развитием данного критерия является оценка популярности страницы в поисковом рейтинге путем ведения статистики её посещаемости (например, Rambler Top 100 предлагает установить на страницы сайта счётчик посетителей, в результате чего при запросе ИПС ставит на первые позиции те страницы, которые лидируют в рейтинге). Наконец, попытку использования «социальной сети» в поисковой системе предпринял сайт Eurekster [http://www.eurekster.com]. Если в Google самыми важными считаются те страницы, к которым ведёт больше всего ссылок, то в Eurekster первыми выдаются страницы, которые наиболее популярны среди посетителей.

Рассмотрим наиболее известные модели поиска: булеву, векторную, вероятностную и латентно-семантическую.

Булева модель поиска возникла одной из первых и является наиболее широко используемой моделью информационного поиска. Её распространение связано с простотой реализации, возможностями высокоскоростного поиска в электронных коллекциях большого объёма, а также совместимостью с другими моделями поиска.

Пусть коллекция C состоит из документов $d_1,...d_n$, а документ d_i содержит множество различных термов $T(d_i)$. Множество $T(d_i)$ будем называть словарём документа d_i. Объединение множеств из n различных термов, встречающихся в документах коллекции C, будем называть множеством *словаря коллекции* $T=U_{i=1,...n}T(d_i)$.

При использовании булевой модели запрос пользователя представляет собой некоторое логическое выражение, в котором ключевые слова запроса связаны логическими операторами *AND*, *OR* и *ANDNOT*. Если пользователь составляет свой запрос, не используя

логических операторов путем простого перечисления ключевых слов, то, по умолчанию, считается, что все ключевые слова соединены логической операцией *AND* (в некоторых системах *OR*). В результат включаются только те документы, которые содержат все ключевые слова запроса (или хотя бы одно из ключевых слов при использовании по умолчанию оператора *OR*). В табл. 5.1 представлен пример поискового индекса булевского поиска.

Таблица 5.1

терм\ документ	*1.htm*	*2.htm*	*3.htm*	*4.htm*	*5.htm*	*6.htm*	*7.htm*	...
арбуз	0	0	1	0	0	0	1	
август	0	0	0	0	0	0	0	
алжир	1	0	0	0	0	0	0	
агротехника	0	0	1	0	1	0	0	
адрес	1	0	0	0	0	0	0	
азия	0	1	0	0	1	0	0	
алый	0	1	0	0	0	0	0	
айва	0	0	1	1	0	1	0	
акация	0	1	0	0	0	0	1	
...								

В строке таблицы содержится информация о включенности терма в документы коллекции (0 – терм не встречается в документе; 1 – терм встречается в документе). Следует отметить, что при программной реализации в виду большой разреженности матрицы удобно хранить строки в виде массива пар (документ, значение). Иногда вместо «1» указывается количественная оценка значимости терма для документа (например, количество повторений терма внутри документа).

К недостаткам булевской модели относят отсутствие возможностей работы с семантикой текстов (синонимия и полисемия слов), определения релевантности документа ключевому слову.

Векторная модель поиска, предложенная Г. Сэлтоном в конце 80-х годов, каждому документу сопоставляет вектор в некотором базисе слов. Пользовательский запрос также преобразуется в вектор. Результатом поиска являются документы, углы между векторами которых и вектором запроса имеют минимальное значение. В табл. 5.2 представлен пример поискового индекса векторной модели поиска. В

каждой строке содержится вектор документа в пространстве термов. В ячейках содержится оценка веса терма для документа. При программной реализации в виду разреженности матрицы удобно хранить строки в виде массива пар (терм, оценка).

Таблица 5.2

док.\ терм	*арбуз*	*август*	*алжир*	*адрес*	*азия*	*алый*	*айва*	..
1.htm	0	0	0,0075	0	0	0	0,021	
2.htm	0	0	0	0	0	0	0	
3.htm	0,023	0	0	0	0	0	0	
4.htm	0	0	0,117	0	0,034	0	0	
5.htm	0,0761	0	0	0	0	0	0	
6.htm	0	0,144	0	0	0,023	0	0	
7.htm	0	0,093	0	0	0	0	0	
8.htm	0	0	0,0567	0,0112	0	0,019	0	
...								

