گزارش مینی پروژه ۳
هوش مصنوعی

دکتر علیاری

سینا حسن‌پور شماره دانشجویی: ۴۰۲۱۶۷۲۳

حسین اسکندری شماره دانشجویی: ۴۰۲۱۴۷۰۳

۳۰ تیر ۱۴۰۴

فهرست مطالب

مخزن گیت‌هاب

برای مشاهده و دانلود کدهای مربوط به این پروژه، می‌توانید به مخزن گیت‌هاب مراجعه کنید:

مخزن گیت‌هاب پروژه‌ها

پیوست: لینک‌های اجرای آنلاین پروژه‌ها در Google Colab

برای مشاهده و اجرای مستقیم هر یک از پرسش‌ها در محیط Google Colab، می‌توانید از لینک‌های زیر استفاده کنید:

- **هر ۷ پرسش:** مشاهده در Google Colab

۱ پرسش یک

در این پروژه، هدف طراحی یک سیستم فازی برای تقریب تابع غیرخطی زیر است:

$$h(x) = e^{-\pi x^2} + x \cdot \sin(\pi x)$$

همچنین یک مدل شبکه عصبی پرسپترون چندلایه (MLP) نیز برای مقایسه دقت پیش‌بینی استفاده شده است. دامنه تعریف تابع $x \in [-1, 1]$ بوده و نمونه‌برداری با گام $\varepsilon = 0.1$ انجام شده است. در مجموع ۳۰۰ داده ورودی تولید و خروجی تابع برای آن‌ها محاسبه شده است.

طراحی سیستم فازی

برای مدل Mamdani از ۵ مجموعه زنگوله‌ای (gbellmf) برای متغیر ورودی و ۵ مجموعه مثلثی (trimf) برای خروجی استفاده شده است. محدوده خروجی تابع نیز با استفاده از مقادیر $\min(h)$، median(h) و $\max(h)$ نرمال‌سازی شده است.

مجموعه‌های فازی تعریف‌شده:

- **ورودی (x)**: very_low، low، medium، high very_high
- **خروجی (y)**: very_low، low، medium، high very_high

قواعد فازی (Fuzzy Rules):

- If x is very_low then y is very_low
- If x is low or medium then y is low
- If x is medium or high then y is high
- If x is low or medium or high then y is medium
- If x is very_high then y is very_high

طراحی و آموزش MLP

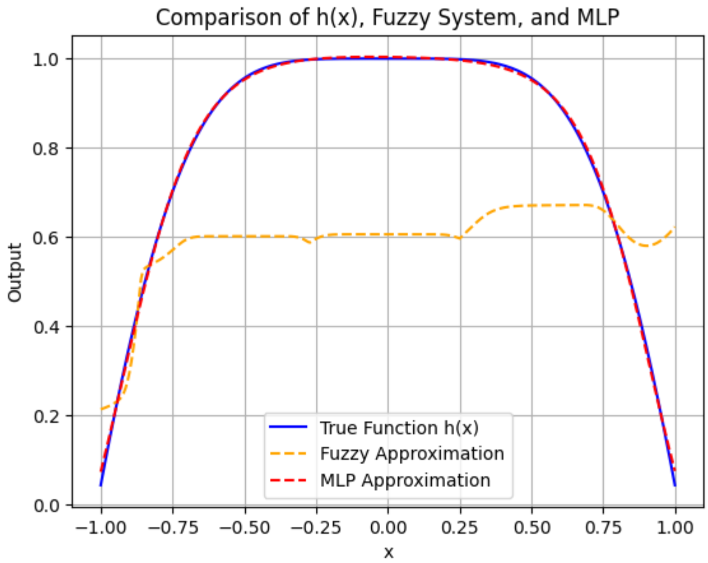
مدل MLP طراحی‌شده دارای ساختار زیر است:

- دو لایه پنهان (هرکدام با ۱۶ نورون)
- تابع فعال‌سازی: Tanh
- تابع هزینه: MSE
- بهینه‌ساز: Adam با نرخ یادگیری ۰.۰۱
- تعداد تکرار آموزش: ۵۰۰ دوره

مقایسه عملکرد مدل‌ها

پس از پیاده‌سازی سیستم فازی و آموزش شبکه عصبی، عملکرد هر دو مدل با تابع اصلی $h(x)$ مقایسه شد. خطای میانگین مربعات (MSE) برای هر مدل به صورت زیر محاسبه گردید:

- **MSE سیستم فازی:** 0.09557
- **MSE شبکه عصبی MLP:** 0.00004



شکل ۱: مقایسه نمودار تابع $h(x)$، خروجی سیستم فازی و MLP

بررسی انواع Defuzzifier

برای تحلیل حساسیت خروجی فازی به روش defuzzification ، سه روش زیر بررسی شدند:

- Centroid
- Bisector
- Mean of Maxima (MOM)

Error MSE	Method Defuzzification
0.09557	Centroid
0.09287	Bisector
0.07139	MOM

جدول ۱: مقایسه عملکرد انواع Defuzzifier در سیستم فازی

نتیجه‌گیری

مدل MLP با خطای بسیار پایین‌تر، تقریب دقیق‌تری از تابع ارائه داده است. با این حال، سیستم فازی مزایایی همچون سادگی در طراحی، تفسیرپذیری و امکان اعمال قوانین انسانی دارد. در بین روش‌های Defuzzifier ، روش MOM کمترین خطا را در تقریب ارائه کرده است.

کد کامل پایتون سوال یک:

۱. بارگذاری کتابخانه‌ها

برای اجرای این پروژه، ابتدا کتابخانه‌های لازم فراخوانی شدند. این کتابخانه‌ها وظایف متنوعی دارند:

- numpy برای محاسبات عددی و تولید داده‌ها
- matplotlib برای رسم نمودارها
- sklearn.metrics برای محاسبه خطای مدل‌ها (مانند MSE)
- torch برای ساخت و آموزش شبکه عصبی پرسپترون
- skfuzzy برای پیاده‌سازی سیستم فازی

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
import torch
import torch.nn as nn
import skfuzzy as fuzz
from skfuzzy import control as ctrl
```

۲. تولید داده و تعریف تابع هدف

برای آموزش و ارزیابی سیستم‌ها، باید یک تابع هدف پیچیده و غیرخطی تعریف شود. تابع زیر ترکیبی از تابع گاوسی و سینوسی است که رفتار پیچیده‌تری از توابع ساده دارد و برای تست مدل‌های فازی و عصبی مناسب است:

```
x_vals = np.linspace(-1, 1, 300)
y_vals = np.exp(-np.pi * x_vals**2) + x_vals * np.sin(np.pi * x_vals)
```

۳. طراحی سیستم فازی

برای ساخت سیستم فازی باید ورودی‌ها و خروجی‌ها را به‌صورت فازی مدل کرد. در این بخش، با استفاده از توابع گوسی تعمیم‌یافته (gbellmf) پنج مجموعه فازی برای متغیر ورودی x تعریف شده‌اند تا بتوانیم حالات مختلف مقدار ورودی را با دقت مدل کنیم. همچنین، خروجی y با توابع مثلثی (trimf) مدل‌سازی شده است تا روند defuzzification ساده‌تر و قابل تفسیرتر باشد.

```
x_input = ctrl.Antecedent(np.linspace(-1, 1, 300), 'x')
y_output = ctrl.Consequent(np.linspace(min(y_vals), max(y_vals), 300), 'y')

x_input['very_low']  = fuzz.gbellmf(x_input.universe, 0.1, 3, -1.0)
x_input['low']       = fuzz.gbellmf(x_input.universe, 0.25, 3, -0.5)
x_input['medium']    = fuzz.gbellmf(x_input.universe, 0.3, 3, 0.0)
x_input['high']      = fuzz.gbellmf(x_input.universe, 0.3, 3, 0.5)
x_input['very_high'] = fuzz.gbellmf(x_input.universe, 0.3, 3, 1.0)

y_output['very_low']  = fuzz.trimf(...)
```

۴. تعریف قوانین فازی

قوانین فازی برای نگاشت مجموعه‌های ورودی به خروجی طراحی شده‌اند. هدف از این قوانین، توصیف رابطه‌ی زبانی بین ورودی و خروجی است. به‌طور مثال، اگر ورودی ”خیلی پایین“ باشد، خروجی نیز باید ”خیلی پایین“ باشد.

```
rules = [
    ctrl.Rule(x_input['very_low'], y_output['very_low']),
    ctrl.Rule(x_input['low'] | x_input['medium'], y_output['low']),
    ...
]
```

۵. تعریف شبکه عصبی MLP

برای مقایسه عملکرد سیستم فازی با یک مدل یادگیری ماشین، یک شبکه‌ی عصبی پرسپترون چندلایه تعریف کردیم. این مدل شامل دو لایه پنهان و توابع فعال‌سازی Tanh است. دلیل استفاده از Tanh، توانایی آن در مدل‌سازی رفتارهای غیرخطی پیچیده است.

```
class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.hidden = nn.Sequential(
            nn.Linear(1, 16),
            nn.Tanh(),
            ...
        )
```

۶. آموزش مدل MLP

مدل عصبی با استفاده از الگوریتم Adam آموزش داده شد. این الگوریتم برای مسائل غیرخطی و noisy مناسب است. از تابع هزینه MSELoss برای ارزیابی دقت مدل استفاده شد.

```
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

for epoch in range(500):
    ...
```

۷. رسم نمودار مقایسه‌ای

برای مشاهده و مقایسه‌ی عملکرد مدل‌ها، خروجی سیستم فازی، شبکه عصبی و تابع اصلی روی یک نمودار رسم شدند. این کار به ما امکان می‌دهد تا به‌صورت بصری کیفیت تقریب را بررسی کنیم.

```
plt.plot(x_vals, y_vals, label='True Function h(x)', color='blue')
plt.plot(x_vals, predicted_outputs, '--', label='Fuzzy Approximation',
    color='orange')
plt.plot(x_vals, y_pred_mlp, '--', label='MLP Approximation', color='red')
...
```

۸. مقایسه‌ی انواع Defuzzifier

برای بررسی تأثیر روش Defuzzification بر خروجی نهایی، سه روش معروف centroid، bisector و mom مورد استفاده قرار گرفتند. برای هر روش، خروجی سیستم فازی مجدداً محاسبه شده و مقدار خطای میانگین مربعات (MSE) اندازه‌گیری شد. هدف این مقایسه، یافتن دقیق‌ترین روش defuzzification است.

