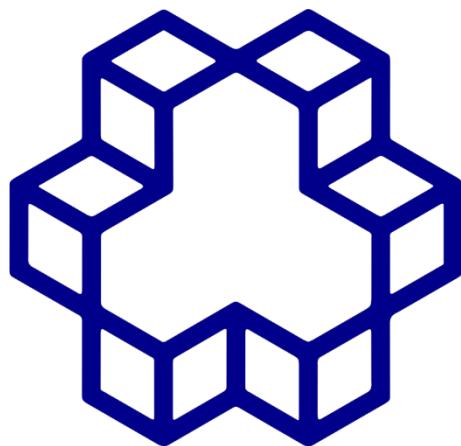


به نام خدای رنگین کمان



دانشگاه صنعتی خواجه نصیرالدین طوسی

نام و نام خانوادگی: سجاد فودازی

شماره دانشجویی: 40007903

استاد: دکتر مهدی علیاری

درس: مبانی سیستم‌های هوشمند

موضوع: گزارشکار مینی پروژه 3



## مطالب

3.....	تصاویر
4.....	جداول
5.....	پرسش اول
5.....	سوال 1
8.....	سوال 2
12.....	پرسش دوم
13.....	پرسش سوم
13.....	سیستم 1
21.....	سیستم 2
31.....	پرسش چهارم
37.....	پرسش پنجم
38.....	شناسایی با ANFIS
42.....	شناسایی با RBF



## تصاویر

تصویر 1 - سیستم نوسانی دائم برای طراحی زیگلر-نیکولز	5.....
تصویر 2 - مقادیر PID در بلوک سیمولینک	6.....
تصویر 3 - پاسخ پله سیستم با تأخیر	7.....
تصویر 4 - پاسخ پله سیستم با اغتشاش	7.....
تصویر 5 - پاسخ پله سیستم با اعتشاش و نویز	8.....
تصویر 6 - مقایسه پاسخ پله	8.....
تصویر 7 - پاسخ پله کنترل شده توسط PIDtuner	9.....
تصویر 8 - پارامتر های بدست آمده با PIDtuner	9.....
تصویر 9 - پاسخ پله سیستم با تأخیر	10.....
تصویر 10 - پاسخ پله سیستم با اغتشاش	10.....
تصویر 11 - پاسخ پله با نویز	11.....
تصویر 12 - مقایسه پاسخ های پله	11.....
تصویر 13 - مقایسه خروجی واقعی و پیشبینی شده در آموزش	16.....
تصویر 14 - مقایسه خروجی واقعی و پیشبینی شده در تست	17.....
تصویر 15 - خطای آموزش	17.....
تصویر 16 - خطای تست	18.....
تصویر 17 - توزیع خطای آموزش	18.....
تصویر 18 - توزیع خطای تست	19.....
تصویر 19 - توابع تعلق ورودی	19.....
تصویر 20 - قوانین فازی مدل ANFIS	20.....
تصویر 21 - نمودار سطحی خروجی مدل فازی	20.....
تصویر 22 - حطا به ازای افزایش epoch	21.....
تصویر 23 - فشار درام واقعی و پیشبینی شده در تست	22.....
تصویر 24 - RMSE فشار درام به ازای تعداد ایپاک	23.....
تصویر 25 - فازی سرفیس فشار درام	23.....
تصویر 26 - فازی سرفیس فشار درام	24.....
تصویر 27 - اکسیژن اضافی واقعی و پیشبینی شده در تست	24.....
تصویر 28 - RMSE اکسیژن اضافی به ازای تعداد ایپاک	25.....
تصویر 29 - فازی سرفیس اکسیژن اضافی	25.....
تصویر 30 - فازی سرفیس اکسیژن اضافی	26.....
تصویر 31 - سطح آب واقعی و پیشبینی شده در تست	26.....
تصویر 32 - RMSE سطح آب به ازای تعداد ایپاک	27.....
تصویر 33 - فازی سرفیس سطح آب	27.....
تصویر 34 - فازی سرفیس سطح آب	28.....



---

تصویر 35 - سطح بخار واقعی و پیش‌بینی شده در تست	28
تصویر 36 - RMSE بخار آب به ازای تعداد ایپاک	29
تصویر 37 - فازی سرفیس بخار آب	29
تصویر 38 - فازی سرفیس بخار آب	30
تصویر 39 - خروجی واقعی و پیش‌بینی شده	35
تصویر 40 - فازی سرفیس	35
تصویر 41 - فازی سرفیس	36
تصویر 42 - مقدار واقعی و پیش‌بینی شده در تست	41
تصویر 43 - مقدار واقعی و پیش‌بینی شده در ولیدیشن	41
تصویر 44 - RMSE به ازای تعداد ایپاک	42
تصویر 45 - مقدار واقعی و پیش‌بینی شده در تست	44
تصویر 46 - مقدار واقعی و پیش‌بینی شده در ولیدیشن	44
تصویر 47 - کاهش خطای افزایش ایپاک	45
تصویر 48 - مقدار RMSE به ازای افزایش نورون	45

## جداول

جدول 1 - طراحی کنترلر با زیگلر-نیکولز	6
---------------------------------------	---



# پرسش اول

## سوال 1

برای سیستم زیر یک کنترل کننده PID با استفاده از زیگلر نیکلز طراحی کنید.

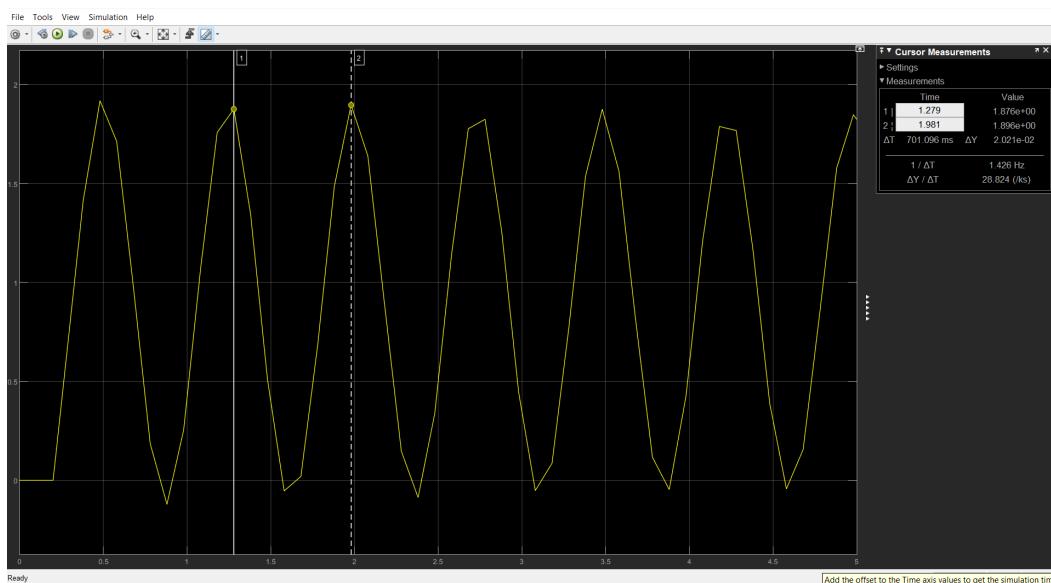
$$T(s) = \frac{s}{s + 1}$$

سیستم زیر را در حالت اضافه شدن تاخیر، نویز و اغتشاش کنترل کنید.

برای شبیه سازی سیستم فوق موفق نشدم و بجاش سیستم زیر را مدل کردم:

$$G(s) = \frac{1}{s + 1}$$

سپس سیستم را در سیمولینک وارد میکنیم و با استفاده transport delay تاخیر 0.2 ثانیه به سیستم وارد میکنیم و فیدبک واحد رو میبندیم و پشت سیستم یک گین ثابت پشت سیستم اضافه میکنیم. با افزایش گین تا زمانی که سیستم نوسانی دائم شود Ku را شناسایی میکنیم. مقدار 9 برای Ku بدست می آید و به ازای این Ku مقدار Tu که مقدار زمان پیک تو پیک ماکس میباشد 0.7 ثانیه میباشد.



