پروژه درس فازی

مرحله 1 و 2

استاد براتی

شبیه سازی یک تابع به روش ونگ مندل و بهینه سازی از الگوریتم SA

پویان رضائی

شماره دانشجویی 403149294

راهنمای اجرای پروژه

1-لطفا ابتدا فولدر پروژه در محیط برنامه نویسی باز شود

2- سپس فایل fuzzy_model.py اجرا شود (اگر با خطایی مواجه شدید در اجرای برنامه اول فایل data_set.py اجرا شود سپس فایل fuzzy_model.py)

3- تمام فایل های دیتاست های آموزشی و تمرینی و عکس های نمودار های تشکیل شده ابتدا نمایش داده میشوند (باید تک تک نمودار ها بسته شوند پس از نمایش تا برنامه ادامه پیدا کند) سپس در فولدر ذخیره می شوند پس از اجرا

آماده سازی داده (Data preparation):

فایل data_set.py:

```
 import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def generate_data_and_plot(num_points, filename, title, is_training=True):
    if is_training:
        x_values = np.linspace(-5, 5, num_points)
        x1, x2 = np.meshgrid(x_values, x_values)

        output = x1**2 + x2**2

        data = pd.DataFrame({'x1': x1.flatten(), 'x2': x2.flatten(), 'F(x1, x2)':
output.flatten()})

    else:
        x1_values = np.random.uniform(-5, 5, num_points)
        x2_values = np.random.uniform(-5, 5, num_points)
        output = x1_values**2 + x2_values**2

        data = pd.DataFrame({'x1': x1_values, 'x2': x2_values, 'F(x1, x2)':
output})

    data.to_csv(filename, index=False)

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    if is_training:
        ax.plot_surface(x1, x2, output, cmap='viridis')
    else:
        ax.scatter(data['x1'], data['x2'], data['F(x1, x2)'], color='red',
marker='o')

    ax.set_xlabel('X Axis')
    ax.set_ylabel('Y Axis')
    ax.set_zlabel('Z Axis')
    plt.title(f'{title} 3D Surface Plot' if is_training else f'{title} Test Data
Scatter Plot')
```

```
    plt.savefig(f'{title} {"3D Surface Plot" if is_training else "Test Data Scatter Plot"}.jpg')

    plt.show()

generate_data_and_plot(41, "training_data.csv", "Training", is_training=True)

generate_data_and_plot(168, "test_data.csv", "Test", is_training=False)
```

در این فایل مرحله آماده سازی دیتا اتفاق می افتد که با استفاده از تابع generate_data_and_plot که دارای 4 ورودی میباشد یک ورودی num_points و filename و title و is_training . متغیر is_training که مشخص میکند که داده ساخته شده برای آموزش هست یا برای تست.

Title برای تیتر پلات های رسم شده استفاده می شود

Filename برای مشخص کردن اسم فایل csv ذخیره شده

Num_points برای داده های تمرینی نشان دهنده مقدار داده ها برای x1 و x2 هست یعنی 41 که ترکیب آن ها 1681 ردیف میشود و برای داده های تست نشان دهنده مقدار داده مورد نیاز است که به صورت همسان پخش شده اند یعنی 168

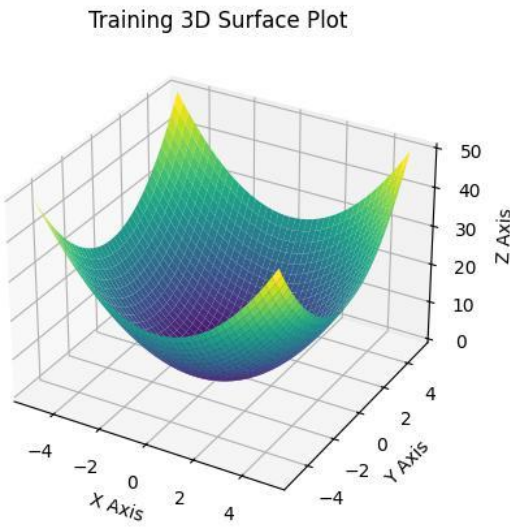
در این تابع از یک if بلاک استفاده شده است که اگر داده های آموزشی باشد با استفاده از تابع کتابخانه numpy یعنی linespace در بازه داده شده دیتا هایی با تعداد داده شده با فواصل یکسان ساخته می شود و با استفاده از تابع meshgrid ترکیب تمام این داده ها به صورت ماتریس ساخته می شود و با حساب کردن مجموع توان 2 متغیر های x_1 و x_2 خروجی آن ها برای مقایسه با داده های پیش بینی شده ساخته می شود و سپس با استفاده از کتابخانه pandas داده هارا به صورت دیتافریم تبدیل میکنیم و برای اینکار داده ها حتما باید تبدیل به ارایه تک بعدی بشوند به همین دلیل از تابع flatten استفاده شده است.

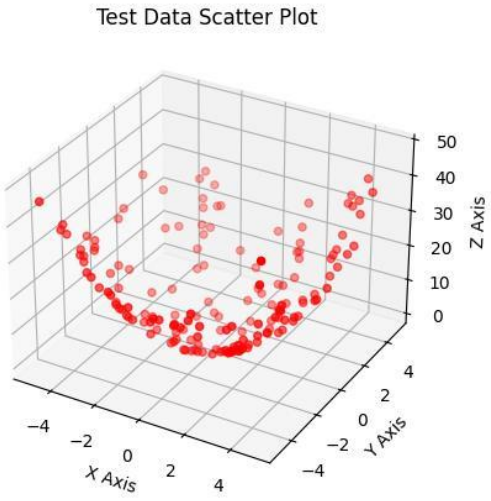
اگر برای درست کردن داده ای تست بخواهیم از این تابع استفاده کنیم بلاک else رخ میدهد که در آن دو متغیر اولیه با استفاده از numpy مقادیری رندوم و یونیفورم به تعداد 168 ساخته می شود و دوباره مجموع توان 2 آن ها محاسبه شده و در اینجا آرایه ها تک بعدی هستند و به صورت مستقیم بدون نیاز به تبدیل آن ها را تبدیل به دیتافریم کرده و خارج از بلاک های شرطی دیتا ها به صورت csv ذخیره می شوند.

برای تفهیم بیشتر از دیتا ها پلات هایی کشیده می شود که بتوان پخش داده ها را مشاهده کرد

(این بخش صرفا برای آماده سازی csv ها و پلات هایی برای فهمیدن بهتر ساختار دیتا های train و test)

نمودار های خروجی:





خلاصه پیاده سازی:

فایل fuzzy_model.py:

این پروژه با استفاده از زبان برنامه نویسی پایتون (python) پیاده سازی شده و از کتابخانه های:

Matplotlib

برای کشیدن پلات های مورد نیاز

Pandas

برای ذخیره دیتا به صورت دیتافریم و فایل (csv) (تنها برای بخش data preparation استفاده شده)

Numpy

برای انجام عملیات های ریاضی و ماتریسی استفاده شده (امکان پردازش سریع تر میدهد)

مدل به صورت یک کلاس شامل چند متغیر و تابع پیاده سازی شده است

```
class FuzzyModel():
```

Constructor این کلاس شامل 4 متغیر می شود ، متغیر اول فازی ست های تعریفی برای x1_data و x2_data و y_data و مرکز فازی ست خروجی (output_centers) و ورودی constructor شامل دیتا آموزشی و تعداد فازی ست های مورد نیاز که به صورت پیش فرض 7 قرار گرفته

```
    def __init__(self, training_data, num_fuzzy_sets=7):
        x1_data = [x[0] for x in training_data]
        x2_data = [x[1] for x in training_data]
        y_data = [x[2] for x in training_data]

        self.input_fuzzy_sets_x1 = self.calculate_fuzzy_sets(x1_data,
num_fuzzy_sets)
        self.input_fuzzy_sets_x2 = self.calculate_fuzzy_sets(x2_data,
num_fuzzy_sets)
        self.output_fuzzy_sets = self.calculate_fuzzy_sets(y_data,
num_fuzzy_sets)

        self.output_centers = {label: (a + b + c) / 3 for label, (a, b, c) in
self.output_fuzzy_sets.items()}
```

بعد با استفاده از 3 تعریف شده و تابع calculate_fuzzy_sets (در بخش تابع ها طرز کار این تابع توضیح داده شده است) فازی ست های مورد نیاز برای دو ورودی و خروجی محاسبه می شود.

و در متغیر output_centers مرکز خروجی ها یا همان centroid ها برای defuzzification از روش center average استفاده می شود ذخیره می شود.

تابع ها

تابع calculate_fuzzy_sets

```
    def calculate_fuzzy_sets(self, data, num_sets=7):
        min_val = np.min(data)
        max_val = np.max(data)
        sets = ["NB","NM","NS","ZR","PS","PM","PB"]

        step = (max_val - min_val) / (num_sets - 1)

        fuzzy_sets = {}
        for i in range(len(sets)):
            label = f"{sets[i]}"
            if i == 0:
```

```
                fuzzy_sets[label] = (min_val, min_val, min_val + step)
            elif i == num_sets - 1:
                fuzzy_sets[label] = (max_val - step, max_val, max_val)
            else:
                fuzzy_sets[label] = (min_val + (i - 1) * step, min_val + i *
step, min_val + (i + 1) * step)

        return fuzzy_sets
```

در این تابع دو ورودی دارد که یکی data هست که ورودی داده های برای انجام پروسه تعریف فازی ست ها هست و دیگر num_sets هست که مقدار فازی ست های مورد نیاز است در این تابع در ابتدا مقدار کمینه و بیشینه داده ها حساب میشود سپس مقدار فاصله (step) ست ها محاسبه میشود.

و لیستی تعریف شده برای ذخیره کردن نام های فازی ست ها.

در ادامه یک دیکشنری تعریف میکنیم به نام fuzzy_sets و با استفاده از یک حلقه به تعداد فازی ست های مورد نیاز یعنی 7 لوپ انجام می شود که در این لوپ چک میشود که اگر فازی ست اول (در این پروژه یعنی NB) مقدار شروع فازی ست با مینیمم فازی ست برابر است و به اندازه یک step پایان فازی ست نسبت به مینیمم فاصله دارد و جلو تر است و اگر فازی ست آخر باشد مقدار پایانی فازی ست با ماکزیمم فازی ست آخر (در این پروژه یعنی فازی ست PB) برابر است و مقدار شروع فازی ست برابر با یک step عقب تر از فازی ست آخر هست. و در غیر این صورت یعنی اگر فازی ست اول یا آخر نباشد به اندازه step ضربدر i (یعنی iterator) به اضافه مقدار منیمم پیدا شده میشود مقدار ماکزیمم فازی ست پیدا می شود و یک step قبل تر از آن میشود شروع فازی ست و یک step جلوتر میشود اتمام فازی ست. و در اخر دیکشنری تشکیل شده برگردانده میشود.

تابع Triangular MF:

```
    def triangular_mf(self, x, a, b, c):
        return np.maximum(0, np.minimum(
            (x - a) / (b - a) if b != a else (1 if x >= a else 0),
            (c - x) / (c - b) if b != c else (1 if x <= c else 0)
        ))
```

تابع بعدی تابع ممبرشیپ مثلثاتی است که به صورت زیر نوشته شده که اگر x منفی شود 0 برمیگردد و اگر بیشتر از بزرگتر از c یا همان پایان فازی ست باشد 0 برمیگردانند در غیر این صورت از فرمول گفته شده در

تابع های تعریف شده در Triangular MF استفاده میکنیم. در این تابع به علت دو فازی ست NB و PB که مثلث کامل نیستن شرایط آن ها هم در نظر گرفته شده.

