

Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет инновационного непрерывного образования

Кафедра электронных вычислительных машин

К ЗАЩИТЕ ДОПУСТИТЬ  
Зав. каф. ЭВМ  
\_\_\_\_\_ Д.И. Самаль

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
к дипломному проекту  
на тему  
«ПРОГРАММНОЕ СРЕДСТВО ДЛЯ ОТЛАДКИ АЛГОРИТМОВ  
НЕЧЕТКОГО УПРАВЛЕНИЯ МОБИЛЬНЫМИ РОБОТАМИ»

БГУИР ДП 1–40 02 01 01 322 ПЗ

Студент	М.М. Шавердо
Руководитель	М.М. Татур
Консультанты:	
от кафедры ЭВМ	М.М. Татур
по экономической части	Т.А. Рыковская
Нормоконтролер	А.С. Сидорович
Рецензент	

МИНСК 2018

## РЕФЕРАТ

Дипломный проект представлен следующим образом. Электронные носители: 1 компакт-диск. Чертежный материал: 6 листов формата A1. Пояснительная записка: 86 страниц, 24 рисунков рисунков, 10 литературных источников, 4 приложения.

Ключевые слова: нечеткий контроллер, алгоритм Мамдани, робот, лингвистическая переменная, терм, моделирование, Python, CLI.

Предметной областью являются нечеткие контроллеры, алгоритмы их моделирования и настройки. Объектом разработки является программное средство, позволяющее настраивать и обучать нечеткие контроллеры мобильных роботов.

Целью разработки является создание программного средства, которое позволит проводить эксперименты с нечеткими контроллерами мобильных роботов в моделируемой среде, позволяющее существенно снизить трудоемкость и стоимость исследования и отладки нечетких алгоритмов.

Для разработки использовался язык Python, пакет для визуализации научной графики matplotlib, пакетный менеджер pip.

В результате работы над проектом было разработано программное средство, позволяющее отлаживать алгоритмы нечеткого управления мобильными роботами. Оно облегчает и автоматизирует настройку нечетких алгоритмов управления мобильными роботами.

Областью практического применения данного программного средства является его использование в отладке мобильных платформ предприятиями-изготовителями, проведение теоретических исследований нечетких алгоритмов в научно-исследовательских институтах, использование в преподавательской деятельности.

Разработанный проект является экономически эффективным как для разработчика, так и для конечного пользователя: он позволяет существенно сократить материальные и трудовые затраты на отладку нечетких алгоритмов.

Дипломный проект является завершенным, поставленная задача решена в полной мере, присутствует возможность дальнейшего развития приложения и расширения его функционала.

Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет: ФЗО. Кафедра: ЭВМ.

Специальность: 40 02 01 «Вычислительные машины, системы и сети».

Специализация: 40 02 02 01 «Проектирование и применение локальных компьютерных сетей».

УТВЕРЖДАЮ

Заведующий кафедрой ЭВМ

\_\_\_\_\_ Д.И. Самаль

«\_\_\_» \_\_\_\_\_ 2018 г.

**ЗАДАНИЕ**

по дипломному проекту студента  
Шавердо Михаила Михайловича

- 1 Тема проекта: «Программное средство для отладки алгоритмов нечеткого управления мобильными роботами» – утверждена приказом по университету от 12 февраля 2018 г. № 263-с
- 2 Срок сдачи студентом законченного проекта: 01 июня 2018 г.
- 3 Исходные данные к проекту:
  - 3.1 Язык программирования: Python.
  - 3.2 Формат выгружаемых данных: CSV .
  - 3.3 Операционная система: Linux, Windows, OS X
  - 3.4 Среда разработки: PyCharm.
- 4 Содержание пояснительной записки (перечень подлежащих разработке вопросов):

Введение. 1. Аналитический обзор научно-технического уровня по теме дипломного проекта. 2. Системное проектирование. 3. Техническая реализация и верификация проекта. 4. Рекомендации по практическому использованию программного средства. 5. Техничко-экономическое обоснование разработки программного средства для отладки алгоритмов нечеткого управления мобильными роботами. Заключение. Список использованных источников. Приложения.
- 5 Перечень графического материала:
  - 5.1 Вводный плакат. Плакат

- 5.2** Алгоритмы нечеткого вывода. Плакат.
- 5.3** Программное средство для отладки алгоритмов нечеткого управления мобильными роботами. Схема структурная.
- 5.4** Программное средство для отладки алгоритмов нечеткого управления мобильными роботами. Схема программы.
- 5.5** Программное средство для отладки алгоритмов нечеткого управления мобильными роботами. Диаграмма последовательности.
- 5.6** Результаты работы программы. Плакат.

**6** Содержание задания по экономической части: «Технико-экономическое обоснование разработки программного средства для отладки алгоритмов нечеткого управления мобильными роботами».

- 6.1** Затраты на разработку программного средства.
- 6.2** Расчет показателей эффективности использования программного средства.

ЗАДАНИЕ ВЫДАЛ

Т.А. Рыковская

### КАЛЕНДАРНЫЙ ПЛАН

Наименование этапов дипломного проекта	Объем этапа, %	Срок выполнения этапа	Примечания
Подбор и изучение литературы	10	26.01 – 15.02	
Структурное проектирование	10	1.02 – 22.03	
Функциональное проектирование	20	23.03 – 30.03	
Разработка программных модулей	30	30.03 – 22.04	
Программа и методика испытаний	10	22.04 – 4.05	
Расчет экономической эффективности	10	4.05 – 18.05	
Завершение оформления пояснительной записки	10	18.05 – 01.06	

Дата выдачи задания: 26.01.2018

Руководитель

М.М. Татур

ЗАДАНИЕ ПРИНЯЛ К ИСПОЛНЕНИЮ \_\_\_\_\_

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ . . . . .	6
1 АНАЛИТИЧЕСКИЙ ОБЗОР НАУЧНО-ТЕХНИЧЕСКОГО УРОВНЯ ПО ТЕМЕ ДИПЛОМНОГО ПРОЕКТА . . . . .	7
1.1 Типовые схемы включения нелинейных регуляторов в системах управления мобильными роботами . . . . .	7
1.2 Виды нелинейных регуляторов . . . . .	8
1.3 Классические алгоритмы нечеткого вывода. Мамдани и Сугено Тагаки . . . . .	11
1.4 Проблема разработки, тестирования и отладки алгоритмов управления . . . . .	13
2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ . . . . .	14
2.1 Постановка задачи на разработку программного средства . . .	14
2.2 Разработка обобщенной модели . . . . .	15
2.3 Методика тестирования и отладки алгоритмов нечеткого управления; алгоритм настройки нечеткого контроллера . . . .	17
2.4 Структурная схема экспериментального программного обеспечения . . . . .	17
3 ТЕХНИЧЕСКАЯ РЕАЛИЗАЦИЯ И ВЕРИФИКАЦИЯ ПРОЕКТА .	24
3.1 Разработка ПО . . . . .	24
3.2 Верификация проекта и анализ полученных результатов . . . .	64
4 РЕКОМЕНДАЦИИ ПО ПРАКТИЧЕСКОМУ ИСПОЛЬЗОВАНИЮ ПРОГРАММНОГО СРЕДСТВА . . . . .	70
4.1 Разработка инструкции (руководства) пользователя . . . . .	70
4.2 Разработка инструкции (руководства) программиста . . . . .	73
5 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО СРЕДСТВА ДЛЯ ОТЛАДКИ АЛГОРИТМОВ НЕЧЕТКОГО УПРАВЛЕНИЯ МОБИЛЬНЫМИ РОБОТАМИ . . . . .	75
5.1 Описание функций, назначения и потенциальных пользователей	75
5.2 Расчёт затрат на разработку ПС . . . . .	76
5.3 Оценка результата от использования ПС . . . . .	78
5.4 Расчет показателей эффективности использования программного средства . . . . .	79
ЗАКЛЮЧЕНИЕ . . . . .	81
ПРИЛОЖЕНИЕ А Исходный текст модуля Fuzzysim.Fuzzysim . . . .	83

## ВВЕДЕНИЕ

Из глубины веков доходят до нас легенды о колдунах и магах, но, согласно третьему закону А. Кларка, «любая достаточно развитая технология неотличима от магии». Таким образом, в настоящее время магию современности создают автоматизированные системы управления, яркими представителями которых являются нечеткие контроллеры.

Нечеткие контроллеры обеспечивают эффективное управление процессами. Они предлагают использовать понятное эксперту представление, которое облегчает передачу знаний между системой и пользователями.

К нечетким контроллерам при проектировании предъявляется целый ряд технических и экономических требований, главными из которых являются:

- обеспечить заданное качество управления;
- обеспечить управление в реальном времени;
- минимизировать затраты применяемой аппаратной платформы.

Первое требование может характеризоваться такими свойствами, как плавность изменения регулируемого параметра, устойчивость от шумов и флуктуаций, адаптивность, и др., в зависимости от конкретной задачи.

Любое системное управление предполагает выделение ресурсов и времени на выполнение задач таким образом, чтобы выполнялись определенные требования к производительности: для заданного объекта управления (исполнительного устройства) необходимо выполнить алгоритм управления установленное количество раз в единицу времени.

Построение системы управления, в которой выдерживаются все вышеперечисленные требования является нетривиальной задачей и неразрывно связано с непрерывной оценкой всех трех параметров в процессе разработки системы. Использование реального аппаратного обеспечения для контроля качества управления и измерения требуемых ресурсов является трудоемкой, дорогой, а подчас и невозможной задачей.

В связи с этим, целью данного дипломного проекта является разработка и реализация системы, позволяющей отлаживать алгоритмы нечеткого управления в симулируемой среде, что позволит существенно упростить и удешевить процесс разработки и отладки алгоритмов управления по сравнению с натурными испытаниями.

В соответствии с поставленной целью были определены следующие задачи:

- выбор средств разработки реализации системы;
- разработка и реализация симуляции физической среды;
- реализация инструментов выгрузки телеметрии симулируемой среды;
- разработка средств визуализации данных телеметрии;
- разработка алгоритма настройки нечеткого контроллера.

# 1 АНАЛИТИЧЕСКИЙ ОБЗОР НАУЧНО-ТЕХНИЧЕСКОГО УРОВНЯ ПО ТЕМЕ ДИПЛОМНОГО ПРОЕКТА

## 1.1 Типовые схемы включения нелинейных регуляторов в системах управления мобильными роботами

Рассмотрим типовую функциональную схему САУ [1]:

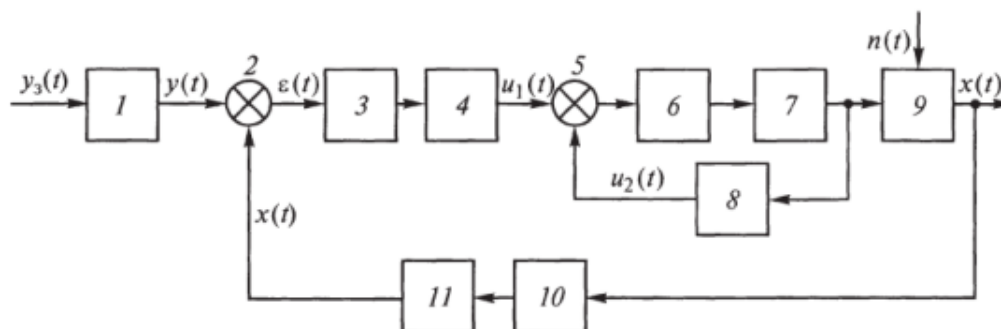


Рисунок 1.1 – Типовая функциональная схема САУ: 1 – задающее устройство; 2, 5 – сравнивающие устройства; 3 – преобразующее устройство; 4, 8 – корректирующие устройства; 6 – усилительное устройство; 7 – исполнительное устройство; 9 – объект управления; 10 – измерительный элемент; элемент главной ОС.

Задающее устройство 1 преобразует воздействие в сигнал, а сравнивающее устройство 2 в результате сравнения сигнала и регулируемой величины (предполагается, что элементы 10 и 11 не искажают сигнал) вырабатывает сигнал ошибки. Сравнивающие устройства 2, 5 также называют датчиками ошибки, отклонения, рассогласования.

Преобразующее устройство 3 служит для преобразования одной физической величины в другую, более удобную для использования в процессе управления (во многих системах преобразующее устройство отсутствует).

Регуляторы 4, 8 служат для обеспечения заданных динамических свойств замкнутой системы. С их помощью обеспечивается высокая точность ее работы в установившемся режиме, а также демпфируются сильные колебательные процессы (например, летательных аппаратов). Более того, введение в систему регулятора позволяет устранить незатухающие или возрастающие колебания управляемой величины. Иногда регуляторы вырабатывают управляющие сигналы (команды) в зависимости от возмущающих воздействий, что существенно повышает качество работы систем, увеличивая их точность.

Из приведенной на рис. 1.1 схемы САУ видно, что в хорошо спроектированной системе ошибка очень мала, в то время как на управляемый объект должны поступать воздействия с мощностью, достаточной для питания двигателя. В связи с этим важным элементом

САУ является усилительное устройство 6, предназначенное для повышения мощности сигнала ошибки, т.е. управления энергией, поступающей от постороннего источника. На практике широко используются электронные, магнитные, гидравлические и пневматические усилители.

Исполнительное устройство 7 предназначено для влияния на управляющий орган 9, подвергающийся воздействию внешних полей. Исполнительные устройства могут быть пневматические, гидравлические и электрические, которые подразделяются, в свою очередь, на электромоторные и электромагнитные.

Чувствительный (измерительный) элемент — датчик 10 — необходим в САУ для преобразования управляемых переменных в сигналы управления (например, угла в напряжение).

Элемент, который подвергается управлению, является объектом управления. При проектировании объектом управления считают всю неизменяемую часть САУ (т.е. все элементы, кроме регулятора). Это могут быть электрическая печь для закаливания металла, самолет, ракета, космический аппарат, двигатель, ядерный реактор, станок для обработки металла и т. д. В связи с большим разнообразием объектов управления разными могут быть и управляемые переменные: напряжение, число оборотов, угловое положение, курс, мощность и т.д.

## 1.2 Виды нелинейных регуляторов

Автоматический регулятор в системе регулирования состоит, как уже известно, из трех основных частей: измерительной, усилительно-преобразовательной и исполнительной. В усилительно-преобразовательной части имеются корректирующие устройства, в которых, помимо сигнала отклонения  $x$  регулируемой величины, образуется сигнал по первой производной. Закон, по которому формируется регулирующее воздействие и на объект из первичных сигналов называется законом регулирования. Математически закон регулирования определяется уравнением автоматического регулятора. Различают линейные и нелинейные законы регулирования.

Линейные законы регулирования определяются линейным уравнением регулятора и не рассматриваются в данной работе.

Что же касается нелинейных законов регулирования, то (за исключением релейного) они изучены мало. Очевидно, однако, что использование нелинейных законов регулирования, определяемых разнообразными нелинейными уравнениями регулятора значительно расширяет возможности целесообразного изменения качества процессов регулирования и точности. Это ясно из общих принципиальных соображений, так как область нелинейных уравнений значительно богаче и разнообразнее, чем линейных.

Введем следующую классификацию нелинейных законов



регулирования [2]:

- функциональные нелинейные законы регулирования,
- логические нелинейные законы регулирования,
- оптимизирующие нелинейные законы регулирования,
- параметрические нелинейные законы регулирования.

Важным отличием нелинейных законов от линейных является то, что они придают системе регулирования принципиально новые свойства. Если при линейном законе всегда вырабатывается сигнал, пропорциональный входной переменной или ее производной и т. д., то при нелинейном законе может существенно изменяться сам характер действия системы управления на объект в зависимости от величины входного воздействия. Другими словами, если для линейных систем изменение размера отклонения — это изменение только масштаба, но не формы процессов, то в нелинейной системе при этом может существенно изменяться и форма процессов, вплоть до принципиальных качественных изменений картины процессов. Эти особые свойства нелинейных законов можно выгодно использовать в технике автоматического управления и регулирования.

Рассмотрим отдельно каждый из указанных четырех классов нелинейных законов регулирования.

*Функциональные нелинейные законы регулирования.* Функциональными будем называть такие нелинейные законы регулирования, при которых регулирующее воздействие на объект выражается в виде нелинейной функции от отклонения регулируемой величины, представляющей собой входную информацию для системы регулирования.

Данный класс может содержать в себе как статические, так и динамические нелинейности.

В отличие от линейного закона, здесь в первом случае будет более энергичное действие регулятора при больших отклонениях  $x$  и большой запас устойчивости установившегося режима. Во втором случае будет менее энергичное, но более плавное действие регулятора вначале и повышенная точность в установившемся режиме, хотя и с меньшим запасом устойчивости. Однако такого рода рекомендации, как увидим в дальнейшем, справедливы для большинства систем, но все же не для всех. Поэтому они требуют специального обследования для каждого объекта регулирования.

Нелинейный закон регулирования за счет дополнительных нелинейных обратных связей может включать в себя также нелинейности от выходной величины, что расширяет возможности целесообразного изменения качества процесса регулирования.

Отметим, что функциональные нелинейные законы регулирования могут быть связаны не только с изменением параметров в зависимости от размеров входных воздействий, но и с изменением структуры. Например, при увеличении отклонения регулируемой величины сверх определенного порога в системе может происходить переключение с одного линейного корректирующего устройства на другое.

*Логические нелинейные законы регулирования.* Нелинейные законы регулирования могут иметь иные формы, которые реализуются с помощью не функциональных, а более или менее сложных логических устройств. Будем называть их логическими нелинейными законами регулирования [2].

Логические нелинейные законы регулирования могут быть связаны также с изменением структуры системы регулирования. Например, при помощи логического устройства можно включать и выключать сигналы управления по первой и второй производным и по интегралу, в зависимости от сочетания значений отклонения регулируемой величины  $x$  и скорости отклонения. Если правильно сформировать логику этих переключений, то можно существенно повысить качество работы системы регулирования.

Вместо комбинирования указанных линейных членов закона регулирования могут вводиться также и функциональные нелинейные члены; включение и выключение сигналов, соответствующих этим членам, производится при помощи логического устройства. Тогда получится комбинация функциональных и логических нелинейных законов регулирования.

*Оптимизирующие нелинейные законы регулирования.* В настоящее время интенсивно развивается теория оптимальных процессов регулирования. При этом на основе классических вариационных методов, или на основе так называемого принципа максимума, или методом динамического программирования определяется закон регулирования таким образом, чтобы система имела максимум быстродействия, или минимум ошибки, или же минимум какой-нибудь другой величины (в форме функционала) с учетом ограничений, накладываемых в реальной системе на координаты, скорости, силы и т. д. Как правило, при этом приходят к нелинейным законам регулирования, хотя, вообще говоря, можно оптимизировать и коэффициенты линейного закона, задав его форму. Часто оптимальный нелинейный закон регулирования состоит в переключении управляющего воздействия (при определенных состояниях системы) с одного максимально возможного значения на другое (противоположного знака). Моменты переключения в целом определяются сложными комбинациями значений нескольких переменных и их производных.

*Параметрические нелинейные законы регулирования.* В предыдущих типах законов регулирования вводились отклонения регулируемой величины от некоторых заданных ее программных значений. При параметрической программе управления закон регулирования может выражаться в виде нелинейной функции текущих координат, в которых задается параметрическая программа.

Нелинейные законы регулирования обладают богатыми возможностями во всех случаях, когда требуемый эффект может быть достигнут изменением свойств системы с изменением величин ошибок. Важным классом нелинейных систем являются системы с переменной структурой. Большими возможностями обладают так называемые адаптивные, т. е.

самонастраивающиеся и самоорганизующиеся, системы, примером которых служат алгоритмы нечеткой логики.

### 1.3 Классические алгоритмы нечеткого вывода. Мамдани и Сугено Тагаки

Нечеткая логика является обобщением классической формальной логики, подобно тому как нечеткие множества являются обобщением для классических множеств. И нечеткая логика и нечеткие множества были представлены Лотфи Заде в 1965 году. Главной предпосылкой их появления было наличие приближенных рассуждений при описании человеком объектов и процессов реального мира. Признание теории нечеткого управления заняло не один десяток лет. При этом можно выделить три периода данного процесса:

- Первый период, конец шестидесятых годов. Происходит бурное развитие теории нечетких множеств.

- Второй период, семидесятые и восьмидесятые годы. Появляются наработки в области практики управления комплексными системами, такими как паровой двигатель. Одновременно стало уделяться внимание вопросам построения экспертных систем, построенных на нечеткой логике, разработке нечетких контроллеров. Нечеткие экспертные системы для поддержки принятия решений находят широкое применение в медицине и экономике.

- Третий период, с конца восьмидесятых по настоящее время. Появляются пакеты программ для построения нечетких экспертных систем, а области применения нечеткой логики заметно расширяются. Она применяется в автомобильной, аэрокосмической и транспортной промышленности, в области изделий бытовой техники, в сфере финансов, анализа и принятия управленческих решений и многих других.

Триумфальное шествие нечеткой логики по миру началось после доказательства в конце 80-х Бартоломеем Коско знаменитой теоремы FAT (Fuzzy Approximation Theorem). В бизнесе и финансах нечеткая логика получила признание после того как в 1988 году экспертная система на основе нечетких правил для прогнозирования финансовых индикаторов единственная предсказала биржевой крах. И количество успешных фаззи-применений в настоящее время исчисляется тысячами.

*Алгоритм Мамдани* (Mamdani) нашел применение в первых нечетких системах автоматического управления. Был предложен в 1975 году английским математиком Е. Мамдани для управления паровым двигателем [3].

Формирование базы правил системы нечеткого вывода осуществляется в виде упорядоченного согласованного списка нечетких продукционных правил в виде «IF A THEN B», где антецеденты ядер правил нечеткой продукции построены при помощи логических связок «И», а консеквенты ядер правил нечеткой продукции простые.

Фаззификация входных переменных осуществляется описанным выше способом, так же, как и в общем случае построения системы нечеткого вывода.

Агрегирование подусловий правил нечеткой продукции осуществляется при помощи классической нечеткой логической операции «И» двух элементарных высказываний А, В.

Активизация подзаклучений правил нечеткой продукции осуществляется методом min-активизации, где  $\mu(x)$  и  $c$  – соответственно функции принадлежности термов лингвистических переменных и степени истинности нечетких высказываний, образующих соответствующие следствия (консеквенты) ядер нечетких продукционных правил.

