Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика, искусственный интеллект и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

# ОТЧЕТ

по лабораторной работе № 2

по курсу «Основы искусственного интеллекта»

на тему: «Алгоритмы нечеткой логики»

Студент ИУ7-13М (Группа) _____ (Подпись, дата) Орду М. А. (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Строганов Ю. В. (И. О. Фамилия)

*2025 г.*

# СОДЕРЖАНИЕ

# 1 Теоретическая часть

## 1.1 Постановка задачи

В одномерном пространстве ($X = 1$) рассматриваются два автомобиля: лидер, управляемый пользователем, и автомобиль-автопилот. Автопилот должен следовать за лидером, поддерживая постоянную дистанцию $D$, не имея информации о скорости лидера. Известно только текущее расстояние между автомобилями. Требуется определить необходимую скорость автопилота $v_{auto}$ на основе нечеткого логического вывода.

Определение ускорения запрещено. Входными переменными являются:

- ошибка по расстоянию $e = D - D$;
- изменение ошибки $\Delta e = \frac{de}{dt}$.

Выходная переменная — скорость автопилота $v_{auto}$.

## 1.2 Этапы нечеткого логического вывода

Нечеткий логический вывод состоит из следующих этапов:

1. **Фаззификация** — преобразование четких входных значений $e$ и $\Delta e$ в степени принадлежности нечетким подмножествам.
2. **Применение базы правил** — вычисление степени активации каждого правила на основе входных значений.
3. **Импликация** — формирование выходных нечетких множеств в соответствии с вычисленной степенью активации $\alpha$.
4. **Агрегация** — объединение всех выходных множеств.
5. **Дефаззификация** — преобразование агрегированного нечеткого множества в четкое значение $v_{auto}$.

## 1.3 Функции принадлежности

Для входных и выходных переменных были выбраны следующие функции принадлежности:

- Ошибка расстояния `Negative`: `Zero`, `Positive`;
- Изменение ошибки `Negative`: `Zero`, `Positive`;
- Скорость `Slow`: `Medium`, `Fast`.

Для задания функций использовались треугольные формы:

$$\texttt{trimf}(x; a, b, c) = \begin{cases} 0, & x < a, \\ \frac{x-a}{b-a}, & a \leq x < b, \\ \frac{c-x}{c-b}, & b \leq x < c, \\ 0, & x \geq c. \end{cases}$$

## 1.4 База правил

Правила нечеткого вывода описывают зависимость между ошибками и требуемой скоростью:

1. ЕСЛИ distance_error = Positive И delta_distance = Negative ТО v_follower = Slow;
2. ЕСЛИ distance_error = Positive И delta_distance = Zero ТО v_follower = Slow;
3. ЕСЛИ distance_error = Positive И delta_distance = Positive ТО v_follower = Medium;
4. ЕСЛИ distance_error = Zero И delta_distance = Negative ТО v_follower = Medium;
5. ЕСЛИ distance_error = Zero И delta_distance = Zero ТО v_follower = Medium;
6. ЕСЛИ distance_error = Zero И delta_distance = Positive ТО v_follower = Fast;
7. ЕСЛИ distance_error = Negative И delta_distance = Negative ТО v_follower = Medium;

8. ЕСЛИ distance_error = Negative И delta_distance = Zero ТО v_follower = Fast;

9. ЕСЛИ distance_error = Negative И delta_distance = Positive ТО v_follower = Fast.

## 1.5 Алгоритм логического вывода

Для варианта лабораторной работы используется алгоритм Ларсена:

$$\mu_{A\wedge B}(z) = \mu_A(x) \cdot \mu_B(y),$$

где:
$\mu_A(x)$ — функция принадлежности входной переменной $x$ к множеству $A$;
$\mu_B(y)$ — функция принадлежности входной переменной $y$ к множеству $B$.

