Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

О. В. ВЕСЕЛОВ

НЕЧЕТКАЯ ЛОГИКА И НЕЙРОННЫЕ СЕТИ В СИСТЕМАХ УПРАВЛЕНИЯ И ДИАГНОСТИКЕ

Учебное пособие

Владимир 2023

УДК 510.6+004.032.26
ББК 22.12+32.818.1
В38

Рецензенты:

Кандидат технических наук, доцент
зав. кафедрой промышленной электроники
и интеллектуальных цифровых систем
Московского государственного технологического университета «СТАНКИН»
В. В. Филатов

Доктор технических наук, профессор
зав. кафедрой информатики и защиты информации
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых
М. Ю. Монахов

Веселов, О. В.

В38 Нечеткая логика и нейронные сети в системах управления и диагностике : учеб. пособие / О. В. Веселов ; Владим. гос. ун-т им. А. Г. и Н. Г. Столетовых. – Владимир : Изд-во ВлГУ, 2023. – 288 с. – ISBN 978-5-9984-1641-5.

Изложены вопросы применения нечеткой логики и нейронных сетей в системах управления и диагностике с использованием MATLAB на основе пакетов NNTool, FuzzyLogic, Simulink.

Предназначено для магистрантов, обучающихся по направлению подготовки 15.04.06 – Мехатроника и робототехника.

Рекомендовано для формирования общепрофессиональных компетенций в соответствии с ФГОС ВО.

Табл. 14. Ил. 165. Библиогр.: 32 назв.

УДК 510.6+004.032.26
ББК 22.12+32.818.1

ISBN 978-5-9984-1641-5

ПРЕДИСЛОВИЕ

Что такое искусственные нейронные сети и нечеткая логика? Что они могут делать? Как работают и как их использовать? Эти и другие вопросы задают специалисты из разных областей. Найти вразумительный ответ нелегко. Университетских курсов мало, а соответствующая литература слишком обширна и специализированна. В большинстве своем пособия написаны на техническом жаргоне и предполагают свободное владение разделами высшей математики.

Не все обучаемые владеют знаниями в необходимом объеме для использования в решении различных задач. Кроме того, уровень знаний студентов может различаться, что вызывает дополнительные сложности. Поэтому материал, приведенный в пособии, детально описывает все операции по использованию тех или иных блоков и команд.

MATLAB, являясь мощным инструментом в проведении исследований, позволяет решать различные задачи в различных отраслях знаний.

Пособие призвано помочь решить две задачи: научится пользоваться средствами программы MATLAB и освоить материал по нейронным сетям и нечеткой логике на основе использования прикладных пакетов «Simulink», NNTool и Fuzzylogic.

В издании рассматриваются вопросы управления и диагностики электромеханических систем. На конкретном примере показаны этапы освоения материала. Изучение начинается с создания модели

ЭМС, изучения ее свойств, получения данных, необходимых для проведения дальнейших исследований. Затем выбирается нейронная сеть, обучается, проходит тренировку и используется в оценке состояния ЭМС. Аналогично строится нечеткий контроллер.

Пособие необязательно читать от начала до конца. Каждый раздел предполагается замкнутым, поэтому для понимания достаточно лишь знакомства с содержанием выбранного раздела. Все важные понятия формулируются обычным языком. Математические выкладки используются, если они делают изложение более ясным.

Издание имеет практическую направленность. Никакой другой метод не позволит добиться столь же глубокого понимания.

Читатели, пожелавшие продолжить углубленное теоретическое изучение, могут воспользоваться литературой, приведенной в библиографическом списке в конце пособия.

ВВЕДЕНИЕ

Мир событий, явлений и объектов в технических системах меняется настолько интенсивно, что знания, которые имелись не в состоянии объяснить явления и события, происходящие сегодня. Необходимы новые знания. Используя терминологию причинно-следственных связей и отношений определим знания как совокупность - приобретение знаний их обработка, представление и использование. Само **знание** определим как форму существования и систематизации результатов познавательной деятельности человека. Учитывая особенности современного цифрового мира, знание приобретает свойства искусственного интеллекта и тогда его можно представить в следующем звучании. **Знание** — в теории искусственного интеллекта, базах знаний и экспертных системах есть совокупность данных, фактов, сведений и правил вывода о мире, включающих в себя информацию о свойствах объектов, закономерностях процессов и явлений, а также правилах использования этой информации для принятия решений.

Процесс научного познания, а также различные формы освоения мира не всегда осуществляются в развёрнутом, логически и фактически доказательном виде. Здесь в процесс включается интуиция. Она представляет собой своеобразный тип мышления, когда отдельные звенья процесса мышления происходят бессознательно, а предельно ясно осознаётся именно итог мысли. Интуиции бывает достаточно для усмотрения истины, но её недостаточно, чтобы убедить в этой истине других и самого себя. Для этого необходимо доказательство. Если его нет, то это гипотеза.

Сложные системы искусственного интеллекта, основанные на нейросетевой технологии, а также экспертные системы, основанные на логической модели баз знаний, демонстрируют поведение, которое имитирует человеческое мышление и интуицию. Обучение таких систем — эвристический процесс, состоящий в нахождении решения

задачи на основе ориентиров поиска, недостаточных для получения логического вывода. Для интуиции характерна быстрота формулирования гипотез и принятия решений, а также недостаточная осознанность его логических оснований.

Логический вывод информации, конкретных и обобщенных сведений и данных производится в базах знаний и экспертных системах, использующих языки средства логического программирования (язык Пролог). Эти системы явно демонстрируют логический вывод новой информации, осмысленных сведений, данных, используя правила логического вывода и факты, закладываемые в базы знаний.

Классификация методов искусственного интеллекта (ИИ)

Есть разные мнения о том, как классифицировать методы ИИ. Предлагается следующая классификация, которая состоит из пяти пунктов (рис.1.):

1. Искусственные нейронные сети
2. Нечеткая логика (нечеткие множества и мягкие вычисления)
3. Системы, основанные на знаниях (экспертные системы)
4. Эволюционное моделирование (генетические алгоритмы, многоагентные системы)
5. Machine Learning (Data Mining и анализ данных и, поиск закономерностей в хранилищах данных)

В рамках исследований и разработок систем ИИ используется разнообразные методы. К таким методам относятся:

- экспертные системы и системы поддержки принятия решений;
- представление знаний;
- поиск в пространстве состояний
- обработка естественного языка;
- машинное обучение и искусственные нейронные сети;
- генетические алгоритмы;
- многоагентные системы.

Кратко опишем содержание и применение некоторых из этих методов.

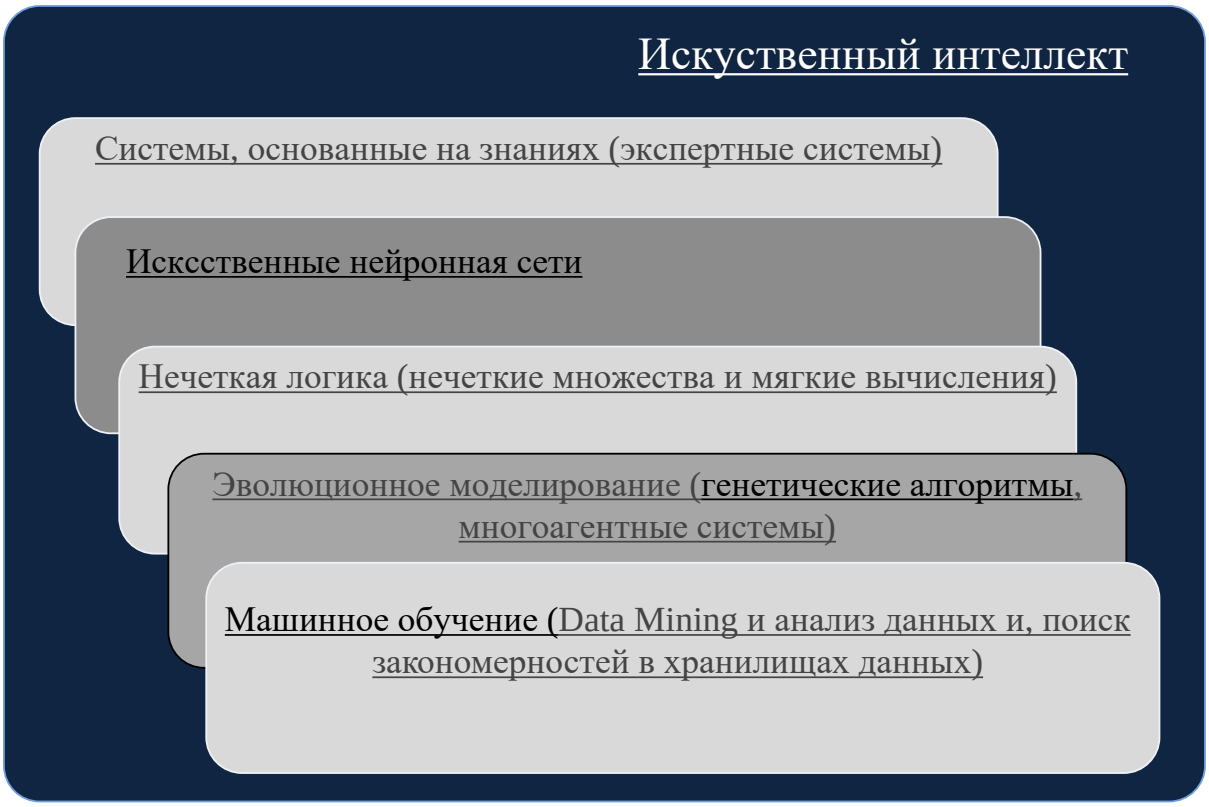


Рис. 1. Средства искусственного интеллекта

Поиск в пространстве состояний — это, в принципе, не технология ИИ, но тот или иной вид поиска всегда лежит в основе всех остальных технологий и методов искусственного интеллекта, поэтому очень важно знать, какие методы поиска существуют, почему все они сводятся к поиску в пространстве состояний системы ИИ и, в конечном итоге с математической точки зрения, к поиску на графе. И, самое главное, каким образом системы ИИ могут для этого применять разного рода эвристики.

Обработка естественного языка — не менее важная фундаментальная технология, которая позволяет системам ИИ общаться с пользователями на обычном языке даже при помощи голоса Это важно в том числе и потому, что пользователями систем ИИ всё чаще и чаще становятся люди, далёкие от программирования. Существует большое количество конкретных методов обработки естественного языка, однако до конца эта задача не решена до сих пор: ИИ системы сложно понимают устную или письменную речь человека, хотя в последнее время в этом направлении сделаны существенные прорывы

В системах ИИ, основанных на знаниях, используются различные формализмы для *представления знаний*. Сегодня существует несколько таких методов, довольно универсальных, которые в той или иной мере отражают возможности людей по описанию своих знаний. Более того, с момента создания первых систем, основанных на знании, к настоящему моменту разработано большое количество математических методов, позволяющих справляться с неполнотой, недостоверностью, неопределённостью, неточностью, нечёткостью и многими другими, что делает системы ИИ способными действовать и принимать решения в условиях неопределённости, как это делает человек.

Так вот чуть ли не главным классом систем, основанных на знаниях, являются *экспертные системы* которые, в свою очередь, часто составляют собой ядро различного рода *систем поддержки принятия решений*.

Экспе́ртная систе́ма (ЭС, expert system) — компьютерная программа, способная заменить специалиста-эксперта в разрешении проблемной ситуации. В информатике экспертные системы рассматриваются совместно с базами знаний как модели поведения экспертов в определенной области знаний с использованием процедур логического вывода и принятия решений, а базы знаний — как совокупность фактов и правил логического вывода в выбранной предметной области деятельности.

Классификация ЭС по решаемой задаче: интерпретация данных, управление, диагностирование, мониторинг, проектирование, прогнозирование, обучение.

Экспертные системы содержат в себе базу знаний в какой-либо проблемной области, в которой собраны (и для динамических систем постоянно актуализируются) знания экспертов, позволяющие принимать обоснованные решения. Такие решения готовятся на основе логического вывода в рамках работы универсальной машины вывода на основе имеющихся знаний и наличествующих фактов. А системы поддержки принятия решений сегодня широко используются для подготовки и обоснования решений в сложных ситуациях, требующих быстрого реагирования, либо в ситуациях, когда естественный интеллект человека не может справиться с объёмом и многообразием данных.

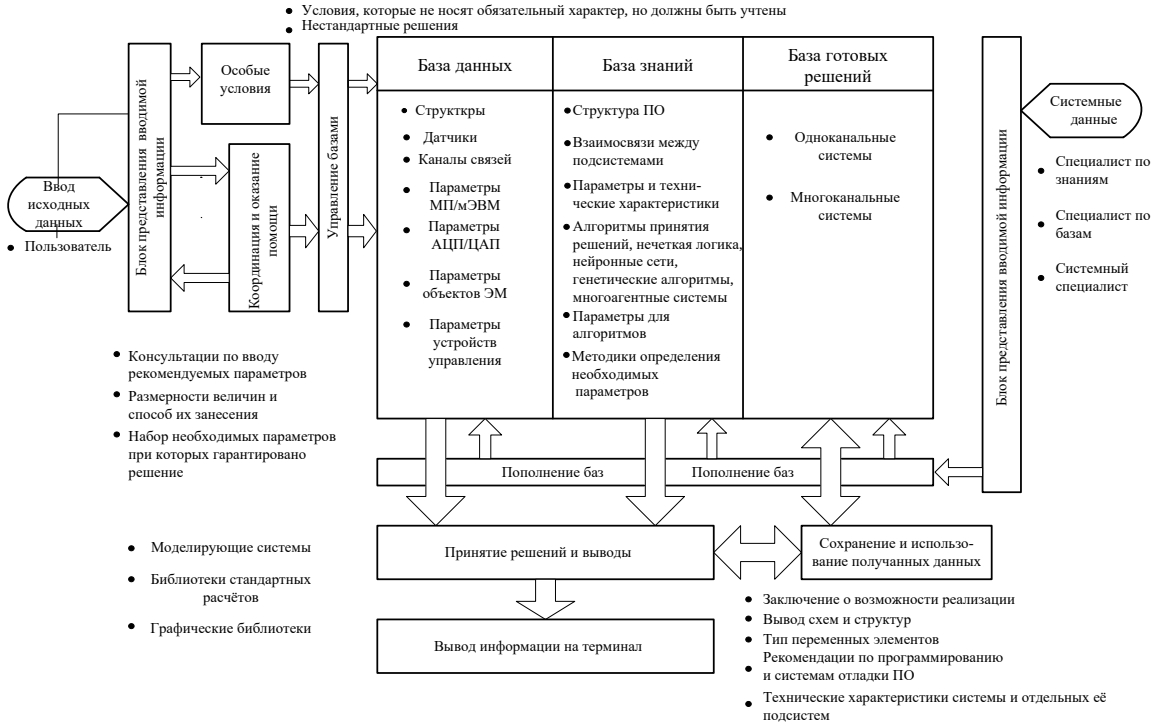


Рис. 2. Экспертная система

В настоящее время наиболее востребованными и широко используемыми являются различные методы *машинного обучения* и, в частности, *искусственные нейронные сети*. Фактически, эти методы решают несколько конкретных задач — аппроксимация функций, классификация, кластеризация, снижение размерности и некоторые другие. Если какая-либо задача сводится к какой-либо из базовых для машинного обучения, то её можно успешно и достаточно эффективно решать этими методами. Проблема в том, что результаты, выдаваемые этими методами, сложно интерпретируются и, как следствие, их сложно объяснить.

В рамках эволюционного подхода наиболее широко распространёнными являются *генетические алгоритмы*. Фактически, это метод оптимизации, который позволяет эвристически найти глобальный экстремум мультимодальной, нелинейной, недифференцируемой функции от многих переменных. Для его использования параметры задачи представляются в виде набора генов, затем к ним применяется процесс искусственного отбора. Очень интересно применять генетические алгоритмы для «выращивания» систем ИИ, призванных оптимально работать в заданной окружающей среде для достижения поставленной цели.

Задача формализуется таким образом, чтобы её решение могло быть закодировано в виде вектора («генотипа») генов, где каждый ген может быть битом, числом или неким другим объектом. В классических реализациях генетического алгоритма (ГА) предполагается, что генотип имеет фиксированную длину. Однако существуют вариации ГА, свободные от этого ограничения.

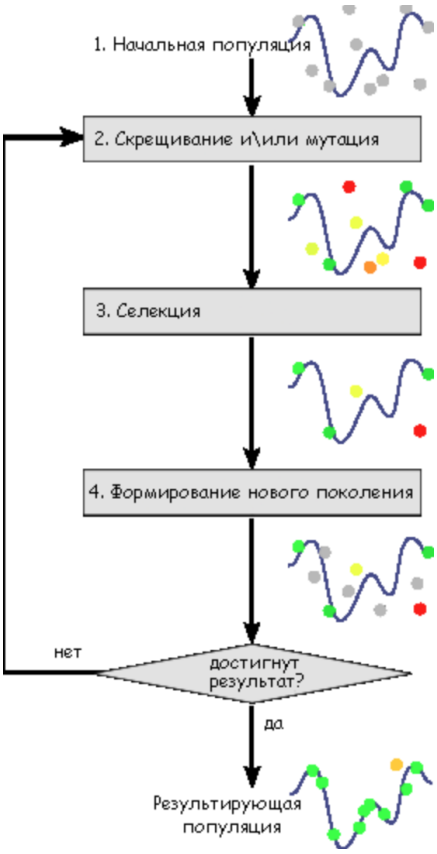


Рис. 3. Генетический алгоритм

Некоторым, обычно случайным, образом создаётся множество генотипов начальной популяции. Они оцениваются с использованием «функции приспособленности», в результате чего с каждым генотипом ассоциируется определённое значение («приспособленность»), которое определяет насколько хорошо фенотип, им описываемый, решает поставленную задачу.

При выборе «функции приспособленности» (или fitness function в англоязычной литературе) важно следить, чтобы её «рельеф» был «гладким».

Из полученного множества решений («поколения») с учётом значения «приспособленности» выбираются решения (обычно лучшие особи имеют большую вероятность быть выбранными), к которым применяются «генетические операторы» (в большинстве случаев «скрещивание» — crossover и «мутация» — mutation), результатом чего является получение новых решений. Для них также вычисляется значение приспособленности, и затем производится отбор («селекция») лучших решений в следующее поколение.

Этот набор действий повторяется итеративно, так моделируется «эволюционный процесс», продолжающийся несколько жизненных циклов (*поколений*), пока не будет выполнен критерий остановки алгоритма (рис.3.). Таким критерием может быть:

- нахождение глобального, либо субоптимального решения;
- исчерпание числа поколений, отпущенных на эволюцию;
- исчерпание времени, отпущенного на эволюцию.

Генетические алгоритмы служат, главным образом, для поиска решений в многомерных пространствах поиска. Таким образом, можно выделить следующие этапы генетического алгоритма:

1. Задать целевую функцию (приспособленности) для особей популяции
2. Создать начальную популяцию

- (Начало цикла)

1. Размножение (скрещивание).
2. Мутирование.
3. Вычислить значение целевой функции для всех особей.
4. Формирование нового поколения (селекция).
5. Если выполняются условия остановки, то (конец цикла), иначе (начало цикла).

Создание начальной популяции.

Перед первым шагом нужно случайным образом создать начальную популяцию; даже если она окажется совершенно неконкурентоспособной, вероятно, что генетический алгоритм всё равно достаточно быстро переведёт её в жизнеспособную популяцию. Таким образом, на первом шаге можно особенно не стараться сделать слишком уж приспособленных особей, достаточно, чтобы они соответствовали формату особей популяции, и на них можно было подсчитать функцию приспособленности (Fitness). Итогом первого шага является популяция H, состоящая из N особей.

Отбор (селекция)

На этапе отбора нужно из всей популяции выбрать определённую её долю, которая останется «в живых» на этом этапе эволюции. Есть разные способы проводить отбор. Вероятность выживания особи h должна зависеть от значения функции приспособленности Fitness(h). Сама доля выживших s обычно является параметром генетического алгоритма, и её просто задают заранее. По итогам отбора из N особей популяции H должны остаться sN особей, которые войдут в итоговую популяцию H'. Остальные особи погибают.

- Турнирная селекция — сначала случайно выбирается установленное количество особей (обычно две), а затем из них выбирается особь с лучшим значением функции приспособленности
- Метод рулетки — вероятность выбора особи тем вероятнее, чем лучше её значение функции приспособленности.

- Метод ранжирования — вероятность выбора зависит от места в списке особей отсортированном по значению функции приспособленност.
- Сигма-отсечение — для предотвращения преждевременной сходимости генетического алгоритма используются методы, масштабирующие значение целевой функции. Вероятность выбора особи тем больше, чем оптимальнее значение масштабируемой целевой функции.

Выбор родителей.

Размножение в генетических алгоритмах требует для производства потомка нескольких родителей, обычно двух.

Можно выделить несколько операторов выбора родителей:

1. Панмиксия — оба родителя выбираются случайно, каждая особь популяции имеет равные шансы быть выбранной
2. Инбридинг — первый родитель выбирается случайно, а вторым выбирается такой, который наиболее похож на первого родителя
3. Аутбридинг — первый родитель выбирается случайно, а вторым выбирается такой, который наименее похож на первого родителя

Инбридинг и аутбридинг бывают в двух формах: фенотипной и генотипной. В случае фенотипной формы похожесть измеряется в зависимости от значения функции приспособленности (чем ближе значения целевой функции, тем особи более похожи), а в случае генотипной формы похожесть измеряется в зависимости от представления генотипа (чем меньше отличий между генотипами особей, тем особи похожее).

Размножение (скрещивание).

Размножение в разных алгоритмах определяется по-разному — оно, конечно, зависит от представления данных. Главное требование к размножению — чтобы потомок или потомки имели возможность унаследовать черты обоих родителей, «смешав» их каким-либо способом.

Почему особи для размножения обычно выбираются из всей популяции H, а не из выживших на первом шаге элементов H' (хотя последний вариант тоже имеет право на существование?). Дело в том, что главный недостаток многих генетических алгоритмов — отсут-

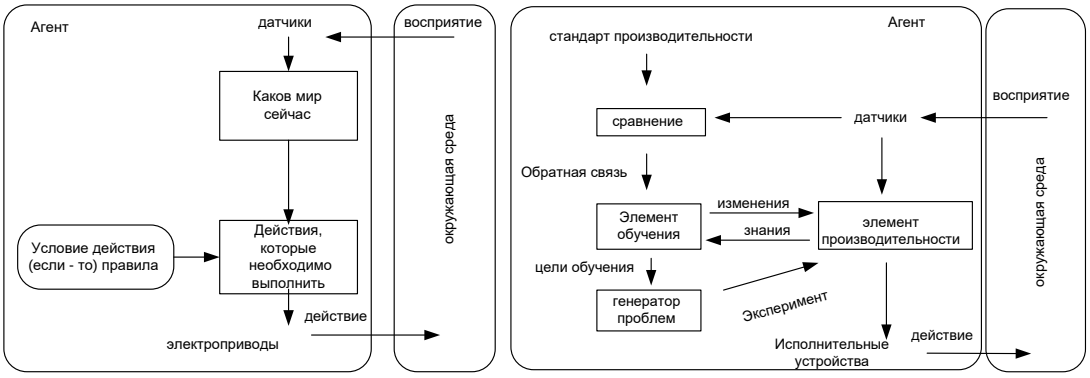
ствие разнообразия (diversity) в особях. Достаточно быстро выделяется один-единственный генотип, который представляет собой локальный максимум, а затем все элементы популяции проигрывают ему отбор, и вся популяция «забивается» копиями этой особи. Есть разные способы борьбы с таким нежелательным эффектом; один из них — выбор для размножения не самых приспособленных, но вообще всех особей. Однако такой подход вынуждает хранить всех существовавших ранее особей, что увеличивает вычислительную сложность задачи. Поэтому часто применяют методы отбора особей для скрещивания таким образом, чтобы «размножались» не только самые приспособленные, но и другие особи, обладающие плохой приспособленностью. При таком подходе для разнообразия генотипа возрастает роль мутаций.

Мутации.

К мутациям относится то же самое, что и к размножению: есть некоторая доля мутантов m, являющаяся параметром генетического алгоритма, и на шаге мутаций нужно выбрать mN особей, а затем изменить их в соответствии с заранее определёнными операциями мутации.

Наконец, *многоагентные системы* являются главным представителем агентного подхода (рис.4). Это интеллектуальные системы, состоящие из большого количества взаимодействующих агентов различной природы, среди которых могут быть обычные технические системы, системы ИИ, и даже люди со своим естественным интеллектом. Главная особенность — агенты в какой-то мере автономны в принятии решений, но вся система в целом решает задачу и достигает цель, которая недоступна для каждого агента в отдельности. И сегодня это одно из самых «горячих» направлений искусственного интеллекта.

Многоагентная система (MAS, англ. *Multi-agent system*)— это система, образованная несколькими взаимодействующими интеллектуальными агентами (рис.4а). Многоагентные системы могут быть использованы для решения таких проблем, которые сложно или невозможно решить с помощью одного агента или монолитной системы.



а б

Рис. 4. Представление агентов: а – простой агент, б – умный (обучаемый) агент

Свойства. МАС также относятся к самоорганизующимся системам, так как в них ищется оптимальное решение задачи без внешнего вмешательства (рис.4б). Под оптимальным решением понимается решение, на которое потрачено наименьшее количество энергии в условиях ограниченных ресурсов.

Главное достоинство МАС— это гибкость. Многоагентная система может быть дополнена и модифицирована без переписывания значительной части программы. Также эти системы обладают способностью к самовосстановлению и обладают устойчивостью к сбоям, благодаря достаточному запасу компонентов и самоорганизации.

Применение МАС. Многоагентные системы применяются в нашей жизни в графических приложениях, например, в компьютерных играх. Агентные системы также были использованы в фильмах. Теория МАС используется в составных системах обороны. Также МАС применяются в транспорте, логистике, графике, геоинформационных системах, робототехнике и многих других. Многоагентные системы хорошо зарекомендовали себя в сфере сетевых и мобильных технологий, для обеспечения автоматического и динамического баланса нагруженности, расширяемости и способности к самовосстановлению.

Основная цель управления электромеханическими системами и технической диагностики состоят в организации эффективных процессов управления и оценка качества технического состояния объектов различной сложности.

Одним из факторов, существенно влияющих на эффективность процесса управления и оценки технического состояния, является качество алгоритмов.

Необходимость увеличения производительности труда на операциях диагноза, сокращения времени обнаружения, поиска и устранения неисправностей, уменьшения объемов и сложности средств диагноза вызывает интерес к разработке методов построения алгоритмов диагноза, требующих минимальных затрат на их реализацию. Построение таких алгоритмов во многих случаях сопряжено с большими вычислительными затратами, и поэтому зачастую удовлетворяются упрощенными алгоритмами, затраты на реализацию которых малы, но не обязательно минимальны.

Интуитивные методы построения алгоритмов диагноза не могут гарантировать получения объективного заключения о действительном техническом состоянии объекта. Отсюда следует необходимость разработки формальных методов построения алгоритмов диагноза технического состояния объектов. Это особенно важно для сложных объектов, насчитывающих десятки, сотни и тысячи функционально и конструктивно взаимосвязанных компонент и зачастую требующих многих часов для обнаружения и поиска неисправностей интуитивными способами. Применение формальных методов, кроме того, позволяет автоматизировать процессы построения алгоритмов диагноза при помощи вычислительных средств.

Эффективность процессов диагноза определяется не только качеством алгоритмов, но и в не меньшей степени качеством средств диагноза. Последние могут быть аппаратурными или программными, внешними или встроенными, ручными, автоматизированными или автоматическими, специализированными или универсальными.

Сбор данных о состоянии объекта диагностики требует применения надежно работающих внешних и встроенных аппаратурных средств диагноза, обеспечивающих высокую точность измерений и автоматическую обработку и документирование данных. При этом будет гарантирована достоверность результатов диагноза, сведено к минимуму влияние субъективных факторов и упрощена статистическая обработка результатов.

В настоящее время в большинстве случаев проектирование сложных объектов ведется без должного учета того, как они будут проверяться и налаживаться в условиях производства или ремонта,

как будут организованы проверка работоспособности, правильности функционирования и поиск неисправностей в условиях их эксплуатации или хранения. Недооценка важности своевременной (на этапе проектирования объектов) и глубокой проработки вопросов организации эффективных процедур диагноза, в том числе автоматизации поиска неисправностей сложных объектов, ведет к непроизводительным материальным затратам, затратам времени и квалифицированной рабочей силы при наладке, профилактике и ремонте.

Среди объективных причин такого положения следует назвать недостаточное развитие теории и методов технической диагностики, слабую проработку принципов построения технических средств диагноза, а также отсутствие налаженного производства таких средств. Существенным является также психологический фактор, состоящий в том, что почти все разработчики считают творческим, созидательным делом непосредственно разработку объектов (изделий, устройств, агрегатов, систем), выполняющих заданные им функции, и не придают должного значения вопросам организации наладки, профилактики и ремонта проектируемых объектов. Усугубляется это обстоятельство тем, что обязательная проработка этих вопросов пока не всегда регламентируется официальными требованиями к проектам новых объектов. Все это приводит к тому, что часто сложные объекты оказываются без хорошо организованных систем проверки правильности их функционирования, не говоря уже о системах поиска неисправностей в условиях применения по назначению. Задачи проверки исправности, проверки работоспособности и поиска неисправностей в условиях изготовления, профилактики, ремонта и хранения во многих случаях вынужденно решаются после того, как объект уже спроектирован или даже изготовлен. При существующем положении заботы по созданию средств диагноза в значительной степени ложатся на изготовителей, эксплуатационников и ремонтников. Создаваемые ими средства, как правило, являются специализированными со всеми присущими «приставной автоматике» недостатками. Затраты на разработку и создание таких средств велики, а эффективность применения низкая.

Многие из указанных недостатков будут исключены, если задачи диагноза решать на этапе проектирования объектов. Иначе говоря, разработку систем и средств диагноза следует считать такой же обязательной и важной частью проекта нового объекта, как и разработку самого объекта или других его систем и средств управления.

Появление компьютеров радикально изменило подход к построению, как систем диагноза, так и методов и алгоритмов диагностирования. Стало возможным хранение данных о результатах диагностирования на любом интервале времени работы оборудования, создавать сложные модели, упрощающие процедуры диагностирования, выполнять более широкий круг задач, охватывающий и прогностику и генезис.

Установление диагноза можно рассматривать как специфический процесс управления, целью которого является определение технического состояния объекта. Это хорошо согласуется с современным пониманием управления как процесса осуществления целенаправленных управляющих воздействий на управляемый объект, а кроме того, четко определяет предмет исследований и задачи технической диагностики с позиций общей теории управления и контроля.

Еще одной стороной описываемого процесса является получение информации. Обычно предполагается два подхода. В первом случае используются специально устанавливаемые датчики ориентирование только лишь на выполнение процедур диагностирования. Во втором, используются датчики, являющиеся неотъемлемой частью диагностируемого объекта, и других датчиков в системе не предусмотрено. При этом не обязательно, что эти датчики дадут тот объем информации, который закладывается в методе диагностирования. В этом случае приходится использовать методы восстановление необходимой информации, прибегая к помощи теории идентификации.

Одним из аспектов повышения качества технических систем является разработка новых подходов, учитывающих особенности современного оборудования и технологии. Появившиеся в последнее время информационные системы, использующие в своем составе компьютеры, накладывают отпечаток на решение задач по обеспечению качества. Одним из методов обеспечения качества является своевременное предсказание состояния технических систем, решаемое на основе методов компьютерной диагностики. Эти методы идут в ногу с развитием интеллектуальных систем (рис.5), и поэтому изучение и использование нейронных сетей, нечеткой логики при решении задач диагностики является весьма эффективными.

Любое оборудование, с помощью которого осуществляется технологический процесс, недолговечно и небезотказно. Поэтому необходимо предопределять неисправности своевременно для того чтобы можно было продлить жизненный цикл техники или заменить ее без поломок и аварий, которые нередко могут привести к большим экономическим затратам и потерям. С этой задачей легко справиться при непрерывном или систематическом (тестовом) контроле и/или диагностировании электромеханических систем, обеспечивая надежность и качество работы. Под надежностью понимается способность системы выполнять заданные функции, сохраняя во времени значения установленных эксплуатационных показателей, определенных в заданных пределах при заданных условиях эксплуатации.

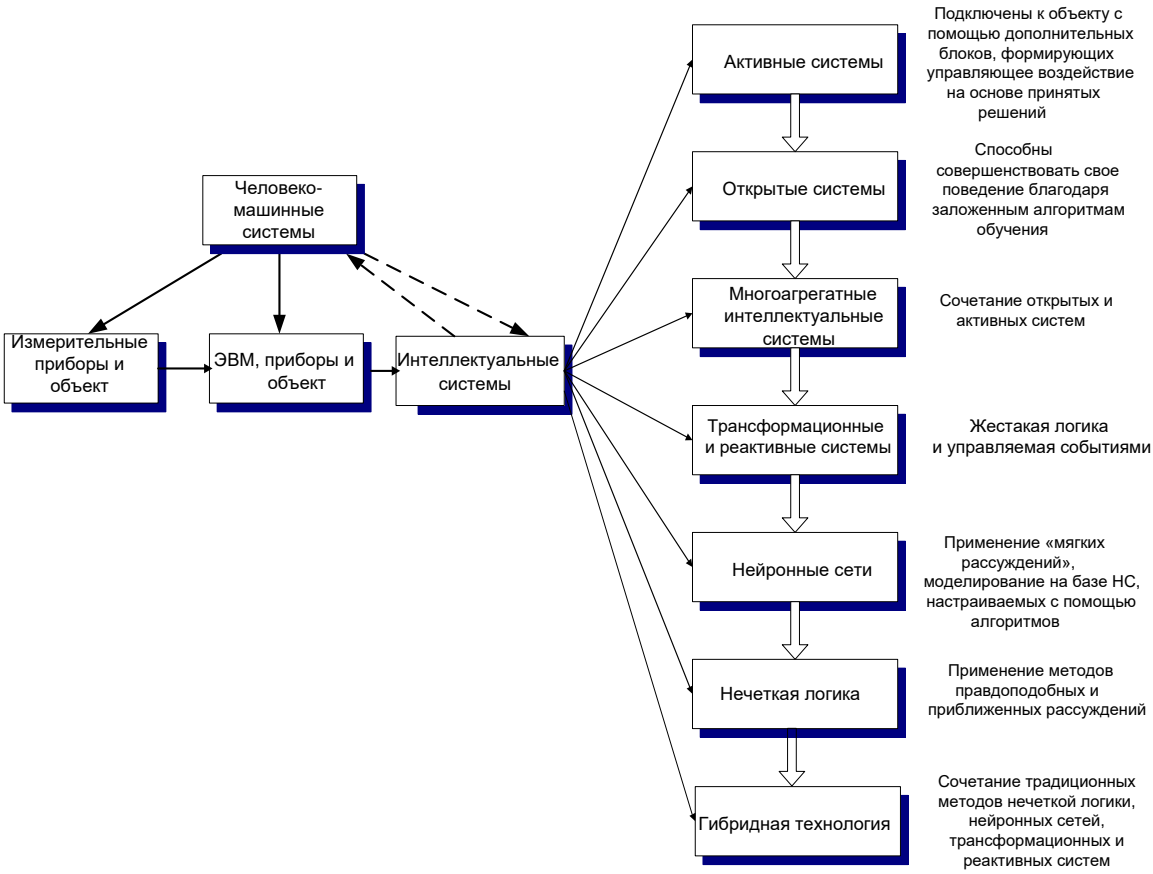


Рис. 5. Эволюция интеллектуальных систем

При диагностировании электромеханических систем информация с различных контрольных точек поступает в устройство диагно-

стики, где происходит анализ полученной информации и прогнозирование возможности дальнейшей работы. Чем больше контрольных точек, тем более достоверную и объективную информацию можно получить о процессах, происходящих внутри объекта. Но не всегда вся информация доступна. Чаще всего требуется наблюдение изменения отдельных параметров, по которым определяется неисправность объекта или готовность к дальнейшей работе. Например, для электропривода (ЭП) наиболее распространенными параметрами диагностирования, доступными для непосредственного измерения, являются скорость, положение и ток, по которым можно судить о состоянии данной ЭМС.

Наибольшей популярностью при решении задач диагностики пользуются интеллектуальные системы (ИС), основанные на знаниях людей – экспертов (например, экспертные системы). При помощи различных приемов и правил, система выстраивает алгоритм обнаружения неисправностей, и сама принимает решения в той или иной ситуации. Важными особенностями ИС являются способность анализа и объяснения своих действий и знаний, приобретение новых знаний и изменение в соответствии с ними своего поведения, обеспечение взаимосвязи между оператором и ЭВМ. На смену многочисленным измерительным приборам пришли новые технологии и компьютерные системы, развитием которых является появление нечеткой логики и нейронных сетей (НС), а также их сочетание, что следует из эволюции ИС.

Глава 1. АРХИТЕКТУРА ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Архитектура системы диагностики (СД) неисправностей ЭМС представляет собой две подсистемы: подсистему приема и обработки информации и подсистему интерпретации полученной информации о состоянии объекта диагностирования. Первая осуществляет прием данных по состоянию объекта и их последующую обработку (распределение данных, оценку переменных и их отображение). Вторая подсистема производит распознавание с помощью искусственной нейронной сети (ИНС) неисправностей ЭМС, а нечеткий контроллер дает рекомендации по реализации дальнейших действий (реализует выводы).

Рис. 1.1. Архитектура интеллектуальной системы

В настоящее время ИС используются при решении задач различных типов: принятие решений в условиях неопределенности (неполноты), интерпретация символов и сигналов, предсказание, диагностика, конструирование, планирование, управление, контроль и др.

Основными этапами проектирования модели НС для оценки технического состояния ЭМС являются:

1) определение логических условий, выделяющих анализируемую выборку в зарегистрированных переходных процессах;

2) задание символьных нечетких переменных для векторов входа и выхода моделируемого объекта;

3) формирование базы «нечетких» правил, описывающей все возможные эксплуатационные режимы работы ЭМС;

4) создание на основании зарегистрированных выборок реальных переходных процессов исправной работы тренировочных, тестирующих и проверочных данных для обучения;

5) обучение адаптивной нейро-нечеткой сети.

Такое представление позволяет сформировать свою систему диагностики на базе нейронной сети, определив все этапы проектирования. То есть создать базу логических условий и правил для системы, спроектировать ее и обучить.

Любая система опирается на исследование объекта, получение и преобразование сигналов, поступающих от объекта (интерфейс), нахождение и устранение неисправностей, и оформление предположений о дальнейшей работе объекта.

На вход ОД подается сигнал, с помощью которого на выходе выдается достоверная информация о работе всей системы в целом. Полученный сигнал через интерфейс преобразуется и поступает на вход нейросети, где согласно алгоритму работы НС исследуется, и выводы о состоянии системы отображаются на экран.

Архитектура интеллектуальных систем автоматического управления представлена на рис.1.2. Отличие от предыдущей структуры только во внешнем интерфейсе, а интеллектуальная часть строится примерно по одному принципу. Ниже будут приведены структуры реализации приводов, как в нечетком базисе, так и с использованием нейронных сетей.

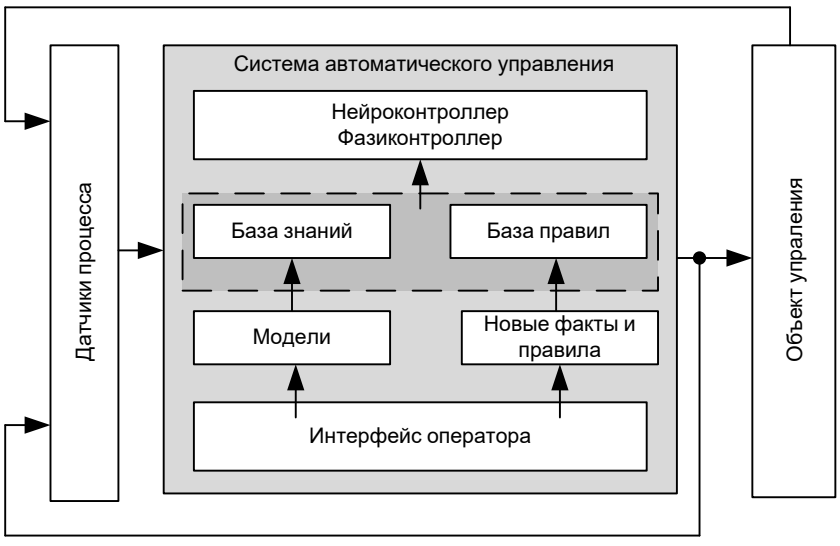


Рис. 1.2. Архитектура системы автоматического управления

1.1. Применение нечетких контроллеров и искусственных нейронных сетей в управлении электромеханическими системами

В истории развития теории автоматического управления четко выделяются три этапа. По классификации, первым был этап классической детерминированной теории автоматического регулирования, охвативший период времени с конца XIX по 40-е годы XX века. В этот период основными задачами управления были задача устойчивости и задача о качестве переходных процессов. Второй этап теории управления начался в 40—50-х годах нашего века и длился примерно до середины 70-х годов. Это этап классической стохастической теории автоматического регулирования. Он характеризуется новой постановкой основной задачи теории управления: учесть случайные возмущения, действующие на систему, и обеспечить хорошую работу в условиях постоянно действующих помех.

Около 25-и лет назад в развитии теории автоматического управления начался новый этап, связанный с адаптивной постановкой основной задачи управления. Ее особенность состоит в отсутствии изначальных знаний о математической модели объекта управления, будь то дифференциальные уравнения или плотности вероятностей случайных внешних воздействий. Объект—это черный ящик, подвергающийся неизвестным случайным воздействиям. Нам доступны только его входы и выходы. Цель системы управления (САУ) состоит в том, чтобы уже в процессе функционирования определить закон регулирования, обеспечивающий оптимальное поведение объекта. Для решения этой задачи в дополнение к основному контуру в систему управления вводится контур адаптации (рис.1.3).

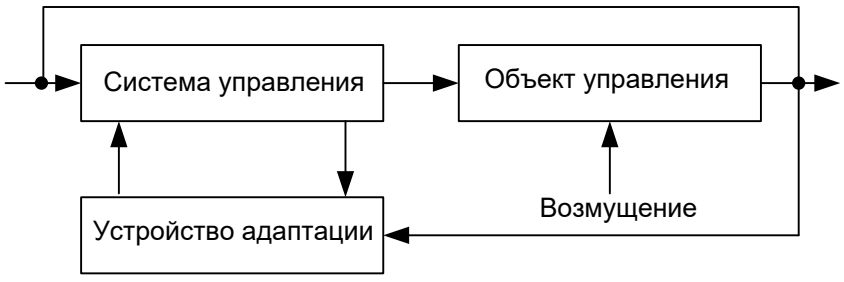


Рис. 1.3. Общая схема адаптивной системы управления

С самого начала третьего этапа огромное внимание уделялось адаптивному управлению линейными стационарными объектами с неиз-

вестными параметрами (например, широко используемые методики, опирающиеся на построение наблюдателей). В рамках этого подхода в 80-х годах началось использование ИНС для решения задач управления. Полученные результаты показали, что ИНС представляют собой на просто новую методику в теории автоматического управления, а целую парадигму. Для нового направления в теории управления Вербосом было введено отдельное название — нейроуправление (neurocontrol).

О цельности нейроуправления говорит то, что в нем, благодаря описанным выше свойствам ИНС, общим для различных нелинейных динамических объектов образом решаются задачи идентификации, синтеза систем управления, их анализа и аппаратной реализации. Результаты, полученные с применением ИНС в рамках адаптивной постановки основной задачи теории управления, легко могут использоваться и классические подходы.

На рис.1.4 представлена некоторая динамическая система (объект управления). В адаптивной постановке объект управления описывается своей функциональной моделью: $F\{\mathbf{u}(t), \mathbf{y}(t)\}$, связывающей вектор входных воздействий $\mathbf{u}(t)$ с вектором выходных сигналов $\mathbf{y}(t)$. Такое описание является портретом динамического поведения объекта и отражает только его функциональные связи.

Рис.1.4 Динамическая система

Введем векторы $\mathbf{u}(t)=(u_1(t),u_2(t),...,u_p(t))^{\mathrm{T}}$, $\mathbf{x}(t)=(x_1(t),x_2(t),...,x_n(t))^{\mathrm{T}}$ и $\mathbf{y}(t)=(y_1(t),y_2(t),...,y_m(t))^{\mathrm{T}}$. Здесь $u_i(t)$ и $y_i(t)$ — вход и выход системы, соответственно, $x_i(t)$ — переменная состояния системы, p — размерность входного пространства, m — размерность выходного пространства, а n — порядок системы.

Классически, динамика такой системы описывается системой дифференциальных уравнений:

$$\frac{d\mathbf{x}(\mathrm{t})}{dt} = \boldsymbol{\Phi}\big(\mathbf{x}(\mathrm{t}), \mathbf{u}(\mathrm{t})\big)$$

$$\mathbf{y}(\mathrm{t})=F(\mathbf{x}(\mathrm{t})).$$

Здесь вектора функции $\Phi = (\phi_1, \phi_2, ..., \phi_\nu)$ и $F = (f_1, f_2, ..., f_m)$ — статические нелинейные преобразования. Φ: $\Re^n \times \Re^n \to \Re^n$, F: $\Re^n \to \Re^m$. Вектор $\mathbf{x}(t)$ описывает состояние системы в момент времени t. Он определяется состоянием системы в начальный момент $t_0 < t$ и входом $\mathbf{u}$, определенном на интервале $[t_0, t)$. Выход системы $\mathbf{y}(t)$ полностью определяется состоянием системы $\mathbf{x}$ в момент t.

Одно из ключевых направлений на пути создания будущих производств, роботов, станков с числовым управлением в машиностроении привлекло научное и инженерное сообщество к использованию в реализации систем автоматического управления методов искусственного интеллекта.

С одной стороны, традиционные методы построения систем не приводят к удовлетворительным результатам, когда исходное описание подлежащей решению проблемы является неполным. Стремление получить исчерпывающую информацию для построения точной математической модели системы приводят к потере времени и средств, в силу сложности структур реальных систем управления.

В 1965 Заде опубликовал основополагающую работу по теории нечётких множеств, в которой изложил математический аппарат теории, а в 1973 представил теорию нечёткой логики, позднее — теорию мягких вычислений (softcomputing), а также — теорию вербальных вычислений и представлений (computing with word sandperceptions).

Это послужило развитию нового направления в решении многих задач, в различных отраслях знаний. Одновременно это стало и началом использования теории нечеткой логики в реализации технических систем.

Основанием для такого утверждения является доказательство Вангом в 1992г. теоремы, что существует нечеткая система, формирующая выходную функцию, такую, что верхняя граница разности исходной функции и выходной меньше или равна некоторой произвольной постоянной. В 1993г. Б.Коско была доказана FAT (Fuzzy Approximation Theorem) теорема, суть которой состоит в том, что любая математическая система может быть аппроксимирована системой на основе нечеткой логики. И, наконец, в 1995г. Кастро показал, что логический контроллер Мамдани при определенных условиях является универсальным аппроксиматором. Эти работы послужили активному применению нечеткой логики в системах автоматического управления.

Применение теории нечетких множеств в проектировании систем автоматического управления объясняется тем, что нечеткие системы разрабатываются быстрее, они получаются проще четких аналогов. Экспертные знания легко внедряются в нечеткие системы, что позволяет быстро создавать системы с понятными для инженера алгоритмами функционирования. Разработанные в последнее десятилетие методы обучения позволяют настроить нечеткую систему для обеспечения требуемого качества функционирования.

Использование нечеткой логики в системах автоматического управления носит специфику, связанную с принципиальными особенностями организационного характера, реализации структуры и алгоритмов функционирования.

Однако, дальнейшее развитие интеллектуальных систем промышленной автоматизации сталкивается с определенными затруднениями. Так, в настоящее время в отношении нечеткой логики в системах автоматического управления имеет место не устоявшаяся терминология, а трактование отдельных моментов затрудняют понимание решаемой задачи. Так же не сложилась методология построения нечетких САУ. Нет четкого определения интеллектуальных САУ, уровня их «интеллекта», отсутствует общепринятая классификация. Число публикаций в этом секторе незначительно. Вместе с тем, внедрение интеллектуальных САУ является неизбежным шагом.

Одно из центральных понятий в системах управления в настоящее время становится понятие искусственного интеллекта. Существует множество его определений, но универсального более или менее приемлемого в технических системах так и нет. Нет и методик определяющих уровень интеллекта той или иной системы управления.

Системы с искусственным интеллектом могут обладать большой памятью, быстродействием, способностью к самообучению и самоорганизации, реализуют выводы, принимают решения, формируют экспертные оценки и являются мощным инструментом в построении САУ. И что особенно важно такие системы могут успешно функционировать в условиях частичной или полной неопределенности [22].

Нечеткую логику так же принято считать разделом искусственного интеллекта.

Применение нечеткой логики в САУ сопряжено с определением места НК в структуре САУ, методиках проектирования и снятие неопределенности в использовании терминологии.

Для оценки уровня искусственного интеллекта САУ необходимо определить: что такое искусственный интеллект, интеллектуальная система и ввести шкалу оценок уровня интеллекта.

Определение 1. Искусственный интеллект–это сбор и обработка информации, извлечение знаний, их представление, запоминание и использование.

Если используемые в определении термины в целом понятны то термин «представление» нуждается в некотором пояснении.

Представление знаний является наиболее важной областью исследований по искусственному интеллекту. Знания могут иметь различную форму описаний объектов, например, в виде причинно следственных отношений или функциональных связей. Наличие адекватных знаний и способность их эффективно использовать означают, что система в определенной мере способна реализовать выводы и принимать решения. Обычно для представления знаний используют нейронные сети (Neural Networks), нечеткую логику (Fuzzy Logic), мягкие вычисления (Soft Computing), генетические алгоритмы (Artificial Neural Networks) эволюционные вычисления (Evolutionary Computing), вероятностные рассуждения (Probabilistic Reasoning), вербальные вычисления и представления (Computing with word sand perceptions). Однако все способы представления знаний имеют какие либо ограничения, связанные с особенностями области применения.

В этой ситуации для достижения поставленной цели необходимо найти способ выражения закономерностей характерных для любой системы. В том числе и для САУ. В этом и состоит суть проблемы представления знаний.

Поэтому важной проблемой, которая еще долго будет актуальной, является разработка теории или метода представления знаний произвольного вида.

Определение 2. Интеллектуальная система – система, состоящая из интеллектуального управляющего объекта и объекта управления**.**

Важными особенностями интеллектуальных систем являются: способность анализа и объяснения своих действий и знаний, приобретение новых знаний и изменение их в соответствии с ними своего поведения, обеспечивая взаимосвязи между реализуемым процессом и целью управления. В техническом плане интеллектуальные САУ должны обладать большой памятью, высоким быстродействием, спо-

собностью к самообучению и самоорганизации, реализовать выводы и принимать решения.

В публикациях по нечеткой логике встречаются понятия система управления и система автоматического управления, причем понятия смешиваются, что существенно затрудняют понимание решаемой задачи. В связи с этим попытаемся разграничить эти понятии.

Определение 3. Система управления это совокупность функционально взаимосвязанных и взаимодействующих частей, совместно выполняющих определенную задачу и обладающую свойствами, не содержащейся в частях системы в отдельности и обеспечивающим управление каким-либо объектом или процессом.

Примером таковых являются системы программного управления.

Определение 4. Система автоматического управления это совокупность объектов управления и управляющего устройства, взаимодействующих между собой в соответствии с алгоритмом управления**.**

В определениях использовались устоявшиеся термины, используемые в теории автоматического управления.

1.2. Структуры нечетких систем автоматического управления

Из сказанного следует, что при построение нечеткого контроллера необходимо следовать известным принципам. При этом, существует специфика применения нечеткой логики в САУ. В подавляющем большинстве работ нечеткий контроллер (НК) рассматривается как система управления, внешняя по отношению к управляющему устройству. Проще говоря, он используется для формирования алгоритма дополнительного управления. В немногих работах НК становится составной частью системы управления и используется как регулятор в прямом канале.

В рассматриваемом случае рассматриваем НК как универсальное средство, обеспечивающее оптимальные характеристики систем, адаптивных регуляторов, экстремальных и регуляторов с особами видами движения и т.п.

Проведенный анализ дал возможность провести некоторую систематизацию использования НК в САУ.

Предлагается следующее представление нечетких систем автоматического управления по степени внедрения НК в структуру САУ.

1. САУ с нечетким контроллером. Система управления в которой в качестве регулятора(ов) используется *нечеткий контроллер,* вырабатывающий управляющее воздействие на основе нечеткого вывода. Из-за заранее введенных знаний в виде базы правил системы нечеткого вывода выполняет определенные функции и не может модифицироваться самостоятельно (рис 1a,b).

В некотором смысле это “жесткий” контроллер - однажды сформированные правила логического вывода, термы и база знаний остаются неизменными. Контроллер может быть установлен в любом месте структуры САУ в прямом канале (рис.1.5a), в цепи обратной связи (рис.1.5b).

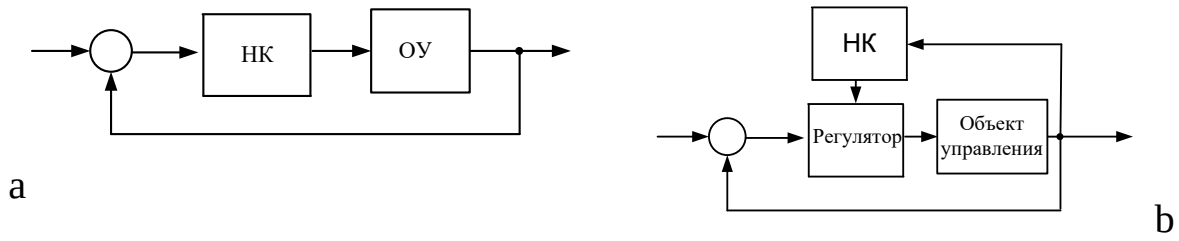


Рис.1.5. САУ с нечетким контроллером

2. Комплексированные нечеткие САУ. Система управления, в контуре которой используется классический регулятор и нечеткий контроллер корректирующий параметры классического регулятора на основе алгоритмов нечеткого вывода (рис.1b). В данном случае входными переменными являются переменные состояния объекта управления, а выходными переменными – параметры закона управления. Система становится более гибкой к изменению свойств объекта и имеется возможность модифицировать управляющее воздействие в соответствии с правилами, основанными на знаниях.

3. Иерархические нечеткие САУ. Система управления в качестве регулятора(ов) используется нечеткий контроллер(ы) с возможностью модификации базы правил при помощи одного из методов нечеткого вывода, формируемого одним из НК (рис.1.6).

При изменении среды функционирования верхний уровень нечеткой САУ выполняет реконфигурацию системы нечеткого вывода для последующих уровней.

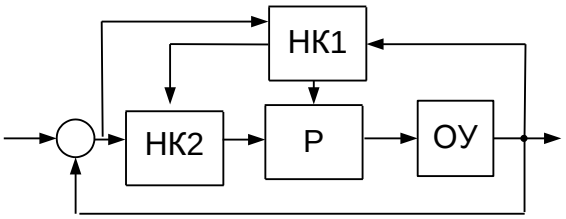


Рис. 1.6. Иерархические нечеткие САУ

4. Гибридные нечеткие САУ (Рис. 1.7.). Система управления в качестве регулятора(ов) применяются нейронные сети и нечеткие контроллеры, в которых используется обучение или самообучение нейронной сети с реализацией выводов и принятие решений в нечетком контроллере, обеспечивающем особые виды движения, оптимальные, адаптивные или иные условия функционирования САУ.

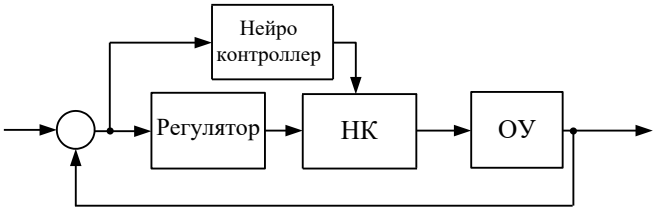


Рис. 1.7. Гибридные нечеткие САУ

5. Каскадные нечеткие САУ в которых контроллеры включаются последовательно, параллельно или последовательно параллельные либо параллельно последовательные.

6. Сетевые нечеткие САУ (Рис.1.8.) содержащие набор НК связанных между собой на условиях причинно следственных отношений, в которых связи обеспечивают обмен информацией таким образом, чтобы каждый НК имел возможность обмениваться с каждым, выполняя решение задачи управления каким-либо образом.

Частные выводы в каждом НК передаются в логическое устройство вывода и заключения (ЛВЗ), где формируется окончательное принимаемое решение.

7. Комбинированные нечеткие САУ такие, в которых в различных сочетаниях используются все выше перечисленные.

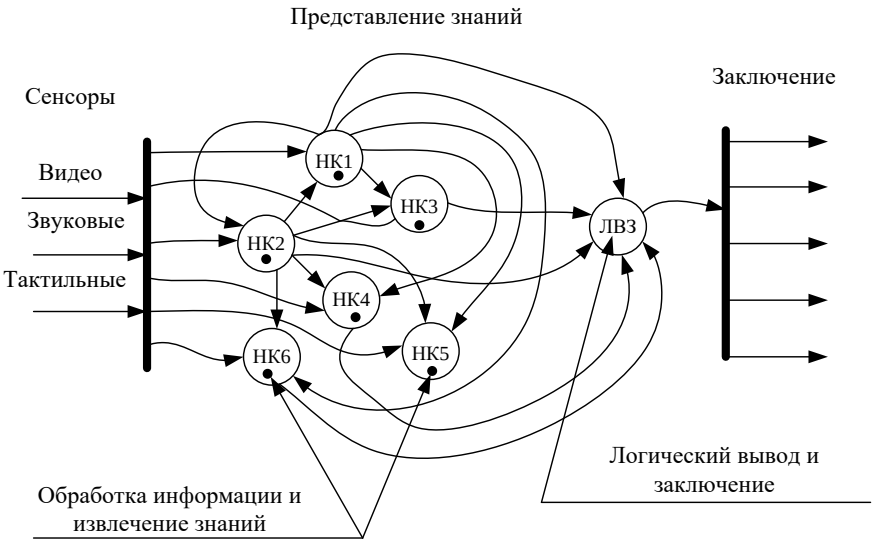


Рис.1.8. Структура модуля сетевой нечеткой САУ

8. Нечеткая САУ (fullfuzzy) (Рис. 1.9.) – система автоматического управления, которая не содержит других решений кроме нечетких контроллеров, связанных между собой каким-либо способом, либо представляющая собой единственный контроллер с множеством входов и выходов..

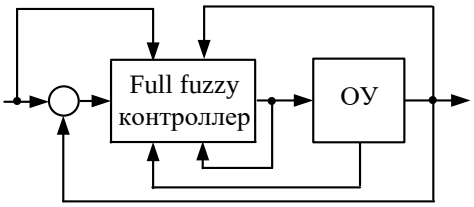


Рис. 1.9. Структура нечеткого full fuzzy модуля

При этом НК должен обладать способностью к самоорганизации и самообучению, изменяя вид функций принадлежности термов и интервалы входных и выходных множеств; (обладать свойством модификации и настраиваться на решение текущей задачи), изменять базу знаний полностью или частично, пополнять или сокращать ее; реализовывать логический вывод и способ дефадзификации.

Особый интерес представляют принципы адаптивного управления. Обычно адаптивная система содержит в качестве «ядра» схему, реализующую один из фундаментальных принципов управления, а *контур адаптации* пристраивают к ней как вторичный, осуществляющий коррекцию параметров. Контур адаптации, обычно состоит из устройства измерения ИУ, вычисления ВУ и управления УУ (Рис.1.10а).

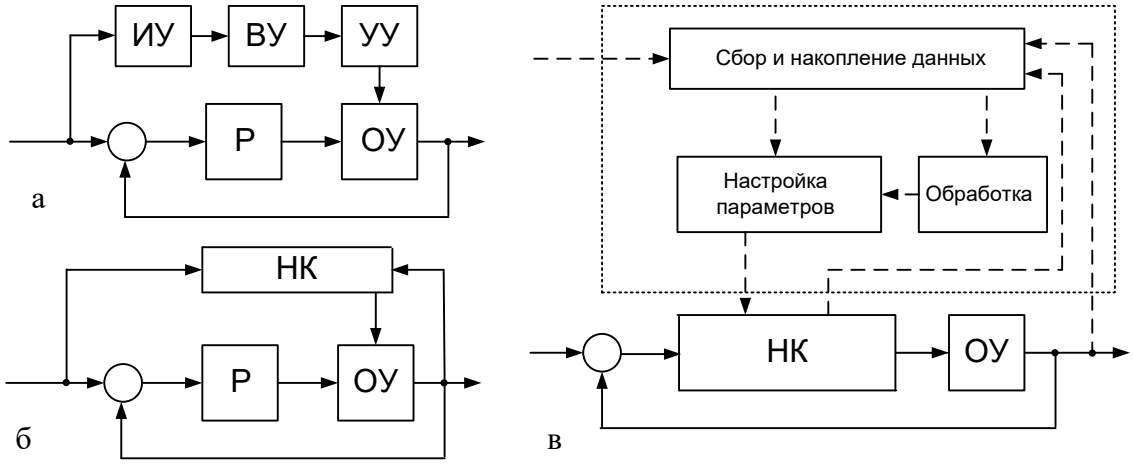


Рис. 1.10. Структуры адаптивных САУ:
а- классическая адаптивная структура без обратной связи, б – адаптивная структура с нечетким контроллером, в - адаптивная структура с адаптивным регулятором в прямом канале: ИУ – измерительное устройство, ВУ- вычислительное устройство, УУ – устройство управления, Р – регулятор, ОУ – объект управления, НК – нечеткий контроллер.

Контур адаптации, так же может быть реализован в виде НК (Рис.1.10б) и очевидно иметь устройства измерения ИУ, вычисления ВУ и управления УУ (Рис.1.10в).

Вместе с тем, так же, как и САУ НК может быть реализован в виде адаптивного устройства. Таким образом, адаптивная САУ реализуется на основе адаптивного НК.

Но в это случае адаптивный НК решает задачу адаптации к изменяющимися входными данными, изменяя свои параметры, либо структуру, либо то и другое с целью достижения оптимального логического вывода для реализации адаптивного управления САУ. В то же время НК может быть установлен в прямом тракте САУ (Рис. 1.10в) и выполнять функции адаптивного регулятора.

Данная классификация нечетких систем не является окончательной, поскольку в настоящее время ведутся работы по дальнейшему развитию интеллектуальности нечетких САУ. Эти цели могут быть достигнуты путем комбинации различных подходов к построению интеллектуальных систем в составе иерархических интеллектуальных контроллеров, органично сочетающих принципы и методы нечеткого вывода, ситуационного управления, инженерии знаний, нейронных сетей и эволюционного моделирования, использующие мягкие вычисления и большие данные.

1.3. Структуры нейронных систем автоматического управления

Использование нейронных сетей можно представить, так же как и в задачах с нечетким контроллером. НС может быть установлена в прямом канале, в канале обратной связи, может реализовывать отдельные устройства, так например, регулятор скорости, или построить систему управления приводом полностью на основе нейронных сетей. В построении систем могут использоваться комбинированные системы, в которых используются в различных сочетаниях нейронные сети, нечеткие контроллеры и традиционно реализованные устройства. В адаптивных системах нейронная сеть может решать задачи наблюдателей, идентификаторов, экстремальных и оптимальных регуляторов. Так же нейронные сети могут использоваться в задачах диагностики.

Задача идентификации является фундаментальной в теории систем и, в частности, теории автоматического управления. Целью идентификации является построение модели идентификации $\hat{P}$ (см. рис.1.11), аппроксимирующей объект P: $\|\hat{y} - y\| = \|\hat{P}(u) - P(u)\| \le \varepsilon$, $u \in U$, для некоторого заданного $\varepsilon > 0$ и определенной нормы $\|.\|$. Здесь $\hat{y} = \hat{P}(u)$ — выход идентификационной модели, U — допустимое множество управления. Причем, как для статической, так и для динамической систем оператор P неявно определен парами сигналов вход-выход $\{\mathbf{u},\mathbf{y}\}$.

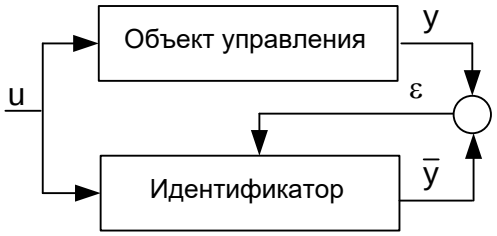


Рис.1.11. Идентификация объекта управления

Благодаря своим универсальным аппроксимирующим свойствам, ИНС представляют собой мощный инструмент для решения задачи идентификации нелинейных статических и динамических объектов управления. Основанные на ИНС дискретные идентификационные модели называются нейроэмуляторами (НЭ) или предикторами. В общем виде они описываются следующим нелинейным уравнением:

$\widehat{y}(k+1) = NN(\hat{\mathbf{y}}(k),(\hat{\mathbf{y}}\text{ (k-1)},\ldots,\hat{\mathbf{y}}\text{ (k-l}_1\text{)}, \mathbf{u}\text{(k)}, \mathbf{u}\text{(k-1)},\ldots, \mathbf{u}\text{(k-l}_2\text{))}$,
где $NN(\ )$ — преобразование вход-выход, выполняемое ИНС, l_1 — глубина задержки обратной связи по выходу НЭ, l_2 — глубина задержки по входу НЭ.

Одношаговый предиктор (рис. 1.12) осуществляет предсказание выходного вектора объекта по его предыстории на один шаг вперед:

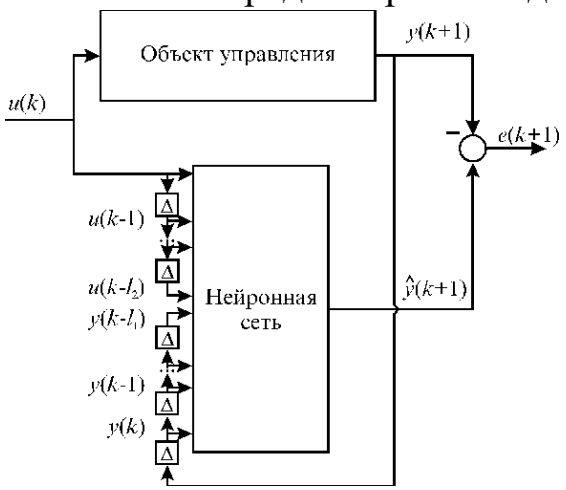


Рис.1.12. Одношаговый предиктор

ИНС могут решать задачи управления различными объектами с использованием адаптивного подхода. Для этой цели можно использовать НС с обучением. Режим обучения требует создание обучающей базы данных и сформированных целей обучения. Последние основаны на классических подходах теории адаптивных систем. Настройка таких контроллеров определяется учителем, являющимся специалистом или экспертом в обозначенной области (рис. 1.13).Чем большим опытом обладает эксперт и чем больше обучающих данных, тем точнее будет решение задачи. Конечно и тип нейронной сети должен отвечать характеру решаемой задачи.

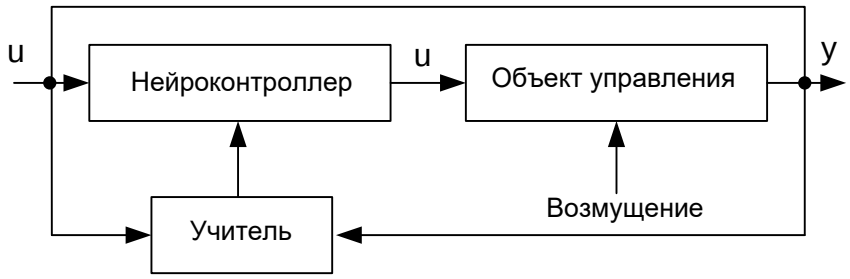


Рис.1.13. Схема системы управления с обучаемым нейроконтроллером

Существует множество подходов к применению ИНС в качестве НК. Можно использовать ИНС, как регуляторы содержащие интегральную, дифференциальную и пропорциональную составляющие, т.е. ПИД-регуляторы (рис. 1.14). Такой тип регуляторов широко используется в регулирующей аппаратуре и прводной технике. Следовательно можно реализовать регулятор скорости, тока, экстремальный регулятор и др.

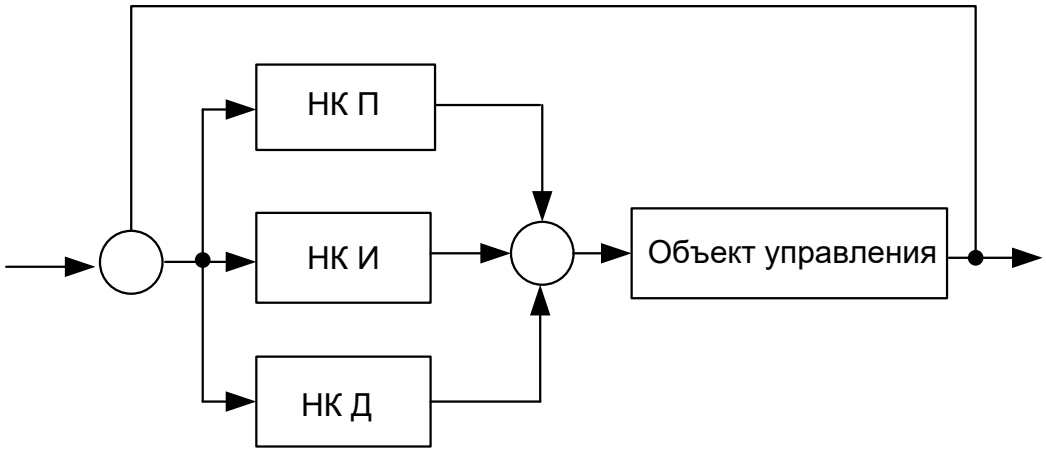


Рис.1.14. Реализация ПИД регулятора в прямом канале

В классе адаптивных систем широко используются модели объектов управления. Наличие модели позволяет более точно регулировать параметры движения. При этом возможно построение структур в двух вариантах. В первом случае повышение точности достигается использование модели объекта как устройство на основе выходного сигнала и сигнала с реального объекта производится поднастройка нейроконтроллера управления координатами привода. Во втором случае строится модель объекта на основе НС. Возможно использование комбинации этих двух подходов. (рис. 1.15). В этом случае объект будет отслеживать желаемую траекторию, определенную эталонной моделью. Необходимым условием функционирования САУ с эталонной моделью является правильный выбор обучающих входных воздействий $\mathbf{u}_r$, так как от вида сигнала и объема информации зависит точность настройки основной системы на эталонную модель.

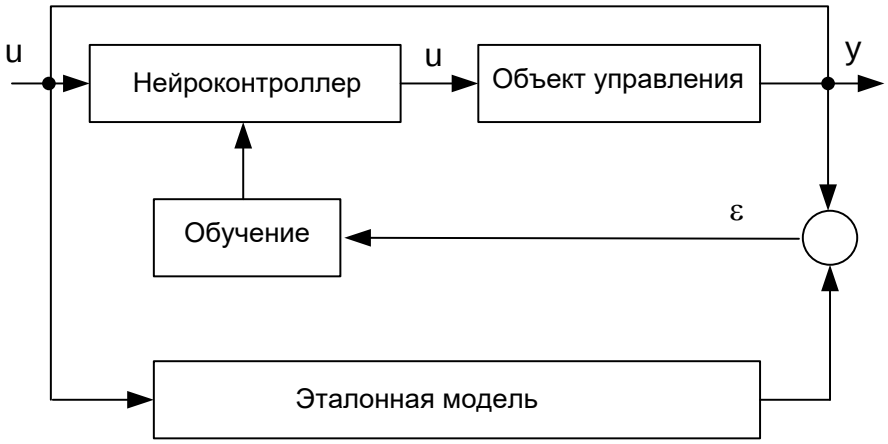


Рис.1.15. Нейросетевая система управления с эталонной моделью

Существуют такие схемы в которых для решения задачи управления необходимо использовать параметры измерение которых напрямую невозможно. В этом случае можно построить НС которая выполняет функцию наблюдателя, т.е. на основе измеряемых сигналов формировать необходимый сигнал управления. Это сигнал служит обучающими данными в процедуре поднастройки НС управления. (рис.1.16).

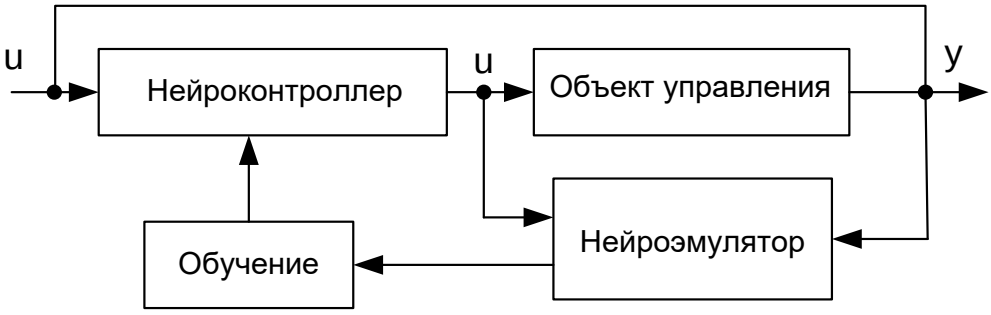


Рис.1.16. Нейросетевая система управления с нейроэмулятором

Приведенные структуры не исчерпывают всех возможностей построения систем управления на основе нейросетей. Все определяется целевой функцией, для какой задачи можно использовать ту или иную структурную схему. Кроме того необходимо решать как реализовывать процесс управления движением в структуре с НС: с учителем или использовать сети с самообучением.

Глава 2. ПОСТРОЕНИЕ МОДЕЛИ СИСТЕМЫ

В качестве объекта будем рассматривать привод постоянного тока с высокомоментным двигателем.

Двигатель постоянного тока (ДПТ) – машина постоянного тока, преобразующая электрическую энергию постоянного тока в механическую энергию.

ДПТ классифицируют по виду магнитной системы статора:

- с постоянными магнитами;
- с электромагнитами.

ДПТ с электромагнитной системой статора, в свою очередь, классифицируют по способу включения обмоток (электромагнитов) статора:

- с независимым включением обмоток;
- с последовательным включением обмоток;
- с параллельным включением обмоток;
- со смешанным включением обмоток.

Основные формулы, используемые при управлении ДПТ:

- крутящий момент, развиваемый двигателем, пропорционален току в обмотке якоря (ротора):

$M = k_m \cdot I$,

где I -ток в обмотке якоря,

k_m - коэффициент крутящего момента двигателя (зависит от конструкции двигателя).

- ток в обмотке ротора по закону Ома прямо пропорционален приложенному напряжению и обратно пропорционален сопротивлению обмотки ротора:

$I = \frac{U}{R}$,

где U- напряжение, приложенное к обмотке ротора,

R – сопротивление обмотки ротора.

- противоЭДС в обмотках якоря пропорциональна угловой частоте вращения ротора:

$E = k_e \cdot \omega$,

где k_e –коэффициент ЭДС двигателя,

ω – угловая скорость вращения ротора.

Следовательно, величиной крутящего момента можно управлять, меняя напряжение на якоре ДПТ. Такой способ применяют для относительно маломощных двигателей.

Для управления более мощными двигателями используют силовые преобразователи с ШИМ регулированием, когда изменяется величина напряжения в зависимости отдлительностиимпульса приложенного к якорю двигателя.

Двигатель постоянного тока с независимым возбуждением (без учета поперечной реакции якоря и влияния действия вихревых токов) при одномассовой расчетной схеме описывается следующей системой дифференциальных уравнений:

$$\left.\begin{aligned} u_я &= i_я \cdot R_я + L_я \cdot \frac{di_я}{dt} + e \\ u_в &= i_в \cdot R_в + L_в \cdot \frac{di_в}{dt} \\ e &= C_E \cdot \Phi \cdot \omega \\ M &= C_\text{E} \cdot \Phi \cdot i_я \\ M - M_C &= J \cdot \frac{d\omega}{dt} \end{aligned}\right\} \qquad (1)$$

где:

$u_я$– напряжение на якорной обмотке двигателя,

e– электродвижущая сила (ЭДС) якоря,

$i_я$– ток якоря,

$R_Я$– сопротивление якорной цепи, Ом;

$L_Я$– индуктивность якорной цепи, Гн;

$R_В$– сопротивление обмотки возбуждения, Ом;

$L_В$– индуктивность обмотки возбуждения, Гн;

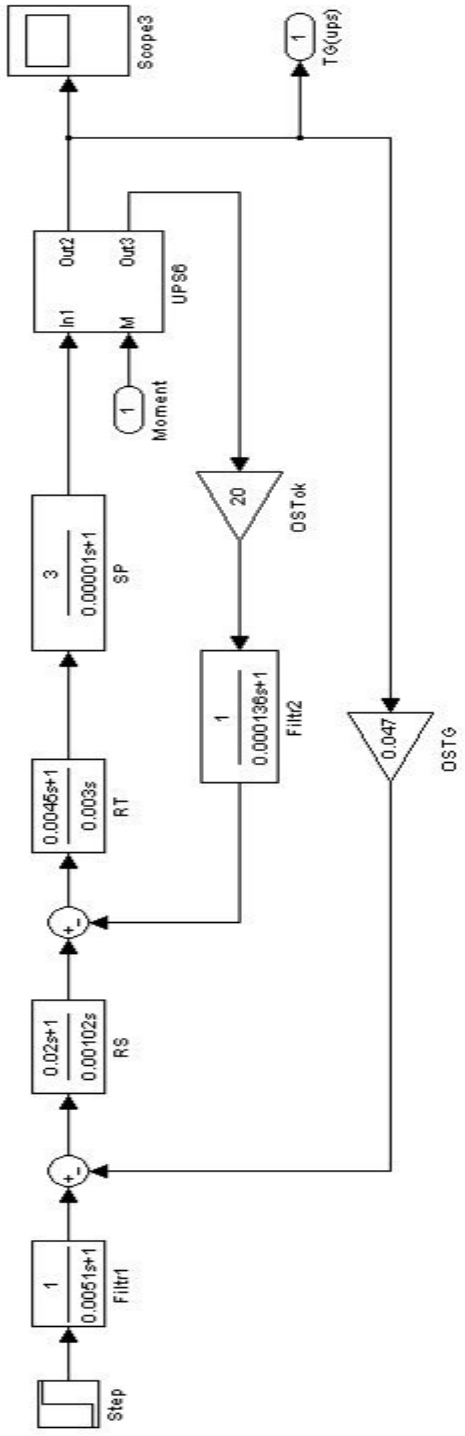


Рис.2.1. Схема для моделирования привода

$C_М$– конструктивный коэффициент двигателя;

$C_Е$– коэффициент ЭДС двигателя;

ω – скорость вращения вала двигателя,

Ф – полезный поток одного полюса, Вб;

J–момент инерции суммарный, приведенный к валу двигателя, кг·м^2;

M– электромагнитный момент двигателя, Н·м.

$M_С$ - момент нагрузки, приведенный к валу двигателя, Н·м.

Модель двигателя постоянного тока с независимым возбуждение полученная на основании уравнений (1) показана на рисунке 2.2.

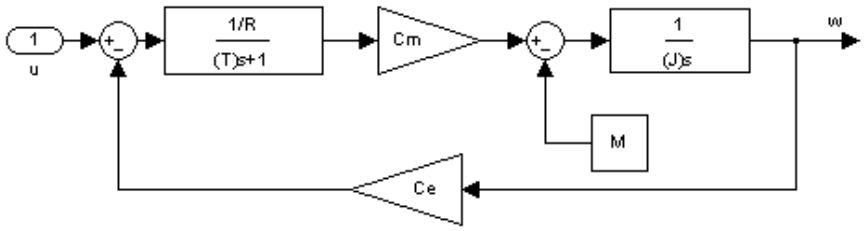


Рис.2.2. Структурная схема ДПТ с НВ

Далее создадим структурную схему для моделирования привода (см. рис.2.3) на основе реальной схемы привода ПРВП 02.

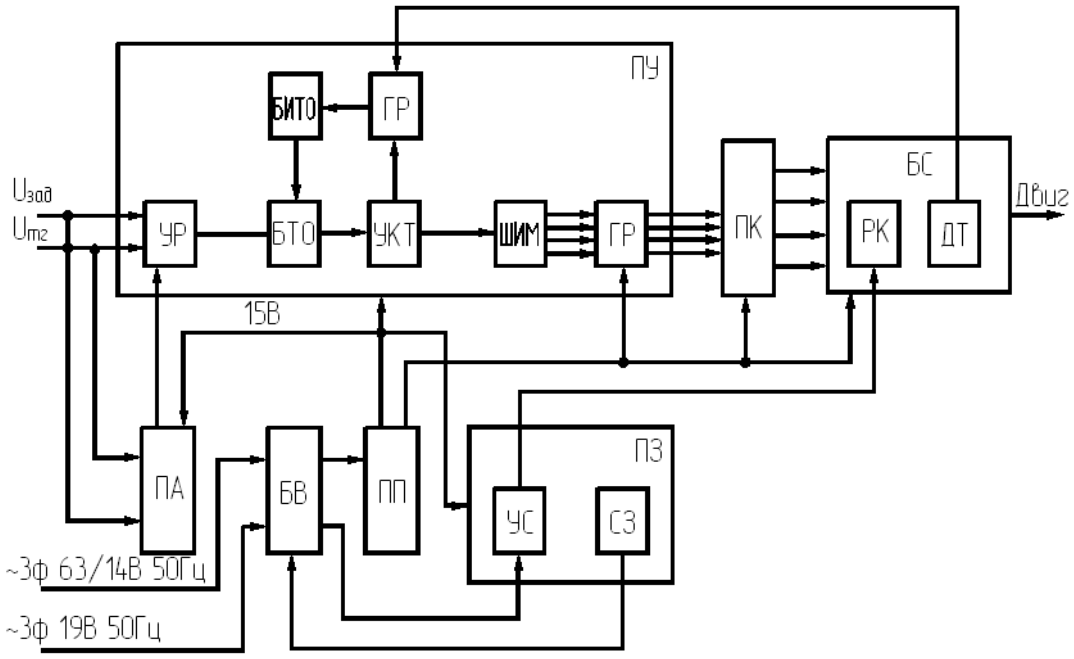


Рис.2.3. Структурная схема привода ПРВП 02

Регулятор скорости, используемый в приводе ПРВП 02 имеет электрическую принципиальную схему, приведенную на рисунке 2.4.

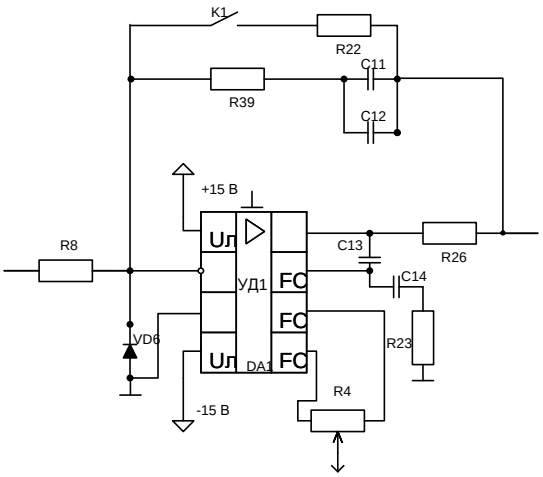


Рис.2.4. Электрическая принципиальная схема регулятора скорости

Найдем параметры регулятора скорости:

$$K = \frac{R_{\text{ВЫХ}}}{R_{\text{ВХ}}} = \frac{R_{39}}{R_{26} + R_{23}} = \frac{75}{43} = 1{,}744;$$

$$T = R * C = R_{39} * \left(C_{11} + C_{12}\right) = 75 \cdot 10^{3} * 0{,}2 \cdot 10^{-6} = 0{,}015\text{с};$$

$$W_{pc}(p) = \frac{(R39 * (C11 + C12))p + 1}{(R8 * (C11 + C12))p} = \frac{0{,}02p + 1}{0{,}00102p}.$$

Регулятор тока имеет электрическую принципиальную схему, изображенную на рисунке 2.5.

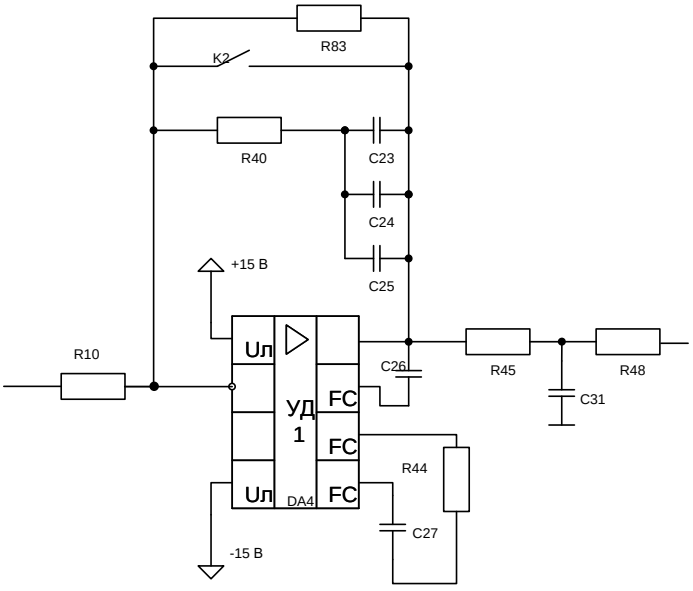


Рис.2.5. Регулятор тока

Найдем параметры регулятора тока:

$$W_{\text{РТ}}(p)=\frac{(R40*(C23+C24+C25))p+1}{(R31*(C23+C24+C25))p}=\frac{0{,}0045p+1}{0{,}003p};$$

$$K=\frac{R_{\text{ВЫХ}}}{R_{\text{ВХ}}};$$

$$R_{\text{ВЫХ}}=\frac{R_{83}*R_{40}}{R_{83}+R_{40}}=\frac{100*15}{100+15}=13{,}044 \text{ кОм};$$

$$K=\frac{R_{\text{ВЫХ}}}{R_{\text{ВХ}}}=\frac{13{,}044}{10}=1{,}304;$$

$$T=R*C=R_{40}*\left(C_{23}+C_{24}+C_{25}\right)=15\cdot10^{3}*0{,}3\cdot10^{-6}=0{,}0045c.$$

Рассчитаем коэффициент обратной связи по току:

$$K_{\text{ОСТОК}}=\frac{2000}{10}=200$$

Рассчитаем коэффициент обратной связи по скорости:

$$K_{\text{ОСсСск}}=\frac{2000}{94}=0{,}043$$

Рассчитаем RC-фильтр, стоящий на входе:

$$W_{filtr1}(p)=\frac{1}{(R3*C5)p+1}=\frac{1}{0{,}0051p+1}$$

Рассчитаем RC-фильтры, стоящие в цепи обратной связи по току:

$$W_{filtr2}(p)=\frac{1}{(R7*C9)p+1}=\frac{1}{0{,}000136p+1}.$$

Меняя значения элементов схемы, создавая отклонения от паспортных, рассчитать параметры звеньев по приведенным формулам и для них собрать экспериментальные данные.

Для того воспользуемся программой «Simulink».

Для запуска программы необходимо предварительно запустить пакет MATLAB. Основное окно пакета MATLAB показано на рисунке 2.6. Там же показана подсказка, появляющаяся в окне при наведении указателя мыши на ярлык «Simulink» в панели инструментов.

После открытия основного окна программы MATLAB нужно запустить программу «Simulink». Это можно сделать одним из трех способов:

- нажать кнопку(«Simulink»)на панели инструментов командного окна MATLAB.
- в командной строке главного окна MATLAB напечатать «Simulink» и нажать клавишу Enter на клавиатуре.

• выполнить команду «Open…» в меню «File» и открыть файл модели (mdl – файл).

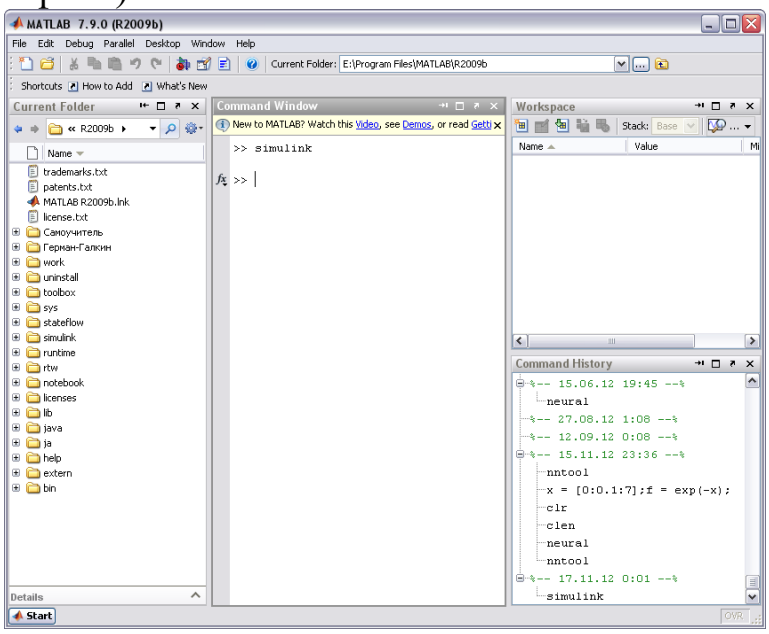


Рис.2.6. Основное окно программы MATLAB

Последний вариант удобно использовать для запуска уже готовой и отлаженной модели, когда требуется лишь провести расчеты и не нужно добавлять новые блоки в модель. Использование первого и второго способов приводит к открытию окна обозревателя разделов библиотеки «Simulink» (рис.2.7.).