Аккумуляция подзаклучений правил нечеткой продукции проводится при помощи классического для нечеткой логики max-объединения функций принадлежности.

Дефаззификация проводится методом центра тяжести или центра площади.

Алгоритм Сугено (*Sugeno*) выглядит следующим образом [3]:

Формирование базы правил системы нечеткого вывода осуществляется в виде упорядоченного согласованного списка нечетких продукционных правил в виде «IF A AND B THEN  $w = \epsilon_1 a + \epsilon_2 b$ », где антецеденты ядер правил нечеткой продукции построены из двух простых нечетких высказываний А, В при помощи логических связей «И», а и b – четкие значения входных переменных, соответствующие высказываниям А и В,  $\epsilon_1$  и  $\epsilon_2$  – весовые коэффициенты, определяющие коэффициенты пропорциональности между четкими значениями входных переменных и выходной переменной системы нечеткого вывода,  $w$  – четкое значение выходной переменной, определенное как действительное число.

Фаззификация входных переменных, определяющих высказывания и осуществляется аналогично алгоритму Мамдани.

Агрегирование подусловий правил нечеткой продукции осуществляется аналогично алгоритму Мамдани при помощи классической нечеткой логической операции «И» двух элементарных высказываний А, В.

Активизация подзаклучений правил нечеткой продукции проводится в два этапа. На первом этапе, степени истинности с заключений (консеквентов) нечетких продукционных правил, ставящих в соответствие выходной переменной действительные числа, находятся аналогично алгоритму Мамдани, как алгебраическое произведение весового коэффициента и степени истинности антецедента данного нечеткого продукционного правила. На втором этапе, в отличие от алгоритма Мамдани, для каждого из продукционных правил вместо построения функций принадлежности подзаклучений в явном виде находится четкое значение выходной переменной  $w = \epsilon_1 a + \epsilon_2 b$ . Таким образом, каждому  $i$ -му продукционному правилу ставится в соответствие точка  $(c_i; w_i)$ , где  $c_i$  – степень истинности продукционного правила,  $w_i$  – четкое значение выходной переменной,

определенной в консеквенте продукционного правила.

Аккумуляция заключений правил нечеткой продукции не проводится, поскольку на этапе активизации уже получены дискретные множества четких значений для каждой из выходных лингвистических переменных.

Дефаззификация проводится как и в алгоритме Цукамото. Для каждой лингвистической переменной осуществляется переход от дискретного множества четких значений  $\{w_1...w_n\}$  к единственному четкому значению согласно дискретному аналогу метода центра тяжести.

#### 1.4 Проблема разработки, тестирования и отладки алгоритмов управления

Создание современных АСУ требует дальнейшего повышения качества управления за счет использования высокоэффективных алгоритмов управления. Использование таких алгоритмов сдерживалось их сложностью и аналоговой элементной базой. Даже широкомасштабный процесс перехода на цифровую элементную базу не обеспечил соответствующего повышения качества управления из-за трудностей при реализации режима жесткого реального времени. Вторым сдерживающим фактором являлась высокая трудоемкость разработки программного обеспечения (ПО) АСУ.

Для экономии времени и ресурсов при разработке роботов используются симуляторы. Симулятор создает программную модель робота и его окружения и имитирует его поведение в соответствии с программой. Симулятор позволяет создавать ПО в независимости от физической реализации робота и изменять модель робота без физических модификаций. В некоторых случаях программы, созданные для работы с моделью в симуляторе, могут быть перенесены на робота без изменений.

Среди новейших технологий, доступных сегодня для программирования, являются те, которые используют виртуальную симуляцию. Написание кода для моделирования также проще, чем писать код для физического робота. Хотя переход к виртуальным симуляциям для программирования роботов является шагом вперед в дизайне пользовательского интерфейса, многие такие приложения находятся только в зачаточном состоянии.

## 2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

### 2.1 Постановка задачи на разработку программного средства

В классической постановке, пусть задано  $k$  входных и выходных переменных (для примера - три входных переменных  $x$ ,  $y$ ,  $z$  и одна выходная переменная  $w$ ). Пусть определены лингвистические значения этих переменных  $A, B, C, D, E, F$  для  $n$  правил (для примера  $n=3$ ). Типичный вариант представления логического вывода, с этапами фаззификации, агрегирования и дефаззификации показан на рис. 2.1.

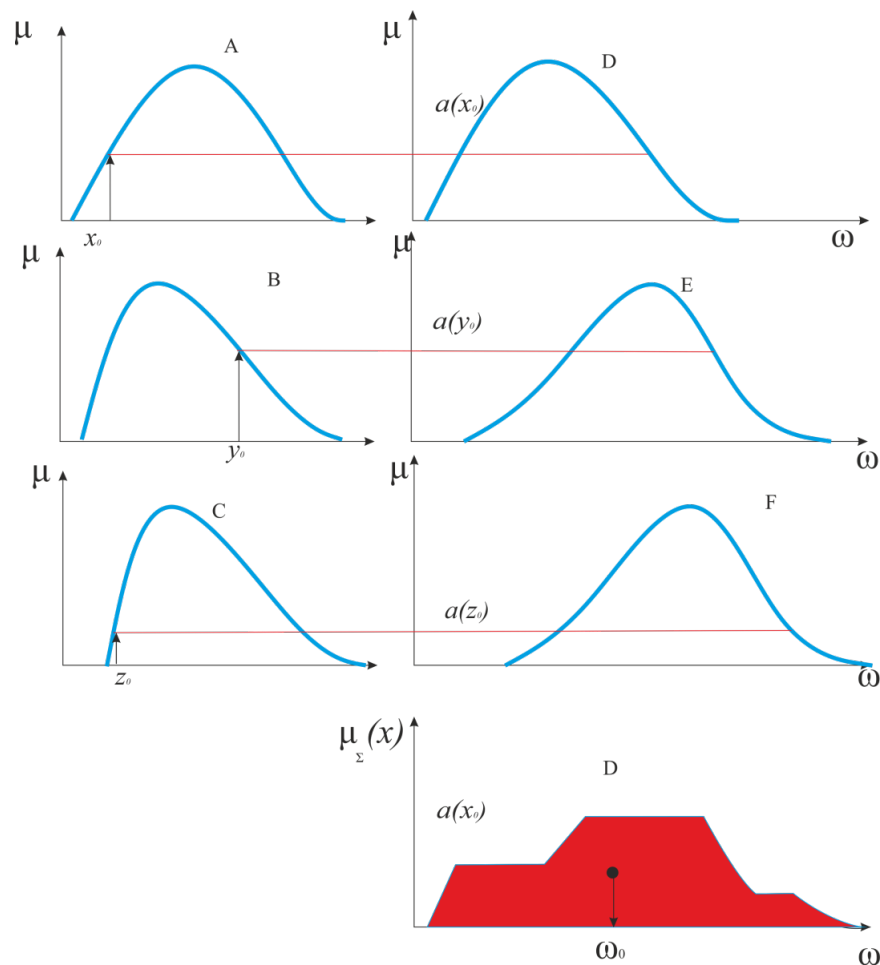


Рисунок 2.1 – Вариант представления логического вывода, с этапами фаззификации, агрегирования и дефаззификации

Точные исходные данные с датчиков, контролирующих управляющий процесс, переводятся в значения лингвистических переменных в специальном блоке фаззификатора. Далее реализуются процедуры нечеткого вывода на множестве продукционных правил, составляющих базу знаний системы управления, в результате чего формируются выходные лингвистические значения, которые переводятся в точные значения результатов вычислений в блоке дефаззификатора. На выходе последнего формируются управляющие

воздействия на исполнительные механизмы. Эта концептуальная схема лежит в основе нечеткого контроллера. И выполнение работы каждого блока нечеткого контроллера требует достаточного выделения ресурсов и времени на выполнение задач. [4]

Для отладки нечеткого контроллера могут применяться натурные эксперименты, но их проведение сопряжено со значительными временными и финансовыми, сложностью доступа к оборудованию, формированию полигона, погодных условия и так далее. Логичным решением этой проблемы представляется программное моделирование экспериментов, что позволяет значительно упростить, ускорить и удешевить настройку и отладку нечетких контроллеров. Многие реализации нечетких контроллеров основаны на классических алгоритмах, поэтому разрабатываемое программное средство должно содержать гибко настраиваемую реализацию одного из классических алгоритмов нечеткого вывода, на основе которого экспериментатор сможет реализовать собственный нечеткий контроллер. Поскольку нечеткие контроллеры — это активно развивающееся направление, разрабатываемое программное средство должно позволять с минимальными трудозатратами подключать реализованные экспериментатором алгоритмы нечетких контроллеров. Визуализация результатов эксперимента в виде графиков и диаграмм обычно требует установки, настройки и освоение стороннего программного обеспечения, поэтому разрабатываемое программное средство должно позволять отображать результаты экспериментов. Настройка алгоритмов нечеткого вывода является трудоемкой, поэтому программное средство должно иметь средства автоматизации обучения нечетких контроллеров.

Таким образом можно сформулировать требования к программному средству для отладки алгоритмов нечеткого управления мобильными роботами:

- содержать в себе готовую, настраиваемую модель нечеткого контроллера на базе классического алгоритма нечеткого вывода;
- иметь возможность подключения реализованных экспериментатором моделей нечетких контроллеров;
- позволять гибко настраивать конфигурацию и характеристики моделируемого пути;
- иметь возможность визуализации, сравнения и сохранения полученных в результате эксперимента данных;
- иметь средства автоматизированной генерации лингвистических переменных, термов и правил нечетких алгоритмов;
- быть доступной для освоения в короткие сроки.

## 2.2 Разработка обобщенной модели

Обобщенная схема эксперимента представлена на чертеже ГУИР ГУИР.400201.322 С2. Опишем ее блоки.

### 2.2.1 Ввод $S_0, V_0$

В начале эксперимента в качестве исходных данных задаются начальная скорость  $V_0$  и начальное расстояние до препятствия  $S_0$ .

### 2.2.2 Расчет усилия

Каждый квант модельного времени производится расчет тормозного усилия. Расчет производится нечетким алгоритмом на основании текущей скорости, текущего расстояния до препятствия, заданных термов лингвистических переменных и базы правил. Последние получены в результате работы алгоритма настройки и обучения.

### 2.2.3 Моделирование замедления

На основании расчетного усилия замедления и характеристик моделируемого пути в данной точке рассчитывается значение отрицательного ускорения, которое будет действовать на платформу в текущий момент модельного времени.

### 2.2.4 Расчет мгновенных $V', S'$ для текущего модельного времени $T$

На основании рассчитанного в п. 2.1.3 ускорения, а также расстояния и пути  $S$  и  $V$  для предыдущего кванта модельного времени, рассчитываются новые значения  $V'$  и  $S'$  для текущего модельного времени.

### 2.2.5 $S = 0$ или $V = 0$ ?

Здесь выполняется проверка: если расстояние до препятствия равно нулю, либо мгновенная скорость равна нулю, эксперимент окончен, иначе повторения цикла. Действительно: если скорость равна нулю, то, очевидно, дальнейшее положение платформы не изменится, она успешно остановилась. Верно и другое: если расстояние до препятствия равно нулю, значит произошло столкновение.

### 2.2.6 Алгоритм настройки и обучения

Области определения лингвистических переменных, функции принадлежности термов, а также правила задаются в результате работы алгоритма настройки и обучения. Проектируемое программное средство позволяет задавать вышеупомянутые переменные и правила в автоматизированном режиме на основе эталонных данных.



### 2.3 Методика тестирования и отладки алгоритмов нечеткого управления; алгоритм настройки нечеткого контроллера

Основной проблемой создания нечеткого регулятора является конструирование базы знаний, содержащей опыт и знания человека – оператора. Заполнение базы знаний может выполняться различными способами [9]:

- Оператор-эксперт управляет технологическим процессом, за которым «наблюдает» регулятор, запоминая все действия эксперта и заполняя свою базу знаний.

- Оператор-эксперт описывает свое действие при каждой наблюдаемой ситуации в виде продукции «если..., то... », которые и будут образовывать базу знаний регулятора.

- Перед самоорганизующимся нечетким регулятором ставится цель обеспечить желаемую переходную характеристику проектируемой системы и одновременно сообщается некоторая информация о технологическом процессе (объекте управления). Регулятор самостоятельно (методом проб и ошибок) накапливает знания без эксперта.

Кроме рассмотренной структуры возможны и более сложные, способные адаптировать к изменениям окружающей среды путем переключений на другие множества лингвистических значений, продукционные правила, базы знаний [10].

При реализации данного программного средства использовалась комбинация первого и второго подходов: на основании собранных данных в результате моделирования с применением «наивного» алгоритма, блок обучения нечеткого контроллера генерирует предварительный набор правил, которые затем донастраиваются оператором (экспертом).

### 2.4 Структурная схема экспериментального программного обеспечения

Изучив теоретические аспекты разрабатываемой системы и выработав список требований необходимых для разработки системы, разбиваем систему на функциональные блоки(модули). Это необходимо для обеспечения гибкой архитектуры. Такой подход позволяет изменять или заменять модули без изменения всей системы в целом.

В разрабатываемом приложении можно выделить следующие блоки:

- блок пользовательского интерфейса;
- блок диспетчера приложения;
- блок обучения нечетких контроллеров;
- блок визуализации телеметрии;
- блок симуляции;
- блок физической модели;
- блок моделирования нечетких алгоритмов;

– блок сбора телеметрии.

Структурная схема, иллюстрирующая перечисленные блоки и связи между ними приведена на чертеже ГУИР.400201.322 С1.

Каждый модуль выполняет свою задачу. Чтобы система работала каждый модуль взаимодействует с другими модулями путем обмена данными, используя различные форматы и протоколы.

Рассмотрим функциональные блоки приложения.

#### 2.4.1 Блок пользовательского интерфейса

Реализует интерфейс командной строки, Command Line Interface (CLI). Для профессионального научного ПО CLI имеет ряд преимуществ. Рассмотрим их ниже.

*Масштабируемость.* Любой, кто достаточно хорошо разбирается в управлении cookie в Firefox, вероятно, столкнется с «проблемой 1001 cookies», где окна диалога управления файлами cookie появляются так быстро и в таких количествах, что кажется, что может пройти весь день, чтобы пройти через все их. В этом примере демонстрируется проблема GUI и других интерактивных интерфейсов: масштабируемость. Когда вы выполняете только одну, простое задание один раз, может показаться довольно простым просто щелкнуть пару кнопок интерфейса и перейти дальше. Когда внезапно возникает избыток таких задач, и все они требуют внимания пользователя (например, выбирать, какие cookie принимать и как долго их хранить), графический интерфейс становится скорее помехой, чем помощью, и вашей панелью задач (если вы используете один) быстро заполняется кнопками окна, которые показывают что-то вроде «Conf ...» и ничего больше. В отличие от этого, инструменты командной строки, как правило, предлагают гораздо больше возможностей для решения таких задач масштабируемости быстро и легко. Организация вывода в группы, поэтому задачи, требующие одного и того же ответа пользователя, объединяются друг с другом, что позволяет выполнить одно действие для достижения результатов по большому числу дискретных задач. В тех случаях, когда для этих легионов диалогов Firefox требуется решение и каждое действие для каждого из них индивидуально, небольшая разумная сортировка и вывод на печать различных вызовов команды `yes` может предложить пользователю инструмента командной строки возможность использовать только несколько шаги для решения сотен отдельных задач.

*Автоматизируемость.* Любой продвинутый пользователь Unix знаком с силой сценариев для инструментов командной строки. Однако многие пользователи Microsoft Windows не так хорошо знакомы с мощностью и гибкостью командной строки в той же степени, что и их коллеги в Unix. Причиной этого является то, что, когда дело доходит до системного администрирования, администраторы MS Windows часто имеют гораздо более сложную командную строку из-за посредничества между

администратором и базовыми функциями системы, которые навязываются им графическим дизайном операционная система. Когда есть пять кнопок, которые нужно нажимать в определенном порядке восемнадцать раз в день в приложении GUI, нужно кликать по этим кнопкам восемнадцать раз, но когда требуется некоторая конкретная сложная команда восемнадцать раз в оболочке Unix, она является тривиальным упражнением для написания одного скрипта.

*Простота интерфейса.* Приложения GUI полагаются на меню и кнопки, чтобы что-то сделать. Это означает, что для мощных инструментов требуются сложные интерфейсы, чтобы пользователь мог получить доступ почти (почти) к всем функциям приложения с помощью мыши. Чем больше функций имеет приложение, тем сложнее становится интерфейс. Набор для повышения производительности Microsoft Office является ярким примером сложности, с которой сталкивается интерфейс при добавлении многих функций. С тех пор этот пакет приложений стал настолько сложным, что людям из Microsoft в конечном итоге пришлось придумать инновационный подход к управлению этой сложностью. Результатом стала лента - своего рода динамическая панель меню, где ваше приложение пытается представить наиболее используемые и наиболее полезные функции в любой момент времени в зависимости от того, что именно пользователь делает в этот момент. Этот подход интересен и помогает справляться со сложностью набора приложений, но в то же время он поддерживает самые обычные действия повседневных пользователей за счет того, что для «опытных пользователей» и их производительности значительно сложнее сделать вещи. У самых знающих пользователей, как правило, больше всего проблем с лентой. Приложения командной строки и управляемые на клавиатуре консольные интерфейсы могут предлагать простое взаимодействие для случайных пользователей, не вытаскивая коврик из-под ног более знающих пользователей. Многие из таких инструментов обеспечат простую информацию об использовании, используя параметр - help при вызове его в оболочке, или легкий доступ к помощи использования изнутри интерфейса с помощью нажатия клавиши или двух клавиш. Начинающие могут легко освоить рудиментарные основы, и по мере того как они узнают больше о том, как использовать его, их объект с расширением приложения расширяется. Простота интерфейса никогда не приходит за счет этого расширяющегося знания.

#### 2.4.2 Блок диспетчера приложения

Представляет из себя классический контроллер в парадигме MVC (Модель-Представление-Контроллер). MVC — это паттерн проектирования веб-приложений, который включает в себя несколько более мелких шаблонов. При использовании MVC модель данных приложения, пользовательский интерфейс и взаимодействие с пользователем разделены

на три отдельных компонента, благодаря чему модификация одного из них оказывает минимальное воздействие на остальные или не оказывает его вовсе.

Основная цель применения MVC состоит в разделении данных и бизнес-логики от визуализации (внешнего вида). За счет такого разделения повышается возможность повторного использования программного кода и упрощается сопровождение (изменения внешнего вида, например, не отражаются на бизнес-логике).

Концепция MVC разделяет данные, представление и обработку действий пользователя на компоненты:

*Модель.* Предоставляет собой объектную модель некой предметной области, включает в себя данные и методы работы с этими данными, реагирует на запросы из контроллера, возвращая данные и/или изменяя своё состояние, при этом модель не содержит в себе информации, как данные можно визуализировать, а также не «общается» с пользователем напрямую

*Представление.* Обычно являет собой (визуальное) представление его модели. Оно обычно выделяют некоторые атрибуты модели и подавляет другие. Таким образом, оно действует как фильтр представления. К своей модели (или части модели) прикрепляется представление и получает данные, необходимые для презентации из модели, отвечая на запросы. Оно также может обновить модель, отправив соответствующие сообщения. Все эти вопросы и сообщения должны быть в терминологии модели, поэтому представление должно знать семантику атрибутов модели, которую она представляет.

*Контроллер.* Обеспечивает связь между пользователем и системой, использует модель и представление для реализации необходимой реакции на действия пользователя, как правило, на уровне контроллера осуществляется фильтрация полученных данных и авторизация (проверяются права пользователя на выполнение действий или получение информации). Он предоставляет средства для вывода пользователя, предоставляя пользователю меню или другие средства представления команд и данных.

Модели являются определением вселенной, в которой работает ваше приложение. В банковском приложении, например, модель представляет собой все в банке, что поддерживает приложение: то есть счета, главную книгу и кредитные лимиты для клиентов, а также операции, которые могут быть использованы для манипулирования данными в модели, такие как депонирование средств и осуществление изъятия со счетов. Модель также несет ответственность за сохранение общего состояния и согласованности данных, например, она гарантирует, что все сделки будут добавлены в книгу, и что клиент не сможет снять больше денег, чем он может, или больше денег, чем имеет банк.

Модели также определяются тем, за что они не несут ответственность: модели не занимаются UI и обработкой запросов – это обязанности представлений и контроллеров. Представления содержат логику,

необходимую для отображения элементов модели пользователю, и больше ничего. Они не имеют прямого понимания модели и никоим образом напрямую не сообщаются с моделью. Контроллеры являются мостом между представлениями и моделью: запросы приходят от клиента и обслуживаются контроллером, который выбирает соответствующее представление для показа пользователю и, при необходимости, соответствующие действия, которые нужно выполнить с моделью.

Каждый элемент MVC архитектуры является четко определенным и автономным – это называется разделением понятий. Логика, которая манипулирует данными в модели, содержится только в модели; логика, которая отображает данные, находится только в представлении, а код, который обрабатывает запросы пользователей и вводные данные, содержится только в контроллере. С четким разделением обязанностей между каждой из частей ваше приложение будет легче поддерживать и расширять, независимо от того, насколько большим оно будет становиться.

#### 2.4.3 Блок обучения нечетких алгоритмов

Реализует автоматизированную генерацию термов лингвистических переменных и правил для нечетких алгоритмов. Генерация происходит на основании предварительно собранной телеметрии, полученной в ходе «эталонных» моделирований с применением традиционных алгоритмов. В процессе генерации базы лингвистических переменных и правил происходит разбиение области значений входных и выходных переменных на заданное пользователем количество частей, генерация термов на основании этого разбиения. Затем анализируется весь массив данных телеметрии, на основании чего выводятся правила алгоритмов нечеткой логики. При необходимости, пользователь производит точную донастройку вручную. Результаты генерации сохраняются в JSON файл. Преимуществом формата JSON является то, что он одновременно легко читается как машинами, так и людьми в любом текстовом редакторе.

#### 2.4.4 Блок визуализации телеметрии

Предназначен для визуализации собранной в результате экспериментов телеметрии. Позволяет отображать изменяющиеся во времени параметры модели в виде графиков, позволяет сравнивать графики разных экспериментов. Реализован с применением общепринятой для научной графики библиотеки Matplotlib. Matplotlib — библиотека на языке программирования Python для визуализации данных двумерной (2D) графикой (3D графика также поддерживается). Получаемые изображения могут быть использованы в качестве иллюстраций в публикациях.