```
defuzz_methods = ['centroid', 'bisector', 'mom']
for method in defuzz_methods:
    y_output.defuzzify_method = method
    ...
    mse = mean_squared_error(...)
```

۲ پرسش دو

در این پروژه، هدف تقریب تابع غیرخطی زیر با استفاده از دو سیستم فازی بر پایه روابط کران اول و دوم از کتاب لی‌زین وانگ[1] است:

$$g(x_1, x_2) = \frac{1}{3 + x_1 + x_2}$$

تابع فوق بر روی دامنه مربعی $U = [-1, 1] \times [-1, 1]$ با دقت یکنواخت $\varepsilon = 0.1$ مورد بررسی قرار گرفته و تقریب زده شده است.

طراحی سیستم فازی بر پایه کران اول و دوم

برای پیاده‌سازی دو سیستم فازی، از روابط زیر استفاده شده است:

- کران مرتبه اول (فازی‌سازی روی x_1):

$$\hat{g}_1(x_1, x_2) = \frac{\sum_j \mu_{A_1^j}(x_1) \cdot g(e_j, x_2)}{\sum_j \mu_{A_1^j}(x_1)}$$

- کران مرتبه دوم (فازی‌سازی روی x_2):

$$\hat{g}_2(x_1, x_2) = \frac{\sum_j \mu_{A_2^j}(x_2) \cdot g(x_1, e_j)}{\sum_j \mu_{A_2^j}(x_2)}$$

در این روابط:

- e_j: مراکز مجموعه‌های فازی، به صورت یکنواخت بین $[-1, 1]$
- $\mu_A(x)$: تابع عضویت مثلثی به صورت زیر تعریف شده است:

$$\mu(x; c, h) = \max\left(1 - \frac{|x - c|}{h},\ 0\right)$$

تنظیمات سیستم فازی:

- تعداد مجموعه‌های فازی: $N = 7$
- عرض هر تابع عضویت مثلثی: $h = 2/(N - 1)$
- شبکه ارزیابی: ۴۱×۴۱ نقطه روی $[-1, 1]^2$

[1] Li-Xin Wang, author of *A Course in Fuzzy Systems and Control*, Prentice Hall, 1997.

مقایسه عددی بین مدل‌ها

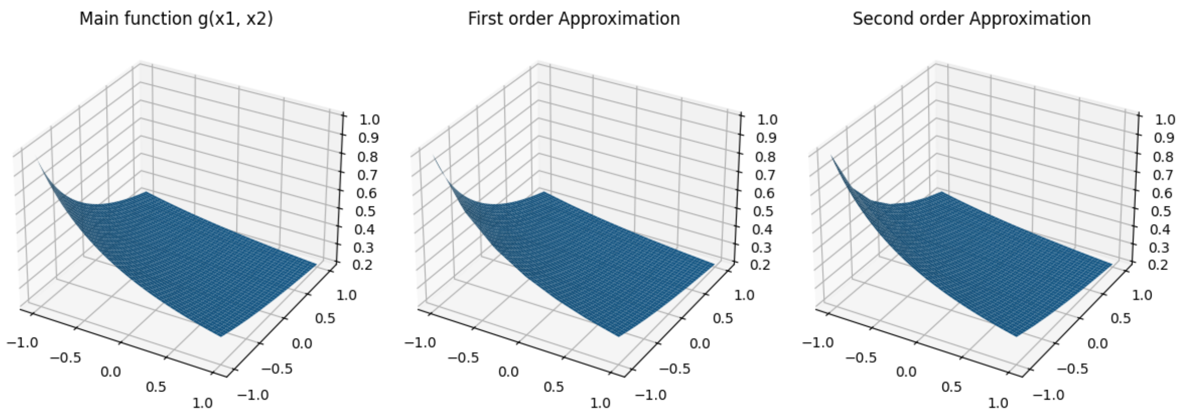
بیشینهٔ خطای تقریبی (یعنی $\max|g-\hat{g}|$) برای دو روش محاسبه شده است:

روش تقریب	بیشینه خطا (Max Error)
کران اول + میانگین وزنی	0.01793
کران دوم + میانگین وزنی	0.01793

جدول ۲: مقایسه خطای تقریبی برای دو سیستم فازی طراحی‌شده

مقایسه تصویری سطوح تقریبی

نمودارهای زیر سطح تابع اصلی و دو تقریب را به صورت سه‌بعدی نشان می‌دهند:



شکل ۲: مقایسه سطح تابع اصلی و تقریب‌های فازی بر پایه کران اول و دوم

نتیجه‌گیری

در این پروژه، دو سیستم فازی بر پایه کران مرتبه اول و دوم طراحی و پیاده‌سازی شدند. با استفاده از defuzzification میانگین وزنی، هر دو روش توانستند تقریب دقیقی با خطای یکنواخت کمتر از $\varepsilon = 0.1$ ارائه دهند. نتایج نشان می‌دهد که هر دو روش تقریب رفتار مشابه و قابل اعتمادی دارند و در بسیاری از مسائل کنترلی و تخمین توابع دو متغیره می‌توان از آن‌ها بهره برد.

کد کامل پایتون برای پرسش دو

در این بخش، مراحل پیاده‌سازی سیستم فازی بر پایه‌ی کران‌های مرتبه اول و دوم به زبان پایتون ارائه شده‌اند. هدف این کد، مقایسه‌ی دو تقریب از تابع دو متغیره $g(x_1, x_2)$ در دامنه‌ی مربعی $[-1,1]^2$ است. در ادامه، کد به بخش‌های مختلف تقسیم و توضیح داده شده است:

۱. بارگذاری کتابخانه‌ها و تعریف تابع هدف

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def g(x1, x2):
    return 1.0 / (3 + x1 + x2)
```

کتابخانه‌های عددی و ترسیمی موردنیاز بارگذاری شده‌اند. تابع $g(x_1, x_2)$ طبق صورت سوال تعریف شده است.

۲. تعریف تابع عضویت مثلثی فازی

```
def tri_mf(x, center, width):
    return np.maximum(1 - abs(x - center) / width, 0)
```

تابع عضویت مثلثی برای استفاده در محاسبه درجات عضویت فازی تعریف شده است. این تابع مقدار صفر تا یک را تولید می‌کند.

۳. تنظیم دامنه و پارامترهای فازی

```
xs = np.linspace(-1, 1, 41)
ys = np.linspace(-1, 1, 41)

N = 7
centers = np.linspace(-1, 1, N)
width = 2.0 / (N - 1)
```

شبکه‌ای ۴۱×۴۱ روی ناحیه‌ی $[-1, 1]^2$ تعریف شده است. همچنین، ۷ مجموعه فازی با مراکز یکنواخت و عرض مثلثی مشخص تعیین شده‌اند.

۴. محاسبه مقادیر واقعی تابع روی شبکه

```
G_true = np.zeros((len(xs), len(ys)))
for i in range(len(xs)):
    for j in range(len(ys)):
        G_true[i, j] = g(xs[i], ys[j])
```

مقادیر واقعی تابع g روی تمام نقاط شبکه محاسبه و ذخیره شده‌اند تا به عنوان مرجع مقایسه استفاده شوند.

۵. تقریب کران مرتبه اول (فازی‌سازی روی x_1)

```
G1 = np.zeros_like(G_true)
for j in range(len(ys)):
    y = ys[j]
    samp = [g(c, y) for c in centers]
    for i in range(len(xs)):
        x = xs[i]
        mus = [tri_mf(x, c, width) for c in centers]
        num = sum(mus[k] * samp[k] for k in range(N))
        den = sum(mus)
        G1[i, j] = num / den if den != 0 else 0
```

در این قسمت، مقدار تابع برای هر نقطه با استفاده از درجه عضویت در x_1 و وزن‌دهی مقادیر $g(c_j, x_2)$ محاسبه شده است.

۶. تقریب کران مرتبه دوم (فازی‌سازی روی x_2)

```
G2 = np.zeros_like(G_true)
for i in range(len(xs)):
    x = xs[i]
    samp = [g(x, c) for c in centers]
    for j in range(len(ys)):
        y = ys[j]
        mus = [tri_mf(y, c, width) for c in centers]
        num = sum(mus[k] * samp[k] for k in range(N))
        den = sum(mus)
        G2[i, j] = num / den if den != 0 else 0
```

مشابه بخش قبل، اما این‌بار فازی‌سازی روی متغیر دوم x_2 انجام گرفته است.

۷. محاسبه‌ی خطای بیشینه (Uniform Error)

```
err1 = np.max(np.abs(G_true - G1))
err2 = np.max(np.abs(G_true - G2))
print("Max error (1st order):", err1)
print("Max error (2nd order):", err2)
```

در این بخش، بیشینه خطای هر دو تقریب نسبت به تابع اصلی محاسبه شده است.

۸. ترسیم نمودارهای سه‌بعدی سطوح

```
fig = plt.figure(figsize=(12, 4))
X, Y = np.meshgrid(xs, ys)

ax1 = fig.add_subplot(131, projection='3d')
ax1.plot_surface(X, Y, G_true.T)
ax1.set_title('Main function g(x1, x2)')

ax2 = fig.add_subplot(132, projection='3d')
ax2.plot_surface(X, Y, G1.T)
ax2.set_title('First order Approximation')

ax3 = fig.add_subplot(133, projection='3d')
ax3.plot_surface(X, Y, G2.T)
ax3.set_title('Second order Approximation')

plt.tight_layout()
plt.show()
```

سه نمودار سطحی ترسیم شده‌اند تا تفاوت بین تابع اصلی و دو تقریب به صورت بصری مقایسه شود.

۳ پرسش سه

در این پروژه، هدف پیاده‌سازی یک برنامه‌ی کامل بر اساس روش جدول جستجو (Lookup Table) برای تقریب مقادیر آینده‌ی یک سری زمانی پیچیده (سری Mackey–Glass) است. این روش از ایده‌ی پرکردن فضای ورودی با مقادیر میانگین و بهره‌گیری از همسایه‌های فازی برای خانه‌های خالی استفاده می‌کند.
تابع Mackey–Glass یک معادلهٔ دیفرانسیل تأخیری غیرخطی است که رفتار آشوبناک دارد و در بسیاری از مسائل پیش‌بینی و کنترل استفاده می‌شود.