تصویر 1 - سیستم نوسانی دائم برای طراحی زیگلر-نیکلز

با استفاده از جدول زیر مقادیر را طراحی میکنیم:



جدول 1 - طراحی کنترلر با زیگلر-نیکولز

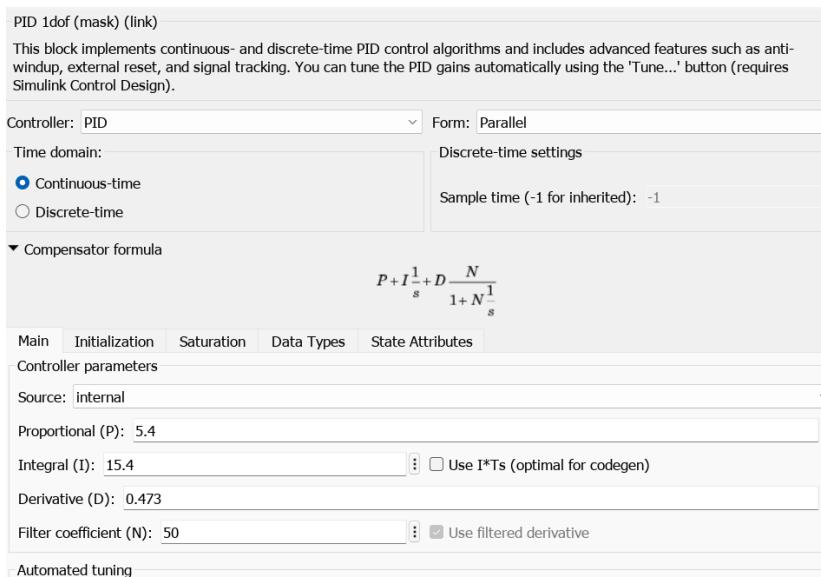
Controller	$K_p$	$K_i$	$k_d$
P	$K_u/2$	-	-
PI	$K_u/2.2$	$P_u/1.2$	-
PID	$K_u/1.7$	$P_u/2$	$P_u/8$

$$K_p = 5.4$$

$$T_i = 0.351$$

$$T_d = 0.08775$$

سپس این مقادیر را به فرمت PID بلوک سیمولینک تبدیل میکنیم و مقادیر را در بلوک PID وارد میکنیم:

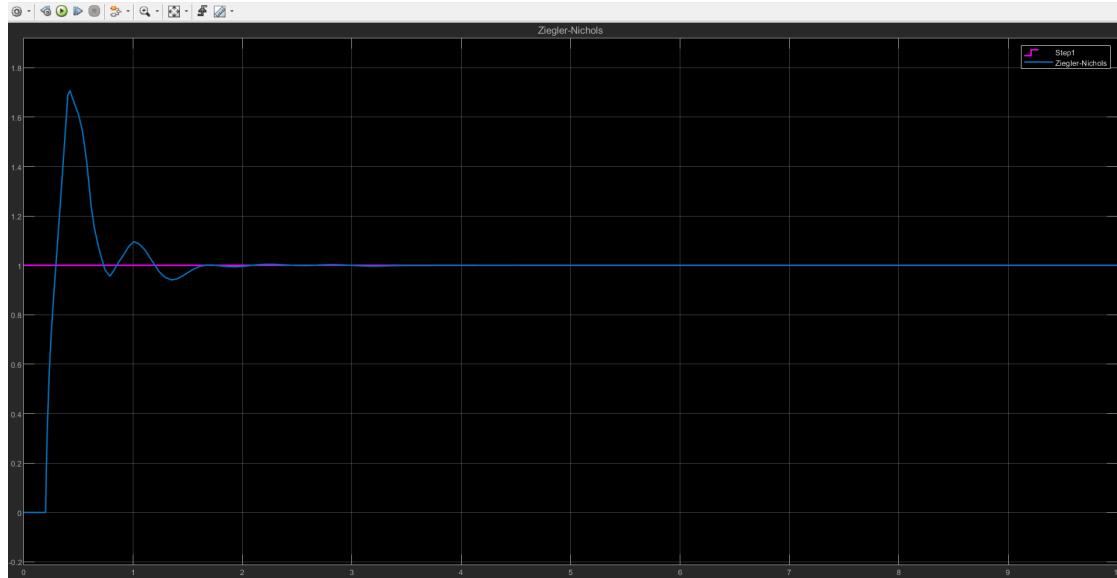


تصویر 2 - مقادیر PID در بلوک سیمولینک

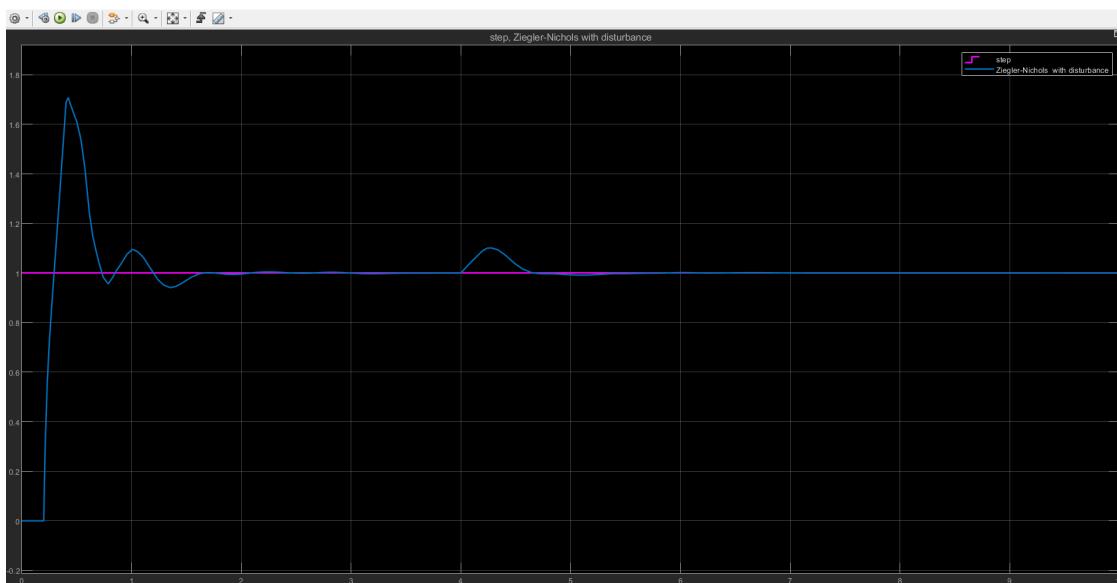
در نهایت سیستم را به سه نحوه باید ران کنیم:

1. سیستم و تاخیر
2. با اغتشاش
3. با نویز

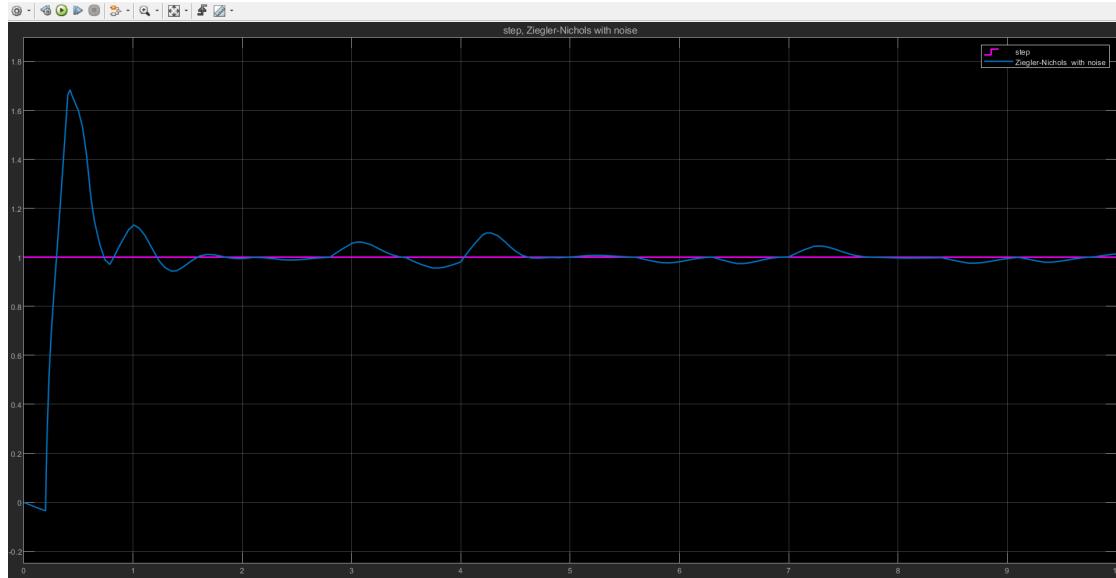
برای اغتشاش از یک بلوک پله ثابت از ثانیه 4 به بعد با مقدار 0.5 استفاده میکنیم. برای نویز از بلوک band-limited white noise استفاده میکنیم. سمپل نویز تصادفی را 0.7 ثانیه میگذاریم. در نهایت نتایج به شکل زیر میباشد:



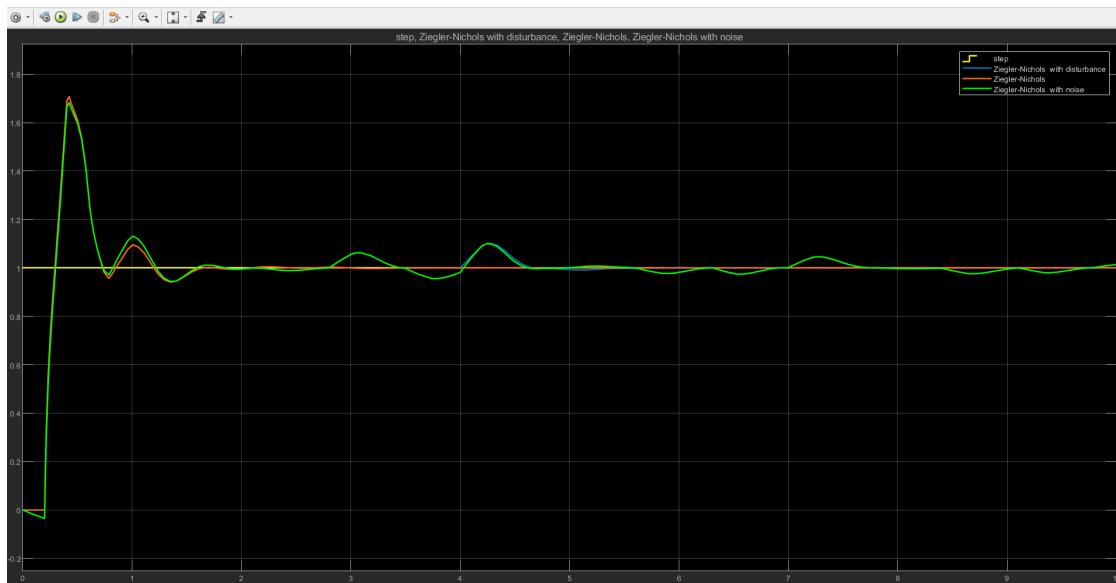
تصویر ۳ - پاسخ پله سیستم با تاخیر



تصویر ۴ - پاسخ پله سیستم با اغتشاش



تصویر ۵ - پاسخ پله سیستم با اغتشاش و نویز



تصویر ۶ - مقایسه پاسخ پله

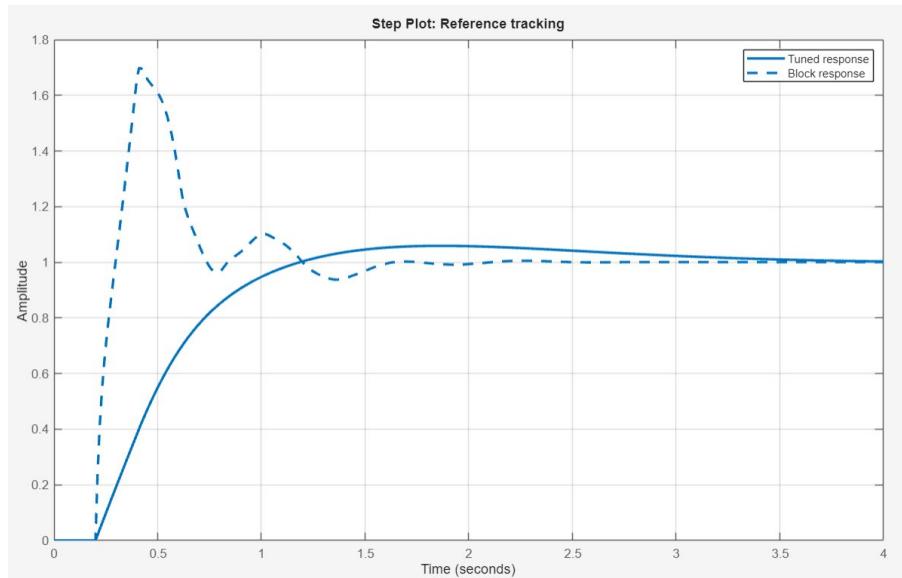
مشاهده میشود که کنترل کننده زیگلر-نیکولز بخوبی توانسته اغتشاش را خیلی سریع حذف کند ولی نتوانسته است نویز های ورودی را فیلتر کند و در برابر نویز عملکرد ضعیفی داشته است.

## سوال 2

سپس ضرایب کنترل کننده را با استفاده از کنترل فازی طراحی کرده و هردو کنترل کننده طراحی شده را با tuning خود متلب مقایسه کنید.



برای کنترل با PIDtuner خود مطلب فقط کافیست از این ابزار در سیمولینک استفاده کنیم. محیط این ابزار:



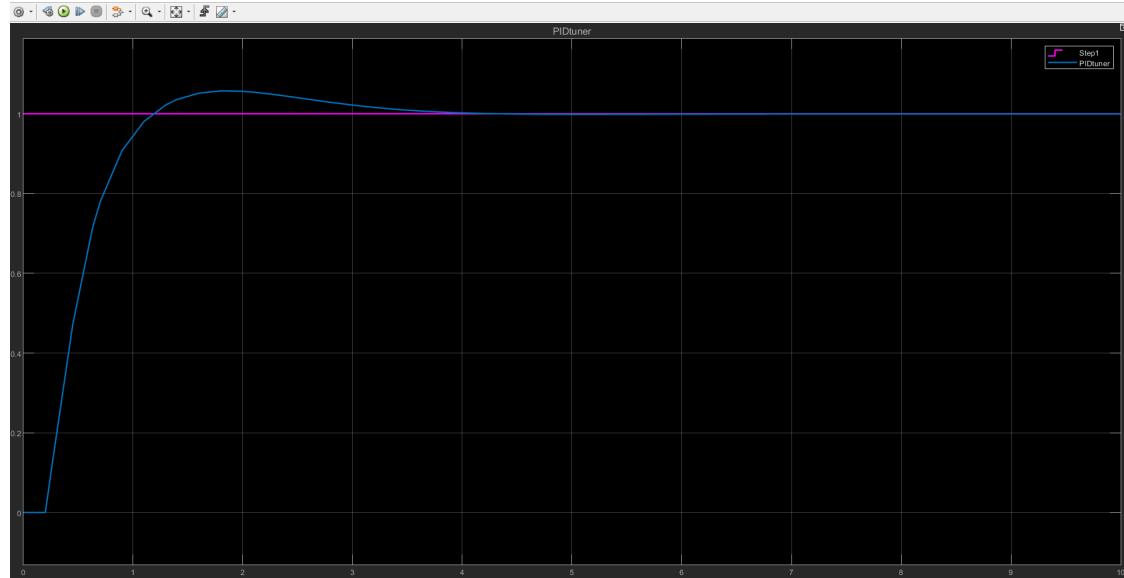
تصویر 7 - پاسخ پله کنترل شده توسط PIDtuner  
پارامتر های زیر نیز برای کنترلر بدست می آیند.

Controller Parameters		
	Tuned	Block
P	1.619	1.619
I	2.3057	2.3057
D	0.096355	0.096355
N	3.7809	3.7809

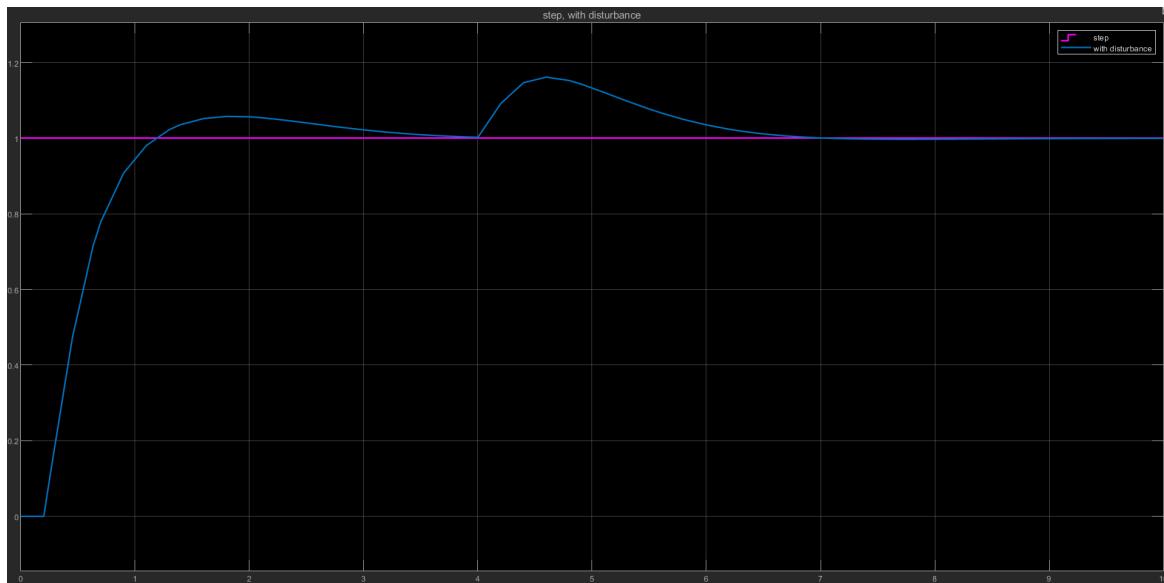
  

Performance and Robustness		
	Tuned	Block
Rise time	0.636 seconds	0.636 seconds
Settling time	3.09 seconds	3.09 seconds
Overshoot	5.91 %	5.91 %
Peak	1.06	1.06
Gain margin	12.5 dB @ 8.09 rad/s	12.5 dB @ 8.09 rad/s
Phase margin	64.5 deg @ 1.78 rad/s	64.5 deg @ 1.78 rad/s
Closed-loop stability	Stable	Stable