تابع fuzzify_input:

```
    def fuzzify_input(self, x, fuzzy_sets):
        membership_degrees = {}
        for label, (a, b, c) in fuzzy_sets.items():
            membership_degrees[label] = self.triangular_mf(x, a, b, c)
        return membership_degrees
```

در این تابع ورودی ها x و fuzzy_sets هستند، که بر اساس فازی ست های تعریف شده آرگومان های آن به تابع ممبرشیپ مثلثاتی پاس داده می شود و یه درجه عضویت محاسبه میشود و در دیکشنری تعریف شده ذخیره میشود در این دیکشنری لیبل فازی ست key هست و درجه عضویت value

تابع generate_rules:

```
    def generate_rules(self, training_data):
        rules = []

        for x1, x2, y in training_data:
            memberships_x1 = self.fuzzify_input(x1, self.input_fuzzy_sets_x1)
            memberships_x2 = self.fuzzify_input(x2, self.input_fuzzy_sets_x2)

            memberships_y = self.fuzzify_input(y, self.output_fuzzy_sets)

            x1_label = max(memberships_x1, key=memberships_x1.get)
            x2_label = max(memberships_x2, key=memberships_x2.get)
            y_label = max(memberships_y, key=memberships_y.get)

            weight = min(memberships_x1[x1_label], memberships_x2[x2_label])


            rules.append(((x1_label, x2_label), y_label, weight))

        rule_dict = {}
        for (antecedent, consequent, weight) in rules:
            if antecedent not in rule_dict or rule_dict[antecedent][1] < weight:
                rule_dict[antecedent] = (consequent, weight)

        self.rule_base = {k: v[0] for k, v in rule_dict.items()}
```

```
        return self.rule_base
```

در تابع generate_rules ورودی ما داده آموزشی ما هست که در این تابع قوانین بر اساس این داده ها ساخته می شود در ابتدا درجه عضویت ورودی ها و خروجی محاسبه می شود سپس فازی ستی که دارای بیشترین درجه عضویت هست انتخاب می شود و وزن قانون بر اساس مینیمم دو قانون یا همان intersection آن ها محاسبه می شود و در لیستی اضافه میشوند سپس باید conflict های قوانین آن حل شود در این قسمت در حلقه تعریف شده مطمئن میشویم که قوانین antecedent یا پیش آمد های غیر تکراری دارای بالاترین مقدار وزن هستند و اگر یک antecedent وجود داشته باشد در دیکشنری که وزن کمتری نسبت به قانونی که تازه محاسبه شده داشته باشد وزن آن قانون آپدیت و با وزن جدید جایگزین می شود. و در اخر یک لیست از قوانین نهایی برگردانده می شود. که مدل فازی ما بر اساس این قوانین کار میکنند.

X1 \ X2	A1	A2	A3	A4	A5	A6	A7
B1	C7	C5	C4	C4	C4	C5	C7
B2	C5	C4	C3	C2	C3	C4	C5
B3	C4	C3	C2	C1	C2	C3	C4
B4	C4	C2	C1	C1	C1	C2	C4
B5	C4	C3	C2	C1	C2	C3	C4
B6	C5	C4	C3	C2	C3	C4	C5
B7	C7	C5	C4	C4	C4	C5	C7

تابع evalutate rules:

```
    def evaluate_rules(self, input_memberships_x1, input_memberships_x2):

        output_memberships = {label: 0 for label in
self.output_fuzzy_sets.keys()}

        for (input1_label, input2_label), output_label in self.rule_base.items():
            if input1_label in input_memberships_x1 and input2_label in
input_memberships_x2:
                firing_strength = min(input_memberships_x1[input1_label],
input_memberships_x2[input2_label])

                output_memberships[output_label] =
max(output_memberships[output_label], firing_strength)

        return output_memberships
```

در این تابع که ورودی آن درجه عضویت متغیر اول و درجه عضویت متغیر دوم هست، طبق rule base ساخته شده درجه عضویت خروجی به ازای ورودی ها ساخته می شود. ابتدا لیبل فازی ست هارا از فازی ست خروجی که در constructor ساختیم پیدا کرده و در یک دیکشنری به عنوان key استفاده میکنیم و تمام value هارا 0 قرار میدهیم (این 0 ها همان درجه عضویت های خروجی هستند که در ادامه آپدیت می شود) سپس یک حلقه داریم که وظیفه iteration در لیست rule base را دارد که با استفاده از یک بلاک if چک میکند که طبق antecedent مورد نظر (antecedent ها یعنی ورودی اول و دوم) مطابق کدام قانون ساخته شده است. سپس اول با استفاده از تابع min قدرت شلیک یا اجرا یا همان firing strength بین دو ورودی محاسبه می شود و سپس با استفاده از تابع max درجه عضویت های خروجی مورد نیاز از از 0 به مقدار firing strength آپدیت می شود و اینکار باعث می شود که اگر چند قانون فازی منتهی به یک خروجی شوند درجه عضویت بالاتر ذخیره می شود.

تابع defuzzify_center_of_average:

```
    def defuzzify_center_of_average(self, memberships):
        numerator = sum(membership * self.output_centers[label] for label,
membership in memberships.items())
        denominator = sum(memberships.values())

        if denominator == 0:
            return 0

        return numerator / denominator
```

این تابع همان تابع دیفازیفیکیشن ما هست که از طریق روش center average انجام میشود. ورودی این تابع درجه عضویت های بدست امده هست (برای توضیح بیشتر درجه عضویت های بدست آمده از تابع evaluate rules) . در این تابع ابتدا numerator حساب می شود که می شود جمع وزن دار ممرشیپ ها یا همان درجه های عضویت که درجه عضویت ضربدر مرکز یا center به دست امده میشود (centroid ها در بخش constructor محاسبه شدند). سپس باید denominator را محاسبه کنیم که می شود جمع تمام درجه عضویت های بدست آمده. سپس برای به دست اوردن crisp value باید numerator بر denominator تقسیم شود که برای جلوگیری از خطای تقسیم بر صفر یک if بلاک قرار داده ایم که اگر مخرج 0 بود 0 را برگرداند.

تابع calculate_mse:

```
    def calculate_mse(self, predictions, targets):
        predictions = np.array(predictions)
        targets = np.array(targets)
        squared_differences = (predictions - targets) ** 2

        mse = np.sum(squared_differences) / (2 * len(targets))
        return mse
```

تابع محاسبه خطا MSE دارای دو ورودی هست ورودی اول پیش بینی های مدل هست و پیش بینی بعدی داده های واقعی و تارگت های ما هست. اول دو لیست ورودی را به آرایه های نامپای تبدیل میکنیم (برای تسهیل و تسریع محاسبه) سپس طبق فرمول گفته شده ابتدا مقدار اختلاف مقادیر پیش بینی شده با مقادیر واقعی به توان 2 را محاسبه کرده (squared error) سپس طبق فرمول داده شده مجموع تمام این مقادیر را تقسیم بر دو برابر تعداد مقادیر محاسبه میکنیم و تابع یک عدد به عنوان معیار برمیگرداند.

تابع predict:

```
    def predict(self, test_cases):
        x1_values = [case[0] for case in test_cases]
        x2_values = [case[1] for case in test_cases]
        actual_output = [case[2] for case in test_cases]

        predictions = []
        targets = []
        for input_value_x1, input_value_x2, actual_output in zip(x1_values,
x2_values, actual_output):
            memberships_x1 =
self.fuzzify_input(input_value_x1,self.input_fuzzy_sets_x1)
            memberships_x2 =
self.fuzzify_input(input_value_x2,self.input_fuzzy_sets_x2)
            output_memberships =
self.evaluate_rules(memberships_x1,memberships_x2)
            crisp_output = self.defuzzify_center_of_average(output_memberships)
            predictions.append(crisp_output)
            targets.append(actual_output)

        mse = model.calculate_mse(predictions, targets)
        print(f"Mean Squared Error (MSE): {mse}")

        return mse, predictions, targets, x1_values, x2_values
```

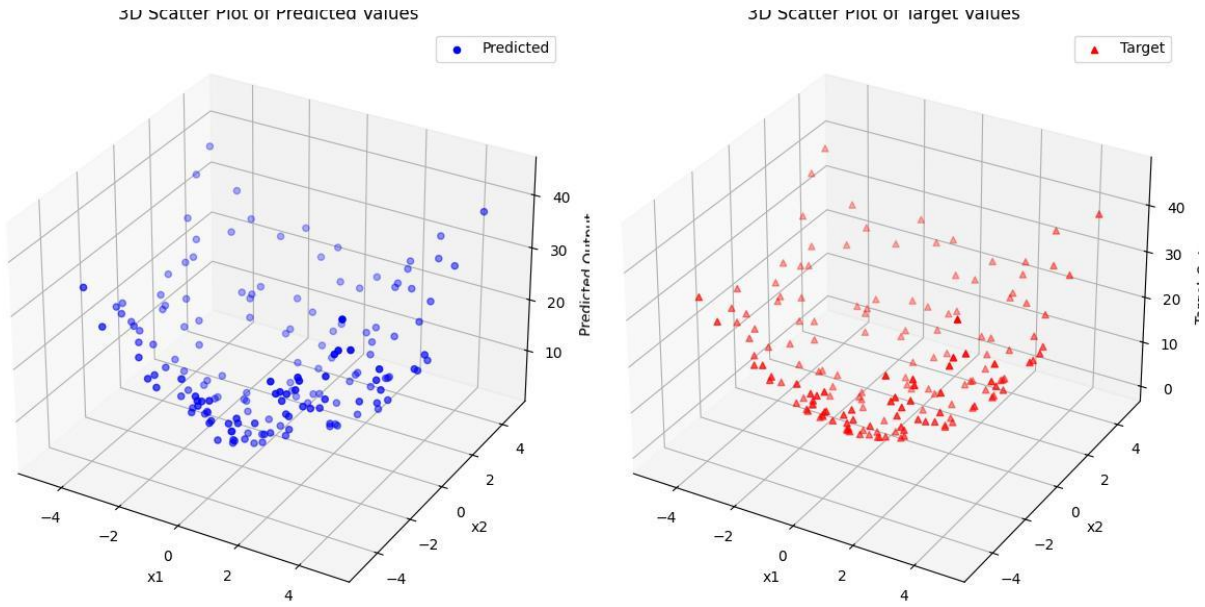
در این تابع یک ورودی به عنوان test_cases میگیرد که شامل یک ستون که مقادیر X1 را دارد و ستون بعدی که مقادیر X2 را دارد و ستون اخر که خروجی $f(x)=x_1^2+x_2^2$ را در بر دارد این تابع به صورت خلاصه تمام تابع های نوشته شده در بالا استفاده می شود (به غیر از generate rule از این تابع مانند تابع train استفاده میکنیم برای استفاده از تابع predict_plot_3d_output اول باید یک دور تابع generate rule را فرخوانی کنیم که قوانین ساخته شود سپس میتوان از تابع predict استفاده کرد) در این تابع بعد از ساخته شدن قوانین ستون های ورودی را به سه لیست مجزا تقسیم میکند سپس دو لیست تعریف کرده یکی برای مقادیر پیش بینی شده و یکی برای مقادیر واقعی (لیست targets صرفا برای خوانایی بهتر کد تعریف شده و میتوان همان استفاده را از طریق سومین لیست تشکیل شده از ورودی ها یعنی actual ouput برد) در ابتدا به ازای هر ردیف ورودی های x_1 و x_2 مقادیر را فازیفای میکنیم طبق توابع تعریف شده سپس با استفاده از تابع evaluate rules و ممبرشیپ های به دست امده و با استفاده قوانین تولید شده درجه عضویت خروجی را به دست میاوریم سپس با استفاده از تابع defuzzifying_center_of_average درجه عضویت خروجی را به یک crisp value تبدیل میکنیم و در انتها آن را به لیست predictions اضافه کرده و مقادیر واقعی را به targets اضافه میکنیم و در انتها با استفاده از دو لیست نهایی شده مقدار خطا را توسط تابع calculate mse به دست میاوریم.