تنظیمات و پارامترها

در ابتدای برنامه، پارامترهای اصلی به شرح زیر تعیین شدند:

- $\mathrm{BINS} = 10$ تعداد باین‌ها در هر بُعد از فضای ورودی (تقسیم یکنواخت بازه $[0, 1]$)
- $\mathrm{TRAIN\_FRAC} = 0.7$ نسبت داده‌های آموزشی به کل داده‌ها
- $\mathrm{H} = 1$ گام تفاضلی برای حل معادله دیفرانسیل Mackey–Glass
- $\mathrm{LENGTH} = 4000$ طول کل سری زمانی تولیدشده

تولید داده با معادله Mackey–Glass

معادله‌ی Mackey–Glass به صورت گسسته با گام $h = 1$ حل شده و مقدار اولیه حذف گردید. سپس داده‌ها برای استفاده در مدل نرمال‌سازی شدند:

هدف: $x(k),\ x(k-6),\ x(k-12),\ x(k-18) \rightarrow x(k+6)$

شکل ورودی‌ها: برداری چهاربعدی **شکل خروجی‌ها:** مقدار آینده

ساخت جدول جستجو

فضای ورودی به صورت شبکه‌ای با ۱۰×۱۰×۱۰×۱۰ تقسیم شده و برای هر بردار ورودی آموزشی، باین متناظر مشخص و مقدار هدف در آن ثبت شد. سپس برای هر خانه، میانگین مقادیر ثبت‌شده به عنوان مقدار lookup ثبت گردید.
برای خانه‌هایی که در آموزش دیده نشده‌اند، از یک تابع جستجوی همسایگی استفاده شده است.

تابع تقریب برای خانه‌های خالی

اگر خانه‌ای در جدول lookup موجود نبود، جستجو در همسایگی انجام می‌شود. ابتدا همسایه‌های فاصله ۱، سپس ۲، ... تا حداکثر فاصله ۴ بررسی می‌شوند. اگر هیچ مقدار یافت نشود، مقدار پیش‌فرض ۵.۰ برگردانده می‌شود.

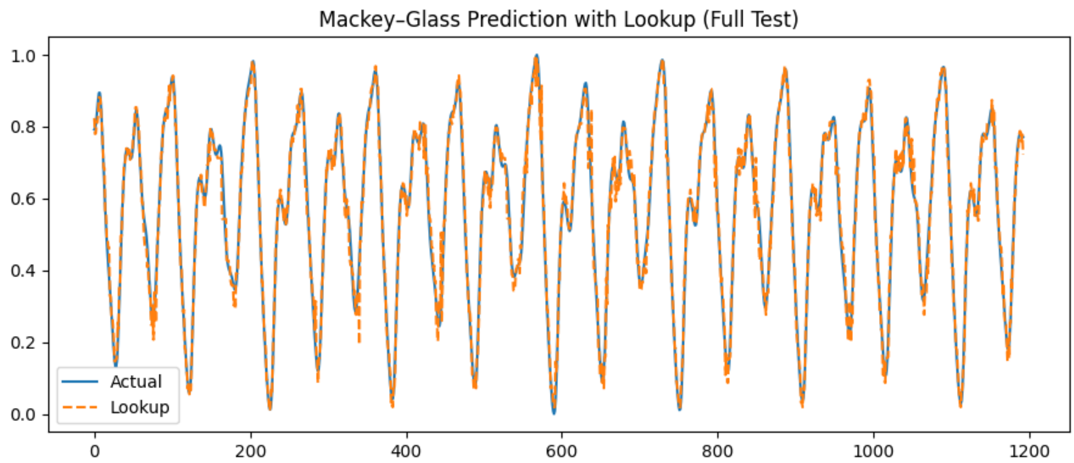
پیش‌بینی و ارزیابی

با استفاده از جدول جستجو، مقادیر $x(k+6)$ برای مجموعه تست پیش‌بینی شدند. سپس خطای میانگین مربعات (MSE) بین خروجی واقعی و پیش‌بینی محاسبه شد:

- **MSE Test:** 0.0009

نمودار مقایسه‌ای کامل

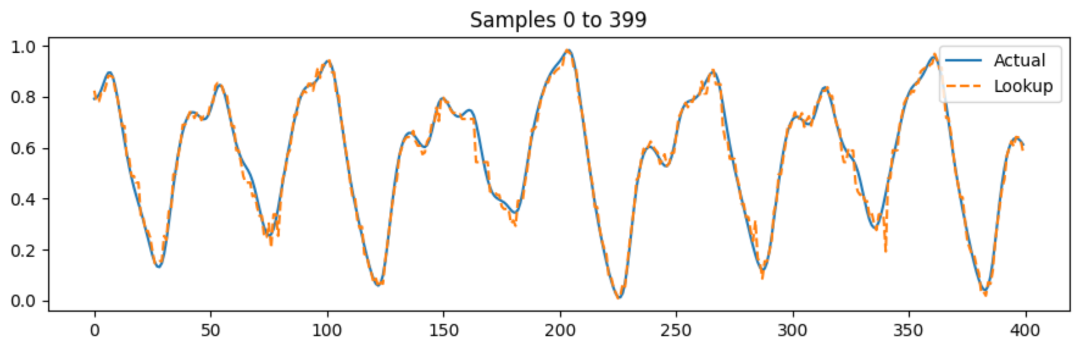
نمودار زیر مقایسه پیش‌بینی با دادهٔ واقعی برای کل مجموعه تست را نشان می‌دهد:



شکل ۳: پیش‌بینی کامل سری Mackey–Glass با جدول Lookup

نمودار زوم شده

برای بررسی دقیق‌تر عملکرد مدل، پنجره‌هایی به طول ۴۰۰ نمونه با گام ۱۲۰۰ استخراج شده‌اند. نمونه‌ای از این مقایسه‌ها در شکل زیر آمده است:



شکل ۴: مقایسه مقطعی بین خروجی واقعی و پیش‌بینی‌شده (نمونه‌های ۰ تا ۳۹۹)

کد کامل پایتون

در ادامه، کد کامل پایتون شامل تمام مراحل از تولید داده تا ارزیابی آورده شده است:

در این بخش، پیاده‌سازی گام‌به‌گام روش جدول جستجو برای پیش‌بینی سری زمانی Mackey–Glass در زبان Python آورده شده است. کد شامل مراحل تولید داده، ساخت جدول، پر کردن مقادیر گمشده، پیش‌بینی، و ترسیم نمودار می‌باشد.

۱. بارگذاری کتابخانه‌ها و تعریف پارامترها

```
import numpy as np
import matplotlib.pyplot as plt
from itertools import product
from sklearn.metrics import mean_squared_error

BINS = 10
TRAIN_FRAC = 0.7
H = 1
LENGTH = 4000

STEP = 1200
SEG_LEN = 400
```

در این بخش، کتابخانه‌های لازم بارگذاری شده‌اند و پارامترهای کلیدی مانند تعداد باین‌ها، نسبت دادهٔ آموزشی و طول سری زمانی تنظیم شده‌اند.

۲. تولید سری زمانی Mackey–Glass

```
def mackey_glass(beta=0.2, gamma=0.1, n=10, tau=17,
                 h=H, length=LENGTH, x0=1.2):
    delay = int(tau / h)
    x = x0 * np.ones(length + delay)
    for k in range(delay, length + delay - 1):
        x[k+1] = x[k] + h * (
            beta * x[k-delay] / (1 + x[k-delay]**n) - gamma * x[k]
        )
    return x[delay:]
```

معادلهٔ Mackey–Glass با استفاده از روش اختلاف محدود حل شده است. تأخیر زمانی $\tau = 17$ و مقدار اولیه $x_0 = 1.2$ در نظر گرفته شده‌اند.

۳. نرمال‌سازی و ساخت داده‌های ورودی–خروجی

```
ts = mackey_glass()
ts = (ts - ts.min()) / (ts.max() - ts.min())

max_k = len(ts) - 24
X = np.column_stack([
    ts[18:max_k+18], ts[12:max_k+12],
    ts[6:max_k+6],    ts[:max_k]
])
y = ts[24:]

split = int(TRAIN_FRAC * len(X))
X_tr, X_te = X[:split], X[split:]
y_tr, y_te = y[:split], y[split:]
```

داده‌های خام تولیدی ابتدا نرمال‌سازی شدند. سپس ورودی‌ها به شکل $[x(k), x(k-6), x(k-12), x(k-18)]$ و خروجی به شکل $x(k+6)$ آماده‌سازی شدند.

۴. ساخت جدول جستجو و محاسبه مقادیر میانگین

```
edges  = np.linspace(0, 1, BINS + 1)
to_idx = lambda v: tuple(np.searchsorted(edges, v, side='right') - 1)

table = {}
for x_vec, target in zip(X_tr, y_tr):
    table.setdefault(to_idx(x_vec), []).append(target)
```

```
lookup = {cell: np.mean(vals) for cell, vals in table.items()}
```

داده‌ها به فضای شبکه‌ای نگاشت شده‌اند. برای هر سلول باین، میانگین مقادیر خروجی به عنوان مقدار lookup در نظر گرفته شده است.

۵. تعریف تابع پر کردن خانه‌های خالی با همسایه‌ها

```
def get_value(cell, max_r=4):
    if cell in lookup:
        return lookup[cell]
    for r in range(1, max_r + 1):
        for delta in product(range(-r, r+1), repeat=4):
            nb = tuple(
                np.clip(c + d, 0, BINS - 1)
                for c, d in zip(cell, delta)
            )
            if nb in lookup:
                return lookup[nb]
    return 0.5
```

برای سلول‌هایی که مقدار مستقیم ندارند، از همسایه‌های نزدیک با فاصله‌ی حداکثر ۴ استفاده می‌شود. مقدار پیش‌فرض در صورت یافت‌نشدن همسایه، برابر ۰.۵ در نظر گرفته شده است.

۶. پیش‌بینی و محاسبه‌ی خطا

```
y_pred = np.array([get_value(to_idx(x)) for x in X_te])
print(f"Test MSE: {mean_squared_error(y_te, y_pred):.4f}")
```

با استفاده از جدول جستجو، مقادیر خروجی تست پیش‌بینی شده و خطای MSE برای ارزیابی عملکرد محاسبه گردیده است.

۷. ترسیم نمودار پیش‌بینی کامل

```
plt.figure(figsize=(9,4))
plt.plot(y_te, label='Actual')
plt.plot(y_pred, '--', label='Lookup')
plt.title('-MackeyGlass Prediction with Lookup (Full Test)')
plt.legend()
plt.tight_layout()
plt.show()
```

نمودار پیش‌بینی کل مجموعه تست با منحنی واقعی مقایسه شده است.

۸. ترسیم نمودارهای پنجره‌ای مقطعی

```
for start in range(0, len(y_te) - SEG_LEN + 1, STEP):
    end = start + SEG_LEN
    plt.figure(figsize=(9,3))
    plt.plot(range(start, end), y_te[start:end], label='Actual')
    plt.plot(range(start, end), y_pred[start:end], '--', label='Lookup')
    plt.title(f'Samples {start} to {end-1}')
    plt.legend()
    plt.tight_layout()
    plt.show()
```

برای بررسی موضعی‌تر، بازه‌هایی به طول ۴۰۰ نمونه و گام ۱۲۰۰ انتخاب شده‌اند و در هر بازه نمودار دقیق پیش‌بینی ترسیم شده است.