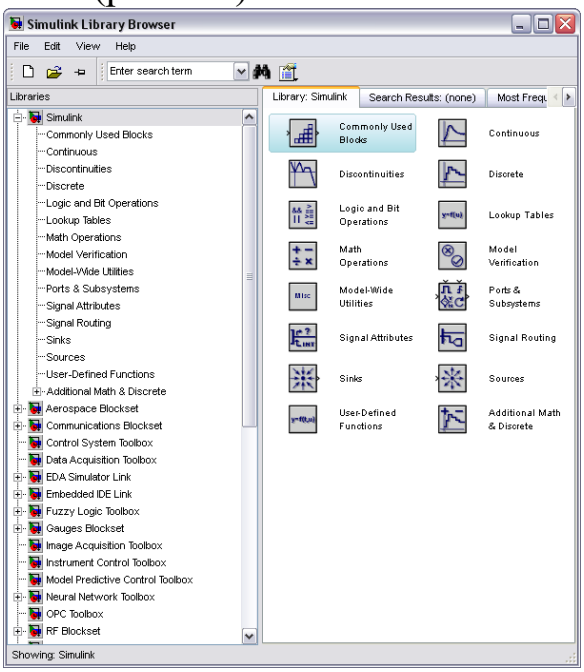


Рис.2.7. Окно обозревателя разделов библиотеки «Simulink»

Глава 3. ОБОЗРЕВАТЕЛЬ РАЗДЕЛОВ БИБЛИОТЕКИ «SIMULINK»

Окно обозревателя библиотеки блоков содержит следующие элементы (рис.3.1.):

1. Заголовок, с названием окна – «Simulink» Library Browser.

2. Меню, скомандами«File», «Edit», «View», «Help».

3. Панель инструментов, с ярлыками наиболее часто используемых команд.

4. Окно комментария для вывода поясняющего сообщения о выбранном блоке.

5. Список разделов библиотеки, реализованный в виде дерева.

6. Окно содержимого раздела библиотеки (список вложенных разделов библиотеки или блоков)

7.Строка состояния, содержащая подсказку по выполняемому действию.

На рисунке 3.1выделена основная библиотека «Simulink» (в левой части окна) и показаны ее разделы (в правой части окна).

Библиотека «Simulink» содержит следующие основные разделы:

– «Continuous» – линейные блоки.
– «Discrete» – дискретные блоки.
– «Functions & Tables» – функцииитаблицы.
– «Math» – блоки математических операций.
– «Nonlinear» – нелинейные блоки.
– «Signals & Systems» – сигналы и системы.
– «Sinks» - регистрирующие устройства.
– «Sources» — источники сигналов и воздействий.
– «Subsystems» – блоки подсистем.

Список разделов библиотеки «Simulink» представлен в виде дерева, и правила работы с ним являются общими для списков такого вида:пиктограмма свернутого узла дерева содержит символ «+», а пиктограмма развернутого содержит символ «-».Для того чтобы развернуть или свернуть узел дерева, достаточно щелкнуть на его пиктограмме левой клавишей мыши (ЛКМ).

При выборе соответствующего раздела библиотеки в правой части окна отображается его содержимое (рис.3.1).

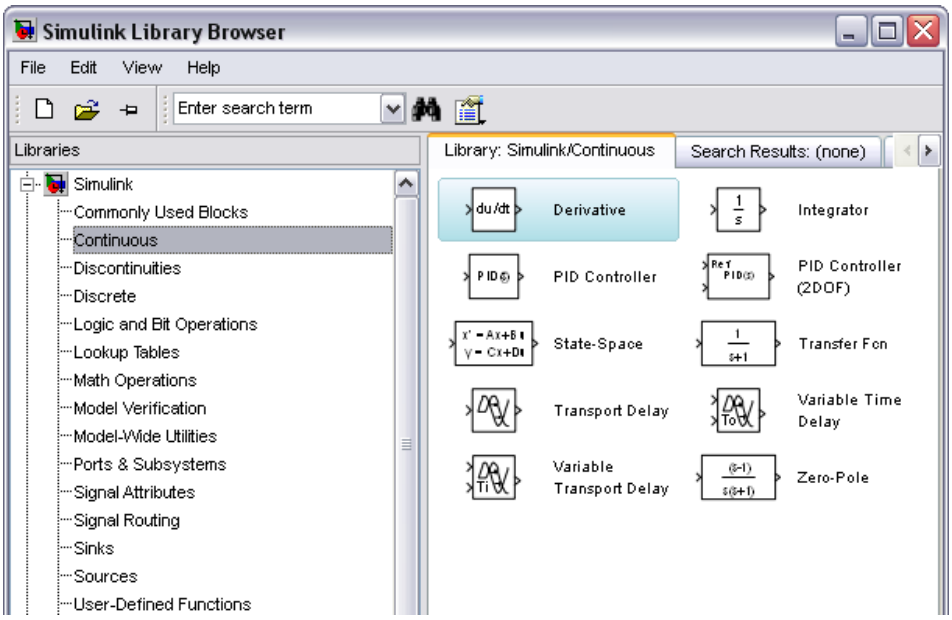


Рис 3.1. Окно обозревателя с набором блоков раздела библиотеки

Для работы с окном используются команды собранные в меню. Меню обозревателя библиотек содержит следующие пункты:

- «File» (Файл) — Работа с файлами библиотек;
- «Edit» (Редактирование) — Добавление блоков и их поиск (по названию);
- «View» (Вид) — Управление показом элементов интерфейса;
- «Help» (Справка) — Вывод окна справки по обозревателю библиотек.

Полный список команд меню обозревателя библиотек приведен в Приложении 1.

Для работы с обозревателем можно также использовать кнопки на панели инструментов (рис.3.2.).

Рис 3.2. Панель инструментов обозревателя разделов библиотек

Кнопки панели инструментов имеют следующее назначение:

- создать новую S-модель (открыть новое окно модели);
- открыть одну из существующих S-моделей;

• изменить свойства окна обозревателя. Данная кнопка позволяет установить режим отображения окна обозревателя "поверх всех окон”. Повторное нажатие отменяет такой режим;

• поиск блока по названию (по первым символам названия).

После того как блок будет найден, в окне обозревателя откроется соответствующий раздел библиотеки, а блок будет выделен. Если же блок с таким названием отсутствует, то в окне комментария будет выведено сообщение «Not found <имя блока>» (Блок не найден).

Для создания модели в среде «Simulink» необходимо последовательно выполнить ряд действий:

1. Создать новый файл модели с помощью команды «File/New/Model», или используя кнопку на панели инструментов (***здесь и далее, с помощью символа «/», указаны пункты меню программы, которые необходимо последовательно выбрать для выполнения указанного действия***).

Вновь созданное окно модели показано на рисунке3.3.

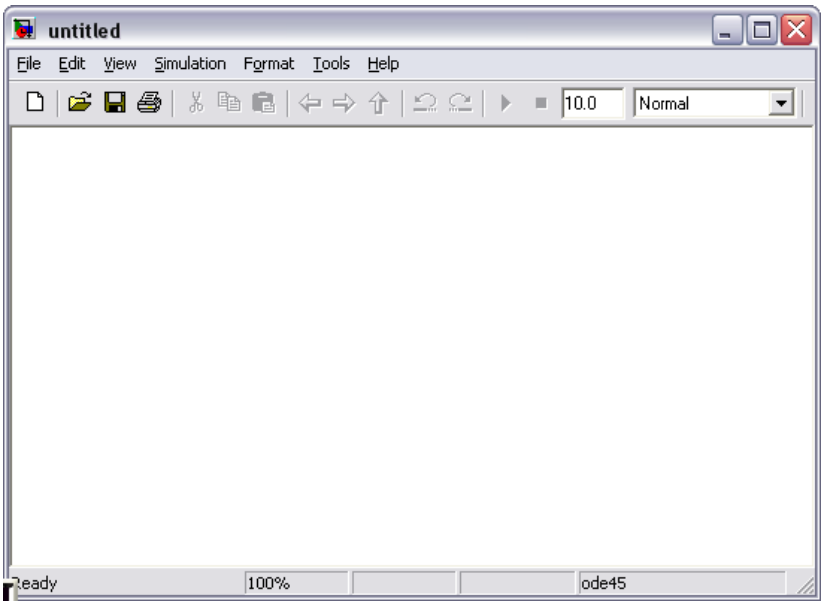


Рис.3.3. Пустое окно модели

2. Расположить блоки в окне модели. Для этого необходимо открыть соответствующий раздел библиотеки (Например, «Sources» - Источники). Далее, указав курсором на требуемый блок и нажав на левую клавишу «мыши» - «перетащить» блок в созданное окно. Клавишу мыши нужно держать нажатой. На рисунке 3.4 показано окно модели, содержащее блоки.

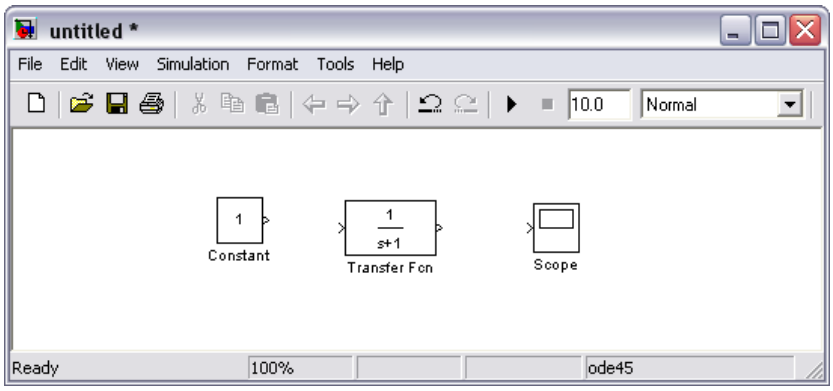


Рис.3.4. Окно модели, содержащее блоки

Для удаления блока необходимо выбрать блок (указать курсором на его изображение и нажать левую клавишу «мыши»), а затем нажать клавишу «Delete» на клавиатуре.

Для изменения размеров блока требуется выбрать блок, установить курсор в один из углов блока и, нажав левую клавишу «мыши», изменить размер блока (курсор при этом превратится в двухстороннюю стрелку).

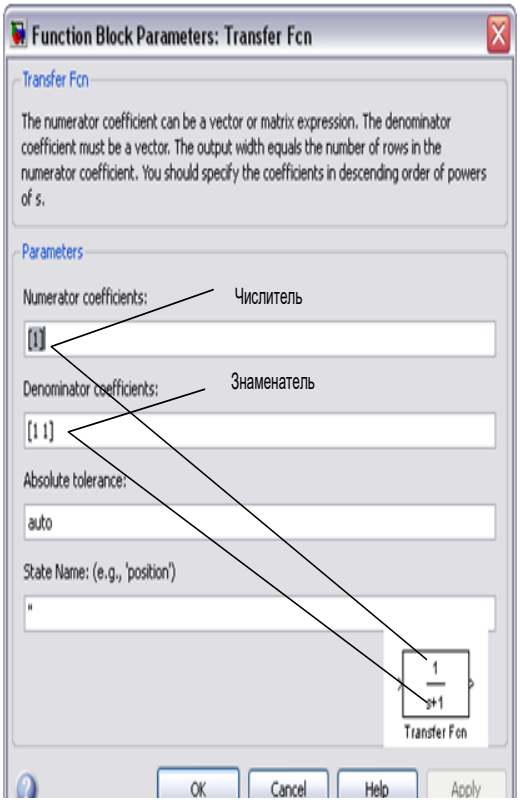


Рис.3.5. Окно редактирования параметров

3. Далее, если это требуется, нужно изменить параметры блока, установленные программой «по умолчанию». Для этого необходимо дважды щелкнуть левой клавишей «мыши», указав курсором на изображение блока. Откроется окно редактирования параметров данного блока. При задании численных параметров следует иметь в виду, что в качестве десятичного разделителя должна использоваться точка, а не запятая. После внесения изменений нужно закрыть окно кнопкой «OK». На рисунке 3.5 в качестве примера показаны блок, моделирующий передаточную функцию и окно редактирования параметров данного блока.

4. После установки на схеме всех блоков из требуемых библиотек нужно выполнить соединение элементов схемы. Для соединения блоков необходимо указать курсором на «выход» блока, а затем, нажать и, не отпуская левую клавишу «мыши», провести линию к

входу другого блока. После чего отпустить клавишу. В случае правильного соединения изображение стрелки на входе блока изменяет цвет. Для создания точки разветвления в соединительной линии нужно подвести курсор к предполагаемому узлу и, нажав правую клавишу «мыши», протянуть линию. Для удаления линии требуется выбрать линию (так же, как это выполняется для блока), а затем нажать клавишу «Delete» на клавиатуре. Схема модели, в которой выполнены соединения между блоками, показана на рисунке 3.6.

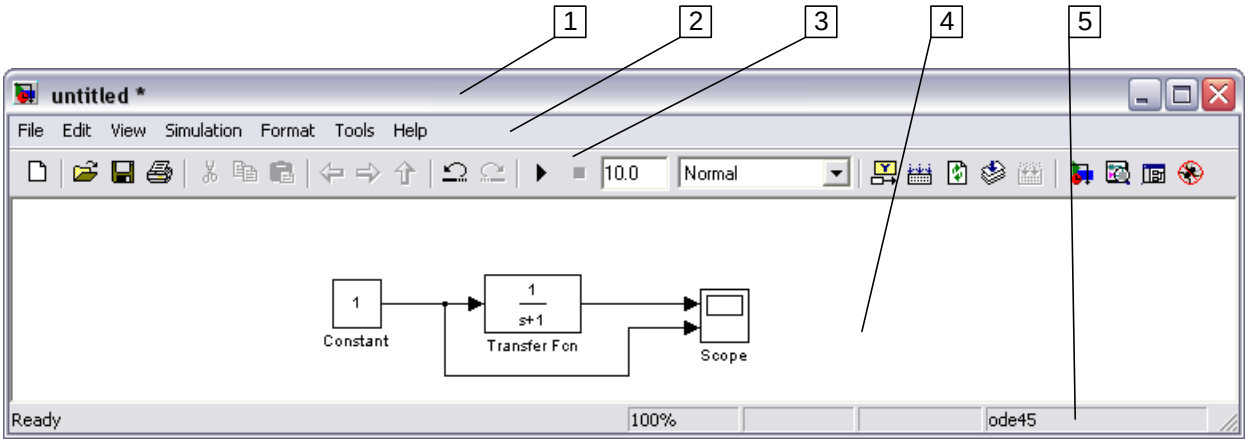


Рис.3.6. Схема модели. 1 - .имя файла, 2 – строка редактора, 3 – кнопка пуск, 4 - окно модели, 5 - метод вычислений

5. После составления расчетной схемы необходимо сохранить ее в виде файла на диске, выбрав пункт меню «File/Save As...» в окне схемы и указав папку и имя файла. Следует иметь в виду, что имя файла не должно превышать 32 символов, должно начинаться с буквы и не может содержать символы кириллицы и спецсимволы. Это же требование относится и к пути файла (к тем папкам, в которых сохраняется файл). При последующем редактировании схемы можно пользоваться пунктом меню «Fille/Save». При повторных запусках программы «Simulink» загрузка схемы осуществляется с помощью меню «File/Open...» в окне обозревателя библиотеки или из основного окна MATLAB.

6. Окно модели

Окно модели содержит следующие элементы (см. рис.3.3, п.1):

- заголовок, с названием окна. Вновь созданному окну присваивается имя «Untitled» с соответствующим номером;
- меню с командами «File», «Edit», «View» и т.д.;
- панель инструментов;

- окно для создания схемы модели;
- строка состояния, содержащая информацию о текущем состоянии модели.

Меню окна содержит команды для редактирования модели, ее настройки и управления процессом расчета, работы файлами и т.п.:

- «File» (Файл) – Работа с файлами моделей;
- «Edit» (Редактирование) – Изменение модели и поиск блоков;
- «View» (Вид) – Управление показом элементов интерфейса;
- «Simulation» (Моделирование) – Задание настроек для моделирования и управление процессом расчета;
- «Format» (Форматирование) – Изменение внешнего вида блоков и модели в целом;
- «Tools» (Инструментальные средства) – Применение специальных средств для работы с моделью (отладчик, линейный анализ и т.п.);
- «Help» (Справка) – Вывод окон справочной системы.

Полный список команд меню окна модели приведен в приложении 1.

Для работы с моделью можно также использовать кнопки на панели инструментов (рис.3.7.).

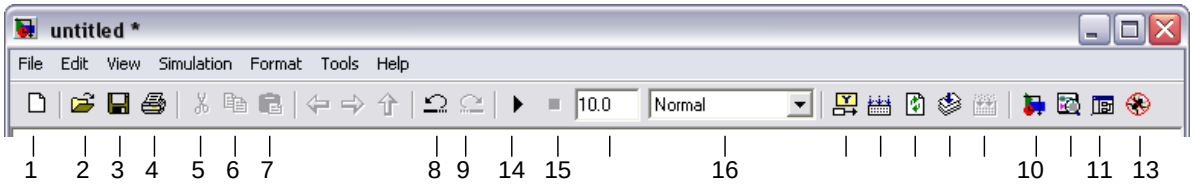


Рис.3.7. Панель инструментов окна модели

Кнопки панели инструментов имеют следующее назначение.

1. «New Model»–открыть новое (пустое) окно модели.
2. «Open Model»–открыть существующий mdl-файл.
3. «Save Model»–сохранить mdl-файл на диске.
4. «Print Model»–вывод на печать блок-диаграммы модели.
5. «Cut»–вырезать выделенную часть модели в буфер промежуточного хранения.
6. «Copy»–скопировать выделенную часть модели в буфер промежуточного хранения.

7. «Paste»–вставить в окно модели содержимое буфера промежуточного хранения.

8. «Undo»–отменить предыдущую операцию редактирования.

9. «Redo»–восстановить результат отмененной операции редактирования.

10. «Library Browser»–открыть окно обозревателя библиотек.

11. «Toggle Model Browser»–открыть окно обозревателя модели.

12. «Debug»–запуск отладчика модели.

13. «Start/Pause/Continue Simulation»–запуск модели на исполнение (команда Start); после запуска модели на изображении кнопки выводится символ, и ей соответствует уже команда Pause (Приостановить моделирование); для возобновления моделирования следует щелкнуть по той же кнопке, поскольку в режиме паузы ей соответствует команда Continue (Продолжить).

14. «Stop»–закончить моделирование. Кнопка становится доступной после начала моделирования, а также после выполнения команды Pause.

15. «Normal/Accelerator»–обычный/ускоренный режим расчета. Инструмент доступен, если установлено приложение ««Simulink» Performance Tool».

В нижней части окна модели находится строка состояния, в которой отображаются краткие комментарии к кнопкам панели инструментов, а также к пунктам меню, когда указатель мыши находится над соответствующим элементом интерфейса. Это же текстовое поле используется и для индикации состояния «Simulink»: Ready (Готов) или Running (Выполнение).

В строке состояния отображаются также:

- масштаб отображения блок-диаграммы (в процентах, исходное значение равно 100%);
- индикатор степени завершенности сеанса моделирования (появляется после запуска модели);
- текущее значения модельного времени (выводится также только после запуска модели);
- используемый алгоритм расчета состояний модели (метод решения).

Глава 4. РАЗРАБОТКА НЕЙРОКОНТРОЛЛЕРА

Каждое приложение нейронной сети уникально, но разработка сети выполняется традиционно:

1. Доступ и подготовка данных.
2. Создание искусственной нейронной сети.
3. Настройка входов и выходов сети.
4. Настройка параметров сети (весов и смещений).
5. Обучение сети.
6. Проверка результатов работы сети.
7. Интегрировать в производственную систему.

Хотя один нейрон и способен выполнять простейшие процедуры распознавания, сила нейронных вычислений проистекает от соединений нейронов в сети.

Искусственные нейронные сети (ИНС) различаются своей архитектурой: структурой связи между нейронами, числом слоев, функцией активации нейронов, алгоримом обучения. С этой точки зрения среди известных ИНС можно выделить статические, динамические сети и fuzzy – структуры; однослойные и многослойные сети. Различия вычислительных процессов в сетях часто обусловлены способом взаимосвязи нейронов, поэтому выделяют следующие виды сетей:

- сети прямого распространения – сигнал проходит по сети от входа к выходу в одном направлении;
- сети с обратными связями;
- сети с боковыми обратными связями;
- гибридные сети.

В целом, по структуре связей ИНС могут быть сгруппированы в два класса: сети прямого распространения – без обратных связей в структуре и рекуррентные сети – с обратными связями. В первом классе наиболее известными и чаще используемыми являются многослойные нейронные сети, где искусственные нейроны расположены слоями. Связь между слоями однонаправленная и в общем случае выход каждого нейрона связан со всеми входами нейронов последующего слоя. Такие сети являются статическими, т.к. не имеют в своей структуре ни обратных связей, ни динамических элементов, а выход зависит от заданного множества на входе и не зависит от предыдущих состояний сети. Сети второго класса являются динамическими, т.к. из-за об-

ратных связей состояние сети в каждый момент времени зависит от предшествующего состояния. (Ниже приведена классификация).

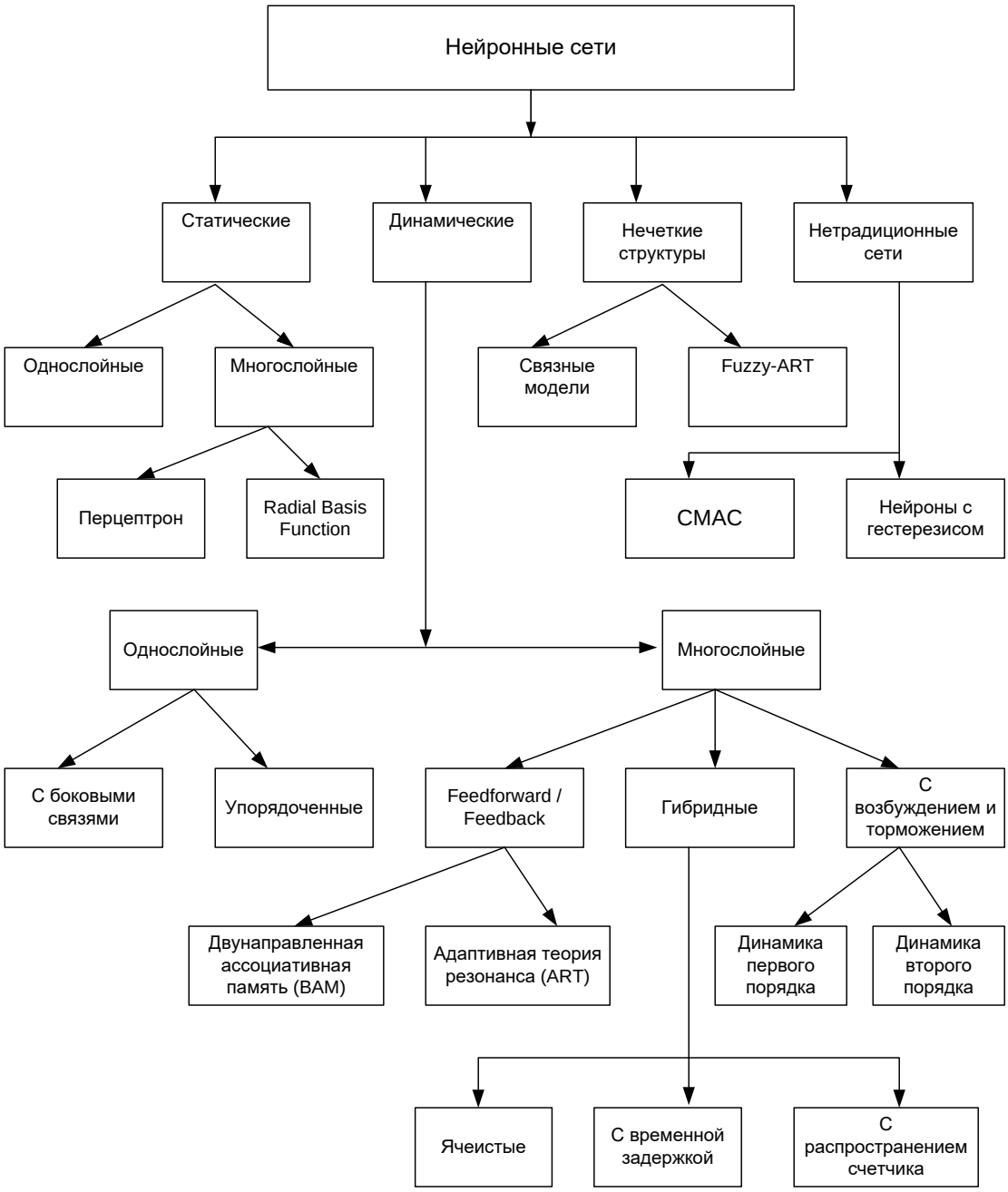


Рис.4.1. Классификация нейронных сетей

Нейрон с одним вектором входа **p** с R элементами $p_1, p_2, .., p_n$ показан на рис. 4.2. Здесь каждый элемент входа умножается на веса w_{11}, w_{12}, … w_{1R}, соответственно и взвешенные значения передаются на сумматор. Их сумма равна скалярному произведению вектора - строки **W** на вектор входа **p**.

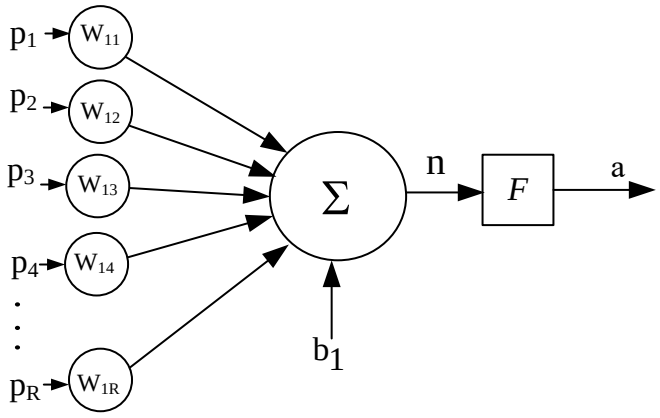


Рис. 4.2. Схема нейрона

Отметим, что вершины-круги слева служат лишь для распределения входных сигналов. Они не выполняют каких- либо вычислений, и поэтому не будут считаться слоем. По этой причине они обозначены кругами, чтобы отличать их от вычисляющих нейронов, обозначенных квадратами. Каждый элемент из множества входов P отдельным весом соединен с каждым искусственным нейроном. А каждый нейрон выдает взвешенную сумму входов в сеть.

Нейрон имеет смещение *b*, которое суммируется с взвешенной суммой входов. Результирующая сумма *n* равна $n=w_{11}\,p_1 + w_{12}\,p_2 +\ldots+ w_{1R}\,p_R + b$ и служит аргументом функции активации.

Структура нейрона, показанная выше, содержит много лишних деталей. При рассмотрении сетей, состоящих из большого числа нейронов, будет использоваться укрупненная структурная схема нейрона (рис. 4.3).

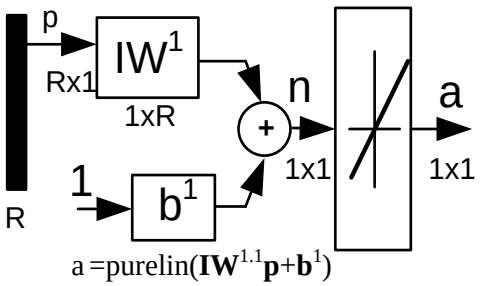


Рис.4.3. Линейный нейрон

Вход нейрона изображается в виде темной вертикальной черты, под которой указывается количество элементов входа *R*. Размер вектора входа **p** указывается ниже символа p равен Rxl. Вектор входа умножается на вектор-строку **W** длины R. Как и прежде, константа 1

рассматривается как вход, который умножается на скалярное смещение *b*. Входом *n* функции активации нейрона служит сумма смещения *b* и произведения **W•p**. Эта сумма преобразуется функцией активации, на выходе которой получаем выход нейрона *a,* который в данном случае является скалярной величиной. Структурная схема, приведенная на рис. 4.3, называется *слоем сети.* Слой характеризуется *матрицей весов* W, смещением b, операциями умножения **W•p**, суммирования и функцией активации. Вектор входов **p** обычно не включается в характеристики слоя.

Как правило, передаточные функции всех нейронов в сети фиксированы, а веса являются параметрами сети и могут изменяться. Некоторые входы нейронов помечены как внешние входы сети, а некоторые выходы - как внешние выходы сети.

При подаче любых чисел на входы сети, получается какой-то набор чисел на выходах сети. Таким образом, работа нейросети состоит в преобразовании входного вектора в выходной вектор, причем это преобразование задается весами сети. Практически любую задачу можно свести к задаче, решаемой нейросетью.

Более крупные и сложные нейронные сети обладают, как правило, и большими вычислительными возможностями. Хотя созданы сети всех конфигураций, какие только можно себе представить, послойная организация нейронов копирует слоистые структуры определенных отделов мозга. Оказалось, что такие многослойные сети обладают большими возможностями, чем однослойные, и в последние годы были разработаны алгоритмы для их обучения.

Нейроны расположены в несколько слоев. Нейроны первого слоя получают входные сигналы, преобразуют их и через точки ветвления передают нейронам второго слоя. Далее срабатывает второй слой и т.д. до к-го слоя, который выдает выходные сигналы для интерпретатора и пользователя. Если не оговорено противное, то каждый выходной сигнал i-го слоя подается на вход всех нейронов i+1-го слоя.

Число нейронов в каждом слое может быть любым и никак заранее не связано с количеством нейронов в других слоях. Теоретически число слоев может быть произвольным, однако фактически оно ограничено ресурсами компьютера или специализированной микросхемы, на которых обычно реализуется НС. Чем сложнее НС тем масштабнее задачи, подвластные ей.

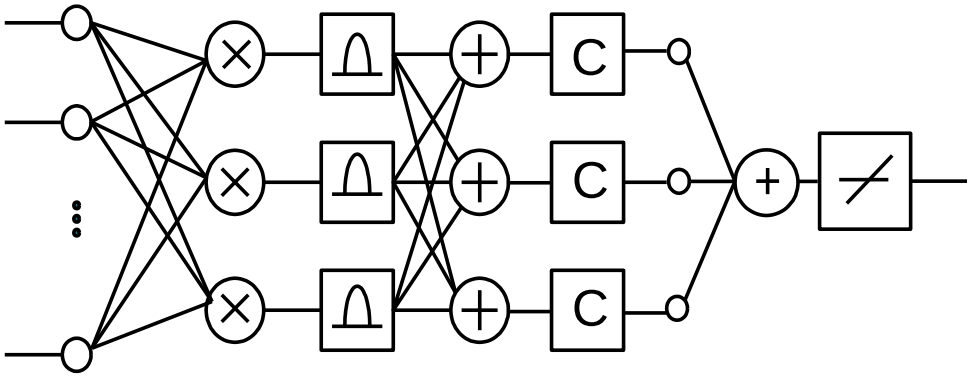


Рис.4.4. Двухслойная нейронная сеть

Стандартный способ подачи входных сигналов: все нейроны первого слоя получают каждый входной сигнал.

Многослойные сети не могут привести к увеличению вычислительной мощности по сравнению с однослойной сетью лишь в том случае, если активационная функция между слоями будет нелинейной. Вычисление выхода слоя заключается в умножении входного вектора на первую весовую матрицу с последующим умножением (если отсутствует нелинейная активационная функция) результирующего вектора на вторую весовую матрицу: (XW1)W2.

Так как умножение матриц ассоциативно, то: X(W1W2).

Это показывает, что двухслойная линейная сеть эквивалентна одному слою с весовой матрицей, равной произведению двух весовых матриц. Следовательно, любая многослойная линейная сеть может быть заменена эквивалентной однослойной сетью. Однослойные сети весьма ограниченны по своим вычислительным возможностям. Таким образом, для расширения возможностей сетей по сравнению с однослойной сетью необходима нелинейная активационная функция.

Сети более общего вида, имеющие соединения от выходов к входам, называются сетями с обратными связями. У сетей без обратных связей нет памяти, их выход полностью определяется текущими входами и значениями весов. В некоторых конфигурациях сетей с обратными связями предыдущие значения выходов возвращаются на входы; выход, следовательно, определяется как текущим входом, так и предыдущими выходами. По этой причине сети с обратными связями могут обладать свойствами, сходными с кратковременной человеческой памятью, сетевые выходы частично зависят от предыдущих входов.

Особое распространение получили трехслойные сети, в которых каждый слой имеет свое наименование: первый – входной, второй – скрытый, третий – выходной.

В дальнейшем будут подробно рассмотрены некоторые виды как статических, так и динамических сетей.

Выбор структуры НС осуществляется в соответствии с особенностями и сложностью задачи. Для решения некоторых отдельных типов задач уже существуют оптимальные, на сегодняшний день, конфигурации. Если же задача не может быть сведена ни к одному из известных типов, разработчику приходится решать сложную проблему синтеза новой конфигурации. При этом он руководствуется несколькими принципами: возможности сети возрастают с увеличением числа ячеек сети, плотности связей между ними и числом выделенных слоев. Введение обратных связей наряду с увеличением возможностей сети поднимает вопрос о динамической устойчивости сети; сложность алгоритмов функционирования сети (в том числе, например, введение нескольких типов синапсов – возбуждающих, тормозящих и др.) также способствует усилению мощи НС. Вопрос о необходимых и достаточных свойствах сети для решения того или иного рода задач представляет собой целое направление нейрокомпьютерной науки. Так как проблема синтеза НС сильно зависит от решаемой задачи, дать общие подробные рекомендации затруднительно. В большинстве случаев оптимальный вариант получается на основе интуитивного подбора.

Очевидно, что процесс функционирования НС, то есть сущность действий, которые она способна выполнять, зависит от величин синаптических связей, поэтому, задавшись определенной структурой НС, отвечающей какой-либо задаче, разработчик сети должен найти оптимальные значения всех переменных весовых коэффициентов (некоторые синаптические связи могут быть постоянными). Этот этап называется обучением НС, и от того, насколько качественно он будет выполнен, зависит способность сети решать поставленные перед ней проблемы во время эксплуатации. На этапе обучения кроме параметра качества подбора весов важную роль играет время обучения. Как правило, эти два параметра связаны обратной зависимостью и их приходится выбирать на основе компромисса.

Оценка состояния систем управления предполагает использование специальных устройств и алгоритмов, предназначенных для определения неисправностей. Исследование влияния нейросети на диагностирование электропривода было проведено на базе ИНС с ла-

теральным торможением – метод Гроссберга, сеть встречного распространения. Принцип работы нейронной сети заключается в настройке параметров нейрона таким образом, чтобы поведение сети соответствовало некоторому желаемому поведению. Регулируя веса или параметры смещения, можно обучить сеть выполнять конкретную работу, возможно также, что сеть сама будет корректировать свои параметры, чтобы достичь требуемого результата. Процесс обучения – это процесс подгонки параметров нейронной сети к той модели процесса или явления, которая реализуется нейронной сетью. При помощи регулирования параметров обучения исследуемая система способна самонастраиваться (самообучаться) на оптимальные режимы в процессе эксплуатации (или на начальном этапе обучения).

Далее приведена таблица основных алгоритмов настройки НС.

Таблица 4.1

Основные алгоритмы нейронных сетей

Английское название	Русское название
Simulated annealing	Обучение обжигом
ART-1 network	Искусственный резонанс-1
BAM network	Двунаправленная автоассоциативная память
Boltzman machine	Больцманово обучение
Back Propagation	Обратное распространение
Counter Propagation	Сеть встречного распространения
Feed-Forward MAXNET	Сеть поиска максимума с прямыми связями
Gaussian Classifier	Гауссов классификатор
Genetic Algorithm	Генетический алгоритм
Hamming Network	Сеть Хэмминга
Hopfield Network	Сеть Хопфилда
INSTAR Network	Входная звезда
Kohonen Network	Сеть Кохонена
MAXNET Network	Сеть поиска максимума
OUTSTAR Network	Выходная звезда
Delta-Bar-Delta Network	Delta-Bar-Delta сеть
Extended DBD Network	Расширенная DBD сеть
Radial Basis Function Network	Сеть радиального основания
Single Layer Perceptron	Однослойный персептрон

В изученной нейросети между нейронами одного слоя имеются постоянные тормозящие связи (латеральное торможение). Согласно заложенному алгоритму система относит образ регулируемого параметра к одному из классов в зависимости от того, на какой из запомненных образов он больше всего похож. Если входящий образ не соответствует ни одному из запомненных, создается новый класс путем его запоминания. Если найден класс с определенным допуском соответствующий входному, то он модифицирует его так, чтобы стать еще больше похожим на входящий. При этом активируются нейроны сети, образуя определенные связи между слоями. Главная особенность систем на базе НС – способность к обучению во время проектирования или в процессе управления реальным объектом. По этому методу система способна обучаться, определяя порядок поиска соответствия входного образа своему прототипу. А решение прекратить поиск зависит от критерия сходства k ($0<k<1$), который контролирует степень подобия и определяется порогом распознавания ρ. При фиксированном пороге сеть автоматически масштабирует свою чувствительность к образам различной сложности. Если входной образ сложный, то система пренебрегает небольшим различиями между входными сигналами и прототипом, а если входной образ содержит мало активных нейронов, то такое же несовпадение может привести к генерации сигнала сброса. Изменение порога чувствительности (например, по результатам работы системы) позволяет регулировать параметры сходства и детально анализировать входной образ.

Как было сказано выше, одним из свойств НС является способность к обучению, то есть эксперт-учитель, формируя нейросеть, закладывает в ее БЗ свои знания, выводы об исправности или неисправности той или иной системы, и НС сама представляет заключения о состоянии системы. Этим значительно упрощается диагностика.

Нейросети также обладают высокой вычислительной мощностью и высокой отказоустойчивостью.

При исследовании влияния свойств нейросети на качество диагностики на вход нейронной сети подается вектор (матрица из 2 строк) с исправным и неисправным сигналами с ЭМС. Результатом обучения является отнесение неисправных точек (отклонение от нормы ко второму классу (2 – неисправность при изменении параметров). Проводится процесс обучения и сеть тестируется для просмотра результатов обучения.

Таким образом, изменяя параметры обучения можно добиться требуемого качества распознавания, выбрав оптимальные характеристики процесса обучения (коэффициент обучения, количество циклов обучения, функция обучения), алгоритма настройки НС и ее архитектуры. Исследование и применение ИС необходимо и актуально. При решении различных задач, требующих точных моментальных выводов, нейросети показывают высокую вычислительную мощность и высокую отказоустойчивость, способность обучаться, самообучаться, адаптироваться к новым знаниям.

4.1. Диагностика неисправностей с использованием нейромоделей

Таким образом, задача диагностики электромеханического устройства на основе нейросетей сводится: к исследованию изменений какого-либо параметра устройства, занесению в базу знаний информации об исправности и неисправности устройства по исследуемому параметру, сопоставление заложенной и полученной информации, и формирование предложений по дальнейшей эксплуатации устройства.

Основной проблемой технической диагностики основанной на нейросетях является автоматизированный сбор, обработка измеренных характеристик объекта и выдача заключения о его состоянии.

Для решения проблем технической диагностики используются разнообразные компьютерные программные комплексы, основанные на различных математических методах. В данной работе рассматриваются вопросы диагностики привода с помощью системы MATLAB с пакетом расширения «Neural Network Toolbox». Перечень дополнительной информации приведен в приложении 1 и 2.

При определении входных данных, необходимых для работы автоматической системы диагностики, основанной на нейронных сетях, анализируют переходные характеристики привода при различных неисправностях в нем, которые могут быть получены, так как показано в первой главе. Набор входных данных, полученный при исследовании привода (вектор состояния привода) отражает минимум, по которому можно определить его состояние. Данные представляют собой последовательность числовых значений - координат переходной характеристики (в векторной форме), снятых через определенные, равные промежутки времени.

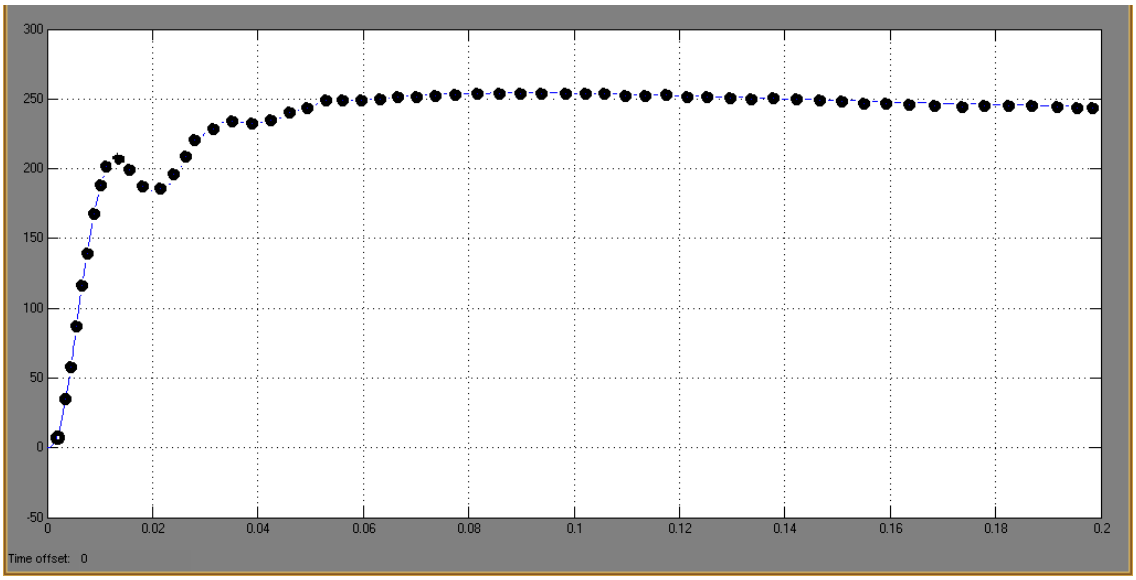


Рис. 4.6. Измеряемые данные

Набор входных данных (вектор состояния привода) полученный из координат переходной характеристики, вводится в компьютер для автоматической постановки диагноза.

При постановке задачи для обучения нейросетей исходят из того, что диагностическая система должна выбирать один из предполагаемых диагнозов из заданного набора, на основе параметров привода, по которым производится оценка его состояния.

4.2. Сбор данных для нейронной сети

Набор данных представляет собой набор *наблюдений*, для которых указаны значения входных и выходных *переменных*. Первый вопрос, который нужно решить, - какие переменные использовать и сколько (и каких) наблюдений собрать. Предположим, что только пять переменных оказывают существенное влияние на работу привода:

1) момент нагрузки (M);
2) сопротивление якоря двигателя (R);
3) коэффициент в блоке широтно-импульсного модулятора (T_{shim});
4) коэффициент в блоке контура тока (Kt);
5) коэффициент в блоке контура скорости (Ks).

Нейронные сети могут работать с числовыми данными, лежащими в определенном ограниченном диапазоне. Это создает проблемы в случаях, когда данные имеют нестандартный масштаб, когда в них имеются пропущенные значения, и когда данные являются не

числовыми. В работе была, например, проблема с начальным значением обучающей выборки, которое во всех случаях было равно нулю. При поиске весов для нейронов нейросеть умножает обучающий пример на случайный весовой коэффициент, находит произведение и сравнивает с заданным результатом. В случае с нулем она не может подобрать правильный вес нейронам, так как при этом в любом случае произведение оказывалось равным нулю. Проблема «эффекта нуля» была решена введением блоков, увеличивающих все значения входных векторов на равные величины, то есть график смещался вверх на определенную величину по оси ординат.

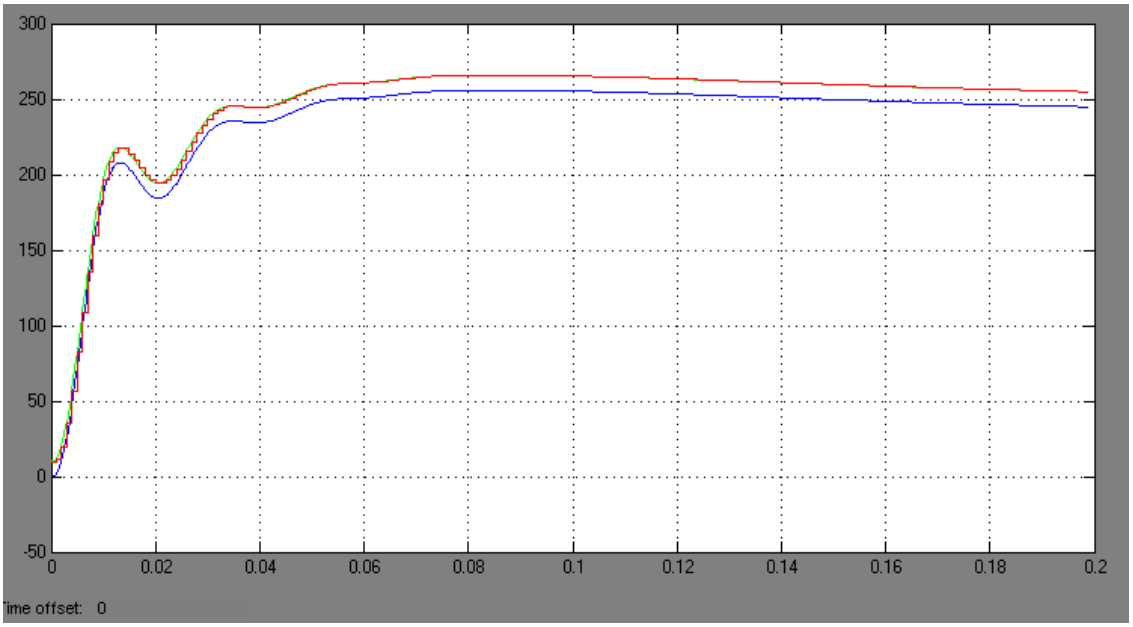


Рис. 4.7. Переходная характеристика модели в процессе сбора данных

Вопрос о том, сколько наблюдений нужно иметь для обучения сети, часто оказывается непростым. Известен ряд эвристических правил, увязывающих число необходимых наблюдений с размерами сети (простейшее из них гласит, что число наблюдений должно быть в десять раз больше числа связей в сети). На самом деле это число зависит также от (заранее неизвестной) сложности того отображения, которое нейронная сеть стремится воспроизвести. С ростом количества переменных количество требуемых наблюдений растет нелинейно, так что уже при довольно небольшом (например, пятьдесят) числе переменных может потребоваться огромное число наблюдений.

Во многих реальных задачах приходится иметь дело с не вполне достоверными данными. Значения некоторых переменных могут быть искажены шумом или частично отсутствовать. Пакет «ST Neural Networks» имеет специальные средства работы с пропущенными значениями (они могут быть заменены на среднее значение этой переменной или на другие ее статистики). Кроме того, нейронные сети в целом устойчивы к шумам. Однако у этой устойчивости есть предел. Например, выбросы, т.е. значения, лежащие очень далеко от области нормальных значений некоторой переменной, могут исказить результат обучения. В таких случаях лучше всего постараться обнаружить и удалить эти выбросы (либо удалив соответствующие наблюдения, либо преобразовав выбросы в пропущенные значения).

В рассматриваемо случае для формирования обучающего множества было взято 200 координатных точек для каждого графика. Из графика на рисунке 4.7 видно преобразование графика, т.е. сначала график смещается вверх по оси ординат на определенную величину, без изменения формы кривой, а затем он из непрерывного преобразуется в ступенчатый с помощью блока дискретизации.

После того, как определено число слоев и число элементов в каждом из них, нужно найти значения для весов и порогов сети, которые бы минимизировали ошибку прогноза, выдаваемого сетью. Именно для этого служат *алгоритмы обучения*. С использованием собранных исторических данных веса и пороговые значения автоматически корректируются с целью минимизировать эту ошибку. По сути, этот процесс представляет собой подгонку модели, которая реализуется сетью, к имеющимся обучающим данным. Ошибка для конкретной конфигурации сети определяется путем прогона через сеть всех имеющихся наблюдений и сравнения реально выдаваемых выходных значений с желаемыми (целевыми) значениями. Все такие разности суммируются в так называемую *функцию ошибок*, значение которой и есть ошибка сети. В качестве функции ошибок чаще всего берется сумма квадратов ошибок, т.е. когда все ошибки выходных элементов для всех наблюдений возводятся в квадрат и затем суммируются. При работе с пакетом «ST Neural Networks» пользователю выдается так называемая среднеквадратичная ошибка (RMS) - описанная выше величина нормируется на число наблюдений и переменных, после чего из нее извлекается квадратный корень - это очень хорошая мера

ошибки, усредненная по всему обучающему множеству и по всем выходным элементам.

Общая схема обучения нейросети включает несколько этапов.

1. Создается обучающая выборка (в данном случае из 420 элементов). Для этого в MATLAB создаем модель для сбора данных, как показано на рисунке 4.8.

2.

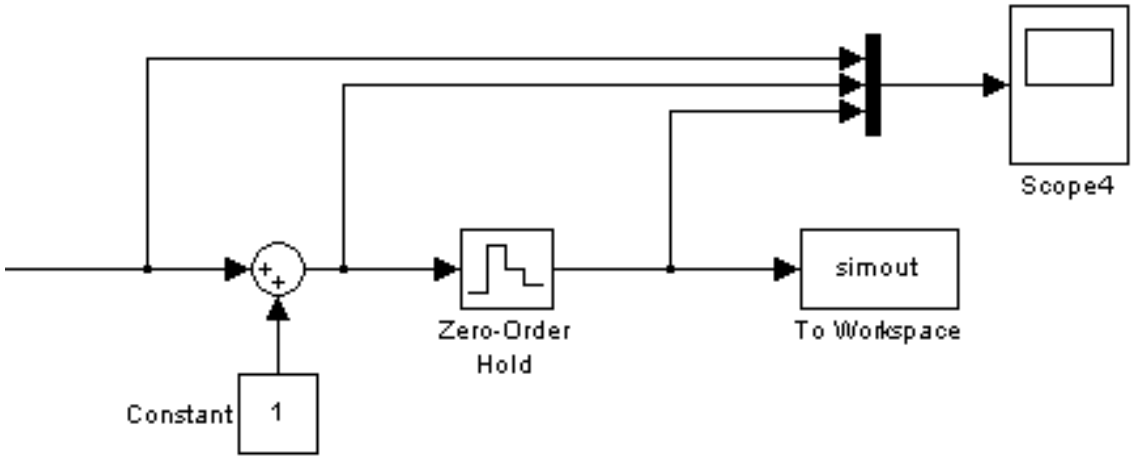


Рис.4.8. Фрагмент модели для сбора обучающей информации

На этом рисунке изображены блок «Constant» и блок суммирования, которые прибавляют к исходному сигналу единицу, для исключения «эффекта нуля». Блок «Zero- Order Hold» преобразует непрерывный сигнал в ступенчатый. Блок «To Workspace» передает координаты «ступеней» в рабочую область MATLAB и создает таблицу (вектор) данных из 200 элементов.

После записи первого вектора данных, меняют переменную с помощью маски и снова записывают данные рабочую область MATLAB. Результаты моделирования объединяются в одну матрицу с числом столбцов равным числу элементов в обучающей выборке. Эта матрица называется матрицей входов. В данной работе фиксировались 20 векторов для каждого из 21 диагноза, которые мы собираемся ставить.Получилась матрица размера 420x200, т.е. 420 примеров для обучения по 200 элементов в каждом.

Для ускорения процесса сбора данных можно использовать модель, приведенную на рисунке 4.9.

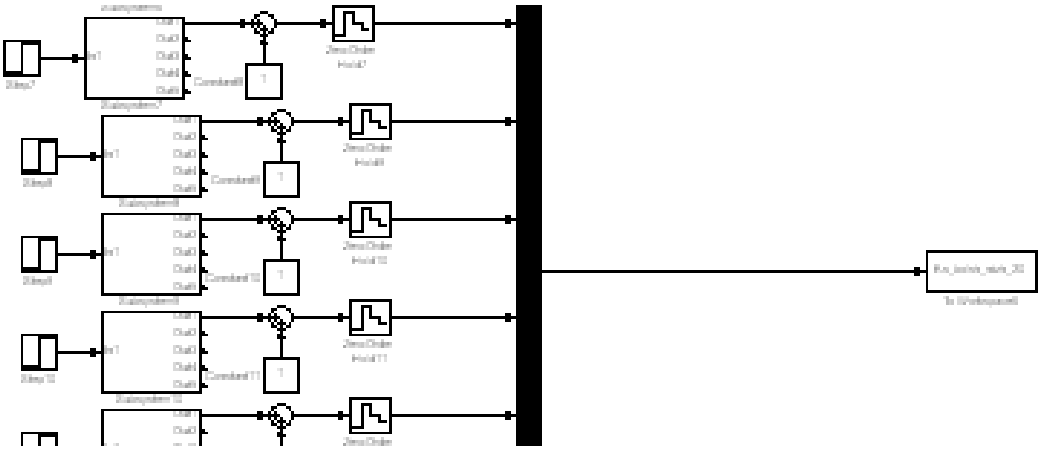


Рис. 4.9. Фрагмент модели для ускорения процесса сбора данных

На данном рисунке изображены 20 параллельно работающих приводов (по числу обучающих векторов для каждой неисправности). В каждом блоке привода с помощью маски меняем значения только той переменной, которую исследуем. Далее запускаем моделирование и в рабочей области MATLAB (рис.4.10.) получаем матрицу значений для одного из 21 состояния привода. Повторяем для другого состояния привода и т.д. для 21 состояния. Получается 21 матрица размера 200x20, которые мы объединяем в одну общую с размерами 420x200. Эта матрица и есть матрица входов. Графики показаны на рис.4.11.

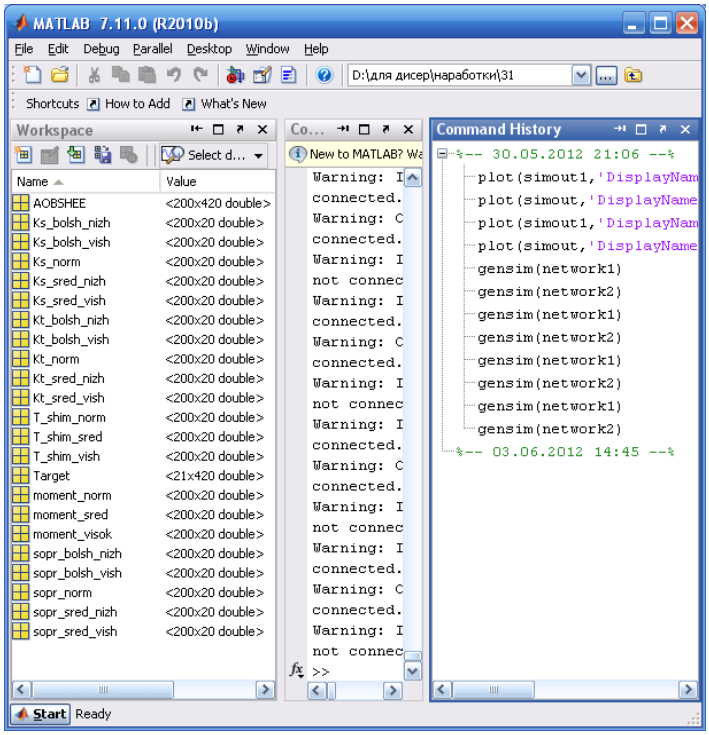


Рис.4.10. Входные данные в рабочей области Матлаб

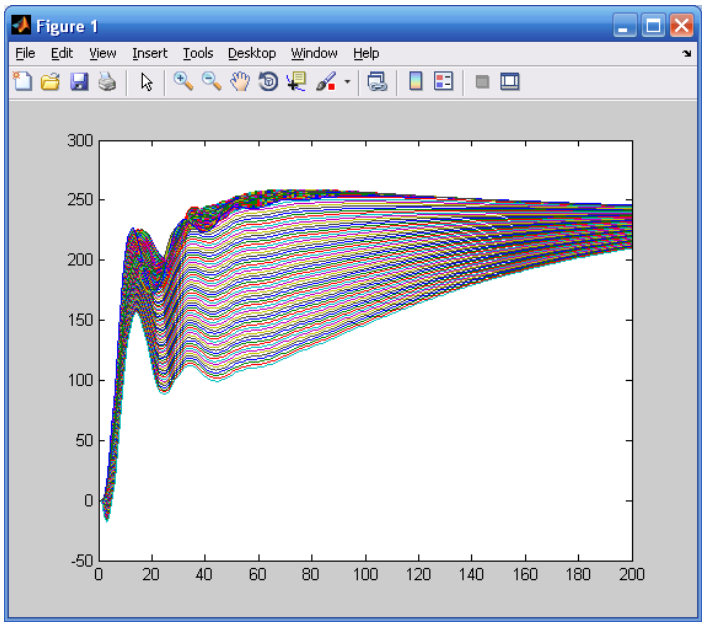


Рис.4.11. Данные в виде графиков

Далее создается матрица целей в виде нулей и единиц. Каждому верно распознанному состоянию будет соответствовать единица на выходе нейросети. Все остальные выходы при этом принимают значение 0. Если нейросеть распознала первое состояние привода, то на её первом выходе должна получиться единица, на остальных выходах нули. Если она распознала второе состояние, единица получается на втором выходе, а на остальных нули. Число столбцов должно совпадать с числом столбцов матрицы входов. То есть каждому столбцу матрицы входов будет соответствовать столбец матрицы целей. В данной работе получилась матрица размера 420x21 (рис.4.12).и целевые функции, представленные на рис. 4.13.

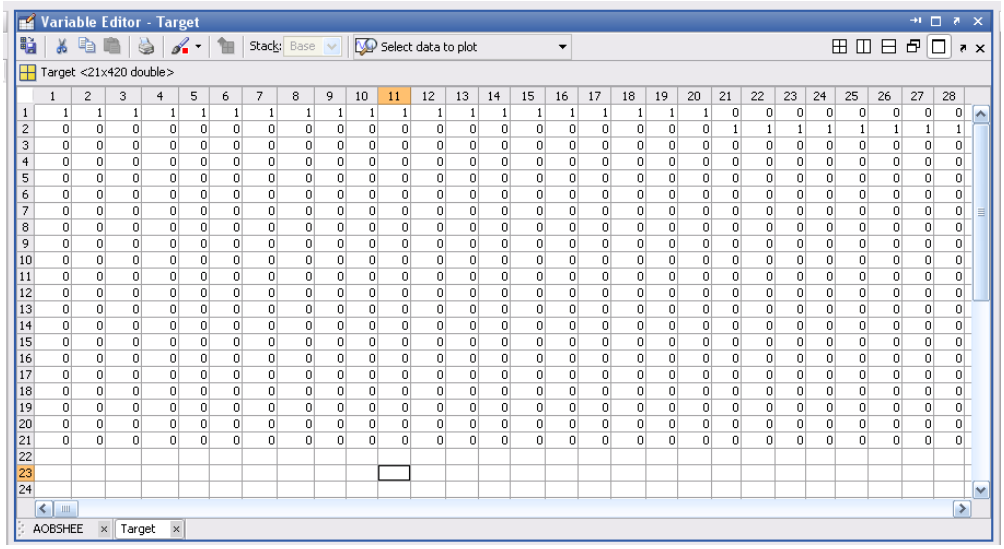


Рис.4.12. Матрица целей в рабочей области Матлаб

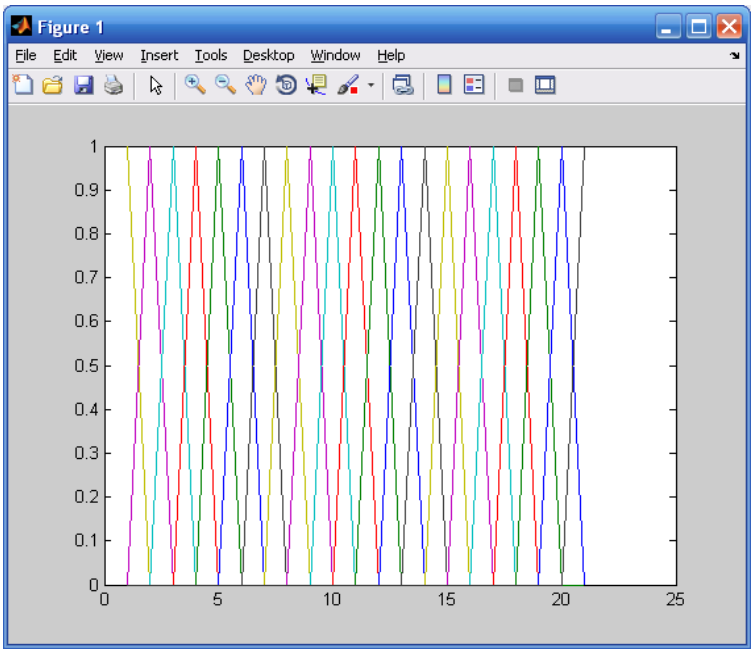


Рис.4.13. Цели в виде графиков

4.3. Выбор нейросети для конкретной задачи и проверки ее работы

После создания матрицы входов и матрицы целей можно приступать к созданию самой нейросети. В главном окне MATLAB в левом нижнем углу нужно нажать кнопку «Start» и выбрать «Toolbox - NeuralNetworkTool» (рис.4.14.).

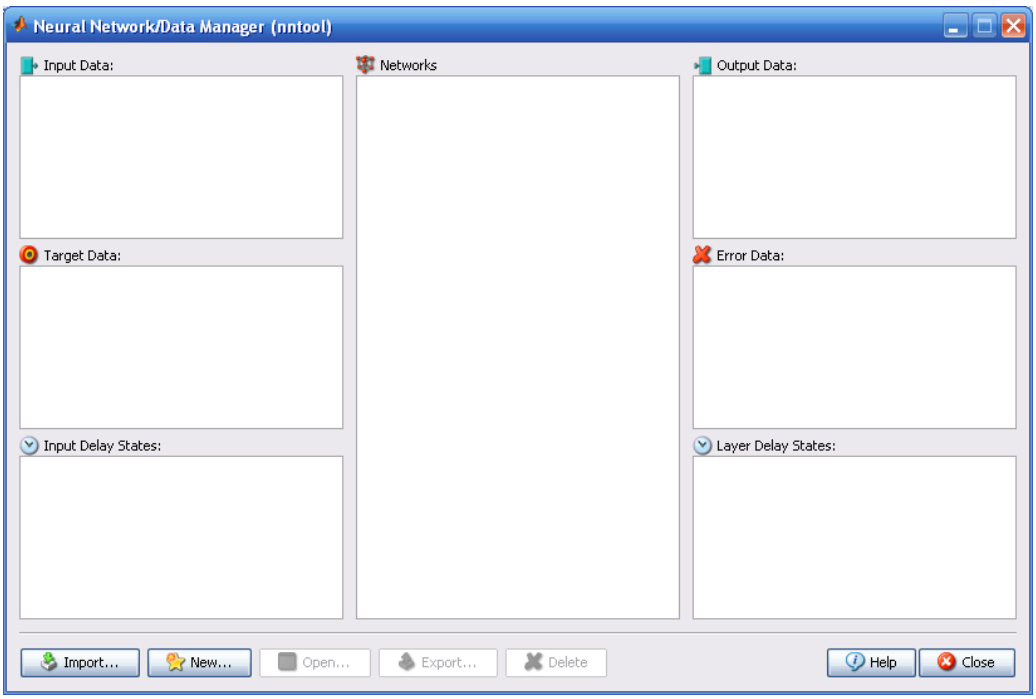


Рис.4.14. Окно создания нейросетей

В открывшемся окне выбираем «Import» и загружаем в появившемся окне матрицу входов и матрицу целей. После того как матрицы загружены, в окне «NeuralNetworkTool» нажимаем кнопку «New» и открываем окно создания сети («CreateNetworkorData»). В разделе «NetworkProperties» выбираем нужный нам тип сети, настраиваем количество слоев и вид функции активации. Так как в настоящее время создано очень много нейросетей, с различными алгоритмами, подбирать сеть под нашу задачу будем методом их перебора, пока не найдем сеть удовлетворяющую нашим запросам. Выбор осуществляем из сетей, предназначенных для классификации образов.

1. Создаем сеть **Feed-forward distributed time delay**. Загружаем для нее данные для обучения, полученные на предыдущем этапе работы (Рис.4.15.).

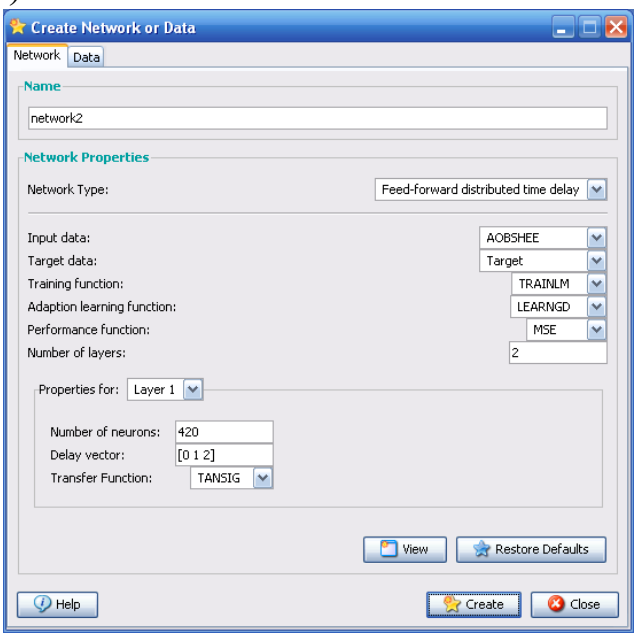


Рис.4.15. Окно настройки создаваемой нейросети

2. Нажимаем кнопку «View», чтоб посмотреть структуру создаваемой сети. (Рис.4.16.)

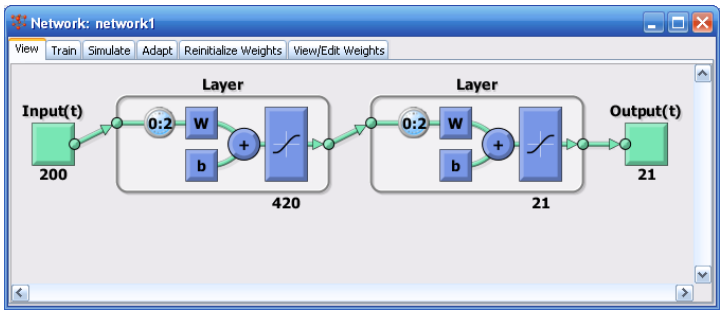


Рис.4.16. Структура создаваемой сети

3. Нажимаем кнопку «Create» чтоб создать сеть. Полученную сеть экспортируем в рабочую область MATLAB, а затем в «Simulink».

Для проверки работоспособности нейросети необходимо создать модель.

Эта модель должна включать в себя:

- электропривод;

-блоки предварительной обработки сигнала;

-собственно блок нейросети;

-блоки обработки данных получаемых с выхода нейросети;

- блоки вывода результатов в форме удобной для дальнейшего использования человеком или ЭВМ.

4. Запускаем моделирование, чтоб проверить работоспособность сети. Изменяя входные данные и наблюдая за выходом нейронной сети, сделаем вывод о ее пригодности для решения данной задачи. Результат работы представлен на рис.4.17.

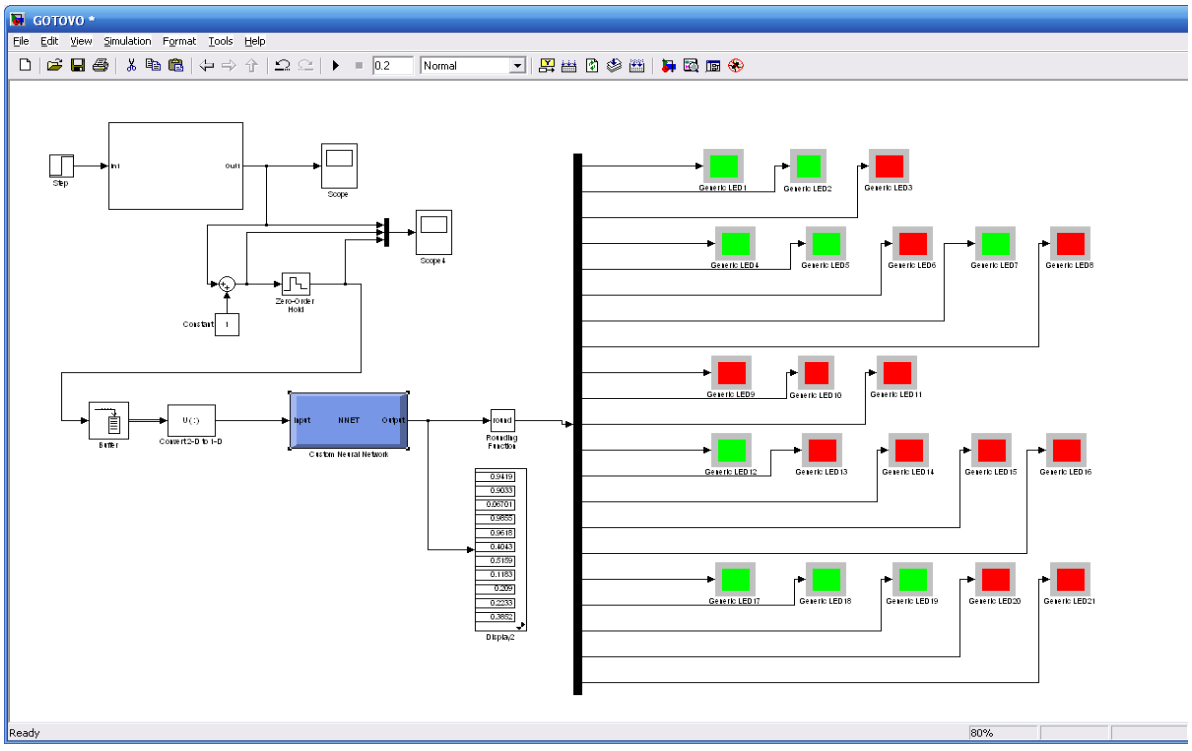


Рис.4.17. Проверка работоспособности сети

Данная нейросеть не может быть применена в нашем случае, так как целевые значения не соответствуют входным данным.

1. Создаем сеть **GeneralizedRegression**. Загружаем для нее данные для обучения.(рис.4.18.)

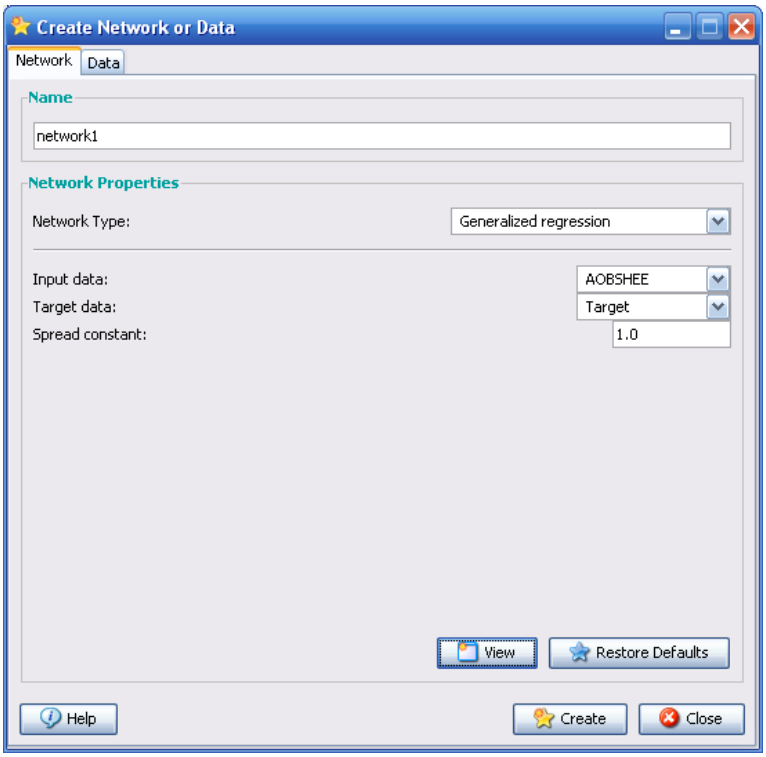


Рис.4.18. Окно настройки создаваемой нейросети

Нажимаем кнопку «View», чтоб посмотреть структуру создаваемой сети (рис.4.19.).

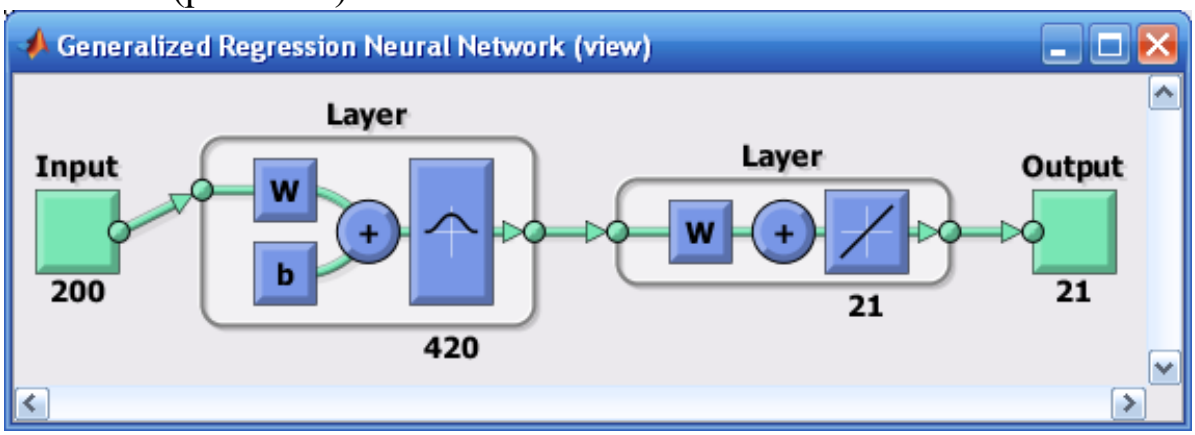


Рис.4.19. Структура создаваемой сети

Нажимаем кнопку «Create» чтоб создать сеть. Полученную сеть экспортируем в рабочую область MATLAB, а затем в «Simulink».

Запускаем моделирование, чтоб проверить работоспособность сети. Изменяя входные данные и наблюдая за выходым сигналом нейронной сети, делаем вывод о ее пригодности для решения данной задачи.

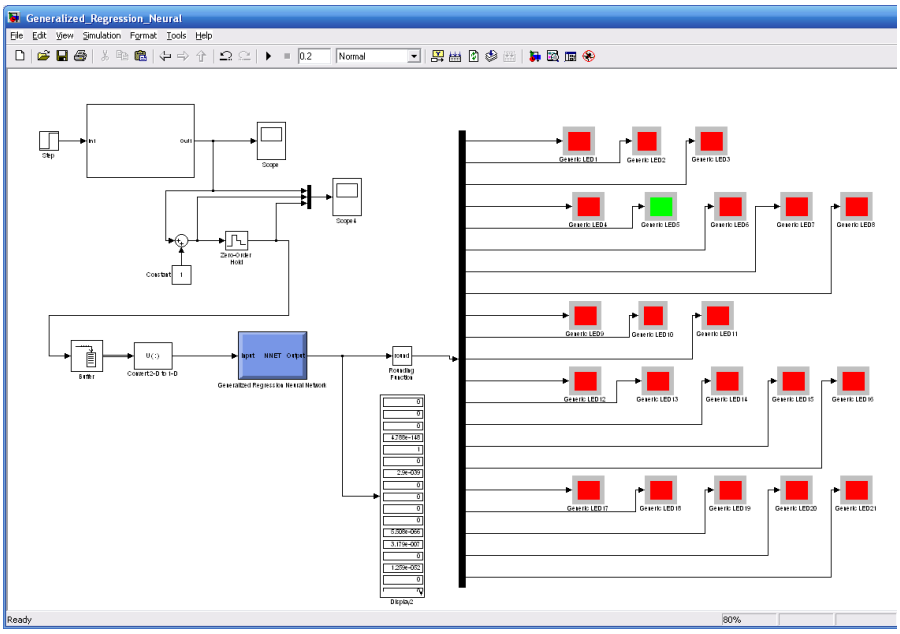


Рис.4.20. Проверка работоспособности сети

Данная нейросеть может быть применена в нашем случае, так как сеть правильно воспроизводит целевые значения в соответствии с входными данными (Рис.4.20). И все таки сеть работает с некоторыми отклонениями, которые можно исключить введением после нейросети блока округления выходных данных. Результаты работы программы приведены на рисунках 4.21, 4.22, 4.23.

Создаем сеть **Layerrecurrent.** Загружаем для нее данные для обучения.

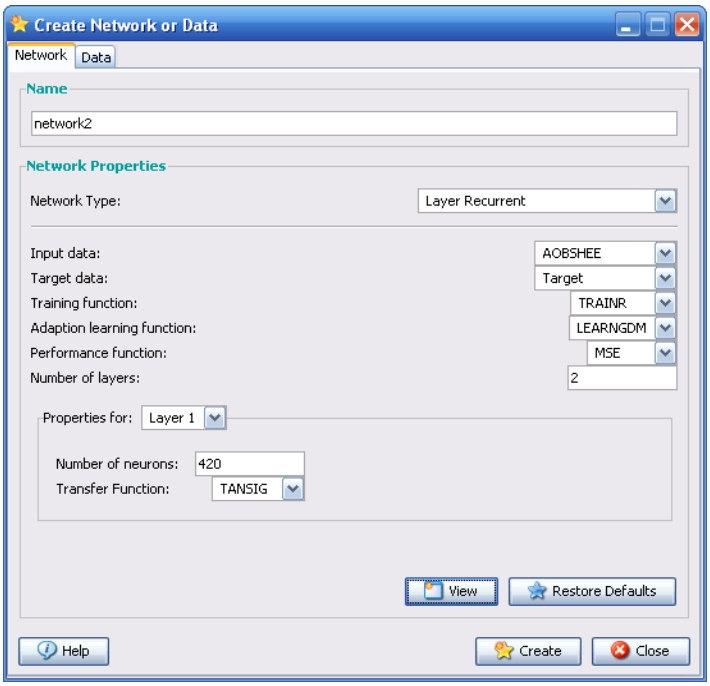


Рис.4.21. Окно настройки создаваемой нейросети

Нажимаем кнопку «View», чтоб посмотреть структуру создаваемой нейронной сети.

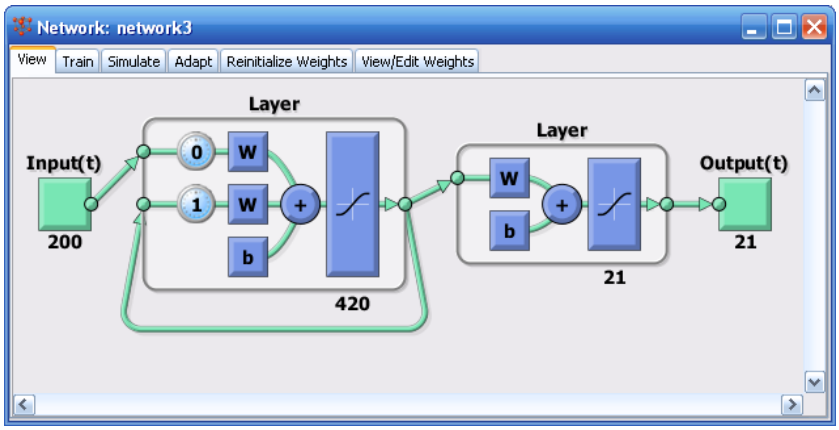


Рис.4.22. Структура создаваемой сети

Нажимаем кнопку «Create» чтоб создать сеть. Полученную сеть открываем, во вкладке Train настраиваем параметры обученияи нажимаем «TrainNetwork» для обучения сети.

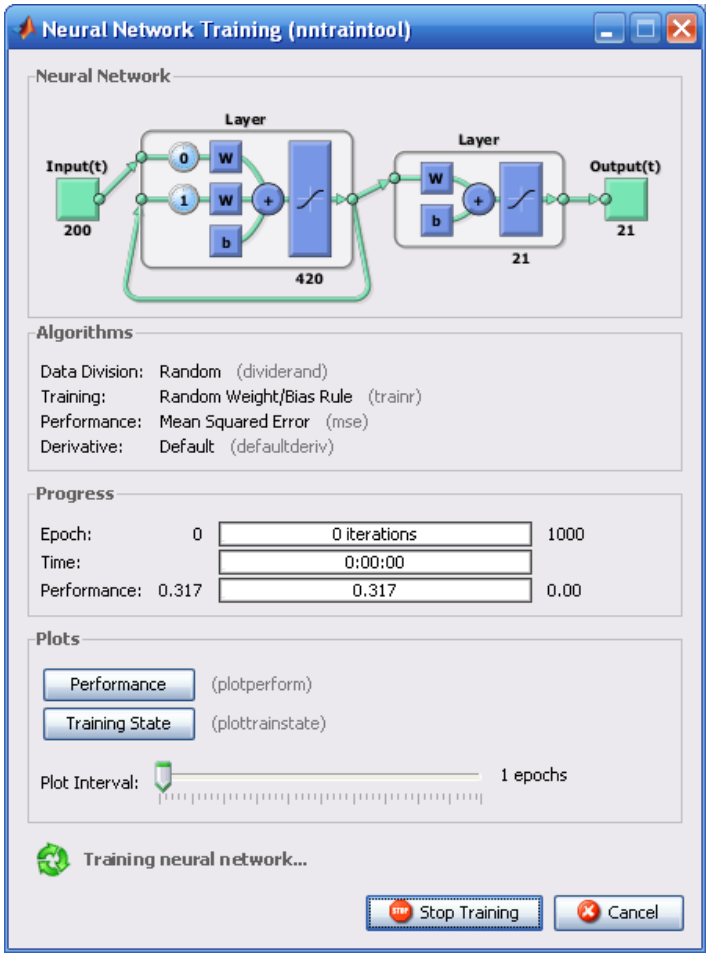


Рис.4.23. Окно тренировки нейросети

Создаем сеть **Probabilistic.** Загружаем для нее данные для обучения (Рис.4.24).

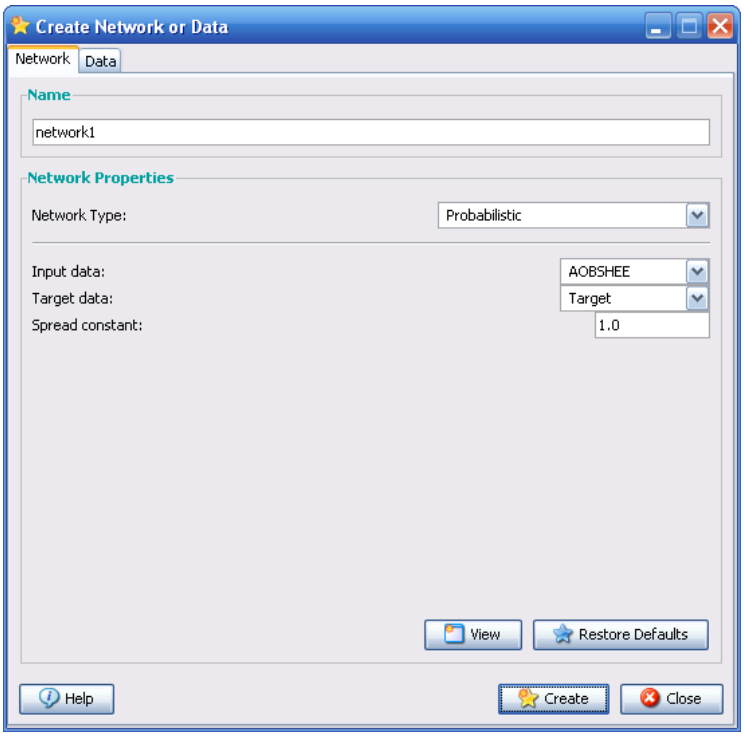


Рис.4.24. Окно настройки создаваемой нейросети

Нажав кнопку «View» можно посмотреть структуру создаваемой нейронной сети (Рис.4.25).

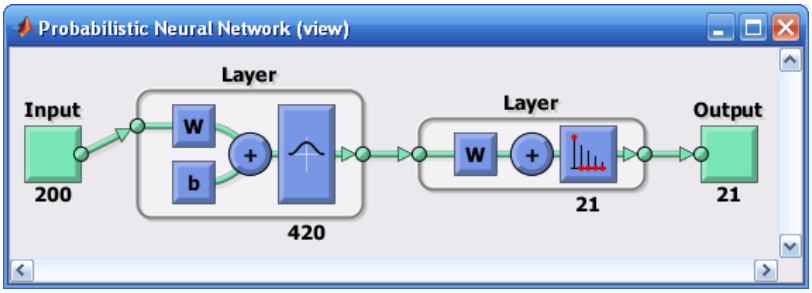


Рис.4.25. Структура создаваемой сети

После нажатия кнопки «Create» автоматически создается сеть и загружается в окно «NeuralNetworkTool».

Созданную нейросеть экспортируем в рабочую область MATLAB, а оттуда с помощью команды gensim в виде блока в окно ««Simulink»Tools» (Рис.4.26).

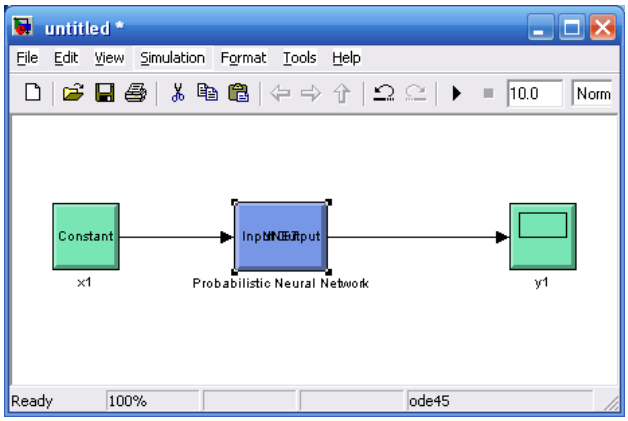


Рис.4.26. Экспорт нейросети в «Simulink»

Для каждого примера в MATLAB вычисляется ошибка обучения. Вычисляется, так же, суммарная ошибка всех примеров обучающей выборки. Если после прохождения нескольких циклов она равна нулю или малому значению, заданному при создании сети, обучение считается законченным, в противном случае циклы повторяются. Число циклов обучения (эпох), также как и время полного обучения, зависят от многих факторов - размера обучающей выборки, числа входных параметров, типа и параметров нейросети и даже от случайного расклада весов синапсов при инициализации сети.