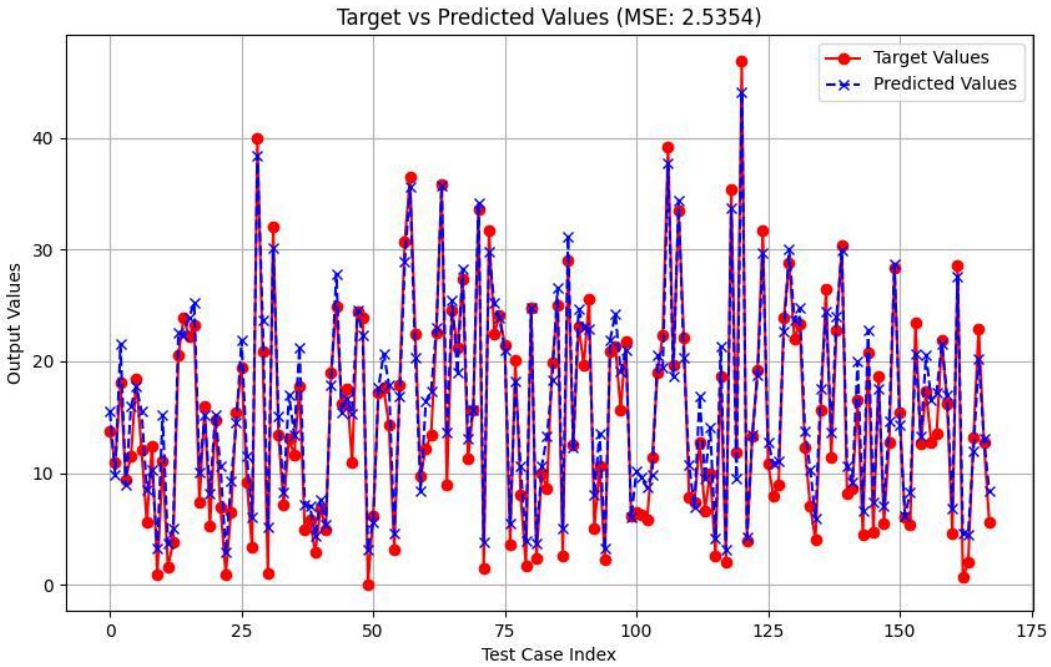
تابع plot_3d_output:

```
    def plot_3d_output(self,mse, predictions, targets,x1_values,
x2_values,title):
        fig1 = plt.figure(figsize=(12, 6))

        ax1 = fig1.add_subplot(121, projection='3d')
        ax1.scatter(x1_values, x2_values, predictions, c='blue', marker='o',
label='Predicted')
        ax1.set_title(f'3D Scatter Plot of Predicted Values Of {title}.png')
        ax1.set_xlabel('x1')
        ax1.set_ylabel('x2')
        ax1.set_zlabel('Predicted Output')
        ax1.legend()

        ax2 = fig1.add_subplot(122, projection='3d')
        ax2.scatter(x1_values, x2_values, targets, c='red', marker='^',
label='Target')
        ax2.set_title(f'3D Scatter Plot of Target Values Of {title}.png')
        ax2.set_xlabel('x1')
        ax2.set_ylabel('x2')
        ax2.set_zlabel('Target Output')
        ax2.legend()

        plt.tight_layout()
```

```
        plt.savefig(f"3D_Scatter_Target_vs_Predicted_{title}.jpg")
        plt.show()

        plt.figure(figsize=(10, 6))
        plt.plot(range(len(targets)), targets, label='Target Values',
color='red', linestyle='-', marker='o')
        plt.plot(range(len(predictions)), predictions, label='Predicted Values',
color='blue', linestyle='--', marker='x')

        plt.title(f'Target vs Predicted {title} Values (MSE: {mse:.4f})')
        plt.xlabel('Test Case Index')
        plt.ylabel('Output Values')
        plt.legend()
        plt.grid(True)

        plt.savefig(f"Line_Plot_Target_vs_Predicted {title}.jpg")
        plt.show()
```

در خط های بعد با استفاده از تابع plot_3d_ouput و خروجی های گرفته شده از تابع predict برای تفهیم بهتر خروجی دو پلات سه بعدی یکی برای داده های پیش بینی شده و یکی برای داده های حقیقی ساخته می شود با کمک کتابخانه matplotlib که در ان می توان اختلاف های آن را مشاهده کرد. یک نمودار خطی ها از داده های پیش بینی شده و حقیقی نیز ساخته می شود که بتوان اختلاف آن هارا بهتر متوجه شد





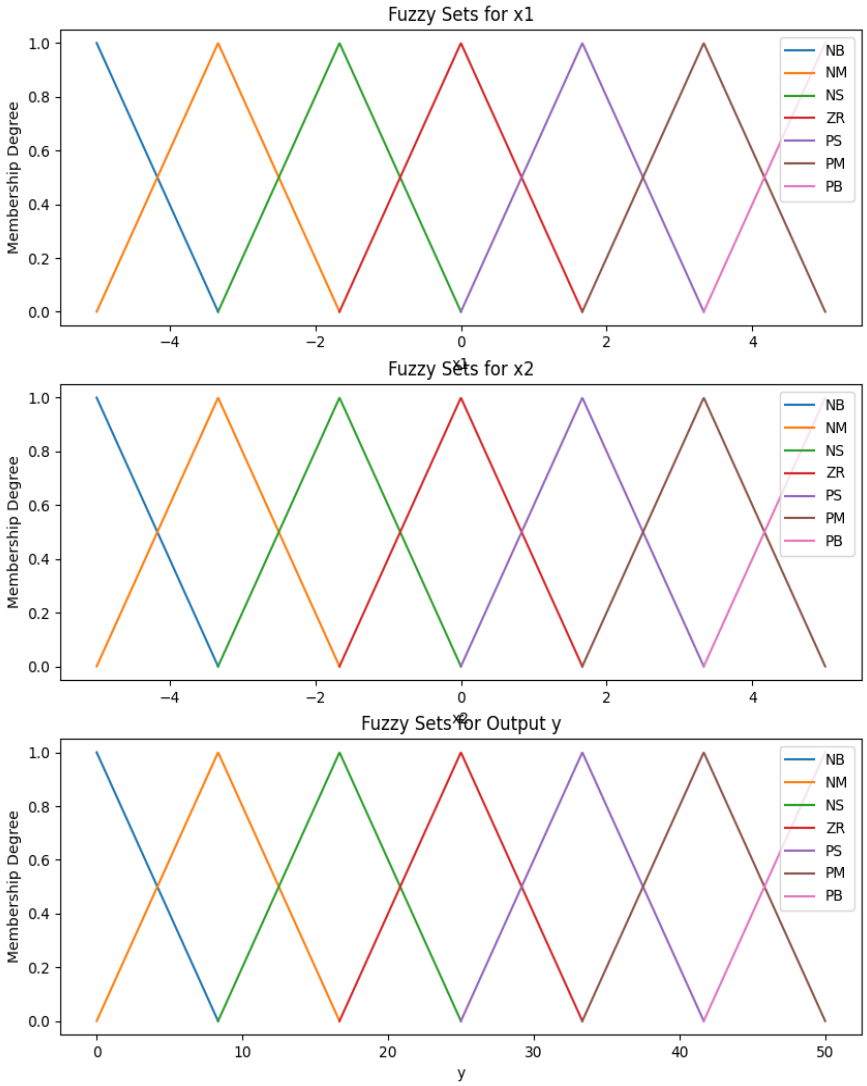
تابع plot_fuzzy_sets:

```
    def plot_fuzzy_sets(self):
        vectorized_mf = np.vectorize(self.triangular_mf)

        fig, ax = plt.subplots(3, 1, figsize=(10, 12))

        ax[0].set_title('Fuzzy Sets for x1')
        for label, (a, b, c) in self.input_fuzzy_sets_x1.items():
            x = np.linspace(a , c , 500)
            y = vectorized_mf(x, a, b, c)
            ax[0].plot(x, y, label=f'{label}')
        ax[0].legend(loc='upper right')
        ax[0].set_xlabel('x1')
        ax[0].set_ylabel('Membership Degree')

        ax[1].set_title('Fuzzy Sets for x2')
        for label, (a, b, c) in self.input_fuzzy_sets_x2.items():
            x = np.linspace(a , c, 500)
            y = vectorized_mf(x, a, b, c)
            ax[1].plot(x, y, label=f'{label}')
        ax[1].legend(loc='upper right')
        ax[1].set_xlabel('x2')
        ax[1].set_ylabel('Membership Degree')

        ax[2].set_title('Fuzzy Sets for Output y')
```

```
        for label, (a, b, c) in self.output_fuzzy_sets.items():
            x = np.linspace(a , c , 500)
            y = vectorized_mf(x, a, b, c)
            ax[2].plot(x, y, label=f'{label}')
        ax[2].legend(loc='upper right')
        ax[2].set_xlabel('y')
        ax[2].set_ylabel('Membership Degree')

        # Show the plots
        plt.savefig("Fuzzy sets Plots")
        plt.tight_layout()
        plt.show()
```

در این تابع که باز هم برای تفهیم بهتر و مطمئن شدن از اینکه فازی ست ها به درستی تعریف شده اند نوشته شده. در این تابع، تابع درجه عضویت مثلثاتی نوشته شده برای فازی ست های ورودی اول یعنی x_1 و فازی ست دوم یعنی x_2 و فازی ست خروجی با 3 ساب پلات (subplot) کشیده میشود .اول با استفاده از کتابخانه نامپای و تابع vectorize (یک تابع wrapper هست که به ما اجازه می دهد که تابع درجه عضویت مثلثاتی خود را روی تموم ورودی ها بدون اینکه از حلقه ها استفاده کنیم اجرا کنیم و تمام خروجی هارا به صورت یک لیست تحویل میگیریم) تابع vectorized_mf را میسازیم.

سپس یک صفحه تعریف میکنیم با سایز 10,12 و دارای 3 ساب پلات برای هر ساب پلات مراحلی که گفته می شود اجرا می شود. اول تیتر ساب پلات را مشخص میکنیم به دلخواه سپس برای فازی ست هایی که در کلاس تعریف شده (در constructor کلاس) و مقادیر تعریف شده و لیبل های هر فازی ست (NB,NM,NS,ZR,PS,PM,PB) را از متغیر آن با یک حلقه گرفته و برای هر کدام از a تا c (یعنی از شروع آن تا پایان آن) 500 مقدار با فاصله یکسان تعریف و در x گذاشته و برای y از تابع vectoriced_mf که تعریف کردیم استفاده میکنیم که خروجی های آن به دست بیاوریم. سپس نمودار را براساس x و y به دست میاوریم و از label های فازی ست برای راهنما ساب پلات ها میگیریم.