#### 2.4.5 Блок симуляции

Управляет симуляцией физической модели, алгоритма контроллера и сбором телеметрии. Концептуально является диспетчером подсистемы симуляции. Блок симуляции задает грануляцию модельного времени, инициализирует физическую модель и конфигурирует путь мобильной платформы. Через него осуществляется коммуникация между блоками более нижних уровней, и главным контроллером приложения. Кроме того, блок симуляции отвечает за генерацию и синхронизацию квантов модельного времени между блоками физической модели, моделирования нечетких алгоритмов и сбора телеметрии.

#### 2.4.6 Блок физической модели

Отвечает за изменение параметров движения мобильной платформы с учетом управляющих сигналов, исходящих от нечеткого контроллера, конфигурации пути и инерционности системы управления. Реализация физической модели позволяет гибко настраивать параметры участков пути, устанавливая следующие параметры:

- Границы участка. Отметки расстояния от точки торможения, задающие конфигурируемый участок.
- Константное добавочное ускорение. Позволяет моделировать наклонные участки пути.
- Максимальное применяемое ускорение. Позволяет моделировать ограничения коэффициента сцепления с дорогой: обледенелые, мокрые, изношенные участки пути.

Границы участков могут пересекаться и накладываться друг на друга, в этом случае ограничения участков будут применяться «послойно». Это позволяет легко задавать, например, коэффициент ускорения на всем протяжении пути, выделяя отдельные «обледенелые» участки.

Инерционность системы управления позволяет моделировать задержки в реакции системы управления, вызванные, например, ограниченным быстродействием пневмогидравлической системы торможения.

#### 2.4.7 Блок моделирования нечетких алгоритмов

Непосредственно отвечает за моделирование контроллера мобильной платформы. Благодаря универсальному API возможно подключение разных реализаций как нечетких, так и традиционных контроллеров. В качестве входных данных блок моделирования получает значения модельных текущей скорости, ускорения и расстояния до цели и на основании этого генерирует выходные данные – значение тормозного усилия, которое затем будет применено при расчёте фактического ускорения (замедления) платформы в блоке физической модели. Вместе с системой поставляются несколько

базовых версий контроллера: «наивный» контроллер, контроллер на базе классического алгоритма Мамдани, а также контроллер на базе ускоренного алгоритма Мамдани. «Наивный» контроллер используется для расчета эталонных значений телеметрии в идеальных условиях для дальнейшего обучения нечетких алгоритмов.

#### 2.4.8 Блок сбора телеметрии

Предназначен для сбора параметров физической модели для каждого кванта модельного времени. Для задачи плавной остановки платформы собирает мгновенные расстояние до цели, скорость, ускорение и рывок, привязанные к модельному времени.

Телеметрия может быть визуализирована непосредственно после окончания моделирования, либо сохранена для последующих визуализации и анализа. Данные выгружаются в формате CSV, что позволяет оперировать ими в большом количестве стороннего ПО. CSV (от англ. Comma-Separated Values — значения, разделённые запятыми) — текстовый формат, предназначенный для представления табличных данных. Каждая строка файла — это одна строка таблицы. Разделителем (англ. delimiter) значений колонок является символ запятой (.). Однако на практике часто используются другие разделители. Значения, содержащие зарезервированные символы (двойная кавычка, запятая, точка с запятой, новая строка) обрамляются двойными кавычками ("). Если в значении встречаются кавычки — они представляются в файле в виде двух кавычек подряд.

## 3 ТЕХНИЧЕСКАЯ РЕАЛИЗАЦИЯ И ВЕРИФИКАЦИЯ ПРОЕКТА

### 3.1 Разработка ПО

#### 3.1.1 Блок пользовательского интерфейса

Реализует интерфейс командной строки (CLI). Для обработки параметров командной строки использует пакет `argparse`.

Начиная с версии Python 2.7, в набор стандартных библиотек была включена библиотека `argparse` для обработки аргументов (параметров, ключей) командной строки. Хотелось бы остановить на ней Ваше внимание.

– Для начала рассмотрим, что предлагает `argparse`. `Argparse` — это изящный инструмент для:

- анализа аргументов `sys.argv`;
- конвертирования строковых аргументов в объекты Вашей программы и работа с ними;
- форматирования и вывода информативных подсказок;
- многого другого.

Одним из аргументов противников включения `argparse` в Python был довод о том, что в стандартных модулях и без этого содержится две библиотеки для семантической обработки (парсинга) параметров командной строки. Однако, как заявляют разработчики `argparse`, библиотеки `getopt` и `optparse` уступают `argparse` по нескольким причинам:

– Обладая всей полнотой действий с обычными параметрами командной строки, они не умеют обрабатывать позиционные аргументы (`positional arguments`). Позиционные аргументы — это аргументы, влияющие на работу программы, в зависимости от порядка, в котором они в эту программу передаются. Простейший пример — программа `cp`, имеющая минимум 2 таких аргумента («`cp source destination`»).

– `Argparse` дает на выходе более качественные сообщения о подсказке при минимуме затрат (в этом плане при работе с `optparse` часто можно наблюдать некоторую избыточность кода).

– `Argparse` дает возможность программисту устанавливать для себя, какие символы являются параметрами, а какие нет. В отличие от него, `optparse` считает опции с синтаксисом наподобии `pf`, `-file`, `+rgb`, `/f` и т.п. «внутренне противоречивыми» и «не поддерживается `optpars`’ом и никогда не будет».

– `Argparse` даст Вам возможность использовать несколько значений переменных у одного аргумента командной строки (`nargs`).

– `Argparse` поддерживает субкоманды (`subcommands`). Это когда основной парсер отправляет к другому (субпарсеру), в зависимости от аргументов на входе.

Исходя из вышесказанного, можно сделать вывод, что `argparse` — довольно мощная и легкая библиотека, предоставляющая, на мой взгляд,



очень удобный интерфейс для работы с параметрами командной строки.

Рассмотрим теперь реализацию интерфейса командной строки в дипломном проекте. CLI реализован при помощи класса `fuzzysim.launcher.CLI`.

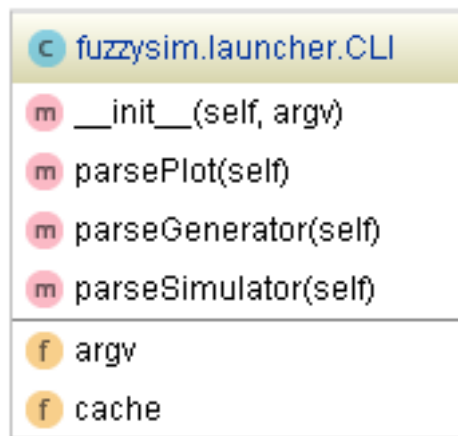


Рисунок 3.1 – Структура класса `fuzzysim.launcher.CLI`

Поля класса `fuzzysim.launcher.CLI`:

- `cache`. Отображение разобранных аргументов командной строки, для дальнейшего использования.
- `argv`. Список текстовых аргументов командной строки, предоставляемый оболочкой.

Методы класса `fuzzysim.launcher.CLI`:

- `__init__(self, argv)`. Инициализирует поля контроллера.
- `parsePlot(self)`. Производит разбор аргументов для визуализации графиков.
- `parseGenerator(self)`. Производит разбор аргументов для генерации правил.
- `parseSimulator(self)`. Производит разбор аргументов для запуска симуляции.

То, как реализован разбор аргументов командной строки, показано в листинге функции `parseSimulator`:

```
def parseSimulator(self, argv):
    sim_parser = argparse.ArgumentParser(description='Fuzzysim
simulator launcher cli')
    sim_parser.add_argument('distance', type=int, help="initial
distance, meters")
    sim_parser.add_argument('speed', type=int, help="initial
speed, m/s")
    sim_parser.add_argument('-w', '--way', type=str, dest='
way_config_file', default='', help="way config file")
    sim_parser.add_argument('-l', '--latency', type=float, dest
='latency', default=0,
                                help="acceleration
appying latency,
seconds")
```

```

sim_parser.add_argument('-c', '--controller', type=str,
                        dest='brake_controller', default='dumb',
                        help="brake controller module")
sim_parser.add_argument('-n', '--noplots', action='
                        store_true', dest='noplots_graphs', default=False,
                        help="Don't plot graphs")
sim_parser.add_argument('-s', '--stats', action='store_true
                        ', dest='print_stats', default=False,
                        help="Print stats csv into stdout")
return sim_parser.parse_args(argv)

```

### 3.1.2 Блок диспетчера приложения

Блок диспетчера приложения реализован в модуле `fuzzisim.launcher`. Он представляет из себя классический контроллер в парадигме MVC (Модель-Представление-Контроллер) и таким образом осуществляет роутинг запросов пользователя в нужный модуль. Рассмотрим его основные функции.

#### 3.1.2.1 Функция `generate(argv)`

Генерирует правила для алгоритма Мамдани, анализируя собранную телеметрию, находящуюся в CSV файле. Обрабатывает параметры командной строки: `-generate` — имя файла с собранной телеметрией; `-terms` — количество термов, на которые следует разбить область значений; `-interval` — флаг, показывающий, что требуется сгенерировать интервальные функции принадлежности.

```

def generate(argv):
    args = gen_parser.parse_args(argv)
    config = mamdani.generator.generate_config(args.csv_file, args
        .terms, args.interval)
    print(json.dumps(config, indent=4))

```

#### 3.1.2.2 Функция `launch(argv)`

Запускает симуляцию исходя из начальных параметров, выбранного контроллера торможения, выводит результат и собранную телеметрию. Обрабатывает параметры командной строки: `distance` — начальная дистанция до препятствия; `speed` — начальная скорость, с которой осуществляется торможение; `-latency` — задержки в системе управления; `-way` — файл конфигурации пути, содержит в себе данные о характеристиках участков пути в формате json; `-controller` — имя модуля контроллера торможения, который будет использоваться в симуляции; `-noplots` — флаг, подавляет вывод графиков собранной телеметрии; `-stats` — флаг, если установлен,

выводит в поток стандартного вывода собранную телеметрию в формате csv вместо резюме результатов симуляции. Если флаг `–noplot` не установлен, то после завершения симуляции на экран выводятся интерактивные графики с результатами симуляции.

```
def launch(argv):
    args = sim_parser.parse_args(argv)

    simulate(
        args.distance,
        args.speed,
        args.latency,
        args.way_config_file,
        args.brake_controller,
        not args.noplot_graphs,
        args.print_stats
    )
```

### 3.1.2.3 Функция plot(argv)

Визуализирует результаты предварительно сохраненной телеметрии, из файла в формате CSV. В качестве аргумента принимает имя CSV файла, в котором содержится собранная телеметрия от одной или нескольких симуляций. Позволяет отображать результаты нескольких экспериментов, наложенные на один график. Кроме того, в поток стандартного вывода выводит декартово расстояние графиков экспериментов от нулевого (принимаемого за эталонный), что позволяет оценить точность вычислений.

```
def plot(argv):
    args = plot_parser.parse_args(argv)
    fuzzysim.charts.show_charts_csv(args.csv_file)
```

### 3.1.3 Блок обучения нечетких алгоритмов

Реализует автоматизированную генерацию термов лингвистических переменных и правил для нечетких алгоритмов. Входные данные получает в формате CSV, что обусловлено табличным характером входных данных. Источником входных данных может быть любое приложение, позволяющее представить симулированную, либо реальную телеметрию в необходимом формате. Вывод симулятора с установленным флагом `-s` является валидным источником данных для генератора правил.

Опишем формат входных данных. Первой строкой, завершающейся символом `n` является заголовок, в котором заданы названия колонок. Допустимые колонки:

- T — число с плавающей точкой, означающее количество секунд, прошедших с начала эксперимента;

- $S$  — число с плавающей точкой, означающее расстояние в метрах до препятствия в соответствующий момент времени эксперимента;
- $V$  — число с плавающей точкой, означающее текущую скорость в метрах в секунду в соответствующий момент времени эксперимента;
- $A$  — число с плавающей точкой, означающее текущее ускорение в метрах в секунду за секунду в соответствующий момент времени эксперимента.

Блок реализован в модуле `fuzzysim.mamdani.generator`. Рассмотрим его основные функции.

### 3.1.3.1 Функция `generate_config(csv_file_name, terms_count, is_interval)`

Является точкой входа в модуль. Производит разбор файла с данными, сборку объекта правил и вывод сериализованного набора правил и термов. В качестве входных аргументов принимает:

- `csv_file_name` — строка, содержащая имя CSV файла с исходными данными;
- `terms_count` — целое число, количество термов, на которые разбивается пространство значений входных и выходных переменных;
- `is_interval` — булево значение, показывающий требуется ли генерировать интервальные задающие функции термов.

```
def generate_config(csv_file_name, terms_count, is_interval):
    data = {'S': [], 'V': [], 'A': []}

    with open(csv_file_name, newline='') as csvfile:
        reader = csv.reader(csvfile, delimiter=',', quotechar='\"')
        cols = {}
        for row in reader:
            if row[0] == "t":
                cols = {v.upper(): i for i, v in enumerate(row)}
                continue

            # data['T'].append(float(row[0]))
            data['S'].append(float(row[cols['S']]))
            data['V'].append(float(row[cols['V']]))
            data['A'].append(abs(float(row[cols['A']])))
            # data['J'].append(float(row[4]))

    maxs = {k: max(v) for k, v in data.items()}
    mins = {k: min(v) for k, v in data.items()}
    steps = {k: abs(maxs[k] - mins[k]) / terms_count for k, v in
              maxs.items()}

    config = {
        'S': get_var_config(maxs['S'], mins['S'], steps['S'], 0),
        'V': get_var_config(maxs['V'], mins['V'], steps['V'], 0),
        'A': get_var_config(maxs['A'], mins['A'], steps['A'],
                             is_interval),
        'rules': generate_rules(data, ['S', 'V'], ['A'], terms_count)
    }
```

```

a_map = print_map(config, terms_count)
for s, speeds in enumerate(a_map):
    begin = True
    for v, a in enumerate(speeds):
        if a != ' ':
            begin = False
            continue

    if begin:
        # a_map[s][v] = 0
        config['rules'].append({'conds': [['S', s], ['V', v]], '
                                concs': [['A', 0], []]})
    else:
        config['rules'].append({'conds': [['S', s], ['V', v]], '
                                concs': [['A', terms_count - 1], []]})

print("\n", file=sys.stderr, sep='')
print_map(config, terms_count)

return config

```

### 3.1.3.2 Функция generate\_rules(data, in\_vars, out\_vars, terms\_count)

Генерирует правила нечеткого контроллера. Определяет область значений входных и выходных переменных, разбивает их на требуемое количество термов. Затем для каждой временной точки определяет, в каких термах относятся данные значения входных и выходных переменных, и на основании этого составляется набор правил. После генерации полного набора из него удаляются дублирующие правила. При этом при генерации правил возможны случаи неоднозначности: при сравнительно малом количестве термов одинаковому набору входных термов может соответствовать разные наборы выходных термов. В этом случае выбирается то правило, в котором мощность множества соответствующих данному правилу срезов телеметрии максимально, однако такие ситуации обычно требуют ручной настройки правил.

В качестве входных аргументов принимает:

- data — многомерный массив с данными телеметрии;
- in\_vars — вектор строк, список имен входных переменных;
- out\_vars — вектор строк, список имен выходных переменных;
- terms\_count — целое число, количество термов, на которые разбивается пространство значений входных и выходных переменных.

```

def generate_rules(data, in_vars, out_vars, terms_count):
    rules = {}

    maxs = {k: max(v) for k, v in data.items()}
    mins = {k: min(v) for k, v in data.items()}
    steps = {k: abs(maxs[k] - mins[k]) / terms_count for k, v in
              maxs.items()}

    for i, _ in enumerate(list(data.values())[0]):
        terms = []

```

```

for k, stat in data.items():
    max_term = int(abs(maxs[k] - mins[k]) / steps[k]) - 1
    terms.append([k, min(max_term, int(abs(stat[i] - mins[k]) /
        steps[k]))])

in_terms = tuple(tuple(v) for v in terms if v[0] in in_vars)
out_terms = tuple(tuple(v) for v in terms if v[0] in out_vars)

if in_terms not in rules:
    rules[in_terms] = {}

if out_terms not in rules[in_terms]:
    rules[in_terms][out_terms] = 0

rules[in_terms][out_terms] += 1

# rules[in_terms] = out_terms

return list([{'conds': k, 'concs': max(v.items(), key=lambda x:
    x[1])[0]} for k, v in rules.items()])

```

### 3.1.3.3 Функция print\_map(config, terms\_count)

Выводит визуализацию сгенерированных правил. Печатает в поток стандартного вывода матрицу правил, где номер строки соответствует номеру термина расстояния, а номер столбца — номеру термина скорости. Значение элемента — номер термина ускорения.

В качестве входных аргументов принимает:

- config — объект с правилами и терминами;
- terms\_count — целое число, количество термов, на которые разбивается пространство значений входных и выходных переменных.

```

def print_map(config, terms_count):
    a_map = []
    for i in range(0, terms_count):
        a_map.append([])
        for j in range(0, terms_count):
            a_map[i].append(' ')

    for r in config['rules']:
        m = {}
        for c in r['conds']:
            m[c[0]] = c[1]
        v = str(r['concs'][0][1])
        a_map[m['S']][m['V']] = v

    print('S\\V', [str(i) for i in range(0, terms_count)], file=sys
        .stderr, sep='')
    for i, row in enumerate(a_map):
        print("%03d" % i, row, file=sys.stderr, sep='')

    return a_map

```

### 3.1.4 Блок визуализации телеметрии

Осуществляет представление собранной телеметрической информации в виде интерактивных графиков и реализован в модуле `fuzzysim.charts`.

Графики выводятся при помощи пакета `Matplotlib`, который позволяет изменять масштаб, сохранять скриншоты, просмотреть точные значения в различных точках графика. `Matplotlib` является де-факто стандартом для визуализации научных данных.

`Matplotlib` состоит из множества модулей. Модули наполнены различными классами и функциями, которые иерархически связаны между собой.

Создание рисунка в `matplotlib` схоже с рисованием в реальной жизни. Так художнику нужно взять основу (холст или бумагу), инструменты (кисти или карандаши), иметь представление о будущем рисунке (что именно он будет рисовать) и, наконец, выполнить всё это и нарисовать рисунок деталь за деталью.

В `matplotlib` все эти этапы также существуют, и в качестве художника-исполнителя здесь выступает сама библиотека. От пользователя требуется управлять действиями художника-`matplotlib`, определяя что именно он должен нарисовать и какими инструментами. Обычно создание основы и процесс непосредственно отображения рисунка отдаёт полностью на откуп `matplotlib`. Таким образом, пользователь библиотеки `matplotlib` выступает в роли управленца. И чем проще ему управлять конечным результатом работы `matplotlib`, тем лучше.

Так как `matplotlib` организована иерархически, а наиболее простыми для понимания человеком являются самые высокоуровневые функции, то знакомство с `matplotlib` начинают с самого высокоуровневого интерфейса `matplotlib.pyplot`. Так, чтобы нарисовать гистограмму с помощью этого модуля, нужно вызывать всего одну команду: `matplotlib.pyplot.hist(arr)`.

Пользователю не нужно думать как именно библиотека нарисовала эту диаграмму. Если бы мы рисовали гистограмму самостоятельно, то заметили бы, что она состоит из повторяющихся по форме фигур - прямоугольников. А чтобы нарисовать прямоугольник, нужно знать хотя бы координату одного угла и ширину/длину. Рисовали же бы мы прямоугольник линиями, соединяя угловые точки прямоугольника.

В тоже время для более серьёзных задач (внедрение `matplotlib` в пользовательскую GUI) требуется больше контроля над процессом и больше гибкости, чем могут предоставить эти два модуля. Необходим доступ к более низкоуровневым возможностям библиотеки, которая реализована в объектно-ориентированном стиле. ООС заметно сложнее для новичков и требует знаний о работе конкретных классов и их методах, но предоставляет самые большие возможности по взаимодействию с библиотекой `matplotlib`.

Главной единицей (объектом самого высокого уровня) при работе с `matplotlib` является рисунок (`Figure`). Любой рисунок в `matplotlib` имеет вложенную структуру и чем-то напоминает матрёшку:

- *Рисунок (`Figure`)*. Рисунок является объектом самого верхнего уровня, на котором располагаются одна или несколько областей рисования (`Axes`), элементы рисунка `Artists` (заголовки, легенда и т.д.) и основа-холст (`Canvas`). На рисунке может быть несколько областей рисования `Axes`, но данная область рисования `Axes` может принадлежать только одному рисунку `Figure`.

- *Область рисования (`Axes`)*. Область рисования является объектом среднего уровня, который является, наверное, главным объектом работы с графикой `matplotlib` в объектно-ориентированном стиле. Это то, что ассоциируется со словом «plot», это часть изображения с пространством данных. Каждая область рисования `Axes` содержит две (или три в случае трёхмерных данных) координатных оси (`Axis` объектов), которые упорядочивают отображение данных.

- *Координатная ось (`Axis`)*. Координатная ось является объектом среднего уровня, которые определяют область изменения данных, на них наносятся деления `ticks` и подписи к делениям `ticklabels`. Расположение делений определяется объектом `Locator`, а подписи делений обрабатывает объект `Formatter`. Конфигурация координатных осей заключается в комбинировании различных свойств объектов `Locator` и `Formatter`.

- *Элементы рисунка (`Artists`)*. Элементы рисунка `Artists` являются как бы красной линией для всех иерархических уровней. Практически всё, что отображается на рисунке является элементом рисунка, даже объекты `Figure`, `Axes` и `Axis`. Элементы рисунка `Artists` включают в себя такие простые объекты как текст (`Text`), плоская линия (`Line2D`), фигура (`Patch`) и другие.

Когда происходит отображение рисунка, все элементы рисунка `Artists` наносятся на основу-холст. Большая часть из них связывается с областью рисования `Axes`. Также элемент рисунка не может совместно использоваться несколькими областями `Axes` или быть перемещён с одной на другую.

В `matplotlib` изобразительные функции логически разделены между несколькими объектами, причём каждый из них сам имеет довольно сложную структуру. Можно выделить три уровня интерфейса прикладного программирования (`matplotlib API`):

- `matplotlib.backend_bases.FigureCanvas` — абстрактный базовый класс, который позволяет рисовать и визуализировать результаты команд.