Как и ранее, в булевской модели поиска предположим, что коллекция C состоит из документов $d \in C$, а каждый документ содержит множество термов $T(d)$. Введём следующие обозначения: $tf(d, t)$ – число вхождений терма t в документ d, $df(t)$ – число документов коллекции C, содержащих терм t; $w(d, t)$ – вес терма t в документе d. Тогда формула для вычисления веса терма в нормализованном векторе (вектор единичной длины) имеет следующий вид:

$$w(d,t)=\frac{tf(d,t)\log(\frac{|C|}{df(t)})}{\sqrt{\Sigma_{t'}(tf(d,t')\log(\frac{|C|}{df(t')}))^2}}.$$

Числитель формулы прямо пропорционален частоте, с которой терм встречается в документе, и связан логарифмической зависимостью с отношением общего числа документов в коллекции к числу документов, содержащих данный терм *t*. Знаменатель формулы равен корню из суммы квадратов оценок для всех термов по данному документу. Очевидно, что вес терма *w(d, t)=0*, если терм *t* не входит в документ *d*.

Сопоставим документу *d* в качестве его образа вектор единичной длины *W(d)*, *i*-компонента которого равна весу *w(d,t)* (размерность вектора равна |*T*|, причем *i* – порядковый номер терма *t* в словаре коллекции *C*). Вес терма *w(d,t)* тем выше, чем чаще встречается терм *t* в документе *d*, и чем реже он встречается в остальных документах коллекции. Таким образом, больший вес получают термы, которые «отличают» данный документ от остальных документов коллекции *C*.

В качестве меры близости документов d_1 и d_2 можно использовать скалярное произведение векторов $sim(d_1, d_2)=W(d_1)*W(d_2)$, равное косинусу угла между векторами-образами документов d_1 и d_2. Величина $sim(d_1, d_2)$ принадлежит диапазону *[0,1]*, а *sim(d, d)=1*. Чем больше величина $sim(d_1,d_2)$, тем более близкими считаются документы d_1 и d_2.

Аналогично, мерой близости запроса *q* документу *d* считается величина *sim(q, d)*. Скалярное произведения двух векторов *A(x1,y1,z1)* и *B(x2,y2,z2)* вычисляется по формуле *S = x1*x2 + y1*y2 + z1*z2*.

К недостаткам векторной модели относится вычислительная трудоёмкость и высокие требования к памяти. Причиной этого является размерность вектора, в общем случае равная размерности словаря термов, который обычно составляет сотни тысяч единиц. Кроме того, не существует эффективного способа нахождения всех документов *d*, для которых выполняется условие $sim(q, d)>\delta$, где δ есть некоторое пороговое значение. Полный перебор документов для оценки меры близости с пользовательским запросом *sim(q, d)* может оказаться неприемлемым для большой коллекции документов.

Основной задачей **вероятностной модели** является более обоснованная оценка веса терма в документе по сравнению с векторной моделью.

Будем использовать словарь коллекции, включающий все термы, встречающиеся хотя бы в одном документе коллекции. Сопоставим документу вектор $x=(x_1,\ldots x_n)$. i-я компонента которого равна 1, если i-й терм входит в данный документ, и компонента равна 0 в противном случае. Здесь, как и ранее, терм задается своим порядковым номером в словаре коллекции, а через n обозначим общее число термов в словаре коллекции. Обозначим через w_1 событие, состоящее в том, что рассматриваемый документ релевантен запросу q, а через w_2 событие, состоящее в том, что рассматриваемый документ не релевантен запросу q. Необходимо вычислить $P(w_i|x)$ – условную вероятность того, что для документа, представленного вектором x, наступает событие w_i. Зная эту вероятность, можно использовать следующее решающее правило: если $P(w_1|x)>P(w_2|x)$, то документ, представленный вектором x, релевантен запросу q.

Теорема Байеса позволяет оценить условную вероятность по формуле

$$P(w_i \mid x) = \frac{P(x \mid w_i)P(w_i)}{P(x)} .$$

При этом используется предположение о независимости вхождения в документ любой пары термов:

$$P(x \mid w_i) = P(x_1 \mid w_i) * \ldots * P(x_N \mid w_i) .$$

Примем следующее обозначение: $p_i=P(x_i=1|w_1)$, $q_i=P(x_i=1|w_2)$.