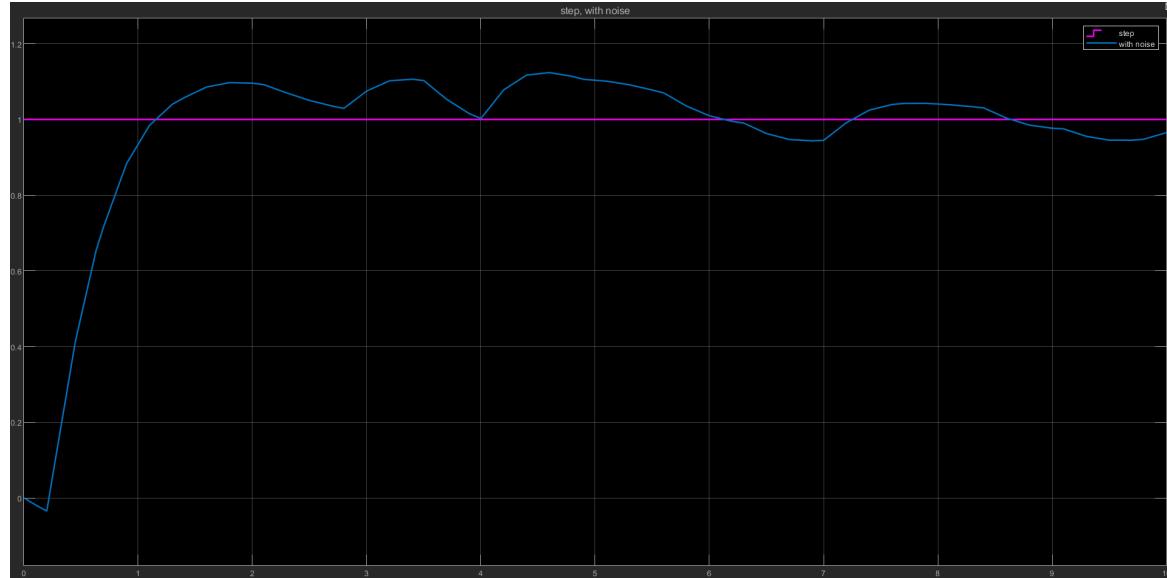
تصویر 8 - پارامتر های بدست آمده با PIDtuner  
حال دوباره در هر سه حالت خروجی میگیریم.



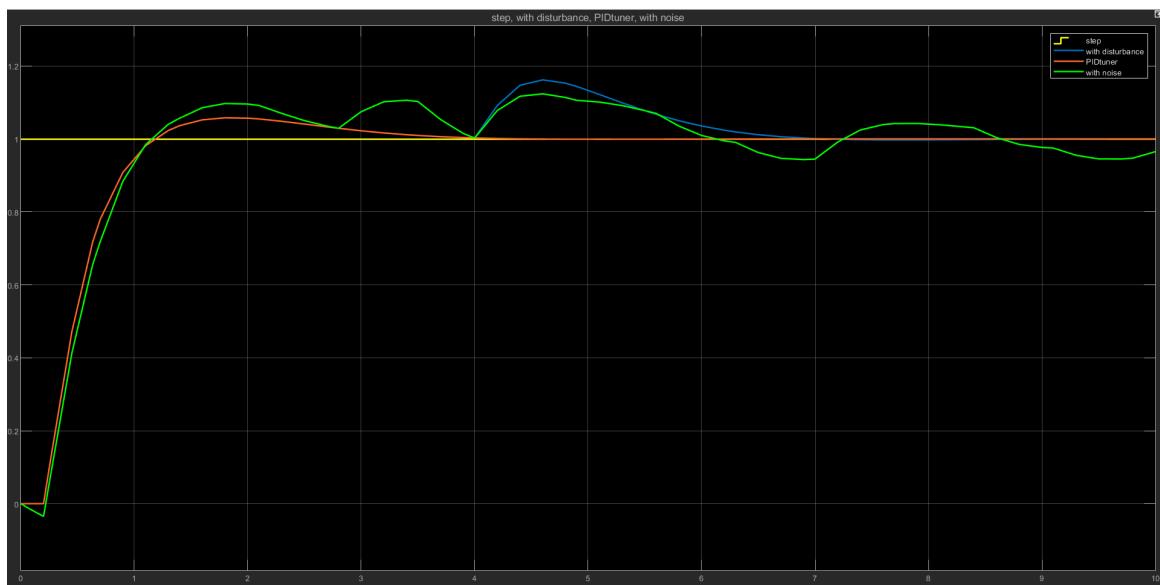
تصویر ۹ - پاسخ پله سیستم با تاخیر



تصویر ۱۰ - پاسخ پله سیستم با اغتشاش



تصویر 11 - پاسخ پله با نویز



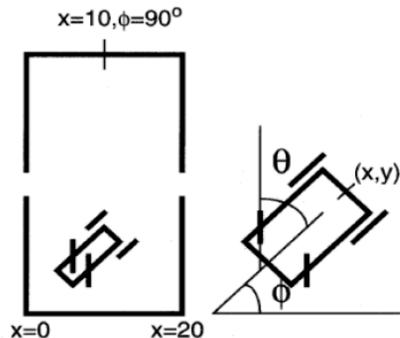
تصویر 12 - مقایسه پاسخ های پله

همانطور که مشاهده میشود باز هم پاسخ خوبی به ازای PIDtuner برای تمام بخش ها بجز سیستم با نویز بدست آورده ایم. همچنین این کنترل کننده اورショット بسیار کمتر و بهتری از زیگلر نیکولز دارد که این اتفاق برای ما مطلوب است. زمان نشست با این کنترل کننده بیشتر شده است که چون اورショット خیلی کمتر است برای ما مطلوب تر است.



## پرسش دوم

حرکت رو به عقب ماشین برای پارک یک مسئله غیرخطی است که در شکل زیر نمایش داده شده است:



مکان ماشین به وسیله سه متغیر  $(x, y, \phi)$  تعیین می‌شود. جایی که  $\phi$  زاویه ماشین نسبت به محور افق می‌باشد. کنترل ماشین از طریق زاویه فرمان  $\theta$  صورت می‌گیرد. فقط حرکت رو به عقب مجاز است. ماشین در هر مرحله با یک تغییر به مکان ثابت می‌رود. برای سادگی فرض شده که فضای کافی بین ماشین و محل پارک وجود دارد به نحوی که لازم  $y$  را به عنوان یک متغیر حالت در نظر بگیریم. هدف از طراحی کنترل این می‌باشد که ورودی‌های آن  $(x, \phi)$  و خروجی آن  $\theta$  بوده به نحوی که حالت نهایی  $x = 0$  و  $\phi = 90^\circ$  می‌باشد:

$$x(k+1) = x(k) + \cos(\phi(k) + \theta(k)) + \sin(\theta(k)) \sin(\phi(k))$$

$$y(k+1) = y(k) + \cos(\phi(k) + \theta(k)) - \sin(\theta(k)) \sin(\phi(k))$$

$$\phi(k+1) = \phi(k) - \left[ \frac{2 \sin(\theta(k))}{b} \right]$$

$$b = 2$$



## پرسش سوم

دو سیستم غیرخطی در جدول 1 برای این سری از تمرین‌ها انتخاب شده‌اند:

شماره	سیستم	تعداد نمونه	ورودی / خروجی
1	Ball and Beam	1000	یک / یک
2	Model of a steam generator at Abbot power plant in Champaign IL	9600	چهار / چهار

این دو سیستم را از این [ویسات](#) دانلود کنید و با دقت قسمت description را مطالعه نمایید. هر دو سیستم را با ANFIS شناسایی کرده و تحلیل کنید.

### 1 سیستم

در توضیحات این دیتا در سایت ذکر شده است که زمان نمونهبرداری برابر با 0.1 ثانیه است. این دیتابست شامل 1000 نمونه است. زاویه‌ی پرتو (beam angle) تنها ورودی و موقعیت توپ (ball position) تنها خروجی این سیستم می‌باشد. در ادامه به شناسایی سیستم با ANFIS می‌پردازیم. ابتدا داده‌ها از فایل اکسل بارگذاری می‌شوند:

```
data = readtable('BallBeamData.xlsx');
```

```
input_data = data.Input;
```

```
output_data = data.Output;
```

سپس داده‌ها به دو بخش آموزشی و آزمایشی تقسیم می‌شوند:

```
random_indices = randperm(num_samples);
train_ratio = 0.8;
num_train = round(train_ratio * num_samples);
train_indices = random_indices(1:num_train);
test_indices = random_indices(num_train+1:end);
```

در اینجا 80 درصد از داده‌ها برای آموزش و 20 درصد باقی‌مانده برای آزمایش در نظر گرفته می‌شود.

در مرحله بعد یک سیستم فازی با استفاده از تابع genfis2 ایجاد می‌شود که در آن ورودی‌ها و خروجی‌های آموزشی برای تولید مدل فازی استفاده می‌شوند:

```
fis = genfis2(input_train, output_train, 0.2);
```

genfis2 با استفاده از الگوریتم‌های خوشبندی، توابع عضویت Gaussian برای هر ورودی تولید می‌کند و سپس سیستم فازی Sugeno را بر اساس این توابع



عضویت و داده‌ها آموزش می‌دهد. این تابع برای تنظیم خودکار پارامترهای مدل فازی استفاده می‌شود و نیازی به تنظیم دستی ندارد.