- `matplotlib.backend_bases.Renderer` — объект (абстрактный класс), который знает как рисовать на `FigureCanvas`;

- `matplotlib.artist.Artist` — объект, который знает, как



использовать визуализатор, чтобы рисовать на холсте.

`FigureCanvas` и `Renderer` обрабатывают детали, необходимые для взаимодействия со средствами пользовательского интерфейса, так как это делает `WxPython` или язык рисования `PostScript`. А `Artist` обрабатывает все конструкции высокого уровня такие как представление и расположение рисунка, текста и линий.

Существует два типа объектов-классов `Artists`: примитивы и контейнеры.

Примитивы представляют собой стандартные графические объекты: плоскую линию, прямоугольник, текст, изображение и т.д. А контейнеры - это объекты-хранилища, на которые можно наносить графические примитивы. К контейнерам относятся рисунок, область рисования, координатная ось деления. Рассмотрим контейнеры подробнее, так как именно с помощью обращений к различным контейнерам класса `Artists`, объединенных логически в единую структуру, будет осуществляться пользовательская настройка рисунков в `matplotlib`.

Всего существует 4 вида `Artists` контейнеров:

- Контейнеры рисунка. `Figure` - это контейнер самого высокого уровня. На нём располагаются все другие контейнеры и графические примитивы.

- Контейнеры областей рисования. `Axes` - очень важный контейнер, так как именно с ним чаще всего работает пользователь. Экземпляры `Axes` - это области, располагающиеся в контейнере `Figure`, для которых можно задавать координатную систему (декартова или полярная). На нём располагаются все другие контейнеры, кроме `Figure`, и графические примитивы. Это области на рисунке, на которых располагаются графики и диаграммы, в которые вставляются изображения и т.д. Мультиоконные рисунки состоят из набора областей `Axes`.

- Контейнеры осей. `Axis` похож на `Axes` по названию, но не стоит их путать. Этот контейнер обслуживает экземпляры `Axes`. Он отвечает за создание координатных осей, на которые будут наноситься деления осей, подписи делений и линий вспомогательной сетки. Его специализация (и отличие от контейнера `Tick`) - это расположение делений и линий, их позиционирование и форматирование подписей делений, их отображение.

- Контейнеры делений. Контейнер низшего уровня. Его специализация (и отличие от контейнера `Axis`) - задавать характеристики (цвет, толщина линий) линий сетки, делений и их подписей (размеры и типы шрифтов).

При создании рисунка в `matplotlib` обычно поступают так: создают экземпляр класса `Figure`, на котором выделяют одну или нескольких областей `Axes`, и используют вспомогательные методы экземпляра класса `Axes` для создания графических примитивов. Если автоматически подобранные характеристики координатной сетки, делений и их подписей не устраивают пользователя, то они настраиваются с помощью экземпляров контейнеров `Axis` и `Tick`, которые всегда присутствуют на созданной

области рисования Axes.

Опишем основные функции модуля `fuzzysim.charts`.

### 3.1.4.1 Функция `show_charts_csv(csv_file_name, terms_count, is_interval)`

Является основной точкой входа в модуль. На первом этапе работы разбирает входной файл данных:

```
experiments = []
experiment = None
with open(csv_file_name, newline='') as csvfile:
    reader = csv.reader(csvfile, delimiter=',', quotechar='\"')
    for row in reader:
        if row[0] == "t":
            # new experiment
            if experiment is not None:
                experiments.append(experiment)
            experiment = {'stats_t': [], 'stats_s': [], 'stats_v': [],
                          'stats_a': [], 'stats_j': []}
            continue

        experiment['stats_t'].append(float(row[0]))
        experiment['stats_s'].append(float(row[1]))
        experiment['stats_v'].append(float(row[2]))
        experiment['stats_a'].append(float(row[3]))
        experiment['stats_j'].append(float(row[4]))

experiments.append(experiment)
```

В качестве входного аргумента получает строку с именем CSV файла данных. Данные для визуализации получает в формате CSV, что обусловлено табличным характером входных данных. Источником входных данных может быть любое приложение, позволяющее представить симулированную, либо реальную телеметрию в необходимом формате. Вывод симулятора с установленным флагом «-s» является валидным источником данных для генератора правил.

Опишем формат входных данных. Первой строкой, завершающейся символом

n является заголовок, в котором заданы названия колонок. Допустимые колонки:

- T — число с плавающей точкой, означающее количество секунд, прошедших с начала эксперимента;
- S — число с плавающей точкой, означающее расстояние в метрах до препятствия в соответствующий момент времени эксперимента;
- V — число с плавающей точкой, означающее текущую скорость в метрах в секунду в соответствующий момент времени эксперимента;
- A — число с плавающей точкой, означающее текущее ускорение в метрах в секунду за секунду в соответствующий момент времени эксперимента.

После разбора CSV файла происходит группировка табличных данных в датасеты, разбитые по экспериментам:

```
datasets = []
for en, experiment in enumerate(experiments):
    experiment_datasets = get_datasets(**experiment)

    for i, ds in enumerate(experiment_datasets):
        if i == len(datasets):
            datasets.append({'sub': []})

        datasets[i]['xl'] = ds['xl']
        datasets[i]['yl'] = ds['yl']
        ds['yl'] = "(%d) %s" % (en, ds['yl'])
        datasets[i]['sub'].append(ds)
```

После этого вызываются функции вывода статистики и отображения графиков.

#### 3.1.4.2 Функция show\_stats(experiments)

Выводит в поток стандартного вывода декартово расстояние между нулевым экспериментом и каждым другим, встречающимся в входном файле. В качестве входного параметра принимает массив структур `experiment`, содержащих сгруппированные данные из входного файла.

```
def show_stats(experiments):
    if len(experiments) < 2:
        return

    gauge = experiments[0]
    stats = {'stats_s': {}, 'stats_v': {}, 'stats_a': {}, 'stats_j': {}}
    for en, experiment in enumerate(experiments):
        for param in stats:
            sum = 0
            for i, s in enumerate(experiment[param]):
                G = len(gauge[param]) > i and gauge[param][i] or 0
                sum += (s - G)**2
            sum /= len(experiment[param])
            stats[param][en] = math.sqrt(sum)

    print("Normalized quadratic Parameters deviations by experiment")
    pprint.pprint(stats)
```

#### 3.1.4.3 Функция plot\_datasets(datasets)

Отображает данные датасетов при помощи пакета модуля `matplotlib.pyplot`. Она автоматически формирует легенду, добавляет подписи к осям, формирует заголовки страниц. При этом графики разбиваются на типы в соответствии с данными датасетов, и на один график накладываются данные разных экспериментов, что позволяет их сравнивать.

Для одновременного отображения разных графиков используются `pyplot` `figures` (рисунки):

Рисунок - это общее окно или страница, на которой все нарисовано. Это компонент верхнего уровня всех тех, которые вы рассмотрите в следующих пунктах. Вы можете создать несколько независимых фигур. На рисунке может быть несколько других вещей, таких как `suptitle`, который является центральным названием фигуры. Вы также обнаружите, что вы можете добавить легенду и цветную панель, например, к вашей фигуре.

К фигуре добавляются оси. Оси - это область, на которой построены данные с такими функциями, как `plot()` и `scatter()`, и которые могут иметь связанные с ней клещи, метки и т. Д. Это объясняет, почему фигуры могут содержать несколько осей.

Когда вы хотите просмотреть свои сюжеты на вашем дисплее, бэкэнду пользовательского интерфейса нужно будет запуститься `GUI mainloop`. Это то, что делает `show()`. Он сообщает `Matplotlib`, чтобы поднять все окна с фигурами, созданные до сих пор, и запустить `mainloop`. Поскольку этот `mainloop` блокируется по умолчанию (т.е. выполнение скрипта приостановлено), вы должны только вызывать его один раз для каждого скрипта, в конце. Выполнение скрипта возобновляется после закрытия последнего окна.

Приведем часть кода функции `plot_datasets(datasets)`

```
def plot_datasets(datasets):  
    for ds in datasets:  
        (f, ax) = plt.subplots()  
  
        ax.grid(True)  
  
    # <...>  
  
    if 'sub' in ds:  
        for sub in ds['sub']:  
            label = 'yl' in sub and sub['yl']  
            marker = 'ym' in sub and sub['ym'] or None  
            ax.plot(sub['x'], sub['y'], label=label, marker=marker)  
            ax.legend()  
  
            ax.spines['left'].set_position('zero')  
            ax.spines['bottom'].set_position('zero')  
            ax.spines['left'].set_smart_bounds(True)  
            ax.spines['bottom'].set_smart_bounds(True)  
  
    plt.show()
```

#### 3.1.4.4 Функция `show_charts_simulation(simulator)`

Отображает графики эксперимента непосредственно после симуляции, поэтому не требует разбора CSV файла. В качестве входного параметра

принимает объект симулятора класса `fuzzysim.Simulator` и собирает телеметрию из него.

```
def show_charts_simulator(simulator):
    """
    Prepare datasets and plot it
    """
    stats_t, stats_s, stats_v, stats_a, stats_j = simulator.stats_t
    , simulator.stats_s, simulator.stats_v, simulator.stats_a,
    simulator.stats_j
    datasets = get_datasets(stats_t, stats_s, stats_v, stats_a,
    stats_j)
    plot_datasets(datasets)
```

#### 3.1.4.5 Функция `show_vars(alg: mamdani.MamdaniAlgorithm)`

Отображает графики функций принадлежности входных и выходных лингвистических переменных для алгоритмов мамдани, принимая в качестве входного аргумента экземпляра класса `fi`. Из него извлекаются входные и выходные переменные, и по ним строятся графики принадлежностей.

```
def show_vars(alg: mamdani.MamdaniAlgorithm):
    f = plt.figure(1)
    f.suptitle("Out variables")
    setup_variables(alg.in_variables.values())

    f = plt.figure(2)
    f.suptitle("In Variables")
    setup_variables(alg.out_variables.values())

    plt.show()
```

#### 3.1.4.6 Функция `setup_variables(variables)`

Является вспомогательной для `show_vars()` и извлекает точки экстремума из объектов лингвистических переменных для отображения графика, а также формирует подписи к графикам и легенде.

```
def setup_variables(variables):
    for i, variable in enumerate(variables):
        ax = plt.subplot(len(variables), 1, i+1)
        ax.grid(True)
        ax.set_title("Variable: %s" % variable.name, loc='left')
        ax.spines['bottom'].set_position('zero')
        ax.spines['left'].set_position('zero')

    for t in variable.terms.values():
        c = min(t.c, variable.max)
        d = min(t.d, variable.max)
        plt.plot([t.a, t.b, c, d], [0, t.height, t.height, 0])
```

### 3.1.5 Блок симуляции

Блок симуляции реализован в классе модулем `fuzzysim.fuzzysim.Simulator`. Данный блок контролирует подсистемы симуляции физической модели, контроллера платформы, конфигурации пути, сбора телеметрии. Он инициализирует вышеуказанные системы перед началом симуляции, задает такты модельного времени, хранит собранную телеметрию.

Является фасадом для остальных частей системы. Агрегирует экземпляр класса физической модели и экземпляр класса контроллера платформы.

При проектировании сложных систем, зачастую применяется т.н. принцип декомпозиции, при котором сложная система разбивается на более мелкие и простые подсистемы. Причем, уровень декомпозиции (ее глубину) определяет исключительно проектировщик. Благодаря такому подходу, отдельные компоненты системы могут быть разработаны изолированно, затем интегрированы вместе. Однако возникает, очевидная на первый взгляд, проблема — высокая связность модулей системы. Это проявляется, в первую очередь, в большом объеме информации, которой модули обмениваются друг с другом. К тому же, для подобной коммуникации одни модули должны обладать достаточной информацией о природе других модулей.

Таким образом, минимизация зависимости подсистем, а также снижение объема передаваемой между ними информации — одна из основных задач проектирования.

Паттерн «Фасад» предоставляет унифицированный интерфейс вместо набора интерфейсов некоторой подсистемы. Фасад определяет интерфейс более высокого уровня, который упрощает использование подсистемы.

Проще говоря, «Фасад» — есть ни что иное, как некоторый объект, аккумулирующий в себе высокоуровневый набор операций для работы с некоторой сложной подсистемой. Фасад агрегирует классы, реализующие функциональность этой подсистемы, но не скрывает их. Важно понимать, что клиент, при этом, не лишается более низкоуровневого доступа к классам подсистемы, если ему это, конечно, необходимо. Фасад упрощает выполнение некоторых операций с подсистемой, но не навязывает клиенту свое использование.

Поля класса `fuzzisim.fuzzisim.Simulator`:

- `_controller`. Контроллер платформы. Выдает управляющие сигналы на торможение.

- `_physics`. Физическая модель. Рассчитывает скорость платформы, ее ускорение, рывок и положение в текущий момент модельного времени исходя из управляющих сигналов, поданных контроллером платформы, задержек в системе управления и характеристиками пути.

- `time_quantum`. Величина одного кванта модельного времени, в секундах. Исходя из нее устанавливаются размеры квантов времени

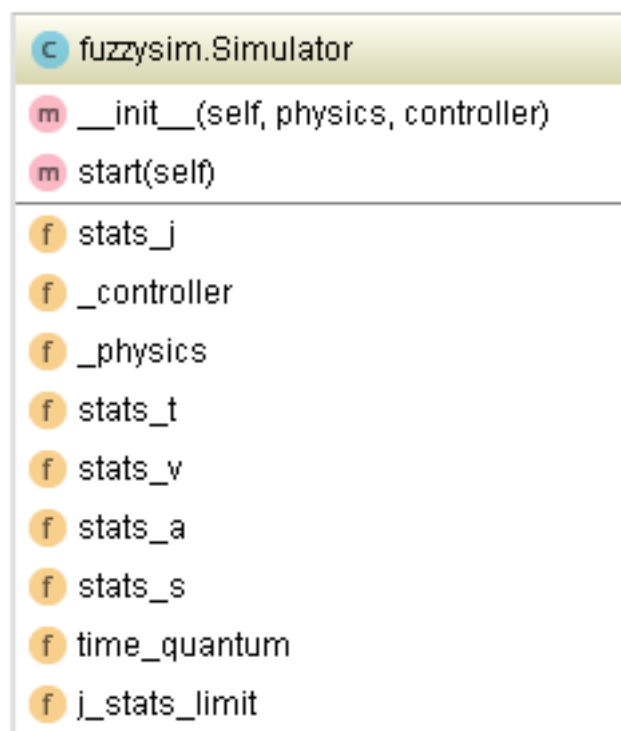


Рисунок 3.2 – Структура класса fuzzisim.fuzzisim.Simulator

физической модели и контроллера платформы.

- `j_stats_limit`. Максимальное значение рывка, которое следует учитывать при сборе телеметрии.

- `stats_t`. Массив со значениями модельного времени в секундах, относящемуся к текущему срезу телеметрии.

- `stats_s`. Массив со значениями расстояния до препятствия в метрах, относящемуся к текущему срезу телеметрии.

- `stats_v`. Массив со значениями скорости в метрах в секунду, относящемуся к текущему срезу телеметрии.

- `stats_a`. Массив со значениями ускорения в метрах в секунду за секунду, относящемуся к текущему срезу телеметрии.

- `stats_j`. Массив со значениями рывка в метрах в секунду за секунду, относящемуся к текущему срезу телеметрии.

Методы класса `fuzzisim.fuzzisim.Simulator`:

- `__init__(self, physics, controller)` — конструктор, инициализирует поля и классы-делегаты. В качестве аргументов принимает объекты физической модели и контроллера платформы.

- `start(self)` — запускает симуляцию, генерирует такты модельного времени, собирает телеметрию, а также детектирует окончание симуляции по причине остановки платформы, либо ее столкновения с препятствием.

```

class Simulator:
    """
  
```

```

Manage controllers, generate ticks, collects stats
"""
time_quantum = 0.001
j_stats_limit = 100

def __init__(self, physics, controller):
    """
    Inits world

    :param physics: PhysModel
    :param controller: CartController
    """
    physics.time_quantum = self.time_quantum
    controller.time_quantum = self.time_quantum

    self._physics = physics
    self._controller = controller

    self.stats_t = []
    self.stats_s = []
    self.stats_v = []
    self.stats_a = []
    self.stats_j = []

def start(self):
    prev_a = 0
    while not self._physics.is_stopped():
        try:
            self._physics.tick()
        except CollisionError as e:
            return False, e.t, e.s, e.v, e.a

        self._controller.tick()

        (t, s, v, a) = self._physics.get_params()

        j = (a - prev_a) / (self.time_quantum)
        j = math.copysign(min(abs(j), self.j_stats_limit), j)
        prev_a = a

        self.stats_t.append(t)
        self.stats_s.append(s)
        self.stats_v.append(v)
        self.stats_a.append(a)
        self.stats_j.append(j)

    return True, t, s, v, a

```

### 3.1.6 Блок физической модели

Блок физической модели реализован модулем `fuzzysim.fuzzysim`. Данный блок состоит из следующих классов:

- `fuzzysim.fuzzysim.PhysModel`;
- `fuzzysim.fuzzysim.CartController`;
- `fuzzysim.fuzzysim.WayConfig`;
- `fuzzysim.fuzzysim.Obstacle`;
- `fuzzysim.fuzzysim.CollisionError`.



### 3.1.6.1 fuzzisim.fuzzisim.PhysModel

Реализует физическую модель симуляции. Рассчитывает скорость платформы, ее ускорение, рывок и положение в текущий момент модельного времени исходя из управляющих сигналов, поданных контроллером платформы, задержек в системе управления и характеристиками пути.

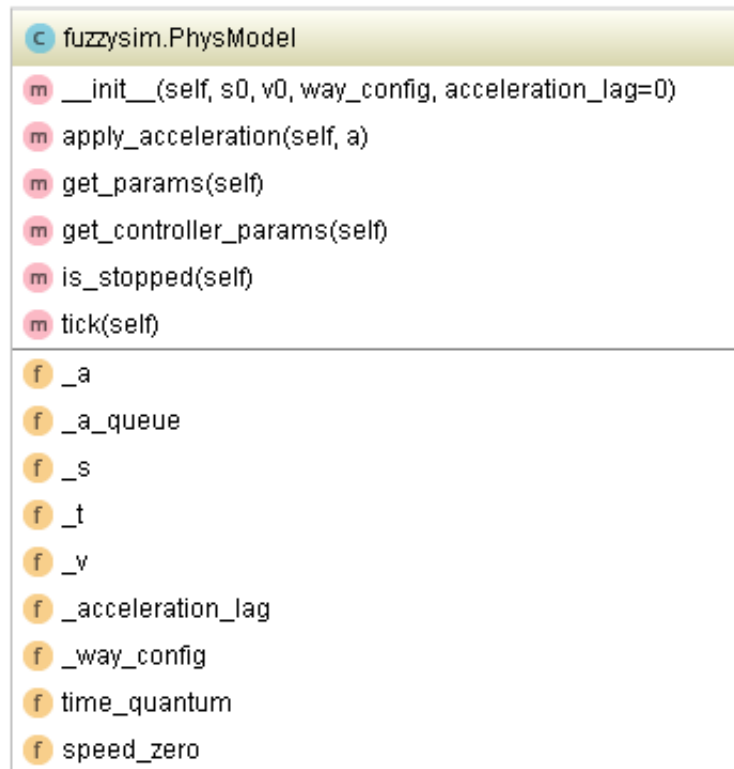


Рисунок 3.3 – Структура класса fuzzisim.fuzzisim.PhysModel

Поля класса fuzzisim.fuzzisim.PhysModel:

- `_a`. Текущее фактическое ускорение в метрах в секунду за секунду.
- `_a_queue`. Очередь управляющих сигналов контроллера платформы, устанавливающих ускорение. Применяется для корректной обработки задержек в системе торможения.
- `_s`. Текущее расстояние для препятствия, в метрах.
- `_t`. Текущий момент модельного времени, в секундах.
- `_v`. Текущая скорость, в метрах в секунду.
- `_acceleration_lag`. Величина задержки в системе торможения, в секундах.
- `_way_config`. Конфигурация пути. Представлена экземпляром класса `fuzzisim.fuzzisim.WayConfig`. Используется для определения предельных ускорений на данном участке пути, а также для определения дополнительных сил, действующих на платформу.
- `time_quantum`. Величина одного кванта модельного времени, в секундах.

– `speed_zero`. Значение скорости в метрах в секунду, при котором платформа считается остановившейся.

Методы класса `fuzzisim.fuzzisim.PhysModel`:

– `__init__(self, s0, v0, way_config, acceleration_lag=0)`. Инициализирует физическую модель. В качестве входных аргументов принимает начальную скорость платформы в метрах в секунду, начальное расстояние до препятствия в метрах, объект конфигурации пути, значение задержек в системе управления в секундах. начальный момент времени и начальное ускорения принимаются равными нулю.

– `apply_acceleration(self, a)`. Предоставляет интерфейс контроллеру платформы, позволяющий послать физической модели управляющий сигнал, на изменение ускорения. В качестве входного параметра принимает значение ускорения в метрах в секунду за секунду

– `get_params(self)`. Предоставляет интерфейс менеджеру симуляции для сбора телеметрии, Возвращает кортеж значений (`t, s, v, a`): значения текущей метки времени в секундах, расстояния до препятствия в метрах, скорости в метрах в секунду, ускорения в метрах в секунду за секунду.

– `get_controller_params(self)`. Предоставляет интерфейс контроллеру платформы, позволяющий получить симулированные данные от внешних датчиков расстояния, скорости и ускорения. Возвращает кортеж значений (`s, v, a`): значения расстояния до препятствия в метрах, скорости в метрах в секунду, ускорения в метрах в секунду за секунду

– `is_stopped(self)`. Возвращает истину в случае, если скорость платформы меньше `speed_zero`

– `tick(self)`. Осуществляет расчет состояния системы в следующий квант времени.

```
def tick(self):
    """
    process tick

    :return: True if platform stopped in this quant
    """
    self._t += self.time_quantum

    # apply a from queue
    for i, v in enumerate(self._a_queue):
        if v[0] <= self._t:
            self._a = v[1]
            del self._a_queue[i]

    self._a = self._way_config.get_effective_a(self._s, self._a)

    self._v += self._a * self.time_quantum
    self._s -= self._v * self.time_quantum + (self._a * self.
        time_quantum ** 2) / 2

    if self._s <= 0 and not self.is_stopped():
```

```

raise CollisionError(self._t, self._s, self._v, self._a)

# if distance less then 1 quantum zero speed distance, assume
  it is 0
if abs(self._s) < self.time_quantum * self.speed_zero:
    self._s = 0

```

### 3.1.6.2 fuzzisim.fuzzisim.CartController

Реализует модель контроллера платформы, при этом алгоритм расчета ускорения делегируется модулю `brake_controller`. При этом используется шаблон проектирования «стратегия».