## 1.6 Алгоритм дефаззификации

Для получения четкого значения скорости используется метод центра тяжести (Centroid method):

$$v = \frac{\int z \cdot \mu(z)\, dz}{\int \mu(z)\, dz}.$$

В качестве альтернативного метода можно применять метод среднего максимума (Mean of maxima, MOM):

$$v = \frac{z_{\min} + z_{\max}}{2}, \quad z_{\min,\max} \in \{z | \mu(z) = \max(\mu)\}.$$

# 2 Практическая часть

## 2.1 Используемые инструменты

Для реализации нечеткой системы использована библиотека `scikit-fuzzy` и язык Python 3. Основные зависимости:

- `numpy` — численные вычисления;
- `matplotlib` — визуализация;
- `scikit-fuzzy` — функции принадлежности и дефаззификация.

## 2.2 Функции принадлежности

На рисунках 2.1-2.3 представлены функции принадлежности нечетких переменных.

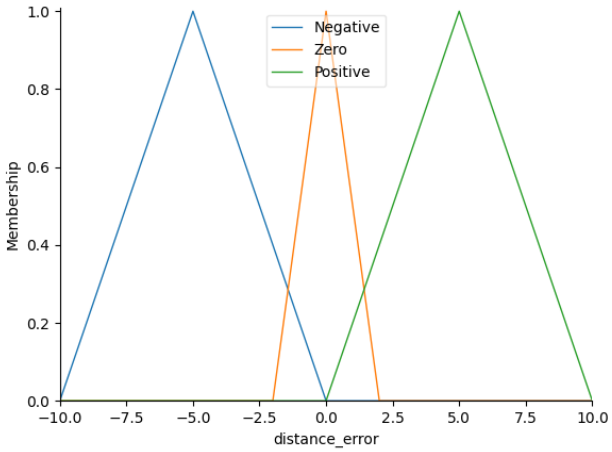


Рисунок 2.1 – Функция принадлежности расстояния между автомобилями

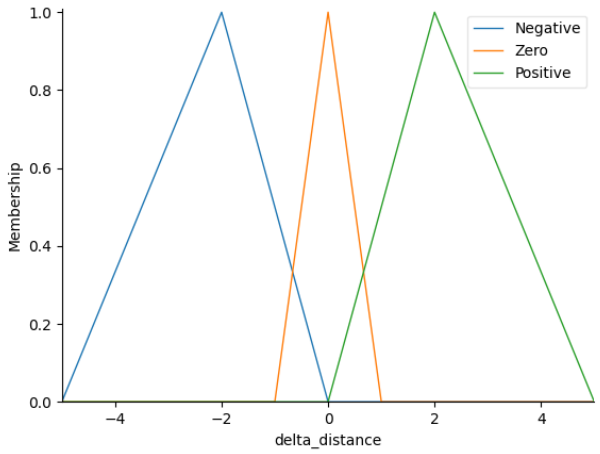


Рисунок 2.2 – Функция принадлежности изменения расстояния

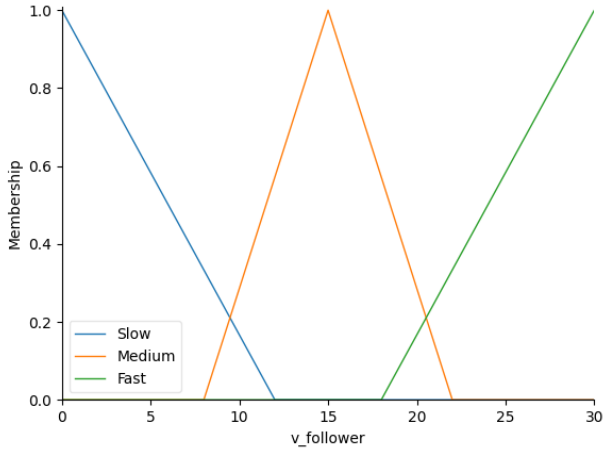


Рисунок 2.3 – Функция принадлежности скорости автопилота

## 2.3 Реализация алгоритма Ларсена

В библиотеке scikit-fuzzy в качестве алгоритма логического вывода применяется алгоритм Мамдани, без возможности выбора альтернативы. Одним из вариантов реализации алгоритма Ларсена с использованием scikit-fuzzy, требует модификации исходного кода библиотеки.