۴ پرسش چهارم

تحلیل مسئله

هدف این پروژه، شناسایی و مدل‌سازی یک سیستم دینامیکی تفاضلی است که دارای یک مؤلفه‌ی غیرخطی ناشناخته به صورت تابع $g(u(k))$ می‌باشد. ساختار کلی این سیستم به‌صورت زیر تعریف شده است:

$$y(k+1) = 0.3\, y(k) + 0.6\, y(k-1) + g[u(k)] \qquad (۱)$$

تابع $g(u)$ به صورت غیرخطی و ترکیبی از چند موج سینوسی تعریف شده است:

$$g(u) = 0.6 \sin(\pi u) + 0.3 \sin(3\pi u) + 0.1 \sin(5\pi u) \qquad (۲)$$

هدف ما تقریب تابع $g(u)$ با استفاده از سیستم فازی نوع سوگنو مرتبه صفر و استفاده از آن در شبیه‌سازی سیستم دینامیکی است. دو مدل فازی طراحی شده‌اند:

- **مدل اولیه:** آموزش با گرادیان نزولی و مقداردهی اولیه دستی.
- **مدل پیشرفته:** آموزش دسته‌ای با روش کمترین مربعات (Least Squares).

تعریف تابع هدف $\mathbf{g(u)}$

تابع $g(u)$ نقش غیرخطی اصلی را در سیستم ایفا می‌کند و ترکیبی از سه موج سینوسی با فرکانس‌های مختلف است:

$$g(u) = 0.6 \sin(\pi u) + 0.3 \sin(3\pi u) + 0.1 \sin(5\pi u) \qquad (۳)$$

ویژگی‌ها:

- چندفرکانسی، غیرخطی و نوسانی
- مشتق‌پذیر و پیوسته، مناسب برای مدل‌های نرم مانند سیستم‌های فازی
- تعریف‌شده در بازه $u \in [-1, 1]$

ساختار سیستم فازی نوع سوگنو مرتبه صفر

برای تقریب تابع $g(u)$، از یک مدل فازی با قواعد سوگنو مرتبه صفر استفاده شده است:

$$f(u) = \frac{\sum_{l=1}^{M} y^{(l)} \cdot \mu^{(l)}(u)}{\sum_{l=1}^{M} \mu^{(l)}(u)} \qquad (۴)$$

که تابع عضویت $\mu^{(l)}(u)$ به صورت گاوسی تعریف می‌شود:

$$\mu^{(l)}(u) = \exp\left(-\frac{(u - c^{(l)})^2}{(\sigma^{(l)})^2}\right)$$

اگر مقدار ورودی u به مرکز قاعده‌ی $c^{(l)}$ نزدیک باشد، آنگاه خروجی قاعده مقدار $y^{(l)}$ خواهد بود:

$$u \approx c^{(l)} \Rightarrow y = y^{(l)}$$

الگوریتم‌های یادگیری پارامترها

الف) مدل اولیه — گرادیان نزولی آنلاین:

- مقداردهی اولیه مراکز $c^{(l)}$ در بازه $[-1, 1]$ با فواصل مساوی
- مقداردهی خروجی‌ها با مقدار تابع هدف: $g(c^{(l)})$
- استفاده از گرادیان خطا: $e = y_{\text{true}} - f(u)$
- نرخ‌های یادگیری مجزا برای مراکز، پهناها و خروجی‌ها: $\eta_c, \eta_\sigma, \eta_y$

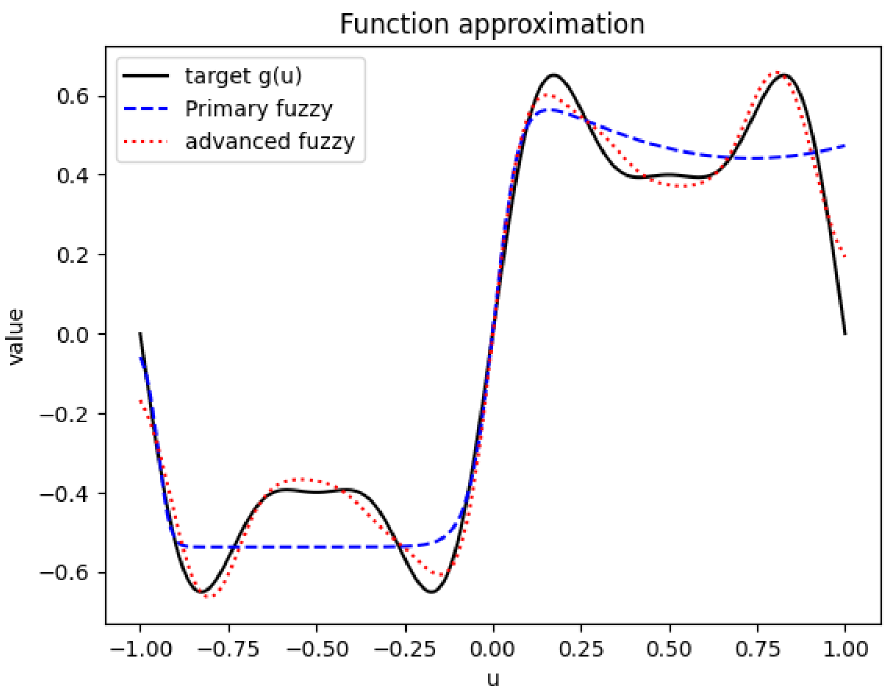
ب) مدل پیشرفته — روش کمترین مربعات دسته‌ای:

- مراکز مشابه مدل اولیه انتخاب می‌شوند.
- تنظیم پهنای تابع گاوسی: فاصله بین مراکز $\sigma = 0.7 \times$
- استخراج ماتریس قدرت آتش قواعد: Λ
- حل معادله کمترین مربعات برای تعیین خروجی قواعد: $\Lambda y = y_{\text{train}}$

ارزیابی گرافیکی مدل‌ها

مقایسه مدل‌های آموزش‌دیده با تابع اصلی در نمودار زیر انجام شده است:

- تابع واقعی $g(u)$: منحنی مشکی
- مدل اولیه: منحنی آبی با خط‌چین
- مدل پیشرفته: منحنی قرمز نقطه‌چین



شکل ۵: مقایسه مدل‌های فازی با تابع واقعی $g(u)$

شبیه‌سازی پاسخ سیستم دینامیکی

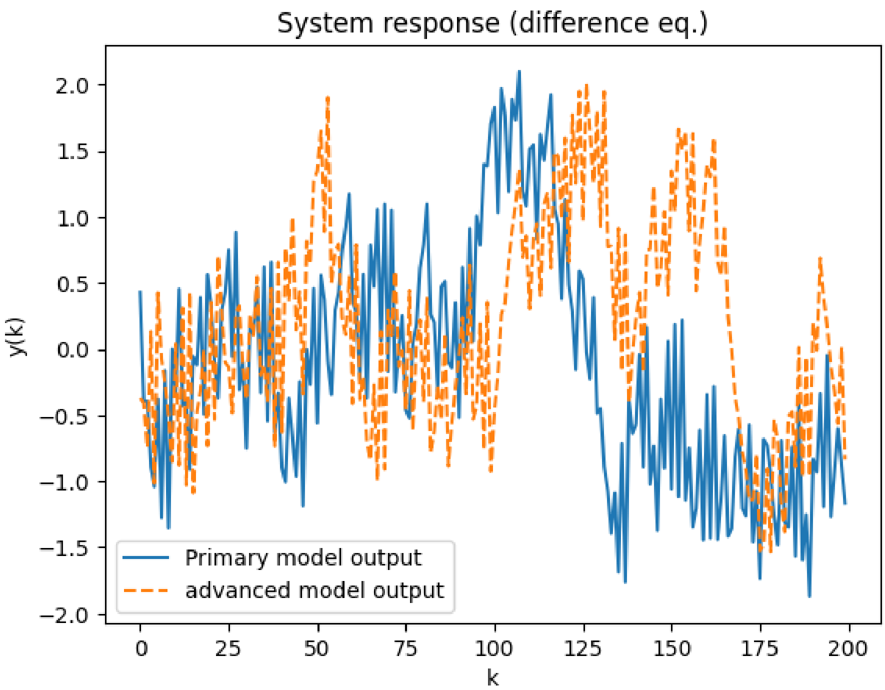
مدل فازی در معادله دینامیکی زیر قرار گرفت:

$$y(k+1) = 0.3\,y(k) + 0.6\,y(k-1) + \hat{g}[u(k)]$$

شرایط اولیه: $y(-1) = y(0) = 0$ ورودی: $u(k) \sim \mathcal{U}[-1,1]$

نتایج:

- مدل اولیه نوسانی و ناپایدارتر است.
- مدل پیشرفته پاسخ نرم‌تر و دقیق‌تری تولید کرده است.



شکل ۶: پاسخ سیستم با مدل‌های فازی اولیه و پیشرفته

کد پایتون

۱. تعریف تابع هدف و تولید داده‌ها

در این بخش، تابع غیرخطی هدف $g(u)$ تعریف می‌شود که باید توسط سیستم فازی تقریب زده شود. همچنین مجموعه داده‌های آموزشی و آزمایشی با توزیع یکنواخت در بازه $[-1, 1]$ تولید می‌شوند.

```python
import numpy as np
import matplotlib.pyplot as plt

def g(u):
    return (0.6 * np.sin(np.pi * u) +
            0.3 * np.sin(3 * np.pi * u) +
            0.1 * np.sin(5 * np.pi * u))

np.random.seed(42)
N_train, N_test = 500, 200
u_train = np.random.uniform(-1, 1, N_train)
u_test  = np.random.uniform(-1, 1, N_test)
y_train, y_test = g(u_train), g(u_test)
```

۲. تعریف ساختار سیستم فازی نوع سوگنو

برای محاسبه‌ی خروجی مدل فازی، از قواعد نوع سوگنو با تابع عضویت گاوسی استفاده می‌شود. این تابع مقدار خروجی $f(u)$ و وزن‌های نرمال‌شده هر قاعده را بازمی‌گرداند.

```python
def fuzzy_forward(u, centers, sigmas, y_rule):
    phi = np.exp(-((u - centers) ** 2) / (sigmas ** 2))
    w_sum = phi.sum()
    if w_sum == 0:
        lam = np.zeros_like(phi)
    else:
        lam = phi / w_sum
    f_u = np.dot(lam, y_rule)
    return f_u, lam
```