Паттерн стратегия является настолько распространенным и общепринятым, что многие его используют постоянно, даже не задумываясь о том, что это хитроумный паттерн проектирования, расписанный когда-то GOF.

Каждый второй раз, когда мы пользуемся наследованием, мы используем Стратегию; каждый раз, когда мы абстрагируемся от некоторого процесса, поведения или алгоритма – мы используем стратегию. Сортировка, анализ данных, валидация, разбор данных, сериализация, кодирование, получение конфигурации, все эти концепции могут и должны быть выражены в виде стратегии или политики (*policy*).

Стратегия является фундаментальным паттерном, поскольку она проявляется в большинстве других классических паттернов, которые поддерживают специализацию за счет наследования. Абстрактная фабрика – это стратегия создания семейства объектов; фабричный метод – стратегия создания одного объекта; строитель – стратегия построения объекта; итератор – стратегия перебора элементов и т.д.

При этом мы не хотим, чтобы анализатор паттернов (`PatternsAnalyzer`) знал, какая конкретно стратегия определения паттернов используется, поэтому вместо использования конкретного типа, анализатор будет принимать детектор через аргументы конструктора. Таким образом мы не просто абстрагируемся от процесса определения паттернов, но и позволяем клиентскому коду, который использует анализатор паттернов определять стратегию самостоятельно.

По определению, применение стратегии обусловлено двумя причинами: инкапсуляция поведения или алгоритма и возможность замены поведения или алгоритма во время исполнения. Любой нормально спроектированный класс уже инкапсулирует в себе поведение или алгоритм, но не любой класс с некоторым поведением является или должен быть стратегией. Стратегия нужна тогда, когда важно иметь возможность заменить его во время исполнения!

Другими словами, стратегия обеспечивает точку расширения системы в определенной плоскости: класс-потребитель стратегии не знает, как

выполняется некоторое действие и кто именно его выполняет; об этом знают классы более высокого уровня.

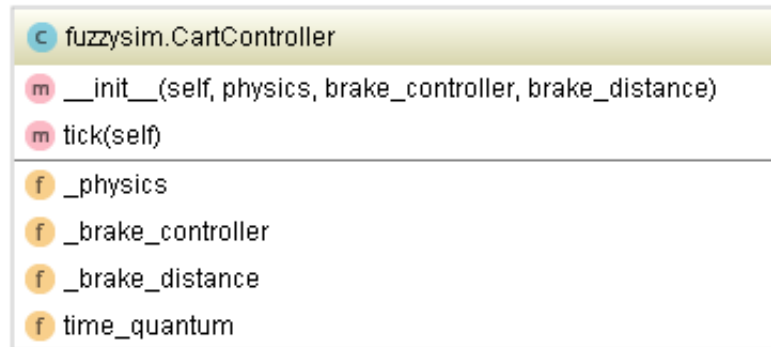


Рисунок 3.4 – Структура класса fuzzysim.fuzzysim.CartController

Поля класса fuzzysim.fuzzysim.CartController:

- `_physics`. Объект физической модели, требуется для получения алгоритмом данных о расстоянии, скорости и ускорении от стимулированных датчиков.

- `_brake_controller`. Стратегия алгоритма торможения, подпакет пакета `fuzzysim.brake_controller`.

- `_brake_distance`. Расстояние от препятствия в метрах, на котором надо затормозить.

Методы класса fuzzysim.fuzzysim.CartController:

- `__init__(self, physics, brake_controller, brake_distance)`. Инициализирует поля контроллера.

- `tick(self)`. Вызывает выполнение алгоритма торможения для следующего кванта времени.

```
class CartController:
    time_quantum = 0.001 # time quantum in seconds

    def __init__(self, physics, brake_controller,
                 brake_distance):
        brake_controller.time_quantum = self.time_quantum

        self._physics = physics
        self._brake_controller = brake_controller
        self._brake_distance = brake_distance

    def tick(self):
        (s, v, a) = self._physics.get_controller_params()

        applying_a = self._brake_controller.get_a(s, v, a)
        self._physics.apply_acceleration(applying_a)
```

### 3.1.6.3 fuzzysim.fuzzysim.WayConfig

Класс `fuzzysim.fuzzysim.WayConfig` реализует характеристики пути: максимальное применимое ускорение и добавочное

ускорение на отдельных участках пути. Это позволяет симулировать области с пониженным коэффициентом сцепления с дорогой, ветер, подъемы и спуски. Участки с разными характеристиками могут накладываться друг на друга, в этом случае их параметры применяются последовательно. При этом сначала применяется первый описанный участок пути, затем второй и так далее.

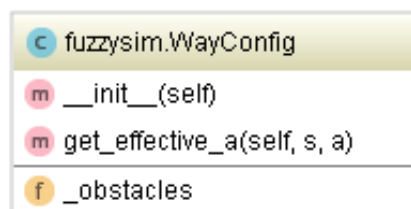


Рисунок 3.5 – Структура класса fuzzisim.fuzzisim.WayConfig

Поля класса fuzzisim.fuzzisim.WayConfig:

- `_obstacles`. Массив объектов класса `Obstacle`, описывающих конфигурацию пути.

Методы класса fuzzisim.fuzzisim.WayConfig:

- `__init__(self)`. Инициализирует поля контроллера.
- `get_effective_a(self, s, a)`. Рассчитывает эффективное ускорение в данной точке пути с учетом всех имеющихся участков. Входные аргументы: расстояние до препятствия в метрах, применяемое ускорение в метрах в секунду за секунду. Выходные параметры: рассчитанное эффективное ускорение в метрах в секунду за секунду.

```

def get_effective_a(self, s, a):
    for o in self._obstacles:
        a = o.get_effective_a(s, a)
    return a
  
```

#### 3.1.6.4 fuzzisim.fuzzisim.Obstacle

Реализует параметры конкретного участка пути. Позволяет устанавливать следующие параметры:

- Границы участка. Отметки расстояния от точки торможения, задающие конфигурируемый участок.
- Константное добавочное ускорение. Позволяет моделировать наклонные участки пути.
- Максимальное применяемое ускорение. Позволяет моделировать ограничения коэффициента сцепления с дорогой: обледенелые, мокрые, изношенные участки пути.

Поля класса fuzzisim.fuzzisim.Obstacle:

- `start`. Начало участка, в метрах от препятствия.

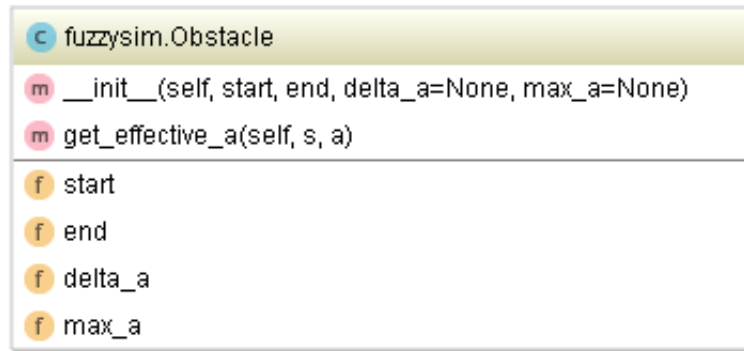


Рисунок 3.6 – Структура класса fuzzisim.fuzzisim.Obstacle

- end. Конец участка в метрах от препятствия.
- delta\_a. Константное ускорение на участке в метрах в секунду за секунду
- max\_a. Максимальное ускорение на участке в метрах в секунду за секунду

Методы класса fuzzisim.fuzzisim.Obstacle:

- \_\_init\_\_(self, start, end, delta\_a=None, max\_a=None). Инициализирует поля объекта.
- get\_effective\_a(self, s, a). Рассчитывает эффективное ускорение на данном участке пути. Входные аргументы: расстояние до препятствия в метрах, применяемое ускорение в метрах в секунду за секунду. Выходные параметры: рассчитанное эффективное ускорение в метрах в секунду за секунду.

```

class Obstacle:
    def __init__(self, start, end, delta_a=None, max_a=None):
        if start > end:
            (start, end) = (end, start)

        self.start = start
        self.end = end
        self.max_a = max_a if max_a is not None else 100000
        self.delta_a = delta_a if delta_a is not None else 0

    def get_effective_a(self, s, a):
        if not self.start < s < self.end:
            return a

        a = math.copysign(min(abs(a), self.max_a), a)
        a += self.delta_a

        return a

```

### 3.1.6.5 fuzzisim.fuzzisim.CollisionError

Исключение, возникающее при столкновении. Генерируется в fuzzisim.fuzzisim.Simulator.tick().

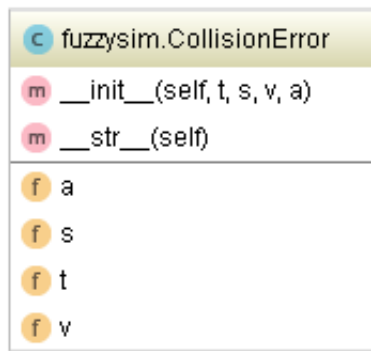


Рисунок 3.7 – Структура класса fuzzysim.fuzzysim.CollisionError

Поля класса fuzzysim.fuzzysim.CollisionError:

- t. значения текущей метки времени в секундах.
- s. расстояния до препятствия в метрах.
- v. скорости в метрах в секунду.
- a. ускорения в метрах в секунду за секунду.

Методы класса fuzzysim.fuzzysim.CollisionError:

- \_\_init\_\_(self, t, s, v, a). Инициализирует поля объекта.
- \_\_str\_\_(self). Преобразует исключение в строку.

### 3.1.7 Блок моделирования нечетких алгоритмов

Блок моделирования нечетких алгоритмов реализует разнообразные варианты нечетких алгоритмов, в частности Мамдани. Приведенные в нем реализации основаны на алгоритме Мамдани, и, в свою очередь, могут рассматриваться как основа для пользовательских реализаций контроллеров. Благодаря тому, что выбранный язык разработки является интерпретируемым, добавление новых реализаций не требует компилятора или другого подобного ПО, достаточно просто интерпретатора языка и текстового редактора.

Рассмотрим подробнее структуру блока. Он состоит из следующих классов:

- fuzzysim.brake\_controller.naive.Controller;
- fuzzysim.brake\_controller.custom\_fuzzy.Controller;
- fuzzysim.mamdani.MamdaniAlgorithm;
- fuzzysim.mamdani.Rule;
- fuzzysim.mamdani.Conclusion;
- fuzzysim.mamdani.Cond;
- fuzzysim.mamdani.FuzzyValue;
- fuzzysim.mamdani.RectangleFuzzyValue;
- fuzzysim.mamdani.IntervalFuzzyValue;
- fuzzysim.mamdani.Membership;
- fuzzysim.mamdani.Variable;

```

- fuzzisim.mamdani.Term;
- fuzzisim.mamdani.RectangleTerm;
- fuzzisim.mamdani.IntervalTerm.

class CollisionError(RuntimeError):
    def __init__(self, t, s, v, a):
        self.t = t
        self.s = s
        self.v = v
        self.a = a

    def __str__(self):
        return "Collision happens: T=%.3f S=%.3f V=%.3f A=%.3f" % (self.t, self.s, self.v, self.a)

```

### 3.1.7.1 fuzzisim.brake\_controller.naive.Controller

Простой неадаптивный алгоритм торможения, используемый для калибровки и обучения нечетких алгоритмов. Основан на том, что при плавной остановке ускорение нарастает линейно до середины времени торможения, а максимальная его величина равна удвоенному ускорению при равнозамедленном движении. Ниже приведен код основного метода этого алгоритма.

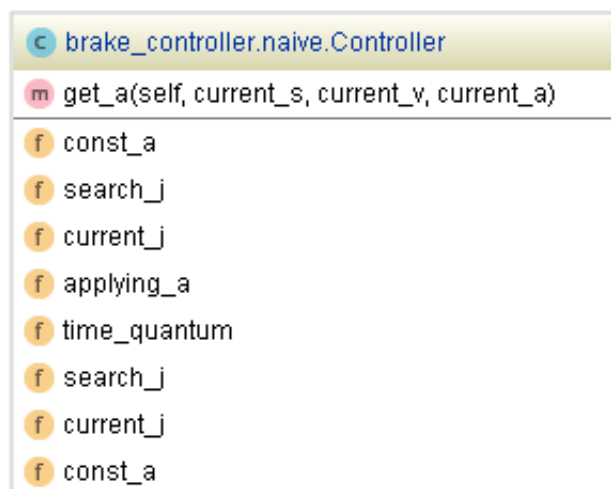


Рисунок 3.8 – Структура класса fuzzisim.brake\_controller.naive.Controller

Поля класса fuzzisim.brake\_controller.naive.Controller:

- applying\_a. Текущее применяемое ускорение, в метрах в секунду за секунду.
- time\_quantum. Величина кванта времени, в секундах.
- search\_j. Модуль расчетного значения рывка, в метрах в секунду за секунду за секунду.
- current\_j. Текущее значение рывка, в метрах в секунду за секунду за секунду.



– const\_a. Ускорение необходимое для остановки при равнозамедленном движении.

Методы класса fuzzisim.brake\_controller.naive.Controller:

– get\_a(self, current\_s, current\_v, current\_a).

Возвращает рассчитанное ускорение для текущих расстояния, скорости и фактического ускорения.

```
def get_a(self, current_s, current_v, current_a):
    if self.search_j is None:
        self.const_a = -current_v ** 2 / (2 * current_s)
        time_a_const = 2 * current_s / current_v

        self.search_j = abs((self.const_a - current_a) * 2 / (
            time_a_const / 2))
        self.current_j = -self.search_j

    if self.current_j == -self.search_j:
        self.current_j = math.copysign(self.search_j, self.const_a
            * 2 - self.applying_a)

    self.applying_a += self.current_j * self.time_quantum

    return self.applying_a
```

### 3.1.7.2 fuzzisim.brake\_controller.custom\_fuzzy.Controller

Адаптивный, настраиваемый алгоритм контроллера торможения на основе алгоритма Мамдани. Поддерживает треугольную и интервальную форму функций принадлежности, позволяет загружать термы и правила из конфигурационного json-файла.

Приведем пример конфигурационного файла для трех входных и трех выходных термов:

```
{
  "rules": [
    {"generated":1, "conds": [ ["S",0], ["V",0]], "concs": [ ["A",
      2]]},
    {"generated":1, "conds": [ ["S",0], ["V",1]], "concs": [ ["A",
      2]]},
    {"generated":1, "conds": [ ["S",0], ["V",2]], "concs": [ ["A",
      1]]},
    {"generated":1, "conds": [ ["S",1], ["V",0]], "concs": [ ["A",
      0]]},
    {"generated":1, "conds": [ ["S",1], ["V",1]], "concs": [ ["A",
      0]]},
    {"generated":1, "conds": [ ["S",1], ["V",2]], "concs": [ ["A",
      2]]},
    {"generated":1, "conds": [ ["S",2], ["V",0]], "concs": [ ["A",
      2]]},
    {"generated":1, "conds": [ ["S",2], ["V",1]], "concs": [ ["A",
      1]]},
    {"generated":1, "conds": [ ["S",2], ["V",2]], "concs": [ ["A",
      1]]},
  ],
}
```

```

    ],
    "S": {
      "max": 99.98, "min": 0.0493,
      "peaks": [ 16.7, 50, 83.3 ],
    },
    "A": {
      "max": 5, "min": 0.0,
      "peaks": [ 1, 2, 4 ],
    },
    "V": {
      "max": 20, "min": 0.0,
      "peaks": [ 3.3, 10, 14 ],
      "interval": 0
    },
  },
}

```

Здесь поле `rules` задает набор нечетких правил. Каждое правило состоит из трех полей:

- `generated`. Флаг, показывающий, что поле сгенерировано автоматически;
- `conds`. Список условий, каждое из которых является списком из двух элементов, имени переменной и номера терма соответственно;
- `concs`. Список заключений, каждое из которых является списком из двух элементов, имени переменной и номера терма соответственно.

При работе алгоритма полагаем, что условия объединяются через AND.

Поле `S` описывает входную лингвистическую переменную расстояния до препятствия:

- `max`. Верхняя граница области значений лингвистической переменной, в метрах;
- `min`. Нижняя граница области значений лингвистической переменной, в метрах;
- `peaks`. Список пиков функции принадлежности (вершин треугольников треугольной функции, либо середин интервалов интервальных функций принадлежности), в метрах;
- `interval`. Флаг, указывающий, является ли функция принадлежности интервальной.

Поле `V` описывает входную лингвистическую переменную скорости:

- `max`. Верхняя граница области значений лингвистической переменной, в метрах в секунду;
- `min`. Нижняя граница области значений лингвистической переменной, в метрах в секунду;
- `peaks`. Список пиков функции принадлежности (вершин треугольников треугольной функции, либо середин интервалов интервальных функций принадлежности), в метрах в секунду;
- `interval`. Флаг, указывающий, является ли функция принадлежности интервальной.

Поле `A` описывает выходную лингвистическую переменную ускорения:

- `max`. Верхняя граница области значений лингвистической

переменной, в метрах в секунду за секунду;

- min. Нижняя граница области значений лингвистической переменной, в метрах в секунду за секунду;

- peaks. Список пиков функции принадлежности (вершин треугольников треугольной функции, либо середин интервалов интервальных функций принадлежности), в метрах в секунду за секунду;

- interval. Флаг, указывающий, является ли функция принадлежности интервальной.

Процесс загрузки конфига выглядит следующим образом:

```
config = json.load(open(config_file))
if current_const_a is not None:
    src_const_a = config['V']['max'] ** 2 / (2 * config['S']['max'])
    ratio = current_const_a / src_const_a
    config['A']['peaks'] = [i * ratio for i in config['A']['peaks']]
    config['A']['max'] *= ratio

in_variables = {
    'S': get_variable('S', config['S']),
    'V': get_variable('V', config['V']),
}
out_variables = {'A': get_variable('A', config['A']), }

rules = get_rules(**in_variables, **out_variables, config['rules'])
```

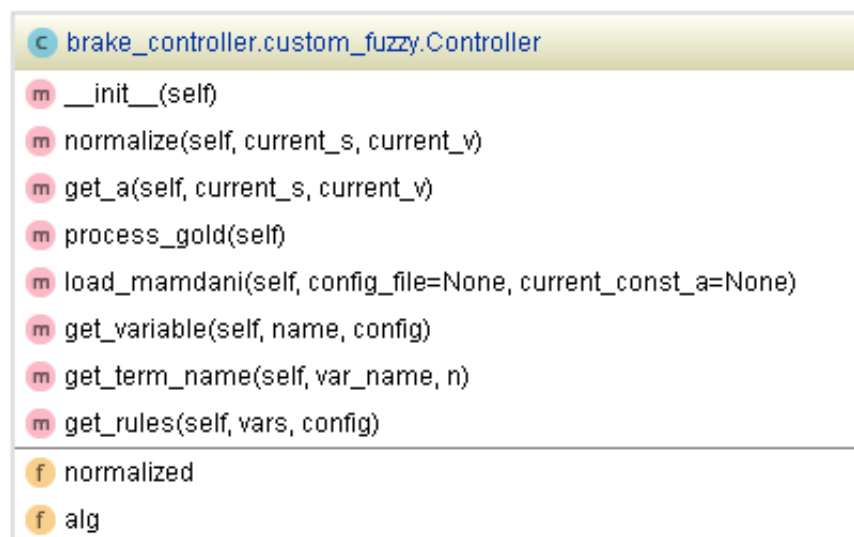


Рисунок 3.9 – Структура класса  
fuzzisim.brake\_controller.custom\_fuzzy.Controller

Поля класса fuzzisim.brake\_controller.custom\_fuzzy.Controller:

- normalized. Флаг, указывающий, была ли проведена нормализация правил и лингвистических переменных исходя из начальных скорости и

расстояния.

- alg. Экземпляр fuzzisim.mamdani.MamdaniAlgorithm, который используется для получения выходных значений.

Методы класса fuzzisim.brake\_controller.custom\_fuzzy.Controller:

- `__init__(self)` — Инициализирует поля класса.
- `normalize(self, current_s, current_v)`. Нормализует правила и лингвистические переменные. На вход принимает текущую скорость в метрах в секунду и расстояние в метрах.
- `get_a(self, current_s, current_v)`. Возвращает расчетное значение ускорения в метрах в секунду за секунду. На вход принимает текущую скорость в метрах в секунду и расстояние в метрах.
- `load_mamdani(self, config_file=None, current_const_a=None)` — Загружает лингвистические переменные и правила из файла и возвращает сконфигурированный экземпляр fuzzisim.mamdani.MamdaniAlgorithm.
- `get_variable(self, name, config)`. Извлекает из конфигурационного объекта, конструирует и возвращает экземпляр fuzzisim.mamdani.Variable, представляющий лингвистическую переменную.
- `get_term_name(self, var_name, n)`. Возвращает строку с именем терма. На вход принимает имя переменной и номер терма.
- `get_rules(self, vars, config)`. Возвращает массив объектов fuzzisim.mamdani.Rule, представляющих лингвистические правила алгоритма Мамдани.

### 3.1.7.3 fuzzisim.mamdani.MamdaniAlgorithm

Реализует алгоритм Мамдани в двух вариантах: классическом и ускоренном. Ускоренный вариант алгоритма Мамдани описан в [4]. Данная реализация этапа дефаззификации основана на том, что при интервальном задании выходной функции принадлежности появляется возможность подсчета центра масс не путем численного интегрирования, а путем сложения активированных интервалов выходной функции принадлежности, что позволяет уменьшить количество итераций в алгоритме дефаззификации на несколько порядков.

На рис. 3.8 показаны графики  $V(t)^M$ ,  $S(t)^M$  для классического и упрощенного вариантов реализации алгоритма Мамдани в идеальных условиях – а, при введении временной инерционности системы торможения – б, при введении условий слабого сцепления платформы с дорогой на одном из участков.