Листинг 2.1 – Исходный код библиотеки scikit-fuzzy, rule.py

```
1 class Rule(object):
2 def __init__(self, antecedent=None, consequent=None,
    label=None, and_func=np.fmin, or_func=np.fmax):
```

Как видно из листинга 2.1, выходом функции логического И является минимальное из двух значений `and_func = np.fmin` (по Мамдани). Заменив этот кусок кода на `and_func=np.multiply`, получим алгоритм вывода Ларсена.

## 2.4 Результаты

### 2.4.1 Моделирование реакции системы на единичное ступенчатое воздействие

На рисунках 2.4-2.6 показаны результаты моделирования системы при единичном ступенчатом воздействии.

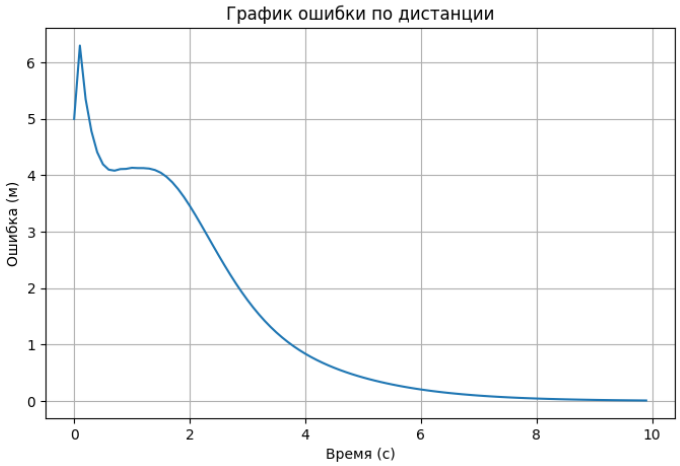


Рисунок 2.4 – График изменения ошибки системы

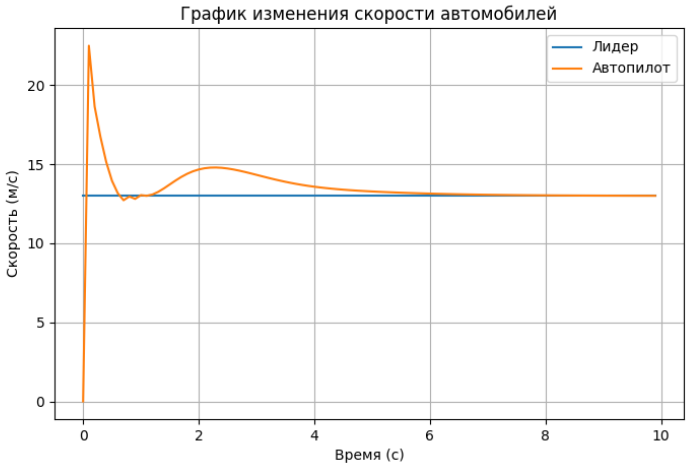


Рисунок 2.5 – График изменения скорости автомобилей

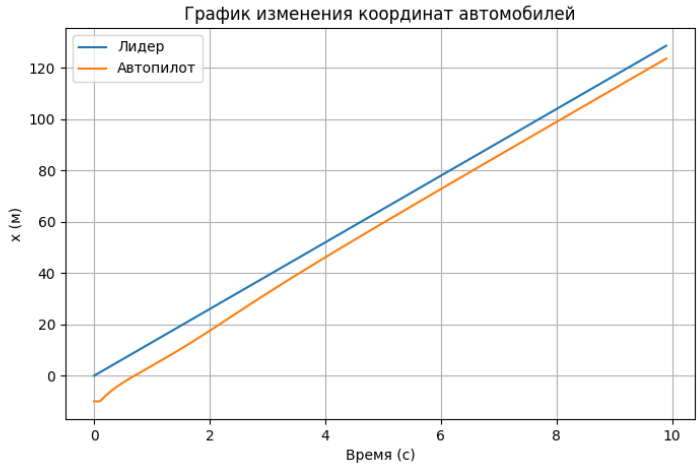


Рисунок 2.6 – График изменения координат автомобилей

### 2.4.2 Моделирование реакции системы на двухступенчатое воздействие

На рисунках 2.7-2.9 показаны результаты моделирования системы при двухступенчатом воздействии.