: پارامتر تنظیمی که کنترل می‌کند که چقدر باید توابع عضویت (membership functions) گسترش یابند. این پارامتر تأثیر زیادی بر دقیقیت مدل دارد. در این کد از مقدار 0.2 استفاده شده است.

```
MaxEpoch = 200;
```

```
ErrorGoal = 0;
```

```
InitialStepSize = 0.01;
```

```
StepSizeDecreaseRate = 0.9;
```

```
StepSizeIncreaseRate = 1.1;
```

```
TrainOptions = [MaxEpoch, ErrorGoal, InitialStepSize, StepSizeDecreaseRate, StepSizeIncreaseRate];
```

```
DisplayInfo = true;
```

```
DisplayError = true;
```

```
DisplayStepSize = true;
```

```
DisplayFinalResult = true;
```

```
DisplayOptions = [DisplayInfo, DisplayError, DisplayStepSize, DisplayFinalResult];
```

```
OptimizationMethod = 1;
```

```
[fis, trainError] = anfis(training_data, fis, TrainOptions, DisplayOptions, [], OptimizationMethod);
```

- تعداد دوره‌های آموزش (MaxEpoch = 200): مدل حداقل 200 بار آموزش داده می‌شود.
- هدف خطای (ErrorGoal = 0): مدل تلاش می‌کند تا خطای را به صفر برسوند.
- اندازه گام اولیه (InitialStepSize = 0.01): مقدار اولیه تغییرات پارامترها در طول آموزش.
- نرخ تغییر اندازه گام: وقتی مدل بهینه می‌شود، اندازه گام کم می‌شود (StepSizeDecreaseRate = 0.9) و وقتی نیاز به تغییرات بیشتر داریم، اندازه گام زیاد می‌شود (StepSizeIncreaseRate = 1.1).
- نمایش اطلاعات: در طول آموزش اطلاعاتی مثل خطاهای و اندازه گام نشون داده بشود.



- تنظیمات روش بهینه‌سازی: روش LSE (Least Squares Estimation) برای بهینه‌سازی استفاده خواهد شد. این روش به طور خودکار وزن‌ها و پارامترهای مدل فازی را تنظیم می‌کند.  
در نهایت با استفاده از anfis، مدل آموزش می‌بینند.

RMSE (Root Mean Squared Error) معیاری است برای سنجش دقت پیش‌بینی مدل. این مقدار جذر میانگین مربعات اختلاف‌ها (خطاهای) بین مقادیر واقعی و پیش‌بینی شده است. از این معیار برای بررسی مدل استفاده می‌کنیم.

- کمتر از 0.5: مدل بسیار دقیق است و پیش‌بینی‌ها تقریباً درست هستند.
- بین 0.5 تا 1: مدل دقت متوسطی دارد. پیش‌بینی‌ها به اندازه کافی دقیق نیستند ولی قابل قبولند.
- بیشتر از 1: مدل دقت کمی دارد و پیش‌بینی‌ها با خطای قابل توجهی همراه هستند.

در کل، هر مقدار کمتر RMSE نشان‌دهنده مدل دقیق‌تر است و هر چه بیشتر باشد، مدل دقت کمتری دارد. RMSE خطاهای رو در همان واحد داده‌ها نشون میده، بنابراین راحت‌تر می‌شه تفاوت‌ها رو درک کرد.

```
function PlotResults(targets, outputs, Name)
```

```
    errors = targets - outputs;  
  
    RMSE = sqrt(mean(errors(:).^2));  
  
    error_mean = mean(errors(:));  
  
    error_std = std(errors(:));
```

```
    figure;  
  
    plot(targets, 'k');  
    hold on;  
  
    plot(outputs, 'r--');  
  
    legend('Target', 'Output');  
  
    title([Name ': Target vs Output']);
```

```
    figure;  
  
    plot(errors);  
  
    legend('Error');  
  
    title([Name ': Errors']);
```

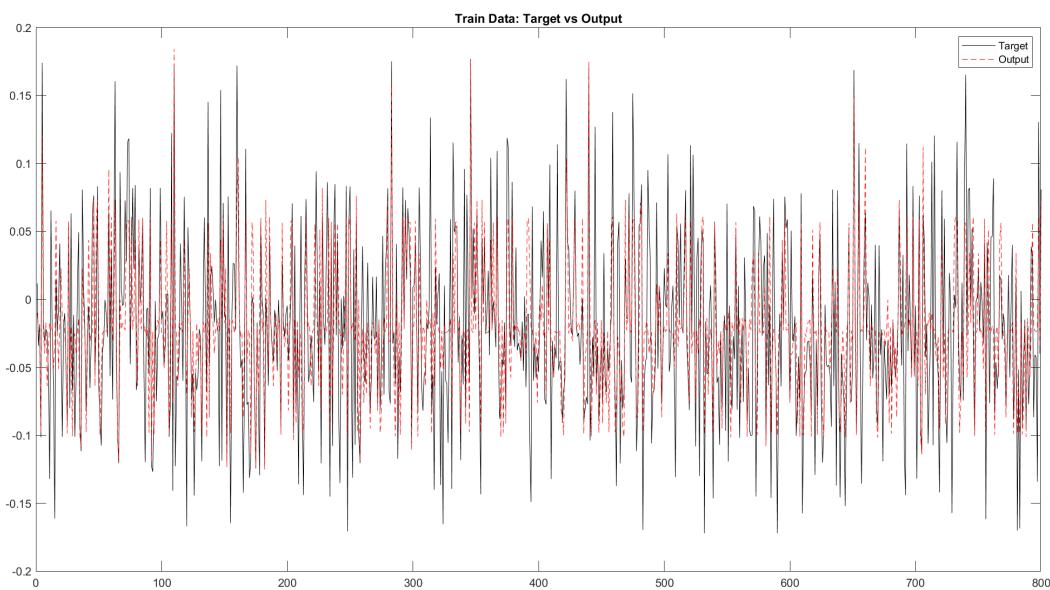


```
xlabel('Sample Index');  
ylabel('Error');  
  
figure;  
histfit(errors);  
title([Name ': Error Distribution']);  
xlabel('Error');  
ylabel('Frequency');  
legend('Error Histogram', 'Fitted Curve');  
end
```

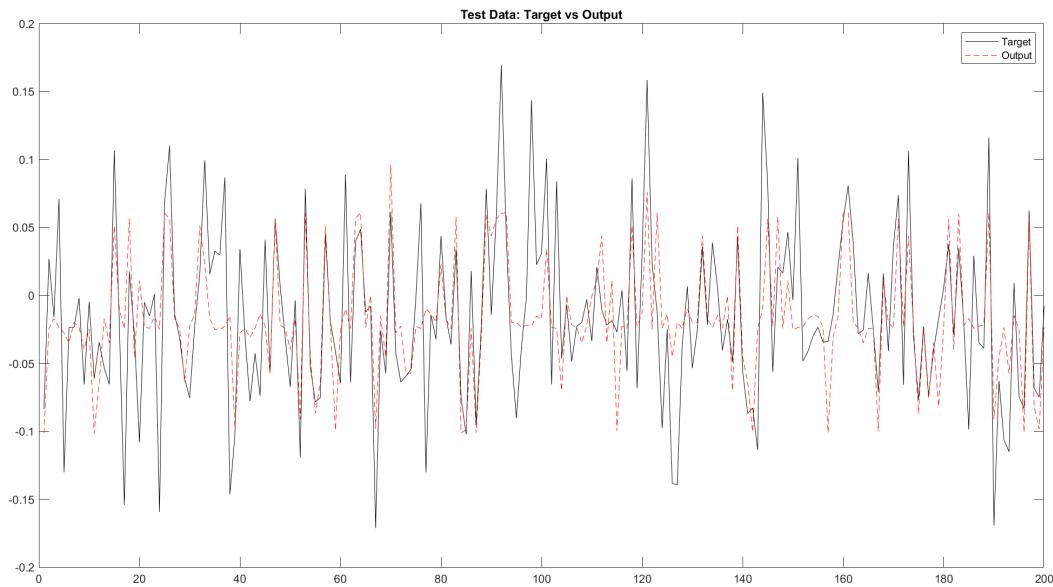
این تابع PlotResults برای نمایش نتایج پیش‌بینی مدل و بررسی خطاهای استفاده می‌شود. به طور خلاصه:

- محاسبه خطاهای: تفاوت بین مقادیر واقعی (targets) و پیش‌بینی شده (outputs) محاسبه می‌شود.
- نمودار 1: مقادیر واقعی و پیش‌بینی شده را در یک نمودار مقایسه می‌کنند.
- نمودار 2: خطاهای را برای هر نمونه به صورت جداگانه نمایش میدهند.
- نمودار 3: توزیع خطاهای را با استفاده از هیستوگرام و منحنی مناسب نمایش میدهند.