Тогда

$$P(x \mid w_1) = \prod_{i=1,\ldots,n} p_i^{x_i}(1-p_i)^{1-x_i}$$

и

$$P(x \mid w_2) = \prod_{i=1,\ldots,n} q_i^{x_i}(1-q_i)^{1-x_i} .$$

Решающее правило может быть представлено в следующей форме: документ, представленный вектором x, соответствует запросу q, если

$$\log\frac{P(x\mid w_1)P(w_1)}{P(x\mid w_2)P(w_2)}>0$$

Левую часть последнего неравенства можно представить как

$$\sum_{i=1,\dots,n} x_i \log\frac{p_i(1-q_i)}{q_i(1-p_i)}+\sum_{i=1,\dots,n}\log\frac{1-p_i}{1-q_i}+\log\frac{P(w_1)}{P(w_2)}$$

Коэффициент при x_i и можно рассматривать как вес терма с номером i в документе, представленном вектором x.

Оценим величины p_i и q_i, используя следующие обозначения: N – общее число документов в коллекции, R – число документов из коллекции, релевантных запросу q, n_i – число документов в рассматриваемой коллекции, индексированных термом с номером i (число документов, в которых встречается терм с номером i); r_i – число документов в рассматриваемой коллекции, релевантных запросу q и индексированных термом с номером i.

В этих обозначениях $p_i \approx r_i/R$ и $q_i \approx (n_i-r_i)/(N-R)$. Тогда в качестве веса терма с номером i в документе, представленном вектором x, можно взять величину

$$W(i)=\log\frac{r_i(N-R-n_i+r_i)}{(n_i-r_i)(R-r_i)}.$$

Для исключения деления на нуль последнее выражение обычно заменяют следующим:

$$W(i)=\log\frac{(r_i+\frac{1}{2})(N-R-n_i+r_i+\frac{1}{2})}{(n_i-r_i+\frac{1}{2})(R-r_i+\frac{1}{2})}.$$

Построенное выражение для веса терма с номером i предполагает наличие информации, которая обычно неизвестна на этапе поиска. Этой неизвестной информацией являются значения величин R и r_i. Здесь возможен следующий подход. В начале поиска используется только известная информация о размере коллекции N и о числе документов, индексированных термом с номером $i = 1,\dots, n$. Величины R и r_i полагаются равными нулю. В процессе поиска с помощью поисковых систем часто используется обратная связь с пользователем (relevance feedback). В этом случае в ответ на запрос пользователя система выдает первую порцию данных - результатов поиска, и

пользователь может пометить некоторые из полученных документов как релевантные запросу. Система использует полученную информацию для автоматического уточнения запроса, например, путем включения в него нескольких новых термов, выделенных из отмеченных документов. Этот процесс может повторяться несколько раз.

Итак, в начале поиска вес терма i вычисляется по формуле

$$W(i) = \log \frac{N - n_i + \frac{1}{2}}{n_i + \frac{1}{2}} \approx \log \frac{N}{n_i} .$$

Далее на каждой итерации relevance feedback известно множество документов, отмеченных пользователем как релевантные его запросу. Их общее число можно принять за некоторую оценку величины R, а число отмеченных документов, индексированных термом с номером i, – за оценку величины r_i. Эту информацию можно использовать для уточнения весов термов, используя ранее приведенную формулу. Таким образом, метод рассчитан на многоитерационный поиск.

Основная цель латентного семантического индексирования (Latent Semantic Indexing – LSI) состоит в выделении в коллекции документов некоторых неявных, скрытых факторов, в терминах которых можно выразить смысл отдельных термов и документов, представленных в коллекции. Предполагается, что это позволит справиться с такими проблемами, как учет синонимии и полисемии.

Согласно методу латентного семантического индексирования, среди термов коллекции отбрасываются наименее значимые. Оставшиеся термы приводятся к пространству ортогональных несводимых друг другу термов. Это на практике позволяет «сблизить» документы из одинаковых предметных областей, выявить документы с уникальным содержанием, учесть синонимию и омонимию термов. Достоинством метода является то, что искажения, возникающие в результате омонимии, при увеличении длины запроса не накапливаются, а взаимно корректируются.

В основе латентного семантического индексирования лежит известный из линейной алгебры метод сингулярного разложения матрицы.

Пусть A является матрицей размерности $m \times n$, где m – число термов в словаре коллекции, а n – число документов. Будем полагать, что $m \geq n$. Элемент a_{td} матрицы A является весом терма в документе. Здесь t является порядковым номером терма в словаре коллекции, а d – порядковым номером документа.

Обозначим через U матрицу размерности $m \times m$, состоящую из ортонормированных собственных векторов матрицы AA^T. В этом случае U^TU является единичной матрицей. Аналогично, через V обозначим матрицу размером $n \times n$, состоящую из ортонормированных собственных векторов матрицы A^TA . В этом случае V^TV также является единичной матрицей.