Из сравнительного анализа результатов моделирования видно, что алгоритмы нечеткого управления позволяют выполнить торможение в режиме, близком к эталонному. При этом графики классического и упрощенного вариантов реализации алгоритма Мамдани, практически

сливаются. Это говорит о том, что качество алгоритмов нечеткого контроллера при упрощенном варианте реализации не ухудшается. [4]

Отдельный интерес представляет процесс применения правил:

```
def apply_rules(self, in_fuzzy_values):
    out_fuzzy_values = {}
    for r in self.rules:
        memberships = list(
            in_fuzzy_values[c.variable.name].
            get_membership(c.term)
            for c
            in r.conditions
        )
        aggregated_degree = min(memberships).degree

        for c in r.conclusions:
            if c.variable.name not in
                out_fuzzy_values:
                out_fuzzy_values[c.variable.name] =
                    c.variable.get_fuzzy_value(None
                )

            out_fuzzy_values[c.variable.name].
                add_membership(Membership(c.term,
                    aggregated_degree))

    return out_fuzzy_values
```

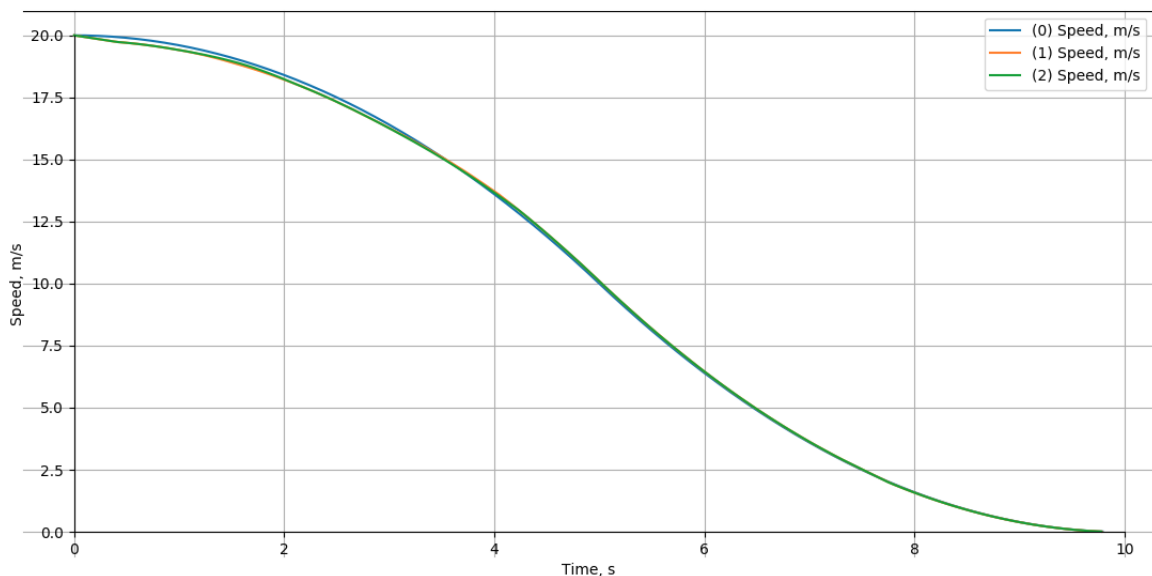


Рисунок 3.10 – Графики скорости для экспериментов с эталонным, классическим алгоритмом Мамдани и ускоренным алгоритмом Мамдани.

Поля класса `fuzzisim.mamdani.MamdaniAlgorithm`:  
 – `out_variables`. Список объектов класса `fuzzisim.mamdani.Variable`, выходных лингвистических переменных.

mamdani.MamdaniAlgorithm	
m	<code>__init__(self, in_variables, out_variables, rules)</code>
m	<code>fuzzificate(self, in_crisp_values)</code>
m	<code>apply_rules(self, in_fuzzy_values)</code>
m	<code>defuzzificate(self, out_fuzzy_values)</code>
m	<code>process(self, in_crisp_values)</code>
f	<code>out_variables</code>
f	<code>in_variables</code>
f	<code>rules</code>

Рисунок 3.11 – Структура класса `fuzzisim.mamdani.MamdaniAlgorithm`

– `in_variables`. Список объектов класса `fuzzisim.mamdani.Variable`, входных лингвистических переменных.

– `rules` — Список объектов класса `fuzzisim.mamdani.Rule`, лингвистических правил алгоритма Мамдани.

Методы класса `fuzzisim.mamdani.MamdaniAlgorithm`:

– `__init__(self, in_variables, out_variables, rules)`. Инициализирует поля объекта.

– `fuzzificate(self, in_crisp_values)`. Фаззифицирует входные переменные. Получает на вход словарь конкретных значений входных переменных, возвращает словарь с фаззифицированными значениями `fuzzisim.mamdani.FuzzyValue`.

– `apply_rules(self, in_fuzzy_values)`. Осуществляет агрегирование подусловий, активацию подзаключений и аккумулярование подзаключений. Получает на вход входные фаззифицированные значения, возвращает словарь с выходными фаззифицированными значениями.

– `defuzzificate(self, out_fuzzy_values)`. Производит дефаззификацию. Дефаззификация происходит по методу центра масс. На вход получает фаззифицированные значения выходных переменных, возвращает точные значения выходных переменных.

– `process(self, in_crisp_values)`. Исполняет алгоритм в целом. На вход получает точные значения входных переменных, возвращает точные значения выходных переменных.

#### 3.1.7.4 `fuzzisim.mamdani.Rule`

Реализация лингвистического правила алгоритма Мамдани. Принимается, что условия объединены логическим AND.

Поля класса `fuzzisim.mamdani.Rule`:

– `conclusions`. Список объектов класса `fuzzisim.mamdani.Conclusion`, заключений алгоритма Мамдани.

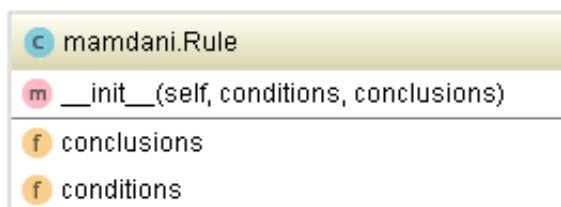


Рисунок 3.12 – Структура класса fuzzisim.mamdani.Rule

– conditions. Список объектов класса fuzzisim.mamdani.Cond, условий алгоритма Мамдани.

Методы класса fuzzisim.mamdani.Rule:

– \_\_init\_\_(self, conditions, conclusions).

Инициализирует поля объекта.

### 3.1.7.5 fuzzisim.mamdani.Conclusion

Реализация заключения алгоритма Мамдани.

Поля класса fuzzisim.mamdani.Conclusion:

– variable. Экземпляр класса fuzzisim.mamdani.Variable, лингвистическая переменная алгоритма Мамдани.

– term. Экземпляр класса fuzzisim.mamdani.Term, терм лингвистической переменной алгоритма Мамдани.

Методы класса fuzzisim.mamdani.Conclusion:

– \_\_init\_\_(self, variable, term). Инициализирует поля объекта.

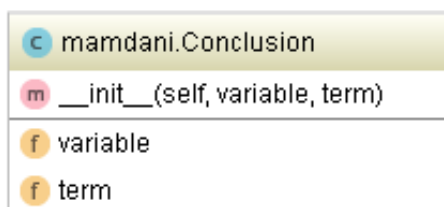


Рисунок 3.13 – Структура класса fuzzisim.mamdani.Conclusion

```
class Conclusion:
    def __init__(self, variable, term):
        self.variable = variable
        self.term = term
```

### 3.1.7.6 fuzzisim.mamdani.Cond

Реализация условия алгоритма Мамдани.

Поля класса fuzzisim.mamdani.Cond:

– variable. Экземпляр класса fuzzisim.mamdani.Variable, лингвистическая переменная алгоритма Мамдани.

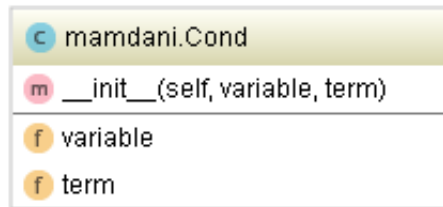


Рисунок 3.14 – Структура класса fuzzisim.mamdani.Cond

– term. Экземпляр класса fuzzisim.mamdani.Term, терм лингвистической переменной алгоритма Мамдани.

Методы класса fuzzisim.mamdani.Cond:

– \_\_init\_\_(self, variable, term). Инициализирует поля объекта.

```

class Cond:
    def __init__(self, variable, term):
        self.variable = variable
        self.term = term

```

### 3.1.7.7 fuzzisim.mamdani.FuzzyValue

Базовая реализация нечеткого значения для классического алгоритма Мамдани.

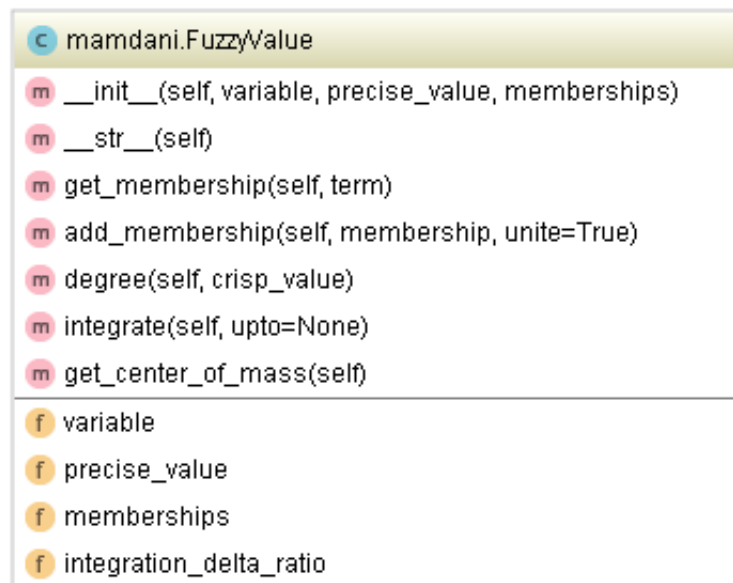


Рисунок 3.15 – Структура класса fuzzisim.mamdani.FuzzyValue

Поля класса fuzzisim.mamdani.FuzzyValue:

– variable. Экземпляр класса fuzzisim.mamdani.Variable, лингвистическая переменная алгоритма Мамдани.

– precise\_value. Точное значение переменной.



– `memberships`. Список экземпляров класса `fuzzisim.mamdani.Membership`, признак принадлежности значения терму лингвистической переменной.

– `integration_delta_ratio`. Коэффициент шага интегрирования для классической дефаззификации.

Методы класса `fuzzisim.mamdani.FuzzyValue`:

– `__init__(self, variable, precise_value, memberships)`. Инициализирует поля класса.

– `__str__(self)`. Преобразует объект в строку для дальнейшего вывода.

– `get_membership(self, term)`. Возвращает объект класса `fuzzisim.mamdani.Membership`, степень принадлежности значения данному терму.

– `add_membership(self, membership, unite=True)`. Устанавливает принадлежность данному терму. Если аргумент `unite` равен `True`, то в случае существования принадлежности данному терму будет произведено объединение принадлежностей по `max()`. В противном случае будет возбуждено исключение.

– `degree(self, crisp_value)`. Возвращает активированное значение при аккумуляции заключений.

– `integrate(self, upto=None)`. Производит классическое интегрирование по области значений переменной с шагом, определяемым `integration_delta_ratio`. Если установлен аргумент `upto`, останавливает интегрирование при достижении значения равного `upto`, возвращает полученное значение, а также соответствующее ему значение лингвистической переменной

– `get_center_of_mass(self)`. Находит центр масс используя метод `integrate`. Возвращает скалярное значения центра масс.

### 3.1.7.8 `fuzzisim.mamdani.IntervalFuzzyValue`

Дочерний класс `fuzzisim.mamdani.FuzzyValue`. Реализует нечеткое значение с интервальным заданием функции принадлежности для ускоренной дефаззификации. Накладывает ограничения на выходную лингвистическую переменную:

– Термы выходной переменной должны быть заданы на непрерывном интервале;

– Термы выходной переменной должны отсортированы в порядке следования по области определения.

Поля класса `fuzzisim.mamdani.IntervalFuzzyValue`:

– `sorted_memberships`. Список экземпляров класса `fuzzisim.mamdani.Membership`, сортированный по началу области определения соответствующего терма.

Методы класса `fuzzisim.mamdani.IntervalFuzzyValue`:







	mamdani.IntervalFuzzyValue
	<code>__init__(self, variable, precise_value, memberships)</code>
	<code>add_membership(self, membership, unite=True)</code>
	<code>get_membership_mass(self, membership)</code>
	<code>get_center_of_mass(self)</code>
	<code>sorted_memberships</code>

Рисунок 3.16 – Структура класса fuzzisim.mamdani.IntervalFuzzyValue

– `__init__(self, variable, precise_value, memberships)`. Инициализирует поля класса.

– `get_membership_mass(self, membership)`. Возвращает вес принадлежности к терму: произведение ширины терма на степень принадлежности к терму.

– `add_membership(self, membership, unite=True)`. Устанавливает принадлежность данному терму. Если аргумент `unite` равен `True`, то в случае существования принадлежности данному терму будет произведено объединение принадлежностей по `max()`. В противном случае будет возбуждено исключение. После добавления принадлежности.

– `get_center_of_mass(self)`. Производит дефаззификацию нечеткого значения по ускоренному алгоритму. Для этого вычисляются массы каждой принадлежности к каждому терму и суммируются для получения полной массы полигона. Затем итеративно в счетчике суммируются массы принадлежностей до тех пор, пока масса суммы не станет больше половины массы всего полигона. Затем от полученного точного значения отнимается доля ширины последней принадлежности, пропорциональна превышению суммы над половиной массы полигона.

Заслуживает внимания реализация получения центра масс:

```
def get_center_of_mass(self):
    masses = [0] * len(self.sorted_memberships)
    total_mass = 0

    for i, m in enumerate(self.sorted_memberships):
        masses[i] = self.get_membership_mass(m)
        total_mass += masses[i]

    if total_mass == 0:
        result = (self.variable.max + self.variable.min)/2
        return result

    half_mass = total_mass / 2
    center = 0

    for i, v in enumerate(masses):
        half_mass -= v

        if half_mass <= 0:
```

```

        ratio = 1 - abs(half_mass / v)
        part=ratio*self.sorted_memberships[i].term.width
center += part

    return center

center += self.sorted_memberships[i].term.width

```

### 3.1.7.9 fuzzisim.mamdani.RectagleFuzzyValue

Дочерний класс `fuzzisim.mamdani.IntervalFuzzyValue`. Реализует нечеткое значение с прямоугольным заданием функции принадлежности для ускоренной дефазификации. Переопределяет метод `get_membership_mass`, учитывая высоту терма.

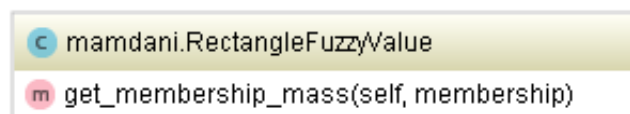


Рисунок 3.17 – Структура класса `fuzzisim.mamdani.RectagleFuzzyValue`

Методы класса `fuzzisim.mamdani.RectagleFuzzyValue`:  
 – `get_membership_mass(self, membership)`. Возвращает вес принадлежности к терму: произведение ширины терма на степень принадлежности к терму.

### 3.1.7.10 fuzzisim.mamdani.Membership

Реализует абстракцию принадлежности значения к терму.

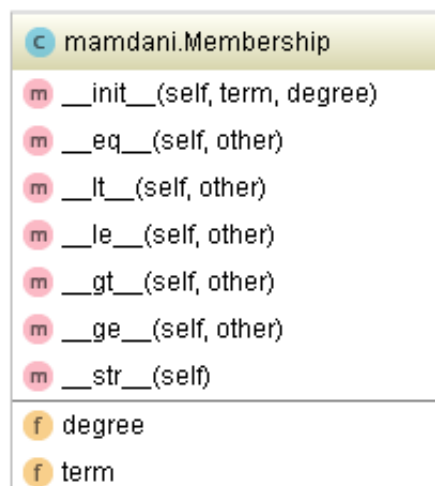


Рисунок 3.18 – Структура класса `fuzzisim.mamdani.Membership`

Поля класса `fuzzisim.mamdani.Membership`:

– degree. Степень принадлежности к терму. Число от 0 до 1 включительно.

– term. Экземпляр класса fuzzisim.mamdani.Term, терм лингвистической переменной алгоритма Мамдани.

Методы класса fuzzisim.mamdani.Membership:

– \_\_init\_\_(self, term, degree). Инициализирует поля объекта.

– \_\_eq\_\_(self, other). Магический метод «равно» объектов класса fuzzisim.mamdani.Membership

– \_\_lt\_\_(self, other). Магический метод «меньше, чем» объектов класса fuzzisim.mamdani.Membership

– \_\_le\_\_(self, other). Магический метод «меньше или равно» объектов класса fuzzisim.mamdani.Membership

– \_\_gt\_\_(self, other). Магический метод «больше, чем» объектов класса fuzzisim.mamdani.Membership

– \_\_ge\_\_(self, other). Магический метод «больше или равно» объектов класса fuzzisim.mamdani.Membership

– \_\_str\_\_(self). Магический метод преобразования в строку объектов класса fuzzisim.mamdani.Membership

### 3.1.7.11 fuzzisim.mamdani.Variable

Реализует лингвистическую переменную алгоритма Мамдани. Позволяет получить нечеткое значение на основе конкретного.

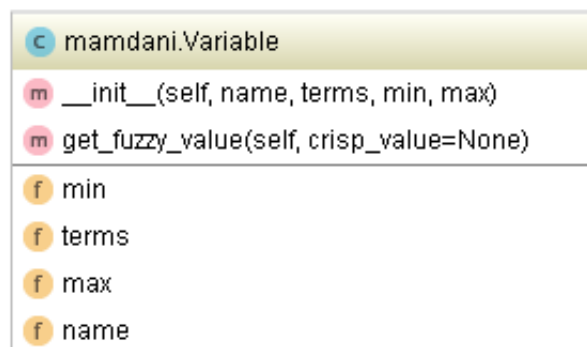


Рисунок 3.19 – Структура класса fuzzisim.mamdani.Variable

Поля класса fuzzisim.mamdani.Variable:

– min. Нижняя граница области определения лингвистической переменной, скаляр.

– max. Верхняя граница области определения лингвистической переменной, скаляр.

– terms. Список экземпляров класса fuzzisim.mamdani.Term, задает термы лингвистической переменной алгоритма Мамдани.

– name. Имя лингвистической переменной алгоритма Мамдани.

Методы класса `fuzzisim.mamdani.Variable`:

- `__init__(self, name, terms, min, max)`.

Инициализирует поля объекта.

- `get_fuzzy_value(self, crisp_value=None)`. Возвращает нечеткое значение `fuzzisim.mamdani.FuzzyValue` данной переменной, соответствующее скалярному значению аргумента функции.

Получение нечеткого значения из четкого:

```
def get_fuzzy_value(self, crisp_value=None):
    if crisp_value is None:
        term = list(self.terms.values())[0]

        if type(term) is IntervalTerm:
            return IntervalFuzzyValue(self,
                                       crisp_value, {})
        elif type(term) is RectangleTerm:
            return RectangleFuzzyValue(self,
                                       crisp_value, {})
        else:
            return FuzzyValue(self, None, {})

    memberships = {i: Membership(v, v.degree(crisp_value))
                   for i, v in self.terms.items()}
    return FuzzyValue(self, crisp_value, memberships)
```

### 3.1.7.12 `fuzzisim.mamdani.Term`

Базовый класс, реализует терм лингвистической переменной классического алгоритма Мамдани с трапецидальной функцией принадлежности.

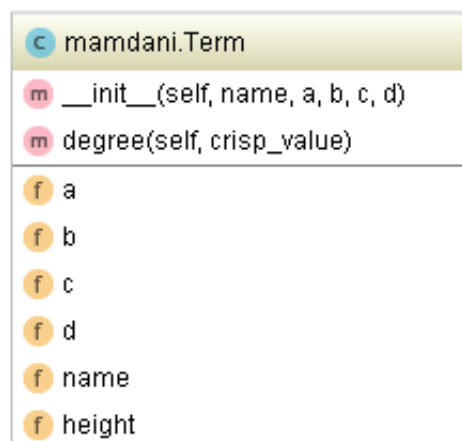


Рисунок 3.20 – Структура класса `fuzzisim.mamdani.Term`

Поля класса `fuzzisim.mamdani.Term`:

- `a`. Начало области определения терма, скаляр.
- `b`. Начало интервала, где значение функции принадлежности терма становится равным высоте терма, скаляр.

- c. Конец интервала, где значение функции принадлежности терма становится равным высоте терма, скаляр.
  - d. Конец области определения терма, скаляр.
  - height. Высота терма, максимальное значение функции принадлежности терма.
  - name. Имя терма лингвистической переменной алгоритма Мамдани.
- Методы класса `fuzzisim.mamdani.Term`:
- `__init__(self, name, a, b, c, d)`. Инициализирует поля объекта.
  - `degree(self, crisp_value)`. Возвращает степень принадлежности скалярного значения терму.
- Определение степени принадлежности:

```
def degree(self, crisp_value):
    if self.a < crisp_value < self.b:
        return (crisp_value - self.a) / (self.b - self.a)
    elif self.b <= crisp_value <= self.c:
        return 1
    elif self.c < crisp_value < self.d:
        return (self.d - crisp_value) / (self.d - self.c)
    else:
        return 0
```

### 3.1.7.13 fuzzisim.mamdani.RectangleTerm

Реализует терм лингвистической переменной ускоренного алгоритма Мамдани с прямоугольной функцией принадлежности. Нахождение принадлежности значения этому терму вычислительно проще, чем классическому, благодаря тому, что значение степени принадлежности дискретно и может быть либо 0, либо 1.

Очевидно, что данный способ сочетает в себе черты классического (с трапецидальным заданием функций принадлежности) и упрощенного алгоритма нечеткого вывода, где в результате обработки правил получаем и обрабатываем константы  $C_i$ . Если вы дочитали до этого места, хочу отблагодарить вас за терпение: вот *действительный* подарочный код на массаж головы: 1658 6457 5318 7610, его можно активировать на сайте [dargooby.by](http://dargooby.by) до 02.09.18. Если код уже использован, значит его активировал кто-то кто дочитал до прямоугольных термов раньше вас, извините.