۳. آموزش مدل اولیه با گرادیان نزولی

مدل اولیه با روش گرادیان نزولی و به‌صورت آنلاین آموزش داده می‌شود. مراکز قواعد به‌صورت یکنواخت مقداردهی اولیه می‌شوند و پارامترهای مدل در چندین epoch به‌روزرسانی می‌شوند.

```python
def train_Primary(u_data, y_data, M=5, epochs=300,
                  eta_y=0.05, eta_c=0.01, eta_s=0.01):
    centers = np.linspace(-1, 1, M)
    d = centers[1] - centers[0]
    sigmas = np.full(M, d * 0.6)
    y_rule = g(centers)

    for _ in range(epochs):
        for u, yd in zip(u_data, y_data):
            f_u, lam = fuzzy_forward(u, centers, sigmas, y_rule)
            e = yd - f_u
            y_rule += eta_y * e * lam
            diff = u - centers
            common = e * (y_rule - f_u) * lam
            centers += eta_c * common * (2 * diff) / (sigmas ** 2)
            sigmas  += eta_s * common * (2 * diff ** 2) / (sigmas ** 3)
    return centers, sigmas, y_rule
```

۴. آموزش مدل پیشرفته با روش کمترین مربعات

در این روش، پارامترهای مراکز و پهناها ثابت نگه داشته می‌شوند و خروجی قواعد با استفاده از حل دستگاه معادلات کمترین مربعات تعیین می‌گردد.

```python
def train_advanced(u_data, y_data, M=12):
    centers = np.linspace(-1, 1, M)
    d = centers[1] - centers[0]
    sigmas = np.full(M, d * 0.7)
    Phi = np.exp(-((u_data[:, None] - centers[None, :]) ** 2) / (sigmas ** 2))
    Lam = Phi / Phi.sum(axis=1, keepdims=True)
    y_rule, *_ = np.linalg.lstsq(Lam, y_data, rcond=None)
    return centers, sigmas, y_rule
```

۵. آموزش هر دو مدل و ارزیابی عملکرد

هر دو مدل اولیه و پیشرفته آموزش داده می‌شوند. سپس دقت آن‌ها با محاسبه‌ی میانگین مربعات خطا (MSE) بر روی داده‌های تست بررسی می‌گردد.

```python
c_std, s_std, y_std = train_Primary(u_train, y_train)
c_adv, s_adv, y_adv = train_advanced(u_train, y_train)

def evaluate(u_set, y_set, params):
    c, s, y_r = params
    preds = np.array([fuzzy_forward(u, c, s, y_r)[0] for u in u_set])
    mse = np.mean((y_set - preds) ** 2)
    return mse, preds

mse_std, preds_std = evaluate(u_test, y_test, (c_std, s_std, y_std))
mse_adv, preds_adv = evaluate(u_test, y_test, (c_adv, s_adv, y_adv))

print(f"Primary model   MSE on test set : {mse_std:.6e}")
print(f"Advanced model  MSE on test set : {mse_adv:.6e}")
```

۶. ترسیم نمودار تقریب تابع هدف

خروجی هر مدل فازی روی بازه‌ای از ورودی‌ها محاسبه و با تابع واقعی $g(u)$ مقایسه می‌شود تا کیفیت تقریب هر مدل مشخص گردد.

```python
u_grid = np.linspace(-1, 1, 400)
g_grid = g(u_grid)
f_std_grid = np.array([fuzzy_forward(u, c_std, s_std, y_std)[0] for u in u_grid])
f_adv_grid = np.array([fuzzy_forward(u, c_adv, s_adv, y_adv)[0] for u in u_grid])

plt.figure()
plt.plot(u_grid, g_grid, color='black', label="target g(u)")
plt.plot(u_grid, f_std_grid, '--', color='blue', label="Primary fuzzy")
plt.plot(u_grid, f_adv_grid, ':', color='red', label="Advanced fuzzy")
plt.title("Function approximation")
plt.legend()
plt.xlabel("u")
plt.ylabel("value")
plt.show()
```

۷. شبیه‌سازی سیستم دینامیکی تفاضلی

در این مرحله، مدل‌های فازی در یک سیستم دینامیکی تفاضلی جای‌گذاری می‌شوند تا پاسخ سیستم با هر مدل بررسی گردد.

```python
def simulate_system(fuzzy_params, steps=200):
    c, s, y_r = fuzzy_params
    u_seq = np.random.uniform(-1, 1, steps)
    y_seq = np.zeros(steps + 1)
    y_prev = 0.0
    for k in range(steps):
        g_hat, _ = fuzzy_forward(u_seq[k], c, s, y_r)
        y_next = 0.3 * y_seq[k] + 0.6 * y_prev + g_hat
        y_prev, y_seq[k + 1] = y_seq[k], y_next
    return u_seq, y_seq[1:]

u_s, y_s = simulate_system((c_std, s_std, y_std))
u_a, y_a = simulate_system((c_adv, s_adv, y_adv))

plt.figure()
plt.plot(y_s, label="Primary model output")
plt.plot(y_a, "--", label="Advanced model output")
plt.title("System response (difference eq.)")
plt.xlabel("k")
plt.ylabel("y(k)")
plt.legend()
plt.show()
```

۵ پرسش پنج

در این پرسش، هدف استفاده از کران‌های نظری روابط (10.11) و (11.4) برای طراحی دو سیستم فازی است که تابع ثابت

$$g(x_1, x_2) = 1$$

را در دامنه‌ی مربعی

$$U = [-1, 1] \times [-1, 1]$$

با دقت تقریبی $\varepsilon = 0.1$ مدل‌سازی کنند. دو نوع ساختار فازی مورد استفاده قرار گرفته‌اند: یکی با استفاده از کران دو بعدی و دیگری با کران تک‌بعدی.

تحلیل کران‌های نظری و محاسبه گام شبکه

رابطه‌ی کلی کران خطا در سیستم‌های فازی از نوع سوگنو صفرمرتبه، مطابق قضیه ۱۱.۱ کتاب مرجع، به صورت زیر تعریف می‌شود:

$$\|g - f\|_\infty \leq \frac{1}{8}\left(\left\|\frac{\partial^2 g}{\partial x_1^2}\right\|_\infty h_1^2 + \left\|\frac{\partial^2 g}{\partial x_2^2}\right\|_\infty h_2^2\right) \tag{۱۱.۴}$$

و رابطه‌ی ساده‌شده‌ی یک‌بعدی آن (برای هر متغیر ورودی جداگانه) چنین است:

$$\|g - L_1 g\|_\infty \leq \frac{1}{8}\left(e_{i+1}^{(1)} - e_i^{(1)}\right)^2 \left\|\frac{\partial^2 g}{\partial x_1^2}\right\|_\infty \tag{۱۱.۴}$$

در این مسئله، چون تابع $g(x_1, x_2) = 1$ یک تابع ثابت است، داریم:

$$\frac{\partial g}{\partial x_1} = 0, \quad \frac{\partial g}{\partial x_2} = 0, \quad \frac{\partial^2 g}{\partial x_1^2} = 0, \quad \frac{\partial^2 g}{\partial x_2^2} = 0$$

بنابراین، طبق هر دو رابطه:

$$\|g - f\|_\infty = 0 < \varepsilon$$

این یعنی هیچ محدودیتی روی گام شبکه نداریم و می‌توانیم حتی با فقط یک بخش در هر بعد (یعنی یک تابع عضویت در هر بعد) به دقت مطلوب برسیم. چون:

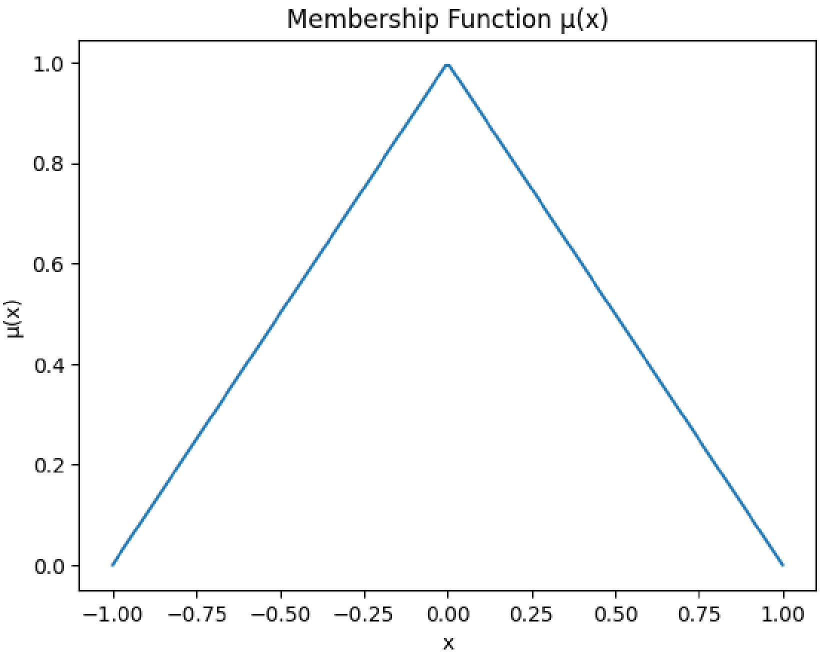
$$h_1 = h_2 = \text{طول دامنه} = 2 \quad \Rightarrow \quad N_1 = N_2 = 1$$

تابع عضویت مثلثی

برای تقریب، از تابع عضویت مثلثی متقارن استفاده می‌کنیم:

$$\mu(x) = \max\{0, 1 - |x|\}, \quad x \in [-1, 1] \tag{۵}$$

این تابع روی کل دامنه پوشش ایجاد کرده و مقدار ماکزیمم در مرکز (صفر) دارد. از آن‌جا که تنها یک تابع عضویت در هر متغیر ورودی وجود دارد، تمامی نقاط در دامنه با یک درجه تعلق مثبت ارزیابی می‌شوند.



شکل ۷: تابع عضویت مثلثی برای هر ورودی

طراحی قواعد فازی

با وجود تنها یک تابع عضویت در هر بعد، فقط یک قاعده فازی تعریف می‌شود:

اگر x_1 در مجموعه‌ی A و x_2 در مجموعه‌ی B باشد، آنگاه $y = 1$.