Результатом сингулярного разложения становятся матрицы U, S (диагональная матрица с сингулярными числами) и V, обладающие рядом свойств, определяющим из которых является следующее свойство: $A = USV^T$.

Строки матрицы U рассматриваются как образы термов. Аналогично, столбцы матрицы V^T рассматриваются как образы документов. Эти строки и столбцы задают искомое представление для термов и документов в k-мерном пространстве скрытых латентных факторов.

Столбцы матрицы V являются собственными векторами матрицы A^TA. Саму матрицу A^TA можно рассматривать как матрицу подобия документов-столбцов матрицы A. В свою очередь, столбцы матрицы V задают направления, по которым следует различать документы, и каждый документ можно представлять точкой в системе координат этих направлений. Для документа с номером i это i-я строка матрицы V или i-й столбец матрицы V^T.

Аналогично, матрицу AA^T можно рассматривать как матрицу подобия документов-строк матрицы A, т.е. термов, встречающихся в коллекции. Столбцы матрицы U задают направления, по которым следует различать термы, и каждый терм следует представлять точкой в системе координат этих направлений. Терм с номером j задается j-й строкой матрицы U.

Естественно предположить, что семантически близкие термы имеют близкие по расстоянию образы в пространстве латентных факторов. Аналогично, образы близких по тематике в рамках данной коллекции документов также должны быть близкими. Запрос пользователя q является вектором размерности m, t-й элемент которого

равен 1, если терм с номером t входит в запрос, и 0 – в противном случае.

Образом запроса q в пространстве латентных факторов является $q^{'}=q^{T}US^{-1}$. Тогда мера близости запроса q и документа d оценивается величиной скалярного произведения векторов $q^{'}$ и $V^{T}[d]$, причем $V^{T}[d]$ обозначает d-столбец матрицы V^{T}.

При пополнении коллекции, для которой уже проведено сингулярное разложение, относительно небольшим числом новых документов, тематика которых уже отражена в коллекции ранее проиндексированными документами, можно не вычислять разложение заново. Для этого достаточно аппроксимировать его, вычисляя образы новых документов на основе использования ранее вычисленных образов термов и весов факторов. Например, пусть d есть вектор весов термов нового документа (новый столбец матрицы A). Тогда его образ вычисляется по формуле: $d^{'}=S^{-1}U^{T}d$.

Считается, что метод латентного семантического индексирования не лишен недостатков. В частности, он мало пригоден для обработки больших коллекций документов из-за высоких требований к оперативной памяти и вычислительным ресурсам. Кроме того, он не позволяет напрямую обновлять индекс поисковой системы при пополнении коллекции документами новых тематик (требуется полный пересчёт). Метод также требует заранее подготовленной таблицы с весами термов в документах, что для больших коллекций может быть составлено только автоматически и может привести к ошибкам в результатах поиска.

В целом, несмотря на постоянный рост индексов известных поисковых систем, экспертные оценки показывают, что увеличение общего числа документов в WWW в целом ухудшило картину доступности информации. Ясно, что только применение совокупности поисковых машин, способно дать полноценную информационную картину для поисковых задач, при решении которых существенна полнота поиска. Тенденции в развитии ИПС показывают, что их совершенствование будет происходить за счет дальнейшей *интеллектуализации* и упрощения работы с ними. Резервами интеллектуализации ИПС являются гибкий язык запросов, «понимание» семантического смысла информации, которая, в отличие от баз данных обычно предполагает работу с не очень быстро изменяющейся информацией и не рассчитана на поддержание сотен

транзакций в минуту. Необходимо научить ИПС «понимать», что от нее хочет пользователь. Для этого широко используются словари, ряды синонимов, тезаурусы предметной области, методы распознания образов, нейронные сети, контент-анализ и т.д. Ставится задача создания системы интеллектуального поиска сопоставимых по качеству результатов поиска с экспертами.