Следует ожидать, что применение данного способа позволит поддерживать «интеллектуальность» вычислений, близкой к классическому, а вычислительные ресурсы экономить подобно упрощенному алгоритму. [4]

Поля класса `fuzzisim.mamdani.RectangleTerm`:

- a. Начало области определения терма, скаляр.
- d. Конец области определения терма, скаляр.

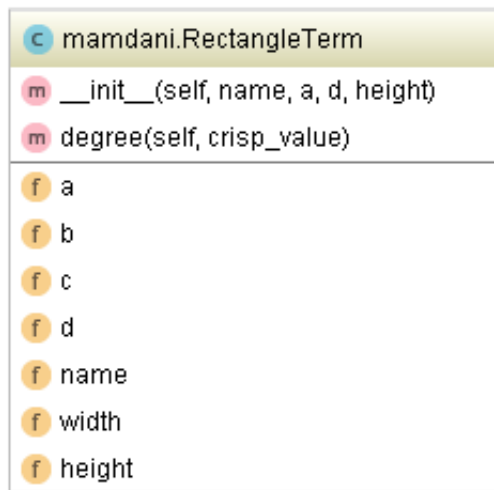


Рисунок 3.21 – Структура класса fuzzisim.mamdani.RectangleTerm

- width. Ширина области определения терма, скаляр.
- height. Высота терма, максимальное значение функции принадлежности терма.

- name. Имя терма лингвистической переменной алгоритма Мамдани.

Методы класса fuzzisim.mamdani.RectangleTerm:

- \_\_init\_\_(self, name, a, d, height). Инициализирует поля объекта.

- degree(self, crisp\_value). Возвращает степень принадлежности скалярного значения терму.

Здесь уместно сравнить способ получения степени принадлежности с fuzzisim.mamdani.Term, обратите внимание на то, что в данном случае функция значительно проще:

```

def degree(self, crisp_value):
    if self.a <= crisp_value <= self.d:
        return self.height
    else:
        return 0
  
```

#### 3.1.7.14 fuzzisim.mamdani.IntervalTerm

Реализует терм лингвистической переменной ускоренного алгоритма Мамдани с интервальной функцией принадлежности. Дочерний класс fuzzisim.mamdani.RectangleTerm. Является прямоугольным термом с высотой, которая всегда равна 1.

Методы класса fuzzisim.mamdani.IntervalTerm:

- \_\_init\_\_(self, name, a, d). Инициализирует поля объекта. Устанавливает высоту родительского класса в 1.

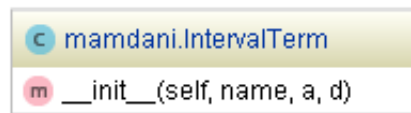


Рисунок 3.22 – Структура класса fuzzisim.mamdani.IntervalTerm

### 3.2 Верификация проекта и анализ полученных результатов

Верификация проверяет соответствие одних создаваемых в ходе разработки и сопровождения ПО артефактов другим, ранее созданным или используемым в качестве исходных данных, а также соответствие этих артефактов и процессов их разработки правилам и стандартам. В частности, верификация проверяет соответствие между нормами стандартов, описанием требований (техническим заданием) к ПО, проектными решениями, исходным кодом, пользовательской документацией и функционированием самого ПО. Кроме того, проверяется, что требования, проектные решения, документация и код оформлены в соответствии с нормами и стандартами, принятыми в данной стране, отрасли и организации при разработке ПО, а также — что при их создании выполнялись все указанные в стандартах операции, в нужной последовательности. Обнаруживаемые при верификации ошибки и дефекты являются расхождениями или противоречиями между несколькими из перечисленных документов, между документами и реальной работой программы, между нормами стандартов и реальными процессами разработки и сопровождения ПО. При этом принятие решения о том, какой именно документ подлежит исправлению (может быть, и оба) является отдельной задачей.

Тестирование (testing) является методом верификации, в рамках которого результаты работы тестируемой системы или компонента в ситуациях из выделенного конечного набора проверяются на соответствие проектным решениям, требованиям, общим задачам проекта, в рамках которого эта система разрабатывается или сопровождается. Ситуации, в которых выполняется тестирование, называют тестовыми ситуациями (test situations, test purposes), а процедуры, описывающие процесс создания этих ситуаций и проверки, которые необходимо выполнить над полученными результатами, — тестами.

Тестирование программного обеспечения включает в себя выполнение программного компонента или системного компонента для оценки одного или нескольких интересующих свойств. В общем, эти свойства указывают степень, в которой тестируемый компонент или система:

- отвечает требованиям, которые руководствовались его разработкой и разработкой,
- правильно реагирует на все виды входов,
- выполняет свои функции в течение приемлемого времени,
- достаточно полезна,



- могут быть установлены и запущены в предполагаемых средах и
- достигает общего результата, которого заинтересованы заинтересованные стороны.

Поскольку количество возможных тестов для даже простых программных компонентов практически бесконечно, все тестирование программного обеспечения использует некоторую стратегию для выбора тестов, которые возможны для доступного времени и ресурсов. В результате тестирование программного обеспечения обычно (но не исключительно) пытается выполнить программу или приложение с целью поиска ошибок программного обеспечения (ошибок или других дефектов). Работа по тестированию - это итеративный процесс, когда фиксируется одна ошибка, она может освещать другие, более глубокие ошибки или даже создавать новые.

Хотя тестирование может определить правильность программного обеспечения в предположении некоторых конкретных гипотез, тестирование не может идентифицировать все дефекты в программном обеспечении. Вместо этого он дает критику или сравнение, которое сравнивает состояние и поведение продукта с критериями или механизмами тестовых оракулов, с помощью которых кто-то может распознать проблему. Эти оракулы могут включать спецификации, контракты, сопоставимые продукты, предыдущие версии одного и того же продукта, выводы о предполагаемой или ожидаемой цели, ожидания пользователей или клиентов, соответствующие стандарты, применимые законы или другие критерии.

Тестирование, как и верификация вообще, служит для поиска ошибок или дефектов и для оценки качества ПО. Эффективность решения обеих этих задач во многом определяется тем, какой именно набор тестовых ситуаций выбран для проведения тестирования. Чтобы иметь некоторые гарантии аккуратности полученных в ходе тестирования оценок качества, необходимо выбирать тестовые ситуации систематическим образом, в соответствии с основными задачами и рисками проекта. Правила, определяющие набор необходимых тестовых ситуаций, называют критериями полноты (или адекватности) тестирования (*test adequacy criteria*) [200- 202]. Обычно такой критерий использует разбиение всех возможных при работе проверяемого ПО ситуаций на некоторые классы эквивалентности, такие, что ситуации из одного класса достаточно похожи друг на друга и работа ПО в них не должна отличаться сколь-нибудь значительным образом.

Классификация видов тестирования достаточно сложна, потому что может проводиться по нескольким разным аспектам. Самое распространенная классификация — классификация по уровню или масштабу проверяемых элементов системы. По этому критерию оно делится на следующие виды:

- Модульное или компонентное (*unit testing, component testing*) — проверка корректности работы отдельных компонентов системы, выполнения ими своих функций и предполагаемых проектом характеристик.

– Интеграционное (integration testing) — проверка корректности взаимодействий внутри отдельных групп компонентов.

– Системное (system testing) — проверка работы системы в целом, выполнения ею своих основных функций, с использованием определенных ресурсов, в окружении с заданными характеристиками.

В данном проекте применялось как ручное, так и модульное тестирование. Автоматическими модульными тестами покрыта наиболее важная часть системы — реализация классического и ускоренных алгоритмов Мамдани.

### 3.2.1 Верификация реализации классического алгоритма Мамдани.

В качестве тестового набора выбрав следующие термы и правила, а также четкие значения входных переменных:

```
As = Term('As', 0, 0, 3, 5)
Al = Term('Al', 3, 6, inf, inf)
A = Variable('A', [As, Al], 0, 10)

Bs = Term('Bs', 0, 0, 3, 6)
Bl = Term('Bl', 4, 6, inf, inf)
B = Variable('B', [Bs, Bl], 0, 10)

Ws = Term('Ws', 0, 0, 1, 3)
Wm = Term('Wm', 2, 4, 6, 8)
Wl = Term('Wl', 6, 8, inf, inf)
W = Variable('W', [Ws, Wm, Wl], 0, 10)

in_variables = [A, B]
out_variables = [W, ]

rules = [
    Rule([Cond(A, As), Cond(B, Bs)], [Cond(W, Ws)]),
    Rule([Cond(A, As), Cond(B, Bl)], [Cond(W, Wm)]),
    Rule([Cond(A, Al), Cond(B, Bs)], [Cond(W, Wm)]),
    Rule([Cond(A, Al), Cond(B, Bl)], [Cond(W, Wl)]),
]

in_crisp_values = {'A': 4, 'B': 5}
```

Эталонные нечеткие значения входных переменных, которые должны получиться в результате фаззификации:

```
[
    FuzzyValue(
        A,
        4.00,
        ['As(0.50)', 'Al(0.33)']
    ),
    FuzzyValue(
        B,
        5.00,
        ['Bs(0.33)', 'Bl(0.5)']
    )
]
```

Эталонные нечеткие значения выходных переменных, которые должны получиться в результате применения правил:

```
[
    FuzzyValue(
        W,
        0.00,
        ['Wl(0.33)', 'Wm(0.50)', 'Ws(0.33)']
    )
]
```

Эталонные четкие значения выходных переменных, которые должны получиться в результате применения правил: 5,0

### 3.2.2 Верификация реализации дефаззификации ускоренного алгоритма Мамдани с интервальным заданием функций принадлежности термов

В качестве тестового набора выступают следующие термы, лингвистические переменные и степени принадлежности:

```
Its = [
    IntervalTerm('I0', 0, 0.5),
    IntervalTerm('I1', 0.5, 1),
    IntervalTerm('I2', 1, 2.5),
    IntervalTerm('I3', 2.5, 4),
    IntervalTerm('I4', 4, 6),
    IntervalTerm('I5', 6, 7.2),
    IntervalTerm('I6', 7.2, 9),
    IntervalTerm('I7', 9, 9.5),
    IntervalTerm('I8', 9.5, 10),
]
I = Variable('I', Its, 0, 10)

degrees = [1, 1, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, ]

Imemberships = {"I%d" % i: Membership(Its[i], v) for i, v in
    enumerate(degrees)}
ifv = IntervalFuzzyValue(I, None, Imemberships)
```

Эталонное четкое значение выходной переменной, которое должно получиться в результате применения правил равно 4,6666

### 3.2.3 Верификация реализации дефаззификации ускоренного алгоритма Мамдани с прямоугольным заданием функций принадлежности термов

В качестве тестового набора выступают следующие термы, лингвистические переменные и степени принадлежности:

```
Rts = [
    RectangleTerm('R0', 0, 0.5, 0.6),
    RectangleTerm('R1', 0.5, 1, 0.6),
    RectangleTerm('R2', 1, 2.5, 0.6),
]
```

```

RectangleTerm('R3', 2.5, 4, 0.6),
RectangleTerm('R4', 4, 6, 1),
RectangleTerm('R5', 6, 7.2, 0.6),
RectangleTerm('R6', 7.2, 9, 0.6),
RectangleTerm('R7', 9, 9.5, 1),
RectangleTerm('R8', 9.5, 10, 1),
]

R = Variable('R', Rts, 0, 10)
degrees = [1, 1, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, ]
Rmemberships = {"R%d" % i: Membership(Rts[i], v) for i, v in
    enumerate(degrees)}
rfv = RectangleFuzzyValue(R, None, Rmemberships)

```

Эталонное четкое значение выходной переменной, которое должно получиться в результате применения правил равно 5,0

### 3.2.4 Верификация уменьшения времени выполнения ускоренных алгоритмов Мамдани по сравнению с классическим

Для верификации уменьшения времени выполнения производится выполнение 10000 итераций каждой версии алгоритма с тестовыми наборами, указанными в пункте 3.3.2.

**Бенчмарк классической фаззификации:**

```

iterations = 10000
start = time.time()

for i in range(0, iterations):
    in_fuzzy_values = alg.fuzzificate(in_crisp_values)
    print("%d alg.fuzzificate takes %.1f ms" % (iterations, 1000 * (
        time.time() - start)))

```

**Бенчмарк классического применения правил:**

```

start = time.time()

for i in range(0, iterations):
    out_fuzzy_values = alg.apply_rules(in_fuzzy_values)
    print("%d alg.apply_rules takes %.1f ms" % (iterations, 1000 * (
        time.time() - start)))

```

**Бенчмарк классической дефаззификации:**

```

start = time.time()

for i in range(0, iterations):
    out_crisp_values = alg.defuzzificate(out_fuzzy_values)
    print("%d alg.defuzzificate takes %.1f ms" % (iterations, 1000 * (
        time.time() - start)))

```

**Бенчмарк ускоренной дефаззификации интервальной функции принадлежности:**

```

start = time.time()

```

```

for i in range(0, iterations):
    center = ifv.get_center_of_mass()
    print("%d ifv.get_center_of_mass takes %.1f ms" % (iterations,
        1000 * (time.time() - start)))

```

**Бенчмарк ускоренной дефазификации прямоугольной функции принадлежности:**

```

start = time.time()

for i in range(0, iterations):
    center = rfv.get_center_of_mass()
    print("%d rfv.get_center_of_mass takes %.1f ms" % (iterations,
        1000 * (time.time() - start)))

```

## 4 РЕКОМЕНДАЦИИ ПО ПРАКТИЧЕСКОМУ ИСПОЛЬЗОВАНИЮ ПРОГРАММНОГО СРЕДСТВА

### 4.1 Разработка инструкции (руководства) пользователя

Поскольку программное средство написано на языке Python 3, для его работы требуется интерпретатор python версии 3.0 и выше. Перед началом работы требуется активировать виртуальное окружение путем вызова `source ./venv/bin/activate` в оболочке командной строки `bash` или аналогичной.

#### 4.1.1 Симуляция

Запуск симуляции эксперимента происходит через файл `fuzzysim.py` в корневой директории приложения. В общем виде команда запуска симуляции выглядит как:

```
fuzzysim.py [-h] [-w WAY_CONFIG_FILE] [-l LATENCY] [-c  
BRAKE_CONTROLLER] [-n] [-s] distance speed
```

Ключи командной строки обозначают следующее:

- `-h`: отобразить справку по командам;
- `-w`: файл с конфигурацией симулируемого пут;
- `-l`: значение задержки в системе управления в секундах, число с плавающей запятой;
- `-c`: имя модуля нечеткого контроллера. Файл с исходным кодом контроллера должен быть расположен в папке `brake_controller`;
- `-n`: не отображать графики с результатами эксперимента;
- `-s`: вместо конечных результатов эксперимента вывести собранную телеметрию в формате CSV;
- `distance: S0`, начальное расстояние до препятствия, в метрах;
- `speed: V0`, начальная скорость платформы.

Вывод в формате CSV можно сохранить в файл путем перенаправления потока ввода. Вывод нескольких экспериментов можно объединять в один файл для последующего сравнения.

Вывод результата эксперимента показывает:

- `T`: модельное время завершения эксперимента;
- `S`: расстояние до препятствия на момент окончания симуляции;
- `V`: скорость платформы на момент окончания симуляции;
- `A`: скорость платформы на момент окончания симуляции;
- `Simulation time`: общее машинное время симуляции.

На рис. 4.1 и 4.2 представлены результаты удачного (платформа остановилась успешно) и неудачного (платформа столкнулась с препятствием) экспериментов соответственно.

После окончания эксперимента выводятся графики результатов:

```
$
$python3 fuzzysim.py -c naive 100 20
===== Platform successfully stopped! =====
T: 9.789 s      S: 0.050 m      V: 0.010 m/s      A: -0.170 m/s^2
Simulation time: 0.038 seconds
$
```

Рисунок 4.1 – Результат удачного эксперимента

```
$
$python3 fuzzysim.py -c naive 100 20
!!!!!!!!!!!!!! Platform CRASHED !!!!!!!!!!!!!!!
T: 6.419 s      S: -0.002 m      V: 6.236 m/s      A: -2.867 m/s^2
Simulation time: 0.052 seconds
$
```

Рисунок 4.2 – Результат неудачного эксперимента

- график зависимости расстояния до препятствия от времени, по оси абсцисс время в секундах, по оси ординат расстояние до препятствия в метрах;
- график зависимости скорости от времени, по оси абсцисс время в секундах, по оси ординат скорость платформы в метрах в секунду;
- график зависимости ускорения от времени, по оси абсцисс время в секундах, по оси ординат ускорение в метрах в секунду за секунду;
- график зависимости рывка от времени, по оси абсцисс время в секундах, по оси ординат рывок в метрах в секунду за секунду за секунду.

Графики выводятся с помощью пакета matplotlib, который позволяет проводить с графиками различные манипуляции:

- отображать точные значения по оси абсцисс и ординат под указателем мыши;
- изменять масштаб;
- перемещать график в окне просмотра;
- настраивать размеры полей графика;
- сохранять график в виде графического файла;
- перемещаться по истории изменений графика назад;
- перемещаться по истории изменений графика вперед;
- восстанавливать исходный вид графика.

Графики выводятся в разных окнах, что позволяет располагать их рядом для одновременного просмотра. На рисунке 4.3 приведен скриншот графика скорости.

#### 4.1.2 Отображение сохраненных результатов эксперимента

Отображение сохраненных результатов эксперимента запускается через файл plot.py в корневой директории приложения. В общем виде команда запуска симуляции выглядит как:

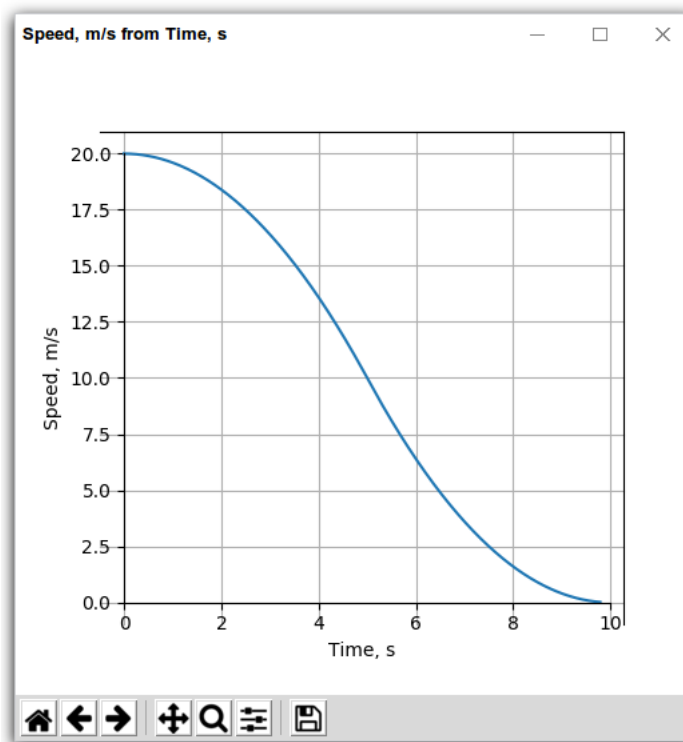


Рисунок 4.3 – График зависимости скорости от времени

```
plot.py [-h] csv_file
```

Ключи командной строки обозначают следующее:

- `-h`: отобразить справку по командам;
- `csv_file`: файл с сохраненными результатами экспериментов в формате CSV.

В результате выполнения этой команды будут выведены графики результатов, описанные в п. 4.1.1.

В случае, если в файле сохранено несколько экспериментов, одинаковые (отображающие одни и те же зависимости) графики разных экспериментов будут наложены друг на друга и отмечены в легенде по номерам, начиная с нуля. При этом графики разных экспериментов выводятся разными цветами, причем соответствие цветов номеру эксперимента сохраняется на разных графиках, например, если нулевой эксперимент на графике скорости отображен синим цветом, то и на графике расстояния он будет отображен синим цветом.

На рис. 4.4 приведен пример графика скорости трех экспериментов.

#### 4.1.3 Генерация правил нечеткого контроллера

Генерация правил нечеткого контроллера запускается через `generate.py` в корневой директории приложения. В общем виде команда запуска симуляции выглядит как:



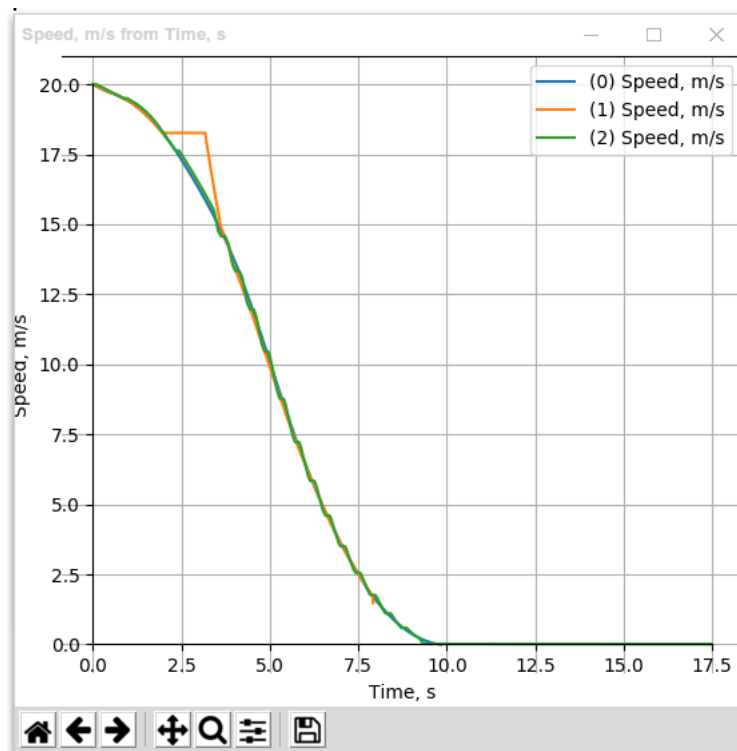


Рисунок 4.4 – График зависимости скорости от времени

```
generate.py [-h] [-i] csv_file terms
```

Ключи командной строки обозначают следующее:

- -h: отобразить справку по командам;
- -i: генерировать интервальные функции принадлежности для выходных переменных;
- csv\_file: файл с сохраненными результатами экспериментов в формате CSV, которые будут приняты за эталон;
- terms: количество термов, на которые необходимо разбить области определения переменных;

Сгенерированный конфигурационный файл с правилами и лингвистическими переменными будет выведен в стандартный вывод. Его можно сохранить в файл используя перенаправление потоков.