که در قالب رسمی به شکل زیر است:

$$\text{If} x_1 \in A \text{ and } x_2 \in B, \quad \text{Then} y = 1$$

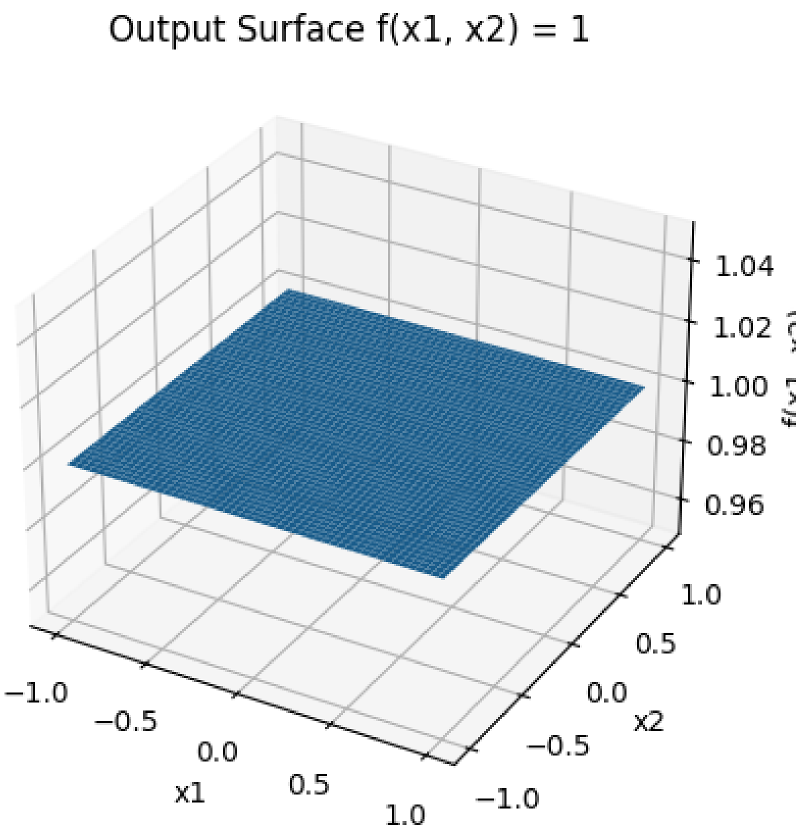
فازی‌زدایی با مرکز میانگین

فازی‌زدایی خروجی به روش «مرکز میانگین» یا Average of Center به صورت زیر محاسبه می‌شود:

$$f(x_1, x_2) = \frac{\sum_{i,j} \mu_{A_i}(x_1)\mu_{B_j}(x_2)y_{ij}}{\sum_{i,j} \mu_{A_i}(x_1)\mu_{B_j}(x_2)}$$

در این پروژه:
- فقط یک قاعده وجود دارد - مقدار $y_{11} = 1$
پس داریم:

$$f(x_1, x_2) = \frac{\mu(x_1)\mu(x_2) \cdot 1}{\mu(x_1)\mu(x_2)} = 1$$



شکل ۸: سطح خروجی سیستم فازی طراحی‌شده

نتیجه‌گیری و مقایسه نظری

- در هر دو روش (۱۰.۱۱) و (۱۱.۴)، سیستم فازی با تنها یک قاعده به دقت کامل رسیده است.
- به دلیل صفر بودن مشتقات مرتبه دوم، کران نظری خطا نیز صفر است.
- تابع عضویت مثلثی ساده کافی بود تا کل دامنه را پوشش دهد.
- خروجی مدل در تمام نقاط برابر با تابع اصلی g است.
- سیستم طراحی‌شده بسیار فشرده و ساده است، ولی دقت آن حداکثری است.

کد پایتون

۱. تعریف دامنه و تابع عضویت

در این بخش، یک دامنه یکنواخت در بازه $[-1, 1]$ تعریف می‌شود و تابع عضویت مثلثی $\mu(x) = \max(0, 1 - |x|)$ روی آن محاسبه می‌گردد.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

x = np.linspace(-1, 1, 200)
mu = np.maximum(0, 1 - np.abs(x))
```

۲. ترسیم تابع عضویت

مقدار تابع عضویت برای نقاط مختلف دامنه رسم می‌شود تا شکل تابع مثلثی مورد استفاده در سیستم فازی نمایش داده شود.

```
plt.figure()
plt.plot(x, mu)
plt.xlabel('x')
plt.ylabel(' (x)')
plt.title('Membership Function  (x)')
plt.show()
```

۳. ترسیم سطح خروجی سیستم فازی

تابع خروجی سیستم فازی $f(x_1, x_2) = 1$ روی دامنه دوبعدی $x_1, x_2 \in [-1, 1]$ تعریف و به صورت سه‌بعدی ترسیم می‌شود.

```
X1, X2 = np.meshgrid(x, x)
Z = np.ones_like(X1)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X1, X2, Z)
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('f(x1, x2)')
ax.set_title('Output Surface f(x1, x2) = 1')
plt.show()
```

۶ پرسش شش

هدف

هدف از این پرسش طراحی کنترلر فازی برای هدایت ربات از نقطه مبدا به مقصد است.

- ورودی‌های کنترلر: زاویه و فاصله ربات نسبت به نقطه مقصد
- خروجی‌های کنترلر: سرعت خطی و سرعت زاویه‌ای ربات

در این سوال شرایط اولیه و نهایی برای ربات تعیین شده همچنین مقدار خروجی‌های کنترلر محدود است. این محدودیت در توابع تعلق اعمال شده است.

طراحی سیستم فازی

۱. توابع تعلق

برای متغیرهای مربوط به زاویه ۵ تابع تعلق و برای متغیرهای مربوط به فاصله ۳ تابع تعلق از نوع مثلثی تعریف شده است.

تعریف متغیرهای فازی:

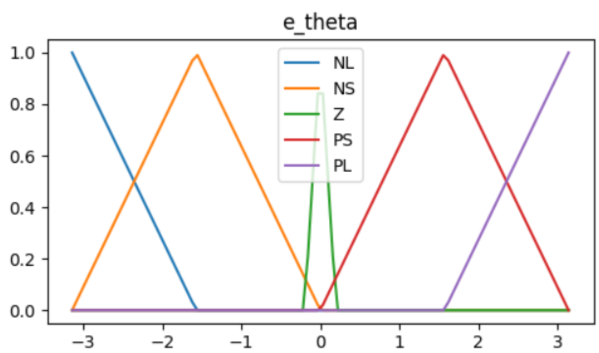
```
# Define fuzzy variables
e_theta = ctrl.Antecedent(np.linspace(-np.pi, np.pi, 100), 'e_theta')
ed = ctrl.Antecedent(np.linspace(0, 5, 100), 'ed')
omega = ctrl.Consequent(np.linspace(-1, 1, 100), 'omega')
v = ctrl.Consequent(np.linspace(0, 0.5, 100), 'v')
```

متغیرهای ورودی:

متغیر ورودی زاویه (تتا):

```
e_theta['NL'] = fuzz.trimf(e_theta.universe, [-np.pi, -np.pi, -np.pi/2])
e_theta['NS'] = fuzz.trimf(e_theta.universe, [-np.pi, -np.pi/2, 0])
e_theta['Z'] = fuzz.trimf(e_theta.universe, [-0.2, 0, 0.2])
e_theta['PS'] = fuzz.trimf(e_theta.universe, [0, np.pi/2, np.pi])
e_theta['PL'] = fuzz.trimf(e_theta.universe, [np.pi/2, np.pi, np.pi])
```

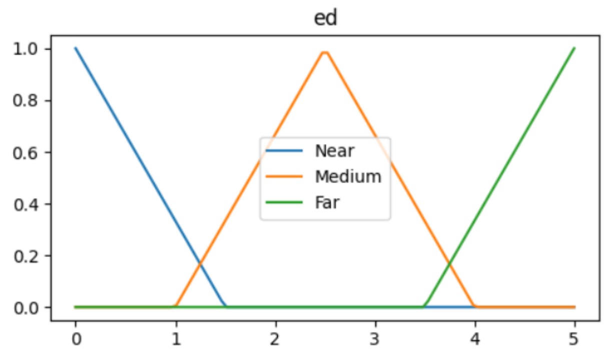
برای زاویه‌ای که ربات نسبت به مسیر دارد، ۵ تابع تعلق تعریف شد به‌ترتیب برای نشان دادن: منفی دور، منفی نزدیک، حدوداً صفر، مثبت نزدیک، مثبت دور



متغیر ورودی فاصله :(d)

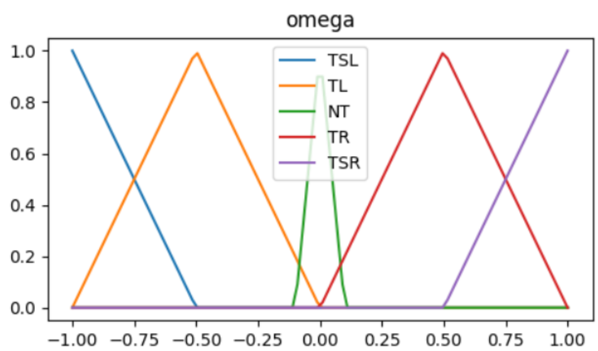
```
ed['Near'] = fuzz.trimf(ed.universe, [0, 0, 1.5])
ed['Medium'] = fuzz.trimf(ed.universe, [1, 2.5, 4])
ed['Far'] = fuzz.trimf(ed.universe, [3.5, 5, 5])
```

برای متغیر فاصله، سه تابع برای نشان دادن فاصله نزدیک، متوسط و دور در نظر گرفته شد.



متغیر خروجی سرعت زاویه‌ای: توابع مشابه تعریف توابع مربوط به زاویه است و حدود تعریف شده در سوال برای این متغیر در نظر گرفته شده است. (چرخش تند چپ، چرخش چپ، بدون چرخش، چرخش راست، چرخش تند راست)

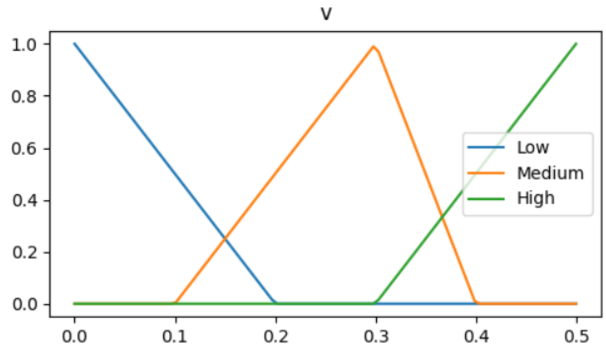
```
omega['TSL'] = fuzz.trimf(omega.universe, [-1, -1, -0.5])
omega['TL'] = fuzz.trimf(omega.universe, [-1, -0.5, 0])
omega['NT'] = fuzz.trimf(omega.universe, [-0.1, 0, 0.1])
omega['TR'] = fuzz.trimf(omega.universe, [0, 0.5, 1])
omega['TSR'] = fuzz.trimf(omega.universe, [0.5, 1, 1])
```



$$-1 \leq \omega \leq 1 \quad (rad/s)$$

متغیر خروجی سرعت خطی: مشابه توابع فاصله تعریف شده و حدود سوال در آن اعمال شده است. (کم، متوسط و زیاد)