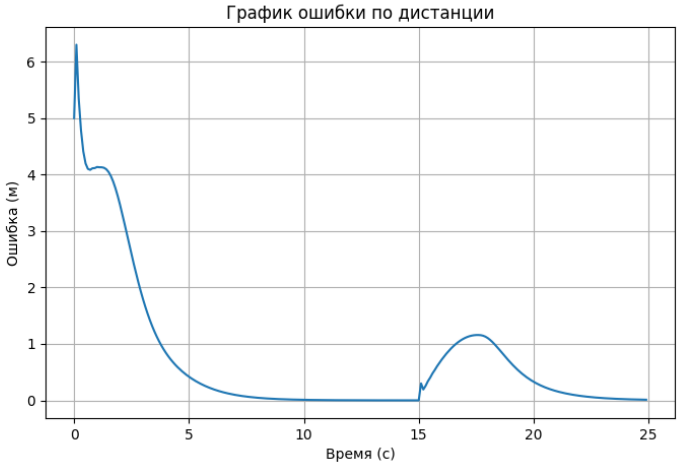


Рисунок 2.7 – График изменения ошибки системы

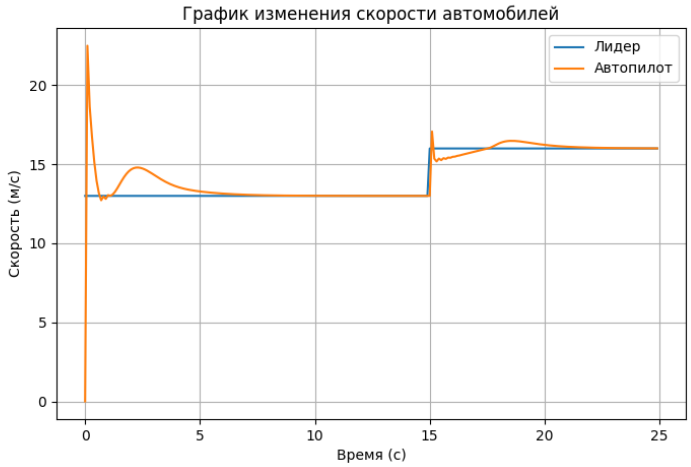


Рисунок 2.8 – График изменения скорости автомобилей

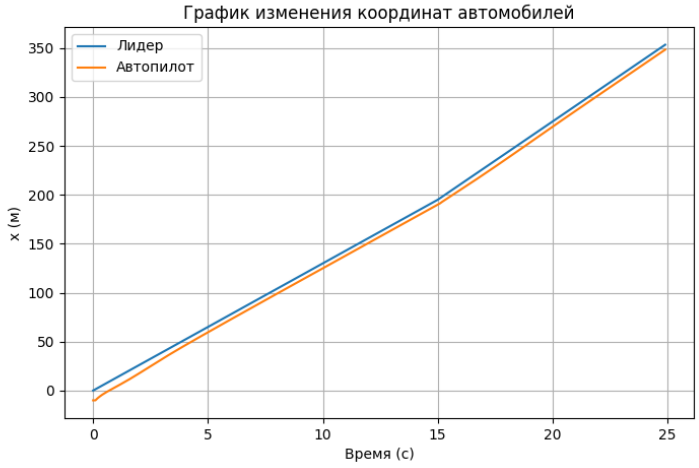


Рисунок 2.9 – График изменения координат автомобилей

Среднеквадратичная ошибка поддержания дистанции при моделировании системы составила:

$$MSE = 0.0540.$$

Шаг интегрирования $dt = 0.1$ и время моделирования $T = 50\,c$.