Fuzzy Sets for x1
Membership Degree
x1
NB
NM
NS
ZR
PS
PM
PB
Fuzzy Sets for x2
Membership Degree
x2
NB
NM
NS
ZR
PS
PM
PB
Fuzzy Sets for Output y
Membership Degree
y
NB
NM
NS
ZR
PS
PM
PB

بلاک اجرایی برنامه یا همان main

```
if __name__ == "__main__":

    data = pd.read_csv("training_data.csv")
    data= data.to_records(index=False)
    model = FuzzyModel(data)
    # Generate rules using Wang-Mendel method
    model.generate_rules(data)
    print("\nGenerated Rule Base:")
    for rule, output in model.rule_base.items():
        print(f"If x1 is {rule[0]} and x2 is {rule[1]}, then output is {output}")
    train_mse_before_optimization, predictions, targets, x1_values, x2_values =
model.predict(data)
    model.plot_3d_output(train_mse_before_optimization, predictions, targets,
x1_values, x2_values,"Train_Before_Optimization")
    optimized_model, best_mse = simulated_annealing(model, data)

    train_mse_after_optimization, predictions, targets, x1_values, x2_values =
optimized_model.predict(data)
    optimized_model.plot_3d_output(train_mse_after_optimization, predictions,
targets, x1_values, x2_values,"Train_Before_Optimization")
    print(f"Best MSE after optimization: {best_mse}")
    print(f"Train MSE Before optimaztion:{train_mse_before_optimization}")
    print(f"Train MSE After optimaztion:{train_mse_after_optimization}")


    model.plot_fuzzy_sets("Original Model")
    optimized_model.plot_fuzzy_sets("Optimized Model")
    mse=0
    for i in range(100):
        x1_values = np.random.uniform(-5, 5, 41)
        x2_values = np.random.uniform(-5, 5, 41)
        output = x1_values**2 + x2_values**2

        data = pd.DataFrame({'x1': x1_values, 'x2': x2_values, 'F(x1, x2)':
output})
        data = data.to_records(index=False)

        test_mse_before_optimization, predictions, targets, x1_values, x2_values
= model.predict(data)
        print(f"The PRE-Optimized MSE {test_mse_before_optimization}")
        mse = mse + test_mse_before_optimization
        if(i%99==0):
            model.plot_3d_output(test_mse_before_optimization, predictions,
targets, x1_values, x2_values,"Test Mse PRE-Optimised")
```

```
    test_mse_before_optimization = mse/100
    mse=0
    for i in range(100):
        x1_values = np.random.uniform(-5, 5, 168)
        x2_values = np.random.uniform(-5, 5, 168)
        output = x1_values**2 + x2_values**2

        data = pd.DataFrame({'x1': x1_values, 'x2': x2_values, 'F(x1, x2)':
output})
        data = data.to_records(index=False)

        test_mse_optimized, predictions, targets, x1_values, x2_values =
optimized_model.predict(data)
        print(f"The Optimized MSE {test_mse_optimized}")
        mse = mse + test_mse_optimized
        if(i%99==0):
            optimized_model.plot_3d_output(test_mse_optimized, predictions,
targets, x1_values, x2_values,"Test Mse Optimised")
    test_mse_optimized = mse/100
    print(f"test_mse_before_optimization:{test_mse_before_optimization}")
    print(f"test_mse_optimized: {test_mse_optimized}")
```

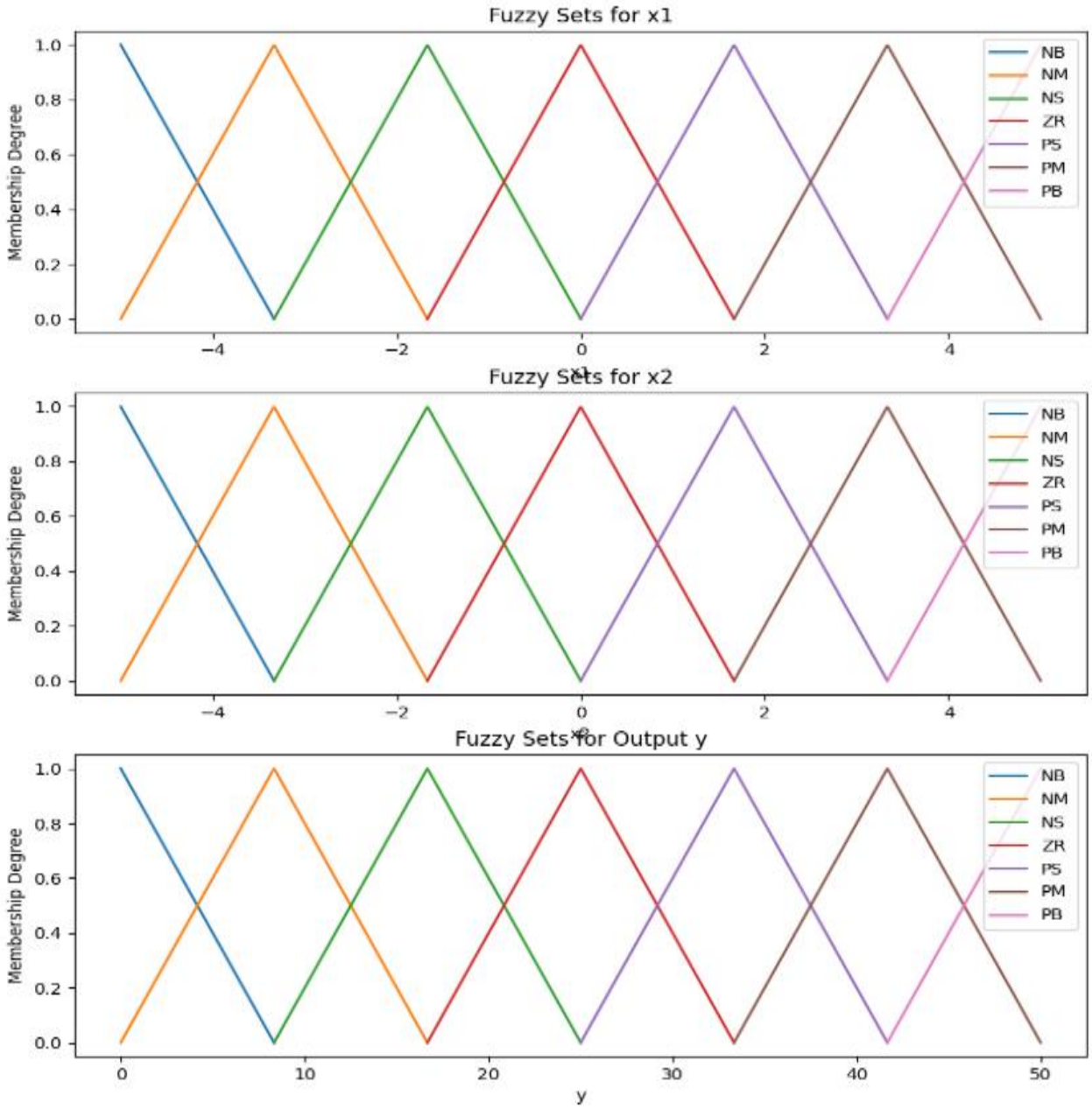
در این بلاک داده های آموزشی که از قبل برای راحتی بیشتر به صورت یک فایل csv ذخیره شده بود را در متغیر data قرار داده ایم و برای اینکه مدل بتواند از داده ها استفاده کند با استفاده از کتابخانه pandas و تابع to_records داده ها را از دیتافریم به آرایه های نامپای تغییر داده ایم و سپس data را به کلاس پاس میدهیم تا فازی ست های مورد نیاز را برای کلاس بسازد و در آبجکت ساخته شده را در model ذخیره میکنیم، سپس با همان داده های آموزشی تابع generate rules را صدا میزنیم که قوانین مورد نیاز ساخته شوند و با استفاده از یک حلقه برای تفهیم بهتر قوانین را پرینت میگیریم.

سپس تابع predict را صدا زده و مقادیر crisp پیش بینی شده برای داده های آموزشی را تشکیل میدهیم و با استفاده از مقادیری که تابع به ما برمیگرداند تابع plot_3d_output را استفاده کرده و نتایج را به صورت پلات ذخیره و نمایش میدهد.

سپس برای داده های تست هم از یک حلقه 100 تایی استفاده کرده که در آن هربار یک سری داده رندوم uniform 168 داده تولید میکند برای x1 و x2 سپس ouput آن هارا حساب کرده و تبدیل به دیتا فریم کرده و سپس دیتافریم را تبدیل به یک ماتریس کرده و آن را به تابع predict کلاس ما داده سپس mse ها را هردور حساب کرده و در دور اخر حلقه برای نمونه پلات های مورد نیاز را توسط تابع plot_3d_ouput تشکیل می شوند و در اخر میانگین تمام mse های به دست اماده حساب میشود (اینکار برای این است که تاثیر رندوم بودن داده های تست را در نتیجه mse کمرنگ تر شود). و تمام این مراحل برای مدل optimize شده ما نیز تکرار می شود.

نتایج:

فازی ست های تشکیل شده:



قوانین ساخته شده:

X1 \ X2	A1	A2	A3	A4	A5	A6	A7
B1	C7	C5	C4	C4	C4	C5	C7
B2	C5	C4	C3	C2	C3	C4	C5
B3	C4	C3	C2	C1	C2	C3	C4
B4	C4	C2	C1	C1	C1	C2	C4
B5	C4	C3	C2	C1	C2	C3	C4
B6	C5	C4	C3	C2	C3	C4	C5
B7	C7	C5	C4	C4	C4	C5	C7

X1 \ X2	1	2	3	4	5	6	7
1	7	5	4	4	4	5	7
2	5	4	3	2	3	4	5
3	4	3	2	1	2	3	4
4	4	2	1	1	1	2	4
5	4	3	2	1	2	3	4
6	5	4	3	2	3	4	5
7	7	5	4	4	4	5	7

Antecedent X1	Antecedent X2	Consequent	Weight
A1	B1	C7	1
A2	B1	C5	0.95
A3	B1	C4	0.95
A4	B1	C4	1
A5	B1	C4	0.95
A6	B1	C5	0.95
A7	B1	C7	1
A1	B2	C5	0.95
A2	B2	C4	0.95
A3	B2	C3	0.95
A4	B2	C2	0.95
A5	B2	C3	0.95
A6	B2	C4	0.95
A7	B2	C5	0.95
A1	B3	C4	0.95
A2	B3	C3	0.95
A3	B3	C2	0.95
A4	B3	C1	0.95
A5	B3	C2	0.95
A6	B3	C3	0.95
A7	B3	C4	0.95
A1	B4	C4	1
A2	B4	C2	0.95
A3	B4	C1	0.95
A4	B4	C1	1
A5	B4	C1	0.95
A6	B4	C2	0.95
A7	B4	C4	1
A1	B5	C4	0.95
A2	B5	C3	0.95
A3	B5	C2	0.95
A4	B5	C1	0.95
A5	B5	C2	0.95
A6	B5	C3	0.95
A7	B5	C4	0.95
A1	B6	C5	0.95
A2	B6	C4	0.95
A3	B6	C3	0.95
A4	B6	C2	0.95
A5	B6	C3	0.95
A6	B6	C4	0.95
A7	B6	C5	0.95
A1	B7	C7	1
A2	B7	C5	0.95
A3	B7	C4	0.95
A4	B7	C4	1
A5	B7	C4	0.95
A6	B7	C5	0.95
A7	B7	C7	1