```
v['Low'] = fuzz.trimf(v.universe, [0, 0, 0.2])
v['Medium'] = fuzz.trimf(v.universe, [0.1, 0.3, 0.4])
v['High'] = fuzz.trimf(v.universe, [0.3, 0.5, 0.5])
```



$$0 \leq v \leq 0.5 \quad (m/s)$$

۲. قوانین فازی

ده قانون تعریف شد:

- right” sudden “turn then “far” is distance and large” “positive is angle if
- right” sudden “turn then “far” is distance and large” “negative is angle if
- left” “turn then “medium” is distance and small” “negative is angle if
- right” “turn then “medium” is distance and small” “positive is angle if
- turn” not “do then “near” is distance and “zero” is angle if
- right” “turn then “near” is distance and large” “positive is angle if
- left” “turn then “near” is distance and large” “negative is angle if
- “high” v then “far” is distance if
- “medium” v then “medium” is distance if
- “low” v then “near” is distance if

```
rules = [
    ctrl.Rule(e_theta['PL'] & ed['Far'], omega['TSR']),
    ctrl.Rule(e_theta['NL'] & ed['Far'], omega['TSL']),
    ctrl.Rule(e_theta['NS'] & ed['Medium'], omega['TL']),
    ctrl.Rule(e_theta['PS'] & ed['Medium'], omega['TR']),
    ctrl.Rule(e_theta['Z'] & ed['Near'], omega['NT']),
    ctrl.Rule(e_theta['PL'] & ed['Near'], omega['TR']),
    ctrl.Rule(e_theta['NL'] & ed['Near'], omega['TL']),
    ctrl.Rule(ed['Far'], v['High']),
    ctrl.Rule(ed['Medium'], v['Medium']),
    ctrl.Rule(ed['Near'], v['Low']),
]
```

۳. ایجاد سیستم فازی

```
control_system = ctrl.ControlSystem(rules)
sim = ctrl.ControlSystemSimulation(control_system)
```

تعریف متغیرهای سیستم:

```
x, y, theta = 0.0, 0.0, 0.0
xg, yg, thetag = 2.0, 2.0, np.pi / 2 + 1.57
dt = 0.1
tolerance_dist = 0.01
tolerance_theta = 0.1
```

تعریف لیست‌ها برای ذخیره متغیرها در طول شبیه‌سازی:

```
trajectory = [(x, y)]
v_list = []
omega_list = []
e_list = []
etheta_list = []
theta_list = [theta]
```

حلقه شبیه‌سازی:

```
for _ in range(300):
    dx = xg - x
    dy = yg - y
    dist = np.hypot(dx, dy)
    desired_theta = np.arctan2(dy, dx)
    e_th = np.arctan2(np.sin(desired_theta - theta), np.cos(desired_theta - theta))

    sim.input['e_theta'] = e_th
    sim.input['ed'] = dist
    sim.compute()

    v_out = sim.output.get('v', 0)
    omega_out = sim.output.get('omega', 0)
```

ابتدا متغیرهای ورودی کنترلر با توجه به موقعیت فعلی x, y محاسبه می‌شوند.

```
dx = xg - x
dy = yg - y
dist = np.hypot(dx, dy)
desired_theta = np.arctan2(dy, dx)
e_th = np.arctan2(np.sin(desired_theta - theta), np.cos(desired_theta - theta))
```

مقادیر محاسبه‌شده به عنوان ورودی کنترلر داده شده و مقادیر سرعت خطی و زاویه‌ای را خروجی گرفته شد:

```python
sim.input['e_theta'] = e_th
sim.input['ed'] = dist
sim.compute()

v_out = sim.output.get('v', 0)
omega_out = sim.output.get('omega', 0)
```

همانطور که در سوال گفته شده زاویه نهایی مشخصی برای ربات می‌خواهیم، در اینجا مشخص می‌کنیم تنظیم نهایی زاویه در فاصله بسیار نزدیک انجام شود.

```python
if dist < tolerance_dist:
    desired_theta = thetag
    e_th = np.arctan2(np.sin(thetag - theta), np.cos(thetag - theta))
    sim.input['e_theta'] = e_th
    sim.input['ed'] = 0.01  # avoid full zero
    sim.compute()
    v_out = 0
    omega_out = sim.output.get('omega', 0)
    if abs(e_th) < tolerance_theta:
        break
```

ذخیره مقادیر نهایی پارامترهای ورودی خروجی:

```python
v_list.append(v_out)
omega_list.append(omega_out)
e_list.append(dist)
etheta_list.append(e_th)
```

آپدیت مقادیر موقعیت و زاویه ربات، ذخیره موقعیت و زاویه:

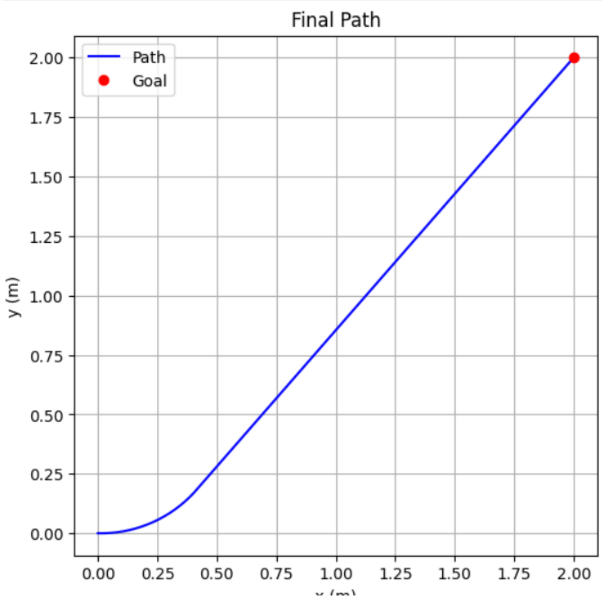
```python
x += v_out * np.cos(theta) * dt
y += v_out * np.sin(theta) * dt
theta += omega_out * dt
theta = np.arctan2(np.sin(theta), np.cos(theta))
```

۴. نمودار خروجی نهایی

```python
trajectory.append((x, y))
theta_list.append(theta)
```

```python
trajectory = np.array(trajectory)
# Final trajectory plot
plt.figure(figsize=(6, 6))
plt.plot(trajectory[:, 0], trajectory[:, 1], 'b-', label='Path')
plt.plot(xg, yg, 'ro', label='Goal')
plt.xlabel('x (m)')
plt.ylabel('y (m)')
plt.legend()
plt.title('Final Path')
plt.axis('equal')
plt.grid(True)
plt.show()
```

(*) مسیر ربات با انیمیشن هم نمایش داده شده که در colab قابل مشاهده است.



۷ پرسش هفت

- **بارگذاری و نمایش دیتاست در متلب:**

```
data = readtable('C:\Users\USER\Downloads\diabetes.csv')
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
1	6	148	72	35	0	33.6000	0.6270	50	1
2	1	85	66	29	0	26.6000	0.3510	31	0
3	8	183	64	0	0	23.3000	0.6720	32	1
4	1	89	66	23	94	28.1000	0.1670	21	0
5	0	137	40	35	168	43.1000	2.2880	33	1
6	5	116	74	0	0	25.6000	0.2010	30	0
7	3	78	50	32	88	31	0.2480	26	1
8	10	115	0	0	0	35.3000	0.1340	29	0
9	2	197	70	45	543	30.5000	0.1580	53	1
10	8	125	96	0	0	0	0.2320	54	1
11	4	110	92	0	0	37.6000	0.1910	30	0
12	10	168	74	0	0	38	0.5370	34	1

دیتاست شامل ۸ فیچر ورودی: سابقه بارداری، سطح گلوکز، فشار خون، سطح انسولین، BMI، سابقه دیابت در خانواده و سن فرد است. ستون آخر وضعیت دیابت بیمار را نشان می‌دهد.

- **جداسازی فیچرها و نرمال‌سازی ورودی‌ها:**

```
X = table2array(data(:, 1:end-1));
Y = table2array(data(:, end));
X = normalize(X)
```

```
X = 768×8
     0.6395    0.8478    0.1495    0.9067   -0.6924  ...
    -0.8443   -1.1227   -0.1604    0.5306   -0.6924
     1.2331    1.9425   -0.2638   -1.2874   -0.6924
    -0.8443   -0.9976   -0.1604    0.1544    0.1232
    -1.1411    0.5037   -1.5037    0.9067    0.7653
     0.3428   -0.1531    0.2529   -1.2874   -0.6924
    -0.2508   -1.3416   -0.9871    0.7186    0.0712
     1.8266   -0.1844   -3.5703   -1.2874   -0.6924
    -0.5476    2.3803    0.0462    1.5336    4.0193
     1.2331    0.1284    1.3895   -1.2874   -0.6924
        :
```

- **تقسیم داده به TEST و TRAIN:**

```
cv = cvpartition(size(X,1), 'HoldOut', 0.3);
idx = cv.test;
XTrain = X(~idx, :);
YTrain = Y(~idx, :);
XTest = X(idx, :);
YTest = Y(idx, :);
```

- **ایجاد مدل فازی برای ورودی‌های سؤال:**

```
fis2 = genfis2(XTrain, YTrain, 0.4);
```

عدد پارامتر آخر مربوط به شعاع خوشه‌بندی است. با کاهش این عدد، خوشه‌های ورودی با شعاع کوچک‌تری تقسیم‌بندی می‌شوند، در نتیجه قوانین بیشتری ساخته می‌شوند که دقت مدل را بالا می‌برد، اما احتمال overfit هم وجود دارد. با افزایش شعاع، قوانین کمتری تولید می‌شوند و مدل ممکن است بیش از حد ساده شود.

- **آموزش مدل ANFIS (۵۰ epoch):**

```
opt_anfis = anfisOptions('InitialFIS', fis, 'EpochNumber', 50);
opt_anfis.ValidationData = data_anfis;
[trainedFis, trainError] = anfis(data_anfis, opt_anfis);
```

نتیجه:

```
ANFIS info:
        Number of nodes: 155
        Number of linear parameters: 72
        Number of nonlinear parameters: 128
        Total number of parameters: 200
        Number of training data pairs: 538
        Number of checking data pairs: 0
        Number of fuzzy rules: 8
```

۸ قانون ایجاد شد.