5.2. Пример реализации процедуры поиска по методу латентного семантического индексирования

```
// Функция реализует поиск в гипертекстовых документах коллекции по запросу из нескольких термов
INT TextSearch(SSearchRequest* pstrSearchRequest, SSearchResult* pstrSearchResult)
{
INT i;
// Процедура поиска (сингулярное разложение) представлена в виде «черного ящика», т.е. из входной структуры функции pstrSearchRequest копируем данные во входную структуру функции поиска по методу LSI
// Для сингулярного разложения подготавливаем вектор запроса в пространстве термов
SSingularSearchRequest* SingularSearchWords = new SSingularSearchRequest;
SingularSearchWords>m_narWords.SetSize (m_NormFormTS.m_sarNormForm.GetSize());

for(i=0; i<SingularSearchWords->m_narWords.GetSize(); i++)
   SingularSearchWords->m_narWords[i] = 0;
for(i=0; i<pstrSearchRequest->m_narNFNumbers.GetSize(); i++)
   SingularSearchWords->m_narWords[pstrSearchRequest->m_narNFNumbers[i]] = 1;
// Вызываем функцию поиска по методу LSI
SSingularSearchResult * SingularSearchResults
=SingularSearchDo(SingularSearchWords,
   m_NormFormTS.m_sIndexFilesPath+"\\pU.txt",
   m_NormFormTS.m_sIndexFilesPath+"\\pS.txt",
   m_NormFormTS.m_sIndexFilesPath+"\\pV.txt");
```

```
    // Очищаем структуру, предназначенную для записи найденных страниц
    pstrSearchResult->m_narPageNum.RemoveAll();
    pstrSearchResult->m_narPageMark.RemoveAll();
    // Копируем найденные страницы с оценками, полученными в функции SingularSearchDo, в выходную структуру найденных документов.
    // Сортировку в данном случае производить нет необходимости, т.к. она была проведена в SingularSearchDo
    for (i=0;i<SingularSearchResults->m_narPages.GetSize() ;i++)
    {
      pstrSearchResult->m_narPageNum.Add(SingularSearchResults->m_narPages[i]);
      pstrSearchResult->m_narPageMark.Add(SingularSearchResults->m_narMarks[i]);
    }
    // Очищаем память от временных структур
    delete SingularSearchWords;
    delete SingularSearchResults;
    // Оставляем только 100 лучших страниц из отсортированных
    if (pstrSearchResult->m_narPageNum.GetSize() > 100)
    {
      for (i=pstrSearchResult->m_narPageNum.GetSize()-1; i >= 100; i--)
      {
            pstrSearchResult->m_narPageNum.RemoveAt(i);
            pstrSearchResult->m_narPageMark.RemoveAt(i);
      }
    }
    return pstrSearchResult->m_narPageNum.GetSize();
    }
```

Ниже приводится *описание поисковых индексов*, используемых в различных вариантах заданий к лабораторной работе.

Поисковый индекс для булевского поиска в текстах. Он хранится в объекте CLVSWordOnPage, который в функции TextSearch() доступен через экземпляр объекта CNormFormTS, указанный в примере. Функция TextSearch содержит уже проинициализированный экземпляр CLVSWordOnPage:

```
class CNormFormTS{
   CLVSWordOnPage m_WordOnPage;
//Объект хранит информацию по употребляемости термов
   INT m_nSizeOfWordOnPageArray;
//Количество элементов в m_WordOnPage
}
class CLVSWordOnPage{
// Номера страниц, содержащих терм, и их оценки хранятся последовательно для всех термов внутри одного линейного массива. Для работы необходимо определять границы, внутри которых идёт информация о данном терме, а также получить номер стартовой позиции в линейном массиве документов (или оценок) для терма с номером n
   INT GetNFPageStartPos(INT n);
// Получение номера документа, содержащего терм. Здесь n является последовательным номером записи в линейном массиве
   INT GetPageNum(INT n);
// Получение оценки терма для документа
   BYTE GetPageMark(INT n);
}
```

Для того чтобы просмотреть все документы, содержащие терм, необходимо проделать следующие действия:

```
// Предположим, что номер терма хранится в переменной nTerm
INT nTerm = 40;
// Определяем границы интервала, в котором хранится информация для данного терма в линейном массиве
INT nStart = m_NormFormTS.m_WordOnPage.GetNFPageStartPos(nTerm);
INT nEnd = -1;
If (nTerm != m_NormFormTS.m_sarNormForm.GetSize()-1)
// Если нормализованная форма не самая последняя в списке, то
   nEnd = m_NormFormTS.m_WordOnPage.GetNFPageStartPos(nTerm+1);
else
// Если нормализованная форма идёт самой последней, то правая граница длиной массива равна
   nEnd = m_NormFormTS.m_nSizeOfWordOnPageArray;
```

```
// Пример доступа ко всем номерам документов и оценкам по данному терму
for(INT i=nStart; i < nEnd; i++)
{
   INT nPage = m_NormFormTS.m_WordOnPage.GetPageNum(i);
   BYTE nMark = m_NormFormTS.m_WordOnPage.GetPageMark(i);
}
```