مقدار خطا یا MSE:

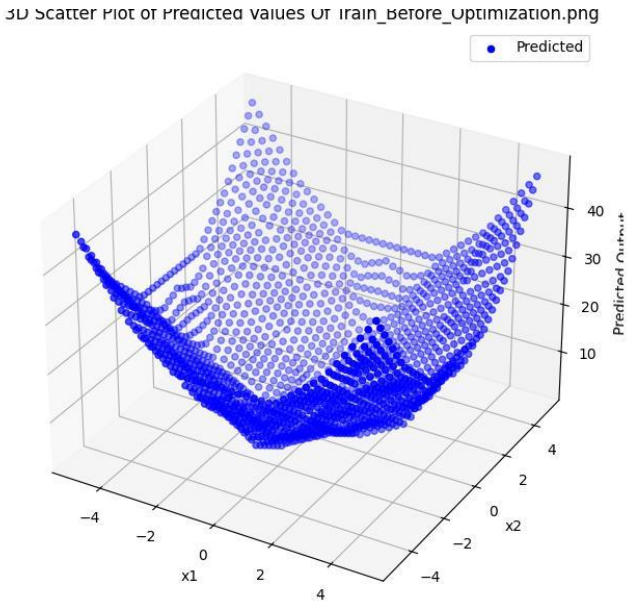
(با هربار تشکیل داده های تست به صورت رندوم این مقدار عوض میشود اما حدود 2 همیشه هست)

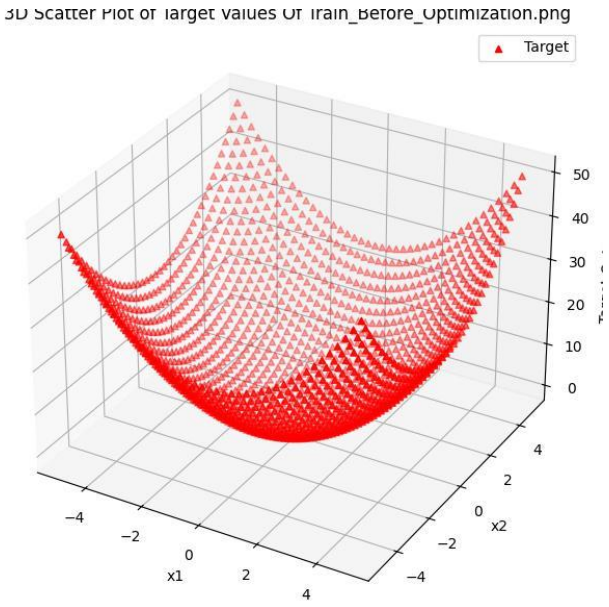
Iteration	MSE	Iteration	MSE	Iteration	MSE
1st iteration	2.453943759	50th iteration	2.66410403	99th iteration	2.341894653
2nd iteration	3.393433232	51st iteration	2.151142817	100th iteration	3.163685195
3rd iteration	2.825454934	52nd iteration	2.645183381		
4th iteration	2.434407618	53rd iteration	2.90476685		
5th iteration	2.265697539	54th iteration	2.331428727		
6th iteration	2.891110241	55th iteration	3.382168383		
7th iteration	2.552479163	56th iteration	2.538121674		
8th iteration	2.685944872	57th iteration	2.683707619		
9th iteration	3.131952988	58th iteration	1.508477408		
10th iteration	2.755815416	59th iteration	2.635334254		
11th iteration	2.899639259	60th iteration	2.678876502		
12th iteration	2.628843993	61st iteration	2.759247155		
13th iteration	3.007304039	62nd iteration	2.159835997		
14th iteration	2.339508695	63rd iteration	2.863250135		
15th iteration	2.056144623	64th iteration	2.42543035		
16th iteration	2.80675386	65th iteration	2.193639543		
17th iteration	2.995912808	66th iteration	2.793010659		
18th iteration	3.857410047	67th iteration	2.574417442		
19th iteration	1.974968937	68th iteration	2.821589938		
20th iteration	2.936644219	69th iteration	2.536344054		
21st iteration	2.839266663	70th iteration	2.275200253		
22nd iteration	2.516687429	71st iteration	2.825109162		
23rd iteration	2.112974867	72nd iteration	2.205635782		
24th iteration	2.173903641	73rd iteration	2.438281918		
25th iteration	2.598362549	74th iteration	2.996116737		
26th iteration	2.895355197	75th iteration	2.658538515		
27th iteration	2.721839882	76th iteration	3.694590913		
28th iteration	2.787783424	77th iteration	2.74407402		
29th iteration	2.566230484	78th iteration	2.619678867		
30th iteration	2.451988229	79th iteration	2.638796469		
31st iteration	2.719319887	80th iteration	1.98558295		
32nd iteration	3.222448867	81st iteration	2.930253981		
33rd iteration	2.972013271	82nd iteration	2.766320832		
34th iteration	2.780203049	83rd iteration	2.492298261		
35th iteration	3.220278788	84th iteration	2.462819494		
36th iteration	2.49454102	85th iteration	2.876286438		
37th iteration	2.472722448	86th iteration	2.864451028		
38th iteration	2.594009044	87th iteration	2.708230642		
39th iteration	2.147055034	88th iteration	2.55140178		
40th iteration	2.050681944	89th iteration	2.400355391		
41st iteration	2.690889573	90th iteration	2.321739743		
42nd iteration	1.862281049	91st iteration	2.378731623		
43rd iteration	2.965159205	92nd iteration	2.966933389		
44th iteration	3.339108742	93rd iteration	2.763100781		
45th iteration	2.832438043	94th iteration	2.302202745		
46th iteration	2.685265255	95th iteration	2.38837253		
47th iteration	2.317320703	96th iteration	2.421585926		
48th iteration	2.06496408	97th iteration	2.320141517		
49th iteration	2.438708156	98th iteration	2.829412137		

Train Before Optimization	2.625699
Test Before Optimization	2.630391

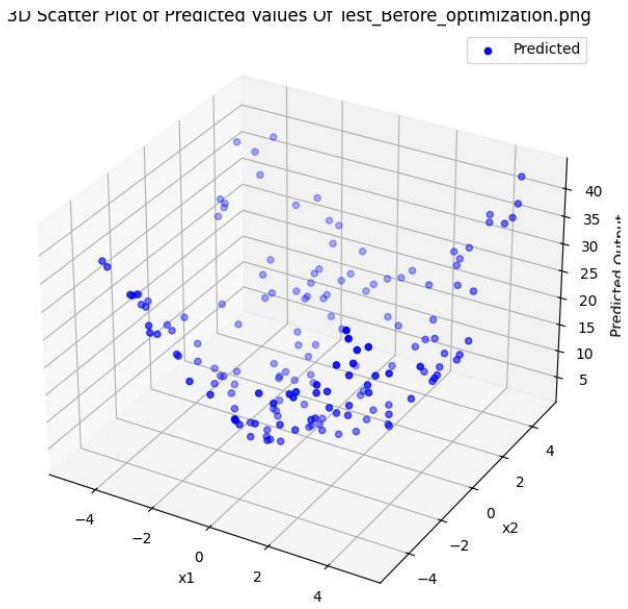
Crisp value های پیشبینی شده (نمودار):

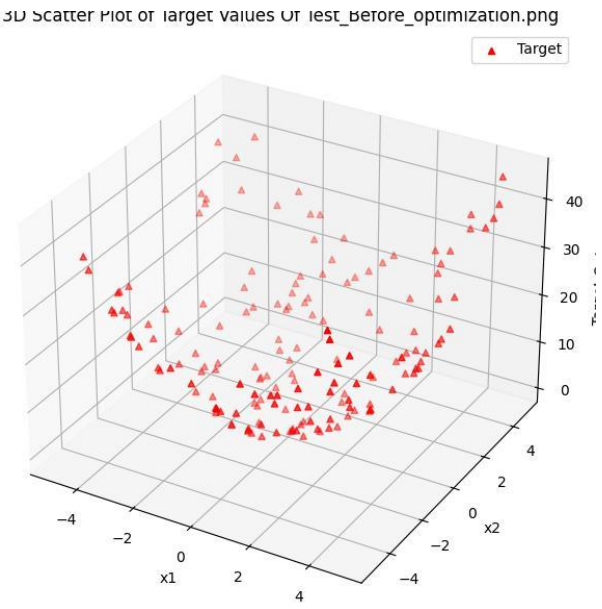
داده های آموزشی:





داده های تست:





پیاده سازی الگوریتم SA:

در این قسمت با استفاده از الگوریتم SA تلاش می شود که مدل خود را تنظیم کرده و MSE یا همان خطا را کاهش دهیم که با استفاده از 3 تابع پیاده سازی شده است.

تابع simulated_annealing:

```
def simulated_annealing(model, training_data, initial_temp=0.1,
cooling_rate=0.98, max_iterations=500):
    """
    Optimize fuzzy sets using the Simulated Annealing (SA) algorithm.
```

```
    Parameters:
        model: FuzzyModel instance.
        training_data: Training dataset.
        initial_temp: Initial temperature for SA.
        cooling_rate: Cooling rate for temperature decay.
        max_iterations: Maximum number of iterations.

    Returns:
        Optimized FuzzyModel and the best MSE achieved.
    """

    current_model = copy.deepcopy(model)
    current_mse = evaluate_model(current_model, training_data)

    best_model = copy.deepcopy(current_model)
    best_mse = current_mse

    temperature = initial_temp

    for iteration in range(max_iterations):


        new_model = copy.deepcopy(current_model)


        choice = random.choice(['input_x1', 'input_x2', 'output'])
        if choice == 'input_x1':
            fuzzy_sets = new_model.input_fuzzy_sets_x1
        elif choice == 'input_x2':
            fuzzy_sets = new_model.input_fuzzy_sets_x2
        else:
            fuzzy_sets = new_model.output_fuzzy_sets

        label = random.choice(list(fuzzy_sets.keys()))
        fuzzy_set = list(fuzzy_sets[label])

        perturbation_range = 0.1 * (fuzzy_set[2] - fuzzy_set[0])
        fuzzy_set[0] += random.uniform(-perturbation_range, perturbation_range)
        fuzzy_set[1] += random.uniform(-perturbation_range, perturbation_range)
        fuzzy_set[2] += random.uniform(-perturbation_range, perturbation_range)


        fuzzy_set[0] = max(fuzzy_set[0], 0)
        fuzzy_set[1] = max(fuzzy_set[1], fuzzy_set[0])
        fuzzy_set[2] = max(fuzzy_set[2], fuzzy_set[1])
```

```
        fuzzy_sets[label] = tuple(fuzzy_set)
        if iteration % 100 == 0:
            print(f"Iteration {iteration}: Temp={temperature:.4f}, Current
MSE={current_mse:.6f}, Best MSE={best_mse:.6f}")
            print(f"Current fuzzy set for {label}:
{new_model.input_fuzzy_sets_x1[label]}")
        new_mse = evaluate_model(new_model, training_data)

        if new_mse < current_mse or random.uniform(0, 1) <
acceptance_probability(current_mse, new_mse, temperature):
            current_model = new_model
            current_mse = new_mse

            if new_mse < best_mse:
                best_model = copy.deepcopy(new_model)
                best_mse = new_mse

        temperature = initial_temp / ( 1 + cooling_rate * iteration)

    return best_model, best_mse
```

ورودی های این تابع شامل یک مدل هست (مدل فازی)

داده های تمرینی

انرژی یا Temperature اولیه (به صورت پیش فرض 0.1)

نرخ کاهش Temperature (به صورت پیش فرض 0.98)

و مقدار iterations که باید اتفاق بیوفتد (به صورت پیش فرض 500)

در این کد ابتدا یک کپی عمیق یا همان deepcopy از مدل فازی اولیه گرفته می شود که تغییرات مورد نیاز روی مدل فازی اصلی اتفاق نیافتد و اینکه بتوانیم مدل هارا از این طریق مقایسه و بهترین مدل را پیدا کنیم.