در این بخش نتایج بدست آمده را مشاهده می‌کنیم:

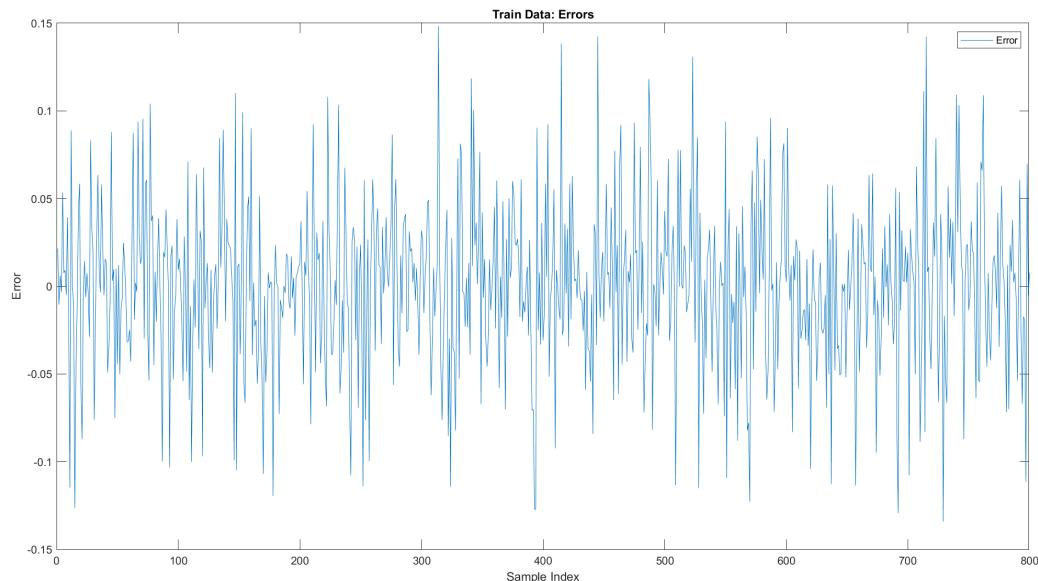


تصویر 13 - مقایسه خروجی واقعی و پیش‌بینی شده در آموزش

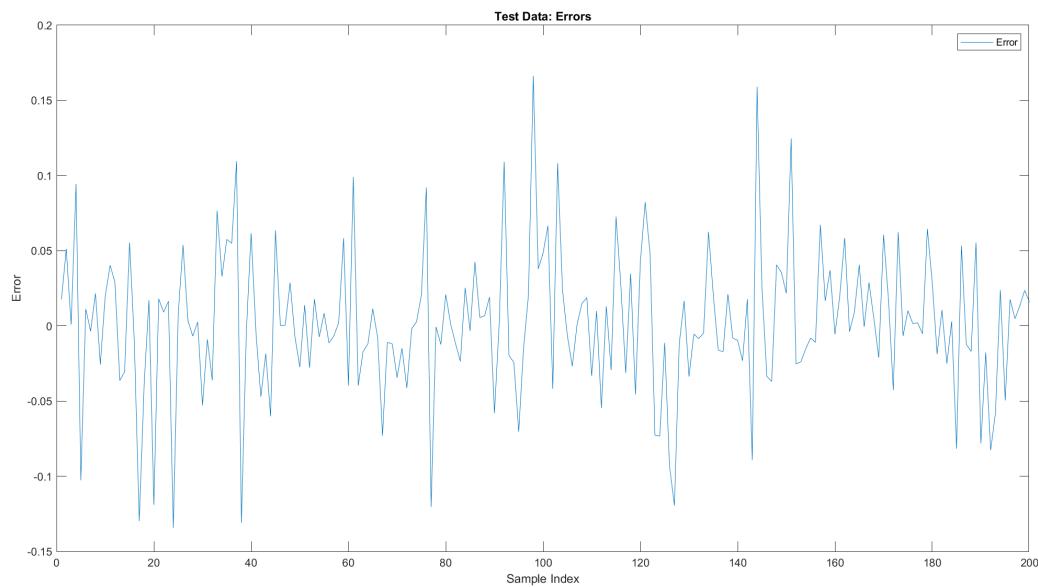


تصویر ۱۴ - مقایسه خروجی واقعی و پیشینی شده در تست

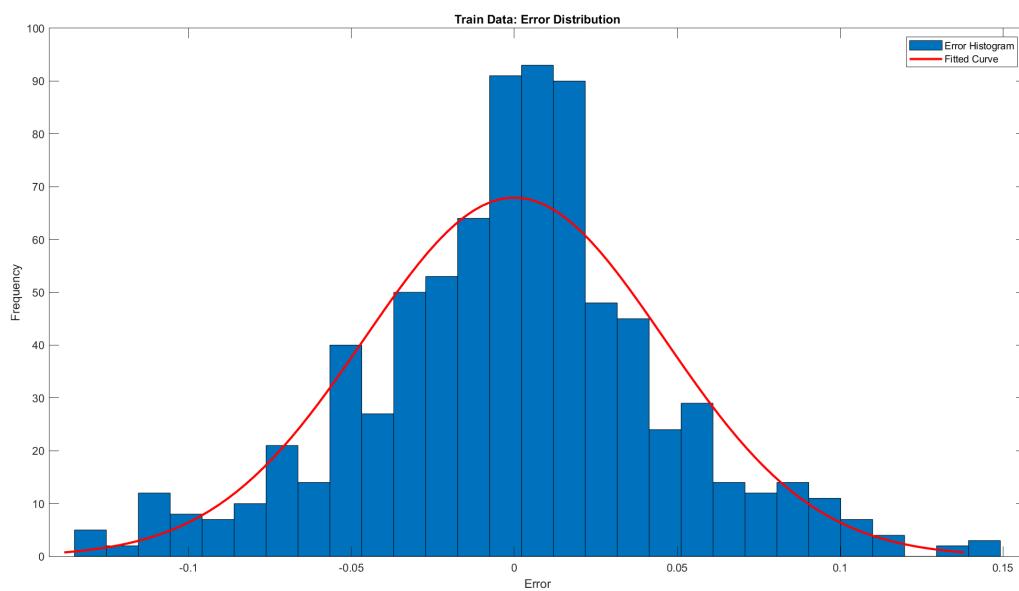
مشاهده میشود که خروجی بصورت نسبی در حال تعیق هدف است اما خیلی منطبق نیست که البته مقادیر بسیار کوچک هستند و قابل قبول است.