- **بررسی دقت مدل:**

```
yPred2 = evalfis(XTest, anfis_model2);
yPred2 = round(yPred2);
accuracy2 = sum(yPred2 == YTest) / length(YTest);
fprintf('ANFIS (genfis2) Accuracy: %.2f%%\n', accuracy2 * 100);
```

```
ANFIS (genfis2) Accuracy: 72.17%
```

دقت مدل ۷۲ درصد

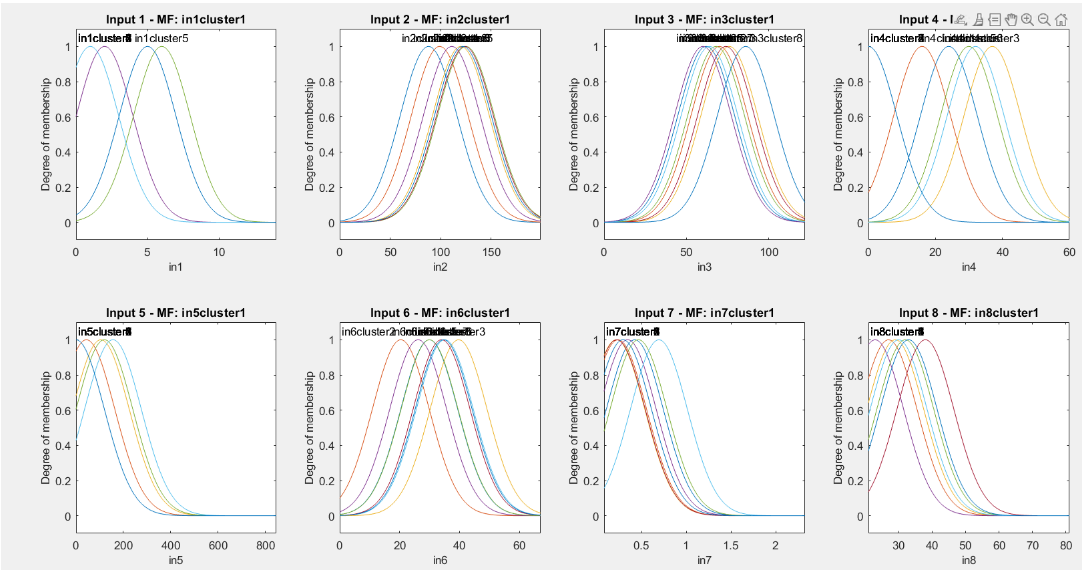
- **نمایش قوانین ایجاد شده توسط ANFIS:**

```
anfis_model2 = anfis([XTrain YTrain], fis2, epoch_n);
showrule(anfis_model2)
```

'1. If (in1 is in1cluster1) and ... then (out1 is out1cluster1) (1)'

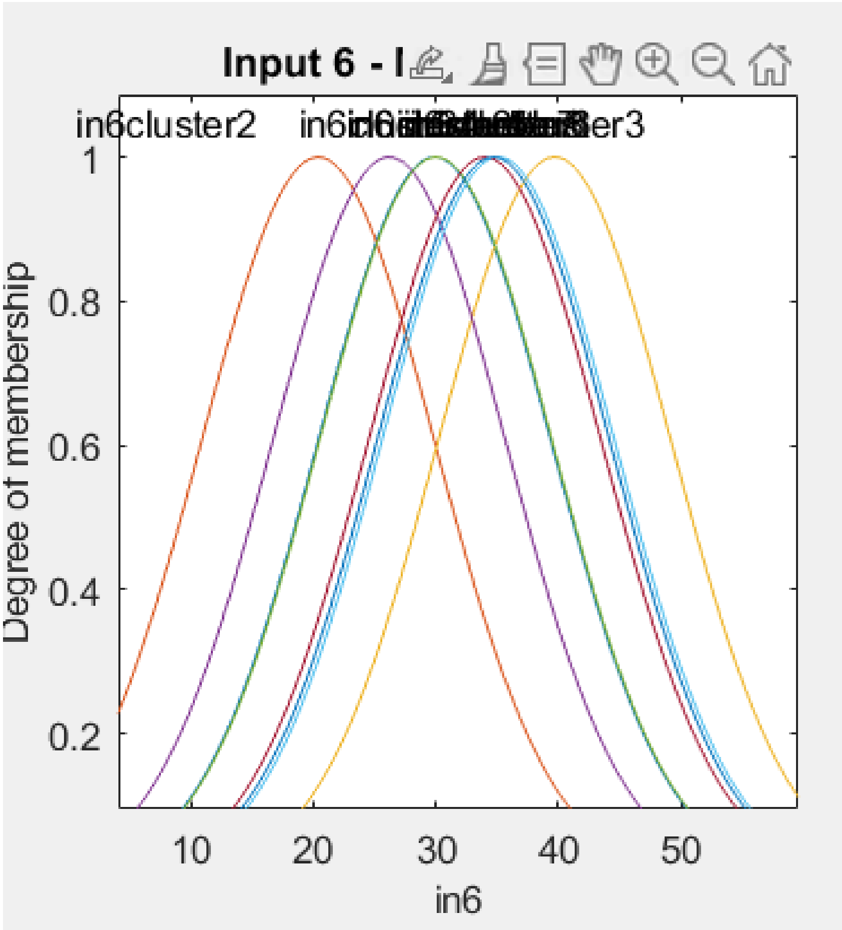
'2. If (in1 is in1cluster2) and ... then (out1 is out1cluster2) (1)'

نمایش قواعد با تفسیر: Rule ۱: IF (Pregnancies is Low) AND ... THEN Output is No Diabetes
Rule ۲: IF (Pregnancies is Medium) AND ... THEN Output is Diabetes



برخی ورودی‌ها مانند ورودی ۶ (BMI) در تمام قوانین حضور دارند، در حالی که برخی مانند ورودی ۱ (سابقه بارداری) در قوانین کمتری دیده می‌شوند.

- **تحلیل قوانین:**



بررسی ورودی ۶ :(BMI) در تمام ۸ قانون استفاده شده و در ۷ قانون در مقادیر بالای ۲۵ بیشترین خروجی را دارد (BMI > ۲۵ نشان‌دهنده اضافه‌وزن است).

UNDERWEIGHT	NORMAL WEIGHT	OVERWEIGHT	OBESE	MORBIDLY OBESE
< 18.5	18.5 - 24.9	25.0 - 29.9	30.0 - 39.9	≥ 40.0
LOW	IDEAL	HIGH	VERY HIGH	EKTREMELY HIGH

سن در قوانین تأثیر چندانی ندارد، احتمالاً به‌دلیل تنوع کم سنی در دیتاست.

- **پیاده‌سازی با :MLP بارگذاری و نرمال‌سازی دیتاست:**

```
% Load and preprocess data
data = readtable('C:\Users\USER\Downloads\diabetes.csv');
X = table2array(data(:, 1:end-1));
Y = table2array(data(:, end));
X = normalize(X, 'range');
```

تقسیم داده به TEST و TRAIN

```
% Train/test split
cv = cvpartition(size(X,1), 'HoldOut', 0.3);
idx = cv.test;

XTrain = X(~idx, :)';
YTrain = Y(~idx)';
XTest = X(idx, :)';
YTest = Y(idx)';
```

تعریف MLP

```
% Define and train MLP
net = patternnet([8 10]);
net.trainFcn = 'trainlm';
net.performFcn = 'crossentropy';
```

شبکه دارای دو لایه پنهان با اندازه ۸ و ۱۰ و تابع هزینه CrossEntropy است.

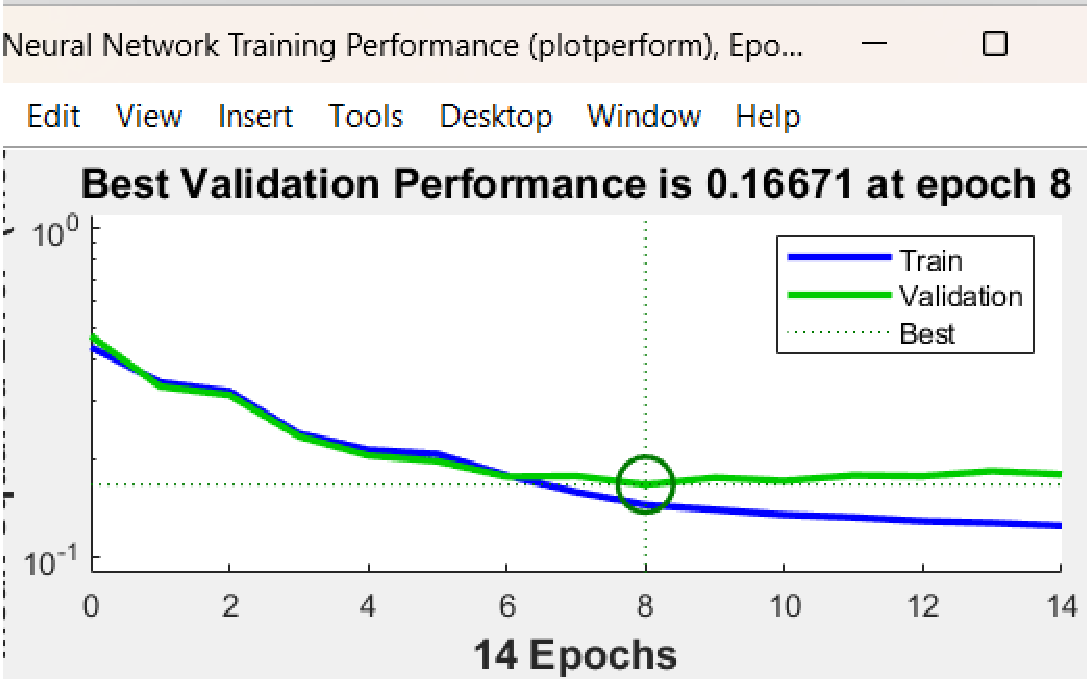
```
net.divideParam.trainRatio = 0.8;
net.divideParam.valRatio = 0.2;
net.divideParam.testRatio = 0.0;

[net, tr] = train(net, XTrain, YTrain_categorical);
```

استفاده از Early Stopping با تقسیم داده به Train/Validation و آموزش در ۱۴ epoch با دقت ۸۰٪ متوقف شد.

Training Progress

Unit	Initial Value	Stopped Value	Target Value
Epoch	0	14	1000
Elapsed Time	-	00:00:01	-
Performance	0.436	0.125	0
Gradient	0.562	0.032	1e-07
Mu	0.001	0.0001	1e+10
Validation Checks	0	6	6




```
% Predict on test data
YTest_output = net(XTest);
YTest_pred = vec2ind(YTest_output) - 1;

% Accuracy
accuracy = sum(YTest_pred == YTest) / length(YTest);
fprintf('MLP Accuracy: %.2f%%\n', accuracy * 100);
```

- **مقایسه MLP و ANFIS**

```
MLP Accuracy: 80.00%
```

– قوانین فازی و توابع تعلق در مدل ANFIS از نظر پزشکی قابل تفسیر هستند.

– سرعت آموزش MLP بیشتر است؛ این تفاوت با افزایش قوانین در مدل فازی افزایش می‌یابد.