سپس از تابع evaluate_model (در ادامه طرز کار این تابع توضیح داده می شود) استفاده میکنیم تا mse اولیه مدل خود را پیدا کنیم. سپس صرفا برای خوانایی بهتر کد از مدل یک کپی گرفته و در متغیری به نام best_model ذخیره کرده و mse به دست اورده را به عنوان best_mse ذخیره میکنیم و temperature اولیه هم را در متغیری به اسم temperature ذخیره میکنیم.

سپس با استفاده از یک حلقه به تعداد iterations های تعریف شده (یعنی 500 بار) شروع میکنیم به ازای هر دور حلقه یک کپی از مدل گرفته و به صورت رندوم از بین فازی ست های ورودی های x_1 و x_2 و فازی ست خروجی یکی انتخاب شده (اینکار برای جلوگیری از این است که الگوریتم روی فازی ست به خصوصی تمرکز نکند که سبب تغییر بیش از حد و اورفیت شود) و به اندازه حدود 10 درصد فاصله ابتدا و انتها دامنه perturbed خود انتخاب میکنیم که به هرکدام از مقادیر a, b, c به صورت رندوم از منفی شروع دامنه تا

مثبت آن عددی انتخاب می شود و به فازی ست مثلثاتی ما اضافه می شود. و برای اینکه مطمئن باشیم که شرط های فازی ست های مثلثاتی برقرار است از تابع max برای پیدا کردن شروع یعنی همان a و قله یعنی b و پایان فازی ست یعنی c استفاده میکنیم.

پس از اینکه فازی ست های جدید برای مدل محاسبه شدند باز با استفاده از تابع evaluate_model استفاده میکنیم تا mse جدید را به دست بیاوریم سپس با استفاده از یک بلاک if چک میکنیم که آیا mse جدید از قدیم کمتر است یا اینکه با استفاده از تابع acceptance_probability (در ادامه طرز کار تابع توضیح داده می شود) چک می شود که احتمال قبول کردن مدل با mse بدتر چقدر هست (اینکار برای این هست که الگوریتم در مراحل اول در local minima گیر نکند و با قبول کردن مدلی با mse بدتر در مراحل بعد به یک mse گلوبال بهتر برسیم) اگر شرط ها برقرار باشد،mse current با mse جدید جایگزین می شوند و model_current هم با مدل جدید جایگزین می شود.

در ادامه در بلاک if توضیح داده شده یک if دیگر نیز وجود دارد که چک میکند که mse جدید آیا از بهترین mse پیدا شده کمتر است و اگر شرط برقرار باشد از مدل یک کپی گرفته و به عنوان best_model آن را ذخیره کرده و mse آن را به عنوان best_mse ذخیره میکند (متغیر ریست می شود) و در آخر temperature طبق فرمول داده شده به مقداری کاهش پیدا میکند.

در آخر پس از اتمام حلقه متغیر بهترین mse یا همان best_mse و بهترین مدل یا همان best_model برگردانده می شود.

تابع evaluated_model:

```
def evaluate_model(model, data):

    predictions = []
    targets = []

    for x1, x2, y in data:
        memberships_x1 = model.fuzzify_input(x1, model.input_fuzzy_sets_x1)
        memberships_x2 = model.fuzzify_input(x2, model.input_fuzzy_sets_x2)
        output_memberships = model.evaluate_rules(memberships_x1, memberships_x2)
        crisp_output = model.defuzzify_center_of_average(output_memberships)
        predictions.append(crisp_output)
        targets.append(y)

    return model.calculate_mse(predictions, targets)
```

در این تابع که ورودی ها:

مدل هست (model) که مدل فازی مورد نظر به ان پاس داده می شود

و ورودی دوم هم داده آموزشی ما هست

در این تابع صرفا از مدل ران گرفته میشود به خروجی آن با داده حقیقی ما مقایسه می شود و توضیحات این توابع در بخش اول توضیحات پروژه نوشته می شود.

تابع acceptance_probability:

```
def acceptance_probability(current_mse, new_mse, temperature):

    if new_mse < current_mse:
        return 1.0
    return np.exp((current_mse - new_mse) / temperature)
```

ورودی این تابع ها:

Mse موجود یا همان current_mse

Mse جدید یا همان new_mse

و انرژی یا همان temperature

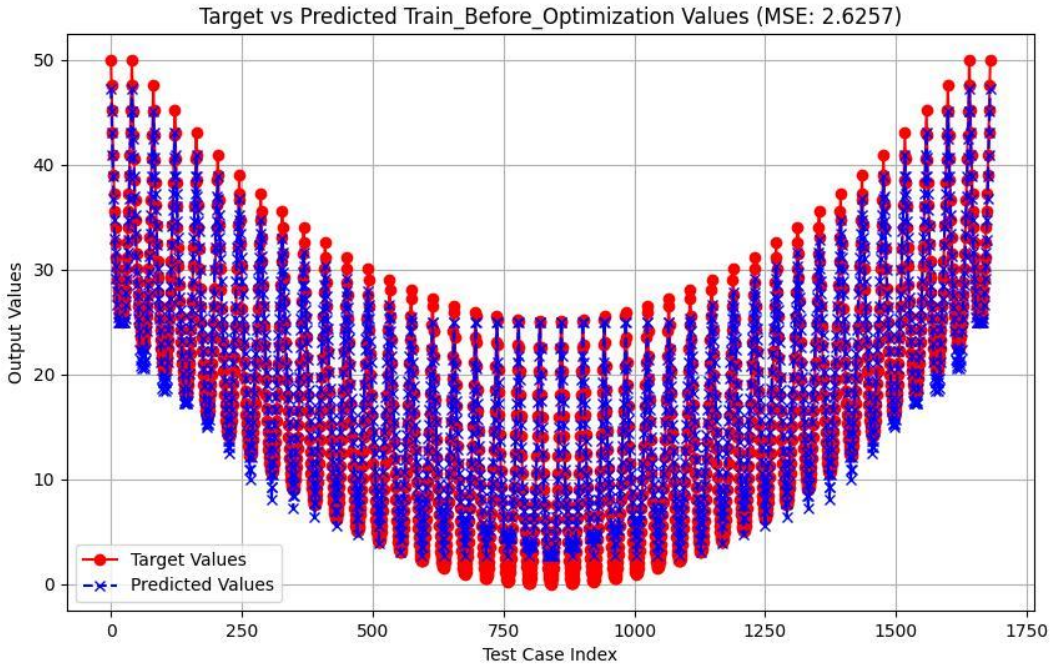
در این تابع اگر mse جدید پایین تر از mse موجود باشد 1 را برمیگرداند در غیر اینصورت طبق فرمول مقدار e به توان اختلاف mse ها تقسیم بر انرژی را برمیگرداند.

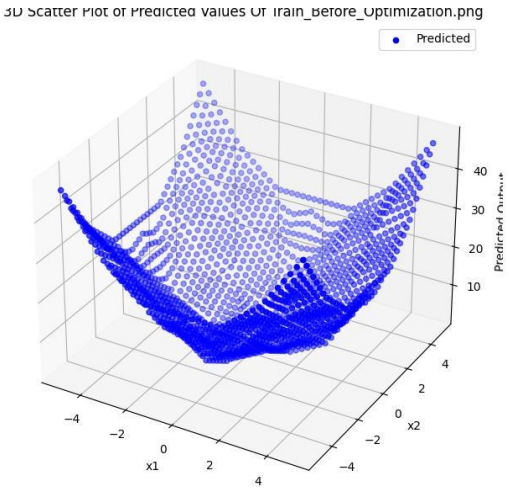
نتایج:

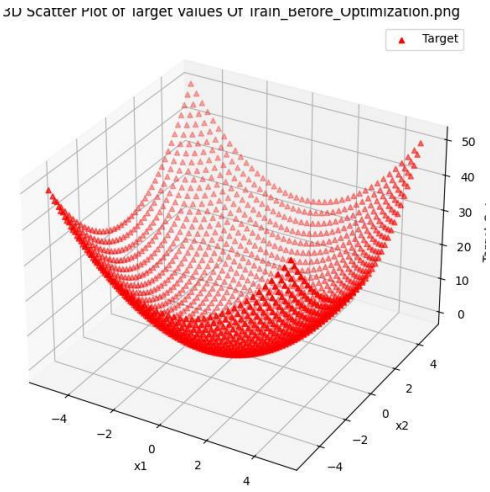
به صورت کلی با استفاده از الگوریتم SA مقدار mse پایین تر امد و پیش بینی ها به داده های حقیقی نزدیک تر شد. برای سریع تر شدن برنامه تعداد iteration به صورت دیفالت 500 قرار داده شده و اگر بیشتر قرار داده شود مثلا 1000 به مقدار mse مطلوب تری میرسیم.