Поисковый индекс для булевского поиска среди изображений. Он хранится в объекте CLVSWordOnPage, который в функции TextSearch() доступен через экземпляр объекта CNormFormTS, указанный в примере. Функция TextSearch содержит уже проинициализированный экземпляр CLVSWordOnPage:

```
class CNormFormTS{
   CLVSWordOnPage m_WordOnPage;
//Объект хранит информацию по употребляемости термов
   INT m_nSizeOfWordFromImgArray;
//Количество элементов в m_WordOnPage
}
class CLVSWordOnPage{
// Номера изображений в подписи терма и их оценки хранятся последовательно для всех термов внутри одного линейного массива. Для работы необходимо определять границы, внутри которых идёт информация по данному терму, а также получить номер стартовой позиции в линейном массиве документов (или оценок) для терма с номером n
   INT GetNFImgStartPos(INT n);
// Получение номера изображения в подписи терма. Здесь n является номером записи в линейном массиве
   INT GetImgNum (INT n);
// Получение оценки терма в подписи к изображению
   BYTE GetPageMark(INT n);
}
```

Для того чтобы просмотреть все изображения, содержащие терм в своём описании, необходимо проделать следующие действия:

```
// Предположим что номер терма хранится в переменной nTerm
INT nTerm = 40;
```

```
// Определяем границы интервала, в котором хранится информация для данного терма в линейном массиве
INT nStart = m_NormFormTS.m_WordOnPage.GetNFImgStartPos(nTerm);
INT nEnd = -1;
If (nTerm != m_NormFormTS.m_sarImgNormForm.GetSize()-1)
// Если нормализованная форма не самая последняя в списке, то
nEnd = m_NormFormTS.m_WordOnPage.GetNFImgStartPos(nTerm+1);
else
// Если нормализованная форма идет самой последней, то правая граница длина массива
nEnd = m_NormFormTS.m_nSizeOfWordFromImgArray;
// Пример доступа ко всем номерам изображений и оценкам по данному терму
for(INT i=nStart; i < nEnd; i++)
{
INT nImg = m_NormFormTS.m_WordOnPage.GetImgNum(i);
BYTE nMark = m_NormFormTS.m_WordOnPage.GetImgMark(i);
}
```

Поисковый индекс для векторного поиска в текстах. Поисковый индекс векторной модели хранится в объекте CLVSVectorModel. Данный объект в функции TextSearch() требуется проинициализировать. Для этого необходимо с диска загрузить данные поискового индекса:

```
m_VectorModel.LoadNormMatrix(m_NormFormTS.m_sIndexFilesPath + "\\vectorind.dat");
```

Чтобы определить число документов коллекции, необходимо ввести:

```
INT nDoc = m_VectorModel.m_parDocVector.GetSize();
```

Определить нормализованную оценку документа i по терму nTerm можно следующим образом:

```
DOUBLE nMark = m_VectorModel.GetMark(i, nTerm);
```

5.3. Порядок выполнения работы

Сущность лабораторной работы состоит в том, чтобы в ходе ее выполнения реализовать функцию поиска, соответствующую варианту задания для уже подготовленной поисковой машины, реализованной на VisualC++ 6.0. При реализации воспользоваться готовым поисковым индексом, как это было представлено выше в примере поиска по модели латентного семантического индексирования.

Этапы выполнения лабораторной работы:

- изучить теоретический материал, рассмотреть пример;
- для поиска в текстах коллекции документов необходимо вставить свой код в функцию TextSearch() объекта CLVSSearch (рис. 5.1):
- для поиска в изображениях коллекции документов необходимо вставить свой код в функцию ImageSearch() объекта CLVSSearch (рис. 5.1);

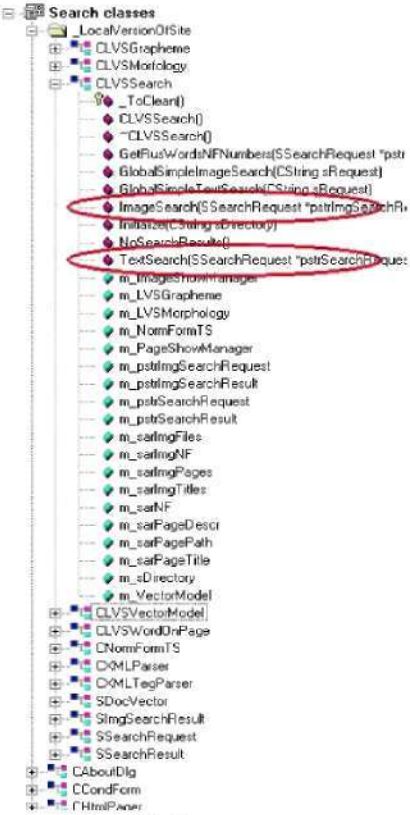


Рис. 5.1. Директория классов

- в директории с исполняемым exe-файлом должны находиться две папки: *files* (содержат поисковый индекс системы) и *sites* (содержит коллекцию гипертекстовых документов). Для сравнения дается исполняемый файл LSISearch.exe, где реализовано латентное