## 2.5 Работа системы при алгоритме логического вывода Мамдани

Для сравнения рассмотрим результаты работы системы при применении алгоритма Мамдани. На рисунках 2.10 и 2.11 приведены результаты моделирования системы при двухступенчатом воздействии, выполненном с

использованием алгоритма Мамдани. Как видно из полученных результатов, переходный режим системы имеет более колебательный характер, чем при использовании алгоритма Ларсена.

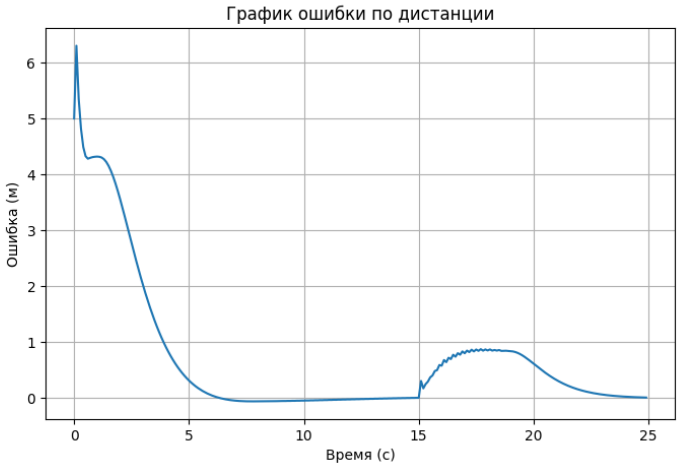


Рисунок 2.10 – График изменения ошибки системы

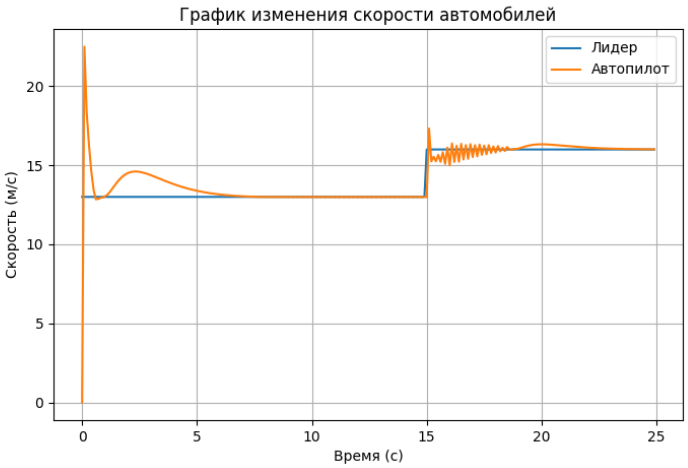


Рисунок 2.11 – График изменения скорости автомобилей

# ПРИЛОЖЕНИЕ А

Листинг А.1 – Исходный код программы

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt
import time


distance_error = ctrl.Antecedent(np.arange(-10, 10.1, 0.1),
    'distance_error')
delta_distance = ctrl.Antecedent(np.arange(-5, 5.1, 0.1),
    'delta_distance')
v_autopilot = ctrl.Consequent(np.arange(0, 30.1, 0.1),
    'v_autopilot', defuzzify_method="centroid")

distance_error['Negative'] =
    fuzz.trimf(distance_error.universe, [-10, -5, 0])
distance_error['Zero'] = fuzz.trimf(distance_error.universe,
    [-2, 0, 2])
distance_error['Positive'] =
    fuzz.trimf(distance_error.universe, [0, 5, 10])

# distance_error.view()

delta_distance['Negative'] =
    fuzz.trimf(delta_distance.universe, [-5, -2, 0])
delta_distance['Zero'] = fuzz.trimf(delta_distance.universe,
    [-1, 0, 1])
delta_distance['Positive'] =
    fuzz.trimf(delta_distance.universe, [0, 2, 5])

# delta_distance.view()

v_autopilot['Slow'] = fuzz.trimf(v_autopilot.universe, [0, 0,
    12])
v_autopilot['Medium'] = fuzz.trimf(v_autopilot.universe, [8,
    15, 22])
```