Работа сети в «Simulink»показа на рисунке Рис.4.27

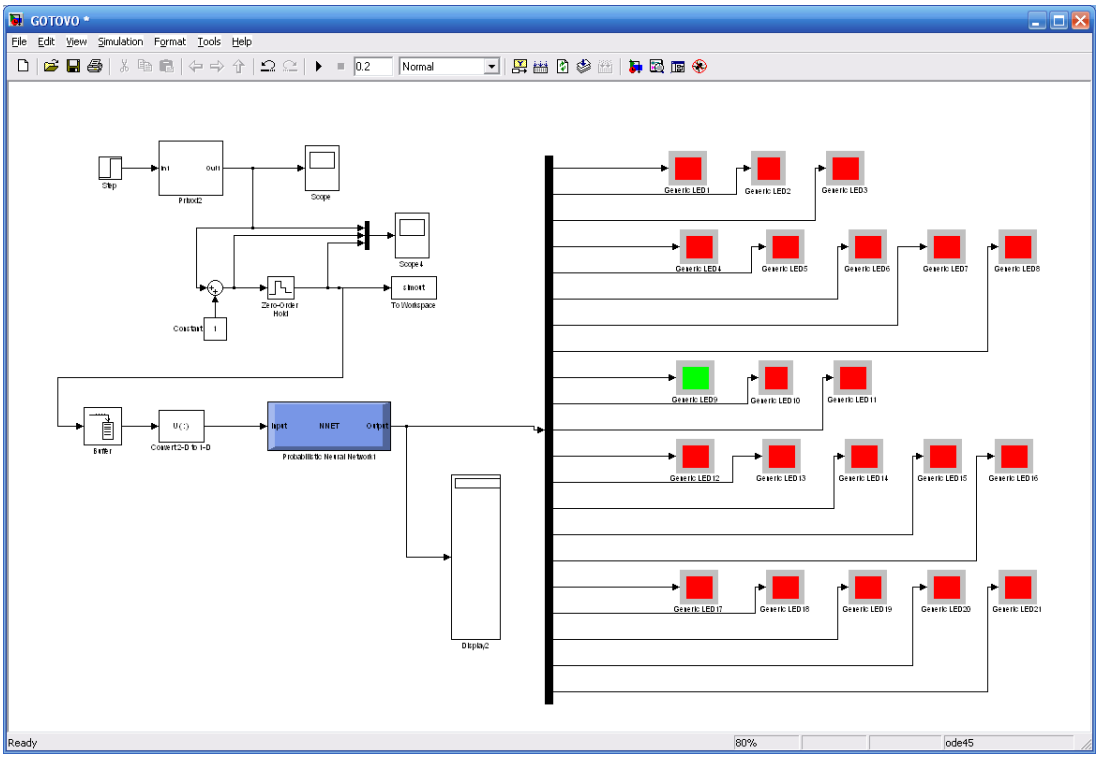


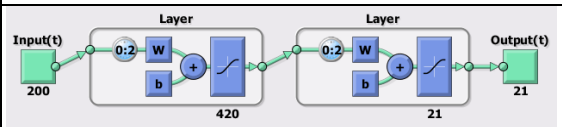
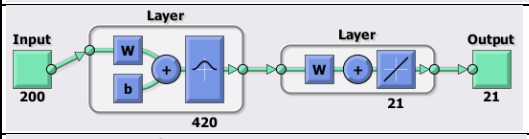
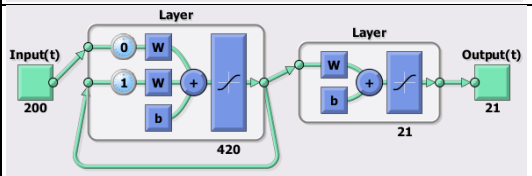
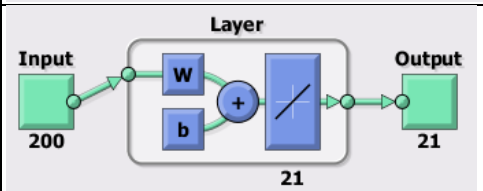
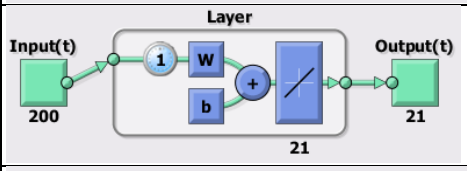
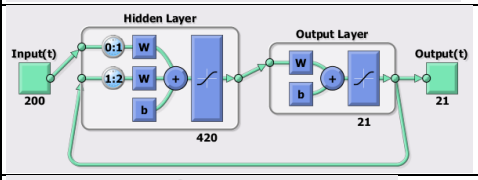
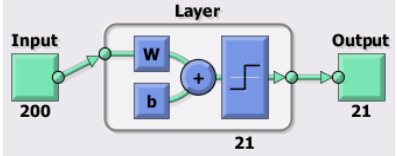
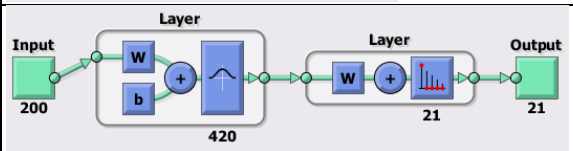
Рис.4.27. Система в работе в «Simulink»

Аналогичные исследования проведем для сетей Linear layer(design), Linear layer (train), NARX, Perceptron.

Результаты работы по поиску наиболее подходящей сети представлены в таблице 2.

Таблица 4.2

Архитектуры нейронных сетей

Тип нейронной сети	Архитектура сети	Число эпох обучения	Точность обучения	Время обучения
Feed-forward distributed time delay		-	-	3 с
Generalized Regression		-	-	3 с
Layerrecurrent		6	0,317	∞
Linear layer(design)		-	-	3 с
Linear layer (train)		173	$1x10^{-6}$	20 мин
NARX		-	-	3 с
Perceptron		221	0,967	5:33 мин
Probabilistic		-	-	3 с

После перебора всех типов нейросетей, была найдена сеть, наиболее точно воспроизводящая заданные зависимости. Ей оказалась вероятностная сеть **Probabilistic Neural Network** одна из разновидностей радиально базисных сетей.

Вероятностная нейронная сеть (PNN – Probabilistic Neural Network) представляет собой параллельную реализацию статистических методов Байеса. В PNN образцы классифицируются на основе оценок их близости к соседним образцам. При этом используется ряд критериев статистических методов, на основе которых принимается решение о том, к какому классу отнести еще не классифицированный образец. Формальным правилом при классификации является то, что класс с наиболее плотным распределением в области неизвестного образца, а также с более высокой априорной вероятностью и с более высокой ценой ошибки классификации, будет иметь преимущество по сравнению с другими классами.

Probabilistic Neural Network- нейронная сеть, которая формирует на выходе оценку вероятности принадлежности объекта к определенному классу. В процессе обучения она приобретает способность оценивать функцию распределения, а ее выход рассматривается как ожидаемое значение модели в определенной точке пространства входов.

Нейросеть работает по следующему циклу.

1. Вектор входных сигналов распространяется по связям между нейронами (прямое функционирование).
2. Измеряются сигналы, выданные выходными нейронами.
3. Производится анализ выданных сигналов, и вычисляется оценка (разница между выданным сетью ответом и требуемым ответом, имеющимся в примере). Чем меньше значение разности, тем лучше распознан пример, тем ближе к требуемому выданный сетью ответ. Разница, равная нулю, означает, что требуемое соответствие вычисленного и известного (целевого) ответов достигнуто и больше ничего не предпринимается.
4. В противном случае на основании числового значения разности вычисляются поправочные коэффициенты для каждого синаптического веса матрицы связей, после чего производится подстройка синаптических весов (обратное функционирование). В коррекции весов синапсов и заключается обучение.
5. Осуществляется переход к следующему примеру и все вышеперечисленные операции повторяются. Проход по всем приме-

рам обучающей выборки с первого по последний считается одним циклом обучения.

Для каждого примера в MATLAB вычисляется ошибка обучения. Вычисляется, так же, суммарная ошибка всех примеров обучающей выборки. Если после прохождения нескольких циклов она равна нулю или малому значению, заданному при создании сети, обучение считается законченным, в противном случае циклы повторяются. Число циклов обучения (эпох), также как и время полного обучения, зависят от многих факторов - размера обучающей выборки, числа входных параметров, типа и параметров нейросети и даже от случайного расклада весов синапсов при инициализации сети.

В соответствии с этим правилом для двух классов *A* и *B* считают, что элемент x принадлежит классу A, если:

$$hA \cdot cA \cdot fA(x) > hB \cdot cB \cdot fB(x),$$

где *h* – априорная вероятность; c – цена ошибки классификации; *f*(*x*) – функция плотности вероятностей. Оценки стоимости ошибки классификации и априорной вероятности предполагает хорошее знание решаемой задачи и в базовой модели сети не используются (считаются одинаковыми).

В общем случае под радиальной базисной нейронной сетью («RadialBasisFunctionNetwork», сеть «RBF») понимается двухслойная сеть без обратных связей, которая содержит скрытый слой радиально симметричных нейронов.

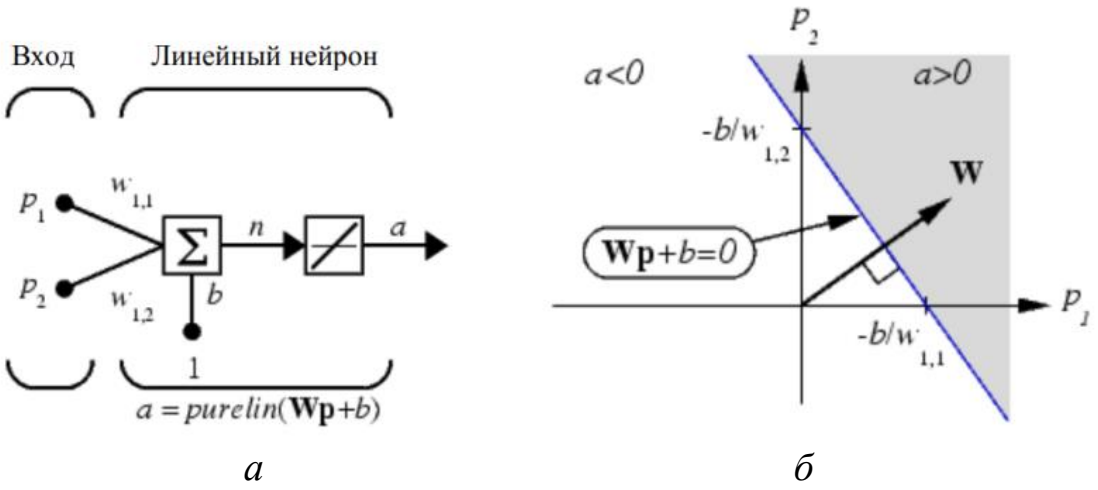
Радиальные базисные нейронные сети состоят из большего количества нейронов, чем стандартные сети с прямой передачей сигналов и обучением методом обратного распространения ошибки, но на их создание требуется значительно меньше времени. Эти сети наиболее эффективны, когда доступно большое количество обучающих векторов.

4.4. Линейные и радиально-базисные сети

Линейные сети

Линейные сети по своей структуре аналогичны персептрону и отличаются лишь функцией активации. Выход линейной сети может принимать любое значение, в то время как выход персептрона ограничен значениями 0 или 1. На рис. 4.28 показан линейный нейрон с

двумя входами. Он имеет структуру, сходную со структурой персептрона. Единственное отличие состоит в том, что используется линейная функция активации purelin.



а *б*

Рис. 4.28. Линейная сеть: а – структура, б – разделяющая линия

Весовая матрица W в этом случае имеет только одну строку, и выход сети определяется выражением

a = purelin(n) = purelin(Wp+b) = Wp+b = w11p1 + w12 p2 +b. (1)

Подобно персептрону, линейная сеть задает в пространстве входов разделяющую линию, на которой функция активации n равна 0 (рис. 4.28б).

Векторы входа, расположенные выше этой линии, соответствуют положительным значениям выхода, а расположенные ниже – отрицательным. Это означает, что линейная сеть может быть применена для решения задач классификации. Однако такая классификация может быть выполнена только для класса линейно отделимых объектов. Таким образом, линейные сети имеют то же самое ограничение, что и персептрон.

Архитектура сети

Линейная сеть, показанная на рис. 4.29а, включает S нейронов, размещенных в одном слое и связанных с R входами через матрицу весов W. На рис. 4.29,б показана укрупненная структурная схема этой сети, вектор выхода а которой имеет размер S×1.

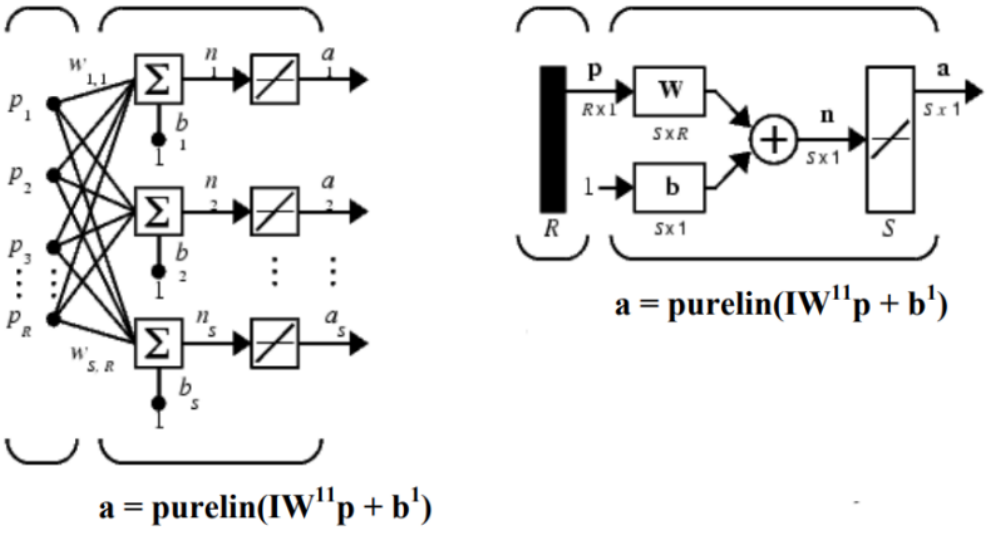


Рис. 4.29. Архитектура линейной сети: а – линейная сеть, б – укрупненная линейная сеть

Создание модели линейной сети

Линейную сеть с одним нейроном, показанную на рис. 1, можно создать следующим образом:

```
clear
net = newlin([-1 1; -1 1],1);
```

Первый входной аргумент задает диапазон изменения элементов вектора входа; второй аргумент указывает, что сеть имеет единственный выход. Начальные веса и смещение по умолчанию равны нулю. Присвоим весам и смещению следующие значения:

```
net.IW{1,1} = [2 3];
net.b{1} =[-4]
```

Теперь можно промоделировать линейную сеть для следующего предъявленного вектора входа:

```
p = [5;6];
a = sim(net,p)
a =
        24
```

Видно, что сеть правильно классифицировала входной вектор.

Линейные сети, как и персептроны, способны решать только линейно отделимые задачи классификации, однако в них используется другое правило обучения, основанное на методе обучения наименьших квадратов, которое является более мощным, чем правило обучения персептрона. Для заданной линейной сети и соответствующего множества векторов входа и целей можно вычислить вектор выхода сети и сформировать разность между вектором выхода и целевым вектором, которая определит некоторую погрешность. В процессе обучения сети требуется найти такие значения весов и смещений, чтобы сумма квадратов соответствующих погрешностей была минимальной, поэтому настройка параметров выполняется таким образом, чтобы обеспечить минимум ошибки. Эта задача разрешима, так как для линейной сети поверхность ошибки как функция входов имеет единственный минимум, и отыскание этого минимума не вызывает трудностей. Как и для персептрона, для линейной сети применяется процедура обучения с учителем, которая использует обучающее множество вида {p1, t1}, {p2, t2},…,{pQ, tQ}. (1) Требуется минимизировать одну из следующих функций квадратичной ошибки:

$$mse = \frac{1}{Q}\sum_{k=1}^{Q} e(k)^2 = \frac{1}{Q}\sum_{k=1}^{Q} (t(k) - a(k))^2$$

или

$$sse = \sum_{k=1}^{Q} e(k)^2 = \sum_{k=1}^{Q} (t(k) - a(k))^2,$$

где mse – средняя квадратичная ошибка; sse – сумма квадратов ошибок.

Процедура настройки в отличие от многих других сетей настройка линейной сети для заданного обучающего множества может быть выполнена посредством прямого расчета с использованием newlind, т. е. можно построить поверхность ошибки и найти на этой поверхности точку минимума, которая будет соответствовать оптимальным весам и смещениям для данной сети. Проиллюстрируем это на следующем примере. Предположим, что заданы следующие векторы, принадлежащие обучающему множеству.

```
clear
P = [1 -1.2]; T = [0.5 1];
```

Структурная схема этого линейного нейрона представлена на рис. 4.30.

Запишем уравнение выхода нейрона

a = purelin(n) = purelin(wp+b) =wp+b

Графическая интерпретация настройки веса и смещения для данного нейрона при двух обучающих множеств сводится к построению прямой, проходящей через две заданные точки и представлена на рис. 4.30.

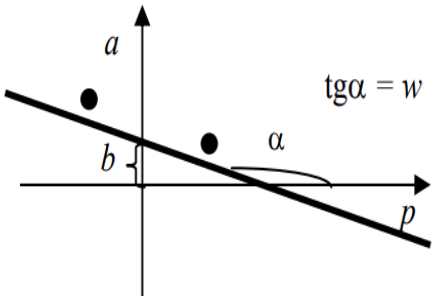


Рис.4.30. Графическая интерпретация

Построим линейную сеть и промоделируем ее:

```
net = newlind(P,T);
Y = sim(net, P)
Y =
            0.5000   1.0000
net.IW{1,1}
 ans =
            -0.2273
net.b
ans =
            [0.7273]
```

Выход сети соответствует целевому вектору, т. е. оптимальными весом и смещением нейрона будут

```
 w =
       –0,2273;
b =
            0,7273.
```

Зададим следующий диапазон весов и смещений:

```
w_range=-1:0.1: 0;
b_range=0.5:0.1:1;
```

Рассчитаем критерий качества обучения

```
ES = errsurf(P,T, w_range, b_range, 'purelin');
```

Построим линии уровня поверхности функции критерия качества обучения в пространстве параметров сети (рис. 4.31):

```
contour(w_range, b_range,ES,20)
hold on
plot(-2.273e-001,7.273e-001, 'x')
hold off
```

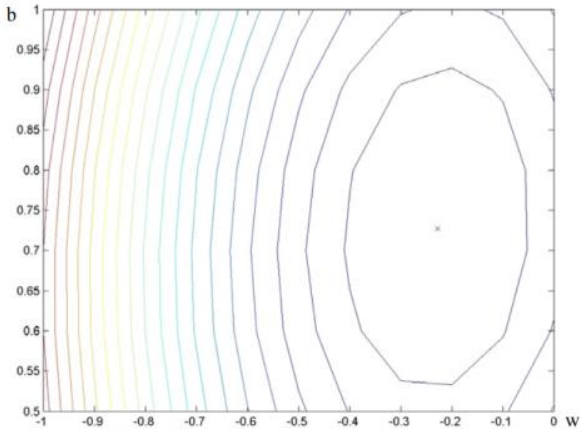


Рис.4.31. Критерия качества обучения

На графике знаком "x" отмечены оптимальные значения веса и смещения для данной сети.

Задача классификации векторов

Линейные сети могут быть применены для решения задач классификации. Если используется процедура обучения train, то параметры сети настраиваются с учетом суммарного значения функции ошибки. Это отличается от процедуры адаптации adapt, для работы которой характерна настройка параметров с учетом ошибки при представлении каждого вектора входа. Затем обучение применяется к скорректированной сети, вычисляются выходы, сравниваются с соответствующими целями и вновь вычисляется ошибка обучения. Если достигнута допустимая погрешность или превышено максимальное число циклов (эпох) обучения, то процедура настройки прекращается. Алгоритм обучения и настройки сходится, если задача классификации разрешима. Проиллюстрируем решение задачи классификации, ранее

решенной с помощью персептрона. Используем для этого простейшую линейную сеть, представленную на рисунке. Обучающее множество представлено следующими четырьмя парами векторов входов и целей:

$$\left\{\mathbf{p}_1=\begin{bmatrix}2\\2\end{bmatrix},t_1=0\right\}\left\{\mathbf{p}_2=\begin{bmatrix}1\\-2\end{bmatrix},t_2=1\right\}\left\{\mathbf{p}_3=\begin{bmatrix}-2\\2\end{bmatrix},t_3=0\right\}\left\{\mathbf{p}_4=\begin{bmatrix}-1\\1\end{bmatrix},t_4=1\right\}$$

Определим линейную сеть с начальными значениями веса и смещения, используемыми по умолчанию, т. е. нулевыми; зададим допустимую погрешность обучения, равную 0.1 (Рис.4.32).

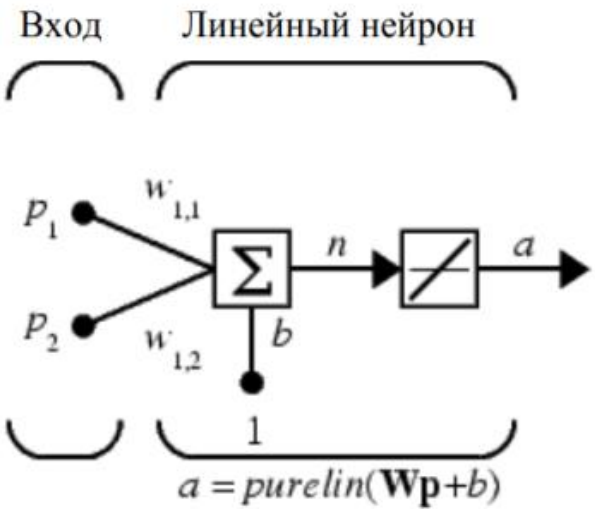


Рис.4.32. Моделируемая сеть

```
clear
p = [2 1 -2 -1;2 -2 2 1]; t = [0 1 0 1];
net = newlin( [-2 2; -2 2],1); % Инициализация линейной сети с двумя входами и одним выходом
net.trainParam.goal= 0.1;
 [net, tr] = train(net,p,t);
TRAINB, Epoch 0/100, MSE 0.5/0.1.
TRAINB, Epoch 25/100, MSE 0.181122/0.1.
TRAINB, Epoch 50/100, MSE 0.111233/0.1.
TRAINB, Epoch 64/100, MSE 0.0999066/0.1.
TRAINB, Performance goal met.
```

Пороговое значение функции качества достигается за 64 цикла обучения, а соответствующие параметры сети принимают значения:

```
weights = net.iw{1,1},
```

```
bias = net.b(1)
weights =
        -0.0615     -0.2194
bias =
        [0.5899]
```

Выполним моделирование созданной сети с векторами входа из обучающего множества и вычислим ошибки сети:

```
A = sim(net, p)
err = t - sim(net,p)
A =
        0.0282   0.9672   0.2741   0.4320
 err =
        -0.0282   0.0328    -0.2741    0.56800
```

Заметим, что погрешности сети весьма значительны. Попытка задать большую точность в данном случае не приводит к цели, поскольку возможности линейной сети ограничены. Демонстрационный пример demolin4 иллюстрирует проблему линейной зависимости векторов, которая свойственна и этому случаю.

Адаптируемые линейные сети ADALINE

На рис. 1 представлена структурная схема цифрового фильтра, отличительной особенностью которого является то, что он включает динамический компонент – линию задержки (ЛЗ) и 1 слой линейной нейронной сети. Последовательность значений входного сигнала {p(k)} поступает на ЛЗ, состоящую из N–1 блока запаздывания; выход ЛЗ – N-мерный вектор pd, составленный из значений входа в моменты времени k, k–1,…, k–N–1. Выход линейного нейронного слоя и фильтра в целом описывается следующим динамическим соотношением:

$$a(k) = \text{purelin}(\mathbf{W}\mathbf{p} + b) = \sum_{i=1}^{R} w_{1i}\, p(k - i + 1) + b\ .$$

Рассмотрим конкретный пример цифрового фильтра, представленный на рис. 4.33.

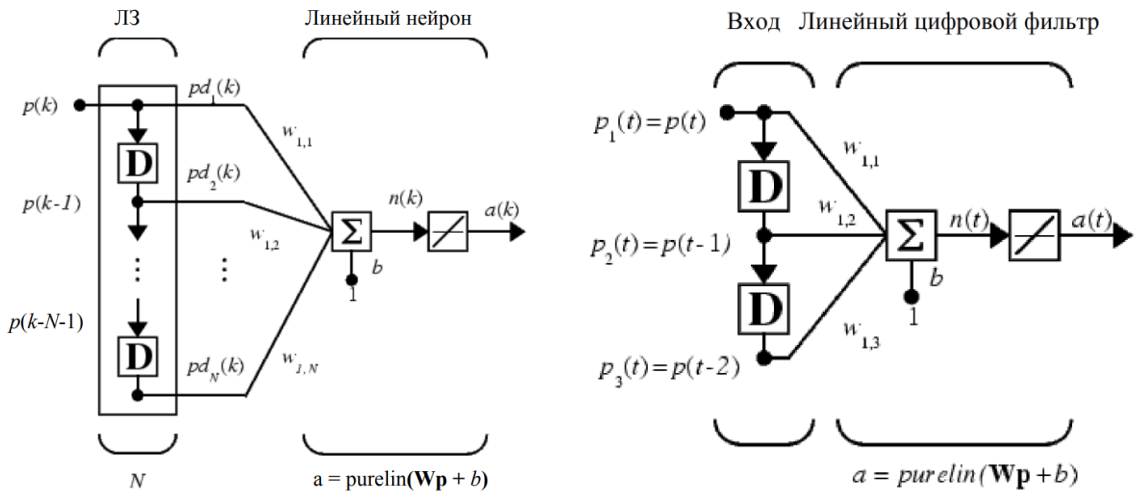


Рис. 4.33. Цифровой фильтр

Предположим, что входной сигнал принимает значения в диапазоне от 0 до 10, и сформируем линейную сеть с одним входом и одним выходом, используя функцию newlin:

```
clear
net = newlin([0,10],1);
```

Введем ЛЗ с двумя тактами запаздывания:

```
net.inputWeights{1,1}.
delays = [0 1 2];
```

определим следующие начальные значения весов и смещения:

```
net.IW{1,1} = [7 8 9];
net.b{1} = [0];
```

зададим начальные условия для динамических блоков линии задержки:

```
pi ={1 2}
pi =
        [1]    [2]
```

Последовательность их задания слева направо соответствует блокам запаздывания, расположенным на рисунке сверху вниз. Этим завершается формирование сети. Теперь определим входной сигнал в следующем порядке значений:

```
p = {3 4 5 6}
p =
        [3]   [4]   [5]   [6]
```
и промоделируем эту сеть:
```
[a,pf] = sim(net,p,pi)
a =
        [46]   [70]   [94]   [118]
pf =
        [5]   [6]
```
Для того чтобы получить желаемую последовательность сигналов на выходе, необходимо выполнить настройку сформированной сети. Предположим, что задана следующая желаемая последовательность для выхода фильтра:
```
T = {10 20 30 40};
```
Выполним настройку параметров, используя процедуру адаптации adapt и 10 циклов обучения:

```
net.adaptParam.passes = 10;
[net,y,E pf,af] = adapt(net,p,T,pi); % Процедура адаптации
```
Выведем полученные значения весов, смещения и выходного сигнала:
```
wts = net.IW{1,1},
bias = net.b{1}, y
wts =
        0.5059   3.1053    5.7046;
 bias =
         -1.5993
 y =
        [11.8558]   [20.7735]   [29.6679]   [39.0036]
```
Если продолжить процедуру настройки, то можно еще точнее приблизить выходной сигнал к желаемому:

```
 net.adaptParam.passes = 500;
 [net,y,E,pf,af] = adapt(net,p,T,pi);  y
y =
        [10.0043]   [20.0018]   [29.9992]   [39.9977]
```
Таким образом, линейные динамические нейронные сети могут быть адаптированы для решения задач фильтрации временных сигна-

лов. Для сетей такого класса часто используется название ADALINE (ADaptive LInear NEtwork) – адаптируемые линейные сети.

Радиальные базисные сети и их архитектура

В общем случае под радиальной базисной нейронной сетью (Radial Basis Function Network, сеть RBF) понимается двухслойная сеть без обратных связей, которая содержит скрытый слой радиально симметричных нейронов. Радиальные базисные нейронные сети состоят из большего количества нейронов, чем стандартные сети с прямой передачей сигналов и обучением методом обратного распространения ошибки, но на их создание требуется значительно меньше времени. Эти сети наиболее эффективны, когда доступно большое количество обучающих векторов. Модель нейрона и архитектура сети На рис.4.34 показан радиальный базисный нейрон с R входами. Функция активации для радиального базисного нейрона имеет вид:

a = radbas(n) =$^{-}$ e^{-n^2}

Вход функции активации определяется как модуль разности вектора весов w и вектора входа p, умноженный на смещение b. Функция активации для радиального базисного нейрона имеет вид

radbas(n) =$^{-}$ e^{-n^2}

при n = 0, a = $e^{-0^2} = 1$;

при n = 0,833, a =$e^{-0{,}833^2} = 0{,}5$;

при n = –0,833, a= $e^{-(-0{,}833)^2} = 0{,}5$.

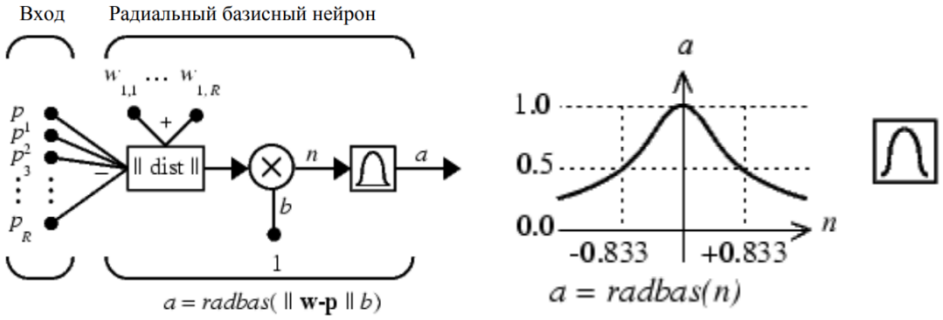


Рис.4.34. Радиально базисный нейрон

Вход функции активации определяется как модуль разности вектора весов w и вектора входа p, умноженный на смещение b: n= ||p – w||b. Радиальный базисный нейрон формирует значение 1, когда

вход p идентичен вектору весов w. Смещение b позволяет корректировать чувствительность нейрона radbas.

Например, если нейрон имел смещение 0.1, то его выходом будет 0.5 для любого вектора входа p и вектора веса w при расстоянии между векторами, равном 8.333, или 0.833/b. Следовательно, при помощи b можно масштабировать разность p и w. Это позволяет решать ряд определенных задач. Например, пусть дана разность p – w = 20 и a = 0,5. Требуется найти такое b, чтобы данное условие выполнялось. Решим эту задачу:

0,5 = radbas(20b) =$^{-}$ e^{-n^2}

0,5= $e^{-(20b)^2}$

ln 0,5= ln $e^{-(20b)^2}$

ln 0,5 =–$(20b)^2$

Вычислим ln 0.5 и запишем:

$-(20b)^2 = -0{,}693$

$(20b)^2 = 0{,}693$

$(400b)^2 = 0{,}693$

b = 0,0416.

Радиальная базисная сеть состоит из двух слоев: скрытого радиального базисного слоя, имеющего S^1 нейронов, и выходного линейного слоя, имеющего S^2 нейронов (рис. 4.35). Входами блока ||dist|| на этом рисунке являются вектор входа p и матрица весов $IW^{1,1}$, а выходом – вектор, состоящий из S^1 элементов. Выход блока ||dist|| умножается поэлементно на вектор смещения b^1 и формирует вход функции активации.

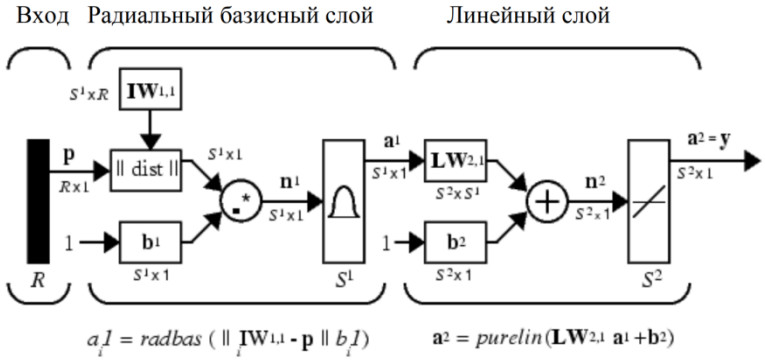


Рис.4.35. Радиальная базисная сеть

Все операции, связанные с созданием радиальной базисной сети в системе MATLAB, оформлены в виде специальных функций newrbe и newrb. Первая позволяет построить радиальную базисную сеть с ну-

левой ошибкой, вторая позволяет управлять количеством нейронов в скрытом слое.

Задачи аппроксимации

Пусть задана функция f(x) графически без аналитического описания (рис.4.36.)

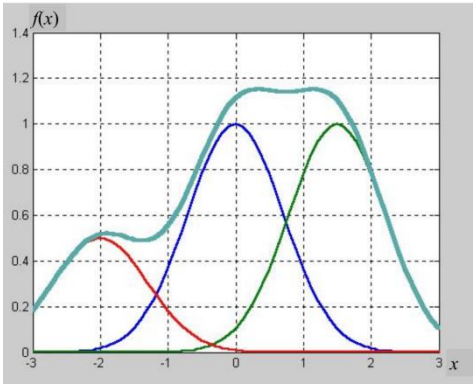


Рис.4.36 Функция f(x)

Необходимо подобрать такую аналитическую зависимость, которая с заданной точностью описывала бы эту функциональную зависимость. В данном случае используется сумма радиальных базисных функций:

$$f(x) = \sum^{N}_{i=1} \alpha_i \varphi_i (x),$$

где $\varphi_i (x)$ – радиальная базисная функция. Идея аппроксимации может быть представлена графически следующим образом. Рассмотрим взвешенную сумму трех радиальных базисных функций, заданных на интервале [–3; 3].

```
clear,
p = -3:.1:3;
a1 = radbas(p);
a2 = radbas(p - 1.5);
a3 = radbas(p + 2);
a = a1 + a2*1 + a3*0.5;
figure(1), clf,
plot(p,a1,p,a2,p,a3*0.5,'LineWidth',1.5),
hold on,
plot(p,a,'LineWidth',3,'Color',[1/3,2/3,2/3]),
grid on,
legend('a1','a2','a3*0.5','a')
```

Как следует из рис.4.36, при задании вектора входа каждый нейрон радиального базисного слоя выдаст значение в соответствии с тем, как близок вектор входа к вектору весов каждого нейрона.

Радиальные базисные нейроны с векторами весов, значительно отличающимися от вектора входа p, будут иметь выходы, близкие к 0, и их влияние на выходы линейных нейронов будет незначительно. Напротив, радиальный базисный нейрон с вектором весов, близким к вектору входа p, выдаст значение, близкое к 1, и это значение будет передано на линейный нейрон с весом, соответствующим выходному слою. Итак, если только один радиальный базисный нейрон имеет выход 1, а все другие имеют выходы, равные или очень близкие к 0, то выход линейного слоя будет равен весу активного выходного нейрона. Однако это исключительный случай. Обычно выход формируют несколько нейронов с разными значениями весов. Согласно рис. 4.36, выполним ручной расчет для анализа a1, a2, a3.

p = 2; w1 = −2; w2 = 0; w3 =1,5; b1 = b2 = b3 =1,

$||p - w_1||$ b_1=|2-(-2)|∗1= 4

$||p - w_2||$ b_2=|2-0|∗1= 2

$||p - w_3||$ b_3=|2-(1.5)|∗1= 0,5

radbas(n_1) =$^{-}$ e^{-4^2}=e^{-16}=0

radbas(n_2) =$^{-}$ e^{-2^2}=e^{-4}=0.000001

radbas(n_3) =$^{-}$ $e^{-0.5^2}$=$e^{-0.25} = 0.8$

Как видно из рис. 4.36, в точке p = 2 суммируется практически только выход третьего нейрона. Влияние в этой точке первого и второго нейронов незначительно. Выходы a1 и a2 близки к 0. Выход a3 = 0,8 близок к 1. Анализируя рис. 4.36, приходим также к выводу, что разложение по радиальным базисным функциям обеспечивает необходимую гладкость. Поэтому их применение для аппроксимации произвольных нелинейных зависимостей вполне оправдано. Рассмотрим пример формирования радиальной базисной сети в системе MATLAB для решения задачи аппроксимации. Сформируем обучающее множество и зададим допустимое значение функционала ошибки, равное 0.01, параметр влияния определим равным 1 и будем использовать итерационную процедуру формирования радиальной базисной сети:

```
P = -1:.1:1;
T = [-.9602 -.5770 -.0729 .3771 .6405 .6600 .4609 .1336 ...
 -.2013 -.4344 -.5000 -.3930 -.1647 .0988 .3072 .3960 ...
```

```
.3449 .1816 -.0312 -.2189 -.3201];
GOAL = 0.01;
SPREAD = 1;
net = newrb(P,T,GOAL,SPREAD); % Создание сети
net.layers{1}.size % Число нейронов в скрытом слое
NEWRB, neurons = 0,
SSE = 3.69051
 ans =
     6
```

Для заданных параметров нейронная сеть состоит из 6 нейронов и обеспечивает следующие возможности аппроксимации нелинейных зависимостей после обучения. Моделируя сформированную нейронную сеть, построим аппроксимационную кривую на интервале [–1; 1] с шагом 0.01 для нелинейной зависимости.

```
figure(1),
clf,
plot(P,T,'sr','MarkerSize',8,'MarkerFaceColor'
,'y')
hold on;
X = -1:.01:1;
Y = sim(net,X); % Моделирование сети
plot(X,Y,'LineWidth',2),
grid on
```

Из анализа рис. 4.37 следует, что при небольшом количестве нейронов скрытого слоя радиальная базисная сеть достаточно хорошо аппроксимирует нелинейную зависимость, заданную обучающим множеством из 21 точки.

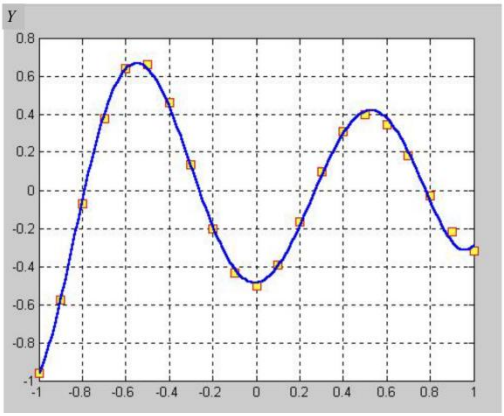


Рис.4.37. Результат работы сети

Сети PNN

Нейронные сети PNN (Probabilistic Neural Network) предназначены для решения вероятностных задач и, в частности, задач классификации. Архитектура сети Архитектура сети PNN базируется на архитектуре радиальной базисной сети, но в качестве второго слоя использует так называемый конкурирующий слой, который подсчитывает вероятность принадлежности входного вектора к тому или иному классу и в конечном счете сопоставляет вектор с тем классом, вероятность принадлежности к которому выше. Структура сети PNN представлена на рис. 1.

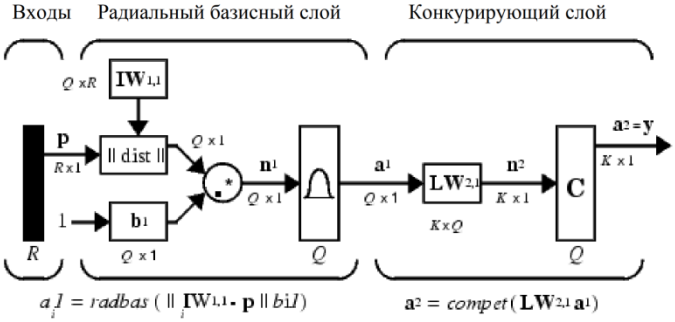


Рис.4.38. Структура сети PNN

Предполагается, что задано обучающее множество, состоящее из Q пар векторов вход/цель. Каждый вектор цели имеет K элементов, указывающих класс принадлежности, и, таким образом, каждый вектор входа ставится в соответствие одному из K классов. В результате может быть образована матрица связности T размера K×Q, состоящая из нулей и единиц, строки которой соответствуют классам принадлежности, а столбцы – векторам входа. Таким образом, если элемент T(i,j) матрицы связности равен 1, то это означает, что j-й входной вектор принадлежит к классу i. Весовая матрица первого слоя IW11 (net.IW{1,1}) формируется с использованием векторов входа из обучающего множества в виде матрицы Pт . Когда подается новый вход, блок ||dist|| вычисляет близость нового вектора к векторам обучающего множества; затем вычисленные расстояния умножаются на смещения и подаются на вход функции активации radbas. Вектор обучающего множества, наиболее близкий к вектору входа, будет представлен в векторе выхода a1 числом близким к 1. Весовая матрица второго слоя LW21 (net.LW{2,1}) соответствует матрице связности T, построенной для данной обучающей последовательности. Эта операция

может быть выполнена с помощью функции ind2vec, которая преобразует вектор целей в матрицу связности T. Произведение T*a1 определяет элементы вектора a1, соответствующие каждому из K классов. В результате конкурирующая функция активации второго слоя compet формирует на выходе значение, равное 1 для самого большего по величине элемента вектора n2 и 0 в остальных случаях. Таким образом, сеть PNN выполняет классификацию векторов входа по K классам.

Синтез сети

Для создания нейронной сети PNN предназначена функция newpnn. Определим 7 следующих векторов входа и соотнесем каждый из них к одному из 3 классов:

P = [0 0;1 1;0 3;1 4;3 1;4 1;4 3];

Tc = [1 1 2 2 3 3 3].

Ставится задача определения: к какому классу принадлежит данный вектор.

Запишем матрицу связности:

$$T = \begin{matrix} 0\,0\,0\,0\,1\,1\,1 \\ 0\,0\,1\,1\,0\,0\,0 \\ 1\,1\,0\,0\,0\,0\,0 \end{matrix}$$

Первая строка – 1 класс; вторая – 2 класс; третья – 3 класс. В данном случае имеем двумерный вектор (функция от двух переменных). Графически эта функция представлена на рис. 2.

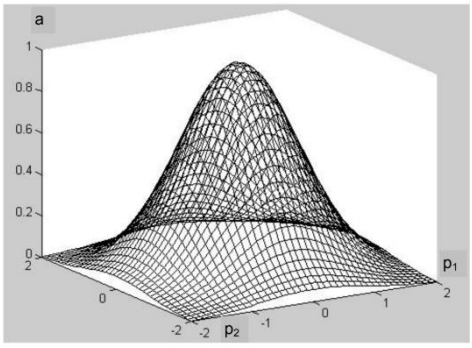


Рис.4.39. Поверхность отклика

Найдем выходы первого слоя по формуле

$a = e^{-(\sqrt{(p_1-w_1)^2+(p_2-w_2)^2}\,b)^2}$

Подадим на все нейроны первый обучающий вектор 0; p1 = p2 = 0.

Тогда на выходах имеем:

$a_1^1 = 1$, $a_2^1 = 0{,}25$, $a_3^1 = 002$, $a_4^1 = 0$, $a_5^1 = 0.001$, $a_6^1 = 0$, $a_7^1 = 0$,

$w_{11}^1 = 0$, $w_{12}^1 = 0{,}1$, $w_{13}^1 = 0$, $w_{14}^1 = 1$, $w_{15}^1 = 3$, $w_{16}^1 = 4$, $w_{17}^1 = 4$,

$w_{21}^1 = 0$, $w_{22}^1 = 1$, $w_{23}^1 = 3$, $w_{24}^1 = 4$, $w_{25}^1 = 1$, $w_{26}^1 = 1$, $w_{27}^1 = 3$,

Подадим на все нейроны второй обучающий вектор p1 = p2 = 0 .

Тогда на выходах имеем:

$$a_1^1 = e^{-(\sqrt{1-0)^2+(1-0)^2}\ b)^2}$$
$$a_2^1 = e^{-(\sqrt{1-1)^2+(1-1)^2}\ b)^2}$$
$$a_3^1 = e^{-(\sqrt{1-0)^2+(1-3)^2}\ b)^2}$$
$$a_4^1 = e^{-(\sqrt{1-1)^2+(1-4)^2}\ b)^2}$$
$$a_5^1 = e^{-(\sqrt{1-3)^2+(1-1)^2}\ b)^2}$$
$$a_6^1 = e^{-(\sqrt{1-4)^2+(1-1)^2}\ b)^2}$$
$$a_7^1 = e^{-(\sqrt{1-4)^2+(1-3)^2}\ b)^2}$$

Весам второго слоя присваивают значения разреженной матрицы

$w_{11}^2 = 1$, $w_{12}^2 = 1$, $w_{13}^2 = 0$, $w_{14}^2 = 0$, $w_{15}^2 = 0$, $w_{16}^2 = 0$, $w_{17}^2 = 0$,

$w_{21}^2 = 0$, $w_{22}^2 = 0$, $w_{23}^2 = 1$, $w_{24}^2 = 1$, $w_{25}^2 = 0$, $w_{26}^2 = 0$, $w_{27}^2 = 0$,

$w_{31}^3 = 0$, $w_{32}^3 = 0$, $w_{33}^3 = 0$, $w_{34}^3 = 0$, $w_{35}^3 = 1$, $w_{36}^3 = 1$, $w_{37}^3 = 1$,

Тогда на выходе 2-го слоя имеем:

$a^2 = \text{compet}(\text{LW}^{2,1}a^1)$;

$$\begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} & W_{15} & W_{16} & W_{17} \\ W_{21} & W_{22} & W_{23} & W_{24} & W_{25} & W_{26} & W_{27} \\ W_{31} & W_{32} & W_{33} & W_{34} & W_{35} & W_{36} & W_{37} \end{bmatrix} \begin{bmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \\ a_4^1 \\ a_5^1 \\ a_6^1 \\ a_7^1 \end{bmatrix}$$

Подставляем значения

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0.25 \\ 0.02 \\ 0 \\ 0.001 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1.25 \\ 0.02 \\ 0.001 \end{bmatrix}$$

получаем

$$a^2 = \text{compet}\begin{bmatrix} 1.25 \\ 0.02 \\ 0.001 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Итак, данный вектор принадлежит к 1-му классу. На этом ручной расчет закончен. Перейдем к созданию и исследованию модели в системе MATLAB:

```
clear
P = [0 0;1 1;0 3;1 4;3 1;4 1;4 3]';
Tc = [1 1 2 2 3 3 3];
```

Вектору Tc поставим в соответствие матрицу связности T в виде разреженной матрицы вида

```
T = ind2vec(Tc)
T =
(1,1)        1
(1,2)        1
(2,3)        1
(2,4)        1
(3,5)        1
(3,6)        1
(3,7)        1
```

которая определяет принадлежность первых двух векторов классу 1, двух последующих – классу 2 и трех последних – классу 3. Полная матрица T имеет вид

```
T = full(T)
T =
     1  1  0  0  0  0  0
     0  0  1  1  0  0  0
     0  0  0  0  1  1  1
```

Массивы P и T задают обучающее множество, что позволяет выполнить формирование сети, промоделировать ее, используя массив входов P, и удостовериться, что сеть правильно решает задачу классификации на элементах обучающего множества. В результате моделирования сети формируется матрица связности, соответствующая массиву векторов входа. Функция vec2ind предназначена для преобразования матрицы связности в индексный вектор:

```
net = newpnn(P,T);
net.layers{1}.size % Число нейронов в сети PNN
```

```
ans = 7
Y = sim(net,P);
Yc = vec2ind(Y);
Yc =
     1  1  2  2  3  3  3
```

Результат подтверждает правильность решения задачи классификации.

Теперь выполним классификацию некоторого набора произвольных векторов p, не принадлежащих обучающему множеству, используя ранее созданную сеть PNN:

```
p = [1 3; 0 1; 5 2]';
```

Осуществляя моделирование сети для этого набора векторов, получаем

```
a = sim(net,p);
ac = vec2ind(a)
ac =
     2 1 3
```

Рассмотрим пример создания и моделирования следующей сети PNN в системе MATLAB:

```
figure(1),
clf reset, drawnow
p1 = 0:.05:5; p2 = p1;
[P1,P2]=meshgrid(p1,p2);
pp = [P1(:) P2(:)];
aa = sim(net,pp');
aa = full(aa);
m = mesh(P1,P2,reshape(aa(1,:),length(p1),length(p2)));
set(m,'facecolor',[0.75 0.75 0.75],'LineStyle','none');
hold on,
view(3)
m = mesh(P1,P2,reshape(aa(2,:),length(p1),length(p2)));
set(m,'FaceColor',[0 1 0.5],'LineStyle','none');
m = mesh(P1,P2,reshape(aa(3,:),length(p1),length(p2)));
set(m,'FaceColor',[0 1 1],'LineStyle','none');
plot3(P(1,:),P(2,:),ones(size(P,2))+0.1,'.','MarkerSize',20)
plot3(p(1,:),p(2,:),1.1*ones(size(p,2)),'*','MarkerSize',10,..
.
 'Color',[1 0 0]),
hold off,
view(2)
```

Результаты классификации представлены на рисунке 4.40 и показывают, что три вектора, отмеченные звездочками, классифици-

руются сетью PNN, состоящей из семи нейронов, абсолютно правильно.

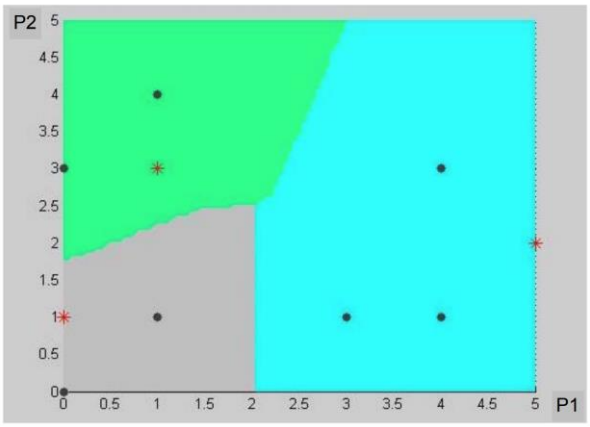


Рис.4.40. Результаты классификации

Итак, сети PNN могут весьма эффективно применяться для решения задач классификации. Недостатком сетей PNN, как и сетей GRNN, является то, что работают они относительно медленно, поскольку выполняют очень большие объемы вычислений по сравнению с другими типами нейронных сетей. Достоинство сетей PNN заключается в быстром обучении.

4.5. Создание нейросети с командной строки

Используя команды создания нейронных сетей выполним следующую последовательностей действий в командной строке.

```
>> P = [0 0; 1 1; 0 3; 1 4; 3 1; 4 1; 4 3]' %обучающая выборка
Tc = [1 1 2 2 3 3 3] %вектор цели

P =
     0     1     0     1     3     4     4
     0     1     3     4     1     1     3
Tc =
     1     1     2     2     3     3     3

T = ind2vec(Tc) %преобразование матрицы связности T в виде
%разреженной матрицы
  T =
    (1,1)        1
    (1,2)        1
```

```
    (2,3)        1
    (2,4)        1
    (3,5)        1
    (3,6)        1
    (3,7)        1
```

```
T = full(T) % Полная матрица
```

```
    T =
         1     1     0     0     0     0     0
         0     0     1     1     0     0     0
         0     0     0     0     1     1     1
```

```
net = newpnn(P,T)% создание сети
net.layers{1}.size % число нейронов в сети PNN
    net =
        Neural Network

                name: 'Probabilistic Neural Network'
            userdata: (your custom info)

        dimensions:

                numInputs: 1
                numLayers: 2
               numOutputs: 1
           numInputDelays: 0
           numLayerDelays: 0
        numFeedbackDelays: 0
        numWeightElements: 42
                sampleTime: 1

        connections:
            biasConnect: [1; 0]
           inputConnect: [1; 0]
           layerConnect: [0 0; 1 0]
          outputConnect: [0 1]
```

```
subobjects:
        input: Equivalent to inputs{1}
       output: Equivalent to outputs{2}

            inputs: {1x1 cell array of 1 input}
            layers: {2x1 cell array of 2 layers}
           outputs: {1x2 cell array of 1 output}
            biases: {2x1 cell array of 1 bias}
      inputWeights: {2x1 cell array of 1 weight}
      layerWeights: {2x2 cell array of 1 weight}

functions:

        adaptFcn: (none)
      adaptParam: (none)
        derivFcn: 'defaultderiv'
       divideFcn: (none)
     divideParam: (none)
      divideMode: 'sample'
          initFcn: 'initlay'
      performFcn: 'mse'
    performParam: .regularization, .normalization
         plotFcns: {}
      plotParams: {1x0 cell array of 0 params}
        trainFcn: (none)
      trainParam: (none)

weight and bias values:

          IW: {2x1 cell} containing 1 input weight matrix
          LW: {2x2 cell} containing 1 layer weight matrix
           b: {2x1 cell} containing 1 bias vector

methods:

          adapt: Learn while in continuous use
      configure: Configure inputs & outputs
        gensim: Generate Simulink model
```

```
                init: Initialize weights & biases
             perform: Calculate performance
                 sim: Evaluate network outputs given inputs
               train: Train network with examples
                view: View diagram
         unconfigure: Unconfigure inputs & outputs
          evaluate:       outputs = net(inputs)
ans =
     7
```

```
Y = sim(net,P) % обучение сети
Y =
     1     1     0     0     0     0     0
     0     0     1     1     0     0     0
     0     0     0     0     1     1     1
```

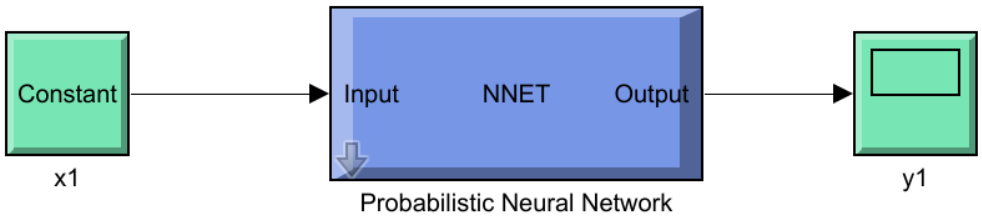
```
Yc = vec2ind(Y) % преобразования матрицы связности в
%индексный вектор
Yc =
     1     1     2     2     3     3     3
gensim(net) % генерация сети в виде пиктограммы.
ans =
    'untitled1'
```

В результате выполнения последней команды на экране появится схема нейронной сети (рис.4.41а).

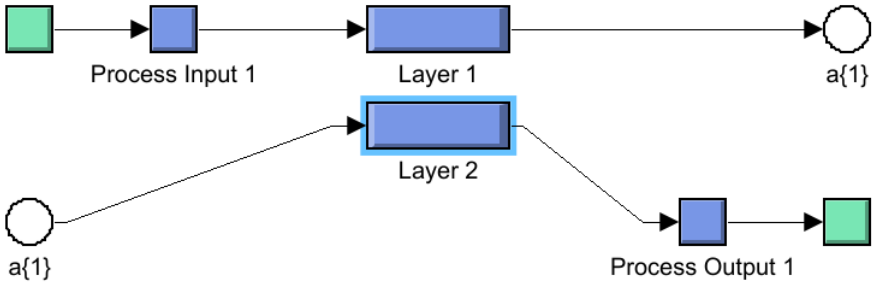
При наведении указателя манипулятора «мышь» на блок NNET и при нажатии на правую клавишу открывается контекстное меню, выбираем опцию Mask, а затем используя опцию Lookundermask можно просмотреть структуру НС. (рис.4.41б).

Полученная сеть содержит два уровня. Layer1 и Layer2. Выполним аналогичные манипуляции для просмотра структуры первого и второго слоев НС (рис.4.41в).

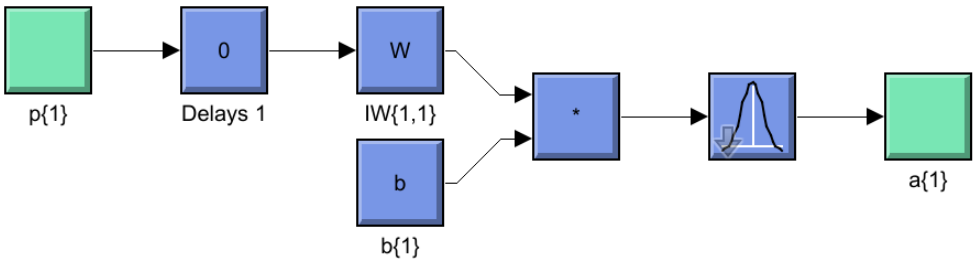
Входной слой Layer1 (рис.4.41в). Выходной слой Layer2 (рис.4.41г).



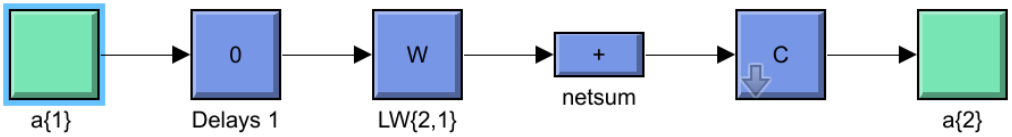
а



б



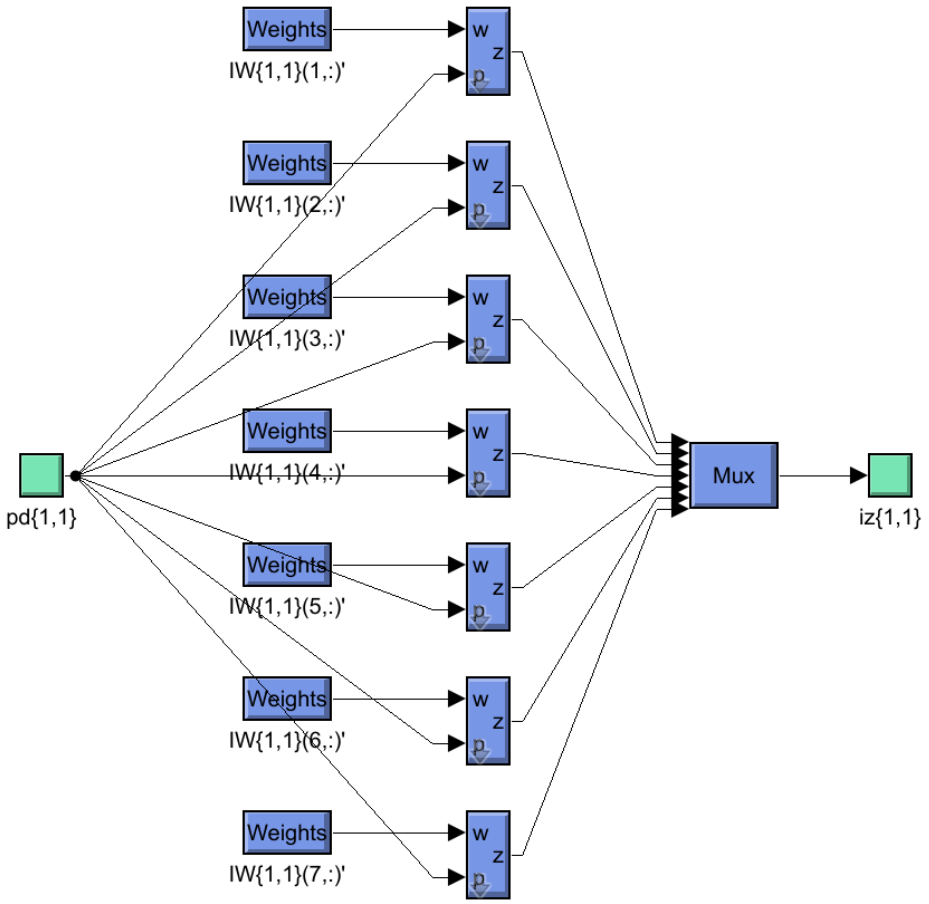
в



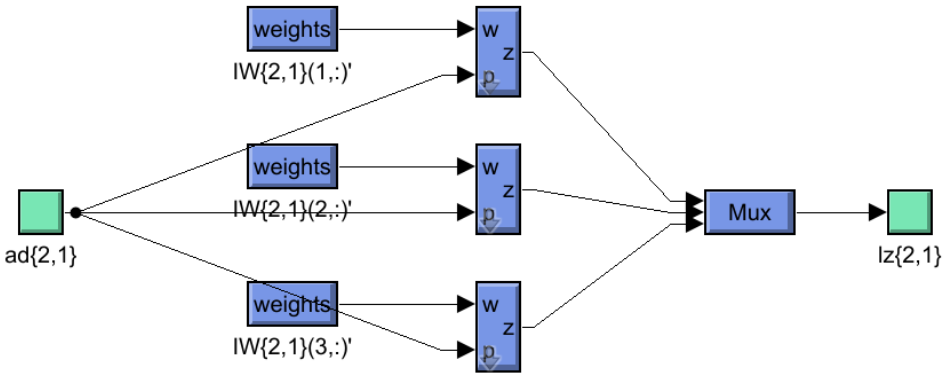
г

Рис.4.41 Структура сети:а - схема нейронной сети,
б - структуру НС, в - первый слой, г - второй слой

Просмотрим структуру входного слоя (рис.4.42а)



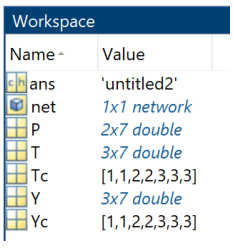
а



б

Рис.4.42. Структура слоев: а - входной слой, б - выходной слой

и выходного слоя (Рис.4.42б). После выполнения программы в рабочей области сформируется следующая структура данных.



4.6. Идентификация параметров с использованием нейросетей

Идентификация динамических объектов в основном состоит в определении структуры и параметров объекта по наблюдаемым данным. Функциональная схема системы идентификации представлена на рис.43.

Выходная величина объекта *y (n)* зависит как от внешнего воздействия и помехи, так и от неизвестного вектора параметров ***c****. Выходная величина настраиваемой модели $\hat{y}(n)$ зависит от вектора настраиваемых параметров ***c***, который пересчитывается в силу алгоритма, обрабатывающего вектор всех наблюдений *z (n).*

Разность выходных величин объекта и настраиваемой модели образует невязку $\varepsilon(z(n), \boldsymbol{c}) = y(n) - \hat{y}(n)$, которая поступает на вход функционального преобразователя. Далее всегда предполагается, что объект работает в стационарном режиме, т. е. вероятностные характеристики последовательностей *y (n), y (u),* а значит, и z(n) не зависят от момента времени *n.* Такой режим называется обычно режимом нормальной работы.

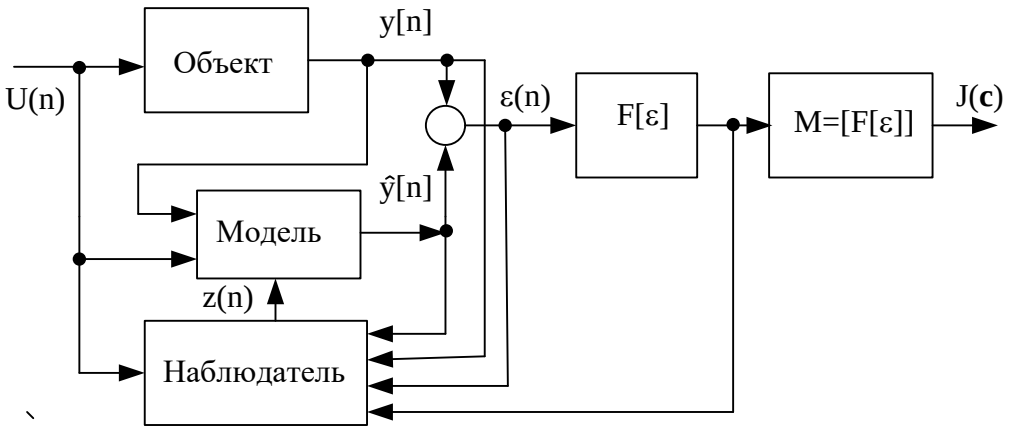


Рис.43. Функциональная схема системы идентификации

Соответствие настраиваемой модели объекту, т.е. качество идентификации, оценивается критерием качества идентификации
$J(c) = M\{F[\varepsilon(z(n), \boldsymbol{c})]\}$.
Здесь $F[.]$ — функция потерь, а M — символ математического ожидания.

Критерий качества идентификации представляет собой *средние потери.* Чем меньше средние потери, тем выше качество идентифи-

кации. Улучшение качества идентификации осуществляется надлежащим выбором структуры настраиваемой модели и изменением ее параметров. Это изменение осуществляется *алгоритмом, идентификации.*

Алгоритм идентификации определяется функцией потерь и структурой настраиваемой модели. По наблюдениям входного воздействия и выходных величин объекта и настраиваемой модели алгоритм идентификации изменяет параметры последней так, чтобы средние потери достигали минимума с ростом *n*. Эти условия соответствуют идентификации в режиме нормальной работы объекта.

Идентификацию можно осуществить иным способом, а именно: провести специальные эксперименты на объекте, затем полученные результаты этих экспериментов обработать, приближенно восстановить средние потери и далее тем или иным способом минимизировать восстановления средних потерь.

Для решения задачи идентификации, как это следует из функциональной схемы необходимо:

1) очертить класс объектов;

2) выбрать для этого класса объектов настраиваемую модель, параметры которой можно изменять;

3) выбрать критерий качества идентификации — средние потери, которые бы характеризовали различие между выходными величинами объекта и настраиваемой модели;

4) сформировать алгоритм идентификации, который используя доступные для наблюдения значения входных и выходных величин, изменял бы параметры настраиваемой модели так, чтобы средние потери с ростом *n* достигали минимума.

Динамические объекты описываются дифференциальными уравнениями (обыкновенными или в частных производных) с неизвестными коэффициентами, либо интегральными уравнениями с неизвестным ядром. Динамические объекты, управляемые цифровыми вычислительными машинами (ЦВМ), а также разнообразные процессы, характеризуемые временными рядами, описываются разностными уравнениями, которые являются дискретными аналогами дифференциальных уравнений, либо уравнениями типа свертки (с неизвестным ядром). Как правило, объекты подвержены воздействию помех, которые непосредственно не измеряются. Эти мешающие воздействия фи-

гурируют в уравнениях динамического объекта. Коэффициенты этих уравнений неизвестны.

На основании сведений об объекте формируется настраиваемая модель. Настраиваемая модель описывается уравнениями, подобными уравнениям объекта, либо соотношениями, содержащими измеряемые входные и выходные величины, характеризующие состояние динамического объекта. Коэффициенты этих уравнений или соотношений являются параметрами настраиваемой модели. Близость настраиваемой модели к динамическому объекту характеризуется функционалом невязки — средними потерями $J(\boldsymbol{c})$.

Одним из устройств системы идентификации является наблюдатель. На него возлагается роль оценки не измеряемых параметров, по выходной характеристике объекта y(n). Рассмотрим один из вариантов построения наблюдателя.

Пример реализации наблюдателя

В качестве объекта наблюдений будем рассматривать регулятор скорости привода. Для оценки коэффициента регуляторов построим имитационную модель привода регулируемого по скорости (рис.44). Будем полагать, что диапазон изменения параметров известен из моделирования. В качестве метода идентификации будем использовать теорию нейронных сетей. Необходимо выполнить три основных этапа. Собрать обучающие данные и на их основе выбрать нейронную сеть. Обучить нейронную сеть и выполнить моделирование.

Для сбора обучающих данных в привод добавлены устройства квантования сигнала, буфер памяти и преобразователь двухстрочного вектора в однострочный рис. . Собранные данные передаются в рабочую область с использованием блока «to workspace».

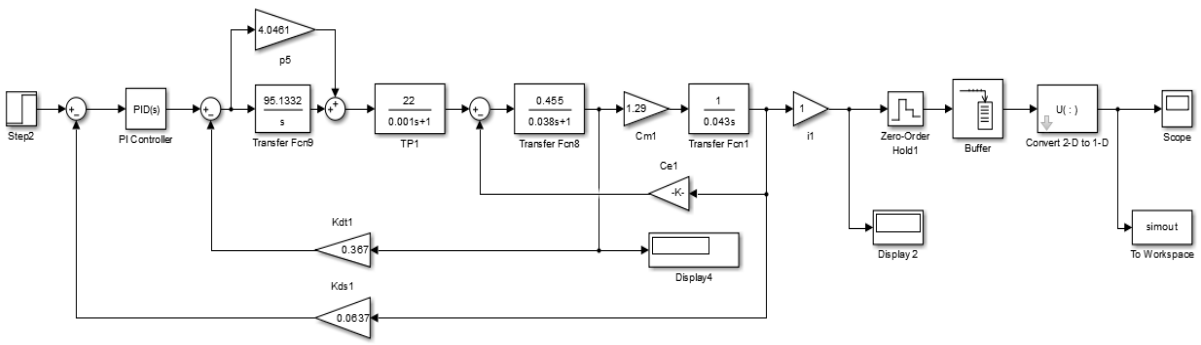


Рис.44. Имитационная модель привода

Выполним ряд операций следующего содержания.

1. Открыть блок «to workspace» в модели привода и установить тип данных «array».

2. Назначить желаемые значения параметра К исследуемого блока, запустить созданную модель. В рабочей области массив будет сгенерирован под переменной, по умолчанию именуемой simout.

3. Сохранить массив с новым именем переменной (чтобы избежать потери данных при перезапуске модели на следующих шагах).

4. Изменить значения параметра К в небольших пределах, снова запустить модель и сохранить новый массив внутри этой переменной, но в новой строке.

5. Повторить шаг 4 для ряда значений, смежных с желаемыми значениями, пока не получим достаточную базу данных для последующего обучения нейронных сетей.

6. Классифицируем базу данных по классам.

7. Для каждого класса создаем векторы целей PNNs.

8. Создать PNN сеть, используя классы и их эквивалентные цели.

9. Объединить созданные PNN_S сети вместе в одну систему, а затем соедините ее с моделью созданного привода.

10. Запустить модель для получения результата.

Реализация сети PNN в Matlab потребует использования промежуточных целей, так как PNN не выводит рациональное число. Лучшим решением является выбор линейной нейронной сети, чтобы сделать линейное отображение между промежуточной целью и реальной целью. Иначе говоря, перейти от номера класса к реальному значению.

Зададим возможные интервалы изменения коэффициента К регулятора, выделив три зоны: зона 1 [15,16), зона 2 (16,17), зона 3 (17,18] и выполним ряд этапов.

Этап 1. Создание PNN для классификации параметра К в эти три зоны, т.е. PNN на этом этапе выделит один из трех кластеров, чтобы впоследствии использовать ее для определения целочисленной части параметра К, если это 15, 16 или 17.

Этап 2. После определения зоны с относительно большой областью на предыдущем этапе, необходимо сузить область оценки чтобы получить более точный результат. Другими словами, определяется дробная часть параметра К.

Чтобы достичь этого, необходимо для $К_n$ создать (по одному для каждой зоны) подзоны, классифицирующие эту зону в интервале от 0 до 9, (из десяти различных подзон для каждого десятизначного числа), В результате мы получим 30 подзон:

1 Подзона: [15,15.1);	11 Подзона: [16, 16.1);	21 Подзона: [17, 17.1);
2 Подзона: [15.1,15.2);	12Подзона:[16.1,16.2);	22 Подзона: [17.1, 17.2);
3 Подзона: [15.2,15.3);	...	...
...		29 Подзона:[17.8 17.9);
10 Подзона: [15.9, 16);	20 Подзона: [16.9, 17);	30. Подзона: [17.9 18).

Например, если сеть PNN на первом этапе идентифицировала сигнал относящийся ко второй зоне [16, 17), то теперь необходимо знать, в какой подзоне произошло отклонение в этой области, т.е. 16.1, 16.2, 16.3.....16.9.

Этап 3. Создаем 30 сетей PNN (по одной для каждой подзоны), чтобы выделить параметр К с точностью до второй цифры справа от десятичной точки. Теперь мы получим 300 возможных значений параметра К, начиная с 15.00 и заканчивая 17.99 с шагом 0.01.

Результат представления классов и подклассов приведены в таб.

Таблица 4.3.

Таблица разбиения на классы

Номер этапа	Количество необходимых кластеров	Количество столбцов / кластеров
1	3	9600
2	30	960
3	300	96

Теперь созданные НС следует объединить для получения окончательного результата наблюдения.

Алгоритм работы наблюдателя

Создание обучающей базы данных

Используя scriptfile в Matlab, пишем исходный код для создания обучающей базы данных (рис.45.), где этот исходный код выполняет шаги со 2 по 6 алгоритма; Таким образом, созданная база данных будет использо-

ваться позже для обучения PNNs 1-го, 2-го и 3-го этапов. В соответствии со значениями переменных L, n1, n2, step1 и step2, которые находятся в скрипте, мы получим 28800 столбцов, которые будут по-разному распределяться на каждом этапе в соответствии с количеством кластеров, необходимых для этого этапа (таблица 4.3).

Создание целевых векторов

Как мы знаем в нашем дизайне, у нас будет более одной нейронной сети на втором и третьем этапах. Однако, мы по-прежнему можем создать только одноразовый целевой вектор на каждом этапе, чтобы его можно было использовать позже для обучения всех сетей этого этапа (это конечно, с разными кластерами обучающей базы данных). Перед созданием целевых векторов, нам нужно вернуться, чтобы взглянуть на размеры кластеров в таблице 4.3, чтобы избежать несоответствия размеров между обучающей базой данных и целевым вектором при обучении наших нейронных сетей.Мы создаем целевые векторы, используя исходные коды (рис. 46), которые написаны в script file MATLAB.

Первый исходный код (рис.46а) разработан для создания целевого вектора (T1) для PNN первого этапа, значениями которого являются первые три натуральных числа 1, 2 и 3. Каждое из которых повторяется столько раз, сколько столбцов в соответствующем кластере (i = 9600), чтобы добиться соответствия размеров. Число 1 представляет K = 15; число 2 представляет K = 16 и 3 число представляет K = 17.

Второй исходный код (рис.46б) написан для создания целевого вектора (T2), который будет использоваться позже для обучения трех PNNs второго этапа.

Здесь мы используем натуральные числа от 1 до 10 в качестве десяти целевых значений, чтобы представить десять различных значений десятых долей значения K. Эти целевые значения также повторяются несколько раз (i = 960) чтобы добиться соответствия размеров с соответствующими кластерами.

Для первого числа 1, **0**; для числа 2 - **1**; .....; для 10 - 0.**9**.

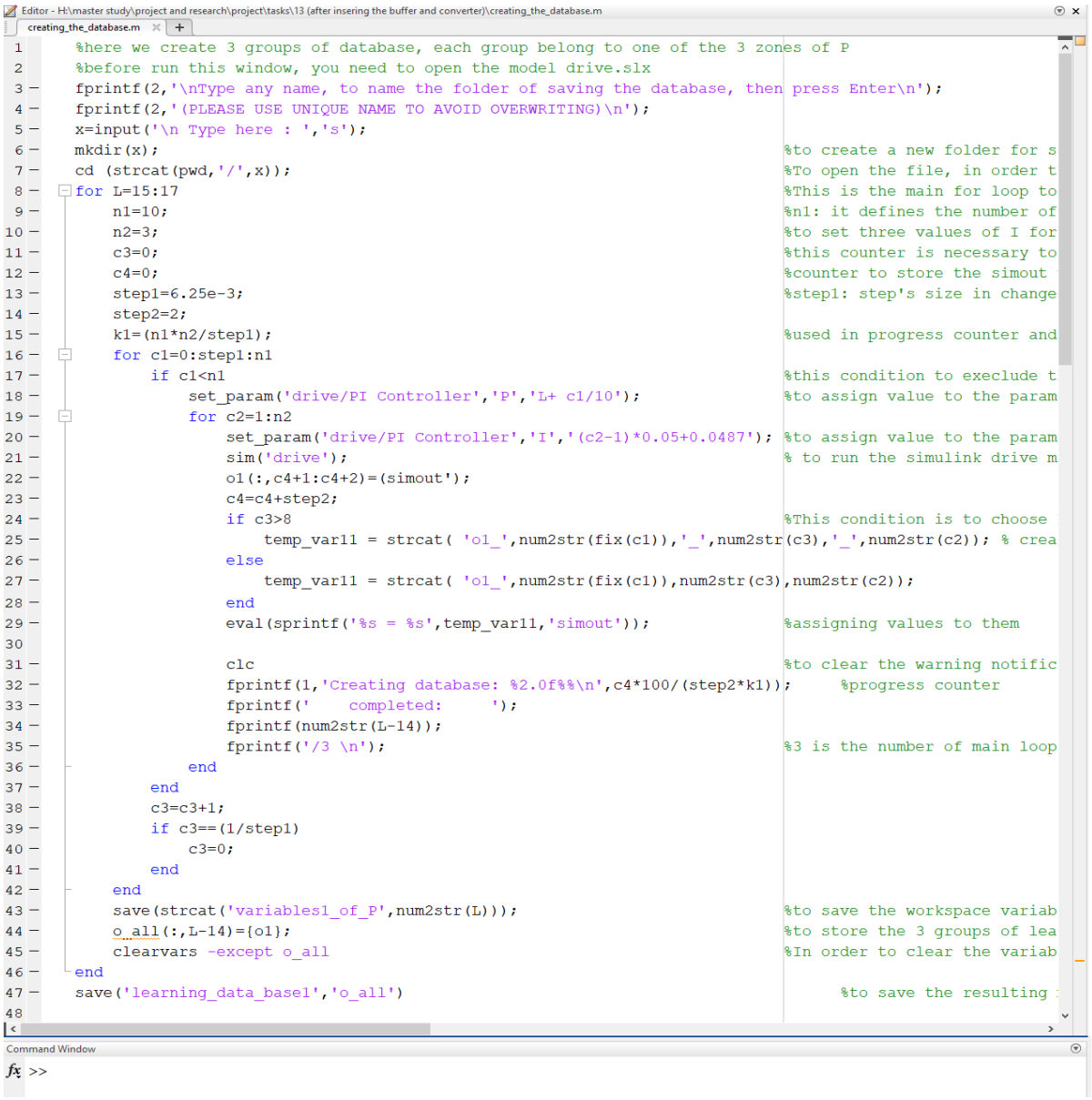
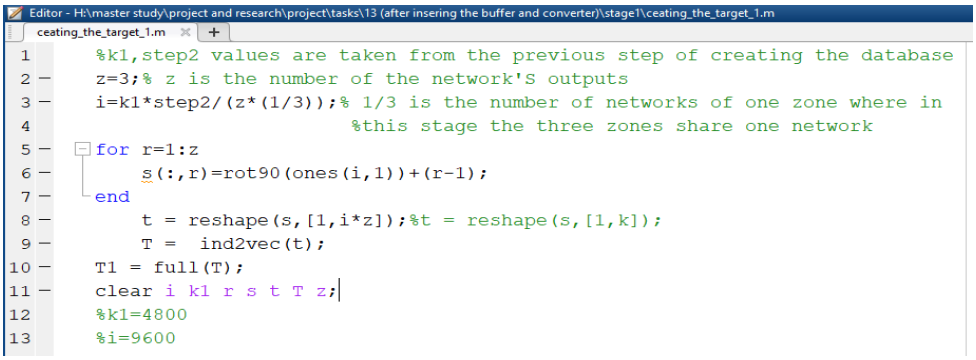
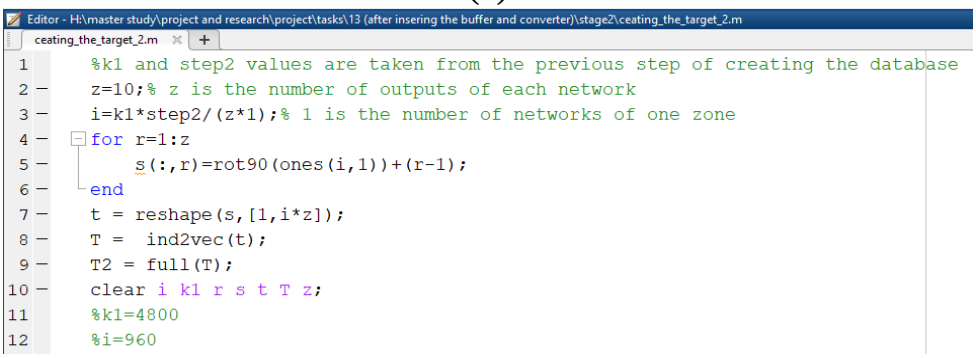


Рис.45. Исходный код создания обучающей базы данных

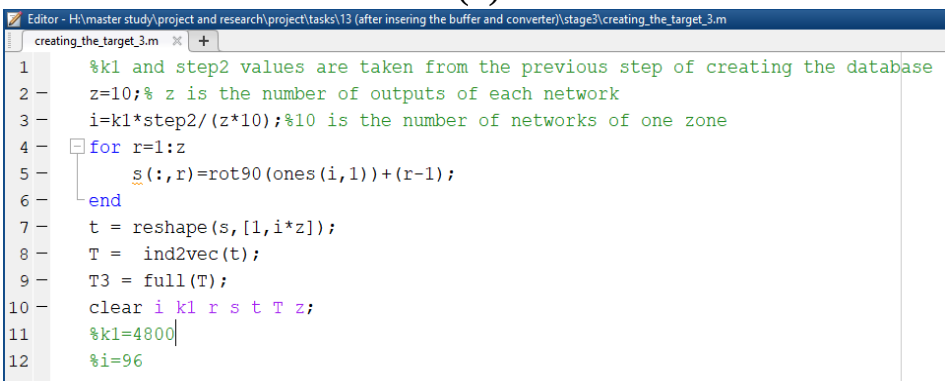
Точно так же для 3-го исходного кода (рис.46.в). Натуральные числа от 1 до 10 являются значениями целевого вектора (T3), но для представления сотых долей значения P, и повторяются количество раз (i = 96). Этот целевой вектор будет использоваться для обучения всех 30 PNNs 3-го этапа. Для первого числа - 0.00, второго – 0,01, …, для 10 – 0,09.



(а)



(б)



(в)

Рис.46. Исходные коды создания целевых векторов: а- для T1, б- для T2, в- для T3

Создание вероятностных сетей PNNs

Создание PNNs также выполняется путем записи исходных кодов в Script файлов MATLAB (рис.47.), где первый исходный код (рис.47а) записан для создания PNN первого этапа и обучения ее всей базе данных (O1) и целевом векторе (T1), в то время как второй исходный код (рис47б) делит обу-

чающую базу данных на три основных кластера, создает три PNNs второго этапа и обучает каждую из них соответствующему кластеру и целевом векторе (T2). Запустив скрипты, мы получим эти PNNs первых двух этапов в рабочей области Workspace; таким образом, мы сохраняем их вручную в папке CurrentFolder (или в любой папке) как файлы с именами (PNN1.mat) и (PNN2.mat), где файл PNN2.mat содержит три PNNs 2-го этапа.

Editor - H:\master study\project and research\project\tasks\13 (after insering the buffer and converter)\stage1\PNN_1.m

PNN_1.m | PNNs_2.m | PNNs_3.m

```
%o_all,T1 values are taken from the previous steps creating_the_database
%and creating_the_target_1 respectively

%%% CREATING THE PNN net1
O1=cell2mat(o_all);
net1=newpnn(O1,T1,1e-3);
clear O1
```

а

Editor - H:\master study\project and research\project\tasks\13 (after insering the buffer and converter)\stage2\PNNs_2.m

PNN_1.m | PNNs_2.m | PNNs_3.m

```
% o_all,T2 values are taken from the previous steps creating_the_database
% and creating the target_2 respectively.
% this source code creates 3 PNNs, each of them learns different cluster of
% the main learning database, but they share the same target vector.
%%% CREATING THE THREE PNNs net2_1, net2_2 AND net2_3
i=size(o_all);
for u=1:i(2)
    O2=cell2mat(o_all(1,u));
net=newpnn(O2,T2,1e-3);
x = strcat('net2_',num2str(u));%creating the variables of the 3 PNN
eval(sprintf('%s = %s',x,'net'));%assigning their networks to them
end
clear i u O2 net x
```

б

Editor - H:\master study\project and research\project\tasks\13 (after insering the buffer and converter)\stage3\PNNs_3.m

PNN_1.m | PNNs_2.m | PNNs_3.m

```
% o_all,T3 and step2 values are taken from the previous steps creating_the_database
% and creating the target_3 respectively.
% this source code creates 30 PNNs, each of them learns different cluster of
% the main learning database, but the same target vector used to train all.
fprintf(2,'\nType any name, to name the folder of saving the networks, then press Enter\n');
fprintf(2,'(PLEASE USE UNIQUE NAME TO AVOID OVERWRITING)\n');
x=input('\n Type here : ','s');
mkdir(x);
cd (strcat(pwd,'/',x));
%%
i=size(o_all);
for u=1:i(2)
    O=cell2mat(o_all(1,u));
    for j=1:10
        O3=O(:,(j-1)*480*step2+1:j*480*step2);
        net=newpnn(O3,T3,1e-3);
        x = strcat('net3_',num2str(j-1));
        eval(sprintf('%s = %s',x,'net'));
    end
    clear O j O3 net x
    save(strcat('PNN3_',num2str(u)));
end
clear i u O j O3 net x
```

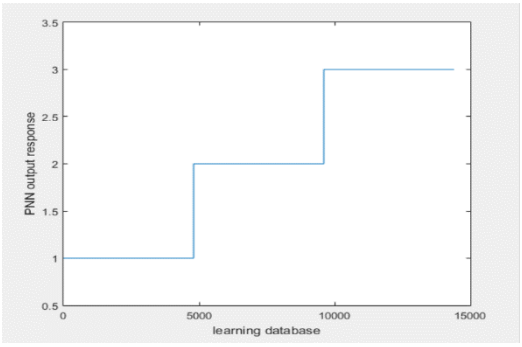
в

Рис.47. Скрипт файл

Точно так же третий исходный код (рис.47в) делит обучающую базу данных на 30 кластеров, затем создает 30 PNNs третьего этапа, и обучает их соответствующим кластерам и целевому вектору (T3). Поскольку исходный код этого этапа содержит команду сохранения, он сохраняет PNNs в папке CurrentFolder в виде трех файлов в папке с именем, которое можно ввести из окна CommandWindow при запуске скрипта. Имена этих трех файлов задаются исходным кодом как (PNN3_1.mat, PNN3_2.mat и PNN3_3.mat). Каждый содержит 10 PNNs.

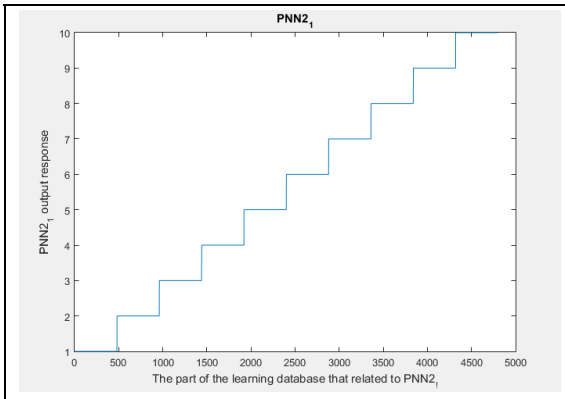
После создания PNNs необходимо протестировать их (правильно ли они созданы и обучены). Тестировать их все - не сложная задача, поэтому мы это сделаем, но поскольку их 34, и принцип тестирования на этапе одинаков для всех его сетей, поэтому в нашем исследовании мы будем обсуждать только по одному с каждого этапа. Тест выполняется с использованием некоторых команд, которые записаны в виде файлов **Script** (рис.48). Запустив эти скрипты, мы получаем выходные характеристики PNNs. Идеальный выходной PNN1 (рис.48.а) после тестирования в обучающей базе данных доказывает, что наша PNN была создана и должным образом обучена. Для второго и третьего этапов, выбираем по первой из каждого из (PNN2_1) и (PNN3_1). Они также дают оптимальные выходные характеристики (Рис.48.б) и (Рис.48.в) соответственно.

```
%%% TESTING THE PNN net1
Y=sim(net1,O1(:,2:2:28800));%
%starting from 2 and stepping 2
%for eliminating the zeros columns
%that caused by using buffer
y=vec2ind(Y);
plot (y)
clear Y
```



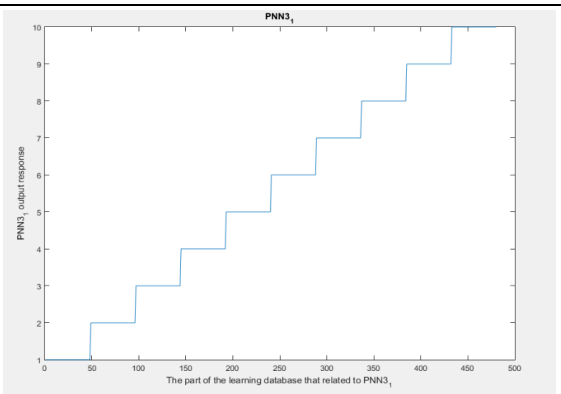
а

```
%%% TESTING THE CREATED ABOVE PNNs
O2_1=cell2mat(o_all(1,1));% for net2_1
O2_2=cell2mat(o_all(1,2));% for net2_2
O2_3=cell2mat(o_all(1,3));% for net2_3
% below only testing the PNN net2_1 because
% the other two have the same procedure
Y=sim(net2_1,O2_1(:,2:2:9600));
y=vec2ind(Y);
plot (y)
```



б

```
25      %%% TESTING THE CREATED ABOVE PNNs
26 -    O3_1=cell2mat(o_all(1,1));
27 -    O3_1=O3_1(:,2:2:960);
28 -    Y=sim(net3_0,O3_1);
29 -    y= vec2ind(Y);
30 -    plot(y);
31 -    clearvars -except o_all T3 step2
```



в

Рис.48. Исходные коды тестирования PNNs и их эквивалентные выходные характеристики:(а) для PNN1, (б) для PNN2_1, (в) для PNN3_1

Использование PNN для определения К

Чтобы использовать наши 34 недавно созданных PNNs для диагностики, нам нужно сначала связать их вместе как одну систему, затем с моделью ЭП. Для этого есть два способа: создание исходного кода и генерация сетей PNNs как блоков Simulink

1- Создание исходного кода

Скрипт исходного кода P_Diagnostic (рис.49) содержит оператор switch-case для связывания PNNs, а также некоторые другие команды, используемые для сопряжения с Simulink моделью ЭП.

Запустив скрипт, состояние будет идентифицированно, но перед запуском надо учесть следующее:

– Загрузка сетей 1-го и 2-го этапов в в окно Workspace из сохраненных файлов (PNN1.mat) и (PNN2.mat) соответственно;

– Открытие модели Simulink (drive.slx);

– Перемещение файлов сетей 3-го этапа (PNN3_1.mat, PNN3_2.mat и PNN3_3.mat) в папку CurrentFolder;

– Также перемещение файла функции проверки (check.m) в папку CurrentFolder;

– Запустить скрипт, затем введите требуемые значения из окна CommandWindow (а требуемые значения - это предполагаемые значения находятся в предопределенных параметрах К, представляют собой их изменяемые значения, которые должны измениться из-за внешних условий, вызывающих изменение выходных характеристик ЭП).