داده های تمرینی:

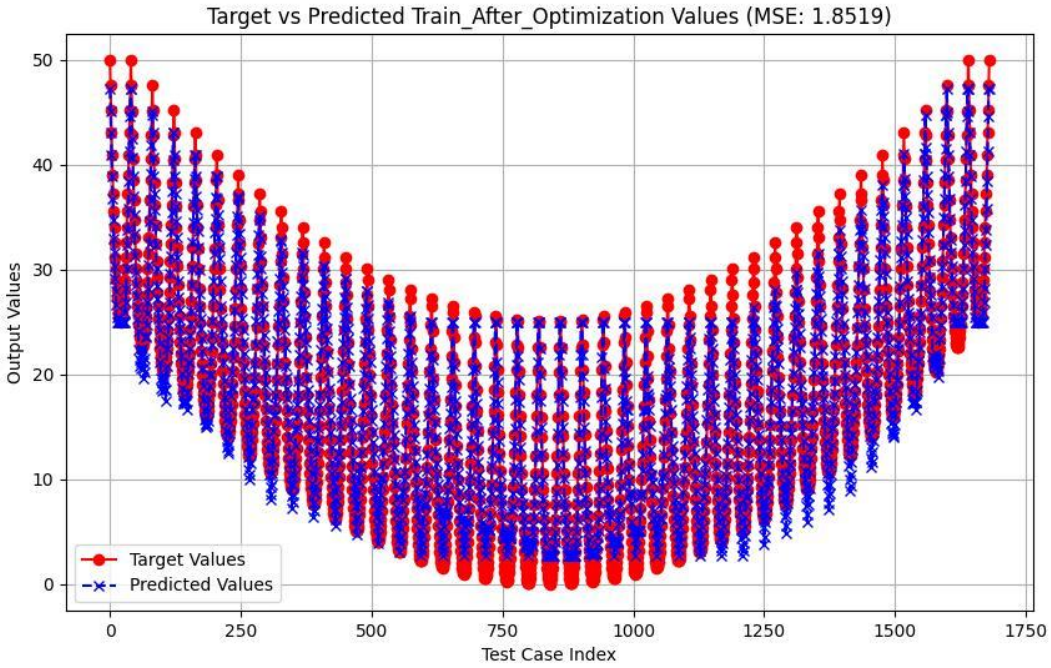
قبل از الگوریتم SA

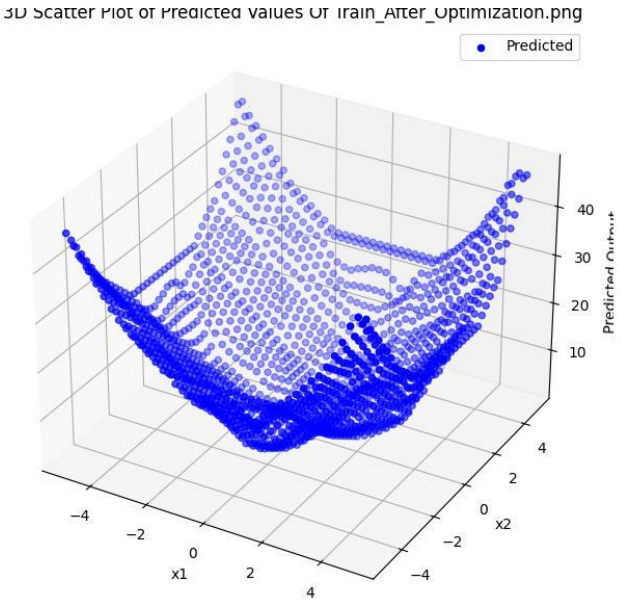


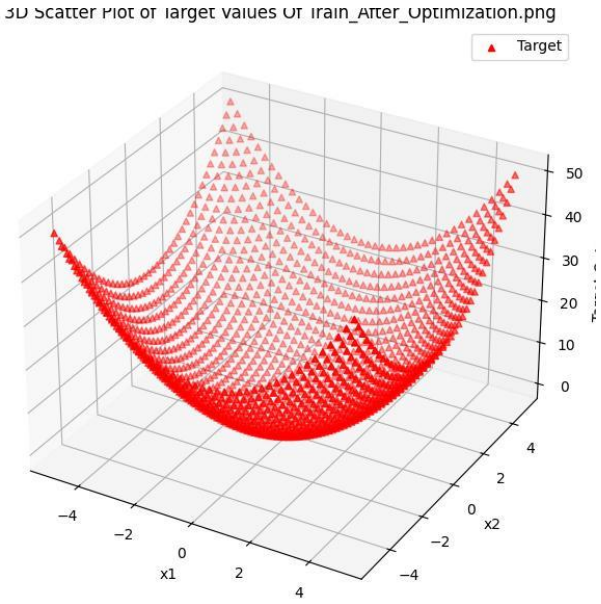




بعد از الگوریتم SA

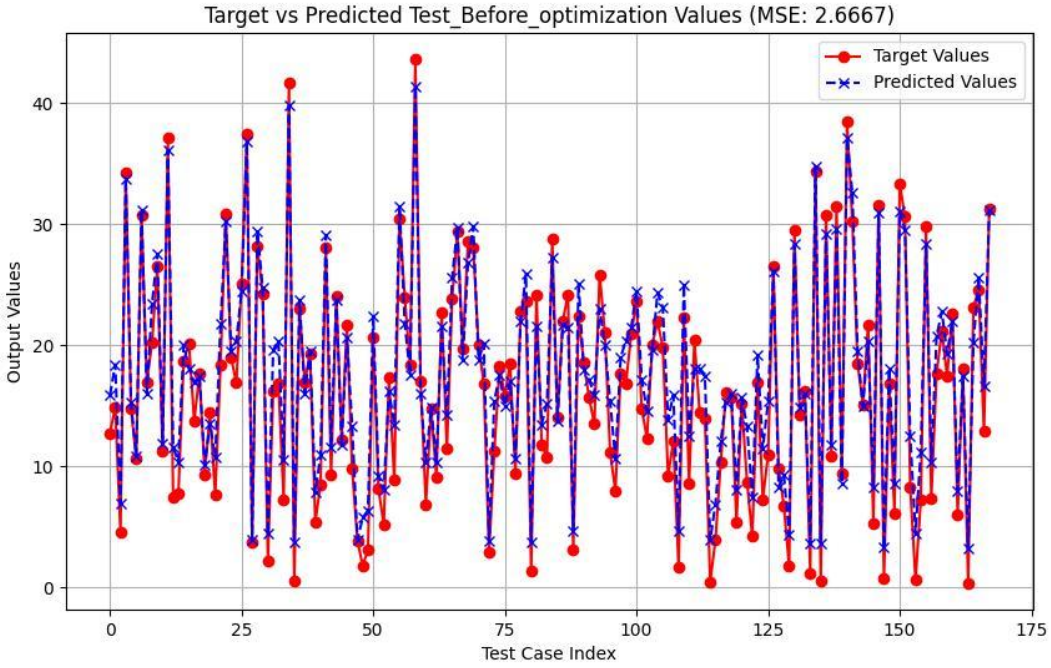


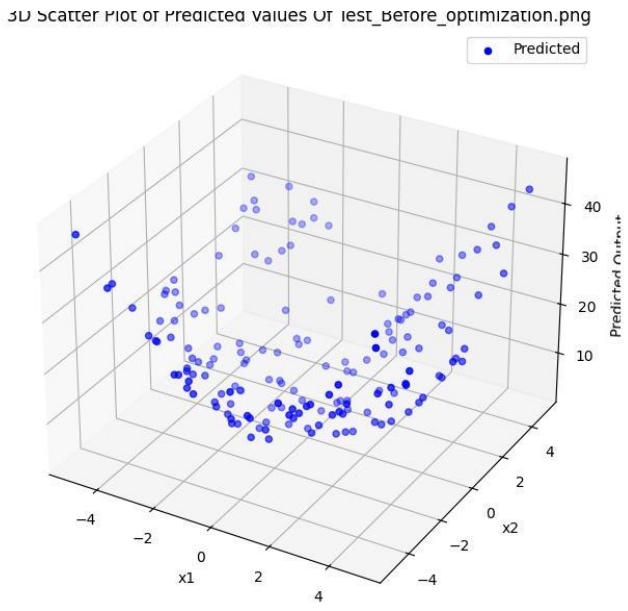


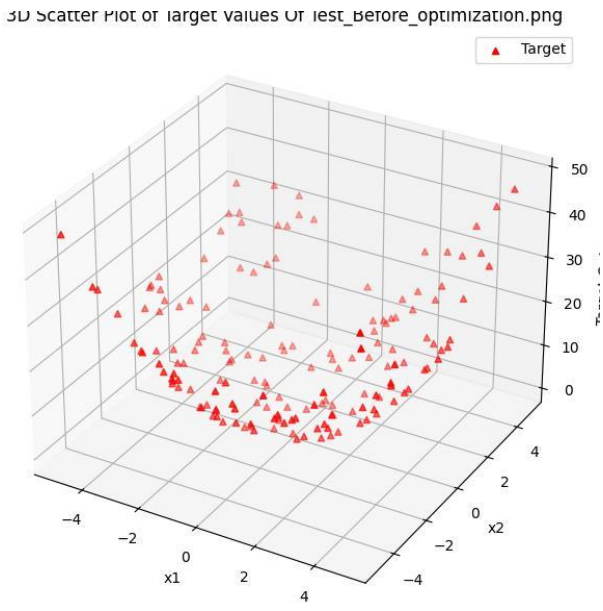


داده های تست:

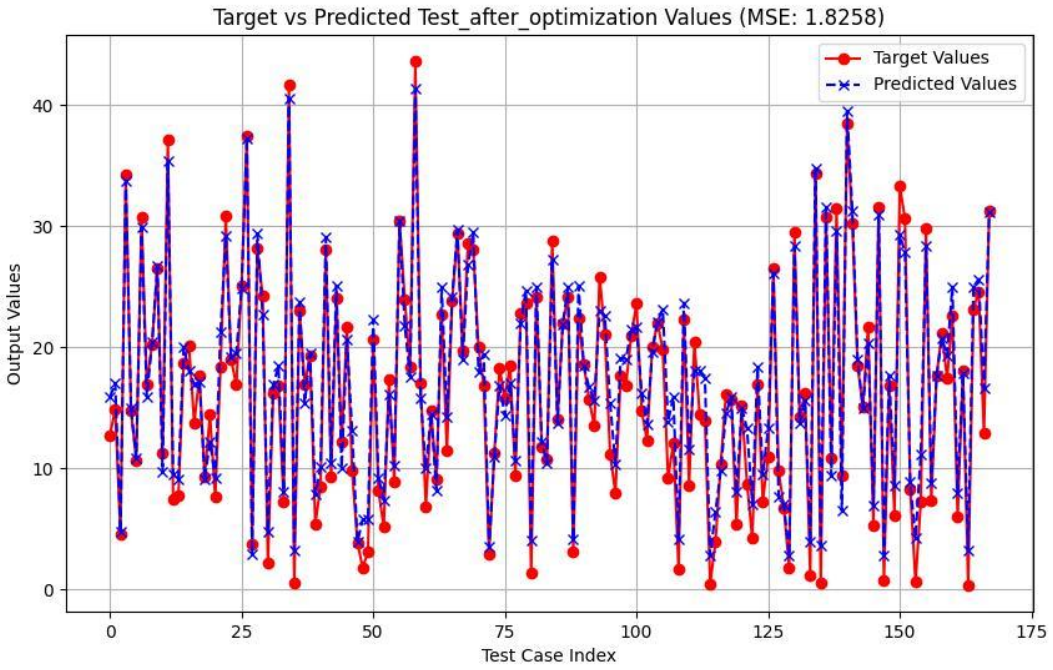
قبل از الگوریتم SA

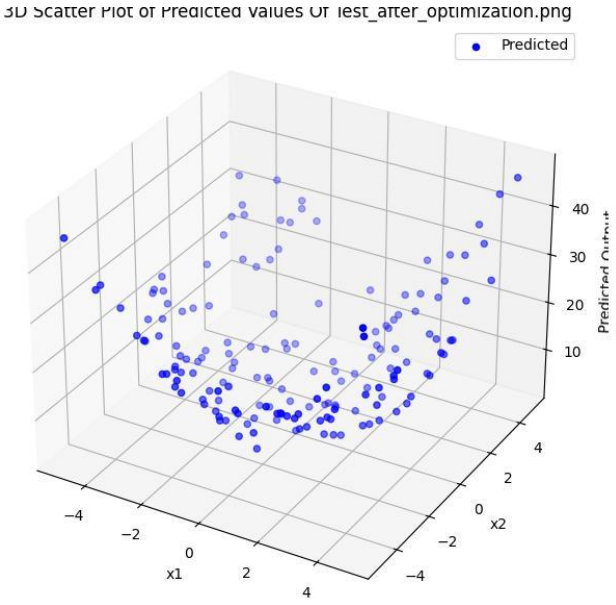


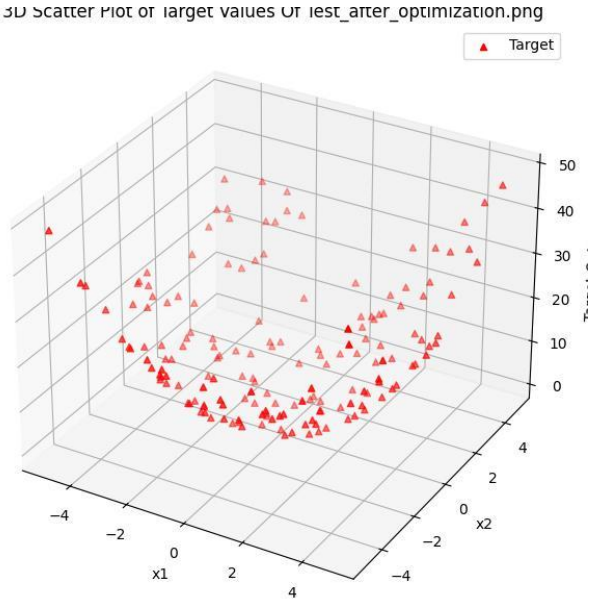




بعد از الگوریتم SA

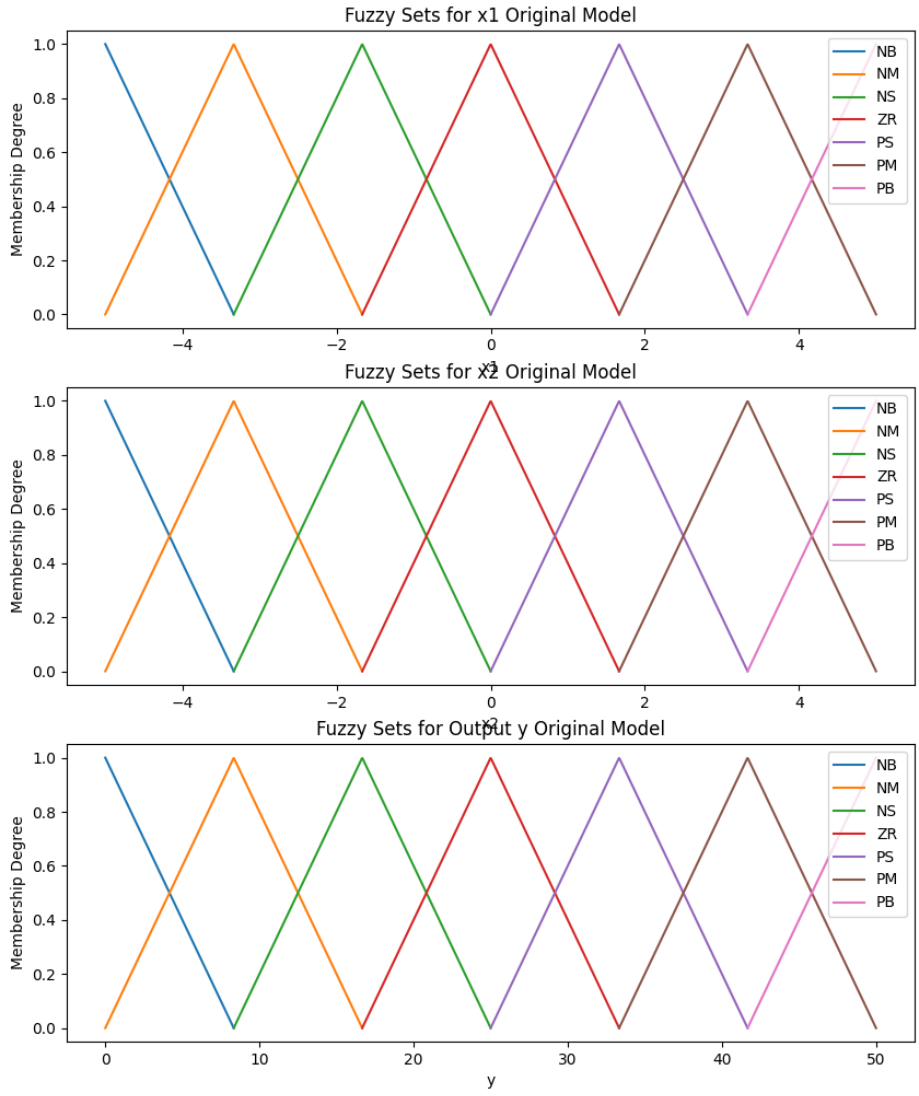




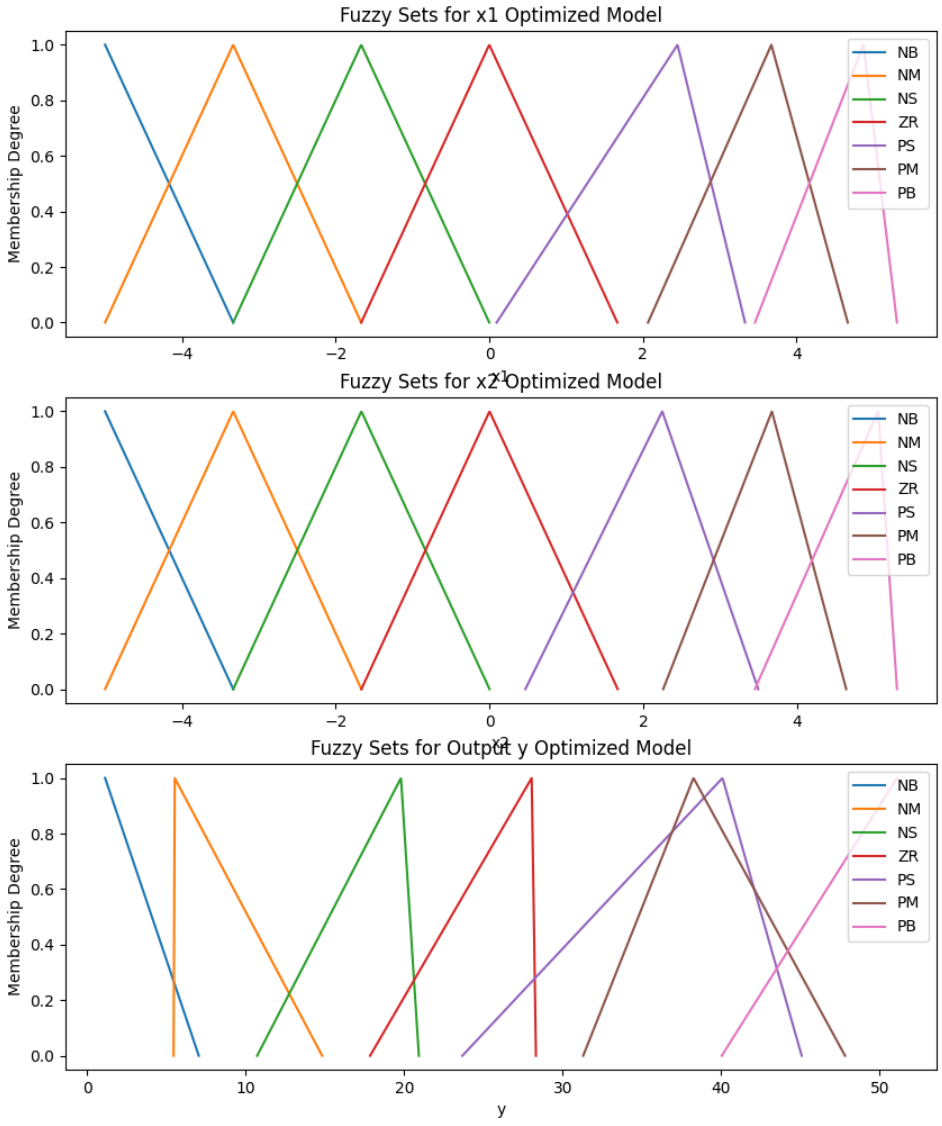


فازی ست ها:

قبل از الگوریتم SA



بعد از الگوریتم SA



خطاها (MSE):

خروجی SA برای داده های آموزش:

Train Before Optimization	2.625698977
Train After Optimization	1.742399641

خروجی SA برای داده های تست:

Iteration	MSE	Iteration	MSE	Iteration	MSE
1st iteration	2.12905913	50th iteration	2.209067962	99th iteration	1.447514069
2nd iteration	1.697973755	51st iteration	1.64442054	100th iteration	2.105806011
3rd iteration	1.976463882	52nd iteration	1.694301353		
4th iteration	2.361504114	53rd iteration	1.541945726		
5th iteration	1.754248332	54th iteration	2.038037822		
6th iteration	1.733804935	55th iteration	2.010672064		
7th iteration	1.661170489	56th iteration	1.709847233		
8th iteration	1.853088066	57th iteration	2.049150719		
9th iteration	1.325997063	58th iteration	1.979978421		
10th iteration	1.681145233	59th iteration	1.800325877		
11th iteration	1.360883657	60th iteration	1.904749422		
12th iteration	1.899641891	61st iteration	2.100451703		
13th iteration	1.830184278	62nd iteration	1.481638734		
14th iteration	1.913085024	63rd iteration	2.047444771		
15th iteration	1.539395737	64th iteration	1.662398605		
16th iteration	1.91831295	65th iteration	1.817592315		
17th iteration	2.094070927	66th iteration	1.695319429		
18th iteration	1.600302284	67th iteration	1.370022271		
19th iteration	1.914340075	68th iteration	1.971323105		
20th iteration	2.136639708	69th iteration	2.251409683		
21st iteration	1.5829648	70th iteration	1.973358722		
22nd iteration	1.359902033	71st iteration	1.911430805		
23rd iteration	2.051112588	72nd iteration	1.768207826		
24th iteration	2.21684714	73rd iteration	1.580428559		
25th iteration	1.657392175	74th iteration	1.922470511		
26th iteration	1.552532327	75th iteration	1.326678571		
27th iteration	2.484897059	76th iteration	2.235353645		
28th iteration	1.893142211	77th iteration	2.4132201		
29th iteration	2.541312734	78th iteration	1.7996284		
30th iteration	2.09677735	79th iteration	1.677142985		
31st iteration	1.858073584	80th iteration	2.091340175		
32nd iteration	1.491571697	81st iteration	1.742744643		
33rd iteration	2.121108142	82nd iteration	1.576070291		
34th iteration	2.318186001	83rd iteration	1.921287188		
35th iteration	1.832794815	84th iteration	1.90041015		
36th iteration	1.282127354	85th iteration	1.419947265		
37th iteration	1.637609254	86th iteration	1.859464889		
38th iteration	1.68969514	87th iteration	1.275153819		
39th iteration	1.938678801	88th iteration	3.060165856		
40th iteration	2.144664206	89th iteration	1.681271044		
41st iteration	1.61737281	90th iteration	2.037209315		
42nd iteration	1.590719046	91st iteration	2.228672837		
43rd iteration	1.555790504	92nd iteration	1.307520745		
44th iteration	1.56993263	93rd iteration	1.913386958		
45th iteration	1.637337212	94th iteration	1.698421276		
46th iteration	1.655004322	95th iteration	1.562279012		
47th iteration	1.335207101	96th iteration	1.831178105		
48th iteration	1.693303011	97th iteration	1.595022349		
49th iteration	1.503021258	98th iteration	2.278468387		

Test Before Optimization	2.630390782
Test After Optimization	1.863843502