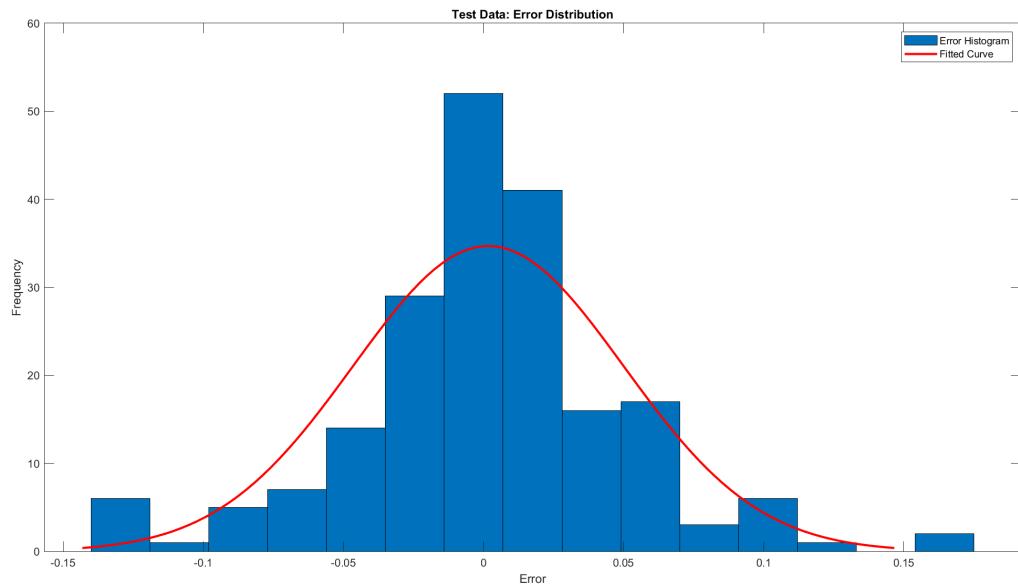
تصویر ۱۵ - خطای آموزش



تصویر ۱۶ - خطای تست

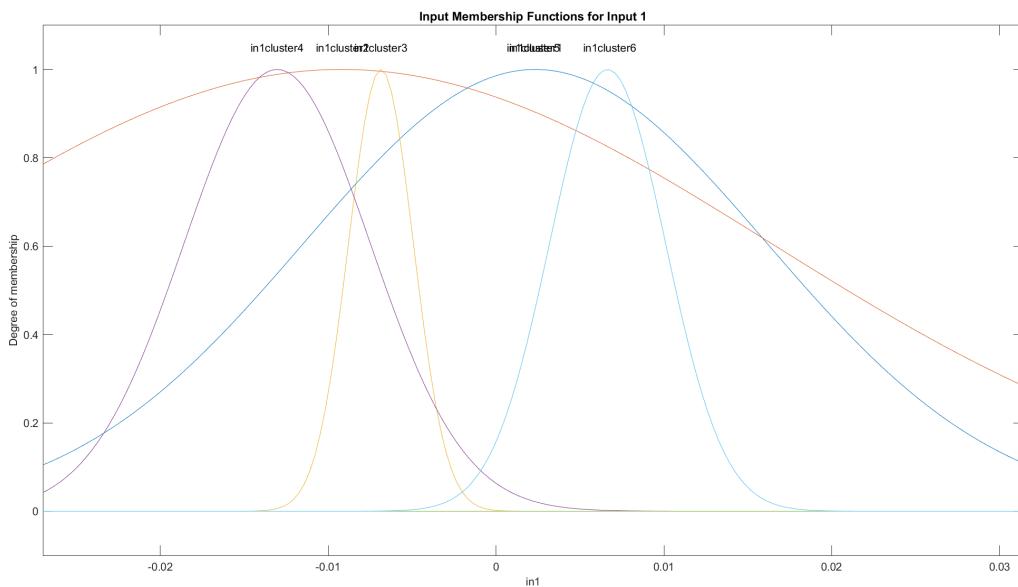


تصویر ۱۷ - توزیع خطای آموزش



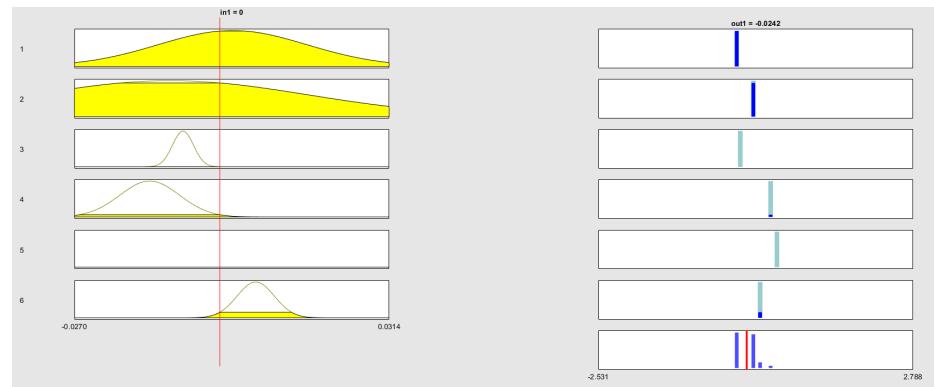
تصویر 18 - توزیع خطای تست

مشاهده میشود که چون داده های بسیار کوچک و نزدیک به صفر هستند در همان بازه کوچک و نزدیک به صفر نیز بیشترین خطا را داریم.



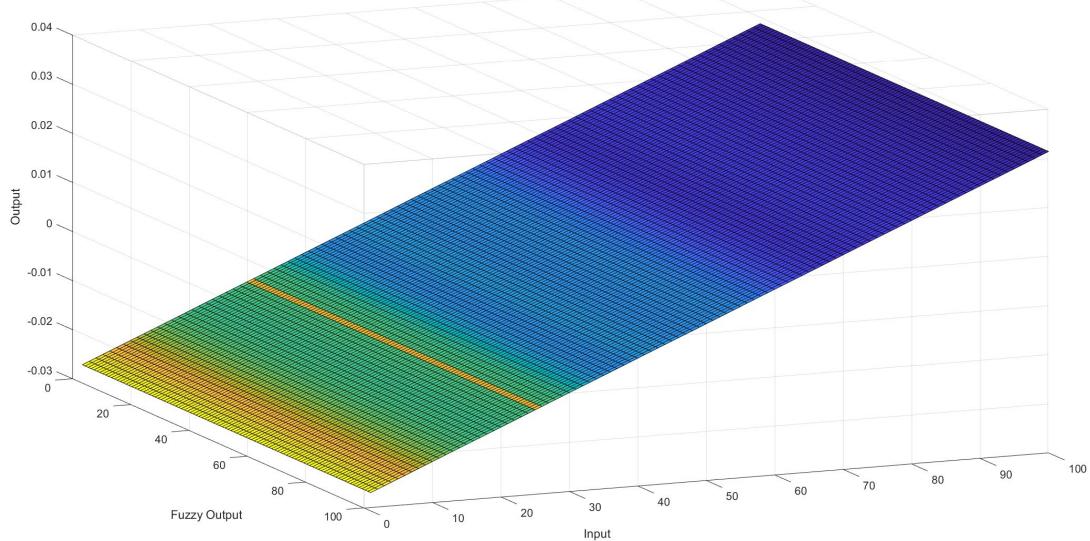
تصویر 19 - توابع تعلق ورودی

5 تابع تعلق گوسی تعریف و تنظیم شده اند.

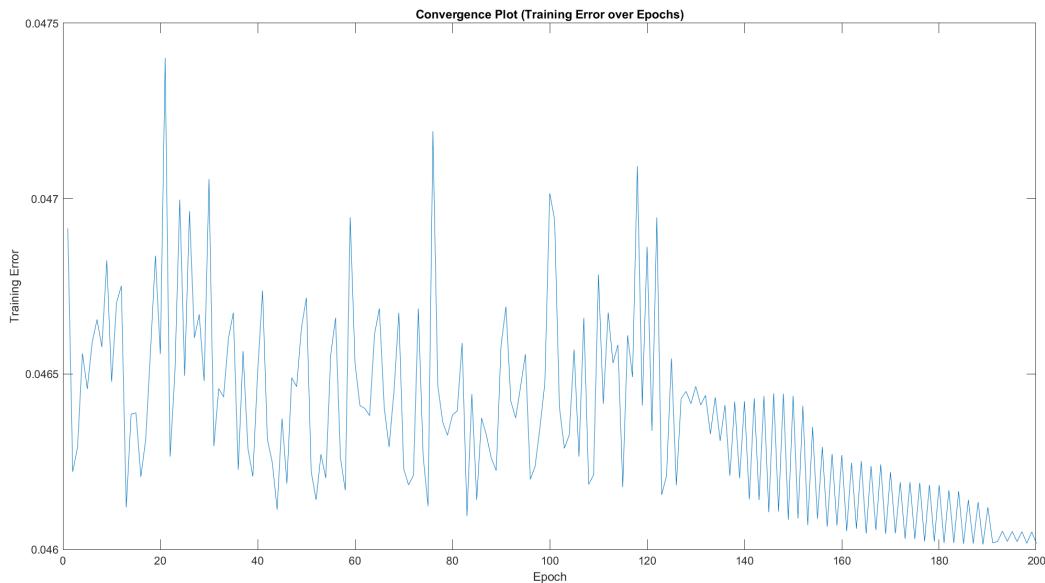


تصویر 20 - قوانین فازی مدل ANFIS

Surface Plot of Fuzzy Inference System



تصویر 21 - نمودار سطحی خروجی مدل فازی



تصویر 22 - حطا به ازای افزایش epoch

مشاهده میشود که به ازای افزایش تعداد آموزش مدل دائم در حال بهتر شدن است.

و در نهایت مقدار RMSE در آموزش و تست:

Train RMSE: 0.046014

Test RMSE: 0.048205

این مقادیر و خروجی ها به ما نشان میدهد که مدل ما خیلی خوب عمل میکند و توانسته است این سیستم غیرخطی را فقط با 5 تابع تعلق به خوبی پیشنبینی کند البته که 1000 نمونه دیتا داشتیم که کمک بزرگی برای این اتفاق بود.

## سیستم 2

این مجموعه داده از مدل یک ژنراتور بخار در نیروگاه Abbott در Champaign، ایالت ایلینوی تهیه شده است. مدل مذکور در مقاله Pellegrineti شرح داده شده است. هر 3 ثانیه این سیستم نمونه برداری شده است و در مجموع 9600 نمونه جمع آوری شده است.

ورودی ها:

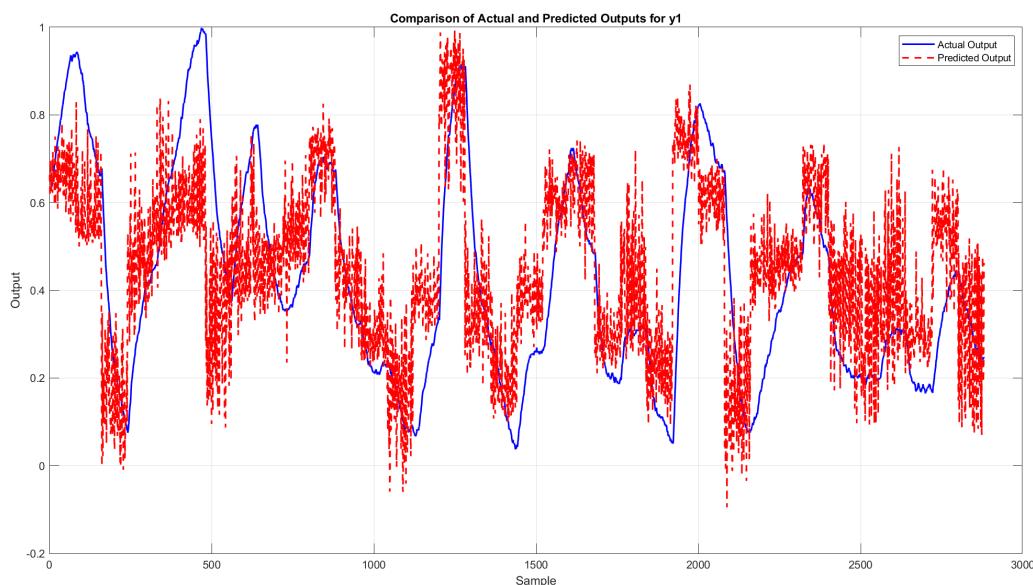
- u1: سوخت (با مقیاس 0 تا 1)
- u2: هوا (با مقیاس 0 تا 1)
- u3: سطح مرجع (بر حسب اینچ)
- u4: اغتشاش، تعریف شده بر اساس سطح بار