семантическое индексирование для текстов и булевская модель с логическим *AND* для изображений;

- проанализировать качество поиска; сравнить реализуемую в варианте поисковую модель с любой другой из приведенных в теоретическом описании лабораторной работы;
- представить отчет о работе, включающий авторские выводы и оценку достижения поставленных целей.

5.4. Содержание отчета

1. Цель работы.
2. Постановка задачи согласно выбранному варианту.
3. Описание самостоятельно проделанной работы (анализ этапов поиска; варианта задания; выполненных этапов; результатов работы на примерах, их сравнительная характеристика).
4. Выводы.

Приложение: необходимые для защиты лабораторной работы "скриншоты" экрана.

5.5. Варианты заданий

Представленные ниже варианты включают различные задачи поиска в текстах и изображениях документов коллекции по булевской модели с логическими операциями *AND* или *OR* между ключевыми словами запроса, а также задачи поиска в текстах документов коллекции по векторной модели. Вариант задания выбирается студентом по согласованию с преподавателем.

Конкретные варианты заданий представлены ниже.

1. Поиск в текстах документов коллекции по булевской модели с логическим *AND*

Организовать пошаговый поиск в текстах документов коллекции по булевской модели с логическим *AND* между ключевыми словами запроса, с учётом заданной оценки веса терма в документе. Оценка вычислена как повторяемость терма внутри документа с учётом участия документа в заголовке и описании документа.

Шаг 1. Записать в выходную структуру pstrSearchResult все документы, содержащие первый терм из запроса (переписываются номера документов и оценка значимости терма для документа).

Шаг 2. Отсечь из множества документов, находящихся в pstrSearchResult те документы, которые не содержат остальных ключевых слов.

Шаг 3. Отсортировать найденные документы по оценке.

Шаг 4. Оставить только 100 лучших документов по результатам поиска.

2. Поиск в текстах документов коллекции по булевской модели с логическим *OR*

Организовать пошаговый поиск в текстах документов коллекции по булевской модели с логическим *OR* между ключевыми словами запроса, аналогично представленному в варианте 1.

3. Поиск в изображениях документов коллекции по булевской модели с логическим *AND*

Организовать пошаговый поиск в изображениях коллекции по булевской модели с логическим *AND* между ключевыми словами запроса с учётом заданной оценки веса терма.

Шаг 1. Записать в выходную структуру pstrImgSearchResult все номера изображений, содержащих первый терм из запроса (переписываются номера изображений и оценка значимости терма для изображения).

Шаг 2. Отсечь из множества документов, находящихся в pstrImgSearchResult те, что не содержат остальных ключевых слов.

Шаг 3. Отсортировать найденные изображения по оценке.

Шаг 4. Оставить только 100 лучших изображений в результатах поиска.

4. Поиск в изображениях документов коллекции по булевской модели с логическим *OR*

Организовать пошаговый поиск в изображениях коллекции по булевской модели с логическим *OR* между ключевыми словами запроса, аналогично представленному в варианте 3.

5. Поиск в текстах документов коллекции с использованием векторной модели

Организовать пошаговый поиск в текстах документов коллекции с использованием поискового индекса векторной модели.

Шаг 1. Инициализировать векторный поисковый индекс (объект CLVSVectorModel)

Шаг 2. Переписать пользовательский запрос в виде вектора в пространстве термов коллекции, содержащего оценки. Вектор следует записать в виде структуры типа SDocVector. Необходимо вектор пользовательского запроса привести к единичной длине.

Шаг 3. Вычислить оценки схожести вектора запроса с векторами документов через скалярное произведение.

Шаг 3. Отсортировать документы по оценке.

Шаг 4. Оставить не более 100 документов с ненулевой оценкой.

6. Поиск в текстах документов коллекции с использованием векторной модели

Организовать пошаговый поиск в текстах документов коллекции с использованием поискового индекса векторной модели, аналогично варианту 5. Документы в результатах поиска должны содержать все термы из запроса пользователя (аналог логического *AND*).