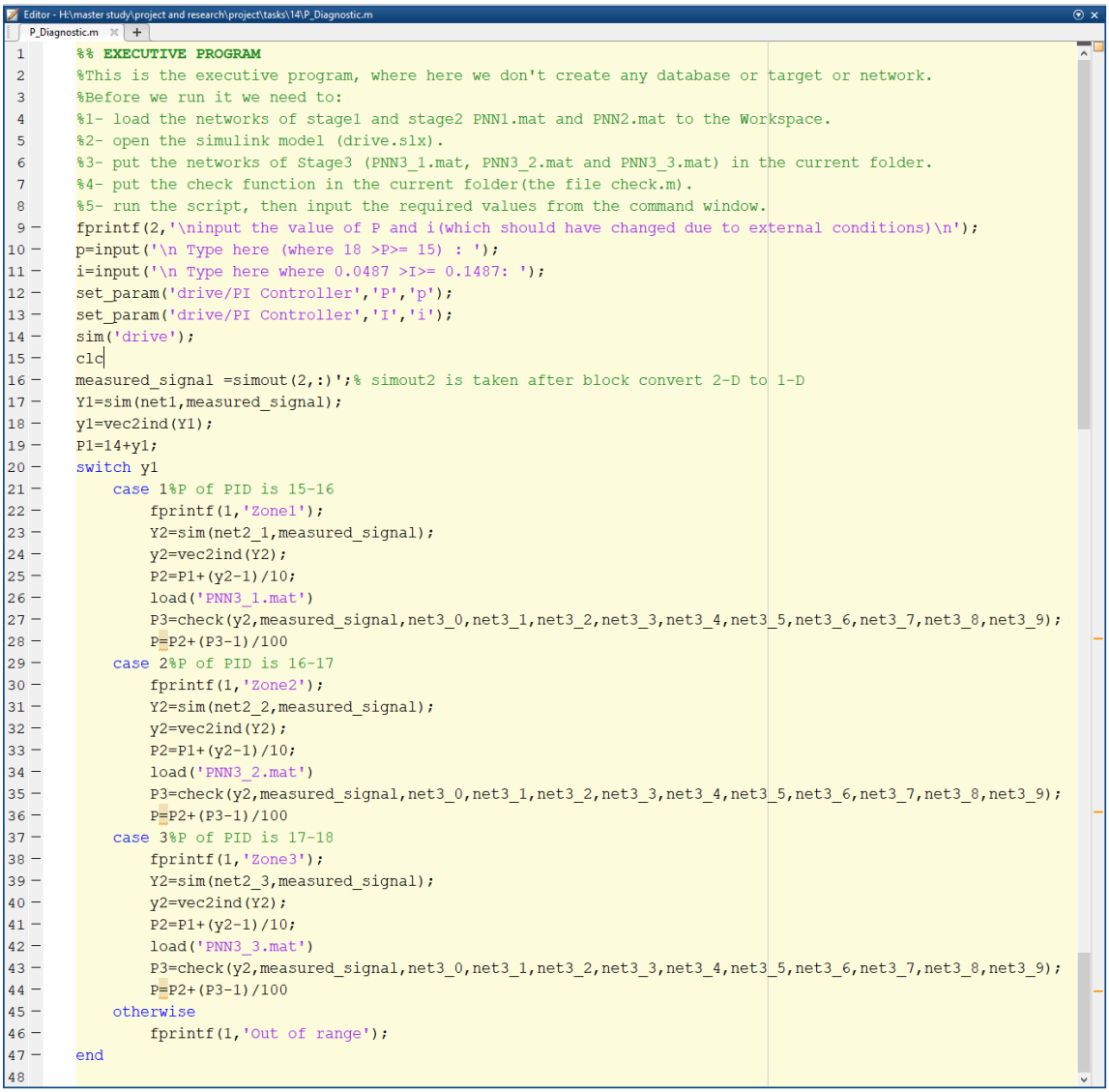


Рис.49. Исходный код для диагностики К-параметра

При запуске файла скрипта (P_Diagnostic.m), окно CommandWindow запросит ввод значений параметров К, так что вводим их в пределах ранее упомянутый диапазон (табл. 4.3); таким образом, они будут отправлены в блок PID в Simulink модели ЭП, что, в свою очередь, повлияет на характеристики выходного сигнала. После этого этот выходной сигнал (названный как “measured signal” в исходном коде) будет возвращен в окно **Workspace** в виде массива, затем скрипт обработает и введет его в PNN1, который выведет значение, определяющее номер зоны и также используется для определения целой части значения P. Затем оператор switch-case согласно выходному значению PNN1 переключит процесс в одну из трех сетей PNN2 и загрузит один из трех файлов PNN3_1, PNN3_2 или PNN3_3 в окно **Workspace**. Массив сигнала будет снова введен, но в эту одну из трех PNN2. В результате будет получено выходное значение, которое используется для определения номера подзоны и десятых долей значения P. С помощью

другого оператора switch-case, который записан как файл функции MATLAB, названный (check function) (рис.50), будет принято решение переключиться на одну из 10 PNNs, на которую будет перенаправлен массив сигнала, производя выходное значение, которое используется для определения сотых долей значения K. Используя некоторые математические операции в скрипте, значение неисправности вычисляется с использованием выходных данных PNNs, на которые переключаются процессы выполнения при запуске скрипта.

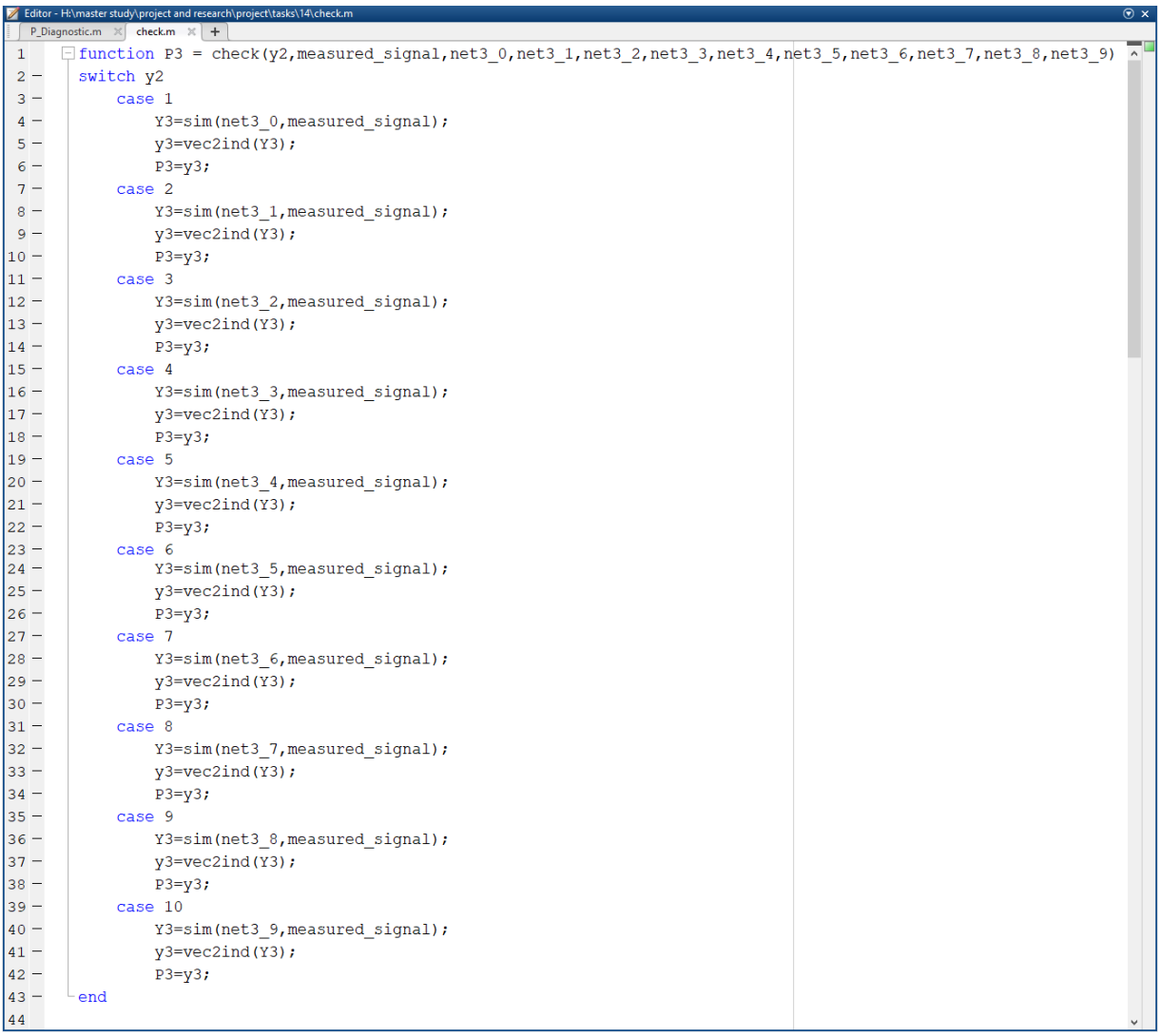


Рис.50. Выбор функции

2- Генерация сетей PNN как блоков Simulink

Мы также можем сгенерировать PNNs как блоки Simulink, чтобы использовать их внутри самой модели ЭП, вместо того, чтобы делать тот интерфейс между скриптом и моделью Simulink, который мы сделали первым способом. Но для этого нам также необходимо создать линейные нейронные сети LNNs и сгенерировать их, чтобы сделать линейное отображение

между промежуточными целями и реальными целями. Поскольку PNNs уже были созданы, теперь надо создать LNNs, тогда сгенерировать все сети обеих типов вместе.

Создание линейной нейронной сети LNN

Перед их созданием нам необходимо определить входные данные LNNs(промежуточные цели) и их цели (реальные цели) (таблица 4.4).

Таблица 4.4.

Необходимые данные для создания LNN

Номер этапа	Входные данные LNN	Цели LNN target
Первый этап (LNN1)	1 0 0 0 1 0 0 0 1	15 16 17
Второй этап (LNN2)	1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1	0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
Третий этап (LNN3)	1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1	0.00 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09

Пусть:**p** обозначает вход; **t** обозначает цель, тогда исходные коды создания LNN следующие:

LNN1	**LNN2**	**LNN3**
>p1= diag (ones (1,3));	> p2=diag (ones (1,10));	>p3=diag(ones(1,10));
> t1= [15:1:17];	> t2 = [0:0.1:0.9];	>t3=[0:0.01:0.09];
>net1=newlind (p1, t1);	>net2 = newlind (p2, t2);	>net3 = newlind(p3,t3);

Для генерации PNNs и LNNs мы применяем следующую команду ко всем 34 PNNs и 3 LNNs одну за другой.

>>**genism** (имя сети, которая будет сгенерирована, как показано в окне Workspace)

Блоки сети будут сгенерированы, как показано на рис.51.

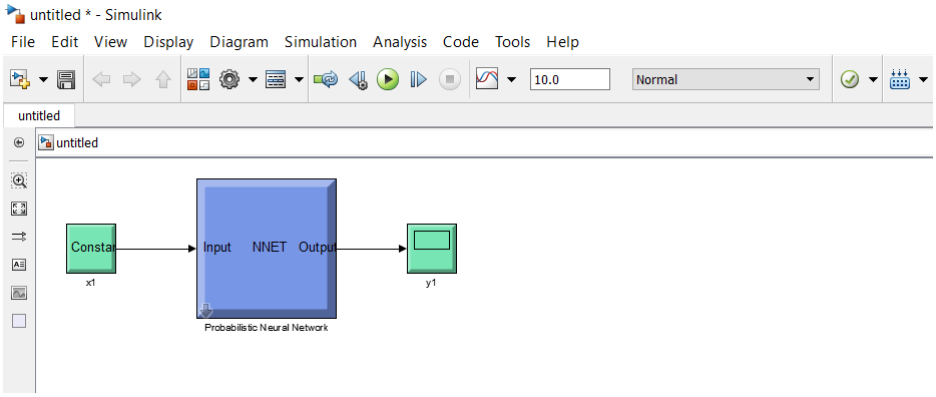


Рис.51. Сгенерированный блок нейронной сети

Мы копируем все эти блоки 34 PNNs и 3 LNNs и вставляем их в Simulink модель ЭП. Затем связываем их вместе и с моделью ЭП, как показано на рис. 52. Там, где есть 5 подсистем, первые две содержат части нашей Simulink модели ЭП без каких-либо изменений, а последние три содержат PNNs 3-го этапа (рис.53). Для проверки работоспособности вводим произвольное значение коэффициента К в PID регулятор (значения, которые должны быть неисправными), а затем запускаем модель, чтобы получить результат на display 3.

Теперь предположим, что текущие значения параметров ПИ-регулятора следующие: К=15,23

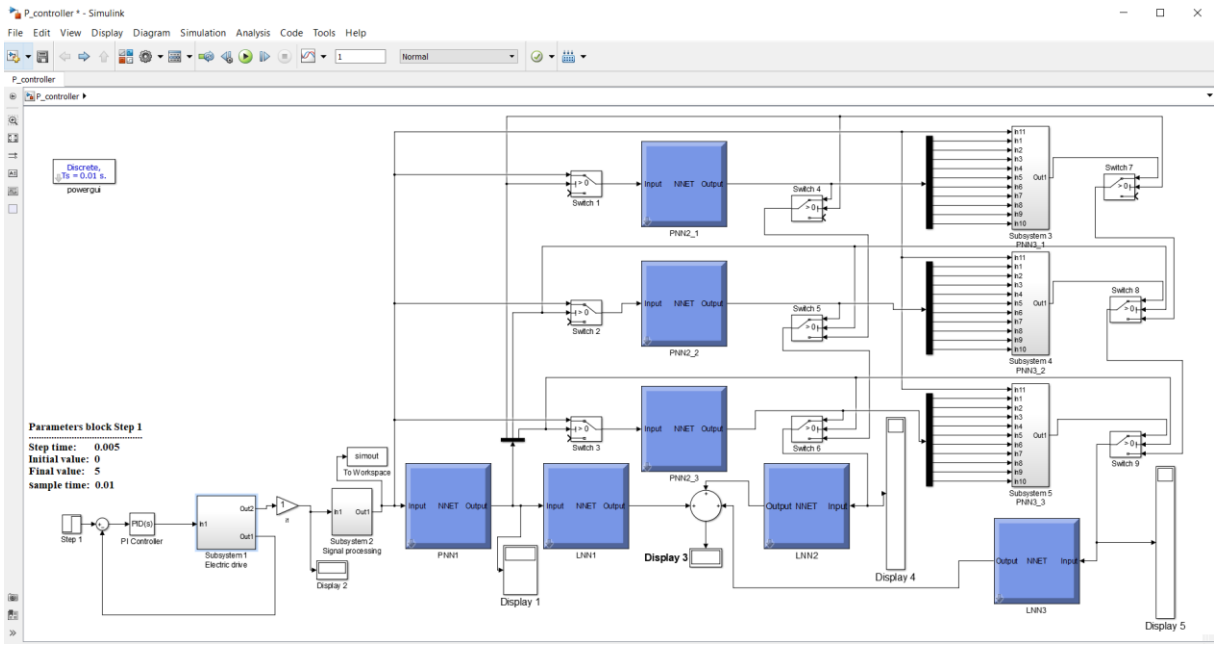


Рис.52. Рабочая модель нейронного наблюдателя

После запуска модели, измеренный сигнал электропривода буде обрабатываться через подсистему 2 (которая содержит три блока обработки сигнала, (рис.44). состоящую из трех элементов 1; 0; 0 в качестве промежуточной целей. Линейная нейронная сеть (LNN1) получит и отобразит ее в реальную цель как 15. Таким образом, целая часть К определена.

Выходной сигнал PNN1 также будет разбит на три уровня с помощью блока Demux:

1-й отсчет, равный 1 (высокий логический уровень), поступит на переключатель 1, переключатель 4 и переключатель 7;

2-й отсчет, равный 0 (низкий логический уровень), поступит на переключатель 2, переключатель 5 и переключатель 8;

3-й отсчет, равный 0 (высокий логический уровень), поступит на переключатель 3, переключатель 6 и переключатель 9.

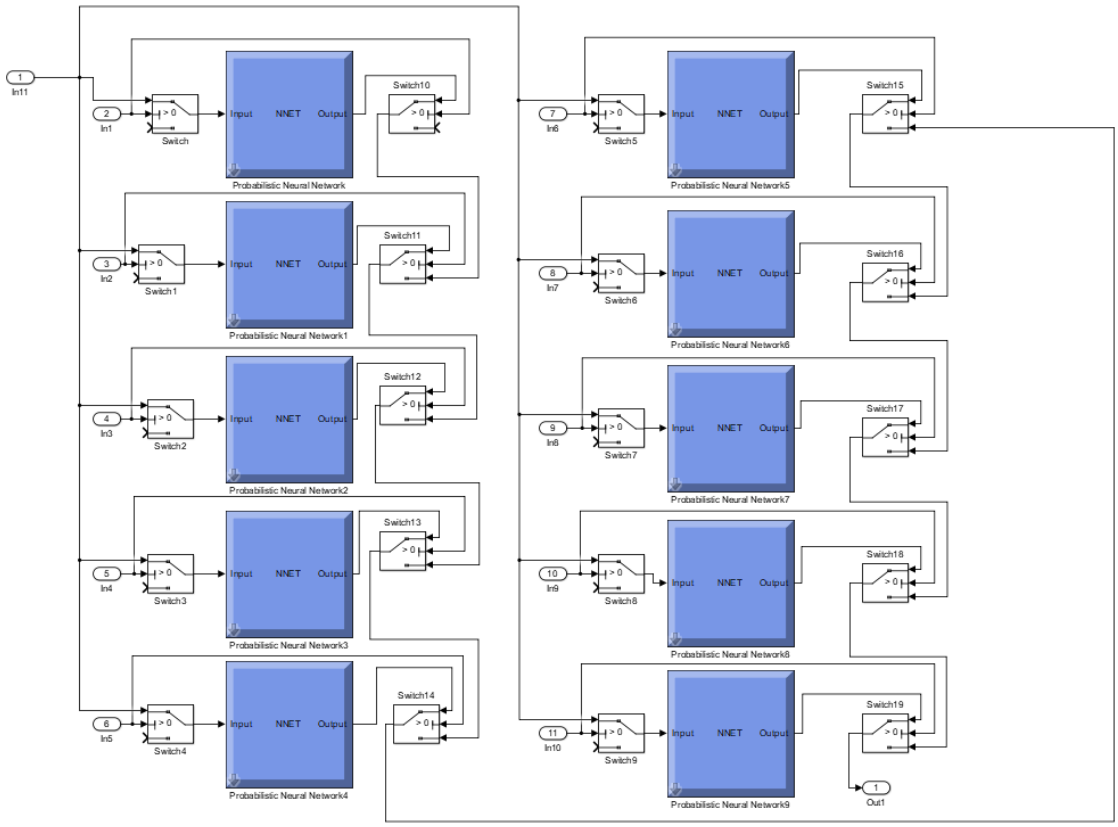
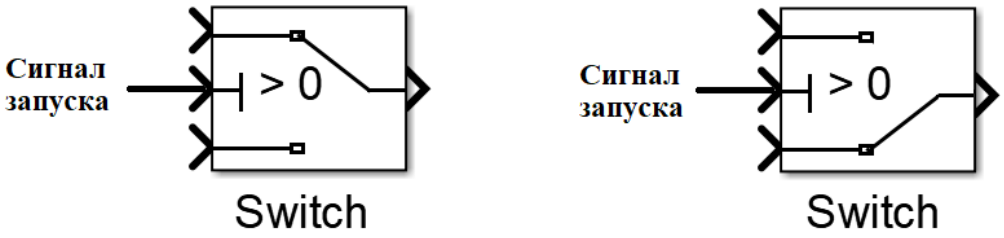


Рис.53. Нейронная сеть наблюдателя второго уровня

Эти переключатели имеют активный высокий уровень, поэтому они включаются, когда сигнал запуска имеет высокий логический уровень (Рис.54.).



сигнал запуска = 1 сигнал запуска = 0

Рис.54. Два статуса переключателя

Таким образом, переключатель 1 включится и позволит измеренному сигналу пройти в первую вероятностную нейронную сеть второго этапа (PNN2_1), в то время как два других переключателя (переключатель 2 и переключатель 3) останутся выключенными, что заблокирует измеренный сигнал от прохождения в другие сети второго этапа. Следовательно, PNN2_1 примет измеренный сигнал и сгенерирует цифровой сигнал из десяти цифровых отсчетов 0; 0; 1; 0; 0; 0; 0; 0; 0; 0 в качестве промежуточной цели. Сигнал промежуточной цели будет переключен на линейную нейронную сеть (LNN2) переключателем 4 (который включен из-за высокого логического уровня) пройдя через переключатели 5 и 6 (они пропускают его, когда они выключены). А LNN2 отобразит его в реальную цель как 0,2. Таким образом, также определяются десятые доли значения К. Тот же принцип для сетей этапа 3, то мы получим сотые доли значения К, равные 0,03, а затем, используя блок суммы, мы получим окончательное значение параметра К = 15 + 0,2 + 0,03 = 15,23.

Глава 5. НЕЧЕТКИЕ МНОЖЕСТВА В СИСТЕМАХ УПРАВЛЕНИЯ И ДИАГНОСТИКИ

5.1. Основные положения теории нечетких множеств

Наиболее важным применением теории нечетких множеств являются контроллеры нечеткой логики. Их функционирование несколько отличается от работы обычных контроллеров; для описания системы вместо дифференциальных уравнений используются знания экспертов. Эти знания могут быть выражены с помощью лингвистических переменных, которые описаны нечеткими множествами. Наиболее перспективным является разработка интеллектуальных САУ, строящихся на базе нечетких нейронных сетей, что позволяет сочетать как методы работы с нечеткой информацией и знаниями, так и способность систем к самостоятельной реконфигурации на программном и аппаратном уровне.

Дать общие рекомендации относительно выбора того или иного решения не представляется возможным – все зависит от области применения и специфики функционирования управляемого объекта. Тем не менее, в построении НК можно следовать двумя путями: от симптомов к заключению - восходящие выводы или от фактического результата к симптомам – нисходящие выводы.

Для САУ, как и для любой технической системы, использующей в своем составе контроллеры с нечеткой логикой можно сформулировать следующую последовательность создания:

- - сформировать цель применения и выбрать структуру НК;

Руководствуясь реализуемыми выходными характеристиками процесса управления, условиями эксплуатации и управляющими воздействиями выбрать структуру НК**.**

Являясь универсальным апроксиматором НК способен вычислять любую функцию, которая может быть использована в его настройке. По принципу построения структуры и алгоритма работы можно рекомендовать следующие структуры НК. Приведенные ниже структуры не являются исчерпывающими.

1. Классический НК. Однажды построенный НК не имеет возможности изменять свои свойства. НК прост в реализации, обеспечивает удовлетворительную точность.

2. Поднастраиваемые НК - настройка параметров в зависимости от изменяющихся условий функционирования. Например, использовать сплайн функции активации, которые можно деформировать, подстраиваясь к изменяемым параметрам системы или расширять (сужать) диапазон носителей множеств. Изменять α - сечение функций активации и т.п.

3. Перестраиваемые НК – где изменение структуры реализуется введением обратных связей, переключением между НК, подключением дополнительных множеств, изменением каналов передачи результатов выводов и многое другое.

4.Адаптивный НК–такой, что при непредвиденных изменениях внешних сигналов на входе или выходе НК на основе анализа автоматически изменяет значения своих параметров или структуру так, чтобы сохранялось заданное качество его работы.

5. Самонастраивающийся НК, в котором определенный параметр имеет область определения, зависящую от условий функционирования, настраивается таким образом, чтобы обеспечить его стабильность. Это может достигаться, например, изменением числа термов или областью их определения. Выбирается тот параметр НК, который белее подвержен нестабильности.

6. НК с эталонной моделью, реализованной в виде “жесткого” НК, в котором разность текущего значения выходного сигнала и модельного, компенсирует изменением базы знаний сводя результат вывода к минимальной ошибке и т.д.

- - формирование базы данных для обучения контроллера, определение числа термов входных и выходных переменных и преобразование фактических данных в лингвистические переменные (фазификация);

Определяется форма сигнала и диапазон изменения параметров в заданных точках системы при различных входных сигналах и различных условиях работы при учете возмущений. По сути, определяется множества измеряемых параметров участвующих в реализации НК. Определяются носители нечетких множеств.

Поскольку множество ситуаций при управлении объектом не постоянно из-за нестабильности и непредсказуемости свойств объекта и внешних воздействий использование аналитических моделей не целесообразно. Предпочтение отдается структурным моделям реальной системы, для которой создается НК.

Формирование базы данных в силу сложности систем и ограничений на измерение переменных, исключительная прерогатива моделей.

Можно прибегнуть и к помощи экспертов для сбора данных, но они будут иметь исключительно лингвистический смысл и для использования в процедуре реального управления их необходимо ранжировать с учетом технических характеристик систем. Мнения экспертов зачастую противоречивы и требуют дополнительных согласований и особенно для систем с неполной информацией о структуре.

- - выбор вида функций принадлежности нечетких переменных;

Практически в качестве функций принадлежности могут использовать любые зависимости наилучшим образом аппроксимирующие входные и выходные данные с ожидаемой точностью. Существенно количество термов на интервале представления, особенности наложения функций принадлежности, способов их задания, конечный и начальный вид функций и в особенности выходных.

- -формирование базы знаний и связи между пространством предпосылок и пространством заключений;

База знаний наиболее существенная составляющая в структуре НК. Обработка знаний основывается либо на правилах, либо на основе нечетких отношений.

В блоке формирования логического решения на основе знаний (базы правил) записываются лингвистические правила. Соответствующей формулировкой правил достигается результат, при котором для любой лингвистической величины управляющего воздействия, как минимум одно из правил оказывается приемлемым. Заметим, что замена одних правил на совокупность других или увеличение их числа, позволяет создать систему любой степени сложности, ориентированную на решение задач управления.

Наиболее часто используется "минимаксный" (Max-Min Inference) метод логического решения, когда путем взаимного наложения выбирается результирующая функция принадлежности с максимальной величиной истинности.

Этот процесс соединяет в себе все основные концепции теории нечетких множеств: функции принадлежности, лингвистические переменные, нечеткие логические операции, методы нечеткой импликации и нечеткой композиции.

Получить решающе правило (заключение) можно используя известные методы Sugeno, Mamdani, Larsen и Tsukamoto. Если эти методы не оправдывают ваших ожиданий, разработайте свой оригинальный метод реализации заключения, что представляет собой не простую задачу. Следует позаботься о выборе метода логического вывода и правила композиции.

- -способ преобразования лингвистических переменных в числовые (дефаззификация).

Существует несколько методов преобразования лингвистических переменных в числовые отличающихся по точности. Это First - of - Maxima- первый максиму, Middle – of - Maxima - средний максимум, Max - Criterion – критерий максимума, Heightdefuzzfication - наибольший из всех α - уровней.

Внедрение в САУ нечетких контроллеров является неизбежным шагом в силу широких возможностей нечеткой логики и ее приложений, учитывая, что микропроцессоры значительно быстрее выполняют логические операции, чем арифметические, а нечеткие алгоритмы проектируются значительно проще.

Приведенная классификация нечетких систем не является окончательной, поскольку в настоящее время ведутся работы по дальнейшему развитию интеллектуальности нечетких САУ. Эти цели могут быть достигнуты путем комбинации различных подходов к построению интеллектуальных систем в составе иерархических интеллектуальных контроллеров, органично сочетающих принципы и методы нечеткого вывода, ситуационного управления, инженерии знаний, нейронных сетей и эволюционного моделирования, использующие мягкие вычисления и большие данные. Поэтому важной проблемой, которая еще долго будет актуальной, является разработка теории или метода представления знаний произвольного вида.

Сложность представляет оценка уровня интеллекта нечетких САУ. Очевидно, что решение связано с экспертными оценками. Можно с уверенность утверждать, что максимумом искусственного интеллекта обладает структура fullfuzzy и минимумом структура с простым нечетким контроллером.

Любой проект проходит ряд этапов, которые неизбежно сталкиваются с задачей убедиться в правильности принятых решений и увидеть полученный результат еще до реального воплощения. В этом случае помогает модель. Сначала формулируются задача и цель мо-

делирования. Затем анализируется объект моделирования, выбирается способ представления модели и средства реализации. Окончательно выполняется эксперимент и анализ результатов моделирования.

5.2. Общая структура нечеткого контроллера

Общая структура микроконтроллера, использующего нечеткую логику, содержит:

- блок фаззификации;
- базу знаний;
- блок решений;
- блок дефаззификации.

Блок фаззификации преобразует четкие величины, измеренные на выходе объекта диагностирования, в нечеткие величины, которые описаны лингвистическими переменными в базе знаний.

Блок решений использует нечеткие условные (if - then) правила, заложенные в базу знаний, для преобразования нечетких входных данных в необходимые управляющие влияния, которые также носят нечеткий характер.

Блок дефаззификации превращает нечеткие данные с выхода блока решений в четкую величину, которая используется для диагностики объекта.

В качестве примера известных микроконтроллеров, использующих нечеткую логику можно назвать 68HC11, 68HC12 фирмы Motorola, MCS-96 фирмы Intel, а также некоторые другие.

Создание модели нечеткого контроллера в MATLAB

Назначение и возможности пакета «Fuzzy Logic Toolbox».

Пакет нечеткой логики — это пакет прикладных программ, относящихся к теории размытых или нечетких множеств и позволяющих конструировать так называемые нечеткие экспертные и/или управляющие системы. Основные возможности пакета:

– построение систем нечеткого вывода (экспертных систем, регуляторов, аппроксиматоров зависимостей);

– построение адаптивных нечетких систем (гибридных нейронных сетей);

– интерактивное динамическое моделирование в «Simulink»;

– пакет позволяет работу:

- в режиме графического интерфейса;
- в режиме командной строки;
- с использованием блоков и примеров пакета «Simulink».

Перечень дополнительной информации приведен в приложении 3 и 4.

Построение нечеткой системы.

Командой (функцией) «Fuzzy» из режима командной строки запускается основная интерфейсная программа пакета «FuzzyLogic» – редактор нечеткой системы вывода. Вид открывающегося при этом окна приведен на рисунке 5.1.

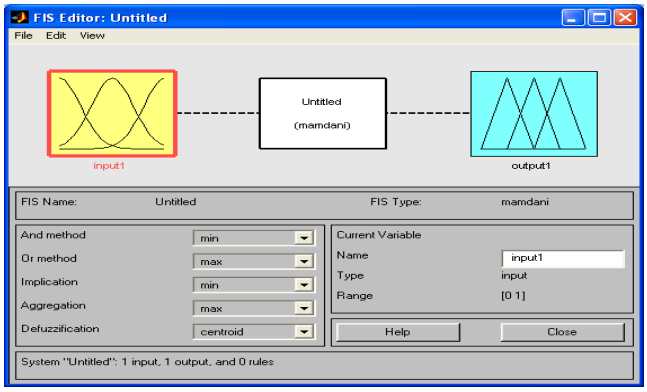


Рис.5.1. Вид окна FIS Editor

Меню редактора содержит следующие позиции:

• «File»– работа с файлами моделей (их создание, сохранение, считывание и печать).

• «Edit»– операции редактирования (добавление и исключение входных и выходных переменных).

• «View»– переход к дополнительному инструментарию.

Создаю нечеткую систему, отображающую зависимость между входными и выходными переменными. В качестве входных переменных задаю отклонения скорости и тока, полученные при разности выходных сигналов модели электродвигателя, работающего при номинальных значениях параметров, и реального двигателя.

Нечеткие выводы.

Используемый в различного рода экспертных и управляющих системах механизм нечетких выводов в своей основе имеет базу знаний, формируемую специалистами предметной области в виде совокупности нечетких предикатных правил вида:

$П_1$: если x есть A_1, то y есть B_1,

$П_2$: если x есть A_2, то y есть B_2,

…

$П_n$: если x есть A_n, то y есть B_n,

где x – входная переменная (имя для известных значений данных), y – переменная вывода (имя для значения данных, которое будет вычислено); A и B – функции принадлежности, определенные, соответственно, на x и y.

Пример подобного правила:

если x – низко, то y – высоко.

Приведем более детальное пояснение. Знание эксперта $A \rightarrow B$ отражает нечеткое причинное отношение предпосылки и заключения, поэтому его можно назвать нечетким отношением и обозначить через R:

$R = A \rightarrow B$, где «$\rightarrow$» называют нечеткой импликацией.

Отношение R можно рассматривать как нечеткое подмножество прямого произведения $X \times Y$ полного множества предпосылок X и заключений Y. Таким образом, процесс получения (нечеткого) результата вывода B' с использованием данного наблюдения A' и знания $A \rightarrow B$ можно представить в виде формулы:

$B' = A' \bullet R = A' \bullet (A \rightarrow B)$,

где «$\bullet$» — операция свертки.

Как операцию композиции, так и операцию импликации в алгебре нечетких множеств можно реализовывать по-разному (при этом, естественно, будет разниться и итоговый получаемый результат), но в любом случае общий логический вывод осуществляется за следующие четыре этапа.

1. Нечеткость (введение нечеткости, фаззификация, «fuzzification»). Функции принадлежности, определенные на входных переменных, применяются к их фактическим значениям для определения степени истинности каждой предпосылки каждого правила.

2. Логический вывод. Вычисленное значение истинности для предпосылок каждого правила применяется к заключениям каждого правила. Это приводит к одному нечеткому подмножеству, которое будет назначено каждой переменной вывода для каждого правила. В качестве правил логического вывода обычно используются только операции «min» (минимум) или «prod» (умножение). В логическом выводе min функция принадлежности вывода «отсекается» по высоте,

соответствующей вычисленной степени истинности предпосылки правила (нечеткая логика «И»). В логическом выводе prod функция принадлежности вывода масштабируется при помощи вычисленной степени истинности предпосылки правила.

3. Композиция. Нечеткие подмножества, назначенные для каждой переменной вывода (во всех правилах), объединяются вместе, чтобы сформировать одно нечеткое подмножество для каждой переменной вывода. При подобном объединении обычно используются операции «max» (максимум) или «sum» (сумма). При композиции max комбинированный вывод нечеткого подмножества конструируется как поточечный максимум по всем нечетким подмножествам (нечеткая логика «ИЛИ»). При композиции sum комбинированный вывод нечеткого подмножества конструируется как поточечная сумма по всем нечетким подмножествам, назначенным переменной вывода правилами логического вывода.

4. В заключение (дополнительно) — приведение к четкости (дефаззификация, «defuzzification»), которое используется, когда полезно преобразовать нечеткий набор выводов в четкое число.

5.3. Алгоритмы нечеткого вывода

Рассмотрим следующие наиболее употребительные модификации алгоритма нечеткого вывода, полагая, для простоты, что базу знаний организуют два нечетких правила вида:

$П_1$: если x есть A_1, и y есть B_1, тогда z есть C_1,

$П_2$: если x есть A_2,и y есть B_2, тогда z есть C_2,

где x и y– имена входных переменных, z – имя переменной вывода, A_1, A_2, B_1, B_2, C_1,C_2– некоторые заданные функции принадлежности, при этом четкое значение z_0 необходимо определить на основе приведенной информации и четких значений x_0 и y_0.

Алгоритм Мамдани (Mamdani)

Данный алгоритм математически может быть описан следующим образом.

1.Нечеткость: находятся степени истинности для предпосылок каждого правила: $A_1(x_0)$, $A_2(x_0)$, $B_1(y_0)$, $B_2(y_0)$.

2.Нечеткий вывод: находятся уровни «отсечения» для предпосылок каждого из правил (с использованием операции «min»):

α_1,= $A_1(x0) \wedge B_1(y_0)$,

$\alpha_2 = A_2(x_0) \wedge B_2(y_0)$,

где через «∧», как и раньше, обозначена операция логического минимума («min»), затем находятся усеченные функции принадлежности :

$C'_1(z) = (\alpha_1 \wedge C_1(z))$,

$C'_2(z) = (\alpha_2 \wedge C_2(z))$.

3.Композиция: с использованием операции «max» (обозначаемой как «у») производится объединение найденных усеченных функций, что приводит к получению итогового нечеткого подмножества для переменной выхода с функцией принадлежности:

$\mu_\Sigma(z) = C(z) = C'_1(z) \vee C'_2(z) = (\alpha_1 \wedge C_1(z)) \vee (\alpha_2 \wedge C_2(z))$.

4. Наконец, приведение к четкости (для нахождения z_0) проводится, например, центроидным методом (как центр тяжести для кривой $\mu_\Sigma(z)$):

$$z_0 = \frac{\int_\Omega z\mu_\Sigma(z)dz}{\int_\Omega \mu_\Sigma(z)dz}$$

Алгоритм иллюстрируется на рисунке 5.2.

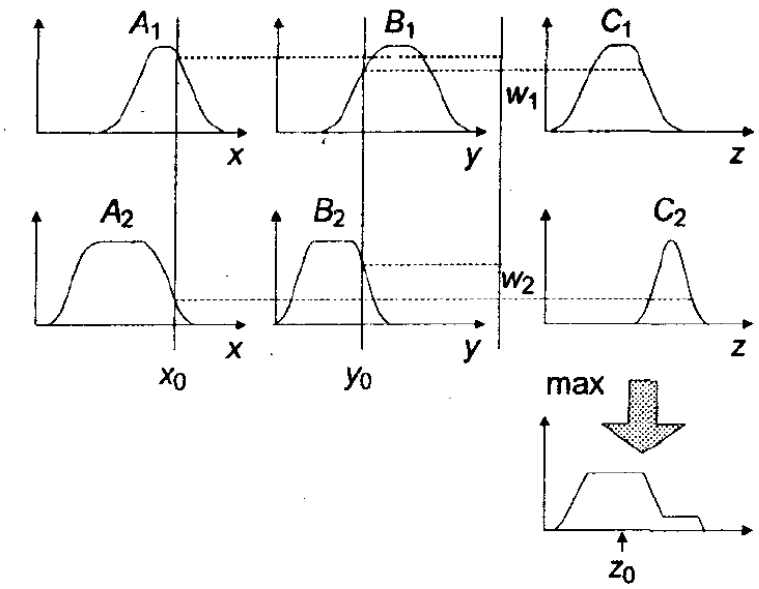


Рис.5.2. Иллюстрация к алгоритму Мамдани

Алгоритм Сугэно (Sugeno)

Сугэно (Sugeno) и Такаги (Takagi) использовали набор правил в следующей форме (как и раньше, приводим пример двух правил):

П$_1$: если x есть A_1 и y есть B_1, тогда $z_1 = a_1x + b_1y$,

П$_2$: если x есть A_2 и y есть B_2, тогда $z_2 = a_2x + b_2y$.

Представление алгоритма.

1.Первый этап – как в алгоритме Мамдани.

2.На втором этапе находятся

$\alpha_1 = A_1(x_0) \wedge B_1(y_0)$, $\alpha_2 = A_2(x_0) \wedge B_2(y_0)$

и индивидуальные выходы правил:

$z_{\cdot 1} = a_1x_0 + b_1y_0$,

$z_{\cdot 2} = a_2x_0 + b_2y_0$.

3.На третьем этапе определяется четкое значение переменной вывода:

$$z_0 = \frac{\alpha_1 z_1^{\bullet} + \alpha_2 z_2^{\bullet}}{\alpha_1 + \alpha_2}.$$

Алгоритм иллюстрируется на рисунке 5.3.

Приведенное представление относится к алгоритму Сугэно 1-го порядка. Если правила записаны в форме:

П$_1$: если x есть A_1 и y есть B_1, то $z_1 = c_1$,

П$_2$: если x есть A_2 и y есть B_2, то $z_2 = c_2$,

то говорят, что задан алгоритм Сугэно 0-го порядка (рис.5.4).

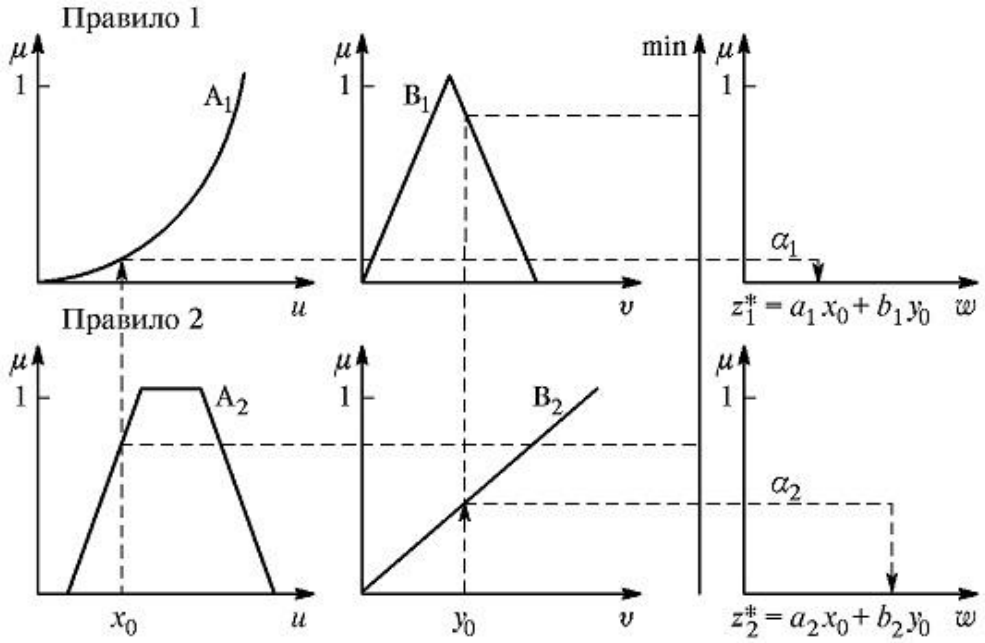


Рис.5.3. Иллюстрация к алгоритму Сугэно

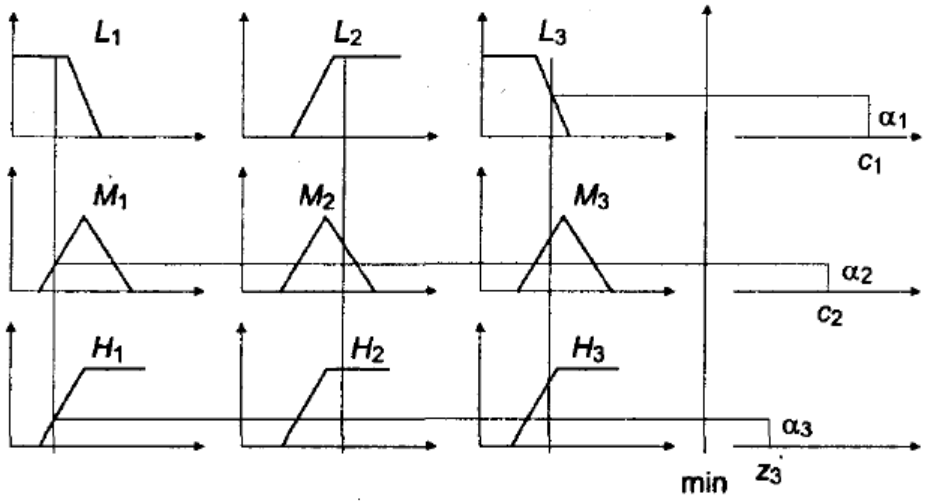


Рис.5.4. Алгоритм Сугэно 0-го порядка

Алгоритм Tsukamoto

Исходные посылки — как у предыдущего алгоритма, но в данном случае предполагается, что функции $C_1(z)$, $C_2(z)$ являются монотонными.

1. Первый этап — такой же, как в алгоритме Mamdani.
2. На втором этапе сначала находятся (как в алгоритме Mamdani) уровни «отсечения» α_1 и α_2, а затем — посредством решения уравнений

$\alpha_1 = C_1(z_1)$, $\alpha_2 = C_2(z_2)$

— четкие значения (z_1 и z_2)для каждого из исходных правил.

3. Определяется четкое значение переменной вывода (как взвешенное среднее z_1 и z_2):

$$z_0 = \frac{\alpha_1 z_1 + \alpha_2 z_2}{\alpha_1 + \alpha_2};$$

в общем случае (дискретный вариант центроидного метода)

$$z_0 = \frac{\sum_{i=1}^{n} \alpha_i z_i}{\sum_{i=1}^{n} \alpha_i}.$$

Пример. Пусть имеем $A_1(x_0) = 0{,}7$, $A_2(x_0) = 0{,}6$, $B_1(y_0) = 0{,}3$, $B_2(y_0) = 0{,}8$, соответствующие уровни отсечения

$a_1 = \min(A_1(x_0), B_1(y_0)) = \min(0{,}7; 0{,}3) = 0{,}3$,

$a_2 = \min(A_2(x_0), B_2(y_0)) = \min(0{,}6; 0{,}8) = 0{,}6$

и значения $z_1 = 8$ и $z_2 = 4$, найденные в результате решения уравнений $C_1(z_1) = 0{,}3$, $C_2(z_2) = 0{,}6$.

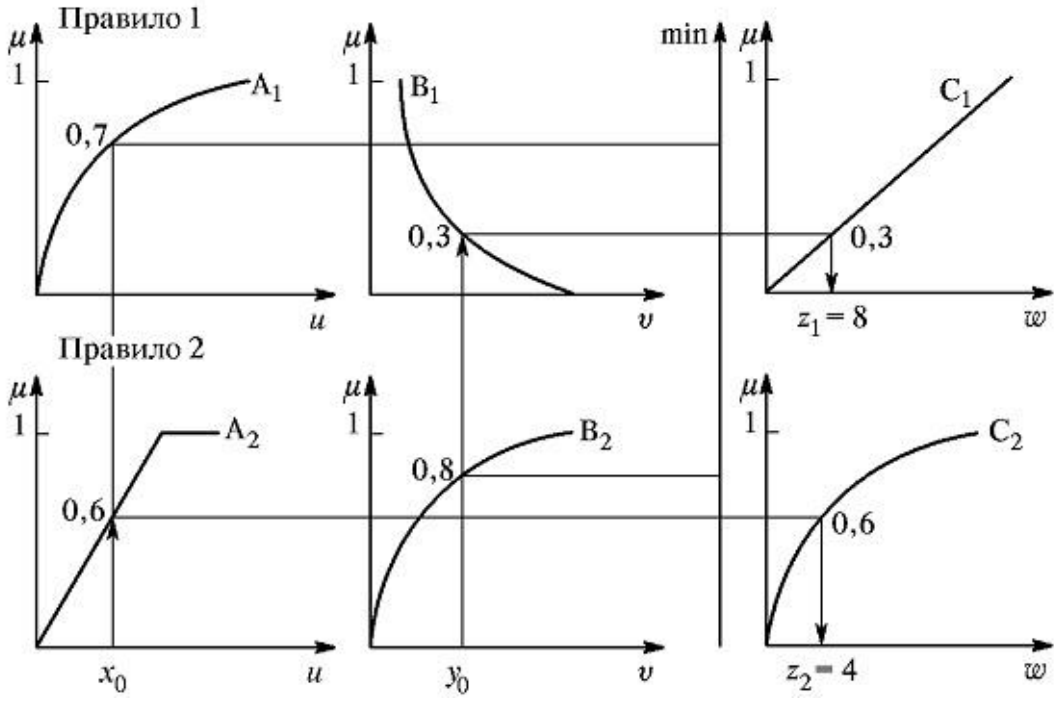


Рис.5.5. Иллюстрации к алгоритму Tsukamoto

При этом четкое значение переменной вывода (см. рис. 5.5)
$z_0 = (8 \cdot 0{,}3 + 4 \cdot 0{,}6) / (0{,}3 + 0{,}6) = 6$.

Алгоритм Larsen

В алгоритме Larsen нечеткая импликация моделируется с использованием оператора умножения.

Описание алгоритма

1. Первый этап — как в алгоритме Mamdani.
2. На втором этапе, как в алгоритме Mamdani вначале находятся значения

$\alpha_1 = A_1(x_0) \wedge B_1(y_0)$,
$\alpha_2 = A_2(x_0) \wedge B_2(y_0)$,

а затем — частные нечеткие подмножества

$\alpha_1 C_1(z)$, $a_2 C_2(z)$.

3. Находится итоговое нечеткое подмножество с функцией принадлежности

$\mu_s(z) = C(z) = (a_1 C_1(z)) \vee (a_2 C_2(z))$

(в общем случае n правил).

4. При необходимости производится приведение к четкости (как в ранее рассмотренных алгоритмах).

Алгоритм Larsen иллюстрируется рис. 5.6.

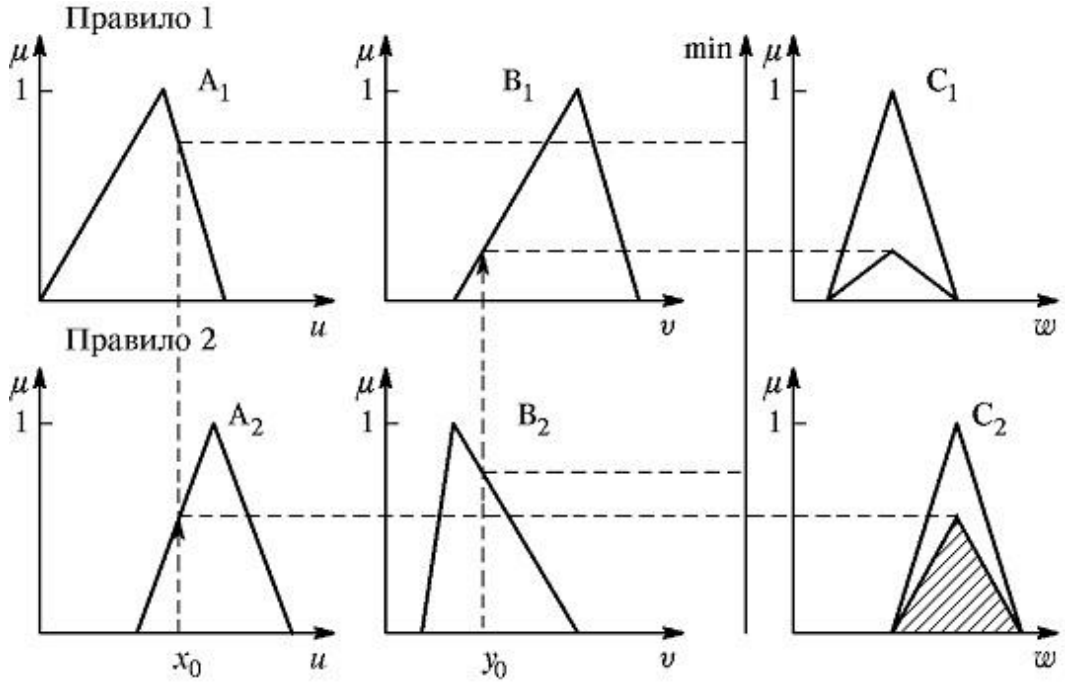


Рис. 5.6. Иллюстрация алгоритма Larsen

Упрощенный алгоритм нечеткого вывода

Исходные правила в данном случае задаются в виде:

П$_1$: если x есть A$_1$ и y есть B$_1$, тогда $z_1 = c_1$,

П$_2$: если x есть A$_2$ и y есть B$_2$, тогда $z_2 = c_2$, где c_1 и c_2 — некоторые обычные (четкие) числа.

Описание алгоритма

1. Первый этап — как в алгоритме Mamdani.

2. На втором этапе находятся числа $\alpha_1 = A_1(x_0) \wedge B_1(y_0)$, $\alpha_2 = A_2(x_0) \wedge B_2(y_0)$.

3. На третьем этапе находится четкое значение выходной переменной по формуле

$$z_0 = \frac{\alpha_1 c_1 + \alpha_2 c_2}{\alpha_1 + \alpha_2},$$

или — в общем случае наличия n правил — по формуле

$$z_0 = \frac{\sum_{i=1}^{n} \alpha_i c_i}{\sum_{i=1}^{n} \alpha_i}.$$

Иллюстрация алгоритма приведена на рис. 5.7.

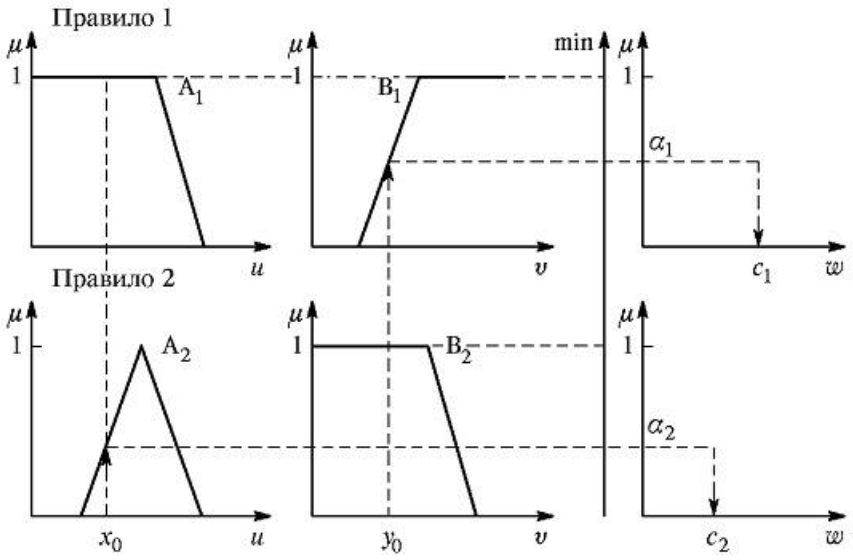


Рис. 5.7. Иллюстрация упрощенного алгоритма нечеткого вывода

Методы приведения к четкости

1. Выше уже был рассмотрен один из данных методов — центроидный. Приведем соответствующие формулы еще раз.

Для непрерывного варианта:

$$z_0 = \frac{\int\limits_{\Omega} zC(z)\,dz}{\int\limits_{\Omega} C(z)\,dz};$$

для дискретного варианта:

$$z_0 = \frac{\sum\limits_{i=1}^{n} \alpha_i z_i}{\sum\limits_{i=1}^{n} \alpha_i}.$$

2. Первый максимум (First-of-Maxima). Четкая величина переменной вывода находится как наименьшее значение, при котором достигается максимум итогового нечеткого множества, т.е. (см. рис. 5.8а)

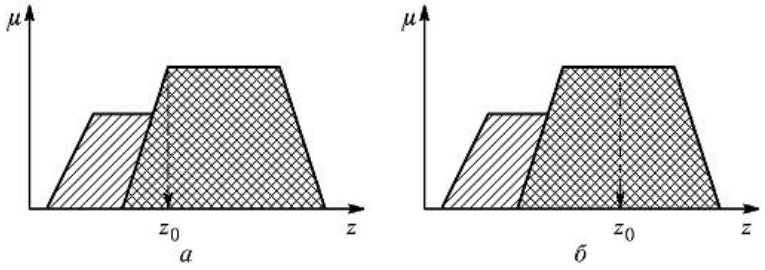


Рис. 5.8. Иллюстрация к методам приведения к четкости: α — первый максимум; б — средний максимум

$$z_0 = \min\left(z | \mathrm{C}(z) = \max_u \mathrm{C}(u)\right).$$

3. Средний максимум (Middle-of-Maxima). Четкое значение находится по формуле

$$z_0 = \frac{\int\limits_{\mathrm{G}} z\,dz}{\int\limits_{\mathrm{G}} dz},$$

где G — подмножество элементов, максимизирующих C (см. рис. 5.8б).

Дискретный вариант (если C — дискретно):

$$z_0 = \frac{1}{N}\sum_{j=1}^{N} z_j.$$

4. Критерий максимума (Max-Criterion). Четкое значение выбирается произвольно среди множества элементов, доставляющих максимум C, т.е.

$$z_0 \in \left\{z | \mathrm{C}(z) = \max_u \mathrm{C}(u)\right\}.$$

5. Высотная дефазификация (Height defuzzification). Элементы области определения Ω для которых значения функции принадлежности меньше, чем некоторый уровень α в расчет не принимаются, и четкое значение рассчитывается по формуле

$$z_0 = \frac{\int\limits_{\mathrm{C}\alpha} z\mathrm{C}(z)\,dz}{\int\limits_{\mathrm{C}\alpha} \mathrm{C}(z)\,dz},$$

где Cα — нечеткое множество α-уровня (см. выше).

5.4. Последовательность создания нечеткого контроллера

Для данной задачи будем использовать алгоритм Сугэно.

Требуемые действия отобразим следующими пунктами.

1. В меню «File» выбираем команду «NewSugenoFIS» (Новая система типа «Sugeno»), при этом в блоке, отображаемом белым квадратом в верхней части окна редактора, появится надпись «Untitled (sugeno)».

2. Щелкнем на блоке, озаглавленном «inputl». Затем в правой части редактора в поле «Name» (Имя) вместо лингвистической пере-

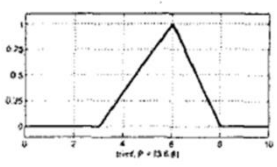
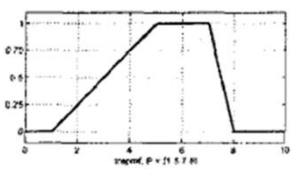
менной «inputl» введем обозначение нашего аргумента, т.е. «отклонение скорости». Лингвистической переменной (linguistic variable) называется переменная, значениями которой могут быть слова или словосочетания некоторого естественного или искусственного языка.

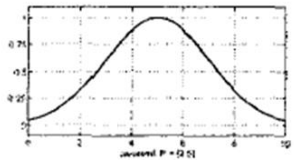
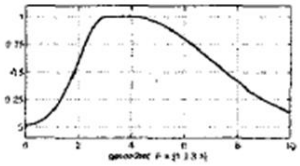
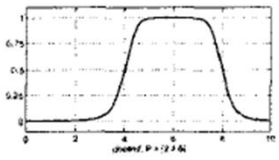
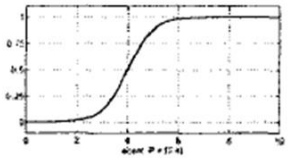
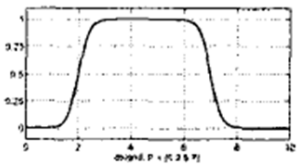
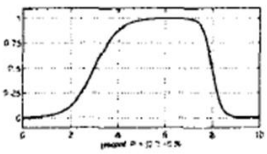
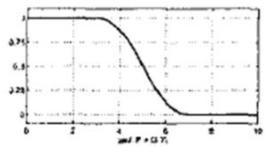
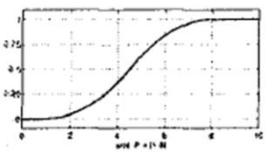
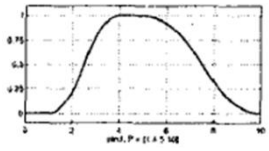
Обратим внимание, что если теперь сделать где-нибудь (вне блоков редактора) однократный щелчок, то имя отмеченного блока изменится на отклонение скорости; то же достигается нажатием клавиши Enter после ввода. Добавлем второй вход (открываю меню «Edit» данного редактора и выбираю в нем команду «Add Variable → Input»). Аналогично обозначаем вход 2 как «отклонение тока».

3. В нечеткой логике для задания функций принадлежности используются типовые формы функций принадлежности (табл.5.1):
Дважды щелкнем на этом блоке. Перед нами откроется окно редактора функций принадлежности –«Membership Function Editor» (рис.5.9). Откроем меню «Edit» данного редактора и выберем в нем команду «Add MFs» (Add Membership Functions–добавить функции принадлежности). При этом появится диалоговое окно (рис.5.10), позволяющее задать тип (MF type) и количество (Number of MFs) функций принадлежности (в данном случае все относится к входному сигналу 1, то есть к переменной «отклонение скорости»). Выберем трапецеидальные функции принадлежности (trapmf), т.к. они наиболее полно описывают все точки, принадлежащие кривой отклонений входных сигналов нечеткой системы, а их количество задаю равным девяти — по числу выбранных значений интервалов отклонений скорости.

Таблица 5.1.

Типовые функции принадлежности

N	Функция активации	Обозначение	Вид функции
1	треугольная	trimf	
2	трапецеидальная	trapmf	

N	Функция активации	Обозначение	Вид функции
3	гауссова	gaussmf	
4	двойная гауссова	gauss2mf	
5	обобщенная колоколообразная	gbellmf	
6	сигмоидальная	sigmf	
7	двойная сигмоидальная	dsigmf)	
8	произведение двух сигмоидальных функций	psigmf)	
9	Z-функция		
10	S-функция		
11	Pi-функция		

Конкретный вид данных функций определяется значениями параметров, входящих в их аналитические представления.

Подтвердим ввод информации нажатием кнопки «ОК», после чего произойдет возврат к окну редактора функций принадлежности. Аналогично задаю функции принадлежности для входа 2, число которых соответствует заданной градации отклонений тока, т.е. пяти.

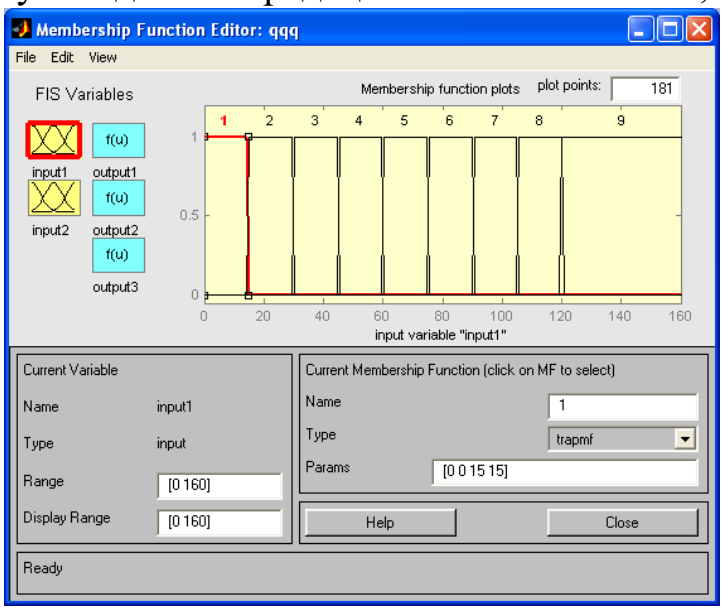


Рис.5.9. Окно редактора функций принадлежности

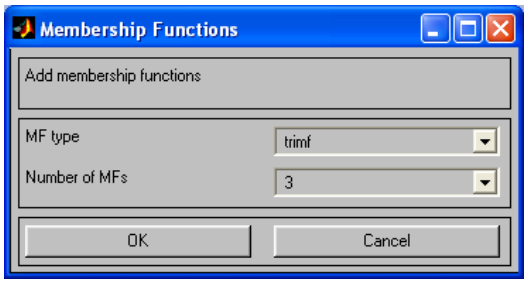


Рис.5.10. Диалоговое окно задания типа и количества функций принадлежности входной переменной

4. В поле «Range» (Диапазон) установим диапазон изменения отклонений скоростиот 0 до 160, т.к. при большем отклонении данного параметра можно сделать вывод о неисправности двигателя. Щелкнем затем левой кнопкой мыши где-нибудь в поле редактора (или нажмем клавишу Enter). Обратим внимание, что после этого произойдет соответствующее изменение диапазона в поле «Display-Range» (Диапазон отображения).

5. Редактирую функции принадлежности, задавая в поле «Params» (Параметры) числовые значения (в данном случае каждой функции принадлежности соответствуют четыре параметра, при этом первый и четвертый определяет размах кривой, а второй и третий —

положение ее основания). В поле «Nam»е изменяю имя для выбранной кривой (завершая ввод каждого имени нажатием клавиши Enter).

Нажмем кнопку «Close» и выхожу из редактора функций принадлежности, возвратившись при этом в окно редактора нечеткой системы (FIS Editor).

6. Добавляю два выхода (открываю меню «Edit» данного редактора и выбираю в нем команду «AddVariable→Output»). Сделаем однократный щелчок на голубом квадрате (блоке), озаглавленном «outputl» (Выход1). В поле Name заменю имя outputl на «перегрев». Подобным образом заменим имя «output2» на «увеличение момента инерции», имя «output3» на «превышение номинальной нагрузки».

7. Дважды щелкну на выделенном блоке, и перейдем к редактору функций принадлежности. В меню «Edit» выберем команду «Add MFs». Появляющееся затем диалоговое окно вида (рис.5.11) позволяет теперь задать в качестве функций принадлежности только линейные (linear) или постоянные (constant) — в зависимости от того, какой алгоритм Сугэно (1-го или 0-го порядка) мы выбираем. В рассматриваемой задаче необходимо выбрать постоянные функции принадлежности. Подтвердим введенные данные нажатием кнопки «ОК», после чего произойдет возврат в окно редактора функций принадлежности.

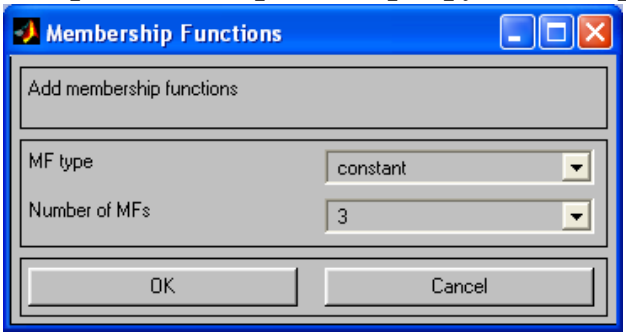


Рис.5.11. Диалоговое окно задания типа и количества функций принадлежности выходной переменной

8. Обратим внимание, что диапазон изменения (Range), устанавливаемый по умолчанию - [0,1], - менять в данном случае не нужно. Изменим лишь имена функций принадлежности (их графики при использовании алгоритма Сугэно для выходных переменных не приводятся), например, задав их как соответствующие числовые значения выходных переменных, одновременно эти же числовые значения введем в поле «Params» (рис.5.12). Затем закроем окно нажатием кнопки «Close» и вернемся в окно FIS-редактора.

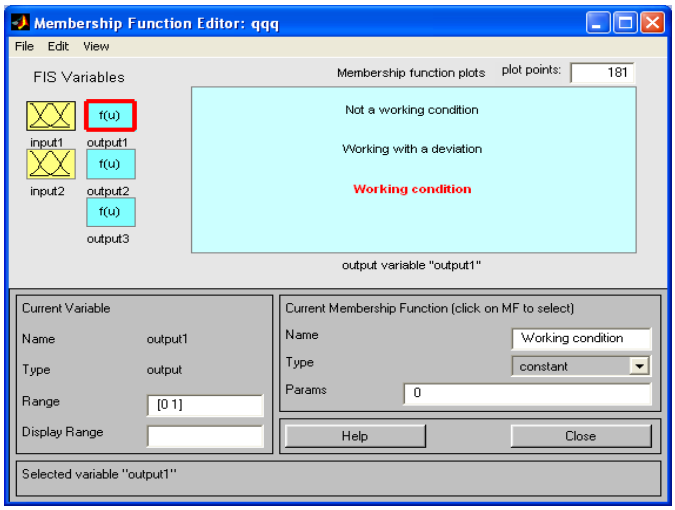


Рис.5.12. Параметры функций принадлежности выходной переменной

Для установки соответствия между функциями принадлежности входных и выходных сигналов создаю систему правил (см. табл.5.2), присваивая каждой функции принадлежности в MATLAB, соответствующее лингвистическое значение переменной, несущее заданный физический смысл (табл.5.1).

Таблица 5.2

Таблица соответствия

Функция принадлежности	Лингвистическое значение
отклонения скорости	
1	очень малое
2	малое
3	много ниже среднего
4	ниже среднего
5	среднее
6	выше среднего
7	много выше среднего
8	большое
9	очень большое
отклонение тока	
1	малое
2	ниже среднего
3	среднее
4	выше среднего
5	большое

Таблица 5.3

Система правил

откл.тока / откл.ск-ти	малое	ниже среднего	среднее	выше среднего	большое
очень малое	увеличение момента нагрузки в допустимых пределах	увеличение момента инерции в допустимых пределах	увеличение сопротивления якорной цепи	не допустимое увеличение сопротивления якорной цепи	не допустимое увеличение сопротивления якорной цепи
малое	увеличение момента нагрузки в допустимых пределах	увеличение момента инерции в допустимых пределах	увеличение сопротивления якорной цепи	-	не допустимое увеличение сопротивления якорной цепи
много меньше среднего	увеличение момента нагрузки в допустимых пределах	увеличение момента инерции в допустимых пределах	увеличение сопротивления якорной цепи	-	не допустимое увеличение сопротивления якорной цепи
меньше среднего	увеличение момента нагрузки в допустимых пределах	увеличение момента инерции в допустимых пределах	увеличение сопротивления якорной цепи	-	не допустимое увеличение сопротивления якорной цепи
среднее	увеличение сопротивления якорной цепи	увеличение момента нагрузки в допустимых пределах	увеличение момента инерции в допустимых пределах	-	не допустимое увеличение сопротивления якорной цепи
больше среднего	увеличение сопротивления якорной цепи	превышение допустимого момента нагрузки	превышение допустимого момента инерции	не допустимое увеличение сопротивления якорной цепи	не допустимое увеличение сопротивления якорной цепи
много больше среднего	не допустимое увеличение сопротивления якорной цепи	превышение допустимого момента нагрузки	превышение допустимого момента инерции	не допустимое увеличение сопротивления якорной цепи	-
большое	не допустимое увеличение сопротивления якорной цепи	-	превышение допустимого момента нагрузки	превышение допустимого момента инерции	-
очень большое	не допустимое увеличение сопротивления якорной цепи	не допустимое увеличение сопротивления якорной цепи	превышение допустимого момента нагрузки	превышение допустимого момента инерции	-

9. Дважды щелкнем на среднем (белом) блоке, при этом раскроется окно еще одной программы — редактора правил (Rule Editor). Введем соответствующие правила. При вводе каждого правила необходимо обозначить соответствие между каждой функцией принадлежности входных переменных и числовым значением выходных переменных. Выберу в левых полях (с заголовками «отклонение скорости» и «отклонение тока») вариант, соответствующий состоянию электродвигателя, описанному в полях выходов и нажму кнопку «Addrule» (Добавить правило). Введенное правило появится в окне правил. Аналогично поступим для всех других значений входных переменных, в результате чего сформируется набор правил (рис.5.13). Закроем окно редактора правил и возвратимся в окно FIS-редактора. Построение системы закончено, и можно начать эксперименты по ее исследованию. Заметим, что для большинства опций были сохранены значения по умолчанию.

10. Предварительно сохраню на диске (используя команды меню «File→Save to diskas») созданную систему под каким-либо именем.

Раскрою меню «View». С помощью его команд «Edit member ship functions» и «Edit rules» можно совершить переход к двум рассмотренным ранее программам – редакторам функций принадлежности и правил (то же можно сделать нажатием клавиш Ctrl+2 и Ctrl+3), но сейчас нас будут интересовать две других команды –«View rules» (Просмотр правил) и «View surface» (Просмотр поверхности).

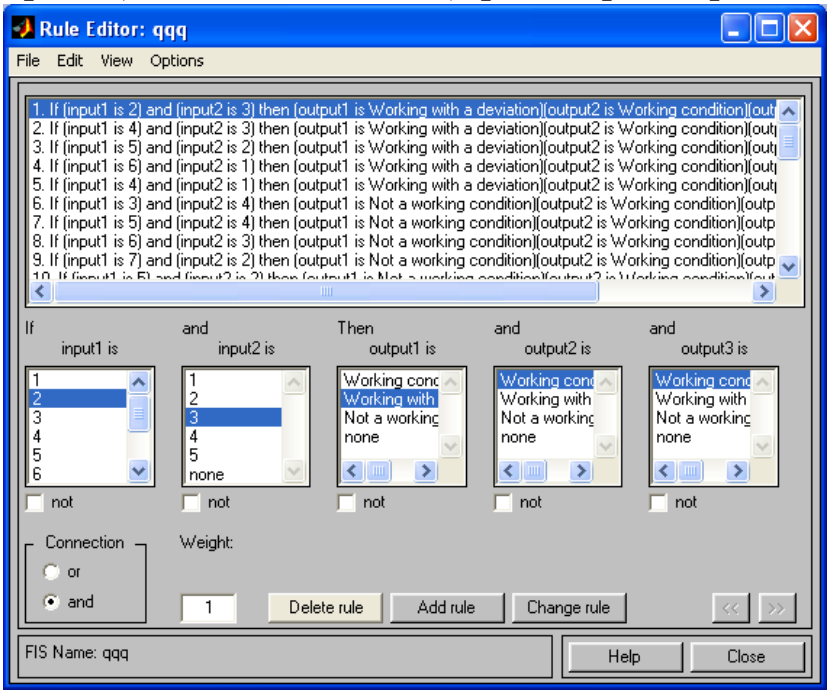


Рис.5.13. Окно редактора правил

Выберем команду «View rules», при этом откроется окно (рис.5.14) еще одной программы - просмотра правил (Rule Viewer).

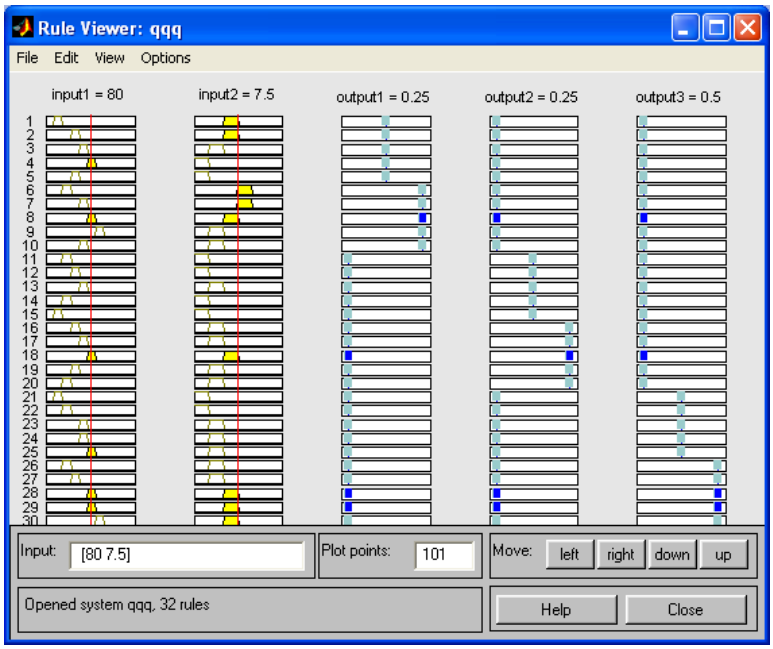


Рис.5.14. Окно просмотра правил

11. В левой части окна в графической форме представлены функции принадлежности входных переменных «отклонение скорости» и «отклонение тока», в правой - функции принадлежности выходных переменныхс пояснением механизма принятия решения. Красная вертикальная черта, пересекающая графики в правой части окна, которую можно перемещать с помощью мыши, позволяет изменять значения переменной входа (это же можно делать, задавая числовые значения в поле «Input» (Вход)), при этом соответственно изменяются значения выходных параметров правой верхней части окна. Таким образом, с помощью построенной модели и окна просмотра правил можно просмотреть все возможные варианты комбинаций входных параметров «отклонение скорости» и «отклонение тока» и соответствующие им выходные переменные. Изменение аргумента путем перемещения красной вертикальной линии очень наглядно демонстрирует, как система определяет значения выхода.

12. Закроем окно просмотра правил и выбором команды меню «View→Views urface» перейдем к окну просмотра поверхности отклика (выхода) (рис.5.15). Видно, что смоделированное системой отображение показывает зависимость выхода от конкретных значений

входных величин. В поле Output можно выбрать выход, для которого необходимо просмотреть зависимость от входных параметров.

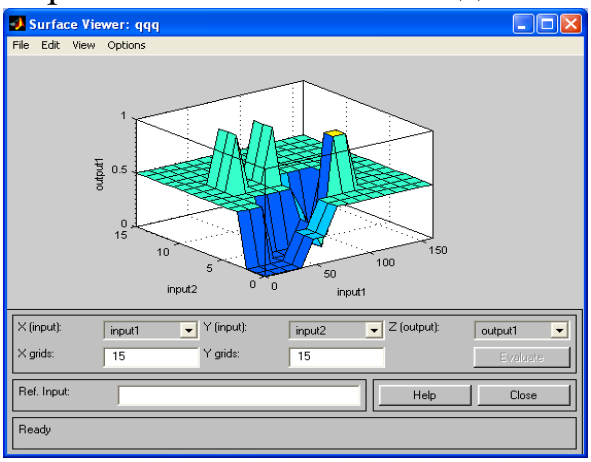


Рис.5.15. Окно просмотра поверхности отклика

С помощью вышеуказанных программ-редакторов на любом этапе проектирования нечеткой модели в нее можно внести необходимые коррективы, вплоть до задания какой-либо особенной пользовательской функции принадлежности. Из опций, устанавливаемых в FIS-редакторе по умолчанию при использовании алгоритма Сугэно, можно отметить следующие:

• логический вывод организуется с помощью операции умножения (prod);

• композиция организуется с помощью операции логической суммы (вероятностного ИЛИ, probor);

• приведение к четкости организуется дискретным вариантом центроидного метода (взвешенным средним, wtaver).

Используя соответствующие поля в левой нижней части окна FIS-редактора, данные опции можно при желании изменить.

Работа Fuzzy Logic с«Simulink».

Системы нечеткого вывода, созданные тем или иным образом с помощью пакета «Fuzzy Logic Toolbox», допускают интеграцию с инструментами пакета «Simulink», что позволяет выполнять моделирование систем в рамках последнего.

Блоки для пакета «Simulink»

FUZBLOCK

Библиотека блоков для внедрения нечетких контроллеров в «Simulink»-модули.

Синтаксис: **fuzblock.**

Библиотека содержит следующие блоки:

- Fuzzy Logic Controller – нечеткий контроллер;
- Fuzzy Logic Controller with Rule viewer – нечеткий контроллер с выводом окна RuleViewer во время моделирования в пакете«Simulink»;
- Membership Functions – библиотека функций принадлежностей.

Для включения системы нечеткого логического вывода в «Simulink»-модуль необходимо выбрать блок «Fuzzy Logic Controller» или «Fuzzy Logic Controller with Ruleviewer», затем сделать двойной щелчок по этому блоку и в появившимся диалоговом окне ввести имя файла или наименование переменной в рабочей области, соответствующим системе нечеткого логического вывода**.** Если система нечеткого логического вывода имеет несколько входов, тогда в «Simulink»-модуле эти входы необходимо мультиплексировать вместе до ввода в нечеткий контроллер. Аналогично, если система нечеткого логического вывода имеет несколько выходов, тогда выходные сигналы блока будут представлены одной мультиплексной линией.

Для построения нетиповых нечетких контроллеров, т. е. отличных от блоков «Fuzzy Logic Controller» и «Fuzzy Logic Controller with Rule viewer» можно использовать блоки, входящие в библиотеку функций принадлежности (Membership Functions):

- Diff. Sigmoidal MF – разница двух сигмоидных функций принадлежности;
- Gaussian MF – гауссовская функция принадлежности;
- Gaussian 2MF – комбинация двух гауссовских функций принадлежности;
- Generalized Bell MF – обобщенная колокообразная функция принадлежности;
- Pi-shared MF – пи - подобная функция принадлежности;
- Probabilistic OR – вероятностная реализация логической операции ИЛИ;
- Probabilistic Rule Agg – вероятностная реализация операции объединения правил;
- Prod. Sigmoidal MF – произведение двух сигмоидных функций принадлежности;

- S-shaped MF – S-подобная функция принадлежности;
- Sigmoidal MF – сигмоидная функция принадлежности;
- Trapezoidal MF – трапециевидная функция принадлежности;
- Triangular MF – треугольная функция принадлежности;
- Z-shaped MF – Z-подобная функция принадлежности.

SFFIS

Оптимизированная под «Simulink» функция нечеткого логического вывода.

Синтаксис:**output = sffis (t, x, u, flag, fis).**

Функция «sffis» - это MEX-файл, специально оптимизированный для использования в пакете «Simulink». Функция «sffis» выполняет нечеткий логический вывод аналогично функции «evalfis». Функция «sffis» имеет 5 входных аргументов:

- t, x, и «flag» - стандартные аргументы S-функций пакета «Simulink»;
- u - вектор значений входных переменных, для которых необходимо осуществить нечеткий логический вывод;
- «fis» - система нечеткого логического вывода.

Функция «sffis» возвращает единственный аргумент output, содержащий результат нечеткого логического вывода

В данной случае была разработана модель электродвигателя, включаемая параллельно реальному двигателю. Сигналы с их выходов сравниваются и подаются на вход нечеткого контроллера. На рисунке 5.16 изображена модель системы диагностики.

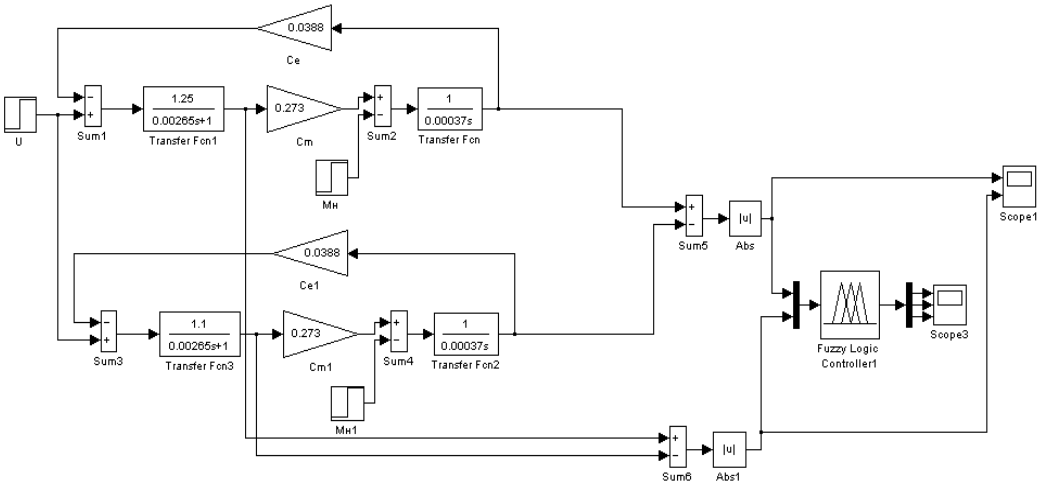


Рис.5.16. Модель системы диагностики

Состояние электродвигателя оценивается по величине отклонения выходных сигналов. В зависимости от значений скорости и тока нечеткий контроллер на определенном выходе формирует сигнал, соответствующий правилам, прописанным в окне редактора правил.

Результаты моделирования нечеткой системы диагностики

На рисунке 5.17 представлены результаты работы нечеткой системы диагностики электродвигателя, работающего при номинальных значениях. Из графиков видно, что на всех выходах нечеткой системы отсутствуют отклонения, что свидетельствует о ее исправности. Далее моделировала различные режимы работы двигателя. Чтобы получить выходные характеристики при перегреве двигателя, меняла сопротивление якорной цепи. Результаты представлены на рисунке 5.17 На рисунке 5.18 – 19 и рисунке 5.20 - 21 получены выходные сигналы нечеткой системы при изменении момента инерции и при увеличении нагрузки на валу двигателя соответственно.

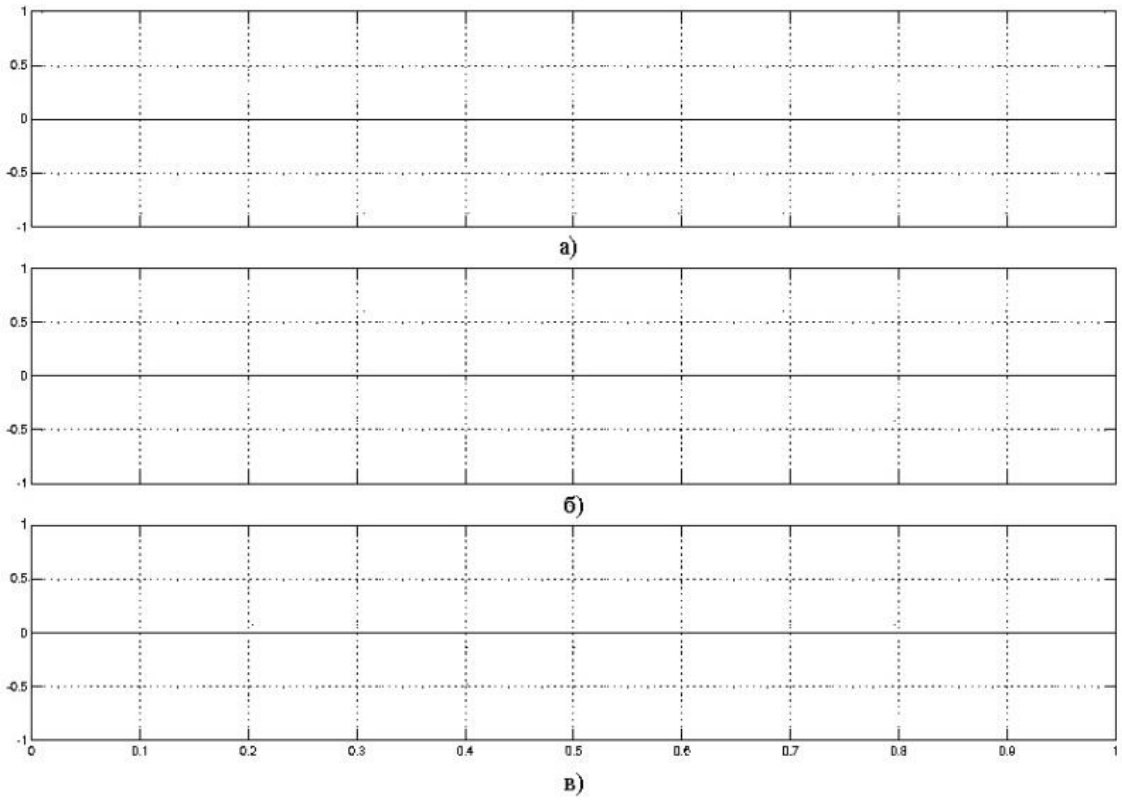


Рис. 5.17. Результаты моделирования системы диагностики при номинальных параметрах: а – перегрев двигателя, б – изменение момента инерции, в – увеличение момента на валу двигателя

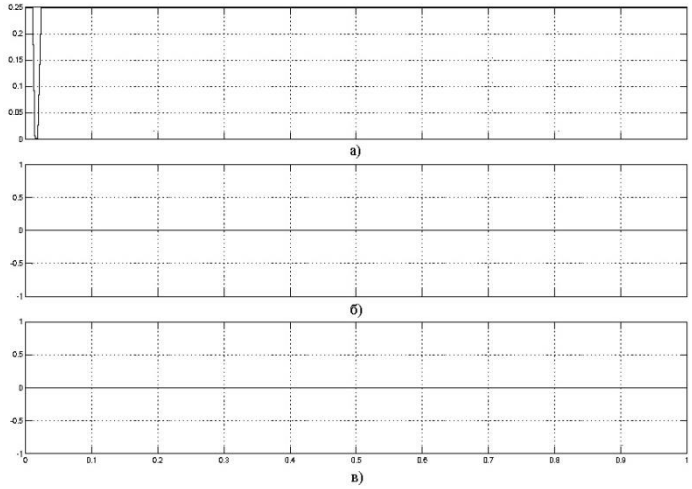


Рис.5.18. Результаты моделирования при изменении температуры двигателя

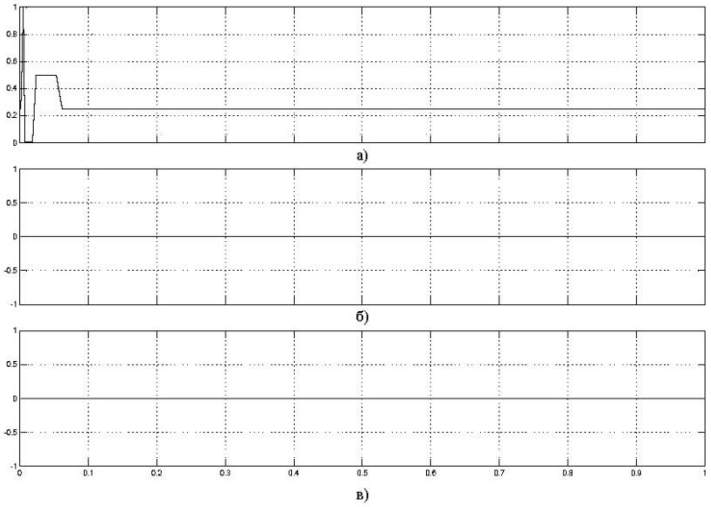


Рис.5.19. Результаты моделирования при дальнейшем нагреведвигателя

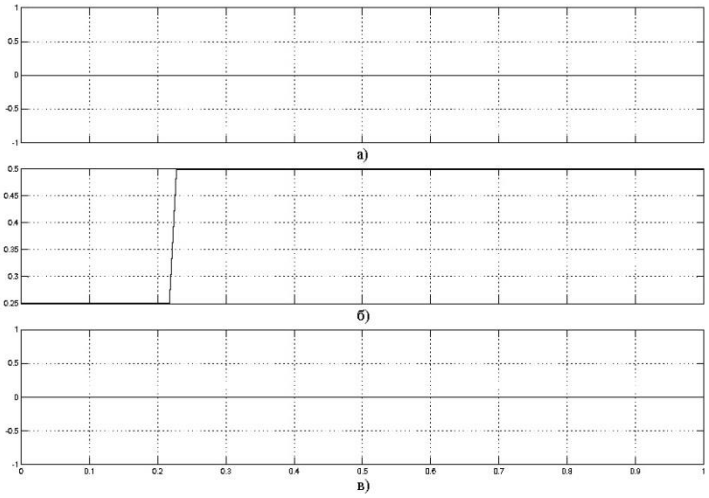


Рис.5.20. Результаты моделирования при изменении момента инерции

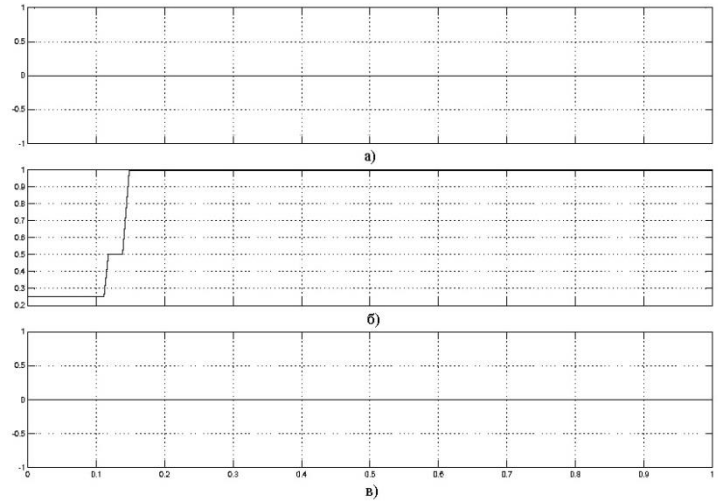


Рис.5.21. Результаты моделирования при дальнейшем изменении момента инерции

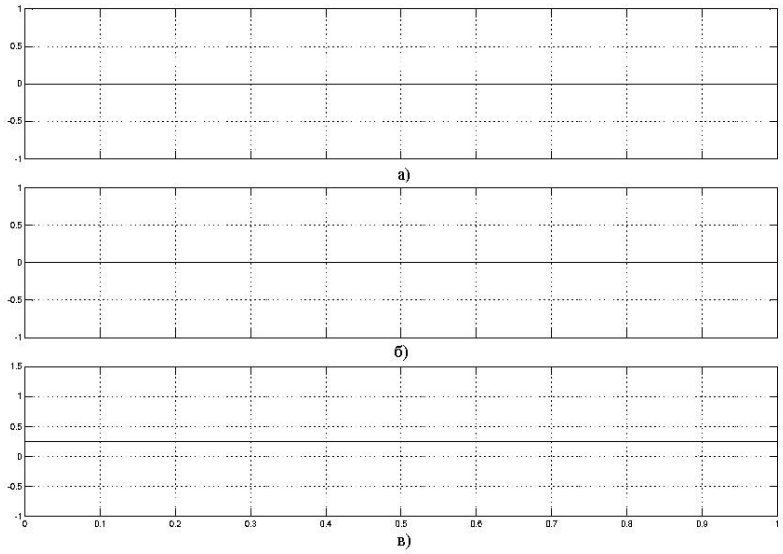


Рис.5.22. Результаты моделирования при увеличении нагрузки на валу двигателя

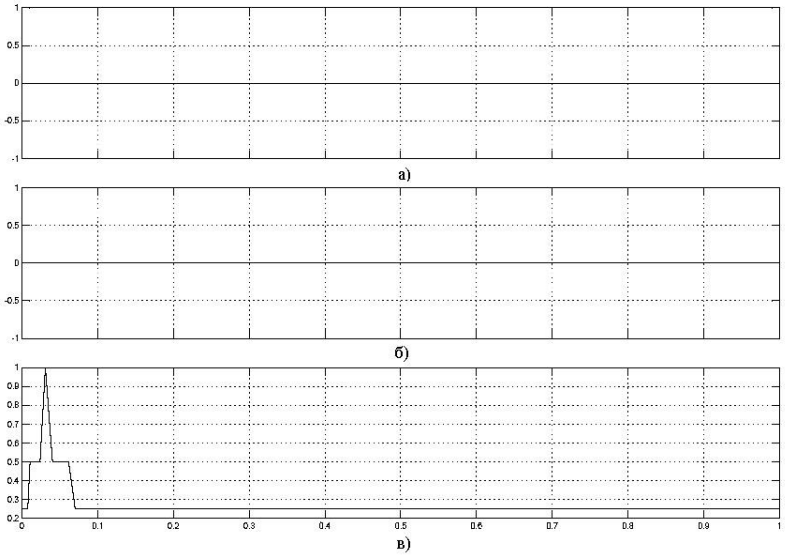


Рис.5.23. Результаты моделирования при дальнейшем увеличении нагрузки на валу двигателя

В процессе работы двигателя, при изменении каких-либо параметров на выходе нечеткой системы диагностики, соответствующем конкретной неисправности, формируется сигнал, при оценке которого можно делать выводы о состоянии двигателя.