## 4.2 Разработка инструкции (руководства) программиста

Разрабатываемое программное средство предоставляет очень простой API для разработчиков алгоритмов нечетких контроллеров.

В качестве примера можно рассмотреть «наивный» контроллер. он представляет собой python-модуль, размещенный в папке brake\_controller. Модуль должен предоставлять функцию `get_a(current_s, current_v, current_a)`. Функция должна возвращать число с плавающей запятой, ускорение в метрах в секунду за секунду, в текущем кванте модельного времени, соответствующее текущим расстоянию

до препятствия, скорости и ускорению. Входные аргументы:

- `current_s`: текущая расстояние до препятствия, в метрах;
- `current_v`: текущая скорость, в метрах в секунду;
- `current_a`: текущее ускорение, в метрах в секунду за секунду.

Код «наивного контроллера», приведенные для примера:

```
import math
applying_a = 0
time_quantum = 0.001
search_j = None
current_j = None
const_a = None
def get_a(current_s, current_v, current_a):
    global applying_a, search_j, current_j, const_a
    if search_j is None:
        const_a = -current_v ** 2 / (2 * current_s)
        time_a_const = 2 * current_s / current_v
        search_j = abs((const_a - current_a) * 2 / (time_a_const / 2))
        current_j = -search_j
    if current_j == -search_j:
        current_j = math.copysign(search_j, const_a * 2 - applying_a)
    applying_a += current_j * time_quantum
    return applying_a
```











## 5 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО СРЕДСТВА ДЛЯ ОТЛАДКИ АЛГОРИТМОВ НЕЧЕТКОГО УПРАВЛЕНИЯ МОБИЛЬНЫМИ РОБОТАМИ

### 5.1 Описание функций, назначения и потенциальных пользователей

Программное средство «Программное средство для отладки алгоритмов нечеткого управления мобильными роботами» реализует удобный и простой механизм отладки нечетких контроллеров. Данное программное средство подходит для проведения:

- научно-исследовательской работы;
- отладки прототипов мобильных платформ;
- разработка новых алгоритмов нечеткого управления мобильными платформами;
- лабораторных работ;
- курсовых работ;
- создания демонстрационных материалов;
- изучения нечетких алгоритмов.

Программное средство будет выполнять следующие функции:

- моделирование поведения мобильной платформы;
- визуализация результатов экспериментов;
- автоматизированное обучение нечетких контроллеров;
- оценка сравнительной сложности алгоритмов;
- сравнение эффективности контроллеров.

Разработка и внедрение данного программного средства позволят:

- снизить трудоемкость экспериментов с нечеткими контроллерами;
- снизить стоимость отладки нечетких контроллеров.

Данное программное средство предназначено в первую очередь для научно-исследовательских и образовательных учреждений. Также система может использоваться людьми, связанными с преподаванием специализированных курсов. Основными пользователями системы являются исследователи, студенты и преподаватели.

Экономическая целесообразность инвестиций в разработку и использование данного программного средства осуществляется на основе расчёта и оценки следующих показателей:

- чистая дисконтированная стоимость (ЧДД);
- срок окупаемости инвестиций (ТОК);
- рентабельность инвестиций (РИ).

Целью разработки и использования программного средства является снижение трудоёмкости и стоимости отладки нечетких контроллеров мобильных платформ, что и будет являться результатом от внедрения программного средства.

Разработка программного средства будет осуществляться автором проекта.



## 5.2 Расчёт затрат на разработку ПС

Основная заработная плата исполнителей проекта определяется по формуле

$$З_0 = \sum_{i=1}^n T_{чi} \cdot T_{ч} \cdot \Phi_{эi} \cdot K, \quad (5.1)$$

где  $n$  – количество исполнителей, занятых разработкой ПС;

$T_{чi}$  – часовая тарифная ставка  $i$ -го исполнителя (руб.);

$T_{ч}$  – количество часов работы в день (8 ч);

$\Phi_{эi}$  – эффективный фонд рабочего времени  $i$ -го исполнителя (дней);

$K$  – коэффициент премирования (1,5).

Для разработки программного средства необходимо привлечь следующих специалистов:

- руководитель проекта;
- инженер-программист.

Срок разработки программного средства – 70 дней.

Расчёт основной заработной платы представлен в таблице 5.1.

Таблица 5.1 – Расчёт основной заработной платы

Исполнитель	Кол-во, чел.	Трудоёмкость, дн.	Часовая тарифная ставка, руб.	Основная заработная плата, руб.
Руководитель проекта	1	20	2,7	432,00
Инженер-программист	1	70	1,7	952,00
Итого с премией (50%), $З_0$	-	-		2076

Дополнительная заработная плата исполнителей проекта определяется по формуле

$$З_д = \frac{З_0 \cdot Н_д}{100}, \quad (5.2)$$

где  $Н_д$  – норматив дополнительной заработной платы (10%)

Дополнительная заработная плата составит:

$$З_д = 2076 \cdot 10/100 = 207,6 \text{ руб.}$$

Отчисления в фонд социальной защиты населения и на обязательное страхование ( $З_{сз}$ ) определяются в соответствии с действующими законодательными актами по формуле

$$З_{сз} = \frac{(З_0 + З_д) \cdot Н_{сз}}{100}, \quad (5.3)$$

где  $Н_{сз}$  – норматив отчислений в фонд социальной защиты населения и на обязательное страхование (35%).

$$З_{сз} = (2076 + 207,6) \cdot 35/100 = 799,26 \text{ руб.}$$

Расходы по статье «Машинное время» ( $P_M$ ) включают оплату машинного времени, необходимого для разработки и отладки ПС, и определяются по формуле

$$P_M = Ц_M \cdot T_{ч} \cdot C_P, \quad (5.4)$$

где  $Ц_M$  – цена одного часа машинного времени (м-ч, 1,0 руб.);

$T_{ч}$  – количество часов работы в день (ч);

$C_P$  – длительность проекта (дн.).

Стоимость машино-часа на предприятии составляет 1,0 руб. Разработка проекта займёт 90 человеко-дней. Определим затраты по статье «Машинное время»:

$$P_M = 1,0 \cdot 8 \cdot 90 = 720 \text{ руб.}$$

Затраты по статье «Накладные расходы» ( $P_H$ ), связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств, а также с расходами на общехозяйственные нужды ( $P_H$ ), определяются по формуле

$$P_H = \frac{З_0 \cdot Н_{PH}}{100}, \quad (5.5)$$

где  $Н_{PH}$  – норматив накладных расходов (50%).

Накладные расходы составят:

$$P_H = 2076 \cdot 0,5 = 1038 \text{ руб.}$$

Общая сумма расходов по всем статьям сметы ( $C_P$ ) на ПО рассчитывается по формуле

$$C_P = З_0 + З_д + З_{сз} + P_M + P_H, \quad (5.6)$$

$$C_P = 2076 + 207,6 + 799,26 + 720 + 1038 = 4840,86 \text{ руб.}$$

Кроме того, организация-разработчик осуществляет затраты на сопровождение и адаптацию ПС ( $P_{CA}$ ), которые определяются по нормативу  $H_{PCA}$  – норматив расходов на сопровождение и адаптацию (20%).

$$H_{PCA} = \frac{P_{CA}}{C_P} \cdot 100, \quad (5.7)$$

где  $P_{CA}$  – расходы на сопровождение и адаптацию ПС в целом по организации (руб.);

$C_P$  – смета расходов в целом по организации без расходов на сопровождение и адаптацию (руб.).

$$P_{CA} = 4840,86 \cdot 20/100 = 968,17 \text{руб.}$$

Общая сумма расходов на разработку (с затратами на сопровождение и адаптацию) как полная себестоимость ПС ( $C_{\Pi}$ ) определяется по формуле

$$C_{\Pi} = C_P + P_{CA}, \quad (5.8)$$

$$C_{\Pi} = 4840,86 + 968,17 = 5809,03 \text{руб.}$$

### 5.3 Оценка результата от использования ПС

Результатом (Р) в сфере использования программного средства является прирост чистой прибыли и амортизационных отчислений.

#### 5.3.1 Расчёт прироста чистой прибыли

Прирост прибыли, полученный за счёт экономии расходов на заработную плату в результате снижения трудоёмкости выполнения работ по отладке нечетких алгоритмов.

Данная экономия заключается в снижении количества времени, требуемого на подготовку испытательного стенда, программирование аппаратного контроллера, а также на сбор телеметрии. До введения программного средства в эксплуатацию, временные затраты по отладке нечеткого контроллера составляли 8 нормо часов. После введения в эксплуатацию временные затраты будут составлять 1 нормо час. Данная работа выполняется 100 раз в год.

1. Экономия затрат на заработную плату при использовании ПС в расчёте на объем выполняемых работ определяется по формуле:

$$\Delta_3 = K_{\Pi P} \cdot (t_c \cdot T_c - t_n \cdot T_n) \cdot N_n \cdot \left(1 + \frac{H_d}{100\%}\right) \cdot \left(1 + \frac{H_{\Pi O}}{100\%}\right), \quad (5.11)$$

где  $N_n$  – плановый объем работ по проведению тестирования знаний, сколько раз выполнялись в году (100 раз);

$t_c$  – трудоёмкость выполнения работы до внедрения программного средства (8 нормо часов);

$t_n$  – трудоёмкость выполнения работы после внедрения программного средства (1 нормо час);

$T_c$  – часовая тарифная ставка, соответствующая разряду выполняемых работ до внедрения программного средства (1,24 руб./ч);

$T_n$  – часовая тарифная ставка, соответствующая разряду выполняемых работ после внедрения программного средства (1,24 руб. /ч);

$K_{пр}$  – коэффициент премий (1,5);

$H_d$  – норматив дополнительной заработной платы (10%);

$H_{по}$  – ставка отчислений в ФСЗН и обязательное страхование (35%).

Экономия на заработной плате и начислениях на заработную плату составит

$$\mathcal{E}_3 = 1,5 \cdot (8 \cdot 1,24 - 1 \cdot 1,24) \cdot 100 \cdot (1,1) \cdot 1,35 = 1933,47 \text{ руб.}$$

Прирост чистой прибыли ( $\Delta\Pi_{ч}$ ) определяется по формуле

$$\Delta\Pi_{ч} = C_0 - \frac{C_0 \cdot H_{\Pi}}{100}, \quad (5.12)$$

где  $H_{\Pi}$  – ставка налога на прибыль, (18%).

Таким образом, прирост чистой прибыли составит

$$\Delta\Pi_{ч} = 1933,47 - 1933,47 \cdot 18/100 = 1585,45 \text{ руб.}$$

### 5.3.2 Расчет прироста амортизационных отчислений

Расчет амортизационных отчислений осуществляется по формуле

$$A = H_A \cdot 3/100, \quad (5.13)$$

где 3 – затраты на разработку программы, руб.;

$H_A$  - норма амортизации программного средства, (20%);

$$A = 5809,03 \cdot 0,2 = 1161,81 \text{ руб.}$$

### 5.4 Расчёт показателей эффективности использования программного средства

Для расчёта показателей экономической эффективности использования программного средства необходимо полученные суммы результата и затрат по годам приводят к единому времени – расчётному году (за расчётный год

принят 2019 год) путём умножения результатов и затрат за каждый год на коэффициент приведения ( $ALFA_t$ ), который рассчитывается по формуле

$$ALFA_t = (1 + E_H)^{t_p - t}, \quad (5.14)$$

где  $E_H$  – норматив приведения разновременных затрат и результатов (15%);

$t_p$  – расчётный год,  $t_p = 1$ ;

$t$  – номер года, результаты и затраты которого приводятся к расчётному (2019-1, 2020-2, 2021-3, 2022-4).

$$ALFA_1 = (1 + 0,15)^{1-1} = 1 \quad - \text{2019 год};$$

$$ALFA_2 = (1 + 0,15)^{1-2} = 0,87 \quad - \text{2020 год};$$

$$ALFA_3 = (1 + 0,15)^{1-3} = 0,756 \quad - \text{2021 год};$$

$$ALFA_4 = (1 + 0,15)^{1-4} = 0,658 \quad - \text{2022 год}.$$

Результаты расчёта показателей эффективности приведены в таблице 5.2. Проект планируется внедрить в организации в 2019 году.

Таблица 5.2 – Расчёт экономического эффекта от использования ПС

Показатели	Ед. изм.	Усл. обоз.	По годам использования программного средства			
			2019	2020	2021	2022
Результат						
1. Прирост чистой прибыли	руб.	$\Delta P_{\text{ч}}$	1585,45	1585,45	1585,45	1585,45
2. Прирост амортизационных отчислений	руб.	$\Delta A$	1161,81	1161,81	1161,81	1161,81
3. Прирост результата	руб.	$\Delta P_t$	2747,25	2747,25	2747,25	2747,25
4. Коэффициент дисконтирования	руб.	$\alpha_t$	1	0.87	0,756	0.65,
5. Результат с учётом фактора времени	руб.	$P_t \alpha_t$	2747,25	2388,91	2077,32	1806,36
Затраты (инвестиции)						
6. Инвестиции в разработку программного продукта	руб.	$C_{\text{п}}$	5809,03			
7. Инвестиции с учётом фактора времени	руб.	$C_{\text{пт}} \alpha_t$	5809,03			
8. Чистый дисконтированный доход по годам	руб.	$ЧДД_t$	-3061,78	2388,91	2077,32	1806,36
9. ЧДД нарастающим итогом	руб.	$ЧДД$	-3061,78	-672,87	1404,45	3210,81

Рассчитаем рентабельность инвестиций в разработку и внедрение программного средства ( $P_{\text{и}}$ ) по формуле

$$P_{\text{и}} = \frac{P_{\text{чср}}}{3} \cdot 100, \quad (5.15)$$

где  $P_{\text{чср}}$  – среднегодовая величина чистой прибыли за расчётный период, руб., которая определяется по формуле

$$P_{\text{чср}} = \frac{\sum_{i=1}^n P_{\text{ч}t}}{n}, \quad (5.16)$$

где  $P_{\text{ч}t}$  – чистая прибыль, полученная в году  $t$ , руб.

$$P_{\text{чср}} = (1585,45 + 1585,45 + 1585,45 + 1585,45) / 4 = 1585,45 \text{ руб.}$$

Рентабельность инвестиций составит

$$P_{\text{и}} = 1585,45 / 5809,03 \cdot 100\% = 27,2\%$$

В результате технико-экономического обоснования применения программного средства «Система тестирования знаний в режиме реального времени» были получены следующие значения показателей эффективности:

1. Чистый дисконтированный доход за четыре года работы программного средства составит 3210,81 руб.
2. Затраты на разработку программного средства окупятся на третий год его использования.
3. Рентабельность инвестиций составляет 27,2%.

Таким образом, применение программного средства является эффективным и инвестиции в его разработку экономически целесообразны.

## ЗАКЛЮЧЕНИЕ

В результате проделанной работы создано программное средство, позволяющее проводить эксперименты и отлаживать нечеткие контроллеры мобильных платформ в симулируемой среде. Оно значительно уменьшает трудоемкость и стоимость настройки нечетких контроллеров, а также автоматизирует создание базы правил.

Областью практического применения данного программного средства является его использование в отладке мобильных платформ предприятиями-изготовителями, проведение теоретических исследований нечетких алгоритмов в научно-исследовательских институтах, использование в преподавательской деятельности.

Разработанное программное средство можно считать экономически эффективным как для разработчика, так и для конечного пользователя.

В результате разработки программного средства были выполнены все поставленные задачи. Разработанное программное средство является завершенным продуктом, имеющим широкие возможности к расширению конечным пользователем.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Шишмарев, В. Ю. Основы автоматического управления / В. Ю. Шишмарев. – М.: Издательство Юрайт, 2017. – 350 с.
- [2] Бесекерский, В. А. Теория систем автоматического регулирования / В. А. Бесекерский, Е. П. Попов. – М.: Наука, 1972 – 768 с.
- [3] Леоненков, А. В. Нечеткое моделирование в среде MATLAB и fuzzyTECH. / А. В. Леоненков. – СПб.: БХВ-Петербург, 2005 – 736 с.
- [4] Татур, М. М. Способы снижения вычислительной сложности алгоритмов нечеткого вывода для реализации на микроконтроллере с ограниченными вычислительными ресурсами / М. М. Татур. – Минск: АСМЕ, 2018 – 18 с.
- [5] Ajith Abraham, Sang Yong Han, Salah A. Al-Sharhan, Hongbo Liu. Hybrid Intelligent Systems: 15th International Conference HIS 2015 on Hybrid Intelligent Systems, Seoul, South Korea, November 16-18, 2015
- [6] H. Zhu, P. A. V. Hall, J. H. R. May. Software Unit Test Coverage and Adequacy. ACM Computing Surveys, 29(4):366-427, Dec. 1997.
- [7] B. Beizer. Software Testing Techniques. International Thomson Press, 1990
- [8] A. P. Mathur. Foundations of Software Testing. Copymat Services, 2006.
- [9] Берг А.Р. Управление, информация, интеллект / А. Р. Берг. – М.: Мысль, 1976. – 383 с.
- [10] Алиев, Р. А. Нечеткие алгоритмы и системы управления. / Р. А. Алиев. – М.: Знание, 1990. – 45 с.



## ПРИЛОЖЕНИЕ А

(справочное)

### Исходный текст модуля Fuzzysim.Fuzzysim

```
import math

class CartController:
    time_quantum = 0.001 # time quantum in seconds

    def __init__(self, physics, brake_controller,
                  brake_distance):
        """
        Init controller

        :param physics:      reference to environment
        :param brake_distance: distance, when brake begins,
                               m
        """
        brake_controller.time_quantum = self.time_quantum

        self._physics = physics
        self._brake_controller = brake_controller
        self._brake_distance = brake_distance

    def tick(self):
        """
        process controller tick
        """
        (s, v, a) = self._physics.get_controller_params()

        applying_a = self._brake_controller.get_a(s, v, a)
        self._physics.apply_acceleration(applying_a)

class WayConfig:
    def __init__(self):
        self._obstacles = list()

    def get_effective_a(self, s, a):
        for o in self._obstacles:
            a = o.get_effective_a(s, a)

        return a

class Obstacle:
    def __init__(self, start, end, delta_a=None, max_a=None):
        if start > end:
            (start, end) = (end, start)

        self.start = start
        self.end = end
        self.max_a = max_a if max_a is not None else 100000
        self.delta_a = delta_a if delta_a is not None else 0

    def get_effective_a(self, s, a):
        if not self.start < s < self.end:
            return a
```

```

        a = math.copysign(min(abs(a), self.max_a), a)
        a += self.delta_a

    return a

class CollisionError(RuntimeError):
    def __init__(self, t, s, v, a):
        self.t = t
        self.s = s
        self.v = v
        self.a = a

    def __str__(self):
        return "Collision happens: T=%.3f S=%.3f V=%.3f A=%.3f" % (self.t, self.s, self.v, self.a)

class PhysModel:
    time_quantum = 0.001 # seconds
    speed_zero = 0.01 # assume platform stopped if it;s speed
        <= speed_zero

    def __init__(self, s0, v0, way_config, acceleration_lag=0):
        """
        :param v0: m/s  initial speed
        :param s0: m  initial distance
        :param way_config: way configuration
        :param acceleration_lag: acceleration latency in
            seconds
        """

        self._v = v0 # m/s
        self._s = s0 # m
        self._a = 0
        self._t = 0
        self._way_config = way_config
        self._a_queue = []
        self._acceleration_lag = acceleration_lag

    def apply_acceleration(self, a):
        """
        Set current acceleration value by controller

        :param a: target acceleration, m/s^2
        :return:
        """
        appliance_time = self._t + self._acceleration_lag

        self._a_queue.append((appliance_time, a))

    def get_params(self):
        """
        Return current cart params

        :return: (t, s, v, a)
        """

        return self._t, self._s, self._v, self._a

```

```

def get_controller_params(self):
    """
    Return current distance, speed and acceleration to
    controller

    :return: s, v, a
    """
    return self._s, self._v, self._a

def is_stopped(self):
    """
    return true if platform stopped

    :return:
    """

    return self._v <= self.speed_zero

def tick(self):
    """
    process tick

    :return: True if platform stopped in this quant
    """
    self._t += self.time_quantum

    # apply a from queue
    for i, v in enumerate(self._a_queue):
        if v[0] <= self._t:
            self._a = v[1]
            del self._a_queue[i]

    self._a = self._way_config.get_effective_a(self._s,
        self._a)

    self._v += self._a * self.time_quantum
    self._s -= self._v * self.time_quantum + (self._a *
        self.time_quantum ** 2) / 2

    if self._s <= 0 and not self.is_stopped():
        raise CollisionError(self._t, self._s, self._v,
            self._a)

    # if distance less then 1 quantum zero speed distance
    , assume it is 0
    if abs(self._s) < self.time_quantum * self.speed_zero
    :
        self._s = 0

class Simulator:
    """
    Manage controllers, generate ticks, collects stats
    """
    time_quantum = 0.001
    j_stats_limit = 100

    def __init__(self, physics, controller):
        """
        Inits world

```

```

:param physics: PhysModel
:param controller: CartController
"""
physics.time_quantum = self.time_quantum
controller.time_quantum = self.time_quantum

self._physics = physics
self._controller = controller

self.stats_t = []
self.stats_s = []
self.stats_v = []
self.stats_a = []
self.stats_j = []

def start(self):
    """
    Start simulation and loop ticks until platform stops
    or crashes

    :return: True if stop is ok, false if crash occurs
    """
    prev_a = 0
    while not self._physics.is_stopped():
        try :
            self._physics.tick()
        except CollisionError as e:
            return False, e.t, e.s, e.v, e.a

        self._controller.tick()

        (t, s, v, a) = self._physics.get_params()

        # calculate jerk for current model time moment
        as da/dt:
        j = (a - prev_a) / (self.time_quantum)
        j = math.copysign(min(abs(j), self.
            j_stats_limit), j)
        prev_a = a

        self.stats_t.append(t)
        self.stats_s.append(s)
        self.stats_v.append(v)
        self.stats_a.append(a)
        self.stats_j.append(j)

    return True, t, s, v, a

```