Контрольные вопросы

1. Укажите соответствие между аспектами, характеризующими информационный объект: а) контент, б) структура, в) контекст, – и описанием этих аспектов: 1) учет смыслового содержания, 2) информационное содержание объекта в виде понятий и отношений между ними, 3) набор атрибутов и информационных связей.

2. Укажите соответствие ИПС: а) документальных, б) гипертекстовых, в) фактографических, – и их характеристик по полноте и шуму: 1) п = 1, ш = 0; 2) п = [0,9; 1,0], ш=[0,1; 0,2]; 3) п = 0,5, ш = 1.

3. Документы, найденные в результате поиска ранжируются: а) по номеру в поисковом индексе, б) по оценке релевантности, в) по размеру документа, г) по булевской модели поиска.

4. Какие проблемы возникают при пополнении индекса поисковой системы в Интернет?

5. Процесс индексирования в поисковых системах это: а) выбор способа представления данных, б) нахождение наиболее

релевантных документов, в) процесс сбора данных о коллекции документов, г) поиск по пользовательскому запросу в коллекции документов, д) формирование коллекции статистических данных.

6. Качество поиска оценивается: а) по выборке нескольких документов из результатов поиска и проверке их на соответствие запросу, б) по точности поиска, в) по полноте поиска, г) по правильности оценки первых трёх документов, д) по скорости поиска.

7. Какой из известных Вам методов ближе всего приблизился к решению проблемы синонимии и омонимии: а) булевский поиск, б) векторный поиск, в) вероятностный поиск, г) латентное семантическое индексирование.

8. Ваша самооценка самостоятельно выполненной работы по критерию 1: 2_1_0 по критерию 2: 2_1_0 по критерию 3: 2_1_0.

Библиографический список

1. Афонин В.Л., Макушкин В.А. Интеллектуальные робототехнические системы. – М.: Интернет-университет информационных технологий, 2005. – 208 с.

2. Базы данных: интеллектуальная обработка информации / В.В. Корнеев и др. – М.: Молгачева С.В., 2000. – 496 с.

3. Башмаков А.И., Башмаков И.А. Интеллектуальные информационные технологии. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2005. – 304 с.

4. Джексон П. Введение в экспертные системы. – М.: Издательский дом «Вильямс», 2001. – 624 с.

5. Зозуля Ю.И. Интеллектуальные нейросистемы. – М.: Радиотехника, 2003. – 144 с.

6. Конюшенко В.В. Начало работы с MATLAB / Пер. с англ. В.В Конюшенко. – konushenko@afrodita.phys.msu.su.

7. Ландэ Д.В. Поиск знаний в Internet. Профессиональная работа / Пер. с англ. – М.: Издательский дом «Вильямс», 2005. – 272 с.

8. Родзин С.И., Ковалев С.М. Информационные технологии: интеллектуализация обучения, моделирование эволюции, распознавание речи. – Ростов-на-Дону: Изд-во СКНЦ ВШ, 2002. – 224 с.

9. Смолин Д.В. Введение в искусственный интеллект. Конспект лекций. – М.: ФИЗМАТЛИТ, 2004. – 208 с.

10. Частиков А.П. и др. Разработка экспертных систем. Среда CLIPS. – СПб.: БХВ-Петербург, 2003. – 608 с.

11. Штовба С.Д. Введение в теорию нечетких множеств и нечеткую логику. – http://matlab.exponenta.ru/fuzzylogic/book1/index.php.

12. Ярушкина Н.Г. Основы теории нечетких и гибридных систем: Учеб. пособие. – М.: Финансы и статистика, 2004. – 320 с.: ил.

Родзин Сергей Иванович
Гребенюк Евгений Юрьевич
Злыгостев Алексей Сергеевич
Шишков Сергей Александрович

СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА
ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Ответственный за выпуск — *Гребенюк Е.Ю.*
Редактор — *Кочергина Т.Ф.*
Корректор — *Селезнева Н.И.*

ЛР № 020565 от 23.06.97 г.
Формат 60×84 $^{1}/_{16}$
Усл.п.л. — 7,3
Заказ № _____

Подписано к печати ____.____.07 г.
Бумага офсетная
Печать офсетная
Уч. – изд.л. — 7,0
Тираж _____ экз.

«С»

Издательство Технологического института ЮФУ
ГСП 17 А, Таганрог, пер. Некрасовский, 44
Типография Технологического института ЮФУ
ГСП 17 А, Таганрог 28, ул. Энгельса, 1