```
v_autopilot['Fast'] = fuzz.trimf(v_autopilot.universe, [18, 30, 30])

# v_autopilot.view()

rule1 = ctrl.Rule(distance_error['Positive'] & delta_distance['Negative'], v_autopilot['Slow'])
rule2 = ctrl.Rule(distance_error['Positive'] & delta_distance['Zero'], v_autopilot['Slow'])
rule3 = ctrl.Rule(distance_error['Positive'] & delta_distance['Positive'], v_autopilot['Medium'])

rule4 = ctrl.Rule(distance_error['Zero'] & delta_distance['Negative'], v_autopilot['Medium'])
rule5 = ctrl.Rule(distance_error['Zero'] & delta_distance['Zero'], v_autopilot['Medium'])
rule6 = ctrl.Rule(distance_error['Zero'] & delta_distance['Positive'], v_autopilot['Fast'])

rule7 = ctrl.Rule(distance_error['Negative'] & delta_distance['Negative'], v_autopilot['Medium'])
rule8 = ctrl.Rule(distance_error['Negative'] & delta_distance['Zero'], v_autopilot['Fast'])
rule9 = ctrl.Rule(distance_error['Negative'] & delta_distance['Positive'], v_autopilot['Fast'])

fuzzy_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4, rule5, rule6, rule7, rule8, rule9])
fuzzy_sim = ctrl.ControlSystemSimulation(fuzzy_ctrl)

dt = 0.1
t = np.arange(0, 25, dt)
d_ref = 5.0
v_leader = 13 + 3 * (t >= 15)

x_leader = np.zeros_like(t)
x_follower = np.zeros_like(t)
v_auto = np.zeros_like(t)

x_leader[0] = 0
x_follower[0] = -10
```

```
v_auto[0] = 0
error_int = 0
Ki = 0.5

start = time.time()
for i in range(1, len(t)):
    x_leader[i] = x_leader[i-1] + v_leader[i-1] * dt
    x_follower[i] = x_follower[i-1] + v_auto[i-1] * dt

    distance = x_leader[i] - x_follower[i]
    error = distance - d_ref
    delta_error = (x_leader[i] - x_leader[i-1]) -
       (x_follower[i] - x_follower[i-1])

    error_int += error * dt
    e_fuzzy = error + Ki * error_int

    fuzzy_sim.input['distance_error'] = e_fuzzy
    fuzzy_sim.input['delta_distance'] = delta_error
    fuzzy_sim.compute()

    v_auto[i] = 0.5 * v_auto[i-1] + 1.5*
       fuzzy_sim.output['v_autopilot']
end = time.time()

# График ошибки по дистанции
plt.figure(figsize=(8, 5))
plt.plot(t, [x_leader[i] - x_follower[i] - d_ref for i in
   range(len(t))])
plt.title('График ошибки по дистанции')
plt.ylabel('Ошибка (м)')
plt.xlabel('Время (с)')
plt.grid()
plt.show()

# График изменения координат автомобилей
plt.figure(figsize=(8, 5))
plt.plot(t, x_leader, label='Лидер')
plt.plot(t, x_follower, label='Автопилот')
plt.title('График изменения координат автомобилей')
plt.ylabel('x (м)')
```

```
plt.xlabel('Время (с)')
plt.legend()
plt.grid()
plt.show()

# График изменения скорости автомобилей
plt.figure(figsize=(8, 5))
plt.plot(t, v_leader, label='Лидер')
plt.plot(t, v_auto, label='Автопилот')
plt.title('График изменения скорости автомобилей')
plt.ylabel('Скорость (м/с)')
plt.xlabel('Время (с)')
plt.legend()
plt.grid()
plt.show()


distance_errors = np.array([x_leader[i] - x_follower[i] -
    d_ref for i in range(len(t))])

mask = (t >= 5) & (t <= 50)

rmse_interval = np.sqrt(np.mean(distance_errors[mask]**2))

print(f"Среднеквадратичная ошибка: {rmse_interval:.4f} м")
print(f"Время выполнения: {end - start:.4f} с")
```