Преимущества нечетких систем

Коротко перечислим преимущества fuzzy-систем по сравнению с другими:

- возможность оперировать нечеткими входными данными: например, непрерывно изменяющиеся во времени значения (динамические задачи), значения, которые невозможно задать однозначно (результаты статистических опросов, рекламные компании и т.д.);
- возможность нечеткой формализации критериев оценки и сравнения: оперирование критериями "большинство", "возможно", преимущественно" и т.д.;
- возможность проведения качественных оценок как входных данных, так и выходных результатов: вы оперируете не только значениями данных, но и их степенью достоверности (не путать с вероятностью!) и ее распределением;
- возможность проведения быстрого моделирования сложных динамических систем и их сравнительный анализ с заданной степенью точности: оперируя принципами поведения системы, описанными fuzzy-методами, вы, во-первых, не тратите много времени на выяснение точных значений переменных и составление описывающих уравнений, во-вторых, можете оценить разные варианты выходных значений.

Рассмотренный в данной работе вариант построения систем диагностирования, основанный на применении нечеткой логики показал значительные преимущества данных систем. Анализ полученных результатов моделирования позволяет сделать выводы о высокой точности определения состояния электромеханической системы с помощью данного метода. При использовании нечеткой логики снижается время идентификации состояния, что позволяет своевременно реагировать на изменение каких-либо параметров и вовремя предупредить неисправность.

5.5. Создание привода с нечетким контроллером

Построение модели и исследование работы привода

В качестве примера рассмотрим электромеханический привод руки робота, который не охвачен обратной связью по выходной координате.

В качестве объекта будем рассматривать привод механизма перемещений с главной обратной связью по положению, причем сам механизм не охвачен обратной связью. Известные результаты исследований показали, что отсутствие полной информации о регулируемой координате резко снижает показатели работы привода, особенно при изменяющихся воздействиях вызванных технологическим процессом.

Рассмотрим систему, которую можно описать каноническими уравнениями второго порядка:

$$\dot{x_1} = x_2$$

$$\dot{x_2} = -a_1 x_1 - a_2 x_2 + u, \qquad (1)$$

где $\dot{x_1}$ и $\dot{x_2}$ переменные состояния и управляющее воздействие:

$$u = -\Psi x_1. \qquad (2)$$

Здесь Ψ параметр переключаемой структуры, величина которого может изменяться по закону:

$$\Psi = \begin{cases} d_1 \text{при} & Sx_1 > 0 \\ d_2 \text{при} & Sx_1 < 0 \end{cases}. \qquad (3)$$

Согласно теории систем с переменной структурой необходимо обеспечить условие, при котором произведение $S\dot{S}<0$.Для простоты реализации рассматривается система второго порядка. В качестве переменных используем ошибку регулятора положения и скорость вращения ротора двигателя. Если обозначить ошибку регулятора положения через x_1, а скорость черезx_2 то уравнение скольжения запишется $S=Cx_1+x_2$. Решение этого уравнения при $c>0$ является устойчивым и, следовательно, систем с переменной структурой при произвольных начальных условиях также устойчива. Закон изменения структуры основан на анализе произведения Sx_1. Для обеспечения стабилизации движения необходимо в управление добавить производную от координаты x_1. Изменением структуры удастся парировать возмущение, недоступное для измерения. В этом случае можно добиться ошибки управления близкой к нулю.

Представим разрабатываемую модель исходной схемой, реализующей рассматриваемый подход (рис.5.24).

Перейдем теперь к выводу расчетных соотношений, которые позволят реализовать подход к построению системы с переменной структурой, предназначенной для управления движением объекта с неконтролируемыми возмущениями

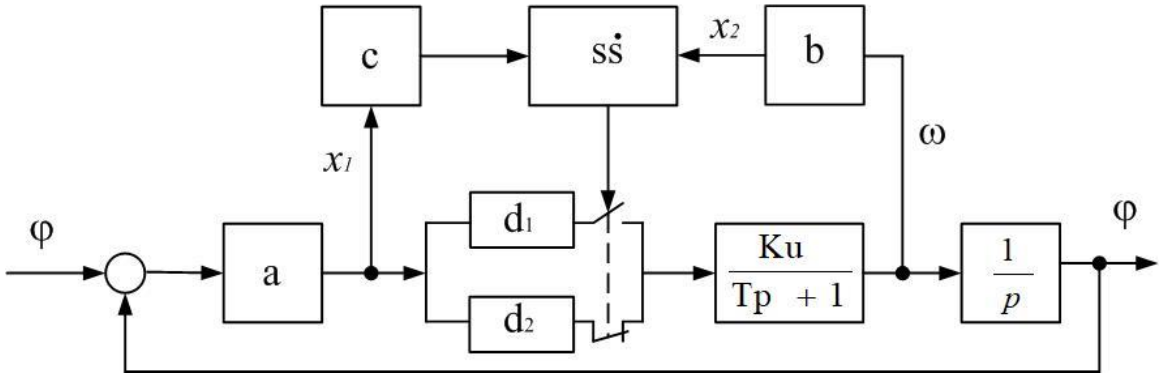


Рис.5.24. Электромеханическая система

Будем рассматривать систему, представленную на рис.5.24 с исполнительным устройством, описываемым уравнением:

$\frac{\varphi}{u} = \frac{k}{p(Tp+1)}$.

Представим предыдущее выражения уравнениями в пространстве состояний, аналогичное (1):

$$\dot{x_1} = \frac{a}{b} x_2,$$
$$\dot{x_2} = \frac{1}{T}(bku - x_2\ )$$

В этом выражении $x_1 = a\varphi$, x_2= bω и a и b коэффициенты, связывающие параметры датчиков с характеристиками привода.

Линия переключения определяется в соответствии с уравнением S=cx_1+x_2 (c –положительная константа), а производная

$\dot{S}$ = (ca/b – 1/T) x_2+(bkΨ/T) x_1

Вблизи линии переключения x_2= –cx_1, поэтому

$\dot{S}$ = (–ca/b + 1/T) c x_1+(bkΨ/T) x_1

В силу необходимости выполнения условия S$\dot{S}$<0, имеем

$[-c^2a/b + c/T + bk\Psi/T]\, x_1 S < 0.$

Переключение структур основано на анализе произведения $x_1 S$

$$\Psi = \begin{cases} -\alpha, \text{ если } x_1 S > 0 \\ \alpha, \text{ если } x_1 S < 0 \end{cases},$$

и с учетом (2, 3) управление изменяет свой знак.

Величина α вычисляется по выражению:

$\alpha = T/bk(c^2a/b + c/T)$.

Теперь можно построить модель привода. В качестве инструментальных средств будем использовать программную систему Matlab. Встроенный в Matlab пакет структурного моделирования Simulink дает возможность простыми и выразительными средствами построит компьютерную модель и выполнить визуализацию результатов. Модель, построенная с использованием стандартных блоков Simulink, приведена на рис.5.25. Схема включает следующие блоки. Схема вычисления произведения $S\dot{S}$ содержит блок суммирования (сумматор 1), блок произведения и инвертор. Переменная *a* перемножается с константой *c* проходя через усилитель C и поступает на вход сумматора, на второй вход которого подается сигнал, представляющий собой произведение коэффициента b на угловую скорость (усилитель b), снимаемую с выхода исполнительного устройства. В результате вычисляется параметр линии переключения $\dot{S}$. Блок произведения обеспечивает перемножения $\dot{S}$ на параметр перемещения a. В результате получаем условие переключения $S\dot{S}$. Выход инвертора управляет переключением структур с помощью ключа.

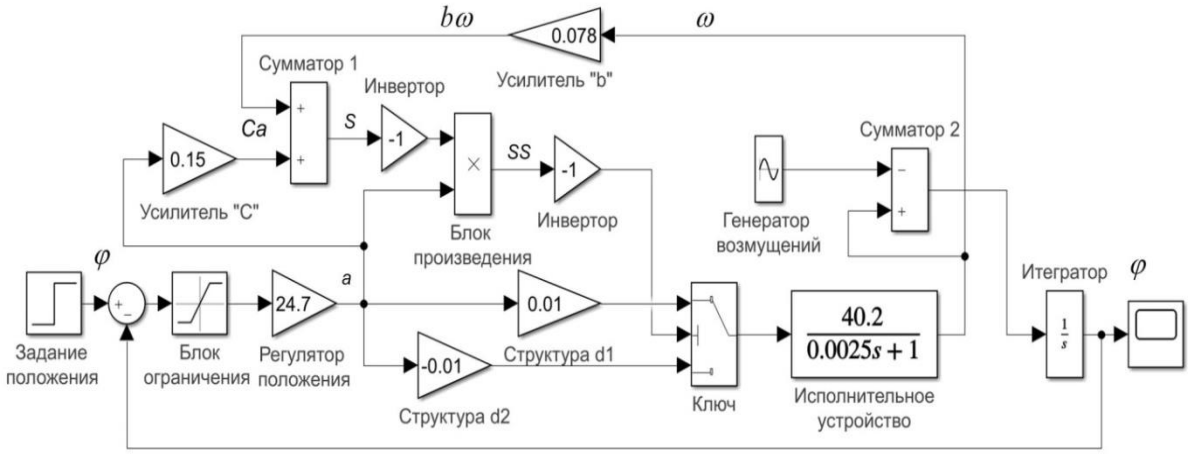


Рис.5.25. Модель привода

Исследование возмущающих режимов проводилось с использованием генератора задающего периодические сигналы различной амплитуды и частоты, построенного с использованием стандартного блока синусоидальных сигналов и сумматора 2.

Для эксперимента установим параметры модели К=40,2, Т=0,0025, a=24,7, b=0,078.

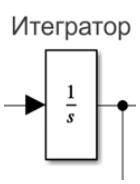
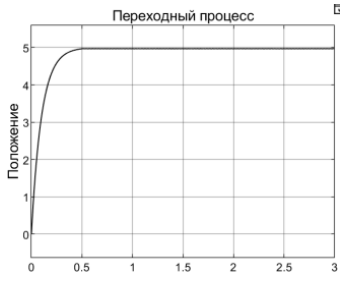
В этом случае параметр α =0,00079(316,6c^2– 400c) = - 0,0417. Величина *c* определяет наклон линии скольжения, значение которой должна быть менее 0,2, а значение коэффициентов переключаемых структур |d| < 0,04.

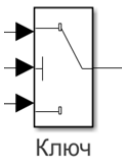
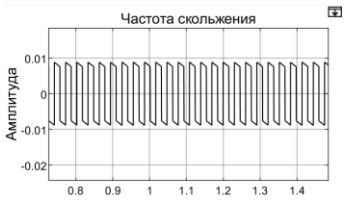
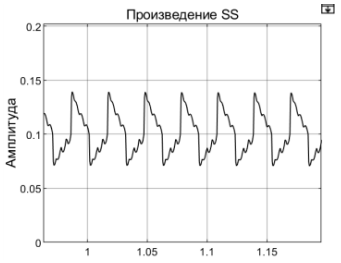
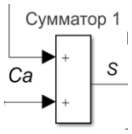
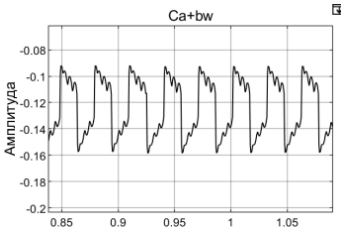
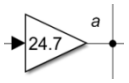
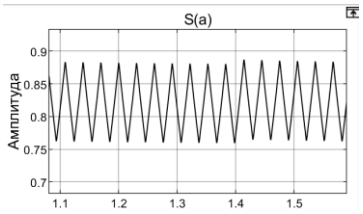
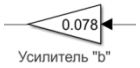
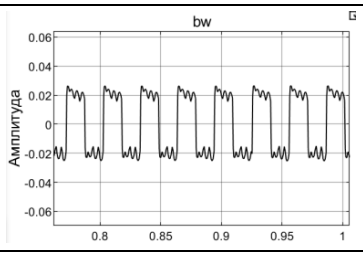
Результаты полученные в процессе моделирования представлены в таблице. В таблице приведено пространство состояния в виде выходных сигналов, точки съема информации и контролируемый параметр. Приведенные результаты соответствуют режиму работы без внешних возмущений. Приведенные расчетные соотношения адекватно отражают поведение работы системы с переменной структурой и скользящим режимом движения.

Проверим влияние изменения параметров привода на выходные характеристики. Изменим параметры регулятора положения в интервале от 10 до 25 и характеристики исполнительного устройства К от 20 до 40 и Т от 0.025с до 0.0003с. Диапазон изменения параметров выбран со значительным разбросом, что не характерно для реальных систем, но отражает возможности рассматриваемой модели. Снижение коэффициента исполнительного устройства в 2 раза не приводит к существенным изменениям, а снижение коэффициента регулятора положения снижает частоту переключения с 30Гц до 16Гц.

Таблица 5.6

Пространство переменных состояния

Контролируемый параметр	Точка контроля	Вид выходного сигнала
Переходный процесс по угловому положению		

Контролируемый параметр	Точка контроля	Вид выходного сигнала
Входное воздействие на исполнительное устройство. Сигнал скольжения (частота переключения).		
Произведение сигнала линии переключения на ее производную		
Сумма сигналов положения и скорости с учетом коэффициентов согласования		
Сигнал перемещения		
Скоростной сигнал		

Остальные параметры не претерпевают изменений. Изменение коэффициента b на порядок увеличивает частоту переключений до 200Гц и повышает точность. Напротив снижение параметра *c* до 0,1 приводит к снижению частоты скольжения при незначительном снижение остальных параметров. Изменение постоянной времени исполнительного устройства при снижении постоянной времени до 0.0025с снижает частоту до 20Гц, а при 0.025с до 8Гц и уровень отклонения от установившегося значения не значительно увеличивается.

Кроме того, что бы убедится в инвариантности к параметрическим изменениям и структурным преобразованиям заменили исход-

ное исполнительное устройство моделью силового преобразователя (апериодическое звено 0.002p+1=7u) и двигателя (колебательное звено $10^{-4}p^2$+0.005p+1=0.8u). Переходный процесс полностью совпадает с исходной схемой.

Единственным дополнением в схеме было согласование коэффициента обратной связи по скорости. Таким образом, правильно сформированное пространство скольжения действительно обеспечивает инвариантность.

Рассмотрим теперь влияние внешнего возмущения на характер движения выходной координаты. Будем задавать возмущение в интервале 0.1В до 2В и частоты от 10рад/с до 316рад/с. При амплитуде от 0,01 В и частоте от 10 до 150 рад/с возмущений частота скользящего режима больше 200Гц и отклонение от установившегося значения не более 0,01%. С ростом амплитуды возмущений до 2 В при той же условиях приводит к не значительному снижению частоты скольжения и отклонению в установившемся режиме 0,4%. Наихудший вариант получен при частоте возмущений 10 рад/с и амплитуде 2 В. Значительно снижается частота скольжения (1,42Гц) при относительно малом уменьшении отклонении от установившегося значения (5,6%). В реальных условиях возмущение не может быть больше возможности исполнительно устройства.

Рассмотренная модель привода реализующая рассмотренный принцип управления способна обеспечить заложенные в нее возможности параметрической компенсации различных видов возмущений. Разрабатываемый привод принципиально реализуем.

Приведенные исследования убедительно доказывают инвариантность к изменению внутренних параметров и внешних возмущений.

К недостаткам рассматриваемой модели привода относится небольшая частота переключений. В силу этого в выходном сигнале появляются периодические отклонения от установившегося значения 0,1% при оптимальном режиме работы и 1% при максимальной амплитуде и низкой частоте возмущений.

Рассматриваемая САУ представляет собой систему с переменной структурой. Характерным видом движения в таких САУ являются скользящие режимы. Важной особенностью является состояние при котором, попав на линию переключения, дальнейшее движение в системе будет осуществлять вдоль этой линии являющейся линией

переключения и никогда с нее не сойдет при любом возмущении. `Скользящий режим интересен тем, что закон движения в них определяется не параметрами объекта, а параметрами созданного подпространства скольжения.

Как показано в исследованиях в ряде случаев именно во время движения в скользящем режиме реализуется оптимальное управление, доставляющее минимум функционалу, который характеризует качество управляемого процесса.

САУ с таким видом движения реализуем на основе НК, выполняющего роль регулятора, формирующего управление на основе измерения положения и угловой скорости. При этом необходимо решить ряд задач являющихся необходимыми при построении НК, содержанием которых является:

- сбор и обработка информации, связанная с определение множеств входных и выходных значений параметров управления и создание базы обучающих данных (сбор данных для обучения); приведение четких значений в их нечеткие переменные;
- извлечение знаний, создание базы знаний и их представление в виде функций активации терм множеств, с последующим выбором метода нечеткого вывода;
- приведение нечетких величин к четким и использование построенного контроллера в реализации САУ.

Для систем автоматического управления нет практических рекомендаций к выбору функций активации, их числа, мощности множеств, диапазонов, что сильно осложняет процесс построения нечетких контроллеров.

С точки зрения обработки нечеткостей нельзя получить результаты, пока не будут исследованы в полной мере исходные данные, оценена общая структуру проблемы, исключены сомнительные данные или будут приняты решения на основе собственного опыта.

Структура системы управления с нечетким контроллером

Структурная схема, реализующая данный подход представлена на рис.5.26

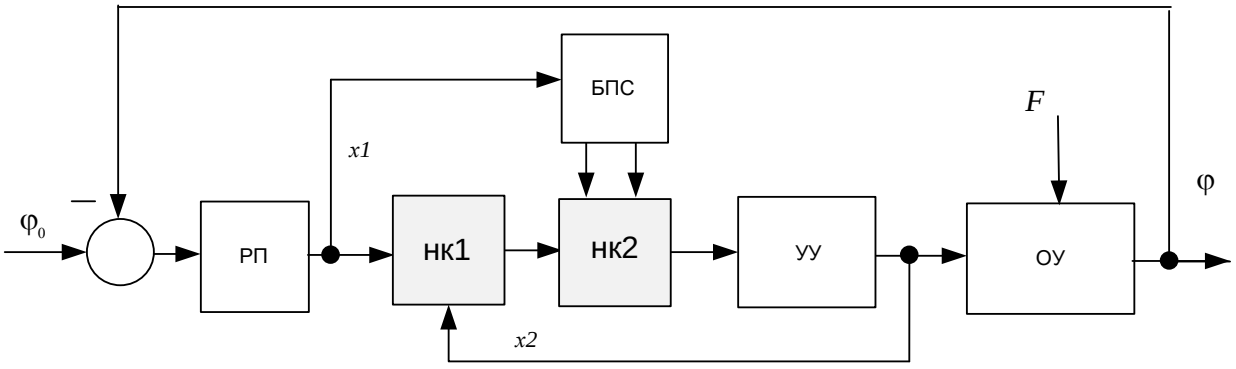


Рис.5.26. Структурная схема электропривода с нечетким регулятором переменной структуры: НК1- нечетки контроллер вычисления S, НК2 - нечетки контроллер переключения структур РП – регулятор положения, БПС – блок переключаемых структур, УУ – управляющее устройство, ОУ – объект управления,. F- возмущение, φ – угловое положение.

Структурная схема представляет собой следящий электропривод с нечетким СПС регулятором. Основными устройствами в схеме являются два нечетких контроллера включенных последовательно НК1 и НК2. НК1 выполняет функции вычисление кривой скольжения, а второй НК, в зависимости от знака произведения $e\dot{e}$ вычисленного в НК1 обеспечивает переключение структур реализованных в блоке переключаемых структур БПС. Входным сигналом на БПС является сигнал с выхода регулятора положения. Этот сигнал используется и в НК1. Устройство управления УУ может быть любым, обеспечивающим угловое перемещение φ исполнительного механизма. Выходной сигнал с УУ используется НК1. Объект управления при возникающих возмущениях Fможет перемещается неравномерно. В качестве входного воздействия φ_0 используются типовые входные воздействия.

Алгоритм работа схемы следующий. На универсальном множестве Р, характеризующем пространство состояний САУ, имеем два подмножества:

- подмножество S характеризуемое функцией принадлежности $\mu_S(x^1)$. Подмножество S множества Р есть множество пар $\{x^1|\mu_S(x^1)\}$, $\forall x^1 \in$ Р описывающих сигнал регулятора положения;
- подмножество V характеризуемое функцией принадлежности $\mu_V(x^2)$. Подмножество V множества Р есть множество пар $\{x^2|\mu_V(x^2)\}$, $\forall x^2 \in$ Р описывающих скорость вращения ротора двигателя.

В качестве функций принадлежности используем Z функцию, которая задается аналитическим выражением

$$f_Z(x; a, b) = \begin{cases} 1, & x < a \\ 0.5 + 0.5\cos\left(\frac{x-a}{b-a}\pi\right), & a \le x \le b \\ 0, & x > a \end{cases} \qquad (1)$$

и S функцию, для которой аналитическое выражение аналогично

$$f_S(x; a, b) = \begin{cases} 0, & x < a \\ 0.5 + 0.5\cos\left(\frac{x-a}{b-a}\pi\right), & a \le x \le b \\ 1, & x > a \end{cases}, \qquad (2)$$

где a, b - числовые параметры, принимающие действительные значения и упорядоченные отношением: a<b.

Разница в том, что рассматриваемы функции принадлежности, порождают нормальные выпуклые нечеткие множества с ядром (-∞, a) и носителем (-∞, b) для Z функции и с ядром (b, +∞) и носителем (a, +∞) для S функции.

При этом, чем меньше уверенность в точности измерения, тем большим будет интервал носителя соответствующего нечеткого множества. Именно по этой причине фаззификация позволяет более адекватно представить объективно присутствующую неточность результатов физических измерений. Это хорошо согласуется с принципом управления, когда любое отклонение от кривой скольжения вызывает соответствующую реакцию для переключения структуры. Входные терм множества определялись по результатам анализа фактических данных: диапазон ошибки регулятора положения, значение скорости, величина отклонения от траектории движения.

Для обеспечения движения необходимо добиться выполнения условия, при котором произведение $e\dot{e}<0$, где e является линией переключения (скольжения), а $\dot{e}$ ее производная.

Это решает НК1. Его выходное множество также имеет сплайн функции принадлежности, характеризующие величину отклонения от реализуемой траектории движения.

В соответствии с принципом управления уравнение скольжения запишется в следующем виде $e = C\mu_S(x^1)+\mu_V(x^2)$.

База знаний НК1 представлена двумя правилами, поскольку согласно принципа работы необходимо оценивать знак e. Оценка осуществлялся с использованием конструкции Мамдани. Изначально

находим α- уровни для каждого из правил с использованием операции MIN:

$\alpha_1 = S_1(x^1) \wedge V_1(x^2)$,

$\alpha_2 = S_2(x^1) \wedge V_2(x^2)$,

а затем соответствующие им функции принадлежности:

$\mu_1(e) = \alpha_1 \wedge K_1(e)$,

$\mu_2(e) = \alpha_2 \wedge K_2(e)$.

Вывод выполним с использованием операции MAX, производя объединение усеченных функций. В результате получаем переменную выхода с функцией принадлежности:

$\mu_\Sigma(e) = (\alpha_1 \wedge K_1(e)) \vee \alpha_2 \wedge K_2(e))$.

Построение модели начинается с создания контроллера с использованием FuzzyLogicToolbox. Используя команду Fuzzy вводимую в командной строке откроем окно FuzzyLogic (рис.5.27).

Во вкладке Edit устанавливаем число входных и выходных переменных. Двойной щелчок правой кнопкой «мыши» на пиктограмме Input1 открывает окно (рис.5.28а), в котором вводим необходимые параметры. Аналогичные действия для Input2 открывает окно для настройки терм множеств для второго входа (рис.5.28б). Открытие окон входа (рис.5.28в) и выхода (рис.5.28г) позволяют ввести фукции активации с учетом диапазонов терм множеств.выполняем и для остальных переменных.

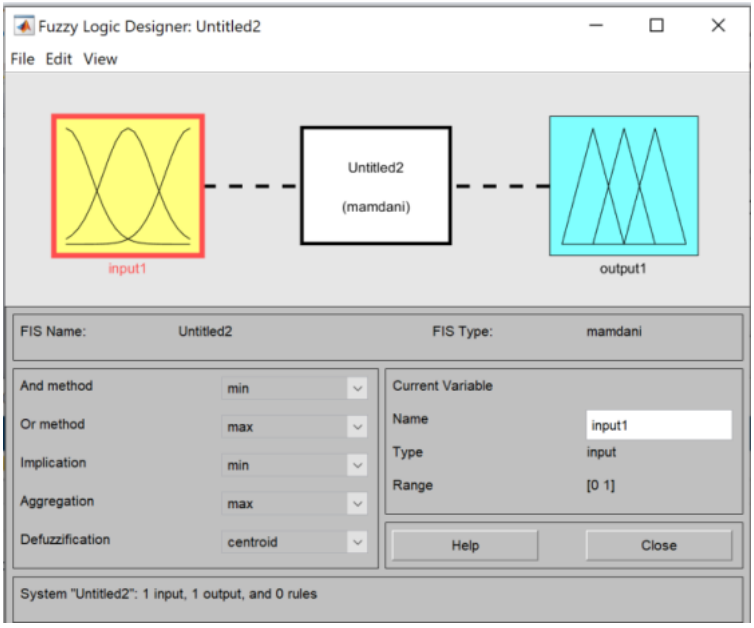


Рис.5.27. Окно FuzzyLogic

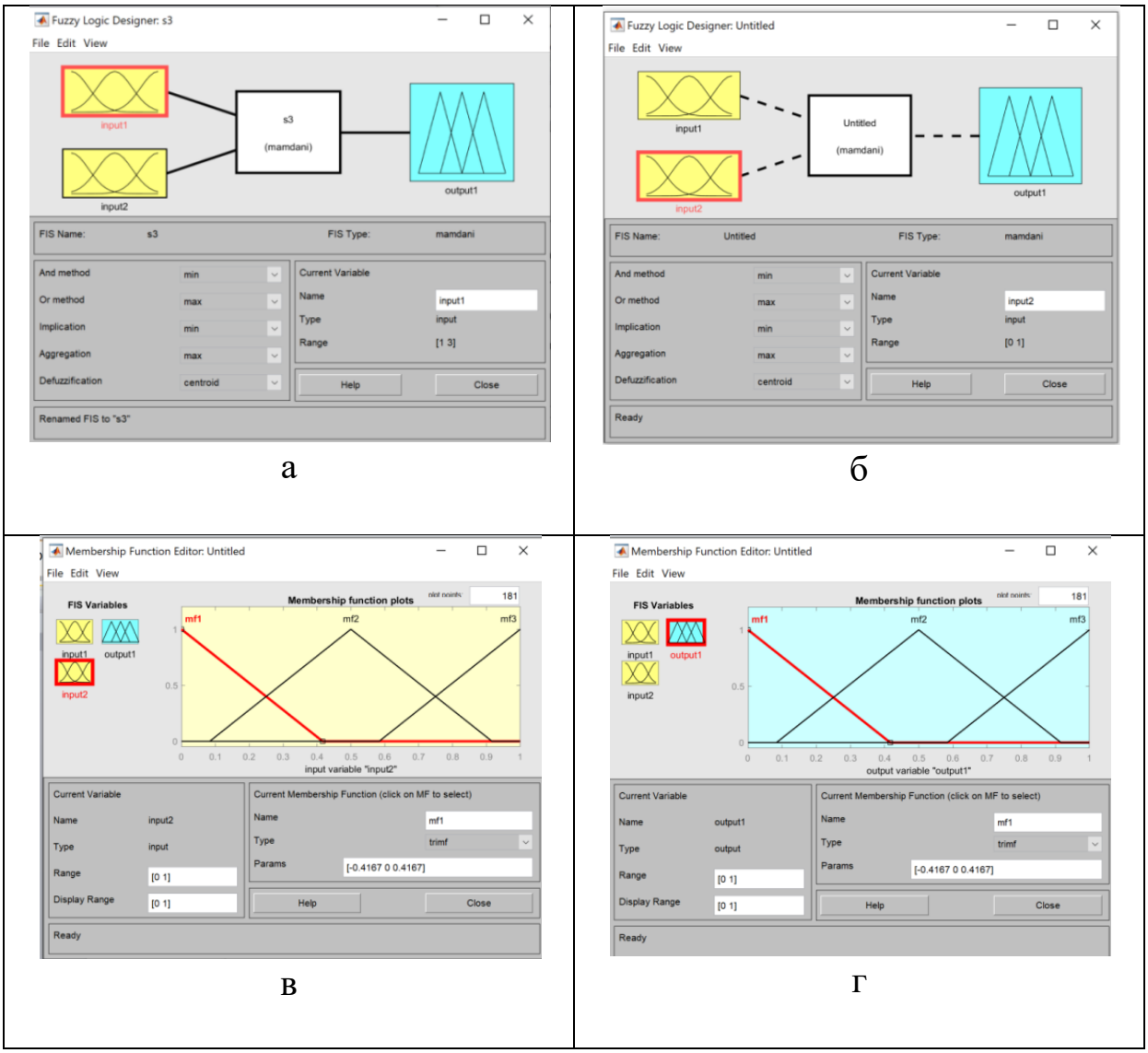


Рис.5.28. Окна настройки входных и выходных переменных:

Выполним настройку переменных $x1$ и $x2$ (см. рис. 5.27). Для них выбраны функции S и Z (рис. 5.29а - $x1$, рис. 5.29б – $x2$.) с диапазонами изменения термов, полученными в результате исследовния модели привода.

Настройка функции принадлежности выхода задается функциями S и Z (рис. 5.29в), а работа алгоритма вывода представлена на рис.5.29г. Рассмотренный контроллер выполняет вычисление произведения функции скольжения и ее производной.

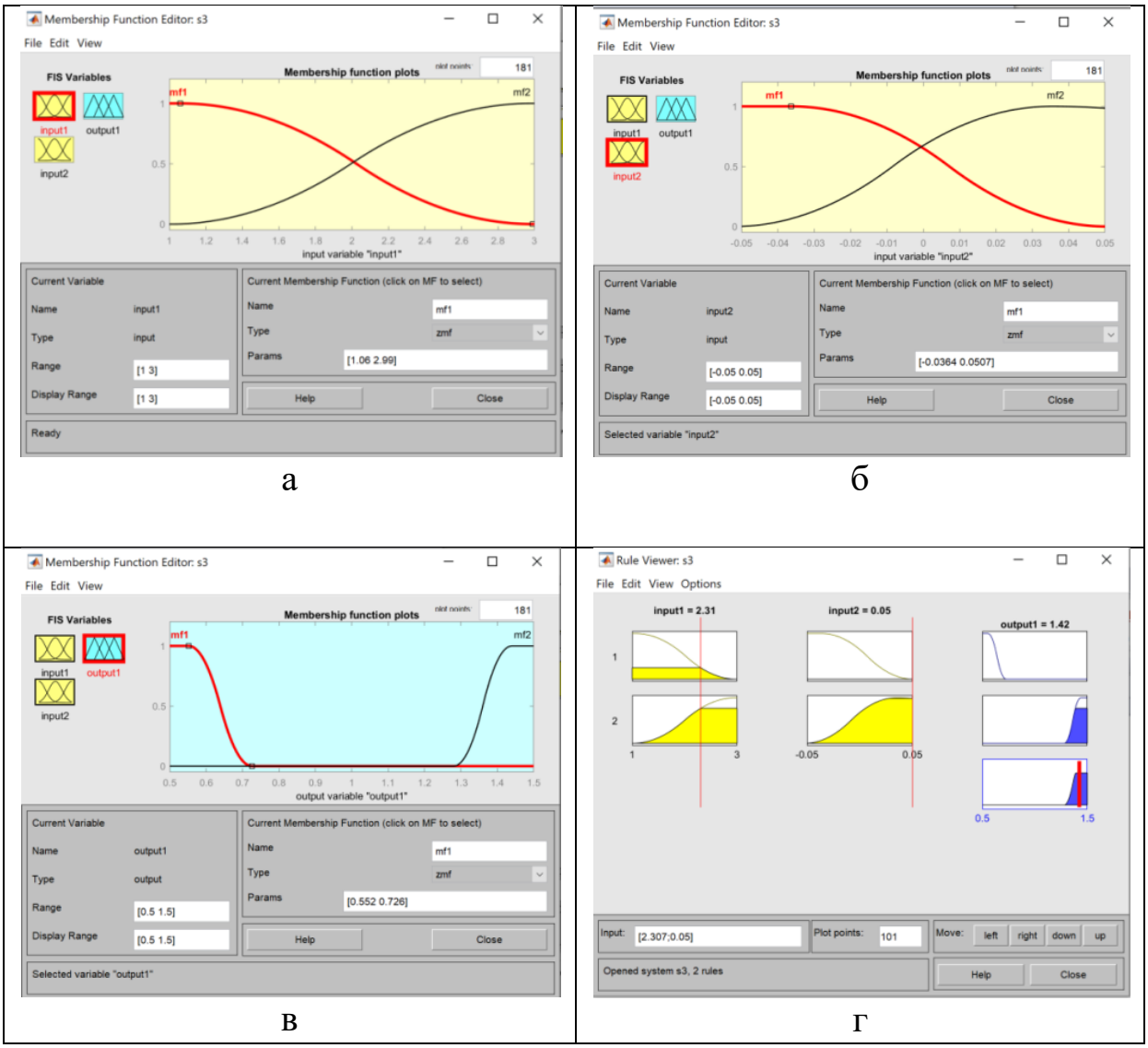


Рис. 5.29. Окна настройки НК1

Изменение структуры реализуется с помощью НК2 (рис.5.30) Входная функция представлена функциями вида (1) и (2). Имеет два терма

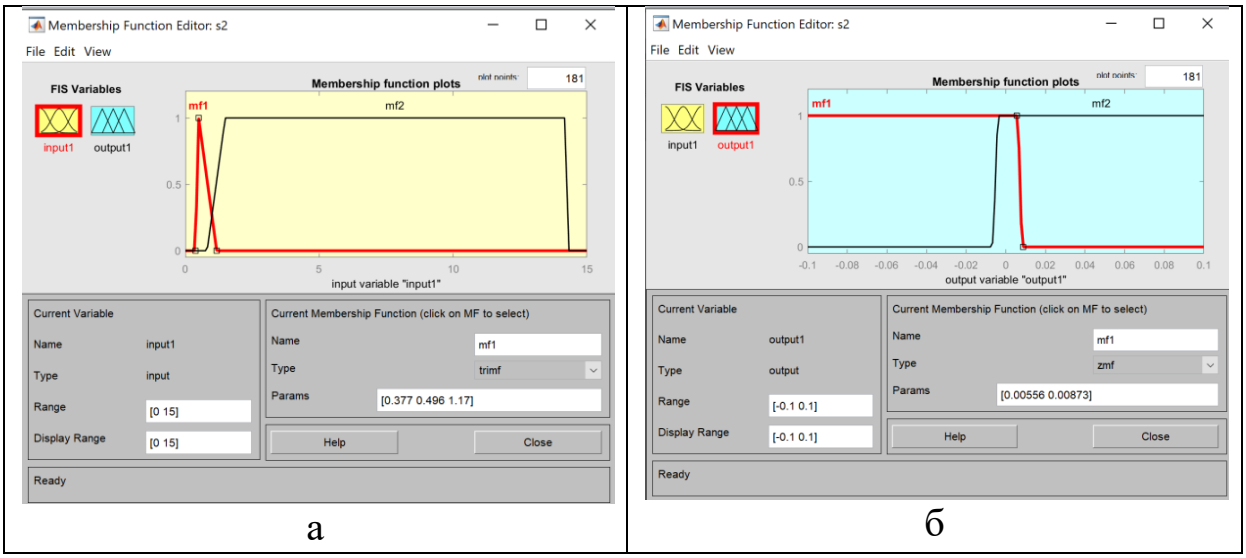


Рис.5.30. Окна настройки НК2

«Больше» и «Меньше» (рис.5.30а). Выходная переменная представлена трапецеидальными функциями (рис.5.30б).

Закон изменения структуры основан на анализе произведения $e\mu_S(x^1)$

$$\Psi = \begin{cases} b \text{ при } e\mu_S(x^1))>0 \\ -b \text{ при } e\mu_S(x^1)<0 \end{cases},$$

а управляющее воздействие: $u=\Psi\mu_S(x^1)$.

Вывод реализуется аналогичным образом, как и для НК1.

В литературе отмечено, что если функция управления формируется из регулируемой величины и ее производных, то движение в скользящем режиме не зависит от параметров объекта и внешних возмущений. В рамках одной линейной структуры для стабилизации необходимо в управление добавить воздействие по x^2- производной от координаты x^1. Таким образом, в результате изменения структуры удается парировать внешнее возмущение, которое недоступно для измерения, и сделать величину ошибки после окончания процесса управления равной нулю.

Исследование работоспособности рассматриваемой структуры автоматического управления было реализовано в пакетах Simulink и FuzzyLogicToolbox программы MatLab на примере электропривода постоянного тока, расположенного в суставе робототехнической системы. Полагалось, что ОУ подвержен влиянию внешнего не измеряемого возмущения, имеющего вид гармонической функции. Исследования проводились для оценки работоспособности и характеристик

нечеткой следящей системы, инвариантности к изменению собственных параметров и способности системы к компенсации не измеряемых внешних возмущений.

Построение модели начинается в Simulink так, как этот описано ранее. Созданные ранее нечеткие контроллеры сохраняются, а затем загружаются в рабочую область. Для этого загружается FuzzyLogic (рис.5.31) и в закладке File выбирается опция ExportToWorkspace.

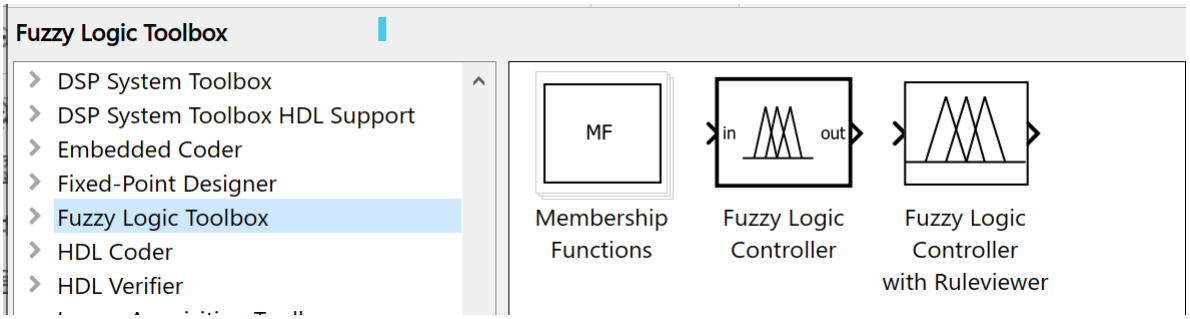


Рис.5.31. Выбор блока Fuzzy Logic

В результате в рабочей области появится файл с выбранным именем (рис.5.32). Если контроллеров несколько процедура повторяется в предложенном порядке. Затем в модель загружается один из вариантов контроллеров и соединяется с входами и выходами основной модели. Если число входов более одного на входе контроллера устанавливается мультиплексор и задается необходимое число входов. При наведении указателя на контроллер и двойном нажатии левой кнопки мыши открывается окно. В открывшемся окошке набираем имя fis - файла. Такая процедура выполняется для всех контроллеров присутствующих в модели. Теперь можно запускать модель и выполнять исследования (рис.5.33.).

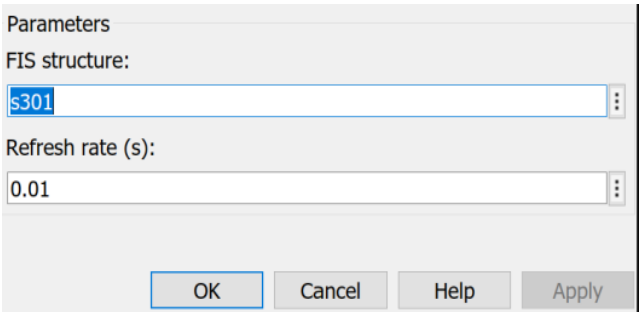


Рис.5.32. Окно сохоанения fis файла

Модель привода представлена на рисунке 36.

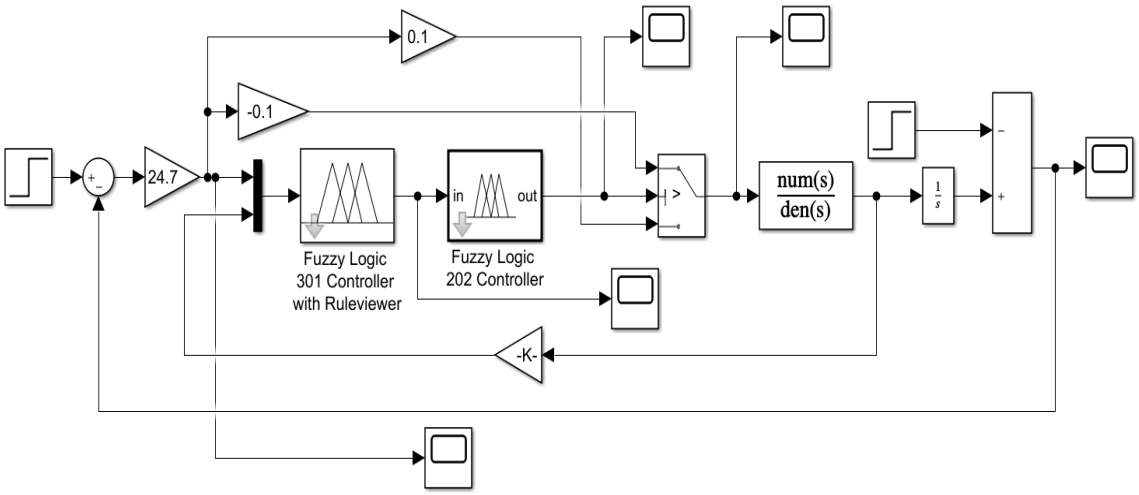


Рис.5.33. Модель интеллектуального привода

Результаты исследования представлены на рис. 5.34.

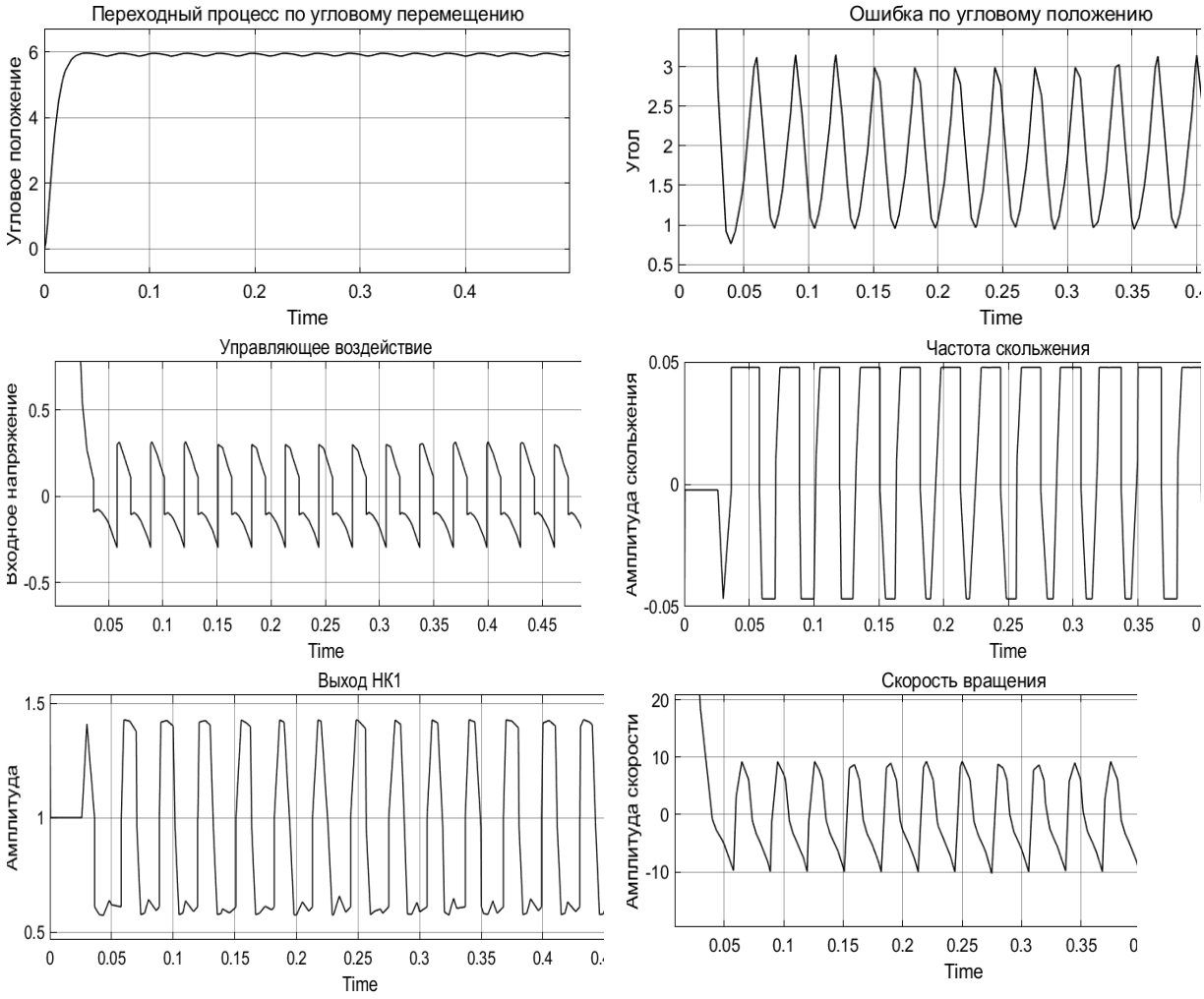


Рис.5.34. Пространство переменных состояния

Анализ влияния возмущений оценивался при изменении амплитуды и частоты возмущений. Амплитуда возмущений изменялась в интервале от 0.5 до 2 и фиксировался процесс движения следящей системы (рис.5.35.).

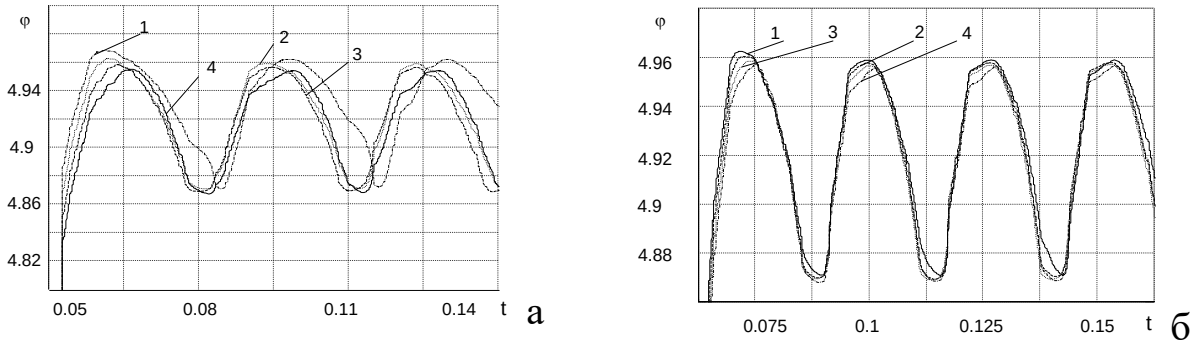


Рис.5.35. Влияние амплитуды и частоты возмущения на выходную координату: а) изменение амплитуды при постоянной частоте возмущений: F_4=0, $F_{3,2,1}$=(0.5…2)Sin(2t+1.57), б) изменение частоты возмущений при постоянной амплитуде: 1– F=2Sin(2t+1.57), 2 - F=2Sin(3t+1.57), 3 – F=2Sin(4t+1.57), 4. - F=2Sin(5t+1.57)\

Рассматривался установившийся режим движения при скачкообразном входном воздействии. Изменение амплитуды устанавливалось до величины 50% от входного задания, что для реальных систем вряд ли целесообразно, так как приводные системы не проектируются по моменту нагрузки до величин вдвое превосходящие допустимые.

Тем не менее, рассматриваемая САУ компенсирует не измеряемое возмущение с незначительным отклонением и в первую очередь за счет принципа действия систем с переменной структурой (рис.5.36).

Изменение коэффициента усиления регулятора положения в диапазоне от 14,7 до 30,2 приведено на рис. (рис.5.36а). С ростом коэффициента усиления уменьшается амплитуда отклонения от воспроизводимой кривой и возрастает частота переключения структур. При этом увеличивается и рассогласование между заданным входным сигналом и сигналом по цепи обратной связи примерно на 2%. Для компенсации этого явления необходима настройка параметров цепи обратной связи. Следует заметить, что изменение коэффициента усиления в 2 раза на практике маловероятно.

Аналогичное изменение параметров управляющего устройства приводит с уменьшением коэффициента усиления к увеличению вре-

мени переходного процесса и в незначительном изменении частоты переключения структур (рис.5.36б).

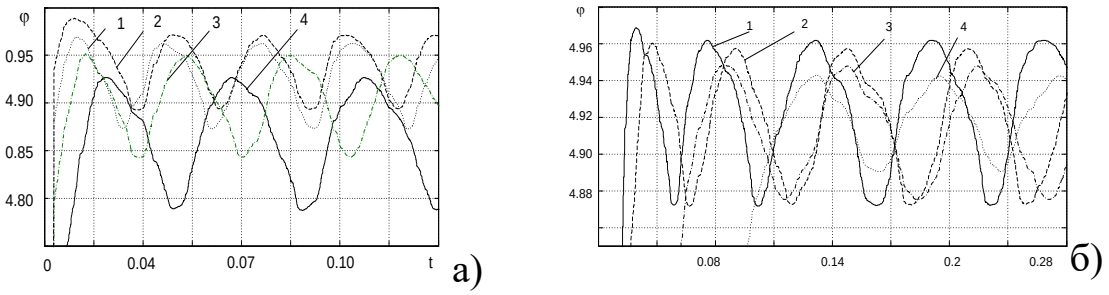


Рис.5.36. Влияние коэффициента усиления регулятора положения а) и управляющего устройства б): а) 1 – $К_{рп}$ = 20,0; 2 – $К_{рп}$ = 14,7; 3 - $К_{рп}$ = 25; 4- $К_{рп}$= 30,2; б) 1 - $К_{оу}$ = 40,2; 2 - $К_{оу}$=20,2; 3 - $К_{оу}$ = 25,2; 4 - $К_{оу}$ = 35.

Изменение параметров САУ не оказывает существенного влияния на выходные характеристики.

Можно предложить процедуру построения НК в следующем виде.

1. Определить цель создания САУ и ее назначение. Построить для наглядности рассуждений структурную схему, отражающую закладываемый принцип построения. По возможности исключить промежуточные переменные, влиянием которых можно пренебречь и которые не используют сигналы обратных связей. В первом приближении управляющее устройство желательно представлять уравнением не высокого порядка. Для реальных систем необходимо учитывать конечные величины, связанные с физической реализацией. Отразить эти ограничения в модели. Переменными могут быть любые существующие в реализуемой структуре. Детальным образом исследовать те сигналы, которые будут использованы в окончательном проекте. Необходимо найти логические связи, между переменными используемые как входные и определить взаимные связи между входными и выходными переменными. Для функциональных зависимостей оценить выходные данные, которые будут использоваться как промежуточные переменные. Определить интервалы множеств.

2. Выбрать или сформировать функцию активации таким образом, что бы число термов по возможности было минимальным. Точечные или интервальные функции дают более грубые результаты, по сравнению с функциями более сложного вида. Можно использовать

точечные функции, но при этом их количество должно быть достаточным для достижения приемлемой точности. Использование точечных или интервальных функций принадлежности допускает применение традиционных арифметических операций.

3. Создание базы знаний должно быть направленно на формирование правил таким образом, что бы соответствовать той логической модели, которую удалось сформировать при анализе причинно следственных отношений между входными и выходными переменными.

Число правил по возможности должно охватывать все функции принадлежности, используемые как во входных, так и выходных переменных. Необходимо исключить те правила, которые порождают противоречия. Реализация вывода может сопровождаться известными методами Мамдани, Сугено, Ларсена и др. отличающимися по точности, что не исключает создание иного метода, использующего нотации нечеткой логики.

Исследование модели показали принципиальную возможность построения приводов с переменной структурой в классе систем использующих нечеткую логику. Рассмотренная система работоспособна, характеристики нечеткой системы показывают инвариантность к изменению собственных параметров и способность к компенсации не измеряемых внешних возмущений. Структура привода с НК реализуется значительно проще. База знаний имеет всего лишь четыре записи, обеспечивающие желаемые выходные характеристики.

5.6. Автоматизированная система создания нечеткого контроллера для диагностики ЭМС

Для реализации диагностики ЭМС с помощью нечеткой логики воспользуемся системой MatLAB и находящейся в ее составе наглядным и эффективным средством составления программных моделей – пакетом визуального программирования SimuLink. Пакет SimuLink позволяет осуществлять исследование (моделирование во времени) поведения динамических нелинейных систем, причем введение характеристик исследуемых систем осуществлять в диалоговом режиме, путем графической сборки схемы соединений элементарных (стандартных или пользовательских) звеньев. В результате такого составления получается модель исследуемой системы.

Построение модели мехатронной системы.

Двигатель постоянного тока с независимым возбуждением (без учета реакции якоря и влияния действия вихревых токов) при одномассовой расчетной схеме описывается следующей системой дифференциальных уравнений:

$$\left.\begin{array}{c} u_я = i_я R_я + L_я \dfrac{di_я}{dt} + e \\ e = C_e \cdot \Phi \cdot \omega \\ M = C_M \cdot \Phi \cdot i_я \\ M = J \cdot \dfrac{d\omega}{dt} + M_c \end{array}\right\} (1)$$

где:

u - *напряжение на якорной обмотке двигателя,*

e - *электродвижущая сила (ЭДС) якоря,*

$i_я$- *ток якоря,*

$R_я$ - *сопротивление якорной цепи, Ом;*

$L_я$ - *индуктивность якорной цепи, Гн;*

C_m - *конструктивный коэффициент двигателя;*

C_e - *коэффициент ЭДС двигателя;*

ω - *скорость вращения вала двигателя,*

Ф - *полезный поток одного полюса, Вб;*

J - *момент инерции суммарный, приведенный к валу двигателя, кг·м2;*

M - *электромагнитный момент двигателя, Н·м.*

M_c - *момент нагрузки, приведенный к валу двигателя, Н·м.*

Модель двигателя постоянного тока с независимым возбуждением, полученная на основании уравнений (1) показана на рис.5.37.

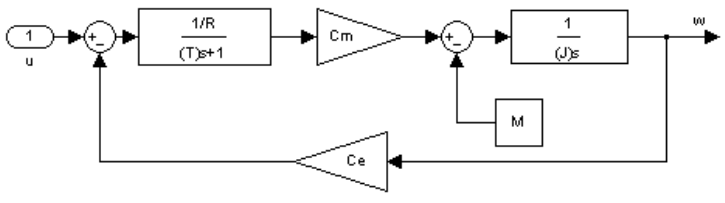


Рис.5.37. Структурная схема ДПТ НВ

Для построения модели в MATLAB, воспользуемся программой Simulink. Для запуска программы необходимо предварительно запу-

стить пакет MATLAB. После открытия основного окна программы MATLAB нужно запустить программу Simulink. Для построения модели в основном окне MATLAB в меню File выберем команду – File – New – Model и на экране появится пустое окно будущей модели.

Начнем с построения в среде Simulink схемы электропривода.

Запустим библиотеку Simulink из панели инструментов открывшегося окна модели, щелкнув левой кнопкой мыши по значку LibraryBrowser. (рис.5.38а).

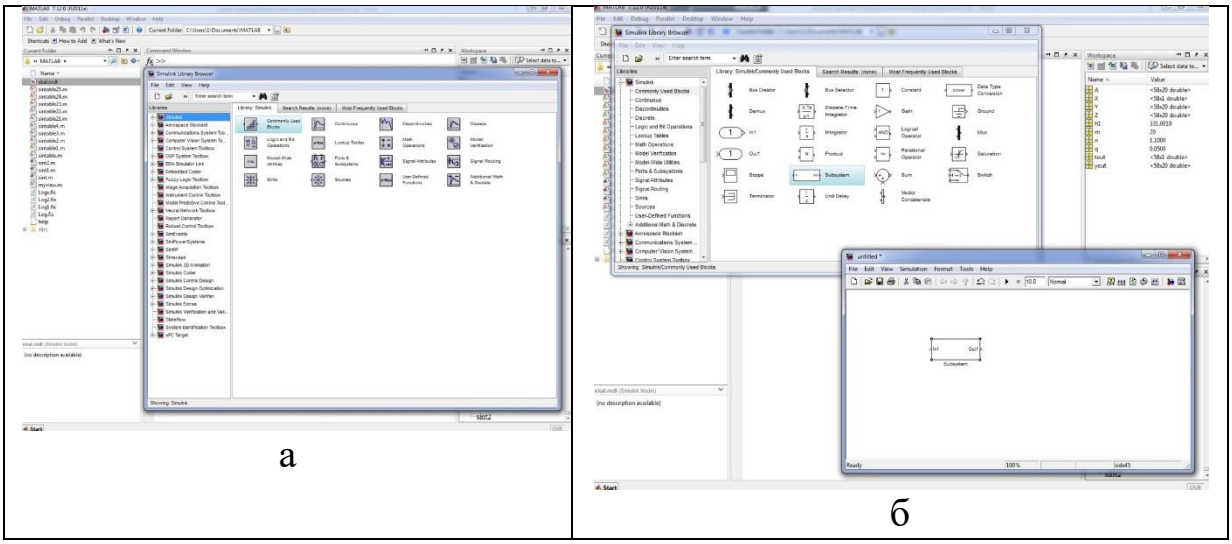
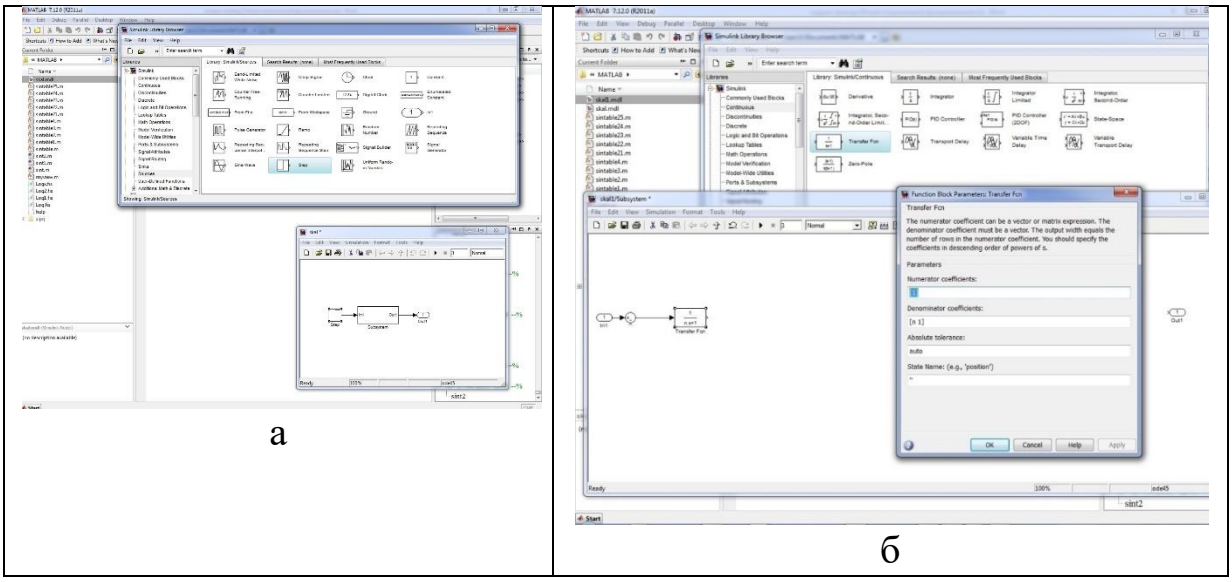


Рис.5.38. Окна построения привода: а – окно LibraryBrowser, б – окно CommonlyUsedBlocks.

Во вкладке CommonlyUsedBlocks, открывшейся библиотеки, выберем блок Subsystem и наведя на него указатель мыши, зажать её левую кнопку, после этого перетащим блок в окно модели (рис.5.38б).

Для вывода в рабочую область MATLAB данных работы модели добавим в окно модели из этой же вкладки блок Out и подключим его к выходу блока Subsystem. Для соединения блоков необходимо указать курсором на “выход” блока, а затем, нажать и, не отпуская левую клавишу “мыши”, провести линию к входу другого блока. После чего отпустить клавишу.

Для исследования переходных процессов модели добавим на вход блока Subsystem сигнал Step из вкладки Sources. Соединяем блоки аналогично предыдущему (рис.5.39а).



а б

Рис.5.39. Окна построения привода: а – окно Subsystem,
б – окно CommonlyUsedBlocks

Для построения модели привода откроем блок Subsystem щелкнув по нему 2 раза левой кнопкой мыши. В открывшемся окне удалим линию связи между входом и выходом, выделив ее щелчком левой кнопкой мыши по ней и затем нажав кнопку Delete на клавиатуре. Сначала добавим блок Sum вкладки Commonly used blocks . Для его настройки откроем его щелкнув по нему левой кнопкой мыши два раза. В открывшемся окне настройки в поле Listofsigns комбинацией символв **|, -,** + добиваемся необходимой конфигурации. Кроме того это блок можно оформить в виде прямоугольника.

Далее из вкладки Continuous добавляем блок Transfer Fnc который в модели показывает передаточную функцию регулятора скорости. Откроем его для настройки, как и в предыдущем блоке. В открывшемся окне в поле denominator coefficients на место постоянной времени впишем переменную n, чтобы иметь возможность менять ее программно (рис.5.39б).

Аналогично добавляем регулятор тока и силовой преобразователь, предварительно поставив на входе каждого блока сумматор для организации обратной связи (рис.5.40).

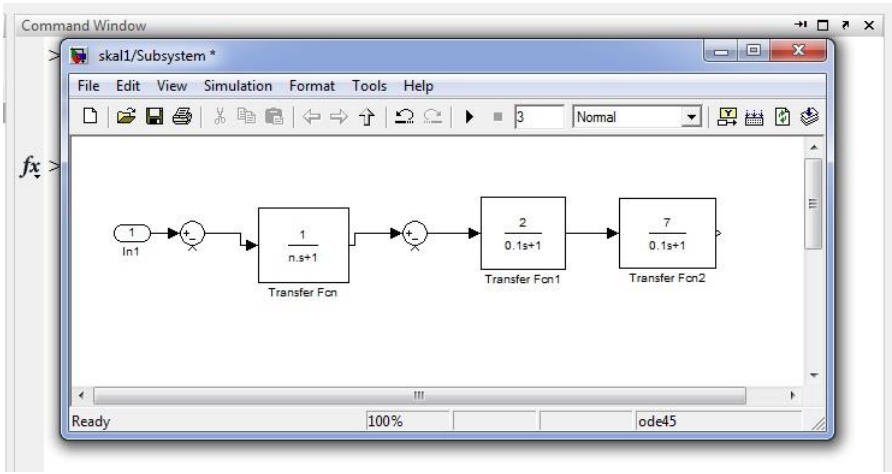


Рис. 5.40.Фрагмен структуры привода

Здесь и далее все коэффициенты, различные постоянные и другие данные необходимо брать из характеристик исследуемого привода.

Далее, добавляем уже знакомы элементы, постоянные и усилители (которые берем из вкладки Commonly used blocks), характеризующие коэффициенты обратных связей создаем модель электропривода (рис.5.41)

Так же, для контроля сигнала из различных точек модели добавляем осциллограф Scope, он строит графики исследуемых сигналов в функции времени. Позволяет наблюдать за изменениями сигналов в процессе моделирования.

Для правильного вывода в рабочую область MATLAB данных работы модели необходимо настроить параметры вывода, для того чтобы подключить процесс, определенный в программе MatLab, как входной в S-модель, предусмотрен механизм включения портов входа и выхода.

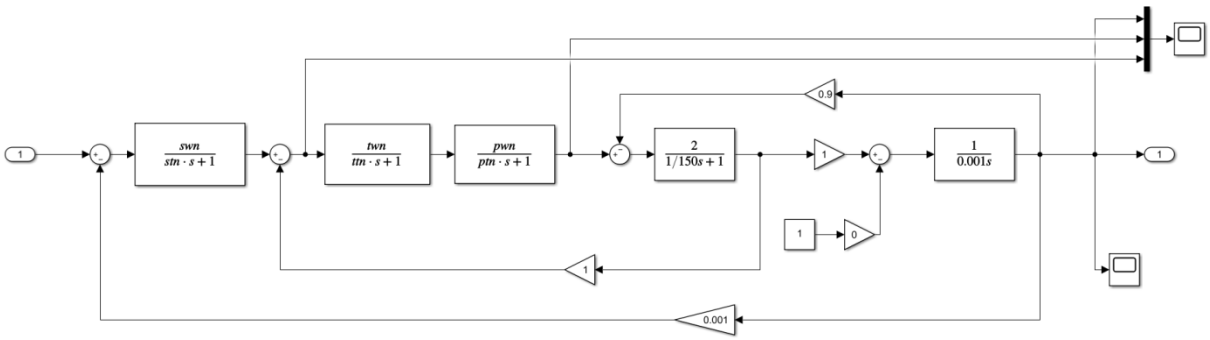


Рис.5.41. Модель привода для построения автоматизированной системы диагности

Для этого нужно сделать следующее:

- в блок-схему S-модели вставить блок порта входа In и подключить к S-модели;

- в меню Simulation окна S-модели вызвать вкладку WorkspaceI/O окна SimulationParameters (команды Parameters);

- установить "галочку" в правом окошке рядом с надписью Input, записав по левую сторону от этой надписи вектор с двумя именами – первое – имя вектора значений аргумента, второе - имя вектора значений входного сигнала при этих значениях аргумента;

- установить значение этих векторов в среде MatLab, - обратиться к выполнению моделирования S-модели.

Наоборот, чтобы вывести некоторые сигналы, которые формируются в S-модели, в рабочее пространство MatLab, нужно: в блок-схему S-модели вставить блоки портов выхода Out и подсоединить к ним необходимые выходные величины других блоков;

- в меню Simulation окна S-модели вызвать вкладку Workspace I/O окна SimulationParameters (команды Parameters); - установить "галочку" в правом окошке с надписью Time;

- установить "галочку" в правом окошке с надписью Out put. В этом случае значения времени будут записываться в рабочее пространство как вектор с именем tout, а соответствующие значения выходных процессов при этих значениях времени – в столбце матрицы yout (в первые столбцы – процессы, которые поданы на первый выходной порт Out1, далее,

- процессы, которые поданы на второй порт Out2 и т.п.).

Конечно, если изменить имена, которые записаны по правую сторону от надписей соответствующих окошек, то эти же данные будут записаны под этими именами. Так же, активизируя окошко Initialstate (начальные значения переменных состояния), можно ввести в S-модель начальные значения переменных состояния системы. Активизировав окошко States (Переменные состояния), можно записать текущие значения переменных состояния системы в рабочее пространство под именем xout (или другим именем, если его записать по правую сторону от этой надписи). Наконец, можно записать и конечные значения переменных состояния в вектор xFinal, если активизировать окошко Finalstate(рис.5.42).

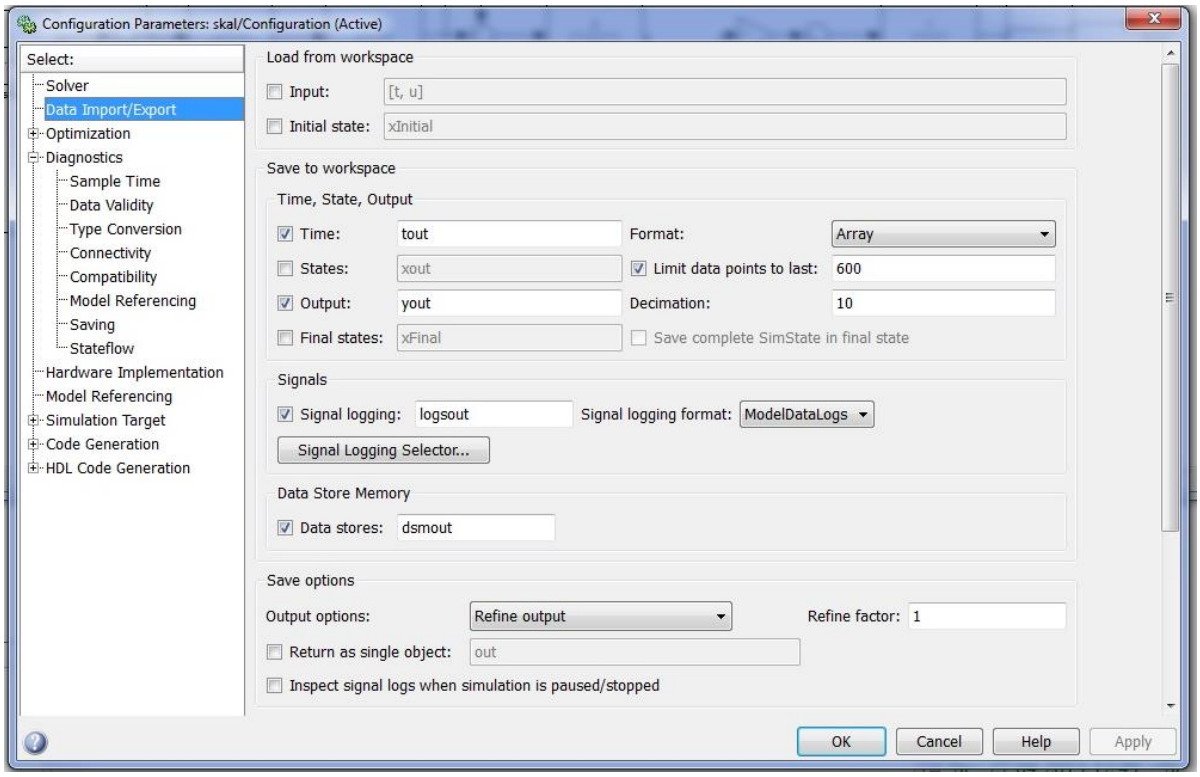


Рис.5.42. Окно настройки вычислительного процесса

В этом же окне определяем количество выводимых значений через Limit data points to last (кол-во точек) равное 600 и Decimation (делитель) равный 10 – получаем 60 выводимых значений.

Для возможности изменять значение переменной n, напишем скрипт - файл:

```
global tout yout Z;
n=0.1;
q=0.05;
m=1.1;
sim('skal');
A=(yout) ;
X=(tout)/3;
n=n+q;
while n<=m
sim('skal');
A=[A,yout];
n=n+q;
end
```

где n – начальное значение, q - шаг изменения, m - конечное значение.

Здесь сначала переменные объявляются глобальными, чтобы использовать их в файл - функциях. Далее организуется цикл с заданным шагом, с каждым новым значением вызывается построенная мо-

дель, происходит расчет с этим значением и данные добавляются в матрицу выходных значений.

Далее в скрипте прописываем вывод графика получаемых значений, интерполируем значения по времени и запуск функции, с помощью которой создается файл fis-редактора.

```
Y=A/max(max(A));
% нормируем получившуюся матрицу выходных значений (интервал 0
- 1).

Z=interp1(X,Y,X,'spline');
% интерполируем нормированную матрицу.

subplot(2,3,1:2);
plot(X,Y)
grid, set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16),
title('Переходнаяхарактеристика');
%colorbar;
%colormap(hsv);
xlabel('Время');
ylabel('Амплитуда')
subplot(2,3,4:5);
plot(Y,X)
grid, set(gca, 'FontName', 'Arial Cyr', 'FontSize', 16),
xlabel('Амплитуда');
ylabel('Время')
subplot(2,3,[3;6]);
axis('off');
h1=text(-0.2,1,'Графики ','FontSize',12);
h1=text(0,0.9,'зависимости переходной','FontSize',10);
h1=text(0,0.8,'характеристики от времени', 'FontSize',10);
h1=text(0.2,0.6,sprintf('при n= g',n), 'FontSize',10);
yout=Y;
sintable('Log.fis');
% в скобках задается имя создаваемого файла
help sintable
```

Далее предствален листинг функции sintable:

```
functionsintable(filename)
% функция создает fis-файл по выходным
% параметрам из рабочей области;
globaltoutyoutZ;
X=(tout)/3;
lt=size(X);
ly=size(yout);
```

```
s=ly(:,2)*ly(:,1);
l=1:ly(:,2);
t=lt(:,1);
%нормирование переходной характеристики;
Y=yout(:,l)/max(max(yout));
[F,mes]=fopen(filename,'w');
% Задание параметров файла;
fprintf(F,'[System]\n');
fprintf(F,'Name=''Log''\n');
fprintf(F,'Type=''sugeno''\n');
fprintf(F,'Version=2.0\n');
fprintf(F,'NumInputs=2\n');
fprintf(F,'NumOutputs=%1.0f',max(l));
fprintf(F,'\n');
fprintf(F,'NumRules=%2.0f',s);
fprintf(F,'\n');
fprintf(F,'AndMethod=''prod''\n');
fprintf(F,'OrMethod=''probor''\n');
fprintf(F,'ImpMethod=''prod''\n');
fprintf(F,'AggMethod=''sum''\n');
fprintf(F,'DefuzzMethod=''wtaver''\n');
fprintf(F,' \n'); %Задание функций принадлежности
```

Здесь задается функция принадлежности по времени, причем временные интервалы очень узкие, чтобы при различных весах функции принадлежности по амплитуде, разброс был несущественным.

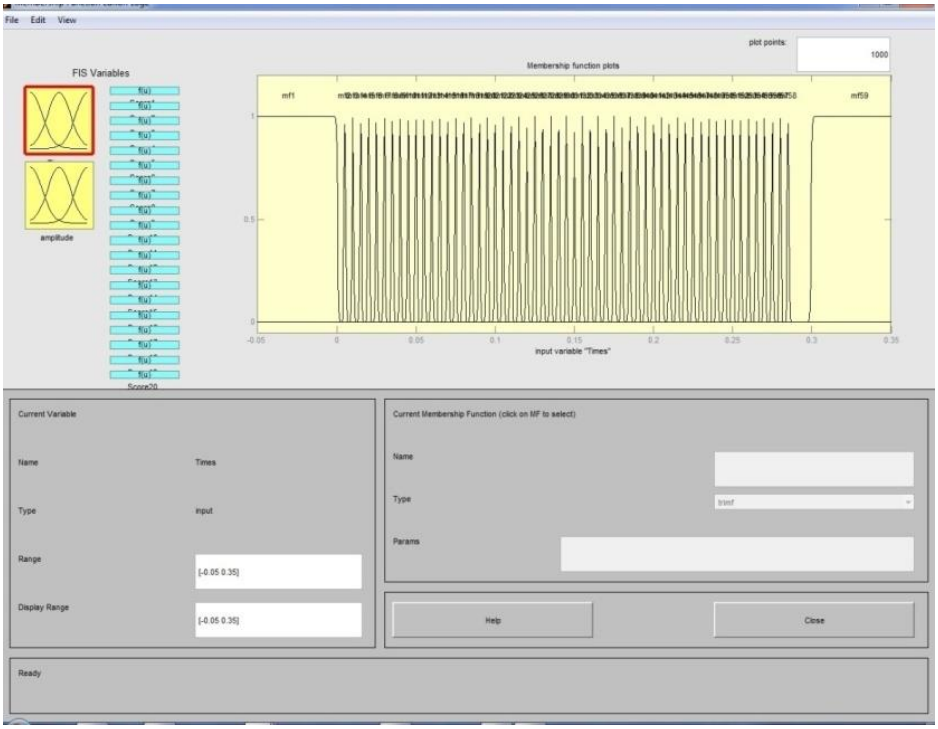


Рис.5.43.

```
fprintf(F,'[Input1]\n');
fprintf(F,'Name=''Times''\n');
fprintf(F,'Range=[-0.05 0.35]\n');
fprintf(F,'NumMFs=%1.0f',(t+1));
fprintf(F,'\n');
fprintf(F,'MF1=''mf1'':''sigmf'',[-3310 0.0005]\n');
n=2;
m=0.005;
while n<=t
fprintf(F,'MF%1.0f',n);
fprintf(F,'=''mf%1.0f''',n);
fprintf(F,':''gaussmf'',[');
fprintf(F,'0.0005');
fprintf(F, ' %1.4f',m');
fprintf(F,']\n');
n=n+1;
m=m+0.005;
end
fprintf(F,'MF%1.0f',(t+1));
fprintf(F,'=''mf%1.0f''',n);
fprintf(F,':''sigmf'',[3310 0.2995]\n');
fprintf(F,' \n');
```

Далее задаются функции принадлежности по амплитуде:

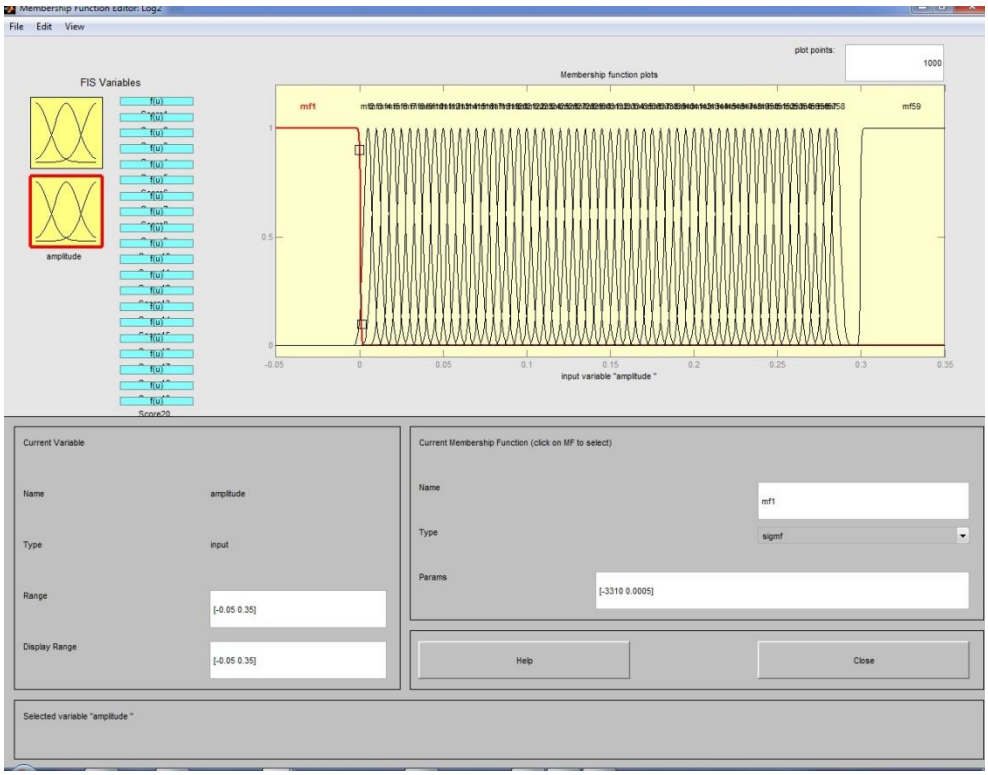


Рис.5.44.

```
fprintf(F,'[Input2]\n');
fprintf(F,'Name=''amplitude ''\n');
fprintf(F,'Range=[-0.05 0.35]\n');
fprintf(F,'NumMFs=%1.0f',(t+1)');
fprintf(F,'\n');
fprintf(F,'MF1=''mf1'':''sigmf'',[-3310 0.0005]\n');
n=2;
m=0.005;
while n<=t
fprintf(F,'MF%1.0f',n);
fprintf(F,'=''mf%1.0f''',n);
fprintf(F,':''gaussmf'',[');
fprintf(F,'0.0025');
fprintf(F, ' %1.4f',m');
fprintf(F,']\n');
  n=n+1;
  m=m+0.005;
end
fprintf(F,'MF%1.0f',(t+1));
fprintf(F,'=''mf%1.0f''',n);
fprintf(F,':''sigmf'',[3310 0.2995]\n');
fprintf(F,' \n');
% Задание выходных параметров;
```

Выходные параметры берутся из интерполированной матрицы и определяются их веса в соответствии с функцией принадлежности по времени.

```
k=1;
while k<=ly(:,2)
fprintf(F,'[Output%1.0f',k);
fprintf(F,' ]\n');
fprintf(F,'Name=''Score%1.0f',k);
fprintf(F,' ''\n');
fprintf(F,'Range=[0 1]\n');
fprintf(F,'NumMFs=');
fprintf(F, ' %1.0f',max(size(Y(:,k)))');
fprintf(F,'\n');
n=1;
while n<=max(size(Y(:,k)))
fprintf(F,'MF%1.0f',n);
fprintf(F,'=''mf%1.0f''',n);
fprintf(F,':''constant'',[');
fprintf(F, ' %1.4f',Y(n,k)');
fprintf(F,']\n');
  n=n+1;
end
```

```
fprintf(F,' \n');
 k=k+1;
end
fprintf(F,' \n');
% Задание правил;
fprintf(F,'[Rules]\n');
k=1;
i=1/t;
while k<=ly(:,2)
  M= yout(1,:);
  n=1;
while n<=ly(:,1)
    a=Z(n,k);
M(1,k)=n;
  b=round((a*t)/1);
  e=rem(a*t,1);
if e<0.5
        e=1-e;
end
if e>0.5
        b=b+1;
end
if b>t
        b=t;
end
if n==1
       b=1;
end
fprintf(F,'%1.0f',n);
fprintf(F,' %1.0f',b);
fprintf(F,',');
fprintf(F,' %1.0f',M);
fprintf(F,' (%1.2f',e);
fprintf(F,') : 1\n');
  n=n+1;
end
 k=k+1;
end
fprintf(F,' \n');
fclose(F);
end
```

Запуск модели на выполнение приводит к появлению в главном окне MATLAB данных (Рис.5.45), .

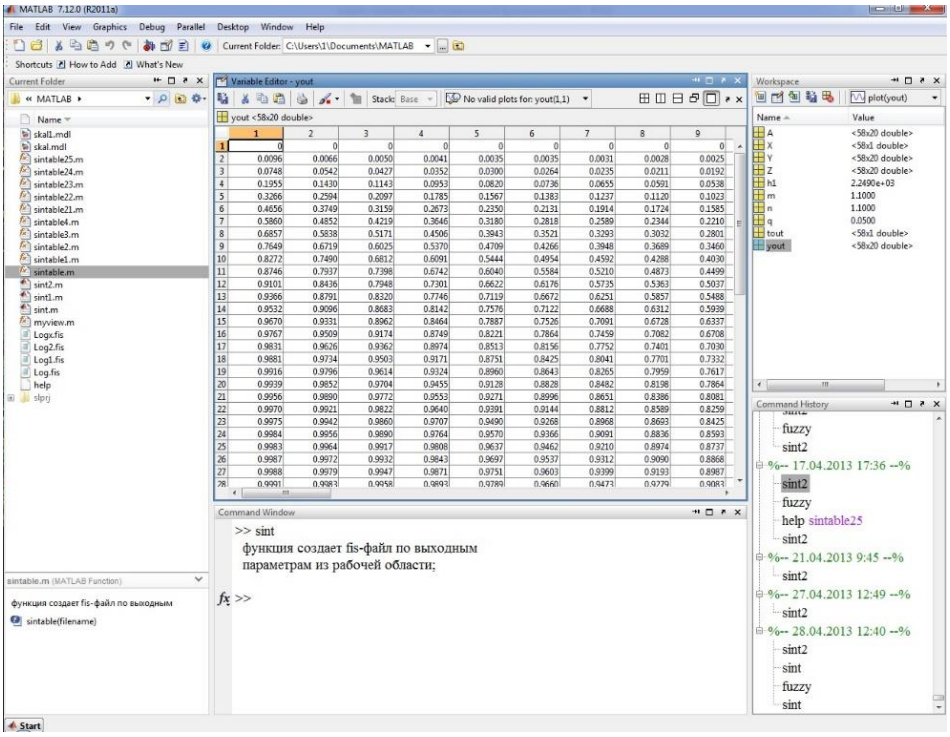


Рис.5.45. Окно данных

И строит график выходных данных Рис.5.46.

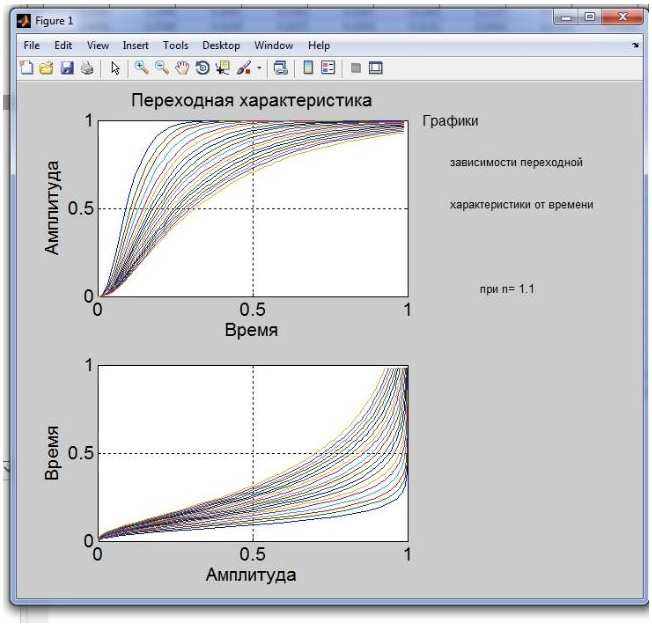


Рис.5.46. График выходных данных

Чтобы проверить правильность написания fis-файла запустим fis-редактор командой fuzzy в главном окне MATLAB (Рис.5.47).

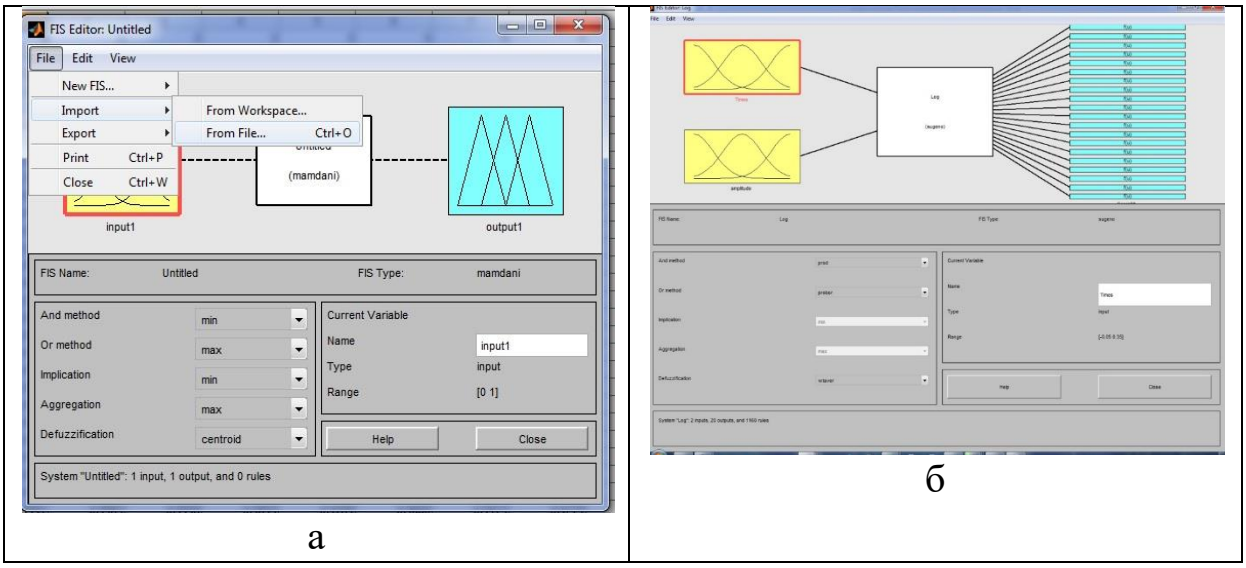


Рис.5.47. Окна проверки результатов: а – запуск редоктора, б - структура контроллера

Далее командой File-Import-FromFile откроем созданный файл и проверяем созданные правила функции принадлежности и другие окна. Далее для применения созданной базы правил в диагностике ЭМС создадим модель с нечетким контроллером. Модель создается аналогично предыдущей с добавлением в нее нечеткого контроллера (рис.5.48).

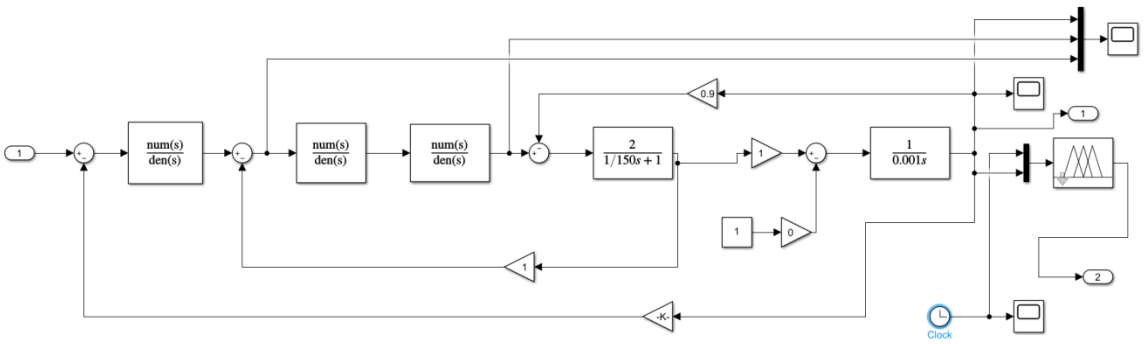
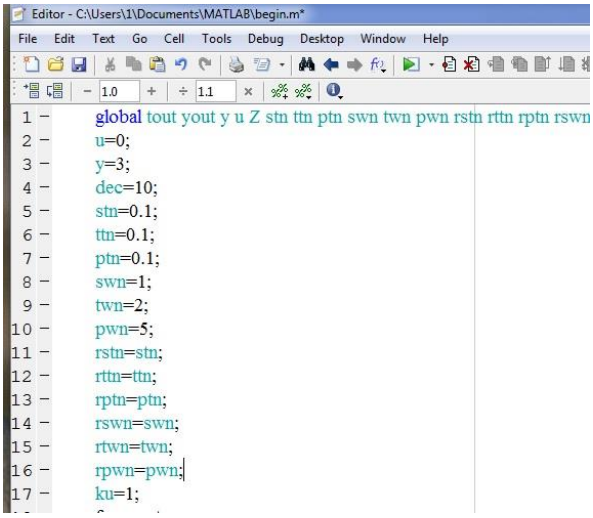


Рис.5.48. Структура модели диагностики с нечетким контроллером

Здесь, в блоках регулятора скорости, тока и силового преобразователя, для возможности программной замены выразим постоянные времени и коэффициенты усиления через переменные. Кроме того, так как у правил две входные переменные, добавим через блок Mux блок времени, для второй переменной Time. Для правильной работы

контроллера модернизируем функцию создания правил, убрав из нее нормирование к единице.

Для начального задания параметров системы пишем скрипт “Begin” , с которого начинается работа программы.




```
global tout yout y u Z stn ttn ptn swn twn pwn rstn rttn rptn
rswn rtwn rpwn  Log X Y m r n1 dec ku;
 u=0;
 y=3;
 dec=10;
 stn=0.1;
 ttn=0.1;
 ptn=0.1;
 swn=1;
 twn=2;
 pwn=5;
 rstn=stn;
 rttn=ttn;
 rptn=ptn;
 rswn=swn;
 rtwn=twn;
 rpwn=pwn;
 ku=1;
fuzzcont
```

С помощью графического интерфейса MATLAB GUI (GraphicUserInterface) создадим графическую оболочку, связывающую созданные ранее скрипты и функции в единую программу для удобного пользования программой (Рис.5.49).

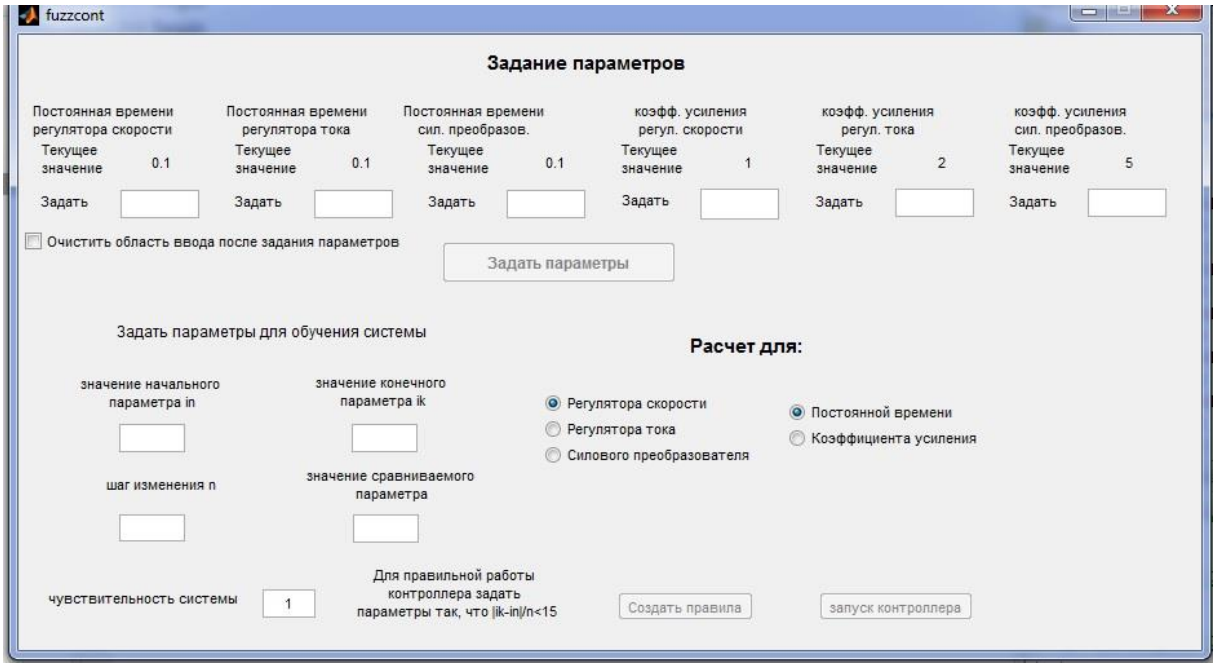


Рис.5.49. Окно задания параметров для построения нечеткого Контроллера

Здесь для трех указанных блоков, в верхней части окна, можно задавать параметры, отличные от установленных. Ниже задаются параметры блока. Выбираем с помощью переключателей, начальные, конечные, шаг для создания правил и значение сравниваемого параметра, для определения исправности этого блока с помощью контроллера. По заданному параметру система строит выходную характеристику и с помощью правил производит диагностику. При снятии выходной характеристики с реального привода, нужно просто подать эту характеристику на вход обученного контроллера. Правила будут созданы по реальным паспортным данным. На выходе контроллера можно получить данные, показывающие адекватность работы блока.

После задания параметров становится активна кнопка «Создать правила». Нажимаем ее – появляется график с кривыми выходных характеристик, по которым создаются правила(рис.5.51).

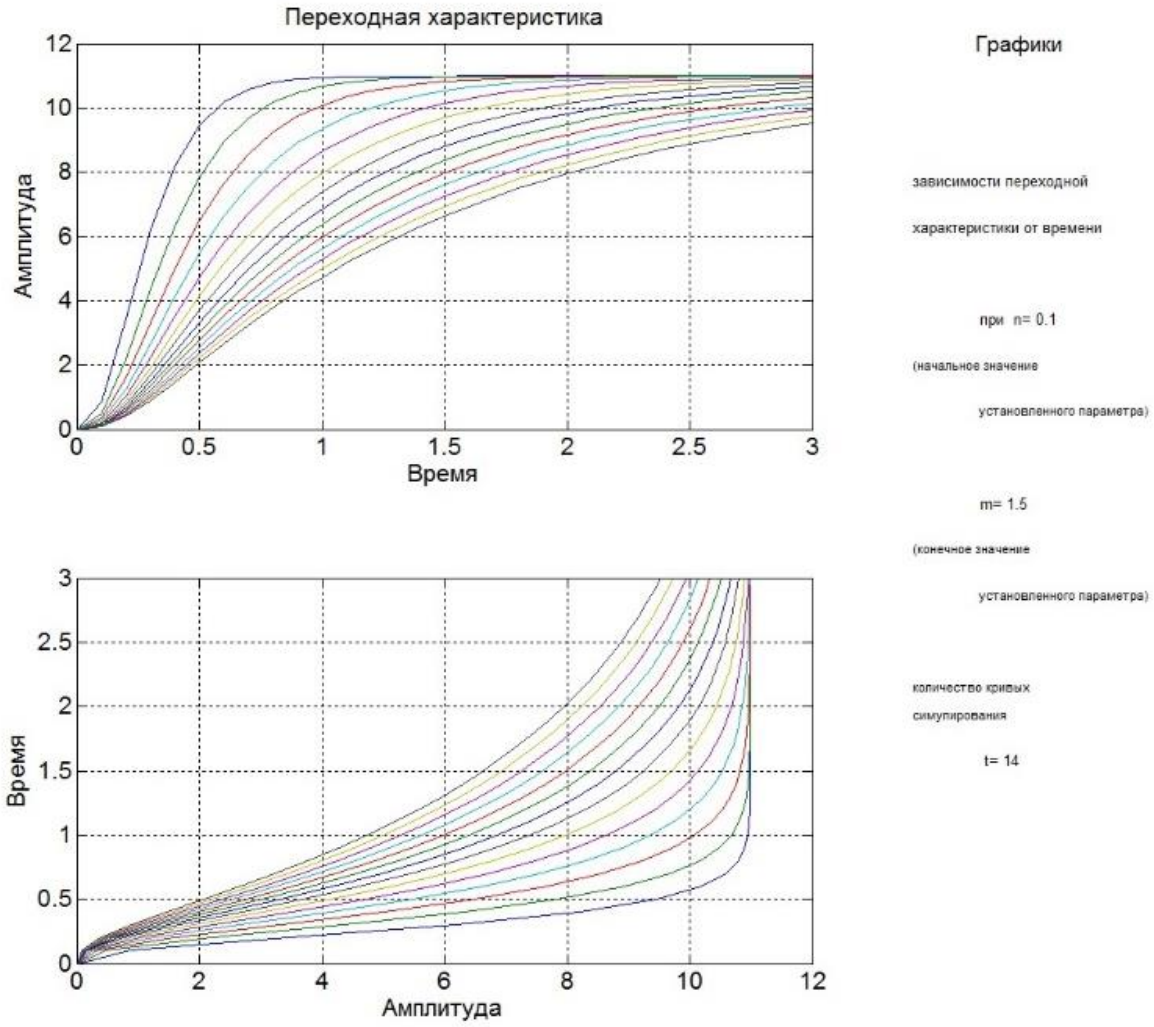


Рис.5.50. Синтезированные кривые

Для создания правил и просмотре их в фаззи – редакторе можно задавать любые параметры, но для корректной работы контроллера необходимо задавать начальные, конечные значения и шаг такими, чтобы количество кривых для обучения не превышало 15. После создания правил, становится активной кнопка, запуска контроллера «запуск контроллера» с нажатием которой появляется окно, в котором наблюдаем работу контроллера (рис.5.51).

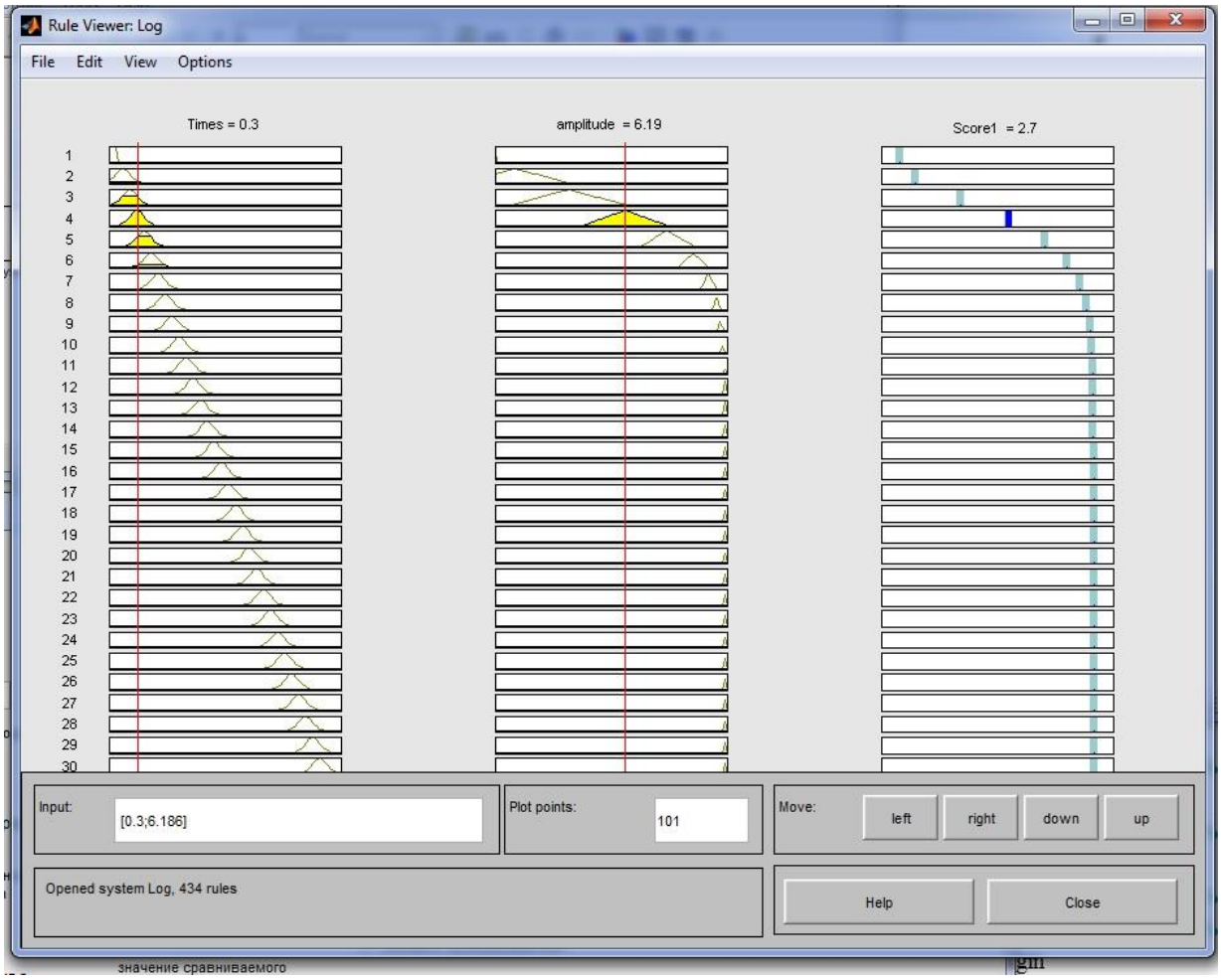


Рис.5.51. Результат работы правил

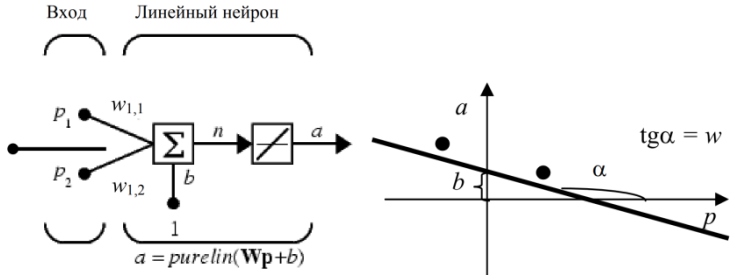
Чтобы посмотреть оценку контроллером заданного параметра, откроем в модели с контроллером окно графика, где будет наглядно изображена характеристика оценки.

ПРАКТИЧЕСКИЕ ЗАНЯТИЯ

Практическое занятие № 1. Создание модели линейной сети

Задание 1

Линейную сеть с одним нейроном, показанную на рис. 1, можно создать следующим образом:



а б

Рис.1. Линейная сеть а – структура сети, б – линейное разделение

```
clear
net = newlin([-1 1; -1 1],1);
```

Первый входной аргумент задает диапазон изменения элементов вектора входа; второй аргумент указывает, что сеть имеет единственный выход. Начальные веса и смещение по умолчанию равны нулю. Присвоим весам и смещению следующие значения:

```
net.IW{1,1} = [2 3];
net.b{1} =[-4].
```

Теперь можно промоделировать линейную сеть для следующего предъявленного вектора входа:

```
p = [5;6];
a = sim(net,p)
      a = 24
```

Видно, что сеть правильно классифицировала входной вектор.

Порядок выполнения работы

1. Для заданного преподавателем варианта задания (таблица) разработать структурную схему линейной нейронной сети.
2. Разработать алгоритм создания и моделирования линейной нейронной сети.
3. Реализовать разработанный алгоритм в системе MATLAB.

4. Определить параметры созданной нейронной сети (веса и смещение) и проверить правильность работы сети для последовательности входных векторов (не менее 5).
5. Построить график, аналогичный представленному на рис. 2, для своих исходных данных.
6. Определить имя функции инициализации значений матриц весов и смещений, принятой по умолчанию для линейных нейронных сетей. Установить новые значения матриц весов и смещения с помощью функции инициализации rands.
7. Распечатать текст программы.
8. Составить отчет, который должен содержать :
– цель лабораторной работы;
– структурную схему нейронной сети;
– алгоритм, текст программы и график;
– выводы.

Варианты заданий

Номер варианта	Количество входов	Диапазоны значений входов	Количество нейронов в слое
1	2	–3…+3	2
2	2	–1…+1	3
3	2	–4…+4	2
4	2	–2…+2	3
5	2	–8…+8	2
6	2	–9…+9	3
7	2	–7…+7	2
8	2	–5…+5	3
9	2	–3…+3	2
10	2	–6…+6	3

Задание 2

Обучение линейной сети. Процедура настройки посредством прямого расчета

Цель работы: изучение процедуры настройки параметров линейных нейронных сетей посредством прямого расчета в системе MATLAB.

Порядок выполнения работы

1. Для заданного преподавателем варианта задания (таблица) построить линейную сеть с помощью функции newlind, промоделировать ее работу и определить значения веса и смещения.
2. Построить график для полученных значений веса и смещения, аналогичный рис. 1б.
3. Построить график линий уровня поверхности функции ошибки в системе MATLAB.
4. Сделать ручной расчет значений функции ошибки не менее чем для пяти точек из заданного диапазона.
5. Сравнить результаты ручных расчетов и расчетов, выполненных в системе MATLAB.
6. Распечатать текст программы.
7. Составить отчет, который должен содержать :
– цель лабораторной работы;
– структурную схему нейронной сети;
– алгоритм, текст программы и графики;
– ручной расчет значений функции ошибки и результаты расчета в системе MATLAB;
– выводы.

Варианты заданий

Номер варианта	Количество входов – 1; количество нейронов – 1				
	Диапазон значений входа	Значения входа персептрона		Целевой выход	
		1-е задание	2-е задание	1-е задание	2-е задание
1	–4…+4	{–2 1}	{–2 1 0 2}	{–1 –1}	{–1 –1 1 0}
2	–2…+2	{0 1}	{0 1 –1 –1}	{0 1}	{0 1 1 0}
3	–4…+4	{–2 1}	{–2 1 0 3}	{2 2}	{2 2 0 –2}
4	–3…+3	{–1 –2}	{–1 –2 1 2}	{1 –1}	{1 –1 –2 0}
5	–2…+2	{0 –1}	{0 –1 –1 1}	{0 1}	{0 1 0 1}
6	–4…+4	{–2 1}	{–2 1 3 2}	{1 –2}	{1 –2 –1 2}
7	–3…+3	{–2 0}	{–2 0 2 –2}	{1 1}	{1 1 –1 –1}
8	–2…+2	{–1 0}	{–1 0 1 1}	{–1 0}	{–1 0 –1 1}
9	–4…+4	{0 2}	{0 2 1 –2}	{0 –2}	{0 –2 –2 –1}
10	–4…+4	{–3 2}	{–3 2 2 3}	{1 –1}	{1 –1 2 –1}

Задание 3

Задача классификации векторов

Цель работы: решение задачи классификации с использованием модели линейной нейронной сети в системе MATLAB.

Порядок выполнения работы

1. Для заданного преподавателем варианта задания (таблица) построить линейную нейронную сеть в системе MATLAB и с ее помощью решить задачу классификации линейно разделимых векторов с точностью 0,01 и максимальным числом эпох 200.

2. Выполнить моделирование созданной линейной сети с векторами входа из обучающего множества и вычислить ошибки сети.

3. Построить персептронную нейронную сеть в системе MATLAB для того же обучающего множества и с ее помощью решить задачу классификации линейно разделимых векторов.

4. Выполнить моделирование созданной персептронной сети с векторами входа из обучающего множества и вычислить ошибки сети.

5. Сравнить результаты моделирования линейной и персептронной линейными сетями.

6. Добавить в обучающее множество такой вектор, чтобы образовались линейно неразделимые векторы. Построить линейную и персептронную нейронные сети для решения задачи классификации нового обучающего множества.

7. Выполнить моделирование созданных сетей с векторами входа из обучающего множества и проверить правильность работы сетей.

8. Распечатать текст программы.

9. Составить отчет, который должен содержать :

– цель лабораторной работы;

– структурную схему нейронной сети;

– ручной расчет настройки сети;

– текст программы и результаты моделирования;

– выводы.

Варианты заданий

Номер варианта	Количество входов – 2; количество нейронов – 1		
	Диапазоны значений входов	Значения входов	Целевые выходы
1	–2…+2	{[–1; –1] [0; 0] [1; –1] [1; 1]}	{1 1 1 0}
2	–4…+4	{[0; 0] [2; –2] [1; –2] [–2; –1]}	{0 1 0 0}
3	–2…+2	{[0; 0] [–1; 1] [–1; 0] [1; 1]}	{0 1 1 0}
4	–4…+4	{[–2; 1] [1; –2] [3; –1] [2; 2]}	{0 0 0 1}
5	–3…+3	{[–2; 1] [0; 1] [2; –1] [–2; –1]}	{0 0 1 0}
6	–2…+2	{[0; 0] [1; 1] [–1; 1] [–1; 0]}	{0 0 1 1}
7	–4…+4	{[–2; 2] [1; 2] [0; 0] [3; –2]}	{0 1 0 1}
8	–3…+3	{[–1; 1] [–2; –1] [1; –2] [2; 0]}	{1 0 0 1}
9	–4…+4	{[–3; 1] [2; –1] [2; 2] [3; –1]}	{1 1 0 0}
10	–3…+3	{[–2; –1] [1; –1] [0; 1] [2; 0]}	{1 0 1 0}

Практическое занятие № 2. Аппроксимация функции одной Переменной

Цель работы

Научиться работать с сетью прямой передачи сигнала, функция **newff**. Разобраться с алгоритмом обратного распространения ошибки.

Краткие теоретические сведения

В лабораторной работе рассматривается нейронная сеть с прямой передачей сигнала (с прямой связью) [2], то есть сеть, в которой сигналы передаются только в направлении от входного слоя к выходному, и элементы одного слоя связаны со всеми элементами следующего слоя. Важнейшим для реализации нейронных сетей является определение алгоритма обучения сети.

В настоящее время одним из самых эффективных и обоснованных методов облучения нейронных сетей является *алгоритм обратного распространения ошибки*, который применим к *однонаправленным многослойным сетям*. В многослойных нейронных сетях имеется множество скрытых нейронов, входы и выходы которых не являются входами и выходами нейронной сети, а соединяют нейроны внутри сети, то есть *скрытые нейроны*. Занумеруем выходы нейронной сети индексом $j = 1,2,\ldots,n$, а обучающие примеры индексом $M = 1,2,\ldots,M_0$.

Тогда в качестве целевой функции можно выбрать функцию ошибки как сумму квадратов расстояний между реальными выходными состояниями y_{jM} нейронной сети, выдаваемых сетью на входных данных примеров, и правильными значениями функции d_{jM}, соответствующими этим примерам. Пусть $\mathbf{x}=\{x_i\}$ – столбец входных значений, где i=1,2,..,n. Тогда $\mathbf{y}=\{y_j\}$ – выходные значения, где j=1,2,…,m. В общем случае $n \neq m$. Рассмотрим разность $y_{jM}-d_{jM}$, где d_{ji} – точное (правильное) значение из примера. Эта разность должна быть минимальна. Введем расстояния согласно евклидовой метрике, определив норму

$$\|\mathbf{y}-\mathbf{d}\|=\sqrt{(\mathbf{y}-\mathbf{d},\mathbf{y}-\mathbf{d})^2}\,. \qquad (1)$$

Пусть целевая функция имеет вид

$$E=\frac{1}{2}\sum_{j,M}(y_{j,M}-d_{j,M})^2\,. \qquad (2)$$

Коэффициент ½ выбран из соображений более короткой записи последующих формул. Задача обучения нейронной сети состоит в том, чтобы найти такие коэффициенты $w_{\beta k}$, при которых достигается минимум $E(\mathbf{w})\,(E\geq 0)$.

На рис.1 показана архитектура нейронной сети с прямой передачей сигнала.

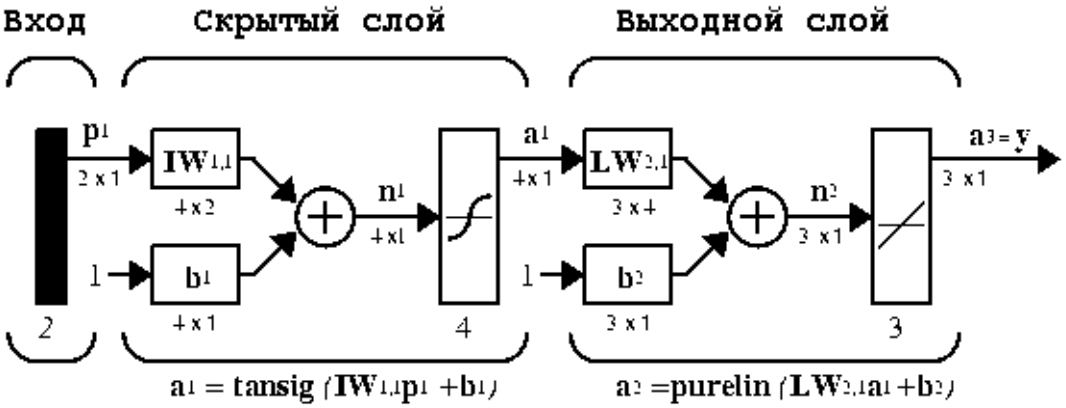


Рис.1. Схема архитектуры нейронной сети с прямой передачей сигнала

Здесь приняты обозначения, используемые в [1], а именно, p^1 - вектор входа, $IW^{i,j}$, $LW^{i,j}$ - матрицы весов входа и выхода, b^i - смещение, a^i - выход слоя, y - выход сети, *tansig* (гиперболическая тан-

генциальная), *purelin* (линейная) - соответствующие функции активации.

Веса и смещения определяются с помощью алгоритма обратного распространения ошибок.

Обучение сети обратного распространения требует выполнения следующих операций:

1. Выбрать очередную обучающую пару из обучающего множества; подать входной вектор на вход сети.
2. Вычислить выход сети.
3. Вычислить разность между выходом сети и требуемым выходом (целевым вектором обучающей пары).
4. Скорректировать веса сети так, чтобы минимизировать ошибку.
5. Повторять шаги с 1 по 4 для каждого вектора обучающего множества до тех пор, пока ошибка на всем множестве не достигнет приемлемого уровня.

Пример решения типовой задачи

Выполнение лабораторной работы состоит из следующих этапов: прежде всего, необходимо оцифровать график функции **y=f(x)**, то есть получить ряд соответствующих значений по горизонтальной и вертикальной осям.

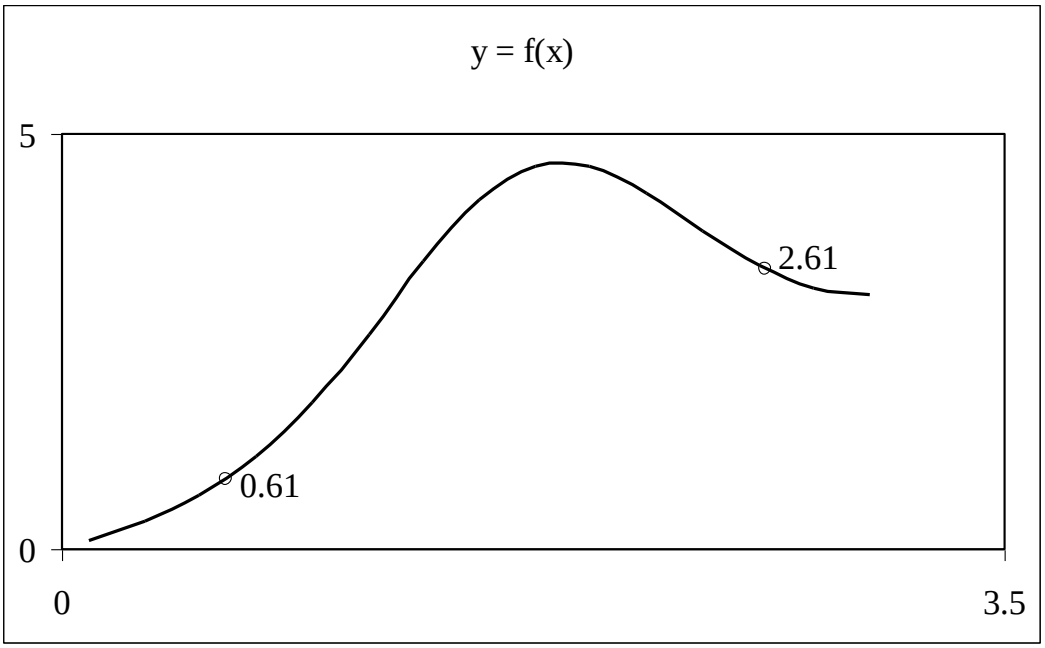


Рис.2. Пример зависимости для функции одной переменной

В примере, показанном на рис.2 путем табуляции с равномерным шагом, были получены два массива, каждый из которых состоит из 15 значений.

По горизонтальной оси – [**0.10 0.31 0.51 0.72 0.93 1.14 1.34 1.55 1.76 1.96 2.17 2.38 2.59 2.79 3.00].**

По вертикальной оси – [**0.1010 0.3365 0.6551 1.1159 1.7632 2.5847 3.4686 4.2115 4.6152 4.6095 4.2887 3.8349 3.4160 3.1388 3.0603]**.

Ниже приводится программа создания, обучения нейронной сети и вывода результатов.

```
x=[0.10  0.31  0.51  0.72  0.93  1.14 ...
  1.34  1.55  1.76  1.96  2.17  2.38 ...
  2.59  2.79  3.00];
y=[0.1010 0.3365 0.6551 1.1159 1.7632 2.5847 ...
  3.4686 4.2115 4.6152 4.6095 4.2887 3.8349 ...
  3.4160 3.1388 3.0603];
net=newff([0 3],[5,1],{'tansig','purelin'},'trainbfg');
net.trainParam.epochs=300;
net.trainParam.show=50;
net.trainParam.goal=1.37e-2;
[net,tr]=train(net,x,y);
an=sim(net,x);
plot(x,y,'+r',x,an,'-g'); hold on;
xx=[0.61  2.61];
v=sim(net,xx)
plot(xx,v,'ob','MarkerSize',5,'LineWidth',2)
```

В результате выполнения программы получаются следующие результаты, отражённые на рис.3 и 4:

В массиве **v** содержатся приближённые значения для двух контрольных точек, указанных на графике (рис.2) **xx=[0.61 2.61]**. При данных параметрах сети получены значения: **v=[1.05 3.35]**. Сравнив эти приближённые значения с точными значениями **[0.85 3.37]**, можно сделать вывод о корректности построения нейронной сети.

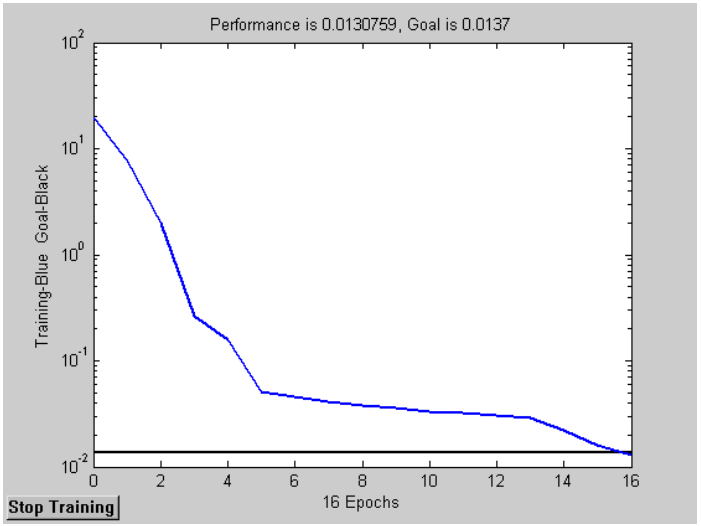


Рис.3. Характеристика точности обучения в зависимости от числа эпох обучения

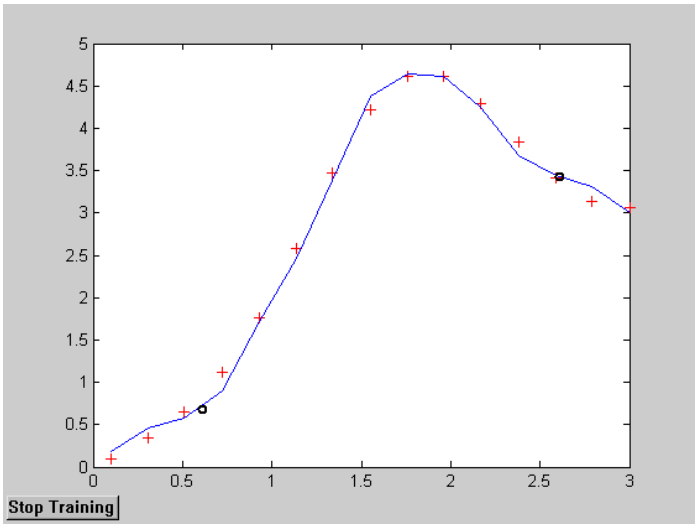
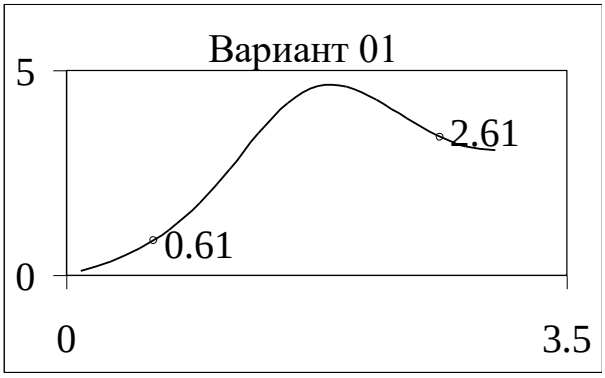
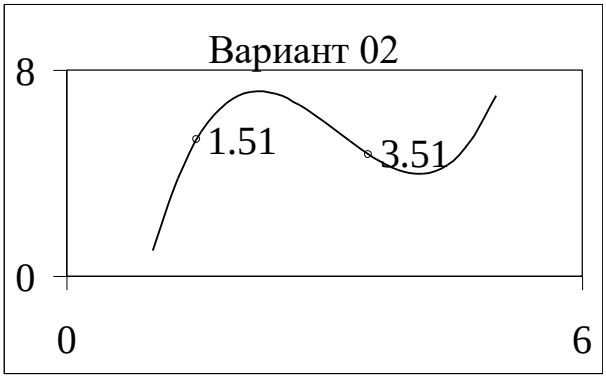
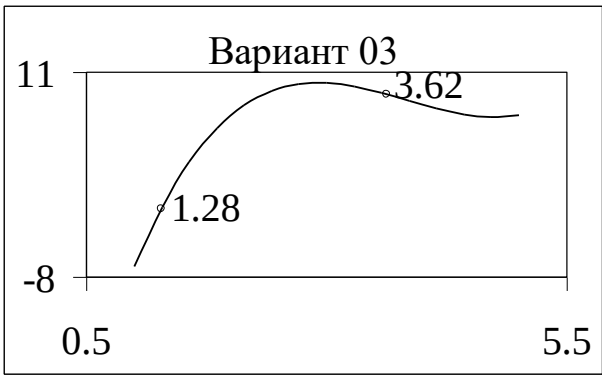


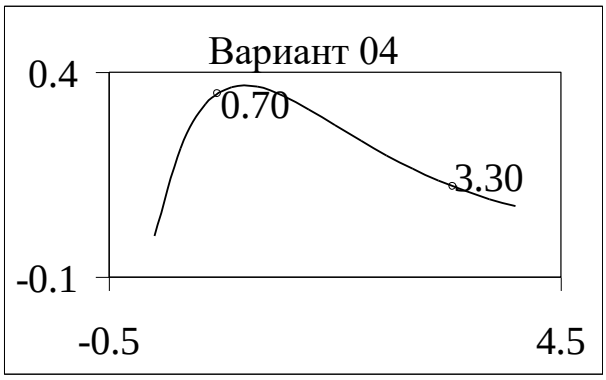
Рис.4. Результаты моделирования сети: + - исходные данные; сплошная линия и символ «о» – результаты моделирования всей зависимости и в контрольных точках

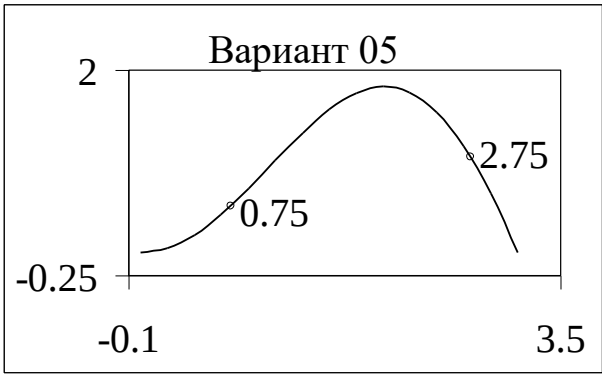
Варианты заданий

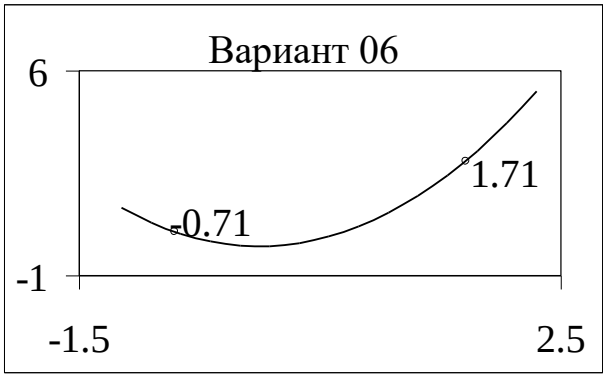


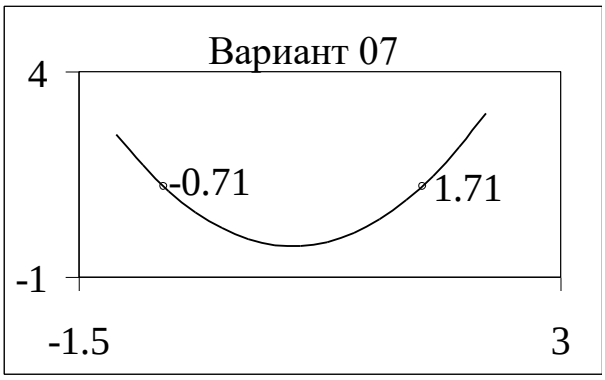


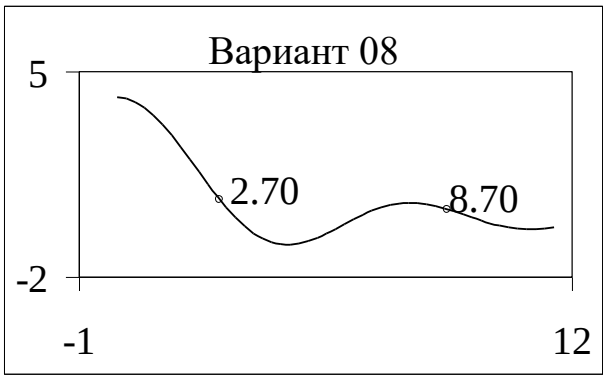

Вариант 03
11
-8
0.5
5.5
3.62
1.28

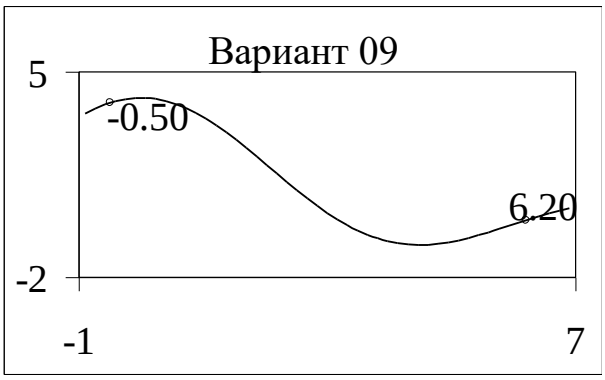

Вариант 04
0.4
-0.1
-0.5
4.5
0.70
3.30

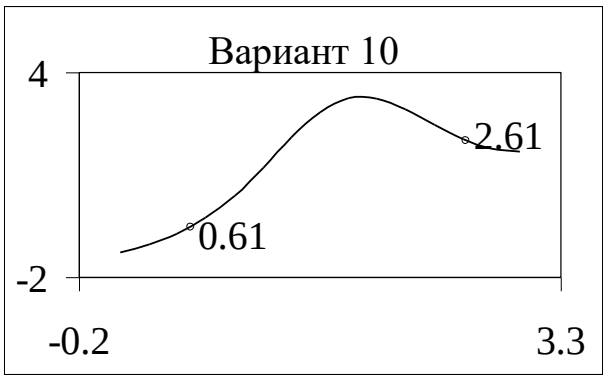

Вариант 05
2
-0.25
-0.1
3.5
2.75
0.75


Вариант 06
6
-1
-1.5
2.5
1.71
-0.71


Вариант 07
4
-1
-1.5
3
-0.71
1.71


Вариант 08
5
-2
-1
12
2.70
8.70


Вариант 09
5
-2
-1
7
-0.50
6.20


Вариант 10
4
-2
-0.2
3.3
2.61
0.61

Отчёт о выполнении работы

Отчёт о выполнении лабораторной работы должен быть выполнен на листах формата А4 и содержать следующие результаты:

1. Исходные данные (рис.2);
2. Текст программы с подробными комментариями;
3. Характеристику точности обучения (рис.3);
4. Результаты моделирования (рис.4);
5. Сопоставление результатов в контрольных точках;
6. Краткие выводы о результатах работы.

Практическое занятие № 3. Аппроксимация функции двух переменных

Цель работы

Научиться работать с радиальной базисной сетью, функции **newrbe** и **newrb**.

Краткие теоретические сведения

Радиальные базисные сети предназначены для аппроксимации функций. Возьмем произвольную непрерывную функцию и представим ее с помощью суммы колоколообразных функций. Аналитически это означает представление $f(x)$ в виде разложения по стандартному набору пространственно локализованных функций:

$$f(x) = \sum_i w_i \varphi\left(\left\|\mathbf{x} - \mathbf{c}_i\right\|\right), \quad (1)$$

где w_i - веса суммирования отдельных откликов, $\mathbf{c}_i$ - центры базисных радиальных функций. Это формула нейронной сети на основе радиальной базисной функции. Расстояние $\left\|\mathbf{x} - \mathbf{c}\right\|$ определяется как расстояние в евклидовом пространстве:

$$\left\|\mathbf{x} - \mathbf{c}\right\| = AB = \sqrt{(x_1 - c_1)^2 + (x_2 - c_2)^2 + \ldots}. \quad (2)$$

Функция **newrbe** формирует радиальную базисную сеть с нулевой ошибкой. Сеть с радиальными базисными функциями представляет собой, как правило, сеть с тремя слоями: обычным входным слоем, скрытым радиальным базисным слоем и выходным линейным слоем. Функция **newrb** формирует радиальную базисную сеть с нену-

левой ошибкой в отличие от **newrbe**. На рис.1 показана архитектура радиальной базисной сети.

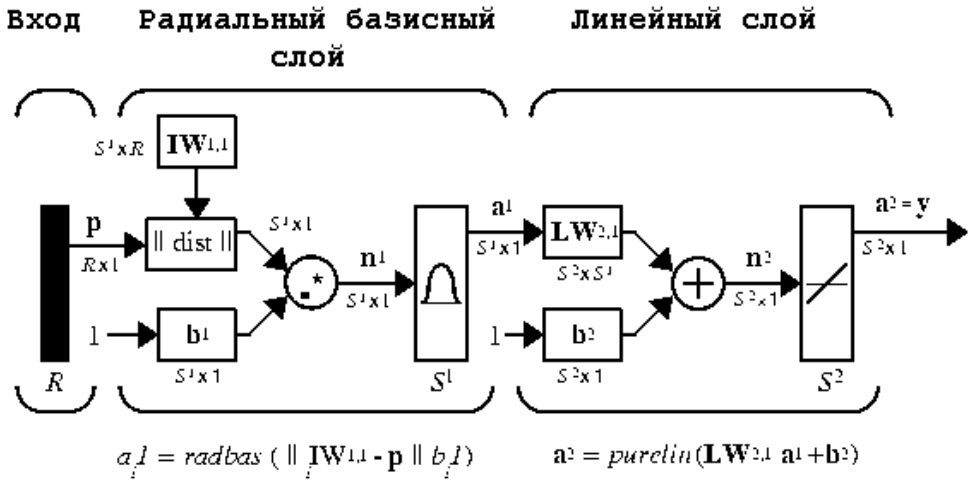


Рис.1. Схема архитектуры радиальной базисной сети

Пример решения типовой задачи

Пусть функция $z = e^{-x^2} \cdot e^{-y^2}$ задана на промежутках $x \in [-1,1]$, $y \in [-1.5,1.5]$; количество точек разбиений по x есть **nx**, а по y - **ny**. Тогда, используя следующий алгоритм построения радиальной базисной сети, можно построить график функции $z = f(x,y)$:

```
x1=-1.0; x2=+1.0; y1=-1.5; y2=+1.5;
nx=7; ny=9;
step_x=(x2-x1)/(nx-1); step_y=(y2-y1)/(ny-1);
step_min = min(step_x,step_y);
[x,y]=meshgrid([x1:step_x:x2], [y1:step_y:y2]);
z=exp(-x.^2).*exp(-y.^2);
surf(x,y,z), title('PS. Press<enter>');
pause;
xx=reshape(x,1,nx*ny);
yy=reshape(y,1,nx*ny);
zz=exp(-xx.^2).*exp(-yy.^2);
p=[xx; yy];
t=zz;
goal = 0.0371;
spread = 1.0*step_min;
net = newrb(p,t, goal,spread);
net.layers{1}.size
```

```
smlt=sim(net,p);
[zz' smlt']
smltr=reshape(smlt,ny,nx);
surf(x,y,smltr), title('AS. Press<enter>');
```

Рис.2 иллюстрирует график исходной функции $z = e^{-x^2} \cdot e^{-y^2}$.

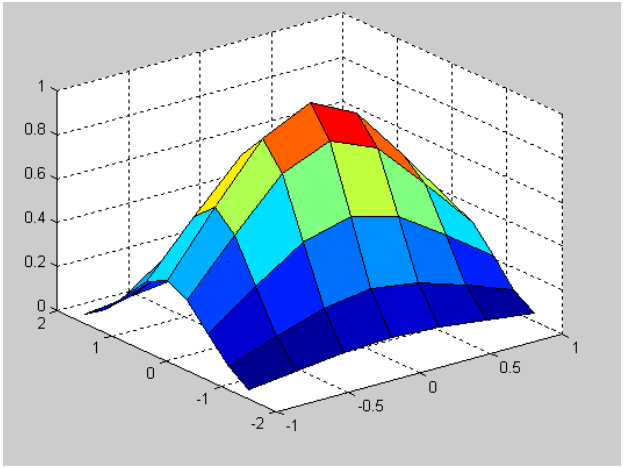


Рис.2. График исходной функции двух переменных

На рис.3 показана характеристика точности обучения радиальной базисной сети и допустимая среднеквадратичная ошибка сети **Goal=0.0371**.

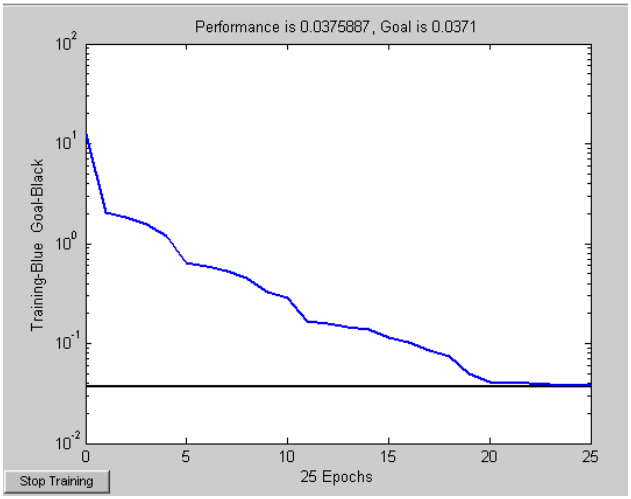


Рис.3. Характеристика точности обучения в зависимости от количества эпох обучения

На рис.4 отображён результат аппроксимации нелинейной зависимости, построенный с помощью радиальной базисной функции.

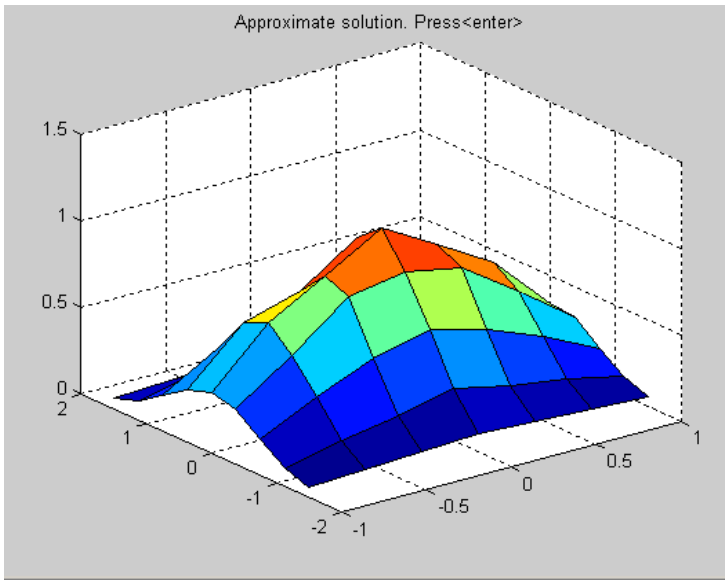


Рис.4. Результат моделирования исходной функции

Сопоставляя рис.2 и рис.4, можно сделать вывод об удовлетворительности полученных результатов. Лучших результатов можно добиться, варьируя параметры **goal** и **spread**.

Варианты заданий

1. $z = \cos(x)\cos(y), x, y \in [-\frac{\pi}{2}, \frac{\pi}{2}]$
2. $z = \frac{\sin(x)}{x}\frac{\sin(y)}{y}, x, y \in [-1,1]$
3. $z = x^2 y^2, x, y \in [-1.5,1.5]$
4. $z = x^3 \sin(y) + 1, x \in [-1,1], y \in [-1.5,1.5]$
5. $z = x^2 \sin(y) - 1, x \in [-1,1], y \in [-1.5,1.5]$
6. $z = e^x + 3y, x \in [0,1], y \in [-2,1]$
7. $z = \sqrt{y^2 + x^3}, x \in [-1,1], y \in [-2,2]$
8. $z = \sqrt{y^3 + x^2}, x \in [-1,1], y \in [-2,2]$
9. $z = \frac{1}{\sqrt{y^2 + x^3}}, x, y \in [-1,1]$
10. $z = \frac{1}{\sqrt{y^3 + x^2}}, x, y \in [-1,1]$
11. $z = x^2 \cos(y) + 1, x \in [-1,1], y \in [-\pi/4, \pi/4]$
12. $z = x^3 \cos(y) + 1, x \in [-0.5,0.5], y \in [-\pi/4, \pi/4]$

13. $z = \sqrt{y^2 + x^4}, x, y \in [-1,1]$
14. $z = \sin\left(\frac{y}{(1+x^2)}\right) + 1, x \in [-1,1], y \in [-\pi, \pi]$
15. $z = \sin(x^2) + y^2, x \in [-1.77,1.77], y \in [-0.3,0.3]$
16. $z = \cos(x^2) + xy, x \in [-0.886,0.886], y \in [-0.3,0.3]$
17. $z = (x-1)^2 \cdot \exp(y^2), x \in [0.5,2], y \in [-1,1]$
18. $z = \frac{\cos(x)}{x^2+1}, x \in [-5,5], y \in [-\pi/2, \pi/2]$
19. $z = \sin(x) \cdot \sin(y), x, y \in [-\pi, 0]$
20. $z = \ln(x) \cdot \ln(y), x, y \in [0.5,2]$

Отчёт о выполнении работы

Отчёт о выполнении лабораторной работы должен быть выполнен на листах формата А4 и содержать следующие результаты:

1. Исходные данные – выбор функции двух переменных и области определения функции, построение графика функции (рис.2);
2. Текст программы с подробными комментариями;
3. Результаты моделирования (рис.3,4);
4. Контрольный пример;
5. Объяснение результатов проделанной работы.

Практическое занятие № 4. Сеть Кохонена, самоорганизующаяся нейронная сеть

Цель работы

Изучить функционирование и процедуру обучения сети Кохонена с помощью функции **newc**.

Краткие теоретические сведения

Сети Кохонена, или самоорганизующиеся карты (Kohonen maps), предназначены для решения задач автоматической классификации, когда обучающая последовательность образов отсутствует (обучение без учителя). Сеть Кохонена является двухслойной. Она содержит слой входных нейронов и собственно слой Кохонена. Слой Кохонена

может быть одномерным, двумерным и трехмерным. В первом случае нейроны расположены в цепочку; во втором – они образуют двумерную сетку (обычно в форме квадрата или прямоугольника), а в третьем – трехмерную систему. Определение весов нейронов слоя Кохонена основано на использовании алгоритмов автоматической классификации (кластеризации или самообучения).

На вход сети подаются последовательно значения векторов $\mathbf{x}_l = (x_1, x_2, ..., x_n)_l$, представляющих отдельные последовательные наборы данных для поиска кластеров, то есть различных классов образов, причем число этих кластеров заранее неизвестно. На стадии обучения (точнее самообучения) сети входной вектор $\mathbf{x}$ попарно сравнивается со всеми векторами $\mathbf{w}_j$ всех нейронов сети Кохонена. Вводится некоторая функция близости d (например, в виде евклидова расстояния). Активный нейрон с номером c слоя Кохонена, для которого значение функции близости $d(\mathbf{x}, \mathbf{w}_c)$ между входным вектором $\mathbf{x}$, характеризующим некоторый образ, к векторам $\mathbf{w}_c$ максимально, объявляется «победителем». При этом образ, характеризующийся вектором $\mathbf{x}$, будет отнесен к классу, который представляется нейроном-«победителем».

Рассмотрим алгоритм самообучения сетей Кохонена. Обозначим функцию близости $z = \|\mathbf{x} - \mathbf{w}\|$. Выигрывает нейрон с

$$\|\mathbf{x} - \mathbf{w}_c\| = \min \|\mathbf{x} - \mathbf{w}_j\|. \qquad (1)$$

Близость $\mathbf{x}'$ и $\mathbf{w}'$ можно переделить, пользуясь скалярным произведением, как

$$z_j = 1 - (\mathbf{x}', \mathbf{w}'_j) = 1 - \sum_{i=1}^{n} x'_i w_{ij} . \qquad (2)$$

На стадии самообучения сети Кохонена осуществляется коррекция весового вектора не только нейрона-«победителя», но и весовых векторов остальных активных нейронов слоя Кохонена, однако, естественно, в значительно меньшей степени – в зависимости от удаления от нейрона-«победителя». При этом форма и величина окрестности вокруг нейрона-«победителя», весовые коэффициенты нейронов которой также корректируются, в процессе обучения изменяются. Сначала начинают с очень большой области – она, в частности, может

включать все нейроны слоя Кохонена. Изменение весовых векторов осуществляется по правилу

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \eta(t) d_{cj}(t)[\mathbf{x}(t) - \mathbf{w}_j(t)], \quad j=1,2,\ldots, n, \qquad (3)$$

где $\mathbf{w}_j(t)$ - значение весового вектора на шаге t самообучения сети, $d_{cj}(t)$ - функция близости между нейронами слоя Кохонена и $\eta(t)$ - изменяемый во времени коэффициент шага коррекции. В качестве $\eta(t)$ обычно выбирается монотонно убывающая функция ($0<\eta(t)<1$), то есть алгоритм самообучения начинается сравнительно большими шагами адаптации и заканчивается относительно малыми изменениями.

В результате n-мерное входное пространство R_n отобразится на m-мерную сетку (слой Кохонена). Это отображение реализуется в результате рекуррентной (итеративной) процедуры самообучения (unsupervised learning). Отличительная особенность этого отображения – формирование кластеров (clusters) или классов. По завершении процесса самообучения на стадии реального использования сети Кохонена неизвестные входные образы относятся к одному из выявленных кластеров (классов) по близости к некоторому весу, принадлежащему определенному кластеру, выявленному на стадии самообучения.

На рис.1 показана архитектура слоя Кохонена.

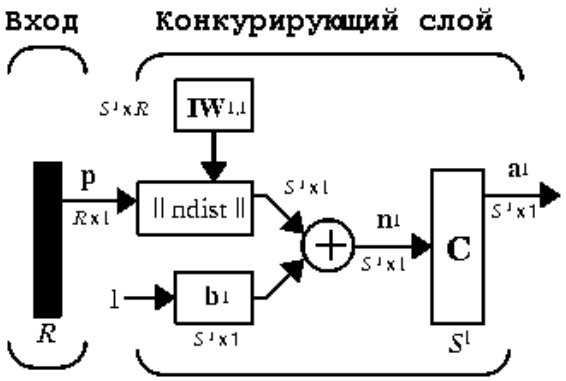


Рис.1. Схема архитектуры слоя Кохонена

Пример решения типовой задачи

Пусть заданы 28 случайных векторов, изображённых на графике крестами (рис.2). Оцифровав данный график, можно получить массив входных данных $P(1,:),P(2,:)$ (табл.1).

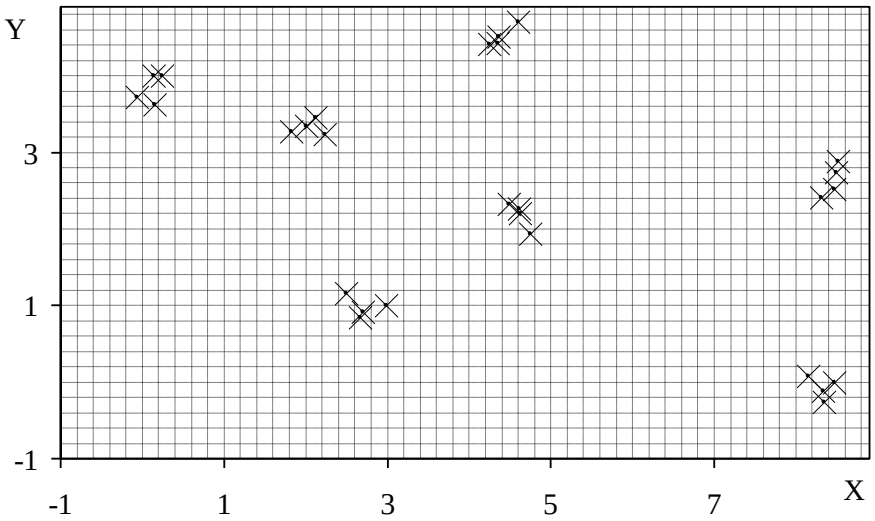


Рис.2. Распределение входных векторов

Таблица 1

Массив входных данных

P(1,:)	3.0	-0.1	4.3	4.6	8.5	1.8	8.3	2.5	0.2	4.6
P(2,:)	1.0	3.7	4.4	2.3	2.7	3.3	-0.1	1.2	3.6	4.7
P(1,:)	4.5	8.5	2.2	8.2	2.7	0.2	4.4	4.6	8.3	2.0
P(2,:)	2.3	2.9	3.2	0.1	0.8	4.0	4.5	2.2	2.4	3.3
P(1,:)	8.5	2.7	0.1	4.3	4.7	8.5	2.1	8.3		
P(2,:)	0.0	0.9	4.0	4.4	1.9	2.5	3.5	-0.3		

Следующий алгоритм демонстрирует процедуру обучения самоорганизующейся нейронной сети Кохонена.

```
plot(P(1,:), P(2,:), '+m');
title('Input vectors');
xlabel('P(1,:)');
ylabel('P(2,:)');
hold on;
nclusters = 7;
nvectors = 4;
a1 = -10;
a2 = +10;
b1 = -5;
b2 = +5;
net = newc([a1 a2; b1 b2], nclusters, 0.1, 0.0005);
```

```
wo = net.IW{1};
bo = net.b{1};
co = exp(1)./bo;+
net.trainParam.epochs=49;
net.trainParam.show=7;
net = train(net,P);
w = net.IW{1};
bn = net.b{1};
cn = exp(1)./bn;
plot(w(:,1),w(:,2),'kp');
```

На рис.3 представлены исходные данные (кресты) и полученные центры кластеризации (звёзды).

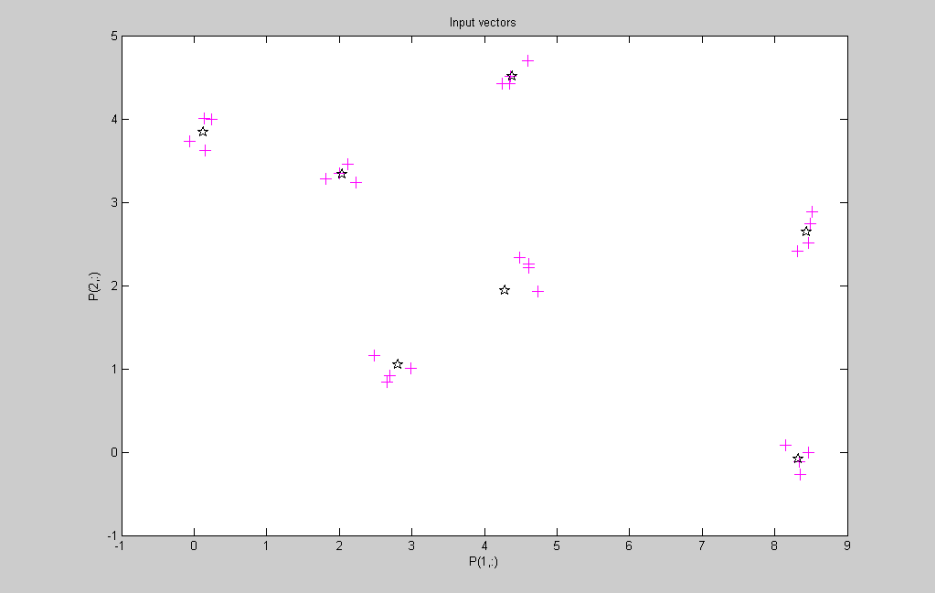
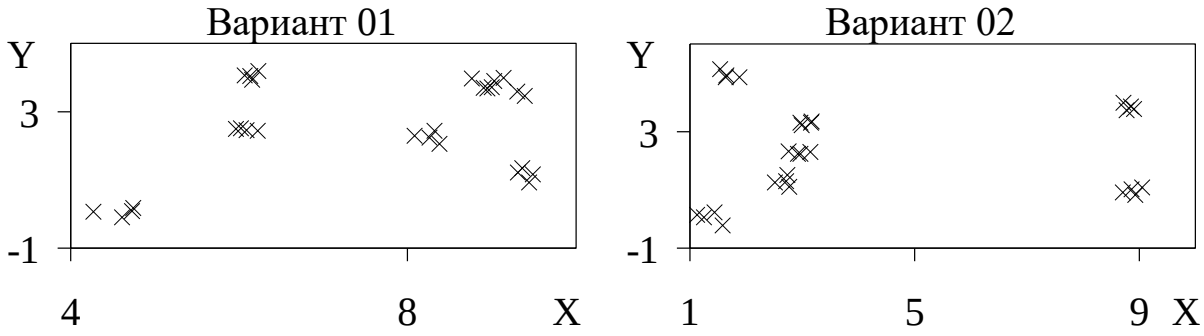
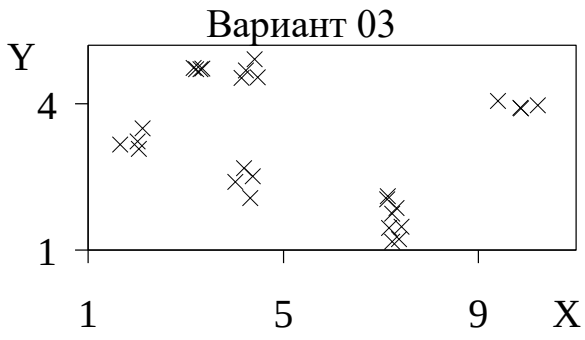
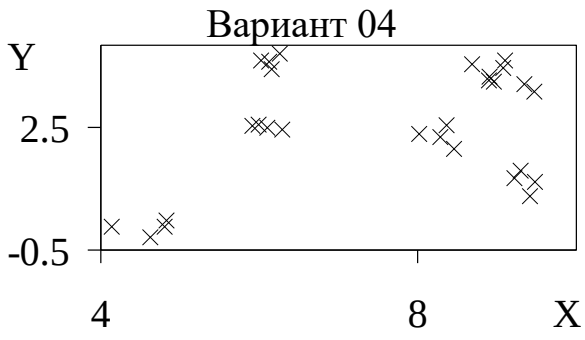


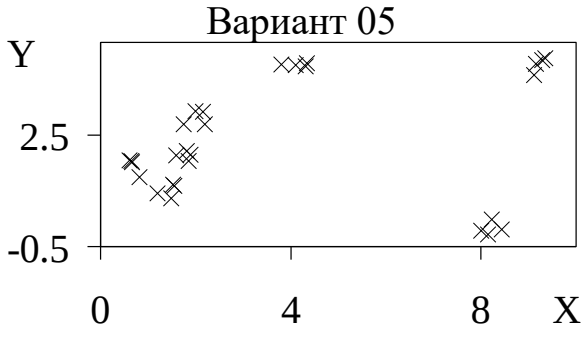
Рис.3. Распределение входных данных (кресты) и положение центров кластеризации (звёзды)

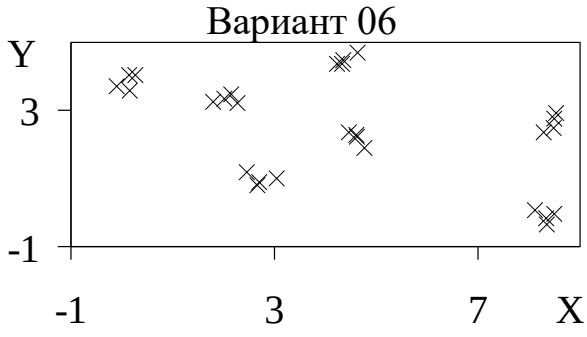
Варианты заданий

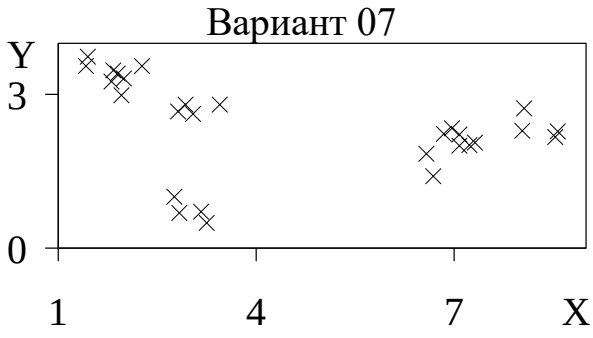


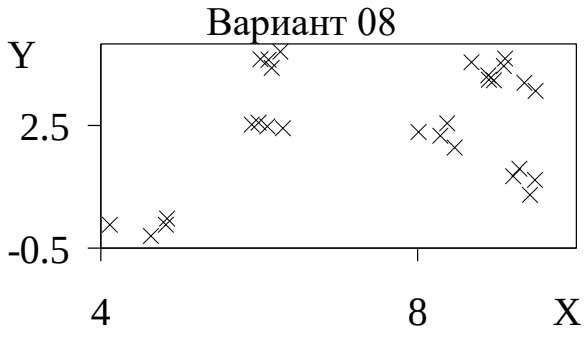


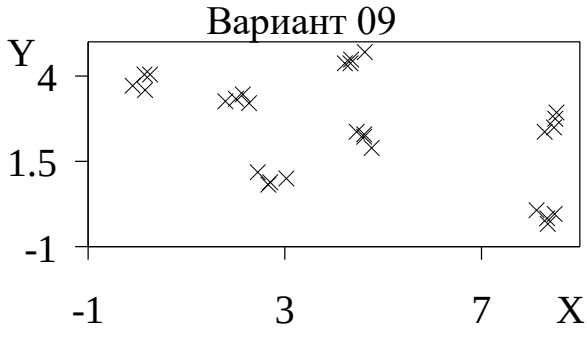


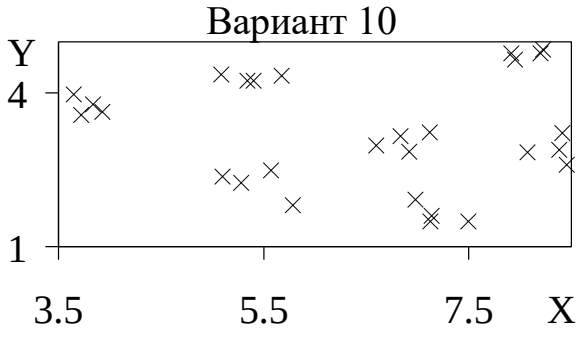












Отчёт о выполнении работы

Отчёт о выполнении лабораторной работы должен быть выполнен на листах формата А4 и содержать следующие результаты:

1. Исходные данные (рис.2);
2. Текст программы с подробными комментариями;
3. Результаты моделирования (рис.3);
4. Краткое описание расположения центров кластеров;
5. Список исходных точек с указанием принадлежности к определенному кластеру;
6. Контрольный пример.

Практическое занятие № 5. Сеть Хопфилда

Цель работы

Научиться работать с сетью Хопфилда **newhop**, исследовать устойчивость сети и её сходимость.

Краткие теоретические сведения

Американский исследователь Хопфилд в 80-х годах 20-го века предложил специальный тип нейросетей. Названные в его честь сети Хопфилда являются рекуррентными или сетями с обратными связями и предназначены для распознавания образов. Обобщенная структура этой сети представляется, как правило, в виде системы с обратной связью выхода с входом.

В сети Хопфилда входные сигналы нейронов являются одновременно и выходными сигналами сети: $x_i(k)=y_i(k\text{-}1)$, при этом возбуждающий вектор особо не выделяется. В классической системе Хопфилда отсутствует связь нейрона с собственным выходом, что соответствует $w_{ii}=0$, а вся матрица весов является симметричной: $w_{ij}=w_{ji}$

$$\hat{W}=\hat{W}^T. \qquad (1)$$

Симметричность матрицы весов гарантирует сходимость процесса обучения. Процесс обучения сети формирует зоны притяжения некоторых точек равновесия, соответствующих обучающим данным. При использовании ассоциативной памяти мы имеем дело с обучающим вектором $\mathbf{x}=(x_1,x_2,\ldots x_n)$, либо с множеством этих векторов, которые в результате проводимого обучения определяют расположение конкретных точек притяжения (аттракторов).

Каждый нейрон имеет функцию активации сигнум со значениями ± 1:

$$sign(a) = \begin{cases} 1, a \geq 0 \\ -1, a < 0 \end{cases}. \quad (2)$$

Это означает, что выходной сигнал *i*-го нейрона определяется функцией:

$$y_i = sign(\sum_{j=1}^{N} w_{ij} x_j + b_i), \quad (3)$$

где N обозначает количество нейронов, $N=n$. Часто постоянная составляющая b_i, определяющая порог срабатывания отдельных нейронов, равна 0. Тогда циклическое прохождение сигнала в сети Хопфилда можно представить соотношением:

$$y_i(k) = sign(\sum_{j=1}^{N} w_{ij} y_j(k-1)) \quad (4)$$

с начальным условием $y_j(0) = x_i$.

В процессе функционирования сети Хопфилда можно выделить два режима: обучения и классификации. В режиме обучения на основе известных обучающих выборок $\mathbf{x}_i$ подбираются весовые коэффициенты w_{ij}. В режиме классификации при зафиксированных значениях весов и вводе конкретного начального состояния нейронов $\mathbf{y}(0) = \mathbf{x}$ возникает переходный процесс, протекающий в соответствии с выражением (2) и заканчивающийся в одном из локальных устойчивых положений, задаваемом биполярным вектором со значениями $y_j = \pm 1$, для которого $\mathbf{y}(k) = \mathbf{y}(k-1)$.

Обучение не носит рекуррентного характера. Достаточно ввести значения (правило Хебба) весов, выразив их через проекции вектора точки притяжения эталонного образа:

$$w_{ij} = \frac{1}{N} x_i x_j, \quad (5)$$

В соответствии с этим правилом сеть дает правильный результат при входном примере, совпадающим с эталонным образцом, поскольку:

$$\sum_{j=1}^{N} w_{ij} x_j = \frac{1}{N} \sum_{j=1}^{N} x_i (x_j x_j) = x_i \sum_{j=1}^{N} 1 \cdot \frac{1}{N} = x_i, \quad (6)$$

так как вследствие биполярности значений элементов вектора $\mathbf{x}$ всегда $x_j^2 = 1$.

При вводе большого количества обучающих выборок $\mathbf{x}^{(k)}$ для $k=1,2,...p$ веса w_{ij} подбираются согласно обобщенному правилу Хебба в соответствии с которым:

$$w_{ij} = \frac{1}{N}\sum_{k=1}^{l} x_i^{(k)} x_j^{(k)}. \qquad (7)$$

Благодаря такому режиму обучения веса принимают значения, определяемые усреднением множества обучаемых выборок. В случае множества обучаемых выборок актуальным становится вопрос о стабильности ассоциативной памяти.

Сеть Хопфилда является автоассоциативной сетью (рис.1). Дискретная сеть Хопфилда имеет следующие характеристики: она содержит один слой элементов; каждый элемент связывается со всеми другими элементами, но не связан с самим собой; за один шаг работы обновляется только один элемент сети; элементы обновляются в случайном порядке; выход элемента ограничен значениями 0 или 1.

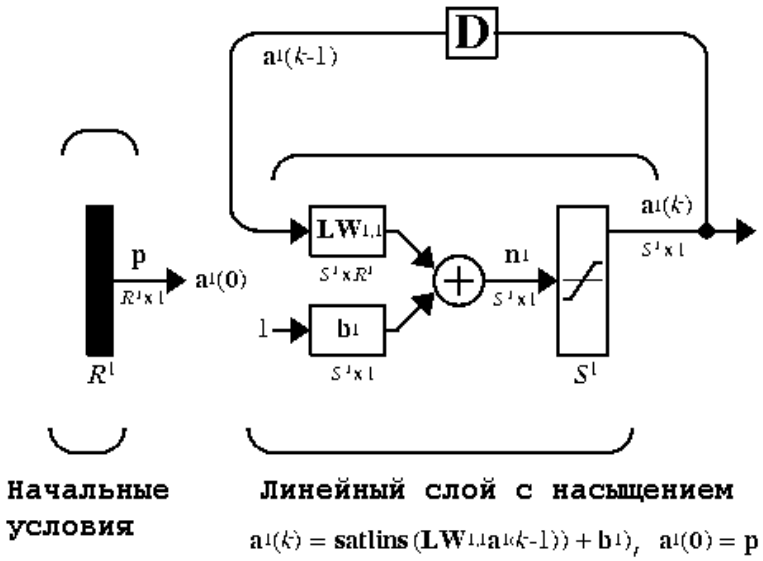


Рис.1. Схема архитектуры модифицированной сети Хопфилда

Пример решения типовой задачи

Рассмотрим сеть Хопфилда с четырьмя нейронами и определим четыре точки равновесия:

```
T = [+1 -1; -1 +1; +1 +1; -1 -1];
T=T';
plot(T(1,:),T(2,:),'rh','MarkerSize',13);
hold on;
axis([-1.1 1.1 -1.1 1.1]);
title('Hopfield Network State Space');
xlabel('a(1)');
```

```
ylabel('a(2)');
net = newhop(T);
[Y,Pf,Af] = sim(net,4,[],T);
Y
PF
AF
pause
color = 'rgbmy';
for i=1:25
  a = {rands(2,1)};
  [y,Pf,Af] = sim(net,{1 20},{},a);
  record=[cell2mat(a) cell2mat(y)];
  start=cell2mat(a);
  plot(start(1,1),start(2,1),'kx',record(1,:),
  record(2,:), color(rem(i,5)+1),'LineWidth',5)
end
```

На рис.2 показано поведение обученной сети при случайных начальных условиях **a**.

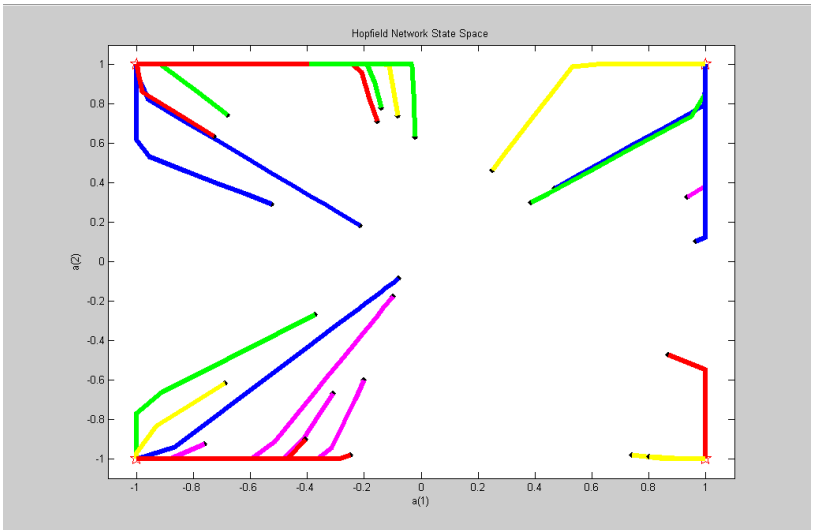


Рис.2. Поведение сети Хопфилда при случайных начальных условиях **a**

Отчёт о выполнении работы

Отчёт о выполнении лабораторной работы должен быть выполнен на листах формата А4 и содержать следующие результаты:

1. Исходные данные;

2. Текст программы с подробными комментариями;

3. Результаты моделирования (рис.2);

4. Контрольный пример;

5. Краткие **письменные** ответы на контрольные вопросы, содержащиеся в приложении.

Практическое занятие № 6. Создание модели для исследования контроллеров

Цель работы

Научиться работать с моделью привода постоянного тока, настройками его регуляторов, исследовать характеристики.

Краткие теоретические сведения

Представим объект обобщенной моделью в пространстве состояний в виде:

$$\dot{\mathbf{x}} = \mathbf{A}x+\mathbf{B}u$$
$$y = \mathbf{C}x, \qquad (1)$$

где $\mathbf{A}-(n\times n)$ - матрица коэффициентов; $\mathbf{B}-(n\times m)$ - матрица управления; $\mathbf{C}-(r\times n)$ - матрица выходного сигнала.

Состояние системы в любой момент времени может быть охарактеризовано положением в n - мерном пространстве состояний изображающей точки, координатами которой являются выходные переменные элементарных звеньев системы.

Этими точками являются: выходное напряжение регулятора скорости x_1, регулятора тока x_2, силового преобразователя x_3, ток двигателя x_4, скорость вращения ротора электродвигателя x_5, промежуточного x_6 и выходного звеньев механической системы x_7, а также упругие моменты x_8 и x_9 (рис. 1).

Для нее запишем уравнения всех элементов в виде::

$$px_1=A_{15}\,x_5+B_{11}\,u_ç$$
$$px_2=A_{21}\,x_1+A_{24}\,x_4+A_{25}\,x_5+B_{21}\,u_ç$$
$$px_3=A_{31}\,x_1+A_{32}\,x_2+A_{33}\,x_3+A_{34}\,x_4+A_{35}\,x_5+B_{31}\,u_ç$$
$$px_4=A_{43}\,x_3+A_{44}\,x_4+A_{45}\,x_5$$
$$px_5=A_{54}\,x_4+A_{55}\,x_5+A_{58}\,x_8 \qquad (2)$$

$$px_6 = A_{66}x_6 + A_{68}x_8 + A_{69}x_9$$
$$px_7 = A_{77}x_7 + A_{79}x_9 + B_{72}M_{\tilde{n}}$$
$$px_8 = A_{84}x_4 + A_{85}x_5 + A_{86}x_6 + A_{88}x_8 + A_{89}x_9$$
$$px_9 = A_{96}x_6 + A_{97}x_7 + A_{98}x_8 + A_{99}x_9 + B_{92}M_c$$

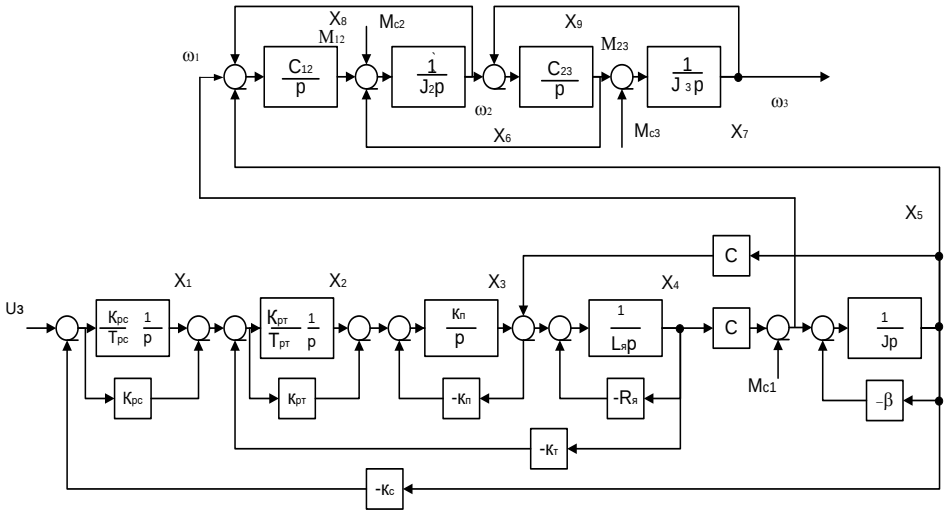


Рис.1.

Вектор переменных состояния представляет собой вектор столбец, состоящий из девяти строк $[x_1 \ldots x_9]^T$. Квадратная матрица **A** имеет размерность 9x9:

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & A_{15} & 0 & 0 & 0 & 0 \\ A_{21} & 0 & 0 & A_{24} & A_{25} & 0 & 0 & 0 & 0 \\ A_{31} & A_{32} & A_{33} & A_{34} & A_{35} & 0 & 0 & 0 & 0 \\ 0 & 0 & A_{43} & A_{44} & A_{45} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & A_{54} & A_{55} & 0 & 0 & A_{58} & 0 \\ 0 & 0 & 0 & 0 & 0 & A_{66} & 0 & A_{68} & A_{69} \\ 0 & 0 & 0 & 0 & 0 & 0 & A_{77} & 0 & A_{79} \\ 0 & 0 & 0 & A_{84} & A_{85} & A_{86} & 0 & A_{88} & A_{89} \\ 0 & 0 & 0 & 0 & 0 & A_{96} & A_{97} & A_{98} & A_{99} \end{bmatrix} \quad (3)$$

Двухстолбцовая матрица внешних воздействий имеет размерность 9x2:

$$\mathbf{B}=\begin{bmatrix} B_{11} & B_{21} & B_{31} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & B_{72} & 0 & B_{92} \end{bmatrix}^T$$

Последний вектор внешних воздействий представлен только двумя элементами:

$$\mathbf{u}=\begin{bmatrix} u_3 \\ M_C \end{bmatrix}$$

Входящая в уравнение матрица Спредставляется вектором строкой и для анализа выходной переменной, например x_5, имеет следующий вид:C=[000010000].

Коэффициенты, входящие в уравнения (2), находят по формулам:

$$A_{15}=-K_{PC}K_C/T_{PC} \qquad A_{44}=-r_я/L_Я$$
$$A_{21}=K_{PT}/T_{PT} \qquad A_{45}=-c/L_Я$$
$$A_{24}=-K_{PT}K_T/T_{PT} \qquad A_{54}=-1/J_1$$
$$A_{25}=-K_{PC}K_C K_{PT}/T_{PT} \qquad A_{55}=-\beta_1/J_1$$
$$A_{31}=K_{PT}K_П/\tau \qquad A_{58}=-A_{54}$$
$$A_{32}=K_П\tau \qquad A_{66}=-\beta_2/J_2$$
$$A_{32}=K_П\tau \qquad A_{66}=-\beta_2/J_2$$
$$A_{33}=-1/\tau \qquad A_{68}=-1/J_2$$
$$A_{34}=-K_{PT}K_T K_П/\tau \qquad A_{69}=-A_{68}$$
$$A_{35}=-K_{PC}K_C K_{PT}/\tau \qquad A_{77}=-\beta_3/J_3$$
$$A_{43}=1/L_Я \qquad A_{79}=-1/J_3$$
$$A_{84}=\beta_{12}\beta_1/J_1 \qquad B_{11}=K_{PC}/T_{PC}$$
$$A_{85}=C_{12}-\beta_{12}\beta_1/J_1 \qquad B_{21}=K_{PC}K_C/T_{PC}$$
$$A_{86}=-C_{12}-\beta_{12}\beta_2/J_2 \qquad B_{31}=K_{PC}K_{PT}K_П/\tau$$
$$A_{88}=-\beta_{12}(1/J_2+1/J_3) \qquad B_{72}=-1/J_3$$
$$A_{89}=\beta_{12}/J_2 \qquad B_{92}=-\beta_{23}/J_3$$
$$A_{96}=C_{23}-\beta_{23}\beta_2/J_2$$
$$A_{97}=-C_{23}-\beta_{23}\beta_2/J_2$$
$$A_{98}=\beta_{23}/J_2$$
$$A_{99}=-\beta_{23}(1/J_2+1/J_3)$$

Используя математические пакеты Matcad или Matlab и программируя уравнения, получим характеристики механической системы и электропривода. Результаты можно получить, если приведенную структурную схему воспроизвести в системе Matlab.

Синтез контуров

Синтез контуров ведется методом последовательной коррекции из условия настройки на оптимум по модулю (ОМ). Особенность синтезированной таким образом каждого контура состоит в том, что время переходного процесса не зависит от постоянной времени его объекта и определяется малой постоянной времени этого контура.

1. Синтез регулятора тока

Структурную схему регулятора тока (РТ) представим в виде структурной схемы рис. 2.

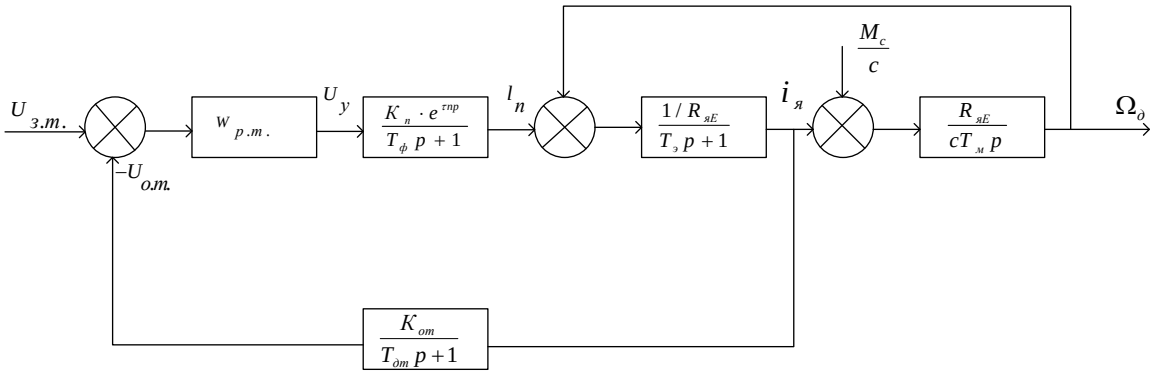


Рис. 2. Расчетная схема регулятора тока

Передаточная функция объекта регулятора тока $W_{р.т.}$ с учетом внутренней о.с. по э.д.с. двигателя имеет вид

$$W_{о.р.т.} = \frac{K_п e^{-\tau_{пр}}}{T_ф p + 1} \times \frac{T_м}{R_\Sigma (T_э T_м p^2 + T_м p + 1)}$$

Отнесем инерционность фильтра и чистое запаздывание к некомпенсируемым инерционностям контура, приняв $T_\mu = \tau_п + \tau_ф$, для этих схем обычно $T_\mu = 0{,}01\ с$.

Пренебрегаем влиянием внутренней о.с. по э.д.с. которая, сказывается на точности регулирования тока, но незначительно влияет на динамические показатели, тогда

$$W_{о.р.т.} = \frac{K_п}{T_\mu p + 1} \times \frac{\frac{1}{R_{я\Sigma}}}{T_э p + 1}$$

В результате последовательной коррекции необходимо получить ПФ разомкнутого контура тока

$$W_{раз.т.} = \frac{\frac{1}{K_{о.т.}}}{\alpha_т T_\mu p\left(T_\mu p + 1\right)}$$

Разделив (3) на (2) получим

$$W_{р.т.} = \frac{T_э p + 1}{T_{и.т.} p} = \frac{T_э}{T_{и.т.}} + \frac{1}{T_{и.т.} p}$$

Получена передаточная функция ПИ регулятора с постоянной интегрирования

$$T_{и.т.} = \frac{K_{о.т.} K_{п.}}{R_{я\Sigma}} \alpha_т T_\mu$$

и коэффициентом усиления пропорциональной части $K_{у.т.} = \frac{T_э}{T_{ит}}$.

Структура ПИ регулятора имеет вид

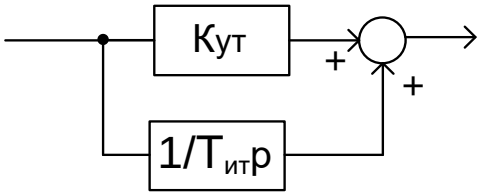


Рис. 3 Структура ПИ регулятора

Входная цепь РТ и корректирующее звено в цепи обратной связи, обеспечивают требуемую ПФ.

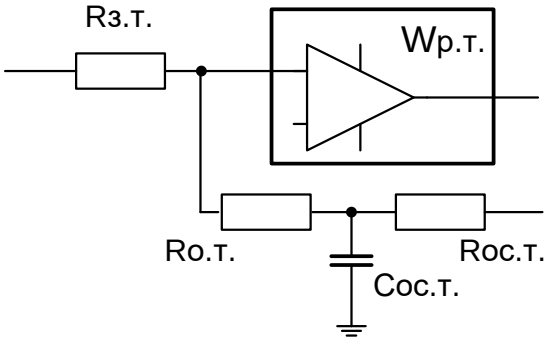


Рис. 4 Регулятор тока

Так как $T_{и.т.} = R_{з.т.} C_{ос.т}$ и $K_{у.т.} = \frac{R_{ос.т}}{R_{о.т}}$,

то компенсирующая постоянная регулятора $T_{д.т.}$ определяется соотношением

$$T_{д.т.} = R_{ос.т} C_{ос.т} = T_{э.}$$

Для расчета параметров схемы необходимо задаться максимальным значением $U_{з.т.} = U_{з.т.макс.}$, соответствующим максимально допустимому току якоря $I_{я.макс}$ и определить коэффициент обратной связи по току

$$K_{о.т.} = \frac{U_{з.т.макс}}{U_{я.макс}} = K_{д.т.}(\frac{R_{з.т.}}{R_{о.т.}})$$

где $K_{д.т.} = \frac{U_{д.т.}}{I_{я.макс}}$.

В расчетах следует принимать $U_{з.т.} = 10\,B$, что соответствует уровню входного напряжения приводов по стандарту СЭВ.

Далее определяется $T_{и.т.}$, при этом принимаются $\alpha_т = 2$, а затем задавались $R_{з.т.}$ и определяют $C_{ос.т.}$, $R_{ос.т}$, и $R_{о.т.}$

Передаточная функция замкнутого контура тока имеет вид

$$W_{зам.т} = \frac{\frac{1}{K_{о.т.}}}{\alpha_т T_\mu p\left(T_\mu p + 1\right)}$$

Настройка контура на технический оптимум ($\alpha_т = 2$), дает минимальное время регулирования $t_{р.} = 4{,}7\ T_\mu$ и $T_\mu = 0{,}01$, $t_{р.} = 0{,}05\ с$.

Синтез регулятора скорости (РС)

Объект регулирования РС состоит из замкнутого контура РТ и механического звена электропривода и может быть представлен структурной схемой рис. 5.

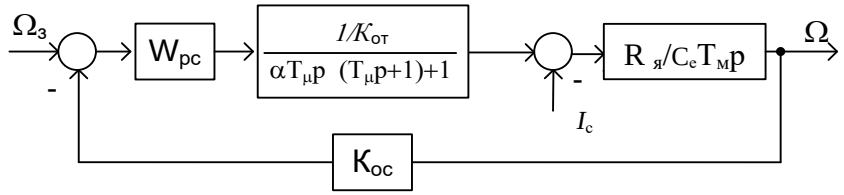


Рис. 5. Структурная схема настройки регулятора скорости

Этой структуре соответствует ПФ объекта регулирования

$$W_{орс} = \frac{\frac{1}{K_{о.т.}}}{\alpha_т T_\mu p\left(T_\mu p + 1\right)} * \frac{R_{яΣ}}{C_e T_м p}.$$

В соответствии с принятой методикой членом при p^2 можно пренебречь, тогда

$$W_{орс} = \frac{\frac{1}{K_{о.т.}}}{\alpha_m T_\mu p + 1} * \frac{R_{яΣ}}{C_e T_м p} \ .$$

Из выражения видно, что для контура скорости некомпенсируемая постоянная в α_m раз больше, чем для контура тока, т.е. $T_{\mu c} = \alpha_m T_\mu$. Последовательная коррекция РС должна обеспечить оптимальную ПФ разомкнутого РС в виде

$$W_{раз.с.} = \frac{\frac{1}{K_{ос}}}{\alpha_c \alpha_m T_\mu p\left(\alpha_m T_\mu p + 1\right)},$$

где $\alpha_c = \frac{T_{ос}}{T_{\mu c}}$ (принимается $\alpha_c = 2$). Получим искомую ПФ разомкнутого контура в виде

$$W_{р.с.} = \frac{C_e K_{от} T_м}{R_{яΣ} K_{ос} \alpha_c \alpha_m T_\mu}.$$

Необходим пропорциональный РС, входные, прямые и обратные цепи, реализующие этим выражение, имеют вид

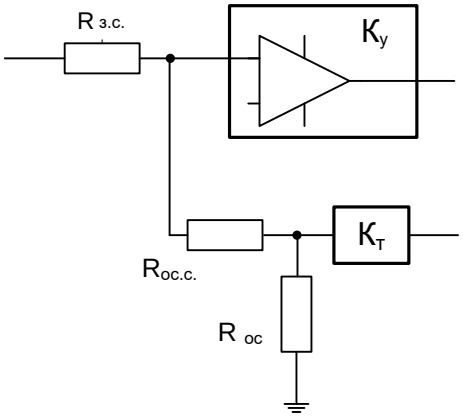


Рис. 6. Структура регулятора

Для определения параметров схемы рис.6. рассмотрим выражения для ошибок в РС, представив его структурную схему в виде рис. 7.

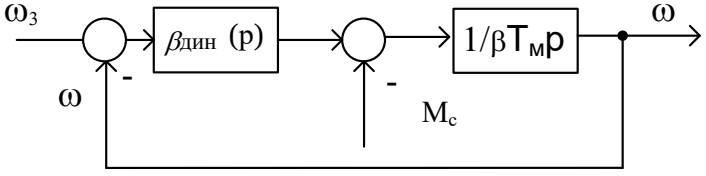


Рис. 7. Структурная схема регулятора

Уравнение динамической механической характеристики для этой структуры будет

$$M(p) = -\beta_{дин.с}(p)(\omega_{оз} - \omega),$$

при этом принимается допущение: пренебрегаем увеличением жесткости механической характеристики за счет внутренней обратной связи по э.д.с., т.к. в связи с малостью некомпенсируемой постоянной времени РТ является быстродействующие в диапазоне малых и средних частот. Тогда

$\beta_{дин.с}(p) = -\frac{\beta T_м}{\alpha_с \alpha_т T_\mu (\alpha_т T_\mu p+1)} = -\frac{\beta_{зам.с}}{\alpha_т (T_\mu p+1)}$,

где $\beta = \frac{R_{я\Sigma}}{C_e}$, $\beta_{зам.с} = \frac{\beta T_м}{\alpha_с \alpha_т T_\mu}$ и уравнение статической характеристики имеет вид

$$\omega = \frac{U_{з.с.}}{K_{о.с.}} - \frac{\alpha_с \alpha_т T_\mu}{\beta T_м} \times M.$$

Жесткость механической характеристики определяется постоянными T_μ и T_μ и их отношением. Если механическая часть привода имеет большой момент инерции и его $T_м$ велика, тогда требуемая жесткость и динамические показатели обеспечиваются при достаточно большом $K_{о.с.}$.

Изображение ошибки по управляющему воздействию имеет вид

$$\delta_{\omega_з}(p) = \frac{\omega_{оз}(p)}{1 - \frac{\beta_{дин.с}(p)}{\beta T_м p}} = \frac{\alpha_с \alpha_т T_\mu p\left(\alpha_т T_\mu p + 1\right)}{\alpha_с \alpha_т T_\mu \left(\alpha_т T_\mu p + 1\right) + 1},$$

при $M_с = 0$ и $\omega_{оз} = const \delta_\omega = 0$,

при $M_с = 0$ и $\omega_{оз} = \frac{E_о}{p}\delta_\omega = \alpha_с \alpha_т T_\mu E_о$.

Изображение ошибки по возмущающему воздействию определяется

$$\mathrm{E}_{\omega_в}(\mathrm{p}) = \frac{\alpha_с \alpha_т \mathrm{T}_\mu \left(\alpha_т \mathrm{T}_\mu \mathrm{p}+1\right)}{\beta \mathrm{T}_м \left[\alpha_с \alpha_т \mathrm{T}_\mu \left(\alpha_т \mathrm{T}_\mu \mathrm{p}+1\right)+1\right]},$$

при $\mathrm{M}_с = const\ \partial_м = \frac{\mathrm{M}_с}{\beta} \times \frac{\alpha_с \alpha_т \mathrm{T}_\mu}{\mathrm{T}_м} = \frac{\mathrm{M}_с}{\beta_{зам.с}}$.

Величина $\beta_{зам.с}$ представляет добротность РС по моменту.

$\beta_{зам.с} = \frac{1}{\alpha_с \alpha_т \mathrm{T}_\mu} = \frac{1}{\mathrm{K}_{о.с.}}$ следовательно $\mathrm{K}_{о.с.} = \alpha_с \alpha_т \mathrm{T}_\mu = \frac{1}{\beta_{зам.с}}$.

Если не задано $\beta_{зам.с}$, то рекомендуется принять $\beta_{зам.с} = 10 - 40$. Далее определяется $\mathrm{K}_у$ Согласно рис.6.

$К_у = \frac{R_{ос.с.}}{R_{з.с.}}$; $К_{о.с.} = К_{т.г.} \frac{R_{ос.с.}}{R_{о.с.} + R_{ос.с.}}$.

Коэффициент передачи тахогенератора $К_{т.г.}$ берется из справочных данных (В/рад(с)). Величина $R_{о.с.}$ берется равной или близкой сопротивлению нагрузки тахогенератора. Затем оправляется $R_{ос.с}$ и величина $R_{о.с.}$. Величина $R_{ос.с.}, R_{з.с.}, R_{о.с.}$ необходимо брать из установленного ряда номиналов резистора, а для обеспечения расчетного значения необходимо (в случае несовпадения расчетных значений с номиналом резисторов) дополнительно с постоянными резисторомпоследовательно устанавливать переменные резисторы согласно рис.8.

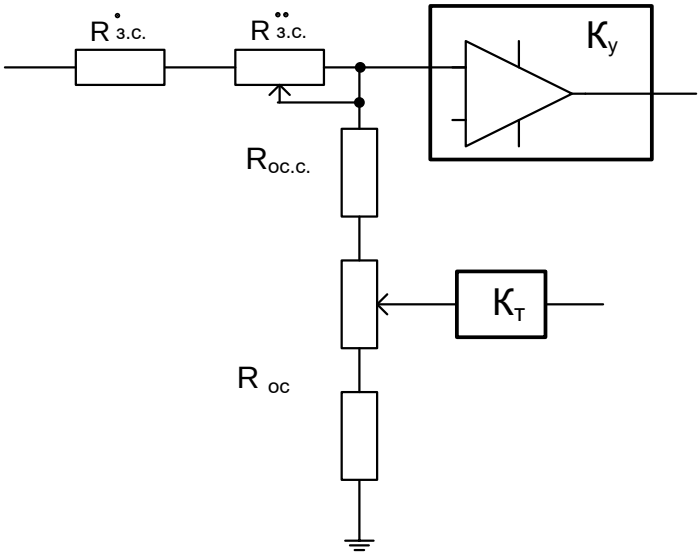


Рис.8. Регулятор скорости

Например, расчетное значение $R_{з.с.} = 4{,}9$ кОм. Выбираем $R_{з.с.} = R_{з.с.*} + R_{з.с.**}; R_{з.с.*} = const = 4{,}7$ кОМ, $R_{з.с.**} = var = 1$ кОм.

Таким образом, рассматриваемый электропривод обладает по управлению астатизмом первого порядка, а по нагрузке имеет статизм, определяемый $\Delta_{зам.с.}$. Точность регулирования скорости достигаемая при этом во многих случаях может быть ниже требуемой. Одним из возможных путей повышения точности регулирования является использования контура регулирования скорости на симметричный оптимум.

Для такой настройки контура регулирования скорости примем, что в схеме на рис.5 регулятор скорости является интегрально-пропорциональным и имеет передаточную функцию

$$W_{р.с.}=\frac{T_{к.с}p+1}{T_{и.с}p},$$

где $T_{к.с}$ и $T_{и.с}$ – компенсирующая постоянная времени и постоянная интегрирования регулятора скорости.

При этом передаточная функция разомкнутого контура скорости при $M_{с.}=0$примет вид

$$W_{раз.с.}=\frac{T_{к.с}p+1}{T_{и.с}p}\times\frac{\frac{C}{K_{о.т.}}}{\alpha_т T_μ p+1}\times\frac{1}{\beta T_м p}$$

Учитывая, что при настройке контура тока на технический оптимум некомпенсируемая инерционность для данного случая равна $T_{μс}=\alpha_т T_μ=2T_μ$ принимает вид

$$W_{раз.с.}=\frac{8T_μ p+1}{8T_μ p}\times\frac{\frac{1}{K_{о.с.}}}{4T_μ p(2T_μ p+1)}$$

Определим параметры ИП регулятора скорости:

$$\frac{1}{32T_μ^2 K_{о.с.}}=\frac{C}{K_{о.т.}\beta T_м T_{ис}};\ T_{кс}=8T_μ$$

Отсюда $T_{ис}=\frac{32T_μ^2 K_{о.с.}c}{K_{о.т.}\beta T_м}$

Динамическая жесткость механической характеристики в данном случае определяется соотношением

$$\beta_{дин.с}=-\frac{\beta T_м(8T_μ p+1)}{32T_μ^2 p(2T_μ p+1)}.$$

Полученное выражение динамической жесткости свидетельствует о том, что настройка на симметричный оптимум обеспечивает астатическое регулирование скорости ($p=0,\beta_{зам.с}=\infty$).

Ошибка по управляющему воздействию

$$\delta_{ω_З}(p)=\frac{ω_{оз}(p)}{1+W_{раз.с}}=\frac{ω_{оз}(p)32T_μ^2 p^2(2T_μ p+1)}{32T_μ^2 p^2(2T_μ p+1)+8T_μ p+1}.$$

Таким образом, астатическая двухконтурная система по управляющему воздействию обладает астатизмом второго порядка. Устранившаяся ошибка при линейном нарастании задания в этой системе равна нулю, а при нарастании задающего сигнала с постоянной второй производной $\left(\frac{d^2ω}{dt^2}\right)_{макс}=ρ_{макс}=const$ ограничена значением $\delta_{ω_{зм}}(2)=16T_μ^2 ρ_{макс}$.

Это дает основания назвать систему с ИП регулятором скорости двукратно-интегрирующей. Поэтому настройка на симметричный опти-

мум двухконтурной системы представляет интерес во всех случаях, когда важно иметь минимальную динамическую ошибку по управлению.

Передаточная функция замкнутого контура скорости при этом имеет вид

$$W_{зам.с.} = \frac{8T_{\mu}p+1/K_{о.с.}}{32T_{\mu}^2p^2(2T_{\mu}p+1)+8T_{\mu}p+1}.$$

Наличие формирующего звена в числителе определяет увеличение перерегулирования при скачке задания в сравнении с настройкой на технический оптимум, а при линейном нарастании задающего сигнала процесс ликвидации ошибки сопровождается перерегулированием тока до 56% от установившегося значения. В системе с ИП регулятором скорости можно получить процессы при изменениях управляющего воздействия, соответствующие настройке на технический оптимум, если на задающий вход регулятора включить фильтр с передаточной функцией

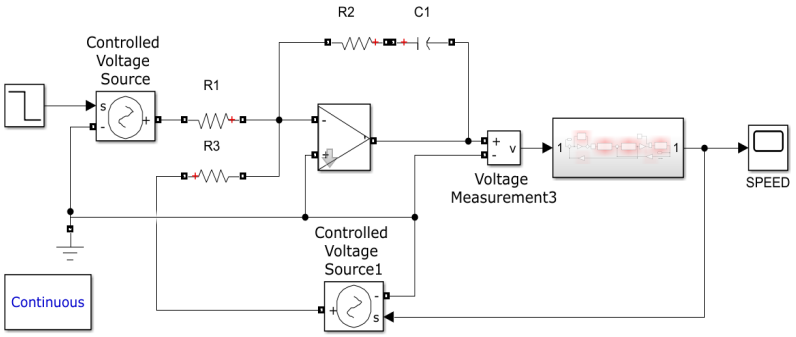
$W_{з.ф.с.} = \frac{1}{8T_{\mu}p+1}$

Однако при этом установившаяся ошибка при линейном нарастании задания уже не равна нулю, а имеет конечное значение

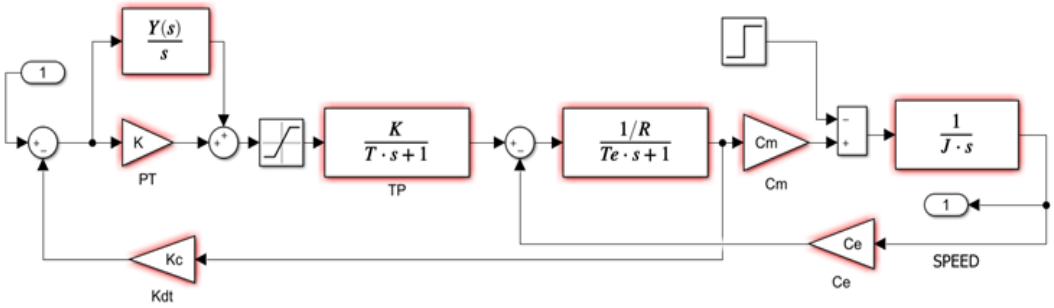
$\delta_{\omega_{зм}(1)}(0) = 8T_{\mu}\varepsilon_{о}$.

Порядок выполнения работы

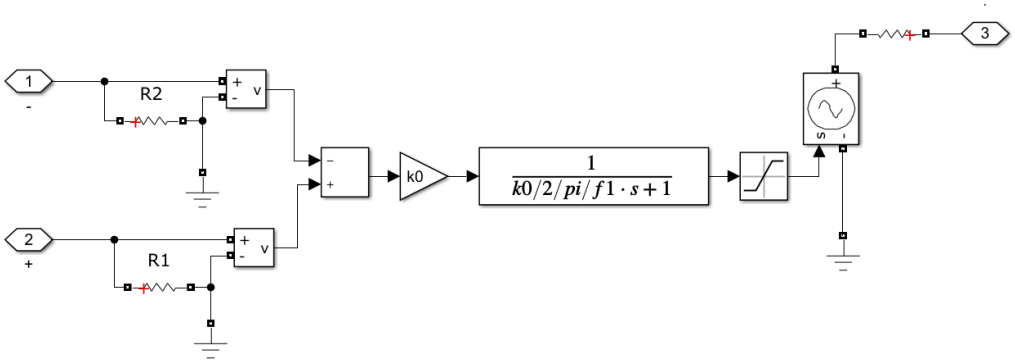
1. Построить в Matlab модель привода согласно рисунка 9а. используя пакеты Simulink и SimPowerSystem.
2. Регулятор скорости выполнить с использованием пакета SimPowerSystem остальные устройства реализовать в Simulink (рис.9б).
3. Усилитель регулятора скорости реализовать по схеме рис.9в. Параметры выбрать из справочника по аналоговым микросхемам (входное сопротивление, коэффициент усиления, граничную частоту).
4. Подставить в собранную модель значения параметров двигателя из таблицы 1.
5. Рассчитать настройку регуляторов на технический/модульный оптимум по приведенной методике.
6. Подставить расчетные значения в модель.
7. Выполнить моделирование.



а



б



в

Рис.9. Модель привода: а – модель, б – подсистема, в – операционный усилитель

Таблица 1

Варианты заданий

Параметры									
№	Тип	$P_н$, Вт	$U_н$, В	$I_н$, А	$J_{рд}$, кг·м^2	$R_я$, Ом	$L_я$, Гн	$M_н$, Н·м	D, Н/кг$^{1/2}$
01	UGSM-02B	44	28	2,6	4,00E-06	3,4	1,30E-03	0,211	105
02	UGI-03M	44	24,4	3,15	2,35E-05	1,59	1,27E-03	0,240	50
03	TS902N2-E6	60	30,8	3,0	2,84E-05	1,7	2,72E-03	0,157	30
04	TS668N4-E6	80	31,3	4,1	3,92E-05	1,3	1,69E-03	0,196	31

№	Тип	$P_н$, Вт	$U_н$, В	$I_н$, А	$J_{рд}$, кг·м^2	$R_я$, Ом	$L_я$, Гн	$M_н$, Н·м	D, Н/кг$^{1/2}$
05	H1008	85	28	5,05	4,90E-06	0,7	1,19E-04	0,177	80
06	H1009	90	28	5,15	3,50E-06	0,55	7,70E-05	0,265	142
07	UGI-10M	100	64	2,5	6,00E-04	5,0	3,00E-03	1,0	40
08	UGSM-12B	114	28	6,2	4,65E-06	0,67	1,07E-04	0,36	167
09	UGSM-03A	120	28	6,4	3,30E-06	0,68	1,02E-04	0,28	154
10	H1420	150	28	7,65	7,65E-05	0,7	2,10E-04	1,32	151
11	UGI-40M	160	44	5,0	1,60E-03	1,05	1,30E-03	1,53	38
12	UGM-06	185	40	6,0	5,70E-05	0,84	9,00E-04	0,59	78
13	TS906N2-E13	200	43	6,6	2,34E-04	1,05	1,47E-03	0,638	42
14	UGI-40L	250	60	6,0	1,60E-03	1,3	2,00E-03	2,4	60
15	ПЯ-250	250	36	12	3,50E-04	0,24	5,00E-05	0,8	43
16	UGM-13	400	68	8,0	1,40E-04	1,03	1,60E-03	1,3	110
17	ДПУ-200	550	92	7,4	7,40E-04	1,53	5,00E-04	1,7	62
18	UGM-25	771	70	14	1,80E-04	0,47	6,00E-04	2,0	149
19	ДПУ-240	1100	122	11	1,88E-03	0,53	2,00E-04	3,5	80

Отчёт о выполнении работы

Отчёт о выполнении лабораторной работы выполняется по стандарту и должен содержать следующие результаты:

Исходные данные.

Результаты расчета настроек в развернутом виде.

Структурная схема (копия экрана из Matlab) модели с указанием установленных параметров.

Результаты моделирования. Представить линейный анализ.

Распечатать графики и представить в отчете (переходный процесс, АЧХ и ФЧХ, импульсную характеристику и т.д.).

Привести модель в распечатанном виде (четкая копия экрана в читаемом формате).

Сформулировать заключение.

Практическое занятие № 7. Создание нечеткого контроллера

Цель: разработать нечеткий контроллер и исследовать его работу в среде MatLab.

Краткие сведения из теории

Технической диагностикой называется наука о распознавании состояния технической системы.

Техническая диагностика изучает методы получения и оценки диагностической информации, диагностические модели и алгоритмы принятия решений. Целью технической диагностики является повышение надежности и ресурса технических систем.

Техническая диагностика благодаря раннему обнаружению дефектов и неисправностей позволяет устранить отказы электромеханической системы в процессе технического обслуживания, что повышает надежность и эффективность эксплуатации, а также дает возможность эксплуатации технических систем ответственного назначения по состоянию. Отказы, неисправности, поломки, сбои, ошибки - неизбежные факторы, дестабилизирующие процесс нормального функционирования объекта и системы управления.

Теоретическим фундаментом для решения основной задачи технической диагностики следует считать общую теорию распознавания образцов. Алгоритмы распознавания в технической диагностике частично основываются на диагностических моделях, устанавливающих связь между состояниями технической системы и их отображениями в пространстве диагностических сигналов. Важной частью проблемы распознавания являются правила принятия решений (решающие правила).

Порядок выполнения работы

1. Воспользоваться моделью привода построенного в лабораторной работе №6.
2. Собрать необходимые данные в контрольных точках. Данные приведены в таблице1.
3. Сформировать функции принадлежности для входных и выходных данных по полученным данным.
4. Выбрать метод обобщения и метод приведения к четкости.
5. Сформировать базу знаний.
6. Выполнить моделирование созданного нечеткого контроллера.
7. Импортировать полученный контроллер в модель привода и выполнить моделирование.

Пример выполнения.

Запускаем Matlab.

Открываем созданную в лабораторной работе №6 модель привода

Добавляем в схему нечеткий контроллер и дисплей из библиотеки блоков Simulink

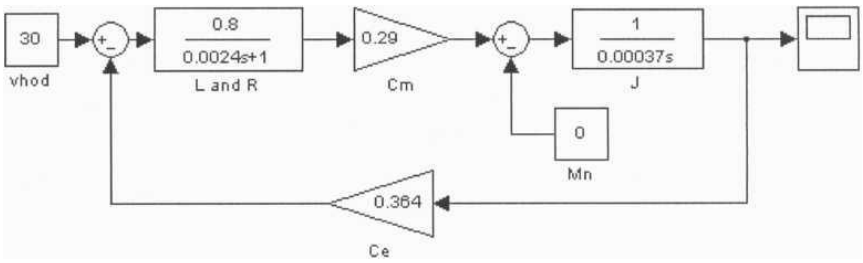


Рис.1. Модель привода

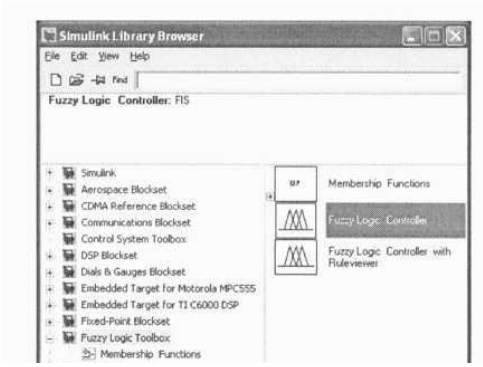


Рис.2. Библиотека Simulink

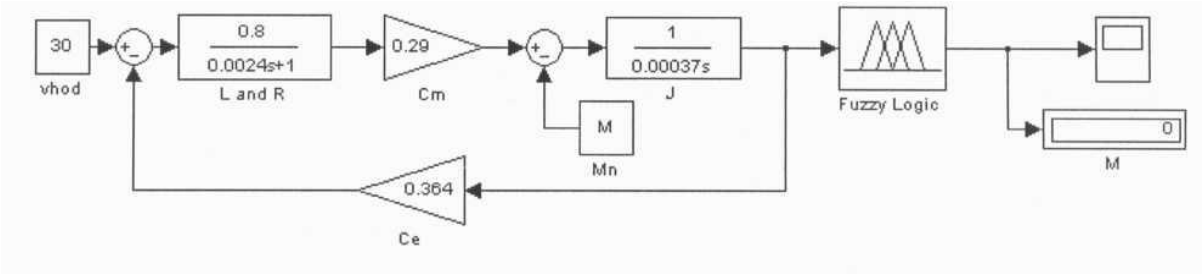


Рис. 3. Модель привода с нечетким контроллером

Введём команду Fuzzy в командную строку MatLab и откроем приложение *FuzzyLogicToolbox (*Рис.4). Открываем в нём созданный файл «pravila».

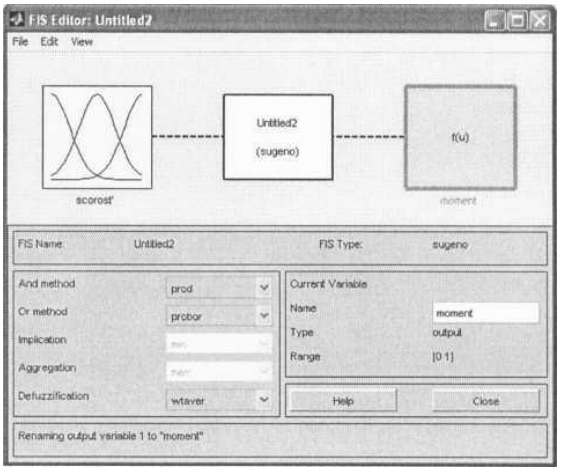


Рис.4. Окно *FuzzyLogicToolbox*

Экспортируем файл в рабочую область среды MATLAB, для этого выбираем в меню *File* подменю *Export* команду *To Workspace...*
Возвращаемся к модели. Дважды щелкнув по пиктограмме нечеткого контроллера, откроем окно, в котором вводим название созданной нечеткой системы знаний, экспортированной в рабочую область.

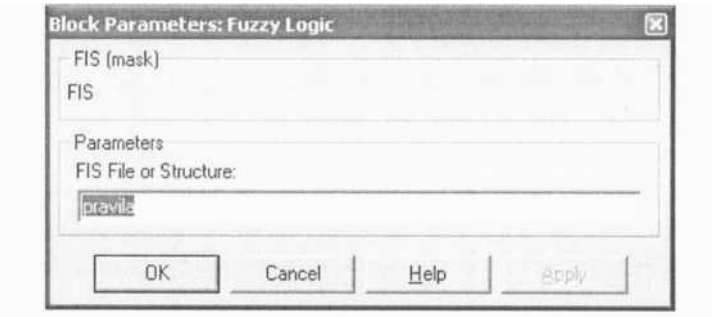


Рис. 5. Окно настроек нечёткого контроллера.

Запускаем систему. Изменяем значения момента и наблюдаем, как изменяются значения на дисплее и осциллографе.

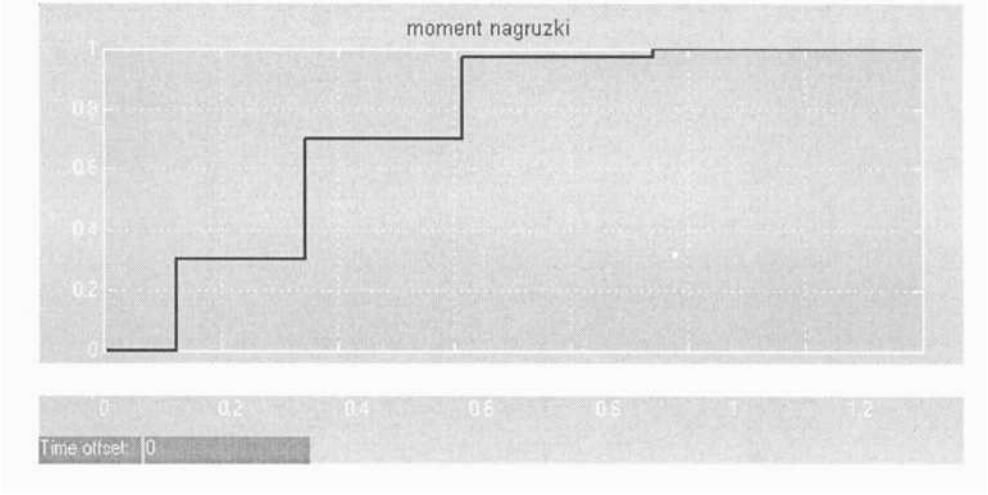


Рис.6. Результат работы контроллера

Отчёт о выполнении работы

Отчёт о выполнении лабораторной работы должен быть выполнен в соответствии с требованиями и содержать следующие результаты:

Терм множества в виде таблиц исходных данных полученных при анализе работы привода.

Графики функций принадлежности входных и выходных переменных с указанием диапазонов.

Полную базу правил с подробными комментариями.

Результаты моделирования нечеткого контроллера.
Структуру модели с встроенным нечетким контроллером.

Обобщенные результаты моделирования представить в виде графика.

Допускается четкая копия экрана.

Таблица 1

Исходные данные

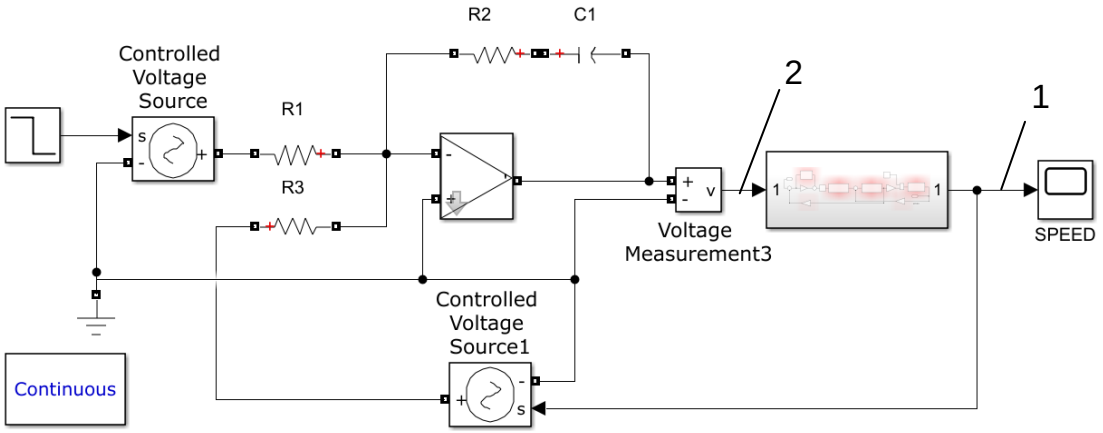
Узел	Изменяемый параметр		Диапазон	Точки съема информации
1. Регулятор скорости	1	R1	±50% от номинала	Выход 1 Выход 2
	2	R2		
	3	R3($К_{ос}$)		
	4	C	±50%	
2. Регулятор тока	1	К	±50%	Выходы 3, 1
	2	Т	±50%	Выходы 3, 3а, 1
	3	$К_{дт}$	±50%	Выходы 3б, 1
3. Силовой преобразователь	1	К	±50%	Выходы 4, 1
	2	Т	±50%	
4. Двигатель	1	$R_я$	±50%	Выходы 5, 5а, 1
	2	$Т_я$	±50%	
	3	$J_я$	±100%	
	4	М	0…$М_н$	Выходы 4, 5, 1

Число векторов не менее десяти.
Точность не менее ±0.1.
Рекомендуемый вид функции активации: треугольная, трапецеидальная, гауссова, обобщенная колоколообразная, сигмоидальная, Z-функция, S-функция.

Таблица 2

Варианты заданий

Номер вариата	Номер задания	Номер вариата	Номер задания	Номер вариата	Номер задания	Номер вариата	Номер задания
1.	1.1	5	2.1	9	3.2	13	4.4
2.	1.2	6	2.2	10	4.1		
3.	1.3	7	2.3	11	4.2		
4	1.4	8	3.1	12	4.3		



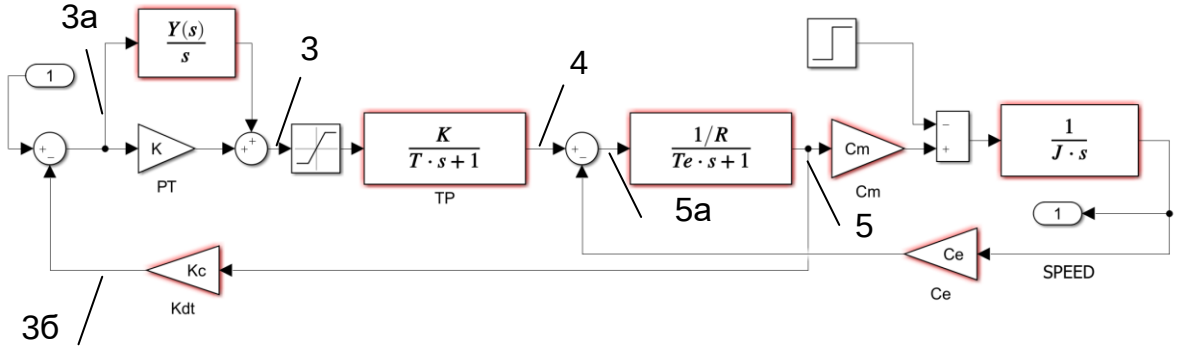


Рис.7. Контрольные точки

Практическое занятие № 8. Создание нейронной сети для классификации

Цель: разработать нейро контроллер и исследовать его работу в среде MatLab.

Порядок выполнения работы

1. Воспользоваться моделью привода построенного в лабораторной работе №6.
2. Собрать необходимые данные в контрольных точках. Данные приведены в таблице1 и на рис.7 в лабораторной работе №7.
3. Сформировать данные по классам и целевые функции для классов.
4. Выбрать тип нейронной сети.
5. Выполнить ручной расчет модели нейронной сети.
6. Выполнить моделирование созданной нейросети.

7. Создать тестовый вектор и проверить работоспособность сети нейронов.
8. Импортировать полученный контроллер в модель привода и выполнить моделирование.

Отчёт о выполнении работы

Отчёт о выполнении лабораторной работы должен быть выполнен в соответствии с требованиями и содержать следующие результаты:

1. Обучающие множества в виде таблиц исходных данных полученных при анализе работы привода.
2. Целевые функции.
3. Структуру выбранной нейросети.
4. Результаты ручного расчета весовых коэффициентов и смещений.
5. Полную структуру нейронной сети в виде рисунков (по слоям).
6. Привести схему модели с встроенным нейроконтроллером.
7. Результаты моделирования нейронного контроллера в схеме со встроенным контроллером.
8. Обобщить результаты по точности моделирования и представить в виде графика.

   Допускается четкая копия экрана из Matlab.

Таблица 1

Исходные данные

Узел	Изменяемый параметр		Диапазон	Точки съема информации
1. Регулятор скорости	1	R1	±50% от номинала	Выход 1 Выход 2
	2	R2		
	3	R3($К_{ос}$)		
	4	C	±50%	
2.Регулятор тока	1	К	±50%	Выходы 3, 1
	2	Т	±50%	Выходы 3, 3а, 1
	3	$К_{дт}$	±50%	Выходы 3б, 1
3.Силовой преобразователь	1	К	±50%	Выходы 4, 1
	2	Т	±50%	
4. Двигатель	1	$R_я$	±50%	Выходы 5, 5а, 1
	2	$Т_я$	±50%	
	3	$J_я$	±100%	
	4	М	0…$М_н$	Выходы 4, 5, 1

Число векторов не менее десяти.
Число элементов в векторе не более 100.
Число классов не менее 3
Точность не менее ±0.1.

Таблица 2

Варианты заданий

Номер вариа-та	Номер зада-ния	Номер вариа-та	Номер зада-ния	Номер вариа-та	Номер зада-ния	Номер вариа-та	Номер зада-ния
1.	1.1	5	2.1	9	3.2	13	4.4
2.	1.2	6	2.2	10	4.1		
3.	1.3	7	2.3	11	4.2		
4	1.4	8	3.1	12	4.3		

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Искусственный интеллект. Его истоки и проблемы.
2. Нейрофизиологические данные об обработке информации в биологических системах.
3. Искусственный нейрон. Идея и техническая реализация.
4. Модели нейронов. Типичные виды функций активации нейрона.
5. Многослойный персептрон.
6. Многослойные однонаправленные сети. Алгоритм обратного распространения ошибки.
7. Вывод конкретных формул алгоритма обратного распространения ошибки для двухслойных сетей с малым числом нейронов (2-3).
8. Градиентные методы. Алгоритм наискорейшего спуска. Недостатки метода. Метод моментов.
9. Радиальные нейронные сети. Обучение. Область применения.
10. Рекуррентные сети. Ассоциативная сеть Хопфилда. Обучение. Распознавание образов.
11. Сеть встречного распространения.
12. Обучение слоя Кохонена. Решение задач кластеризации.
13. Статистический подход к обучению нейронной сети. Машина Больцмана и ее модификации.
14. Применение нейронных сетей. Сбор данных для нейронных сетей.
15. Задача регрессии и прогнозирования временных рядов.
16. Основные характеристики пакета MATLAB. Простейшие вычисления. Работа с массивами. Графики функций. Сессия. М-файлы. Mat-файлы.
17. Нейронные сети в пакете MATLAB.

ЗАКЛЮЧЕНИЕ

Нейронные сети и нечеткая логика – удобный и эффективный инструмент в создании систем управления и диагностики. Использование такого инструмента сопряжено с решением нескольких принципиальных задач. Первая из них – работа с данными, используемыми для создания баз данных и знаний. Получение данных и извлечение знаний связано с исследованием проектируемой системы, основанным на использовании моделей. Чем точнее построена модель, тем точнее будут полученные результаты. Вторая задача – выбрать либо нейроконтроллер, либо нечеткий контроллер. При этом необходимо установить, можно ли решить конкретную задачу классическими методами, и только после анализа принимать решение об использовании нейронных сетей или нечеткой логики. Особенно эффективно их использование в том случае, если отсутствует аналитическое описание процесса, а есть некоторые экспериментальные данные, полученные в результате исследований.

Готовых рецептов в конструировании контроллеров практически нет, и разработчик сталкивается с задачей, охватывающей огромный пласт знаний, в котором интуиция играет существенную роль. Интуитивно, например, выбираются функции принадлежности для НК или метод обучения ИНС. Для упрощения понимания сложных вопросов в пособии рассматриваются решения, которые помогают на первых порах создать простые устройства. Каждый пункт создания описан последовательно, шаг за шагом. Детально описать процесс создания ИНС или НК – трудоемкая задача, которая требует большого объема текстовой информации. И даже в случае детального описания

все равно останутся без внимания какие-либо знания, потому как любое новое знание требует описания. Частично это компенсируется моделированием готовых решений. Даже в этом случае получают первичные навыки работы с ИНС и НК, которые позволят в дальнейшем создавать новые проекты. Недостаток информации компенсируется использованием дополнительной литературы.

Моделирование в MATLAB можно выполнять либо с помощью построения моделей средствами специальных приложений, либо использования языка программирования программной среды. Оба подхода приведены в тексте пособия.

Заметим, что развитие методов искусственного интеллекта входит в решение различных задач, в том числе и в промышленности. Многие задачи с использованием ИИ дают возможность построить системы, обладающие «умом» и способные решать технические задачи более эффективно, нежели если бы это выполнял человек.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Нейронные сети: полный курс, 2е издание. : Пер. с англ. - М. Издательский дом "Вильямс", 2006. - 1104 с.
2. В.П. Дьяконов MATLAB 6.5 SPl/7 + Simulink 5/6@. Основы применения. Серия «Библиотека профессионала». М.: СОЛОН-Пресс, 2005.- 800 с.
3. Половко А. М., Бутусов П. Н. MATLAB для студента. - СПб.: БХВ-Петербург, 2005. -320 с.
4. С.Д.Штовба. Введение в теорию нечетких множеств и нечеткую логику. [Электронный ресурс] http://matlab.exponenta.ru/ fuzzylogic/book2/index.php 26.02.2012
5. Michael Wooldridge, An Introduction to MultiAgent Systems, John Wiley & Sons Ltd, 2002, paperback, 366 p.
6. Carl Hewitt and Jeff Inman. DAI Betwixt and Between: From «Intelligent Agents» to Open Systems Science IEEE Transactions on Systems, Man, and Cybernetics. Nov./Dec. 1991.
7. Gerhard Weiss, ed. by, Multiagent Systems, A Modern Approach to Distributed Artificial Intelligence, MIT Press, 1999, ISBN 0-262-23203-0.
8. Jacques Ferber, Multi-Agent Systems: An Introduction to Artificial Intelligence, Addison-Wesley, 1999.
9. Fraser, Alex Simulation of genetic systems by automatic digital computers. I. Introduction (англ.)//Aust. J. Biol. Sci.: journal. - 1957. - Vol. 10. - p. 484-491.
10. Fraser, Alex ; Donald Burnell. Computer Models in Genetics (англ.). - New York: McGraw-Hill Education, 1970.
11. Crosby, Jack L. Computer Simulation in Genetics (англ.). - London: John Wiley & Sons, 1973- ISBN 0-471-18880-8.
12. Fogel, David B. (editor). Evolutionary Computation: The Fossil Record (англ.). — New York: Institute of Electrical and Electronics Engineers, 1998.
13. 17 Barricelli, Nils Aall. Numerical testing of evolution theories. Part II. Preliminary tests of performance, symbiogenesis and terrestrial life (англ.) // Acta Biotheoretica : journal. - 1963. - No. 16. - P. 99-126.

14. Щербаков С. В. Методика диагностирования сложных технических систем с применением нечетко-множественного подхода // Промышленные АСУ и контроллеры. 2008. №11. с.28-31.
15. Брадаи Р., Шетат Б., Ладыгин А. Н. Повышение эффективности регулятора скорости частотно-регулируемого асинхронного электропривода средствами искусственного интеллекта // Электротехника. 2008. №12. с.41-50.
16. Усков А. А. Алгоритм синтеза нечётких логических регуляторов на основе самоорганизации // Приборы и системы. Управление, контроль, диагностика. 2004. №8. с.1-3
17. Усков А. А., Киселёв Е. В. Системы управления с нечеткими супервизорными ПИД регуляторами // Приборы и системы. Управление, контроль, диагностика. 2005. №9. с.31-33
18. Усков А. А. Эмпирический принцип синтеза нечётких логических регуляторов // Приборы и системы. Управление, контроль, диагностика. 2004. №1. с.16-18.
19. Амосов О. С. Адаптивная нелинейная фильтрация случайных последовательностей на основе нейронных сетей и нечёткой логики // Приборы и системы. Управление, контроль, диагностика. 2004. №5. с.48-53.
20. Усков А. А. Принципы построения систем управления с нечёткой логикой // Приборы и системы. Управление, контроль, диагностика. 2004. №6. с.7-11.
21. Мещеряков В. Н., Карантаев В. Г. Разработка Fuzzy-регулятора для электропривода станка с технологической обратной связью // Приборы и системы. Управление, контроль, диагностика. 2004. №11. с.24-26.
22. Соколов Ю. А. Комбинированное нечеткое управление процессом нагрева с дискретной подачей // Приборы и системы. Управление, контроль, диагностика. 2009. №5. с.1-4.
23. Толстов М. В., Усков А. А. Нечеткий системный стабилизатор синхронного генератора электроэнергетических систем // Приборы и системы. Управление, контроль, диагностика. 2007. №11. с.23-26.
24. Солдатов В.В., Липа О.А., Сироткин И.И. Синтез нечетких ПИД регуляторов с учетом технологических и технических ограничений. //Приборы и системы. Управление, контроль, диагностика. 2009. №8. с.12-15.

25. Сидельников С.И. Применение модели объекта для прогнозирования его состояния по результатам управляющих решений. // Приборы и системы. Управление, контроль, диагностика. 2009. № 3. с. 53-56.
26. Вент Д.П., Сидельников С.И., Родин С.И. Нечеткое регулирование нелинейных объектов // Приборы и системы. Управление, контроль, диагностика. 2006. № 7. С. 12-14.
27. Леденёва Т. М., Татаркин Д. С. Особенности проектирования систем нечеткого логического вывода // Информационные технологии. 2007. №7.- с.12-19.
28. Соколов А. М. Методы и алгоритмы нечеткого моделирования механических систем // Информационные технологии. 2007. №3 с.13-20.
29. Колдаев А.И. Моделирование интеллектуальной системы поддержки принятия решений при управлении технологическим процессом. // Информационные технологии. 2009. № 3. с. 42-46.
30. Кудинов Ю.И., Иванченко К. С., Кудинов И. Ю. Разработка и идентификация нечетких моделей прогнозирования качества // Мехатроника, автоматизация, управление. 2007. №12. с.12-15.
31. Глушков С. В. Автоматическое управление курсом судна с использованием регулятора на нечеткой логике // Мехатроника, автоматизация, управление. 2007. № 12. с.32-36.
32. Поляхов Н. Д., Приходько И. А., Карачев А. А., Вейнмейстер А. В., Беспалов А. В. Интеллектуальное управление в технических системах // Мехатроника, автоматизация, управление. 2007. №10. с.11-15.

ПРИЛОЖЕНИЯ

Приложение 1

СОЗДАНИЕ И МОДЕЛИРОВАНИЕ НЕЙРОСЕТИ В MATLAB

Чтобы запустить «NNTool», необходимо выполнить одноимённую команду в командном окне MATLAB:

>> nntool.

После этого появится главное окно NNTool, именуемое "Окном управления сетями и данными" (Network/Data Manager) (рис.1).

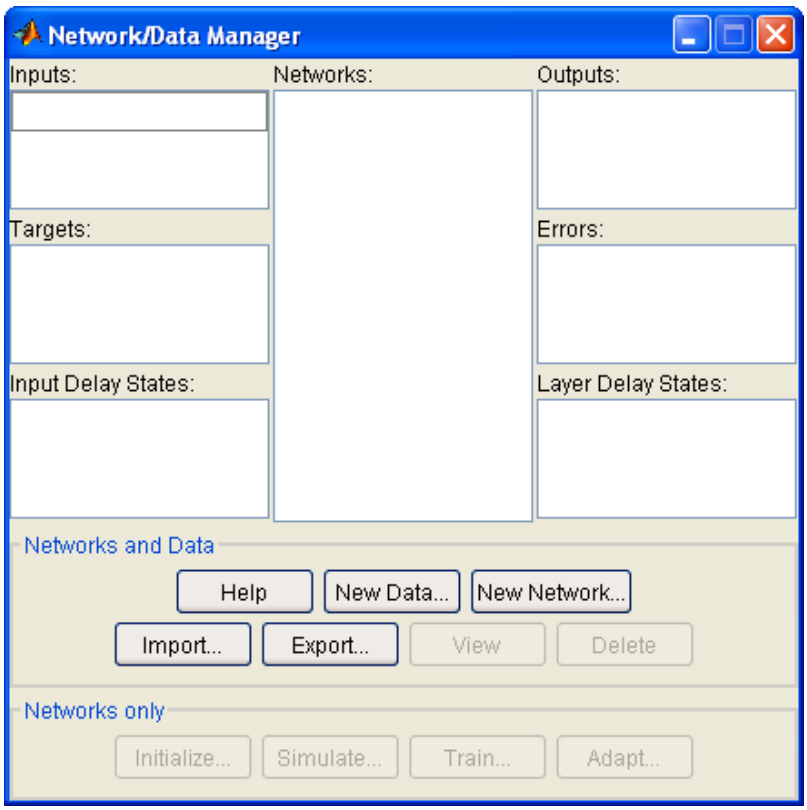


Рис.1. Главное окно NNTool

Панель "Сети и данные" (Networks and Data) имеет функциональные клавиши со следующими назначениями:

- Помощь (Help)- краткое описание управляющих элементов данного окна;
- Новые данные (New Data…)- вызов окна, позволяющего создавать новые наборы данных;

- Новая сеть (New Network…)- вызов окна создания новой сети;
- Импорт (Import…)- импорт данных из рабочего пространства MATLAB в пространство переменных NNTool;
- Экспорт (Export…)- экспорт данных из пространства переменных NNTool в рабочее пространство MATLAB;
- Вид (View)- графическое отображение архитектуры выбранной сети;
- Удалить (Delete)- удаление выбранного объекта.

На панели "Только сети" (Networks only) расположены клавиши для работы исключительно с сетями. При выборе указателем мыши объекта любого другого типа, эти кнопки становятся неактивными.

При работе с NNTool важно помнить, что клавиши View, Delete, Initialize, Simulate, Train и Adapt (изображены на рис.1 как неактивные) действуют применительно к тому объекту, который отмечен в данный момент выделением. Если такого объекта нет, либо над выделенным объектом невозможно произвести указанное действие, соответствующая клавиша неактивна.

Рассмотрим создание нейронной сети с помощью NNTool на примере.

Пусть требуется создать нейронную сеть, выполняющую логическую функцию "И".

Создание сети

Выберем сеть, состоящую из одного персептрона с двумя входами. В процессе обучения сети на её входы подаются входные данные и производится сопоставление значения, полученного на выходе, с целевым (желаемым). На основании результата сравнения (отклонения полученного значения от желаемого) вычисляются величины изменения весов и смещения, уменьшающие это отклонение.

Итак, перед созданием сети необходимо заготовить набор обучающих и целевых данных. Составим таблицу истинности для логической функции "И", где P1 и P2 - входы, а A - желаемый выход (табл.1).

Таблица 1

Таблица истинности логической функции "И"

P1	P2	A
0	0	0
0	1	0
1	0	0
1	1	1

Чтобы задать матрицу, состоящую из четырёх векторов-строк, как входную, воспользуемся кнопкой New Data. В появившемся окне следует произвести изменения, показанные на рис.2, и нажать клавишу "Создать" (Create).

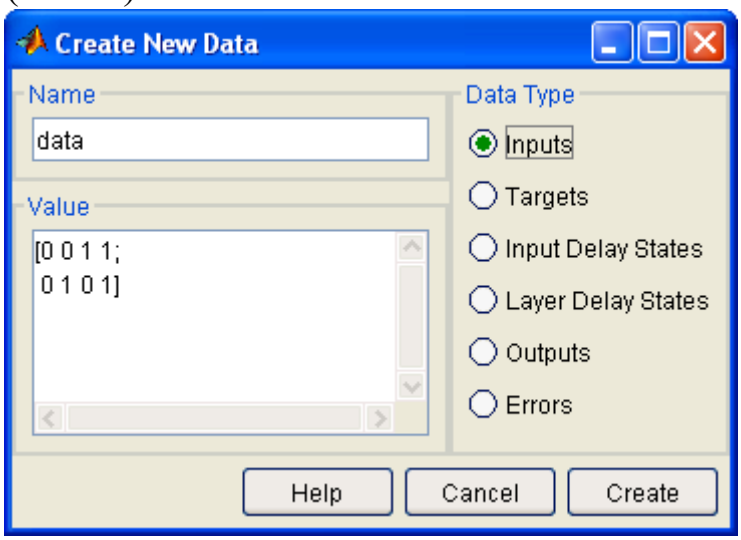


Рис.2. Задание входных векторов

После этого в окне управления появится вектор data в разделе Inputs. Вектор целей задаётся схожим образом (рис.3).

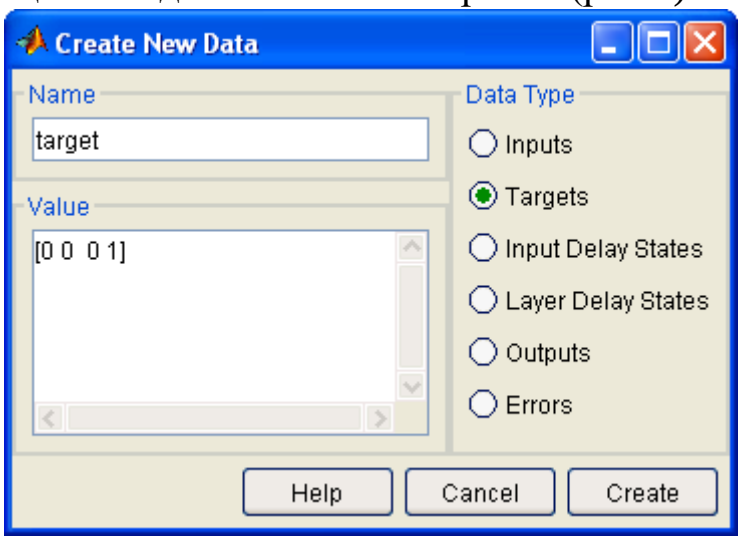


Рис.3. Задание целевого вектора

После нажатия на Create в разделе Targets появится вектор target.

Данные в поле "Значение" (Value) могут быть представлены любым понятным MATLAB выражением. К примеру, предыдущее определение вектора целей можно эквивалентно заменить строкой вида

bitand([0 0 1 1], [0 1 0 1]).

Теперь следует приступить к созданию нейронной сети. Выбираем кнопку New Network и заполняем форму, как показано на рис.4.

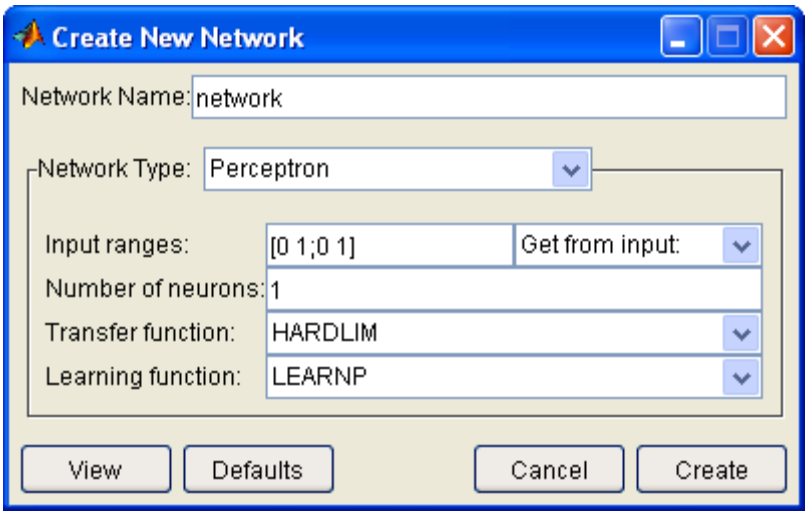


Рис.4. Окно "Создание сети"

При этом поля несут следующие смысловые нагрузки:

- Имя сети (Network Name) - это имя объекта создаваемой сети.
- Тип сети (Network Type)- определяеттип сети и в контексте выбранного типа представляет для ввода различные параметры в части окна, расположенной ниже этого пункта. Таким образом, для разных типов сетей окно изменяет своё содержание.
- Входные диапазоны (Input ranges)- матрица с числом строк, равным числу входов сети. Каждая строка представляет собой вектор с двумя элементами: первый - минимальное значение сигнала, которое будет подано на соответствующий вход сети при обучении, второй - максимальное. Для упрощения ввода этих значений предусмотрен выпадающий список "Получить из входа" (Get from input), позволяющий автоматически сформировать необходимые данные, указав имя входной переменной.

- Количество нейронов (Number of neurons)- число нейронов в слое.
- Передаточная функция (Transfer function)- в этом пункте выбирается передаточная функция (функция активации) нейронов.
- Функция обучения (Learning function)- функция, отвечающая за обновление весов и смещений сети в процессе обучения.

С помощью клавиши "Вид" (View) можно посмотреть архитектуру создаваемой сети (рис.5). Так, мы имеем возможность удостовериться, все ли действия были произведены верно. На рис.5 изображена персептронная сеть с выходным блоком, реализующим передаточную функцию с жёстким ограничением. Количество нейронов в слое равно одному, что символически отображается размерностью вектора-столбца на выходе слоя и указывается числом непосредственно под блоком передаточной функции. Рассматриваемая сеть имеет два входа, так как размерность входного вектора-столбца равна двум.

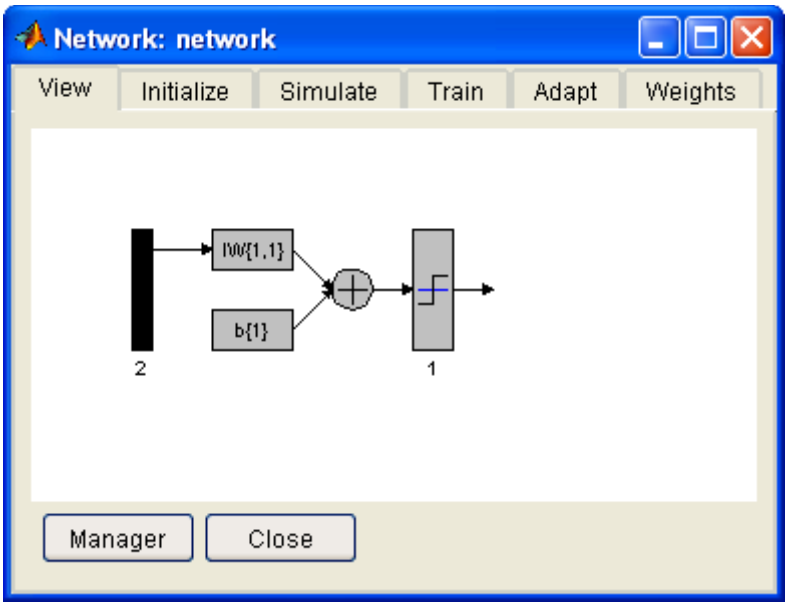


Рис.5. Предварительный просмотр создаваемой сети

Итак, структура сети соответствует нашему заданию. Теперь можно закрыть окно предварительного просмотра, нажав клавишу "Закрыть" (Close), и подтвердить намерение создать сеть, нажав "Создать" (Create) в окне создания сети.

В результате проделанных операций в разделе "Сети" (Networks) главного окна NNTool появится объект с именем network.

Обучение

Наша цель - построить нейронную сеть, которая выполняет функцию логического "И". Очевидно, нельзя рассчитывать на то, что сразу после этапа создания сети последняя будет обеспечивать правильный результат (правильное соотношение "вход/выход"). Для достижения цели сеть необходимо должным образом обучить, то есть подобрать подходящие значения параметров. В MATLAB реализовано большинство известных алгоритмов обучения нейронных сетей, среди которых представлено два для персептронных сетей рассматриваемого вида. Создавая сеть, мы указали LEARNP в качестве функции, реализующей алгоритм обучения (рис.4).

Вернёмся в главное окно NNTool. На данном этапе интерес представляет нижняя панель "Только сети" (Networks only). Нажатие любой из клавиш на этой панели вызовет окно, на множестве вкладок которого представлены параметры сети, необходимые для её обучения и прогона, а также отражающие текущее состояние сети.

Отметив указателем мыши объект сети network, вызовем окно управления сетью нажатием кнопки Train. Перед нами возникнет вкладка "Train" окна свойств сети, содержащая, в свою очередь, ещё одну панель вкладок (рис.6). Их главное назначение - управление процессом обучения. На вкладке "Информация обучения" (Training info) требуется указать набор обучающих данных в поле "Входы" (Inputs) и набор целевых данных в поле "Цели" (Targets). Поля "Выходы" (Outputs) и "Ошибки" (Errors) NNTool заполняет автоматически. При этом результаты обучения, к которым относятся выходы и ошибки, будут сохраняться в переменных с указанными именами.

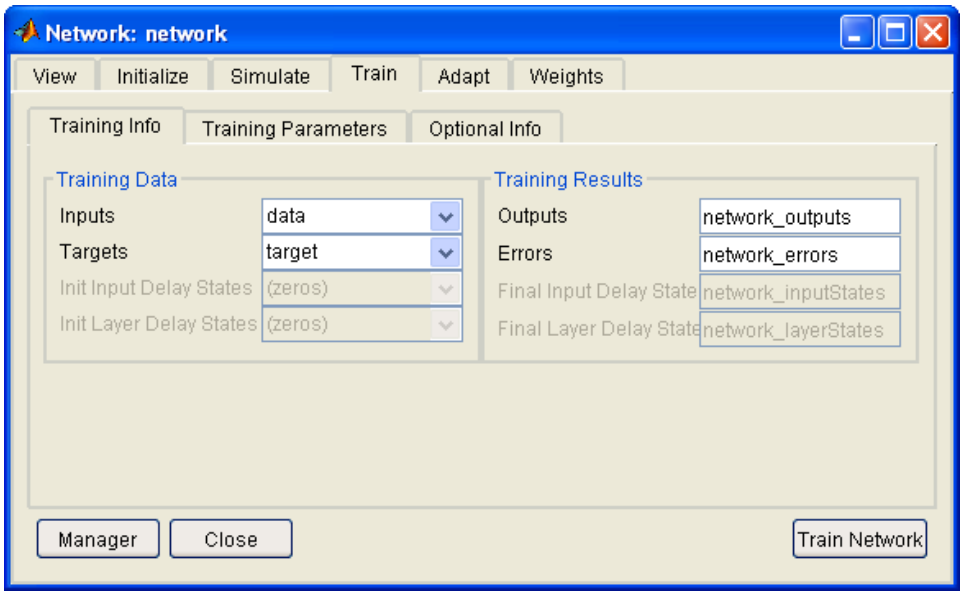


Рис.6. Окно параметров сети, открытое на вкладке "обучение" (Train)

Завершить процесс обучения можно, руководствуясь разными критериями. Возможны ситуации, когда предпочтительно остановить обучение, полагая достаточным некоторый интервал времени. С другой стороны, объективным критерием является уровень ошибки.

На вкладке "Параметры обучения" (Training parameters) для нашей сети (рис.7) можно установить следующие поля:

Количество эпох (epochs)- определяет число эпох (интервал времени), попрошествии которых обучение будет прекращено.

Эпохой называют однократное представление всех обучающих входных данных на входы сети.

Достижение цели или попадание (goal)- здесь задаётся абсолютная величина функции ошибки, при которой цель будет считаться достигнутой.

Период обновления (show)- период обновления графика кривой обучения, выраженный числом эпох.

Время обучения (time)- по истечении указанного здесь временного интервала, выраженного в секундах, обучение прекращается.

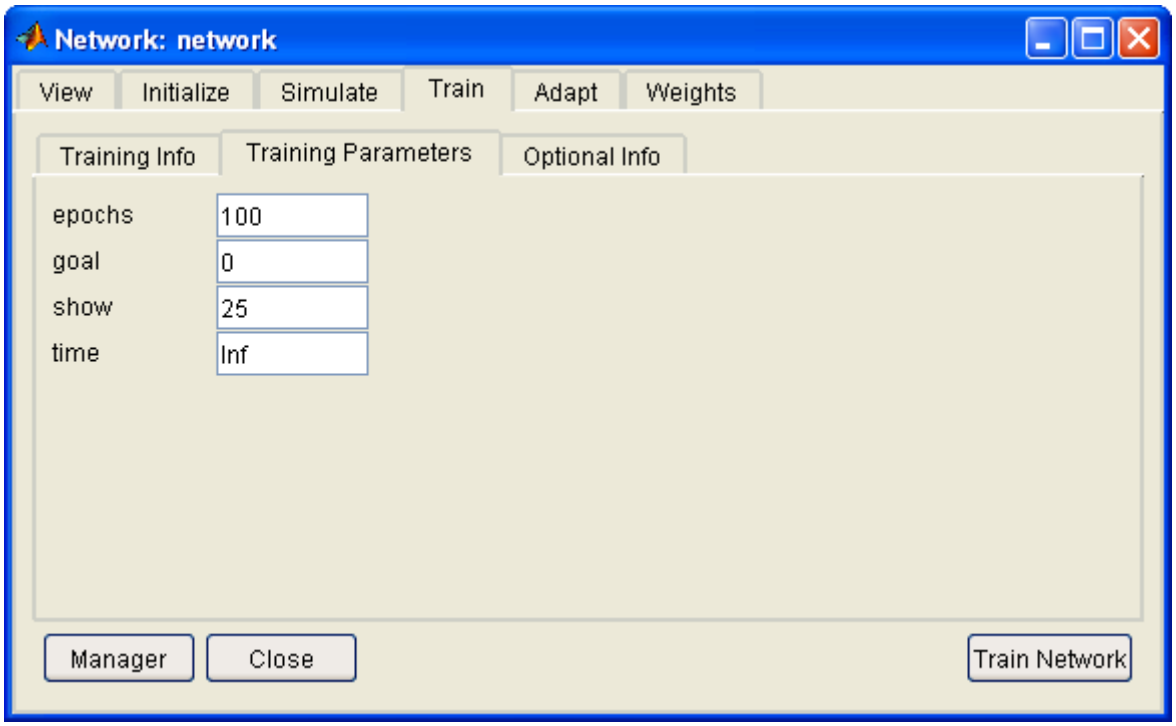


Рис.7. Вкладка параметров обучения

Принимая во внимание тот факт, что для задач с линейно отделимыми множествами (а наша задача относится к этому классу) все-

гда существует точное решение, установим порог достижения цели, равный нулю. Значения остальных параметров оставим по умолчанию. Заметим только, что поле времени обучения содержит запись Inf, которая определяет бесконечный интервал времени (от английского Infinite - бесконечный).

Следующая вкладка "Необязательная информация" (Optional Info) показана на рис.8.

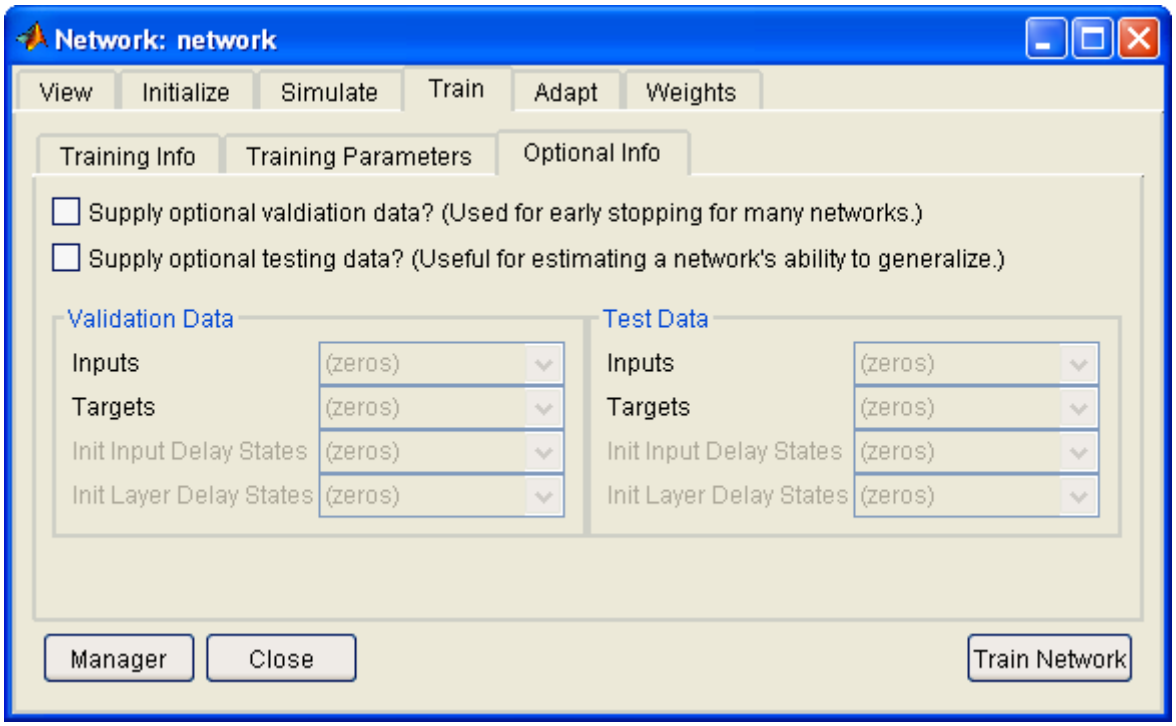


Рис.8. Вкладка необязательной информации

Рассмотрим вкладку обучения (Train). Чтобы начать обучение, нужно нажать кнопку "Обучить сеть" (Train Network). После этого, если в текущий момент сеть не удовлетворяет ни одному из условий, указанных в разделе параметров обучения (Training Parameters), появится ок-но, иллюстрирующее динамику целевой функции - кривую обучения. В нашем случае график может выглядеть так, как показано на рис. 6.9. Кнопкой "Остановить обучение" (Stop Training) можно прекратить этот процесс. Из рисунка видно, что обучение было остановлено, когда функция цели достигла установленной величины (goal = 0).

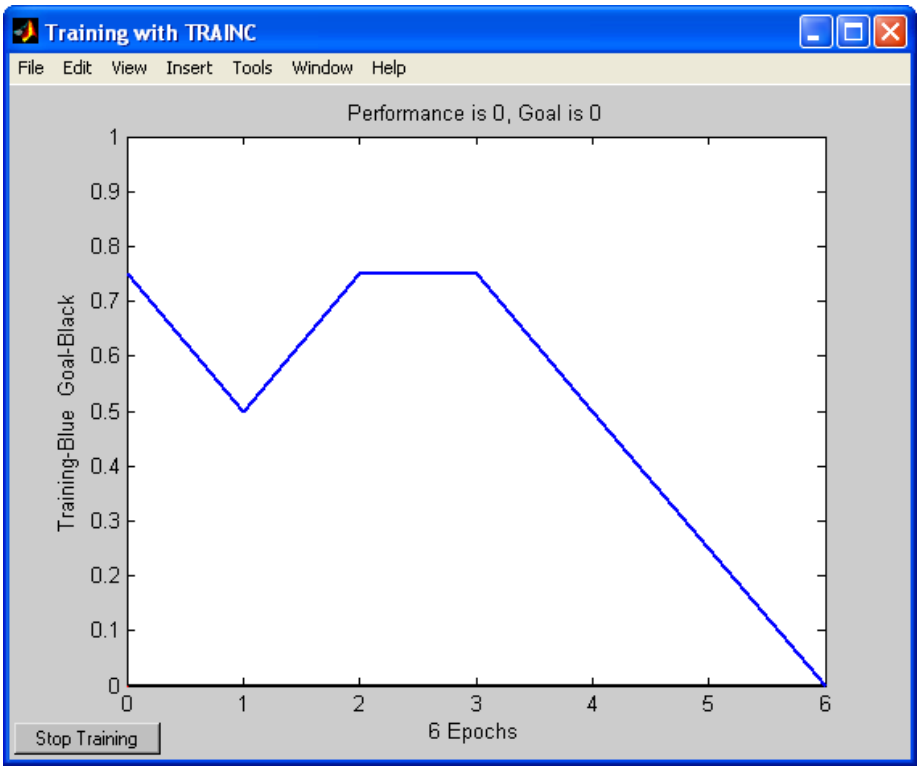


Рис. 6.9. Кривая обучения

Следует отметить, что для персептронов, имеющих функцию активации с жёстким ограничением, ошибка рассчитывается как разница между целью и полученным выходом.

Итак, алгоритм обучения нашёл точное решение задачи. В методических целях убедимся в правильности решения задачи путём прогона обученной сети. Для этого необходимо открыть вкладку "Прогон" (Simulate) и выбрать в выпадающем списке "Входы" (Inputs) заготовленные данные. В данной задаче естественно использовать тот же набор данных, что и при обучении data1. При желании можно установить флажок "Задать цели" (Supply Targets). Тогда в результате прогона дополнительно будут рассчитаны значения ошибки. Нажатие кнопки "Прогон сети" (Simulate Network) запишет результаты прогона в переменную, имя которой указано в поле "Выходы" (Outputs). Теперь можно вернуться в основное окно NNTool и, выделив мышью выходную переменную network1, нажать кнопку "Просмотр" (View). Содержимое окна просмотра совпадает со значением вектора целей - сеть работает правильно.

Следует заметить, что сеть создаётся инициализированной, то есть значения весов и смещений задаются определённым образом.

Перед каждым следующим опытом обучения обычно начальные условия обновляются, для чего на вкладке "Инициализация" (Initialize) предусмотрена функция инициализации. Так, если требуется провести несколько независимых опытов обучения, инициализация весов и смещений перед каждым из них осуществляется нажатием кнопки "Инициализировать веса" (Initialize Weights).

Вернёмся к вкладке "Необязательная информация" (Optional Info) (рис.8). Чтобы понять, какой цели служат представленные здесь параметры, необходимо обсудить два понятия: переобучение и обобщение.

При выборе нейронной сети для решения конкретной задачи трудно предсказать её порядок. Если выбрать неоправданно большой порядок, сеть может оказаться слишком гибкой и может представить простую зависимость сложным образом. Это явление называется переобучением. В случае сети с недостаточным количеством нейронов, напротив, необходимый уровень ошибки никогда не будет достигнут. Здесь налицо чрезмерное обобщение.

Для предупреждения переобучения применяется следующая техника. Данные делятся на два множества: обучающее (Training Data) и контрольное (Validation Data). Контрольное множество в обучении не используется. В начале работы ошибки сети на обучающем и контрольном множествах будут одинаковыми. По мере того, как сеть обучается, ошибка обучения убывает, и, пока обучение уменьшает действительную функцию ошибки, ошибка на контрольном множестве также будет убывать. Если же контрольная ошибка перестала убывать или даже стала расти, это указывает на то, что обучение следует закончить. Остановка на этом этапе называется ранней остановкой (Early stopping).

Таким образом, необходимо провести серию экспериментов с различными сетями, прежде чем будет получена подходящая. При этом чтобы не быть введённым в заблуждение локальными минимумами функции ошибки, следует несколько раз обучать каждую сеть.
Если в результате последовательных шагов обучения и контроля ошибка остаётся недопустимо большой, целесообразно изменить модель нейронной сети (например, усложнить сеть, увеличив число нейронов, или использовать сеть другого вида). В такой ситуации рекомендуется применять ещё одно множество - тестовое множество наблюдений (Test Data), которое представляет собой независимую выборку из входных данных. Итоговая модель тестируется на этом

множестве, что даёт дополнительную возможность убедиться в достоверности полученных результатов. Очевидно, чтобы сыграть свою роль, тестовое множество должно быть использовано только один раз. Если его использовать для корректировки сети, оно фактически превратится в контрольное множество.

Установка верхнего флажка (рис.8) позволит задать контрольное множество и соответствующий вектор целей (возможно, тот же, что при обучении). Установка нижнего - позволяет задать тестовое множество и вектор целей для него.

Обучение сети можно проводить в разных режимах. В связи с этим, в NNTool предусмотрено две вкладки, представляющие обучающие функции: рассмотренная ранее вкладка Train и "Адаптация" (Adapt). Adapt вмещает вкладку информация адаптации (Adap-tion Info), на которой содержатся поля, схожие по своему назначению с полями вкладки Training Info и выполняющие те же функции и вкладку параметры адаптации (Adaption Parameters). Последняя содержит единственное поле "Проходы" (passes). Значение, указанное в этом поле, определяет, сколько раз все входные векторы будут представлены сети в процессе обучения.

Приложение 2

СПИСОК ОСНОВНЫХ ФУНКЦИЙ NEURALNETWORKTOOLBOX:

Функции сгруппированы по назначению.

Функции для анализа

rrsurf – поверхность ошибки нейрона с единственным входом

maxlinlr – максимальная скорость обучения для линейного нейрона

Функции отклонения

boxdist – расстояние между двумя векторами

dist – евклидова весовая функция отклонения

linkdist – связанная фунция отклонения

mandist – весовая функция отклонения Манхеттена

Функции графического интерфейса

nntool – вызов графического интерфейса пользователя

Функции инициализации слоя

initnw – функция инициализации Нгуен-Видроу (Nguyen-Widrow)

initwb – функция инициализации по весам и смещениям

Функции обучения

learncon – обучающая функция смещений

learngd – обучающая функция градиентного спуска

learngdm – обучающая функция градиентного спуска с учетом моментов

learnh – обучащая функция Хэбба

learnhd – обучающая функция Хэбба с учетом затухания

learnis – обучающая функция instar

learnk – обучающая функция Кохонена

learnlv1 – LVQ1 обучающая функция

learnlv2 – LVQ2 обучающая функция

learnos – outstar обучающая функция

learnp – обучающая функция смещений и весов перцептрона

learnpn - обучающая функция нормализованных смещений и весов перцептрона

learnsom – обучающая функция самоорганизующейся карты весов

learnwh – правило обучения Уидроу-Хоффа (Widrow-Hoff)

Линейные функции поиска

srchbac – одномерная минимизация с использованием поиска с возвратом

srchbre – одномерная локализация интервала с использованием метода Брента (Brent)

srchcha - одномерная минимизация с использованием метода Караламбуса (Charalambous)

srchgol - одномерная минимизация с использованием золотого сечения

srchhyb - одномерная минимизация с использованием гибридного бисекционного поиска

Функции вычисления производных от входов сети

dnetprod – вычисление производной от входов сети с перемножением входов

dnetsum – вычисление производной от входов сети с суммированием входов

Входные функции сети

netprod – функция произведения входов

netsum – функция суммирования входов

Функции инициализации сети

initlay – функция послойной инициализации сети

Функции использования сети

adapt – разрешает адаптацию сети

disp – отображает свойства нейронной сети

display – отображает имена переменных и свойства сети

init – инициализация нейронной сети

sim – моделирование нейронной сети

train – тренировка нейронной сети

Функции создания новой сети

network – создание нейронной сети пользователя

newc – создание конкурентного слоя

newcf – создание каскадной направленной сети

newelm – создание сети обратного распространения Элмана (Elman)

newff – создание однонаправленной сети

newfftd – создание однонаправленной сети с входными задержками

newgrnn – создание обобщенной регрессонной нейронной сети

newhop – создание рекуррентной сети Хопфилда

newlin – создание линейного слоя

newlind – конструирование линейного слоя

newlvq – создание квантованной сети

newp – создание перцептрона

newpnn – конструирование вероятностной нейронной сети

newrb – конструирование сети с радиальным базисом

newrbe – конструирование точной сети с радиальными базисными функциями

newsom – создание самоорганизующейся карты

Функции производных функционирования

dmae – средняя абсолютная ошибка вычисления производной

dmse – средне-квадратичная ошибка производной

dmsereg - средне-квадратичная ошибка производной w/reg

dsse – суммарная квадратичная ошибка производной

Функции выполнения

mae - средняя абсолютная ошибка

mse – средне-квадратичная ошибка

msereg - средне-квадратичная ошибка w/reg

sse - суммарная квадратичная ошибка

Функции графики

hintonw – график Хинтона для матрицы весов

hintonwb – график Хинтона для матрицы весов и векторов смещений

plotbr – график функционирования сети при регулярной тренировке (Bayesian)

plotep – изображение положений весов и смещений на поверхности ошибки

plotes – изображение поверхности ошибок единичного входного нейрона

plotpc – изображение линии классификации в векторном пространстве перцептрона

plotperf – графическое представление функционирования сети

plotpv - графическое представление входных целевых векторов

plotsom – графическое представление самоорганизующейся карты

plotv – графическое представление векторов в виде линий, выходящих из начала координат

plotvec - графическое представление векторов различными цветами

Функции предварительной и пост обработки

postmnmx – ненормализованные данные, которые были нормализованы посредством prenmmx

postreg – линейный регрессионный анализ выходов сети по отношению к целевым значениям обучающего массива

poststd – ненормированные данные, которые были нормированы с помощью функции prestd

premnmx – нормирование данных в диапазоне от –1 до +1

prepca – анализ главных компонент для входных данных

prestd – нормирование данных к единичному стандартному отклонению и нулевому среднему

tramnmx – преобразование данных с предварительно вычисленными минимумом и максимумом

trapca – преобразование данных с использованием PCA матрицы, вычисленной с помощью функции prepca

trastd – преобразование данных с использованием предварительно вычисленных значений стандартного отклонения и среднего

Функции поддержки «Simulink»

gensim – генерация блока «Simulink» для моделирования нейронной сети

Топологические функции

gridtop – топологическая функция в виде сеточного слоя

hextop – топологическая функция в виде гексагонального слоя

randtop – топологическая функция в виде случайного слоя

Функции тренировки

trainb – пакетная тренировка с использованием правил обучения для весов и смещений

trainbfg – тренировка сети с использованием квази–Ньютоновского метода BFGS

trainbr – регуляризация Bayesian

trainc – использование приращений циклического порядка

traincgb – метод связанных градиентов Пауэлла-Била (Powell-Beale)

traincgf - метод связанных градиентов Флетчера-Пауэлла (Fletcher-Powell)

traincgp - метод связанных градиентов Полака-Рибира (Polak-Ribiere)

traingd – метод градиентного спуска

traingda – метод градиентного спуска с адаптивным обучением

traingdm - метод градиентного спуска с учетом моментов

traingdx - метод градиентного спуска с учетом моментов и с адаптивным обучением

trainlm – метод Левенберга-Маркара (Levenberg-Marquardt)

trainoss – одноступенчатый метод секущих

trainr – метод случайных приращений

trainrp – алгоритм упругого обратного распространения

trains – метод последовательных приращений

trainscg - метод шкалированных связанных градиентов

Производные функций активации

dhardlim – производная ступенчатой функции активации

dhardlms – производная симметричной ступенчатой функции активации

dlogsig – производная сигмоидной (логистической) функции активации

dposlin – производная положительной линейной функции активации

dpurelin – приозводная линейной функции активации

dradbas – производная радиальной базисной функции активации

dsatlin – производная насыщающейся линейной функции активации

dsatlins – производная симметричной насыщающейся функции активации

dtansig – производная функции активации гиперболический тангенс

dtribas – производная треугольной функции активации

Функции активации

compet – конкурирующая функция активации

hardlim – ступенчатая функция активации

hardlims – ступенчатая симметричная функция активации

logsig – сигмоидная (логистическая) функция активации

poslin – положительная линейная функция активации

purelin – линейная функция активации

radbas – радиальная базисная функция активации

satlin – насыщающаяся линейная функция активации

satlins – симметричная насыщающаяся линейная функция активации

softmax – функция активации, уменьшающая диапазон входных значений

tansig – функция активации гиперболический тангенс

tribas – треугольная функция активации

Полезные функции

calca – вычисляет выходы сети и другие сигналыСписок функций Neural Network

calca1 – вычисляет сигналы сети для одного шага по времени

calce – вычисляет ошибки слоев

calce1 - вычисляет ошибки слоев для одного шага по времени

calcgx – вычисляет градиент весов и смещений как единственный вектор

calcjejj – вычисляет Якобиан

calcjx – вычисляет Якобиан весов и смещений как одну матрицу

calcpd – вычисляет задержанные входы сети

calcperf – вычисление выходов сети, сигналов и функционирования

formx – формирует один вектор из весов и смещений

getx – возвращает все веса и смещения сети как один вектор

setx – устанавливает все веса и смещения сети в виде одного вектора

Векторные функции

cell2mat – объединяет массив элементов матриц в одну матрицу

combvec – создает все комбинации векторов

con2seq – преобразует сходящиеся векторы в последовательные векторы

concur – создает сходящиеся векторы смещений

ind2vec – преобразование индексов в векторы

mat2cell – разбиение матрицы на массив элементов матриц

minmax – вычисляет минимальные и максимальные значения строк матрицы

normc – нормирует столбцы матрицы

normr – нормирует строки матрицы

pnormc – псевдо-нормировка столбцов матрицы

quant – дискретизация величины

seq2con – преобразование последовательных векторов в сходящиеся векторы

sumsqr – сумма квадратов элементов матрицы

vec2ind – преобразование векторов в индексы

Функции инициализации весов и смещений

initcon – "сознательная" функция инициализации

initzero – инициализация с установкой нулевых значений весов и смещений

midpoint – инициализация с установкой средних значений весов

randnc – инициализация с установкой нормализованных значений столбцов весовых матриц

randnr - инициализация с установкой нормализованных значений строк весовых функций

rands – инициализация с установкой симметричных случайных значений весов и смещений

revert – возвращение весам и смещениям значений, соответствующих предыдущей инициализации

Функции весовых производных

ddotprod – производная скалярного произведения

Весовые функции

dist – Евклидово расстояние

dotprod – весовая функция в виде скалярного произведения

mandist – весовая функция – расстояние Манхеттена
negdist – весовая функция – отрицательное расстояние
normprod – нормированное скалярное произведение

Приложение 3

РАБОТА С FIS-РЕДАКТОРОМ

1. FIS-редактор

FIS-редактор предназначен для создания, сохранения, загрузки и вывода на печать систем нечеткого логического вывода, а также для редактирования следующих свойств:

тип системы;
наименование системы;
количество входных и выходных переменных;
наименование входных и выходных переменных;
параметры нечеткого логического вывода.

Загрузка FIS-редактора происходит с помощью команды fuzzy. В результате появляется интерактивное графическое окно (см. на рис. 5). На этом же рисунке также указаны функциональные назначения основных полей графичекого окна. В нижней части графического окна FIS-редактора расположены кнопки Help и Close, которые позволяют вызвать окно справки и закрыть редактор, соответственно.

FIS-редактор содержит 8 меню. Это три общесистемных меню - File, Edit, View, и пять меню для выбора параметров нечеткого логического вывода – And Method, Or Method, Implication, Aggregation и Defuzzification.

Меню File

Это общее меню для всех GUI-модулей используемых с системами нечеткого логического вывода. Общий вид меню показан на рис. 1.

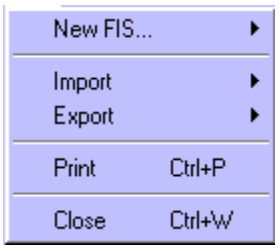


Рис.1. Меню File

С помощью команды New FIS… пользователь имеет возможность создать новую систему нечеткого логического вывода. При выборе этой команды появятся две альтернативы: Mamdani и Sugeno, которые определяют тип создаваемой системы. Создать систему типа Mamdani можно также нажатием Ctrl+N.

С помощью команды Import пользователь имеет возможность загрузить ранее созданную систему нечеткого логического вывода. При выборе этой команды появятся две альтернативы From Workspace… и From disk, которые позволяют загрузить систему нечеткого логического вывода из рабочей области MATLAB и с диска, соответственно. При выборе команды From Workspace… появится диалоговое окно, в котором необходимо указать идентификатор системы нечеткого логического вывода, находящейся в рабочей области MATLAB. При выборе команды From disk появится диалоговое окно (рис. 2), в котором необходимо указать имя файла системы нечеткого логического вывода. Файлы систем нечеткого логического вывода имеют расширение .fis . Загрузить систему нечеткого логического вывода с диска можно также нажатием Ctrl+N или командой **fuzzy FIS_name**, где FIS_name – имя файла системы нечеткого логического вывода.

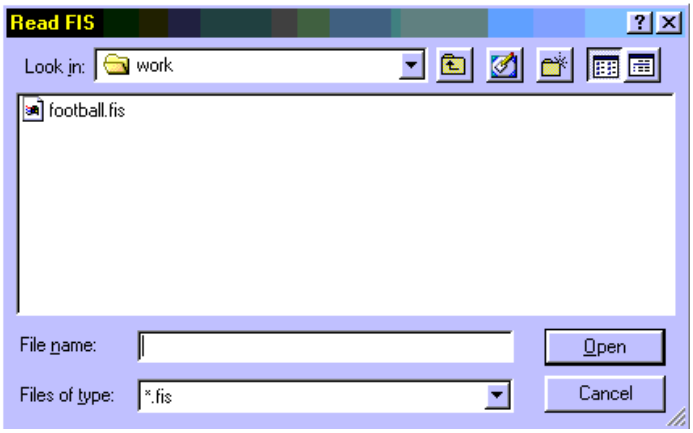


Рис.2 – Окно загрузки системы нечеткого логического вывода с диска

При выборе команды Export появятся две альтернативы To Workspace… и To disk, которые позволяют скопировать систему нечеткого логического вывода в рабочую область MATLAB и на диск, соответственно. При выборе команды To Workspace… появится диалоговое окно, в котором необходимо указать идентификатор системы нечеткого логического вывода, под которым она будет сохранена в рабочей области MATLAB. При выборе команды To disk появится

диалоговое окно, в котором необходимо указать имя файла системы нечеткого логического вывода. Скопировать систему нечеткого логического вывода в рабочую область и на диск можно также нажатием Ctrl+T и Ctrl+S, соответственно.

Команда Print позволяет вывести на принтер копию графического окна. Печать возможна также по нажатию Ctrl+P.

Команда Close закрывает графическое окно. Закрытия графического окна происходит по нажатию Ctrl+W или однократного щелчка левой кнопки мыши по кнопке Close.

Меню Edit.Общий вид меню приведен на рис. 3.

Рис.3 – Меню Edit

Команда Undo отменяет ранее совершенное действие. Выполняется также по нажатию Ctrl+Z.

Команда Add Variable… позволяет добавить в систему нечеткого логического вывода еще одну переменную. При выборе этой команды появятся две альтернативы Input и Output, которые позволяют добавить входную и выходную переменную, соответственно.

Команда Remove Selected Variable удаляет текущую переменную из системы. Признаком текущей переменной является красная окантовка ее прямоугольника. Назначение текущей переменной происходит с помощью однократного щелчка левой кнопки мыши по ее прямоугольнику. Удалить текущую переменную можно также с помощью нажатия Ctrl+X.

Команда Membership Function… открывает редактор функций принадлежностей. Эта команда может быть также выполнена нажатием Ctrl+2.

Команда Rules… открывает редактор базы знаний. Эта команда может быть также выполнена нажатием Ctrl+3.

Меню View

Это общее меню для всех GUI-модулей, используемых с системами нечеткого логического вывода. Общий вид меню показан на

рис. 4. Это меню позволяет открыть окно визуализауии нечеткого логического вывода (команда Rules или нажатие клавиш Ctrl+5) и окно вывода поверхности “входы-выход”, соответствующей системе нечеткого логического вывода (команда Surface или нажатие клавиш Ctrl+6).

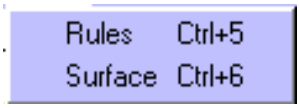


Рис.4 – Меню View

Меню And Method

Это меню позволяет установить следующие реализации логической операции И:

min – минимум;

prod – умножение.

Пользователь также имеет возможность установить собственную реализацию операции И. Для этого необходимо выбрать команду Custom… и в появившемся графическом окне напечатать имя функции, реализующей эту операцию.

Меню Or Method

Это меню позволяет установить следующие реализации логической операции ИЛИ:

max – умножение;

probor - вероятностное ИЛИ.

Пользователь также имеет возможность установить собственную реализацию операции ИЛИ. Для этого необходимо выбрать команду Custom… и в появившемся графическом окне напечатать имя функции, реализующей эту операцию.

Меню Implication

Это меню позволяет установить следующие реализации импликации:

min – минимум;

prod – умножение.

Пользователь также имеет возможность установить собственную реализацию импликации. Для этого необходимо выбрать команду Custom… и в появившемся графическом окне напечатать имя функции, реализующей эту операцию.

Меню Aggregation

Это меню позволяет установить следующие реализации операции объединения функций принадлежности выходной переменной:

max – максимум;

sum – сумма;

probor - вероятностное ИЛИ.

Пользователь также имеет возможность установить собственную реализацию этой операции. Для этого необходимо выбрать команду Custom… и в появившемся графическом окне напечатать имя функции, реализующей эту операцию

Меню Defuzzification

Это меню позволяет выбрать метод дефаззификации. Для систем типа Мамдани запрограммированы следующие методы:

centroid – центр тяжести;

bisector –медиана;

lom – наибольший из максимумов;

som – наименьший из максимумов;

mom – среднее из максимумов.

Для систем типа Сугэно запрограммированы следующие методы:

wtaver – взвешенное среднее;

wtsum – взвешенная сумма.

Пользователь также имеет возможность установить собственный метод деффазификации. Для этого необходимо выбрать команду Custom… и в появившемся графическом окне напечатать имя функции, реализующей эту операцию.

2. Редактор функций принадлежности

Редактор функций принадлежности - (Membership Function Editor) редактор предназначен для задания следующей информации о терм - множествах входных и выходных переменных:

- количество термов;

- наименования термов;

- тип и параметры функций принадлежности, которые необходимы для представления лингвистических термов в виде нечетких множеств.

Редактор функций принадлежности может быть вызван из любого GUI-модуля, используемого с системами нечеткого логического вывода, командой Membership Functions… меню Edit или нажатием клавиш Ctrl+2. В FIS-редакторе открыть редактор функций принадлежности можно также двойным щелчком левой кнопкой мыши по

полю входной или выходной переменных. Общий вид редактора функций принадлежности с указанием функционального назначения основных полей графического окна приведен на рис. 5. В нижней части графического окна расположены кнопки Help и Close, которые позволяют вызвать окно справки и закрыть редактор, соответственно.

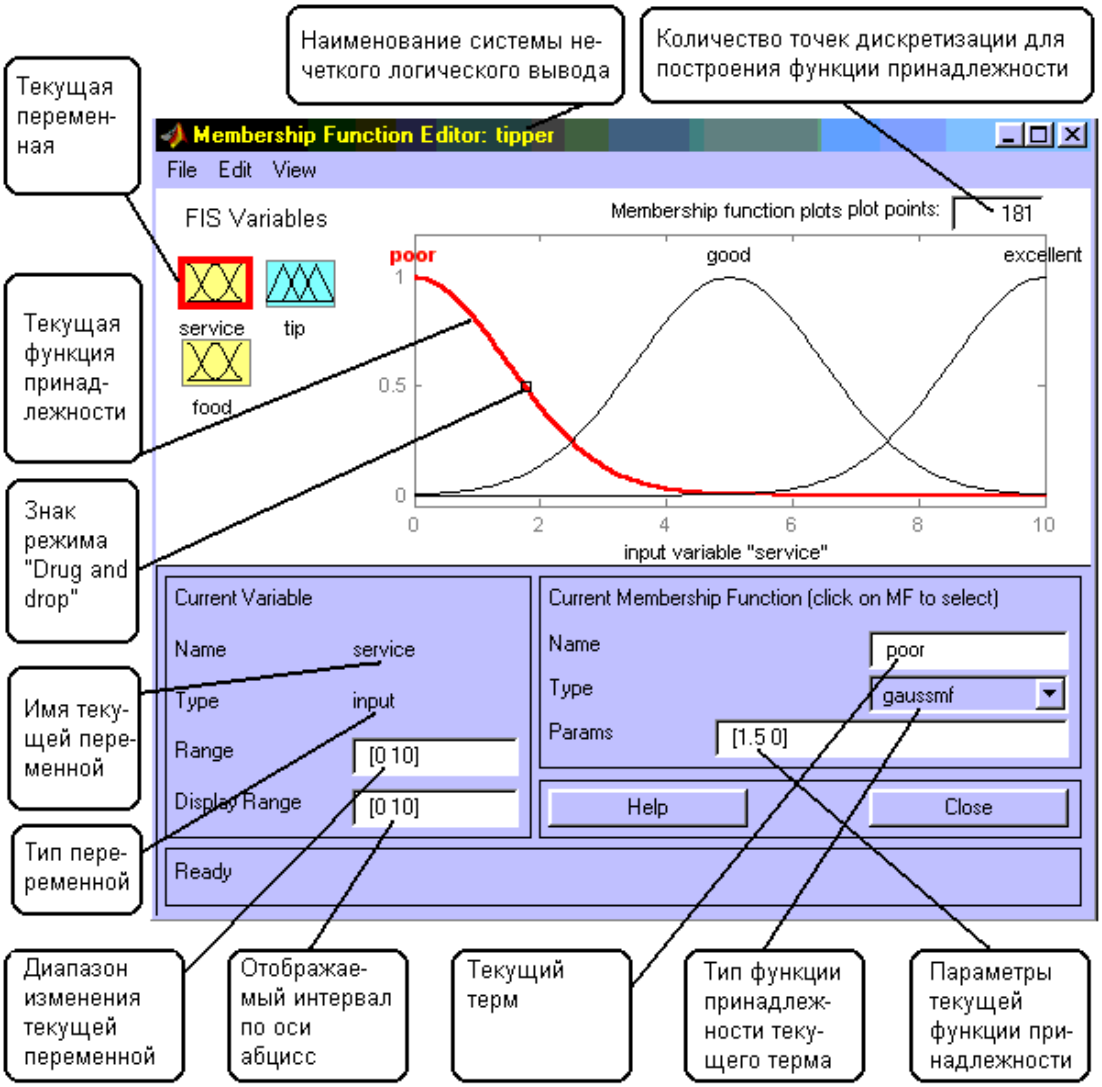


Рис.5. Редактор функций принадлежности

Редактор функций принадлежности содержит четыре меню - File, Edit, View, Type и четыре окна ввода информации – Range, Display Range, Name и Params. Эти четыре окно предназначенны для задания диапазона изменения текущей переменной, диапазона вывода функций принадлежности, наименования текущего лингвистического терма и параметров его функции принадлежности, соответственно. Параметры функции принадлежности можно подбирать и в графическом режиме, путем изменения формы функции принадлежности с

помощью технологии “Drug and drop”. Для этого необходимо позиционировать курсор мыши на знаке режима “Drug and drop” (см. рис. 5), нажать на левую кнопку мыши и не отпуская ее изменять форму функции принадлежности. Параметры функции принадлежности будут пересчитываться автоматически.

Меню File и View одинаковые для всех GUI-модулей используемых с системами нечеткого логического вывода. Они подробно описаны в разделе 1.

Меню Edit. Общий вид меню приведен на рис. 6.

Рис.6. Меню Edit

Команда Undo отменяет ранее совершенное действие. Выполняется также по нажатию Ctrl+Z.

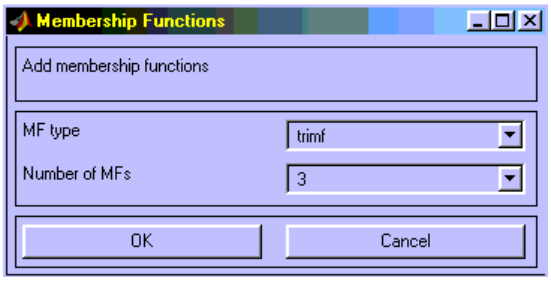


Рис.7. Выбор количества термов и типа функций принадлежности

Команда Add MFs… позволяет добавить термы в терм-множество, используемое для лингвистической оценки текущей переменной. При выборе этой команды появится диалоговое окно (рис. 7), в котором необходимо выбрать тип функции принадлежности и количество термов. Значения параметров функций принадлежности будут установлены автоматически таким образом, чтобы равномерно покрыть область определения переменной, заданной в окне Range. При изменении области определения в окне Range параметры функций принадлежности будут промасштабированы.

Команда Add Custom MF… позволяет добавить один лингвистический терм, функция принадлежности которого отличается от встроенных. После выбора этой команды появится графическое окно

(рис. 8), в котором необходимо напечатать лингвистически терм (поле MF name),, имя функции принадлежности (поле M-File function name) и параметры функции принадлежности (поле Parameter list).

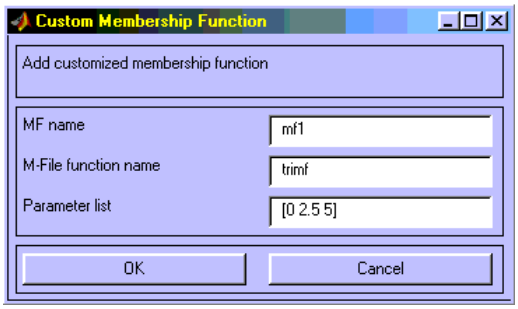


Рис.8. Задание лингвистического терма с невстроенной функцией принадлежности

Команда Remove Selected MF удаляет текущий терм из терм-множества текущей переменной. Признаком текущей переменной является красная окантовка ее прямоугольника. Признаком текущего терма является красный цвет его функции принадлежности. Для выбора текущего терма необходимо провести позиционирования курсора мыши на графике функции принадлежности и сделать щелчок левой кнопкой мыши.

Команда Remove All MFs удаляет все термы из терм-множества текущей переменной.

Команда FIS Properties… открывает FIS-редактор. Эта команда может быть также выполнена нажатием Ctrl+1.

Команда Rules… открывает редактор базы знаний. Эта команда может быть также выполнена нажатием Ctrl+3.

Меню Type

Это меню позволяет установить тип функций принадлежности термов, используемых для лингвистической оценки текущей переменной. На рис. 9 приведено меню Type, в котором указаны возможные типы функций принадлежности.

Рис.9. Меню Type

3. Редактор базы знаний

Редактор базы знаний (Rule Editor) предназначен для формирования и модификации нечетких правил. Редактор базы знаний может быть вызван из любого GUI-модуля, используемого с системами нечеткого логического вывода, командой Rules… меню Edit или нажатием клавиш Ctrl+3. В FIS-редакторе открыть редактор базы знаний можно также двойным щелчком левой кнопкой мыши по прямоугольнику с названием системы нечеткого логического вывода, расположенного в центре графического окна.

Общий вид редактора базы знаний с указанием функционального назначения основных полей графического окна приведен на рис. 10. В нижней части графического окна расположены кнопки Help и Close, которые позволяют вызвать окно справки и закрыть редактор, соответственно.

Редактор функций принадлежности содержит четыре системных меню File, Edit, View, Options, меню выбора термов входных и выходных переменных, поля установки логических операций И, ИЛИ, НЕ и весов правил, а также кнопки редактирования и просмотра правил.

Для ввода нового правила в базу знаний необходимо с помощью мыши выбрать соответствующую комбинацию лингвистических термов входных и выходных переменных, установить тип логической связки (И или ИЛИ) между переменными внутри правила, установить наличие или отсутствие логической операции НЕ для каждой лингвистической переменной, ввести значение весового коэффициента правила и нажать кнопку Add Rule. По умолчанию установлены следующие параметры:

логическая связка переменных внутри правила – И;

логическая операция НЕ – отсутствует;

значение весового коэффициента правила – 1.

Возможны случаи, когда истинность правила не изменяется при произвольном значении некоторой входной переменной, т.е. это переменная не влияет на результат нечеткого логического вывода в данной области факторного пространства. Тогда в качестве лингвистического значения этой переменной необходимо установить none.

Для удаления правила из базы знаний необходимо сделать однократный щелчок левой кнопкой мыши по этому правилу и нажать кнопку Delete Rule.

Для модификации правила необходимо сделать однократный щелчок левой кнопкой мыши по этому правилу, затем установить необходимые параметры правила и нажать кнопку Edit Rule.

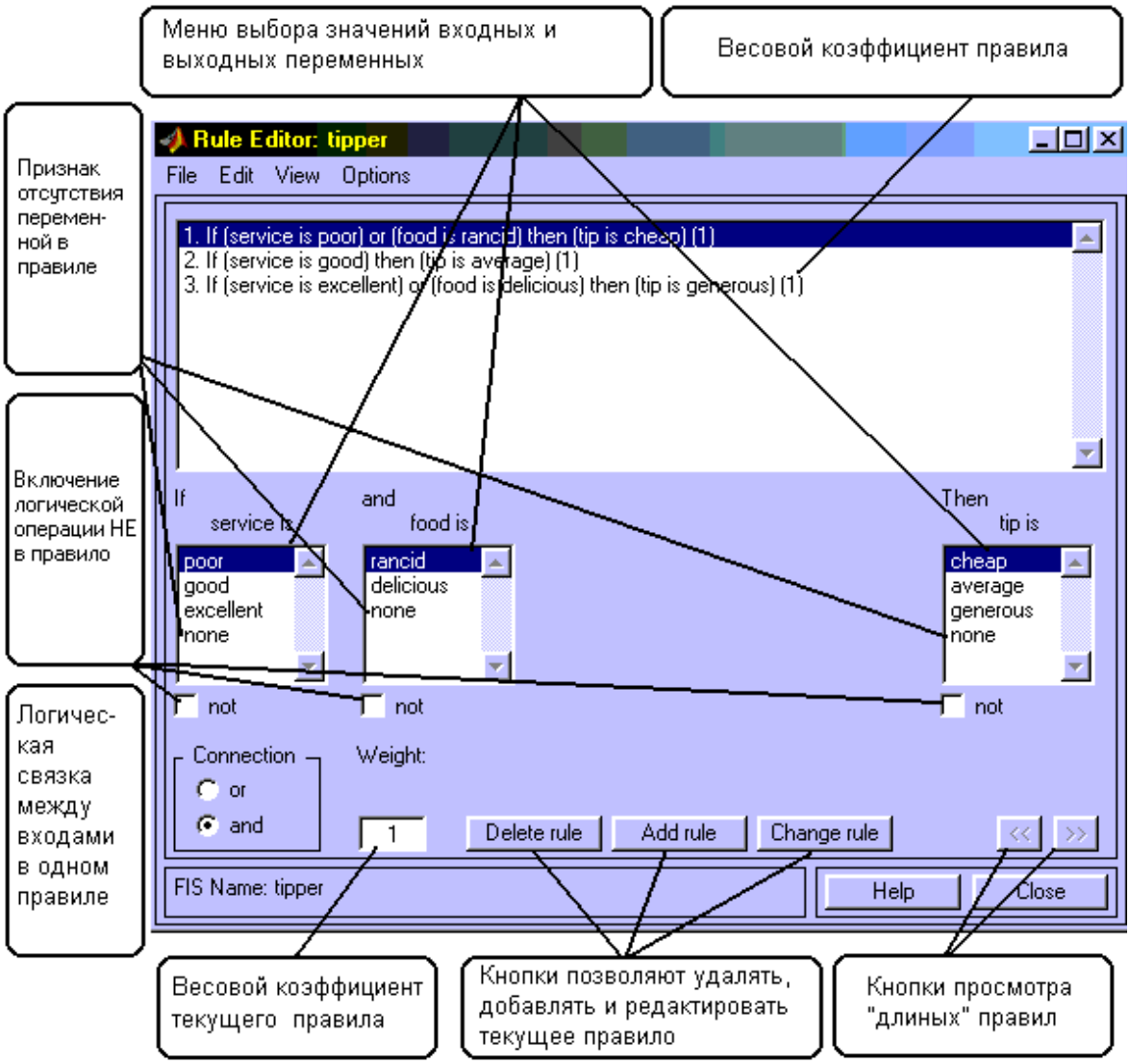


Рис.10. Редактор базы знаний

Меню File и View одинаковые для всех GUI-модулей используемых с системами нечеткого логического вывода. Они подробно описаны в разделе 1.

Меню Edit

Общий вид меню приведен на рис. 11.

Рис.11. Меню Edit

Команда Undo отменяет ранее совершенное действие. Выполняется также по нажатию Ctrl+Z.

Команда FIS Properties… открывает FIS-редактор. Эта команда может быть также выполнена нажатием Ctrl+1.

Команда Membership Function… открывает редактор функций принадлежностей. Эта команда может быть также выполнена нажатием Ctrl+2.

Меню Options

Это меню позволяет установить язык и формат правил базы знаний (рис. 12).

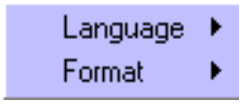


Рис.12. Меню Options

При выборе команды Language появится список языков English (Английский), Deutsch (Немецкий), Francais (Французский), из которого необходимо выбрать один.

При выборе команды Format появится список возможных форматов правил базы знаний: Verbose - лингвистический; Symbolic – логический; Indexed – индексированный. Различные форматы база знаний демо-системы нечеткого логического вывода Tipper приведены на рис. 10 (формат правил Verbose), рис. 13 (формат правил Symbolic) и рис. 14 (формат правил Indexed).

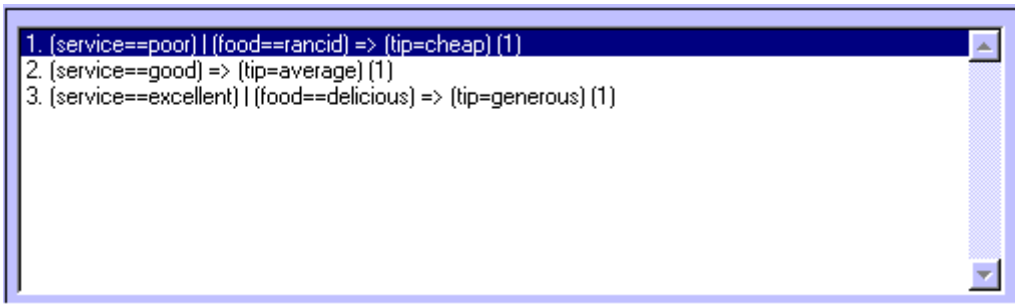


Рис.13. База знаний в формате Symbolic

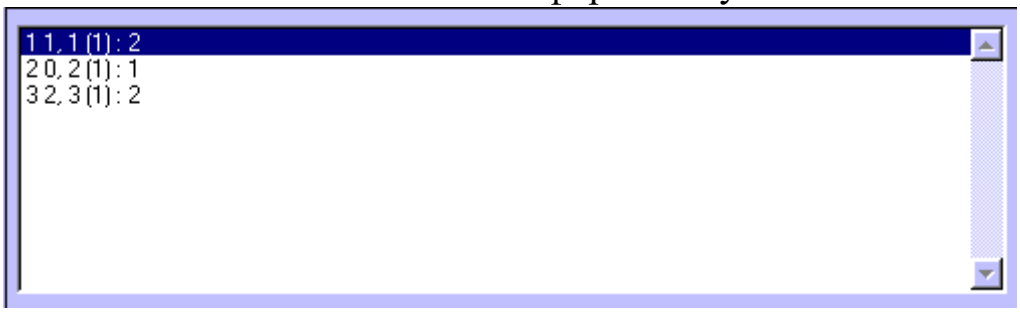


Рис.14. База знаний в формате Indexed

4. Визуализация нечеткого логического вывода

Визуализация нечеткого логического вывода осуществляется с помощью GUI-модуля Rule Viewer. Этот модуль позволяет проиллюстрировать ход логического вывода по каждому правилу,

получение результирующего нечеткого множества и выполнение процедуры дефаззификации. Rule Viewer может быть вызван из любого GUI-модуля, используемого с системами нечеткого логического вывода, командой View rules … меню View или нажатием клавиш Ctrl+4. Вид Rule Viewer для системы логического вывода tipper с указанием функционального назначения основных полей графического окна приведен на рис. 15.

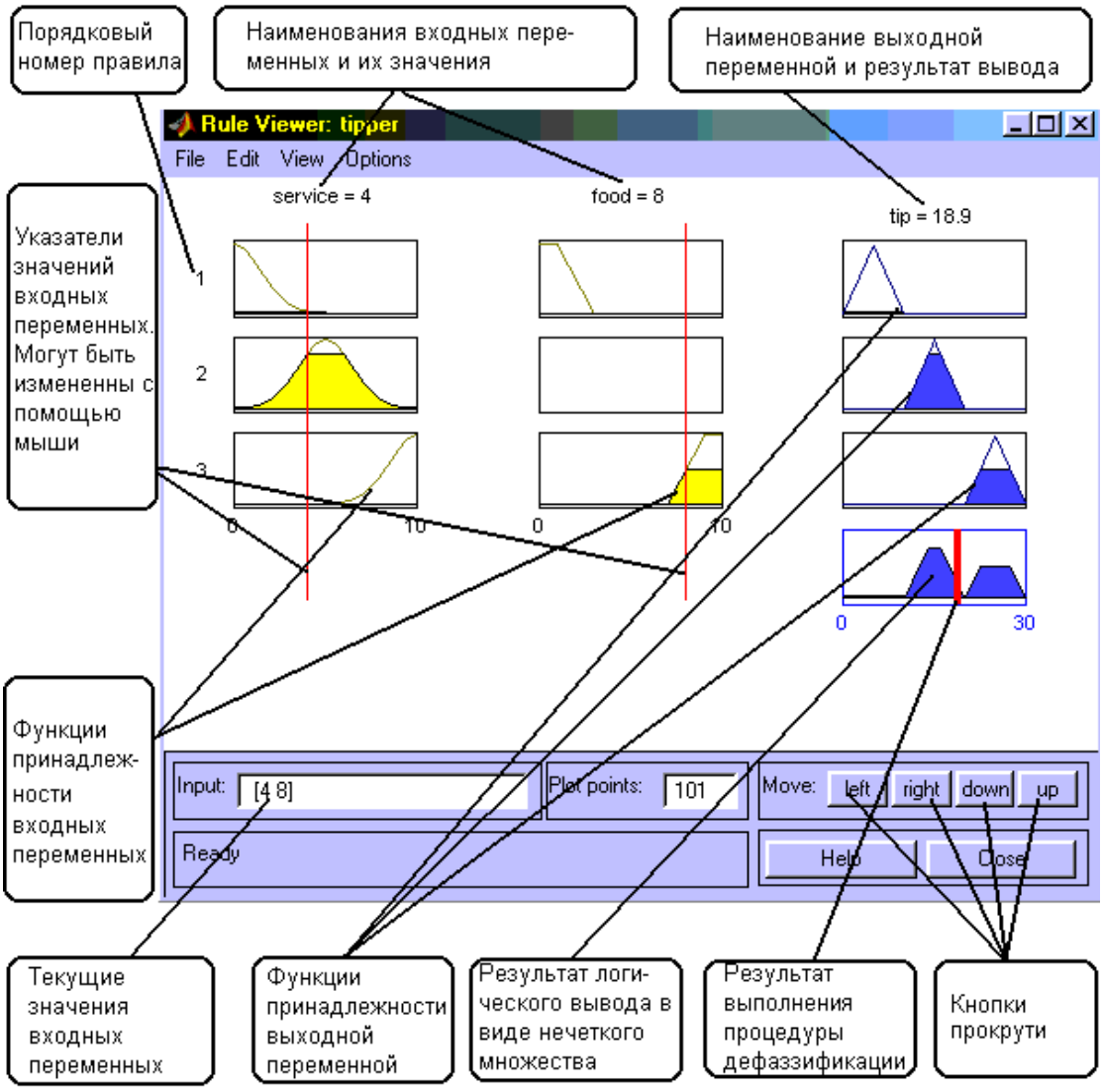


Рис.15 – Визуализация логического вывода для системы tipper с помощью Rule Viewer

Rule Viewer содержит четыре меню - File, Edit, View, Options, два поля ввода ввода информации – Input и Plot points и кнопки прокрутки изображения влево-вправо (left-right), вверх-вниз (up-down). В нижней части графического окна расположены также кнопки Help и Close, которые позволяют вызвать окно справки и закрыть редактор, соответственно.

Каждое правило базы знаний представляется в виде последовательности горизонтально расположенных прямоугольников. При этом первые два прямоугольника (см. рис. 15) отображают функции принадлежностей термов посылки правила (ЕСЛИ-часть правила), а последний третий прямоугольник соответствует функции принадлежности терма-следствия выходной переменной (ТО-часть правила). Пустой прямоугольник в визуализации второго правила означает, что в этом правиле посылка по переменной food отсутствует (food is none). Желтая заливка графиков функций принадлежностей входных переменных указывает насколько значения входов, соответствуют термам данного правила. Для вывода правила в формате Rule Editor необходимо сделать однократный щелчок левой кнопки мыши по номеру соответствующего правила. В этом случае указанное правило будет выведено в нижней части графического окна.

Голубая заливка графика функции принадлежности выходной переменной представляет собой результат логического вывода в виде нечеткого множества по данному правилу. Результирующее нечеткое множество, соответствующее логическому выводу по всем правилам показано в нижнем прямоугольнике последнего столбца графического окна. В этом же прямоугольнике красная вертикальная линия соответствует четкому значению логического вывода, полученного в результате дефаззификации.

Ввод значений входных переменных может осуществляться двумя способами:

путем ввода численных значений в поле Input;

с помощью мыши, путем перемещения линий-указателей красного цвета.

В последнем случае необходимо позиционировать курсор мыши на красной вертикальной линии, нажать на левую кнопку мыши и не отпуская ее переместить указатель на нужную позицию. Новое численное значения соответствующей входной переменной будет пересчитано автоматически и выведено в окно Input.

В поле Plot points задается количество точек дискретизации для построения графиков функций принадлежности. Значение по умолчанию – 101.

Меню File и View одинаковые для всех GUI-модулей используемых с системами нечеткого логического вывода. Они подробно описаны в разделе 1.

Меню Edit

Общий вид меню приведен на рис. 16.

Рис.16. Меню Edit

Команда FIS Properties… открывает FIS-редактор. Эта команда может быть также выполнена нажатием Ctrl+1.

Команда Membership Functions… открывает редактор функций принадлежностей. Эта команда может быть также выполнена нажатием Ctrl+2.

Команда Rules… открывает редактор базы знаний. Эта команда может быть также выполнена нажатием Ctrl+3.

Меню Options

Меню Options содержит только одну команду Format, которая позволяет установить один из следующих форматов вывода выбранного правило в нижней части графического окна:

Verbose - лингвистический;

Symbolic – логический;

Indexed – индексированный.

5. Визуализация поверхности “входы - выход”

Визуализация поверхности “входы-выход” осуществляется с помощью GUI-модуля Surface Viewer. Этот модуль позволяет вывести графическое изображение зависимости значения любой выходной переменной от произвольных двух (или одной) входных переменных. Surface Viewer может быть вызван из любого GUI-модуля, используемого с системами нечеткого логического вывода, командой View surface … меню View или нажатием клавиш Ctrl+4. Общий вид модуля Surface Viewer с указанием функционального назначения основных полей графического окна приведен на рис. 17.

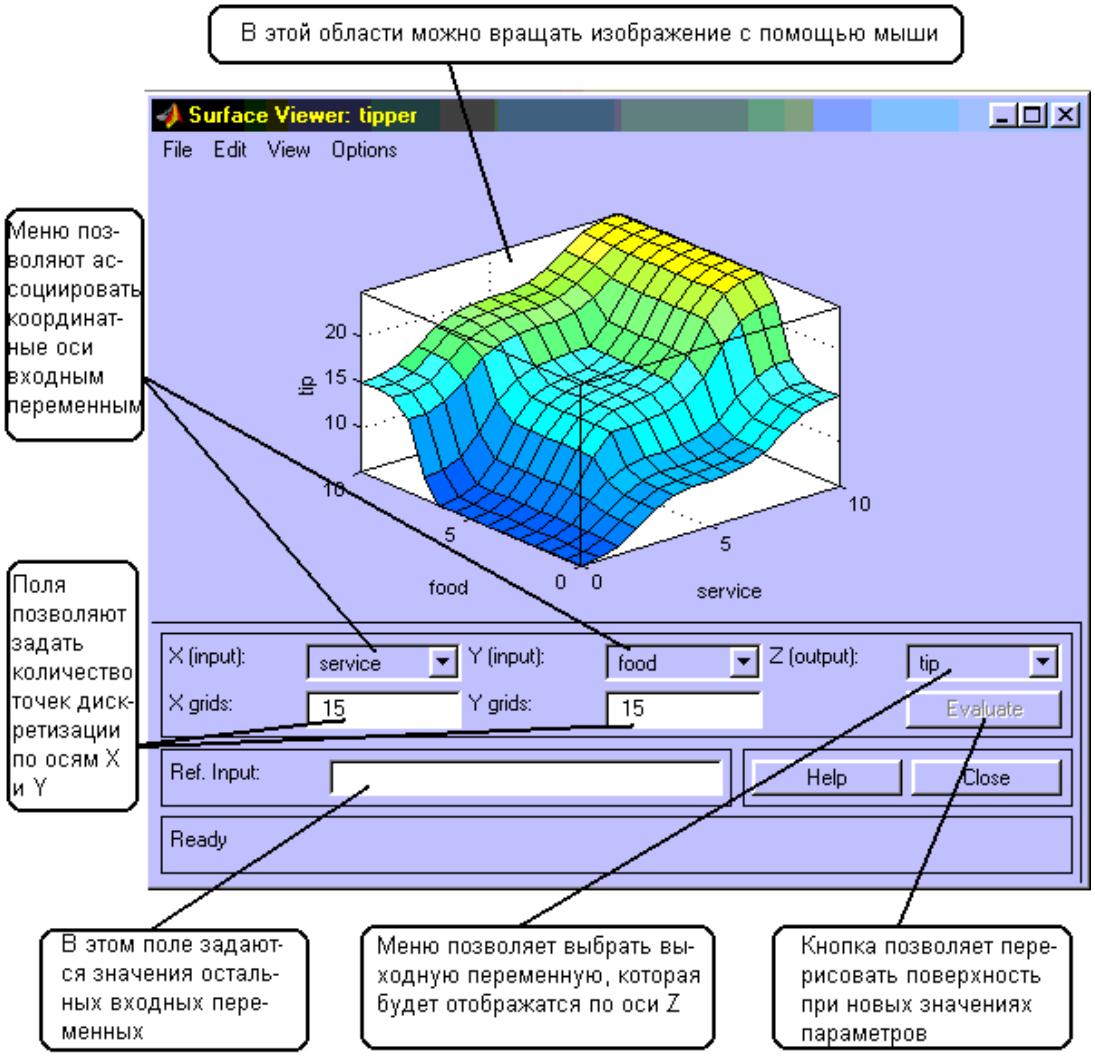


Рис.17. Визуализация поверхности “входы - выход” для системы tipper с помощью Surface Viewer

Surface Viewer содержит верхних системных меню - File, Edit, View, Options, три меню выбора координатных осей - X (input), Y (input), Z (output), три поля ввода ввода информации – X girds, Y girds, Ref. Input и кнопку Evaluate для построения поверхности при новых параметрах. В нижней части графического окна расположены также кнопки Help и Close, которые позволяют вызвать окно справки и закрыть редактор, соответственно.

Surface Viewer позволяет вращать поверхность “входы - выход” с помощью мыши. Для этого необходимо позиционировать курсор мыши на поверхности “входы-выход”, нажать на левую кнопку мыши и не отпуская ее повернуть графическое изображение на требуемый угол.

Поля X girds и Y girds предназначены для задания количества точек дискретизации по осям X и Y, для построения поверхности “входы - выход”. По умолчанию количество дискрет по каждой оси равно 15. Для изменения этого значения необходимо установить маркер на поле X girds (Y girds) и ввести новое значение.

Поле Ref. Input предназначено для задания значений входных переменных, кроме тех, которые ассоциированы с координатными осями. По умолчанию это значения середины интервалов изменения переменных. Для изменения этого значения необходимо установить маркер на поле Ref. Input и ввести новые значение.

Меню File и View одинаковые для всех GUI- модулей используемых с системами нечеткого логического вывода. Они подробно описаны в разделе 1.

Меню координатных осей

Меню X (input), Y (input), Z (output) позволяют поставить в соответствие осям координат входные и выходные переменные. При этом входные переменные могут отображаться только по осям X и Y, а выходные переменные только по оси Z. В Surface Viewer предусмотрена возможность построения однофакторных зависимостей “вход-выход”. Для этого в меню второй координатной оси (X (input) или Y (input)) необходимо выбрать none.

Меню Edit

Общий вид меню приведен на рис. 16. Назначения команд меню описано в разделе 4.

Меню Options

Меню Options изображено на рис. 18. Оно содержит команды Plot, Color Map и Always evaluate.

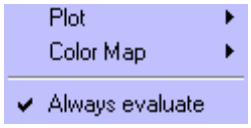


Рис.18. Меню Options

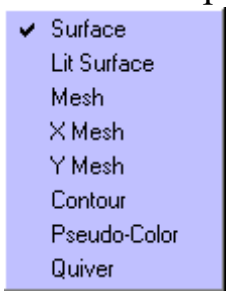


Рис.19. Меню Plot

Команда Plot позволяет управлять форматом вывода поверхности “входы-выход”. При выборе этой команды появляется меню (рис. 19) в котором необходимо выбрать формат вывода поверхности. На рис. 20 приведены поверхности “входы-выход” для системы tipper для всех поддерживаемых форматов.

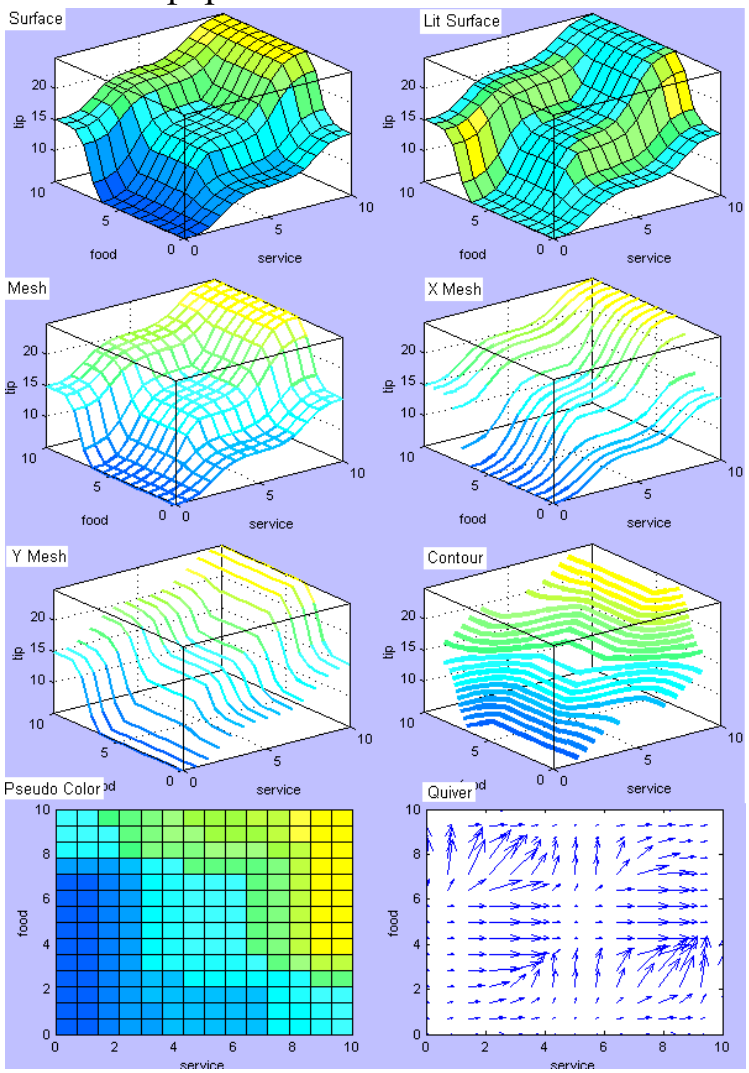


Рис.20. Примеры форматов поверхности “входы - выход”

Команда Color Map позволяет управлять палитрой цветов при выводе поверхности “входы-выход”. При выборе этой команды появляется меню, в котором необходимо выбрать одну из палитр:

default – использовать палитру, установленную по умолчанию;

blue – холодная сине-голубая палитра;

hot – теплая палитра, состоящая из черного, красного, желтого и белого цветов;

HSV – палитра насыщенных цветов: красный, желтый, зеленый, циан, голубой, мажента, красный.

Команда Always evaluate позволяет установить / отменить режим автоматического, т.е. без нажатия кнопки Evaluate, перерисовывания поверхности “входы - выход" при любом изменении параметров

Приложение 4

СИСТЕМА КОМАНД ДЛЯ ПРОГРАММИРОВАНИЯ ФУНКЦИЙ НЕЧЕТКОЙ ЛОГИКИ

ADDMF

Добавляет функцию принадлежности к системе нечеткого логического вывода

Синтаксис:***FIS_name=addmf(FIS_name, varType, varIndex, mfName, mfType, mfParams*)**

Функцию принадлежности можно добавить только к существующей в рабочей области MATLAB системе нечеткого логического вывода. Другими словами система нечеткого логического вывода должна быть каким-то образом загружена в рабочую область или создана с помощью функции newfis. Функция addmf имеет шесть входных аргументов:

FIS_name – идентификатор системы нечеткого логического вывода в рабочей области MATLAB;

varType – тип переменной, к которой добавляется функция принадлежности. Допустимые значения - ‘input’- входная переменная и ‘output’ – выходная переменная;

varIndex – порядковый номер переменной, к которой добавляется функция принадлежности;

mfName – наименование добавляемой функции принадлежности (терм). Задается в виде строки символов;

mfType – тип (модель) добавляемой функции принадлежности. Задается в виде строки символов;

mfParams – вектор параметров добавляемой функции принадлежности.

Порядковый номер функции принадлежности в системе нечеткого логического вывода соответствует порядку добавления с помощью функции addmf, т.е. первая добавленная функция принадлежности всегда будет иметь порядковый номер 1. С помощью функции addmf невозможно добавить функцию принадлежности к несуще-

ствующей переменной. В этом случае необходимо вначале добавить переменную к системе нечеткого логического вывода с помощью функции addvar.

Пример.

FIS_name=addmf(FIS_name, ‘input’, 1, ‘низкий’, ‘trapmf’, [150, 155, 165, 170])

Строка добавляет в терм-множество первой входной переменной нечеткой системы FIS_name терм ‘низкий’ с трапециевидной функцией принадлежности с параметрами [150, 155, 165, 170].

ADDRULE

Добавляет правила в базу знаний системынечеткого логического вывода

Синтаксис: **FIS_name= addrule (FIS_name, ruleList)**

Правила можно добавить только к существующей в рабочей области MATLAB системе нечеткого логического вывода. Функция addrule имеет два входных аргумента:

FIS_name – идентификатор системы нечеткого логического вывода в рабочей области MATLAB;

ruleList – матрица добавляемых правил.

Матрица правил должна быть задана в формате indexed. Количество строк матрицы ruleList равно количеству добавляемых правил, т.е. каждая строка матрицы соответствует одному правилу. Количество столбцов матрицы равно m+n+2, где m (n) – количество входных (выходных) переменных системы нечеткого логического вывода.

Первые m столбцов соответствуют входным переменным, т.е. задают ЕСЛИ-часть правил. Элементы этих столбцов содержат порядковые номера термов, используемых для лингвистической оценки соответствующих входных переменных. Значение 0 указывает, что соответствующая переменная в правиле не задана, т.е. ее значение равно none.

Следующие n столбцов соответствуют выходным переменным, т.е. задают ТО-часть правил. Элементы этих столбцов содержат порядковые номера термов, используемых для лингвистической оценки соответствующих выходных переменных.

Предпоследний столбец матрицы содержит весовые коэффициенты правил. Значения весовых коэффициентов должны быть в диапазоне [0, 1].

Последний столбец матрицы задает логические связки между переменными внутри правил. Значение 1 соответствует логической операции И, а значение 2 – логической операции ИЛИ.

Пример.

FIS_name=addrule(FIS_name, [1 1 1 1 1; 1 2 2 0.5 1])

Строка добавляет в базу знаний системы FIS_name два правила, которые интерпретируются следующим образом:

Если вход1=MF1 и вход2=MF1, то выход1=MF1 с весом 1,

Если вход1=MF1 и вход2=MF2, то выход1=MF2 с весом 0.5,

где MF1 (MF2) – терм с порядковым номером 1 (2).

ADDVAR

Добавляет переменную в систему нечеткого логического вывода

Синтаксис: **FIS_name= addvar (FIS_name, varType, varName, varBound)**

Переменную можно добавить только к существующей в рабочей области MATLAB системе нечеткого логического вывода. Функция addrvar имеет четыре входных аргумента:

FIS_name – идентификатор системы нечеткого логического вывода в рабочей области MATLAB;

varType – тип добавляемой переменной. Допустимые значения - ‘input’- входная переменная и ‘output’ – выходная переменная;

varName – наименование добавляемой переменной. Задается в виде строки символов;

varBound – вектор, задающий диапазон изменения добавляемой переменной.

Порядковый номер переменной в системе нечеткого логического вывода соответствует порядку добавления с помощью функции addvar, т.е. первая добавленная переменная будет иметь порядковый номер 1. Входные и выходные переменные нумеруются независимо.

Пример.

FIS_name=addrule(FIS_name, ‘input’, ‘Рост’, [155 205])

Строка добавляет в систему нечеткого логического вывода FIS_name входную переменную ‘Рост’, заданную на интервале [155 205].

ANFIS

Настройка систем нечеткогологического вывода типа Сугэно

Синтаксис:**[fis, error] = anfis(trndata)**

[fis, error] = anfis(trndata, initfis)

[fis, error, stepsize] = anfis(trndata, initfis, trnopt, dispopt, [], optmethod)

[fis, error, stepsize, chkfis, chkerror] = anfis(trndata, initfis, trnopt, dispopt, chkdata)

ANFIS являетсяаббревиатурой Adaptive Network – based Fuzzy Inference System. Это основная функция настройки систем нечеткого логического вывода типа Сугэно. Настройка представляет собой итерационную процедуру нахождения таких параметров системы нечеткого логического вывода, в частности параметров функций принадлежности, которые минимизируют расхождения между результатами логического вывода и экспериментальными данными, т. е. между действительным и желаемым поведениями системы. Экспериментальные данные, по которым настраивается функции принадлежности, представляются в виде обучающей выборки.

В основу функции anfis положен один из первых методов построения нейро-нечетких систем для аппроксимации функций, предложенный в 1991 году Янгом (Jang). Для идентификации параметров системы нечеткого логического вывода типа Сугэно функция anfis может использовать метод обратного распространения ошибки, а также гибридный алгоритм, основанный на комбинации метода обратного распространения ошибки и метода наименьших квадратов. Функция anfis может использовать дополнительный аргумент для проверки модели – тестирующую выборку.

Функция anfis может имеет шесть входных аргументов:

trndata – идентификатор обучающей выборки. Этот входной аргумент является обязательным. Обучающая выборка представляет собой матрицу, каждая строчка которой является парой “входы-выход”. Последний столбец матрицы соответствует вектору значений выхода, а все остальные столбцы – входным данным. Заметим, что функцию anfis можно использовать для проектирования систем типа SISO (один вход - один выход) и типа MISO (много входов – один выход);

initfis – идентификатор исходной системы нечеткого логического вывода, функции принадлежности которой будут настроены с по-

мощью anfis. Исходная система нечеткого логического вывода должна быть системой типа Сугэно нулевого или первого порядка. Еще одним ограничением anfis является недопустимым использования типов функций принадлежностей и методов дефаззификации, определяемых пользователем. При отсутствии аргумента initfis функция anfis вызовет функцию genfis1, которая генерирует типовую систему нечеткого логического вывода. Эта типовая система будет использовать по две функции принадлежности гауссовского типа для каждой входной переменной. В случае, когда пользователь хочет получить систему с иным количеством функций принадлежности, необходимо задать в виде аргумента initfis количество функций принадлежности для оценки каждой входной переменной. Если количество функций принадлежности одинаково для всех входов, тогда достаточно указать только одно значение.

trnopt – вектор параметров настройки:

trnopt(1) – количество итераций (значение по умолчанию – 10);

trnopt(2) – допустимая ошибка обучения (значение по умолчанию – 0);

trnopt(3) – исходная длина шага (значение по умолчанию – 0.01);

trnopt(4) – коэффициент уменьшения длины шага (значение по умолчанию – 0.9);

trnopt(5) – коэффициент увеличения длины шага (значение по умолчанию – 1.1);

dispopt – вектор, указывающий какие промежуточные результаты обучения выводить в командное окно MATLAB во время настройки. Для вывода информации на экран необходимо установить соответствующую координату вектора в 1. Вектор ***dispopt*** имеет четыре координаты:

dispopt(1) – ANFIS-информация: количество функций принадлежности входных и выходной переменных и т.п.;

dispopt(2) – ошибка обучения;

dispopt(3) – длина шага, (выводится в случае его изменения);

dispopt(4) – окончательный результат.

По умолчанию значения всех координат равны 1;

chkdata – идентификатор тестирующей выборки. Тестирующая выборка используется для исследования свойства обобщения системы нечеткого логического вывода, т. е. способности системы выдавать правильные результаты для данных, отсутствующих в обучающей

выборке. Формат тестирующей выборки такой же как и обучающей. Обычно элементы тестирующей выборки несколько отличаются от элементов обучающей выборки. Тестирующая выборка особенно важна для задач обучения с большим количеством входов и (или) для задач с зашумленными данными. При формировании обучающей и тестирующей выборок необходимо стремится к обеспечению свойств представительности выборок;

optmethod – метод оптимизации, используемый для настройки. Допустимые значения: 1 – гибридный алгоритм и 0 – метод обратного распространения ошибки. По умолчанию применяется гибридный алгоритм, который использует метод обратного распространения ошибки (метод наискорейшего спуска) для настройки функций принадлежности входных переменных и метод наименьших квадратов для настройки функций принадлежности выхода. Точнее говоря, метод наименьших квадратов используется для нахождения коэффициентов линейных функций, связывающих входы и выход в каждом правиле. Метод по умолчанию используется всегда, за исключением случая, когда значение аргумента optmethod равно 0.

Функция anfis может имеет пять выходных аргументов:

fis – система нечеткого логического вывода, параметры которой настроены таким образом, что минимизируют расхождения между результатами логического вывода и экспериментальными данными из обучающей выборки;

error – вектор, содержащий значения ошибки обучения на каждой итерации. Ошибка обучения рассчитывается как среднее квадратическое отклонение между результатами нечеткого логического вывода и значениями выходной переменной из обучающей выборки;

stepsize – вектор, содержащий значения длины шага алгоритма оптимизации на каждой итерации;

chkfis – система нечеткого логического вывода, параметры которой настроены таким образом, что минимизируют расхождения между результатами логического вывода и экспериментальными данными из тестирующей выборки. Для получения такой системы необходимо задать пятый входной аргумент функции anfis – тестирующую выборку;

chkerror - вектор, содержащий значения ошибки тестирования на каждой итерации. Ошибка тестирования рассчитывается как среднее квадратическое отклонение между результатами нечеткого логи-

ческого вывода и значениями выходной переменной из тестирующей выборки. Для расчета ошибки тестирования необходимо задать пятый входной аргумент функции anfis – тестирующую выборку.

Процесс настройки прекращается по достижению требуемой ошибки обучения настройки или после выполнения указанного количества итераций.

Функция anfis может быть вызвана с одним входным аргументом – обучающей выборкой и одним выходным аргументом – системой нечеткого логического вывода, параметры которой настроены по обучающей выборке. При вызове функции anfis значения неинициализированых аргументов принимаются равными по умолчанию. При инициализации только части аргументов, например, первого и шестого – обучающей и тестирующей выборок, значения остальных аргументов (для нашего примера - второго, третьего, четвертого и пятого) необходимо задать как [] или NaN. Значение [] или NaN указывает, что соответствующий аргумент принимает значения, установленные по умолчанию. Значения, установленные по умолчанию могут быть изменены путем непосредственного редактирования файла anfis.m.

Пример.

Синтезируется и настраивается система нечеткого логического вывода fis, моделирующая зависимость y=sin(x) в диапазоне [0, 1].

```
x=(0:0.04:1)’;
y=sin(x);
trndata=[x y];
[fis, error, stepsize]=anfis(trndata)
```

CONVERTFIS

Преобразовывает систему нечеткого логического вывода,заданную в формате Fuzzy Logic Toolbox v.1 в формат Fuzzy Logic Toolbox v.2

Синтаксис: **FIS_new= convertis (FIS_old)**

Функция преобразовывает систему нечеткого логического, заданную матрицей FIS_old в формате Fuzzy Logic Toolbox v.1 в структуру FIS_new, в соответствии с форматом Fuzzy Logic Toolbox v.2.

DEFUZZ

Дефаззификация нечеткого множества

Синтаксис:**crisp = defuzz (x, mf, method)**

Выполняет операцию дефаззификации, т. е. преобразование нечеткого множества в четкое число. Функция defuzz имеет три входных аргумента:

x – универсальное множество, на котором задано нечеткое множество, подлежащее дефаззификации;

mf – вектор степеней принадлежности элементов множества x нечеткому множеству, подлежащему дефаззификации;

method – метод дефаззификации. Допустимые значения:

‘centroid’ – центр тяжести;

‘bisector’ – медиана;

‘mom’ – центр максимумов;

‘som’ - наименьший из максимумов;

‘lom’ - наибольший из максимумов.

Более подробно методы дефаззфикации описаны в разделе 1. Если метод дефаззификации отличается от вышеуказанных, тогда он должен быть представлен в виде **m**-функции. В этом случае значения аргументов x и mf будут переданы этой функции для выполнения дефаззификации.

Пример.

Проводится дефаззификация нечеткого множества с трапециевидной функцией принадлежности с параметрами [0, 2, 4, 10]);, заданного на универсальном множестве {0, 0.1, 0.2,…,10}.

```
x=0:0.1:10’;
mf=trapmf(x, [0, 2, 4, 10]);
crisp=defuzz(x, mf, ‘centroid’)
```

DISCFIS

Дискретизация функций принадлежностивсех термов, входящих в систему нечеткого логического вывода

Синтаксис: **[XI, YI, XO, YO, R] = discfis (fis, numPts)**

Функция используется для ускорения нечеткого логического вывода путем дискретизации функций принадлежности всех термов, входящих в систему нечеткого логического вывода. Функция discfis иметь два входных аргумента:

fis – система нечеткого логического вывода;

numPts – необязательный входной аргумент, задающий количество точек дискретизации функций принадлежности. Значение по умолчанию равно 181. Это означает, что все нечеткие множества

представляются в виде 181 пары чисел “элемент универсального множества – степень принадлежности”. При уменьшении точек дискретизации возрастает скорость выполнения логического вывода и уменьшается точность вычислений, и наоборот.

Функция discfis возвращает 5 выходных аргументов:

XI – матрица абцисс-координат функций принадлежностей термов входных переменных. Размер матрицы numPts x Ninp, где Ninp – количество термов, используемых для лингвистической оценки входных переменных;

YI – матрица степеней принадлежности элементов матрицы XI соответствующим термам. Размер матрицы numPts x Ninp, причем первый столбец содержит степени принадлежности первому терму первой входной переменной, а последний – последнему терму последней входной переменной;

XO – матрица абцисс-координат функций принадлежности термов выходных переменных. Размер матрицы numPts x Nout, где Nout – количество термов, используемых для лингвистической оценки выходных переменных;

YO – матрица степеней принадлежности элементов матрицы XO соответствующим термам. Размер матрицы numPts x Nout, причем первый столбец содержит степени принадлежности первому терму первой выходной переменной, а последний – последнему терму последней выходной переменной. Для системы типа Сугэно матрица YO является нулевой;

R – список правил базы знаний в индексном формате.

Пример:

fis = readfis (‘tipper’);

[XI, YI, XO, YO, R] = discfis (fis)

Дискретизация функций принадлежностей демонстрационной системы нечеткого логического вывода “Tipper”.

DSIGMF

Функция принадлежности в виде разности между двумя сигмоидными функциями

Синтаксис: **y = dsigmf(x, params)**

Функция принадлежности в виде разности между двумя сигмоидными функциями задается формулой . Применяется для задания гладних ассиметричных функций принадлежности.

Функция dsigmf имеет два входных аргумента:

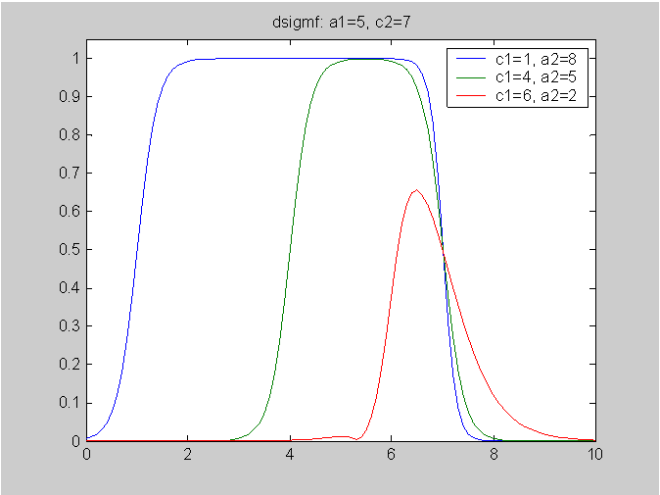
x – вектор, для координат которого необходимо рассчитать степени принадлежности;

params – вектор параметров функции принадлежности. Порядок задания параметров – [a1 c1 a2 c2].

Функция dsigmf возвращает выходной аргумент y, содержащий степени принадлежности координат вектора x.

Пример:

```
x = 0: 0.1: 10;
y1 = dsigmf (x, [5 1 8 7] );
y2 = dsigmf (x, [5 4 5 7] );
y3 = dsigmf (x, [5 6 2 7] );
plot (x, [y1; y2; y3])
title ('dsigmf: a1=5, c2=7')
ylim([0 1.05])
legend('c1=1, a2=8', 'c1=4, a2=5', 'c1=6, a2=2')
```



Построение графиков функций принадлежности в виде разности между двумя сигмоидными функциями с разными параметрами на интервале [0, 10].

EVALFIS

Выполнение нечеткого логического вывода

Синтаксис:**output = evalfis(input, fis)**

output = evalfis(input, fis, numPts)

[output, IRR, ORR, ARR] = evalfis(input, fis)

[output, IRR, ORR, ARR] = evalfis(input, fis, , numPts)

Выполняет нечеткий логический вывод. Функция evalfis может иметь три входных аргумента, первые два из которых обязательные:

input – матрица значений входных переменных, для которых необходимо выполнить нечеткий логический вывод. Матрица должна иметь размер M x N, где N – количество входных переменных; M – количество входных данных. Каждая строчка матрицы представляет один вектор значений входных переменных;

fis – идентификатор системы нечеткого логического вывода;

numPts – необязательный входной аргумент, задающий количество точек дискретизации функций принадлежности. Значение по

умолчанию равно 101. Это означает, что все нечеткие множества представляются в виде 101 пары чисел “элемент универсального множества – степень принадлежности”. При уменьшении точек дискретизации возрастает скорость выполнения логического вывода и уменьшается точность вычислений, и наоборот.

Функция evalfis может иметь четыре выходных аргумента:

output – матрица значений выходных переменных, получаемая в результате нечеткого логического вывода для вектора входных значений input. Матрица имеет размер M x L, где M – количество входных данных; L – количество выходных переменных в fis;

IRR – матрица размером NR x N, где NR – количество правил в fis; N – количество входных переменных. Матрица содержит степени принадлежности входных значений термам, входящих в базу знаний;

ORR – матрица размером numPts x (NR*L), где numPts – количество точек дискретизации; NR – количество правил в fis; L – количество выходных переменных в fis. Каждый столбец матрицы содержит функцию принадлежности выходной переменной, получаемую в результате вывода по одному правилу. Функция принадлежности дискретизируется на numPts точках и представляется в виде множества степеней принадлежности;

ARR – матрица размером numPts x L, где numPts – количество точек дискретизации; L – количество выходных переменных в fis. Матрица содержит функции принадлежности выходных переменных, получаемые в результате нечеткого логического вывода по всей базе знаний. Функции принадлежности дискретизируются на numPts точках и представляются в виде множества степеней принадлежности.

Аргументы IRR, ORR и ARR являются необязательными, они содержат промежуточные результаты нечеткого логического вывода. В случае задания нескольких входных данных значения аргументов IRR, ORR и ARR будут рассчитаны только для последнего вектора входных данных. Эти аргументы используются когда необходимо отследить процесс логического вывода или когда необходимо реализовать нестандартную процедуру нечеткого вывода.

Пример.

Первая строчка загружает демо-систему нечеткого логического вывода tipper, предназначенную для определения процента чаевых в ресторане. Вторая строчка рассчитывает размер чаевых, в случае если service=3 и food=8.

fis = readfis('tipper');
tip = evalfis([3 8], fis)

EVALMF

Вычисление значений произвольнойфункции принадлежности

Синтаксис:**y = evalmf (x, params, type)**

Описание:

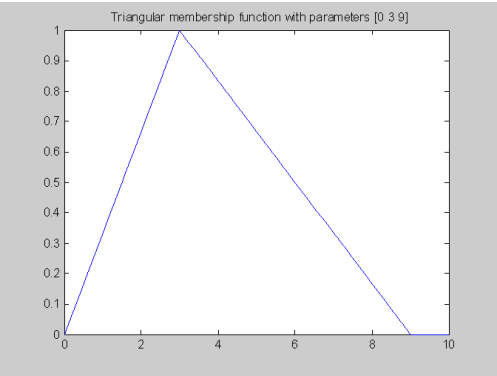
Позволяет вычислить значения произвольной функции принадлежности Функция evalmf иметь три входных аргумента:

x – вектор, для координат которого необходимо рассчитать степени принадлежности;

params – вектор параметров функции принадлежности, порядок задания которых определяется ее типом;

type – тип функции принадлежности. Значение типа функции принадлежности может быть задано в виде строчки символов или числом:

1 - 'trimf';
2 - 'trapmf';
3 - 'gaussmf';
4 – ‘gauss2mf';
5 - 'sigmf';
6 - 'dsigmf';
7 - 'psigmf';
8 - 'gbellmf';
9 - 'smf';
10 - 'zmf';
11 - 'pimf'.



При задании другого типа функции принадлежности предполагается, что она определена пользователем и задана соответствующим m-файлом.

Функция evalmf возвращает выходной аргумент y, содержащий степени принадлежности координат вектора x.

Пример:

x = 0: 0.1: 10;y = evalmf (x, [0 3 9], 1);plot (x, y)title (‘Triangular membership function with parameters [0 3 9]')

Построение графика треугольной функции принадлежности с параметрами [0 3 9] на интервале [0, 10].

EVALMMF

Вычисление степеней принадлежностей для нескольких функций принадлежностей

Синтаксис: **y = evalmmf (x, params, types)**

Описание:

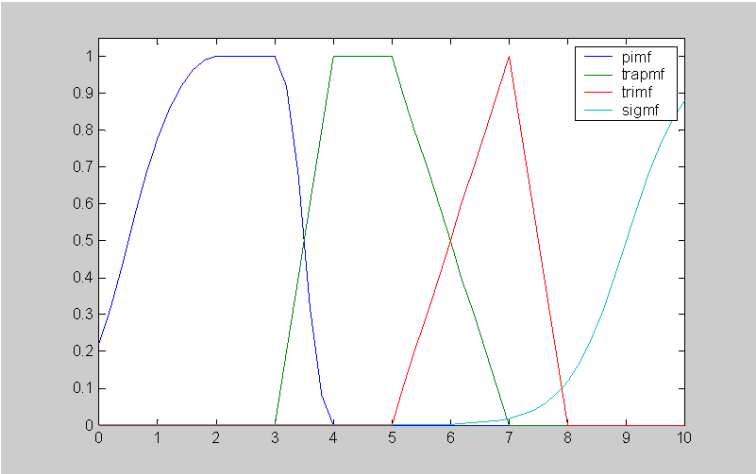
Позволяет вычислить значения нескольких функции принадлежности нечетких множеств, заданных на одном и том же универсальном множестве. Функция evalmmf иметь три входных аргумента:

x – вектор, для координат которого необходимо рассчитать степени принадлежности;

params – матрица параметров функции принадлежности. Первая строка матрицы определяет параметры первой функции принадлежности, вторая строка – параметры второй функции принадлежности и т.д.;

types – матрица типов функции принадлежности. Первая строка матрицы задает тип первой функции принадлежности, вторая строка – тип второй функции принадлежности и т.д. Значение типа функции принадлежности может быть задано в виде строчки символов или числом: 1 - 'trimf'; 2 - 'trapmf'; 3 - 'gaussmf'; 4 – ‘gauss2mf'; 5 - 'sigmf'; 6 - 'dsigmf'; 7 - 'psigmf'; 8 - 'gbellmf'; 9 - 'smf'; 10 - 'zmf'; 11 - 'pimf'. При задании другого типа функции принадлежности предполагается, что она определена пользователем и задана соответствующим m-файлом.

Функция evalmmf возвращает матрицу y, содержащую степени принадлежности координат вектора x. Первая строка матрицы содержит значения первой функции принадлежности, вторая строка – значения второй функции принадлежности и т.д.



Пример:

```
x = 0:0.2:10;para = [-1 2 3 4; 3 4 5 7; 5 7 8 0; 2 9 0 0];type = str2mat('pimf', 'trapmf', 'trimf', 'sigmf');mf = evalmmf(x, para, type);
plot(x', mf');
ylim([0 1.05])
legend('pimf', 'trapmf', 'trimf', 'sigmf');
```

Построение графиков подобной, трапециевидной, треугольной и сигмоидной функций принадлежности на интервале [0, 10].

FCM

Кластеризация на основе нечеткого c-means алгоритма

Синтаксис: **[V, M, obj_fcn] = fcm(X, c)**

[V, M, obj_fcn] = fcm(X, c, options)

Описание:

На основе нечеткого c-means алгоритма выполняет кластеризацию данных. Этот алгоритм кластеризации предложил Джеймс Бэздэк (James Bezdek) в 1981 году.

Задача нечеткой кластеризации ставится следующим образом.

Дано: $X_k = (X_1, X_2,\ldots X_p)$

– объекты, подлежащие кластеризации (n – количество объектов). Каждый объект представляет собой точку в p-мерном пространстве признаков ();

$X_k = (X_1, X_2,\ldots X_p)$

c – количество кластеров ().

Необходимо каждому элементу множества X поставить в соответствие степени принадлежности к классам.

Элементы одного кластера должны быть так близки каждый каждому, как это только возможно, и, одновременно, кластеры должны быть на наибольшем удалении друг от друга. Для обеспечения управляемости процесса кластеризации необходимо использовать меру близости, в качестве которой обычно определяют расстояние между двумя объектами (точками в p-мерном пространстве) и в виде вещественной функции.

Дополнительно, если функция удовлетворяет правилу треугольника, тогда эта функция является метрикой, хотя выполнения этого свойства не всегда необходимо для задач кластеризации.

Любое разбиение множествана нечеткие подмножества () может быть полностью описано функцией принадлежности.

Обозначим через степень принадлежности объекта к подмножеству и через множество всех действительных матриц размером. Тогда нечетким c-разбиением (или матрицей степеней принадлежности) называется матрица при выполнении следующих условий:

$\mu_{ik} \in [0,1]$

$$\sum_{i=1}^{c} \mu_{ik} = 1$$

$$\sum_{k=1}^{n} \mu_{ik} \in (0, n)$$

В отличие от четкого, при нечетком с-разбиении любой объект одновременно принадлежит к различным кластерам, но с разной степенью. Условия (2) и (3) требуют только, чтобы сумма степеней принадлежности объекта ко всем кластерам была нормализована к 1, а также, чтобы количество кластеров, к которым принадлежит объект, не превышало .

Обозначим центры кластеров, т.е. точки в р-мерном пространстве, вокруг которых сконцентрированы соответствующие объекты. При использовании эвклидового расстояния задача нечеткой кластеризации состоит в нахождении такой матрицы степеней принадлежностии таких координат центров кластеров, которые обеспечивают минимум следующего критерия:

$$\sum_{i=1}^{c}\sum_{k=1}^{n}(\mu_{ik})^{m} \cdot \|X_k - V_i\|^2 \to \min$$

$$V_i = \frac{1}{\sum_{k=1}^{n}\mu_{ik}} \sum_{k=1}^{n}(\mu_{ik})^{m} \cdot X_k$$

где i=$\overline{1, c}$

центр i-го кластера, так называемый экспоненциальный вес (m>=1).

Значение экспоненциального веса устанавливается до начала кластеризации. Экспоненциальный весвлияет на матрицу степеней принадлежности. Чем больше , тем конечная матрица с-разбиения становится более “размазанной”, и приона примет вид , что является очень плохим решением, т. к. все объекты принадлежат ко всем кластерам с одной и той же степенью. Также, экспоненциальный вес позволяет при формировании координат центров кластеров усилить влияние объектов с большими значениями степеней принадлежности и уменьшить влияние объектов с малыми значениями степеней принадлежности. На сегодня не существует теоретически обоснованного правила выбора значения . Обычно устанавливают .

Аналитического решения задачи нахождения оптимальных координат центров кластеров и матрицы степеней принадлежности не

существует, поэтому она решается численно. Один из итерационных алгоритмов решения этой задачи реализован в функции fcm.

Функция fcm может иметь три входных аргумента:

1. X – матрица, представляющая данные, подлежащие кластеризации. Каждая строка матрицы соответствует одному объекту (образу);
2. c – количество кластеров, которое должно быть получено в результате выполнения функции fcm. Количество кластеров должно быть больше 1 и меньше числа образов, заданных матрицей X;
3. options – необязательный аргумент, устанавливающий параметры алгоритма кластеризации:

options(1) – значения экспоненциального веса (значение по умолчанию – 2.0);

options(2) – максимальное количество итераций алгоритма кластеризации (значение по умолчанию – 100);

options(3) – минимально допустимое значение улучшения целевой функции за одну итерацию алгоритма (значение по умолчанию – 0.00001);

options(4) – вывод промежуточных результатов во время работы функции fcm (значение по умолчанию – 1).

Для использования значений по умолчанию можно вести NaN в качестве значения соответствующей координаты вектора options.

Алгоритм кластеризации останавливается когда выполнено максимальное количество итераций или когда улучшение значения целевой функции за одну итерацию меньше указанного минимально допустимого значения.

Функция fcm имеет три выходных аргумента:

1. V – матрица координат центров кластеров, полученных в результате кластеризации. Каждая строка матрицы соответствует центру одного кластера;

2. M – матрица степеней принадлежности образов к кластерам. Каждая строка матрицы соответствует функции принадлежности одного кластера.

3. obj_fcn – вектор значений целевой функции на каждой итерации алгоритма кластеризации.

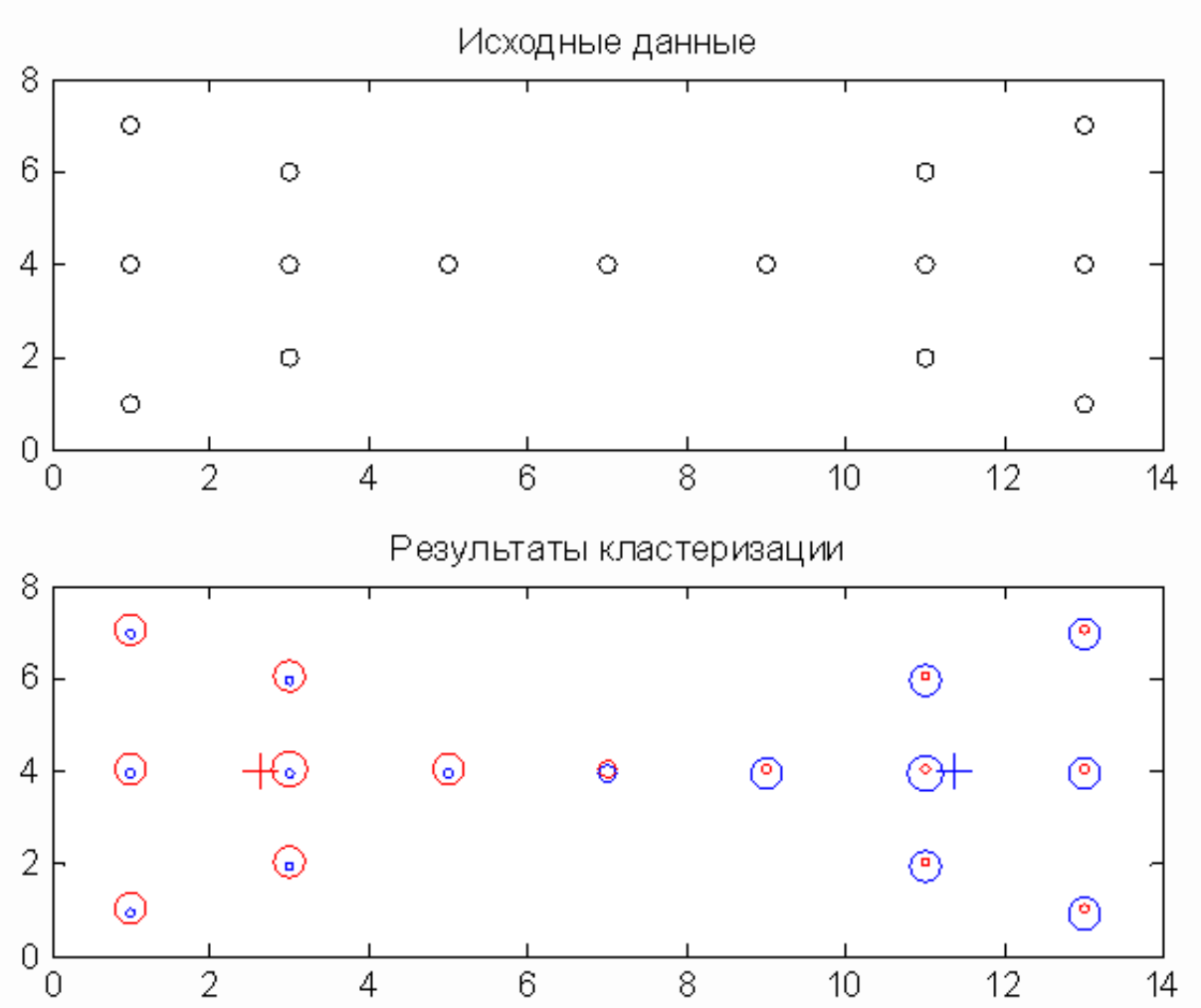
Примечание: при описании нечеткого c-means алгоритма использована книга Zimmermann H.-J. Fuzzy Set Theory - and Its Applications.3rd ed.- Kluwer Academic Publishers, 1996.- 435p.

Пример.

Проводится кластеризация объектов, образующих фигуру типа “бабочка”. Результаты кластеризации приведены ниже на рисунке. Центры кластеров указаны маркером ‘+’. Размер маркера ‘o’ пропорционален степени принадлежности объекта кластеру. Расположенный в центре объект имеет одинаковые степени принадлежности к красному и к синему кластерам. Это обеспечивает симметричное разбиение объектов по кластерам, что невозможно при четкой кластеризации.

```
X=[1 1; 1 4; 1 7; 3 2; 3 4; 3 6; 5 4; 7 4; 9 4; 11 2; 11 4; 11 6; 13 1; 13 4; 13 7];
[V M opt]=fcm(X, 2)
subplot(2,1,1);
plot(X(:,1), X(:,2), 'ko', 'markersize', 6)
text='Исходные данные';
title(text)
xlim([0 14])
ylim([0 8])
subplot(2,1,2);
plot(V(1,1), V(1,2), 'r+', 'markersize', 10)
hold on
plot(V(2,1), V(2,2), 'b+', 'markersize', 10)
        for i=1:15
        plot(X(i,1), X(i,2)+.05, 'ro', 'markersize', M(1,i)*8+2);
        plot(X(i,1), X(i,2)-.05, 'bo', 'markersize', M(2,i)*8+2);
        end
 xlim([0 14])
```

```
ylim([0 8])
text='Результаты кластеризации';
title(text)
```



FINDROW

Нахождение строки в матрице, совпадающей с входной строкой

Синтаксис: **rownum = findrow (str, strmat)**

Описание:

Функция findrow возвращает номер строки матрицы strmat, содержащую стринг, заданный аргументом str. Если таких строк несколько, то функция findrow возвратит номера всех строчек. Функция findrow не связана с нечеткими множествами и нечеткой логикой, она используются для выполнения алгоритмов обработки информации, необходимых для других функций Fuzzy Logic Toolbox.

Пример:

```
rownum = findrow('два ', ['один'; 'два '; 'три '])
```

Функция возвращает значения 2, которое означает, что стринг 'два ' содержится во второй строке матрицы ['один'; 'два '; 'три

FSTRVCAT

Конкатенация матрицразличных размеров

Синтаксис: **aout = fstrvcat(a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11)**

Описание:

Функция fstrvcat формирует матрицу aout, строки которой содержат вектора a1, a2, a3, …. Для того, чтобы сформировать корректную матрицу, к некоторым векторам добавляется необходимое количество нулевых символов. Количество входных аргументов не должно превышать 11. Входные аргументы функции fstrvcat могут быть заданы как векторами, так и матрицами. Это позволяет формировать выходную матрицу aout пратически любого размера.

Функция fstrvcat не связана с нечеткими множествами и нечеткой логикой, она используются для выполнения алгоритмов обработки информации, необходимых для формирования fis-структуры.

Пример:

aout = fstrvcat ('hi', 'blue', [110 101 101 100 108 101])

Функция fstrvcat возвращает следующую матрицу:

aout =

1041050000

9810811710100

110101101100108101,

строки которой содержат значения входных аргументов в формате ASCII.

FUZARITH

Выполнение нечетких арифметических операций

Синтаксис:**c = fuzarith(x, a, b, operator)**

Описание:

Выполняет арифметические операции сложения, вычитания, умножения и деления над нечеткими числами.

Функция fuzarith имеет четыре входных аргумента:

x – универсальное множество, на котором заданны нечеткие числа;

a – вектор, задающий первый операнд. Представляет собой вектор степеней принадлежности элементов универсального множества

первому нечеткому множеству. Другими словами, аргументы x и a образует первое нечеткое число;

b – вектор, задающий второй операнд. Представляет собой вектор степеней принадлежности элементов универсального множества второму нечеткому множеству, т.о. аргументы x и b образует второе нечеткое число;

operator – арифметическая операция:

'sum' – сложение;

'sub' – вычитание;

'prod' – умножение;

'div' – деление.

Выходной переменной функции fuzarith является вектор степеней принадлежности элементов универсального множества x результату выполнения нечеткой арифметической операции. Размерности векторов x, a, b и c должны быть одинаковыми.

В функции fuzarith нечеткие арифметические функции выполняются по следующему алгоритму:

- преобразование нечетких чисел-операндов в α-уровневые нечеткие множества;
- выполнения арифметической операции для каждого α-уровня в соответствии с принципом обобщения;
- преобразование результирующего нечеткого числа из -уровнего представления к традиционному виду.

Функция fuzarith использует стандартные процедуры интерполяции для выполнения указанных выше преобразований. Количество α-уровней равно 101.

Обратим внимание на то, что операция деления не может быть выполнена если универсальное множество содержит как отрицательные, так и положительные числа, что связано с делением на ноль.

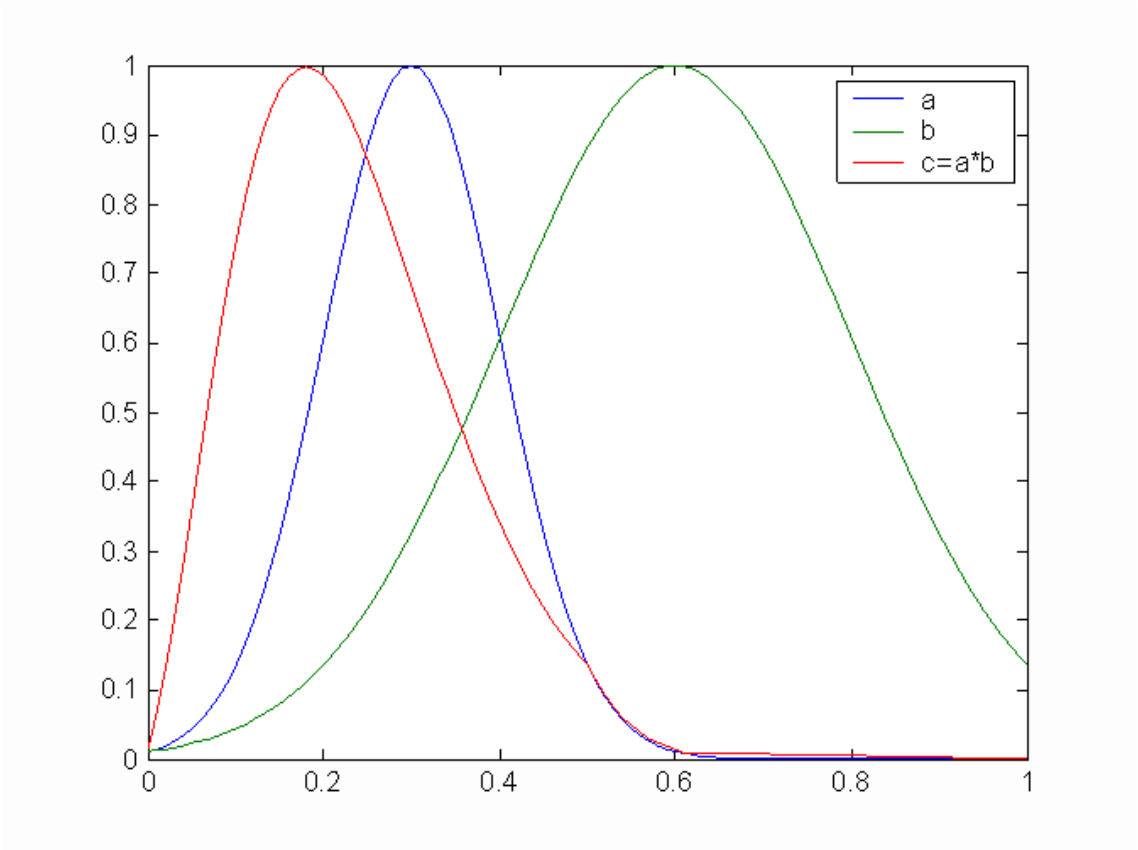
При выполнении арифметически операций над нечеткими числами, как правило, носитель результирующего нечеткого числа отличен от носителей нечетких операндов. Функция fuzarith выводит степени принадлежности результата только для универсального множества нечетких операндов, т.е. для множества x. Для вывода всего результирующего нечеткого числа необходима простая модификация функции fuzarith – вывод результата в виде нечеткого множества в α-уровневом разложении. Результирующее нечеткое число в виде раз-

ложений по α-уровням представлено в функции fuzarith переменной intervalC.

Пример:

Рассчитывается нечеткое число **c** как произведение нечетких чисел a и b с гауссовскими функциями принадлежности, заданных на универсальном множестве {0, 0.01, …, 1}. Графики функций принадлежности показаны ниже на рисунке.

```
x=0:0.01:1;
 a=gaussmf(x, [0.1 0.3]);
b=gaussmf(x, [0.2 0.6]);
 c=fuzarith(x, a, b, 'prod');
plot(x, a, x, b, x, c)
legend('a', 'b', 'c=a*b')
```



ОГЛАВЛЕНИЕ

Учебное издание

ВЕСЕЛОВ Олег Вениаминович

НЕЧЕТКАЯ ЛОГИКА И НЕЙРОННЫЕ СЕТИ
В СИСТЕМАХ УПРАВЛЕНИЯ И ДИАГНОСТИКЕ

Учебное пособие

Издается в авторской редакции

Подписано в печать 20.06.23.
Формат 60×84/16. Усл. печ. л. 16,74. Тираж 40 экз.
Заказ
Издательство
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых.
600000, Владимир, ул. Горького, 87.