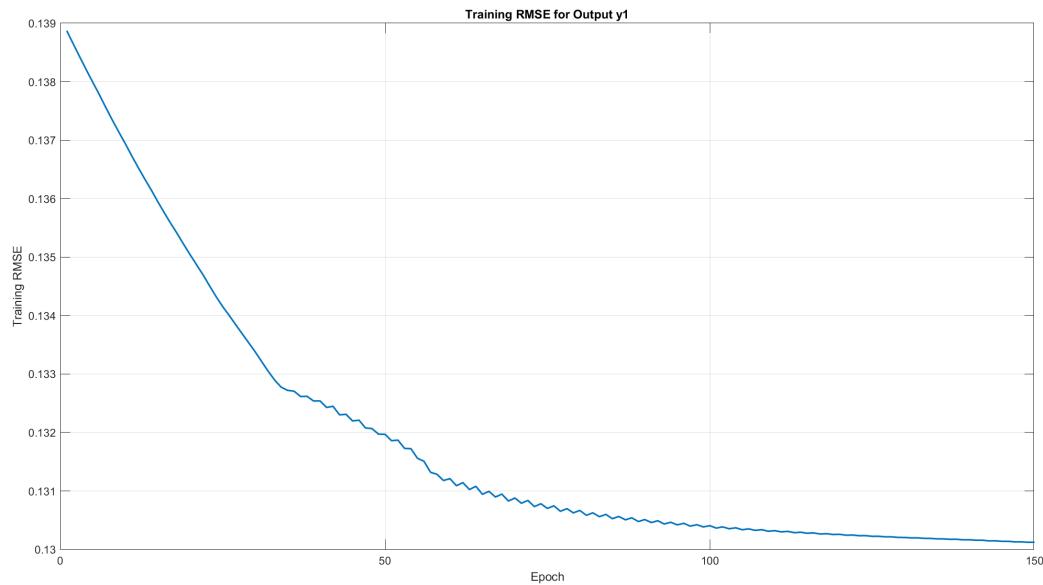
## خروجی‌ها:

- y1: فشار درام (بر حسب PSI)
- y2: اکسیژن اضافی در گازهای خروجی (درصد)
- y3: سطح آب در درام
- y4: جریان بخار (کیلوگرم بر ثانیه)

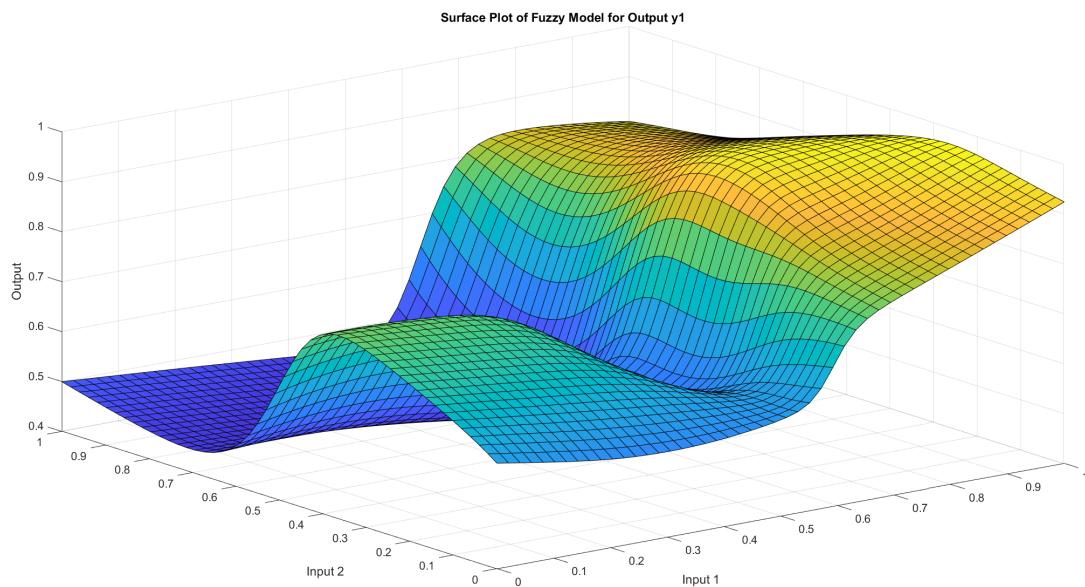
نحوه کدنویسی و آموزش این سیستم را نیز مانند سیستم قبلی انجام میدهیم با این تفاوت که با خاطر بیشتر شدن ابعاد ورودی و تعداد نمونه ها از 150 ایپاک و  $\text{spread} = 0.5$  استفاده میکنیم تا مدل سریع تر آموزش ببیند. همچنین داده ها را ابتدا نرمالایز میکنیم. در ادامه نتایج مشاهده میشود:



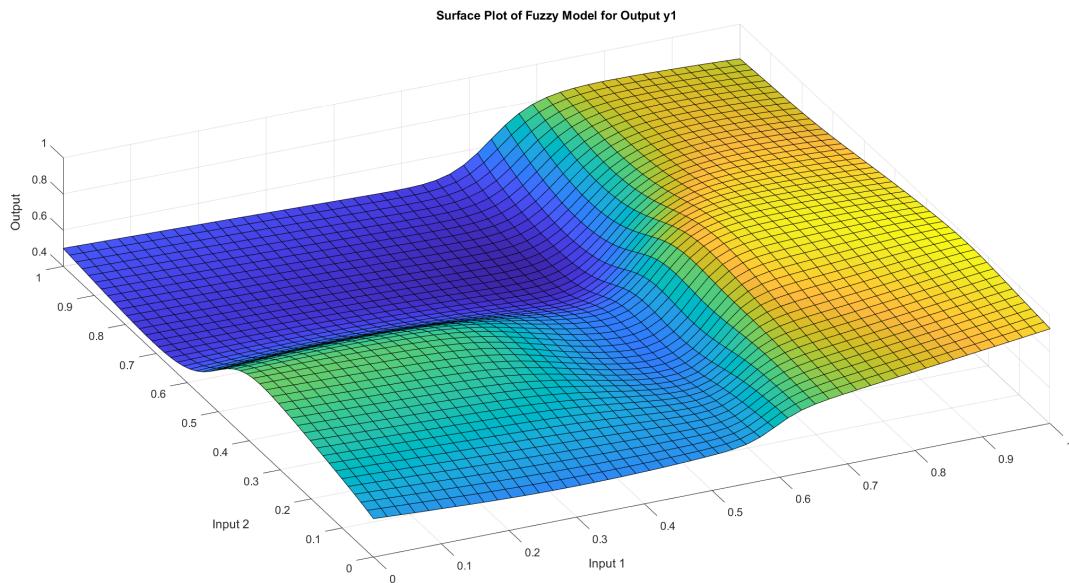
تصویر 23 - فشار درام واقعی و پیش‌بینی شده در تست



تصویر 24 - RMSE فشار درام به ازای تعداد ایپاک



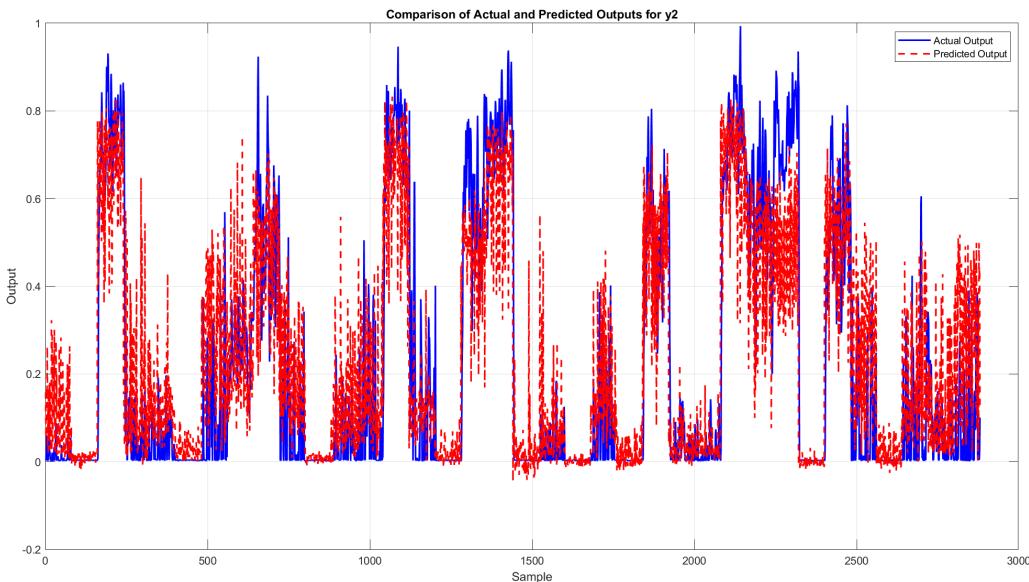
تصویر 25 - فازی سرفیس فشار درام



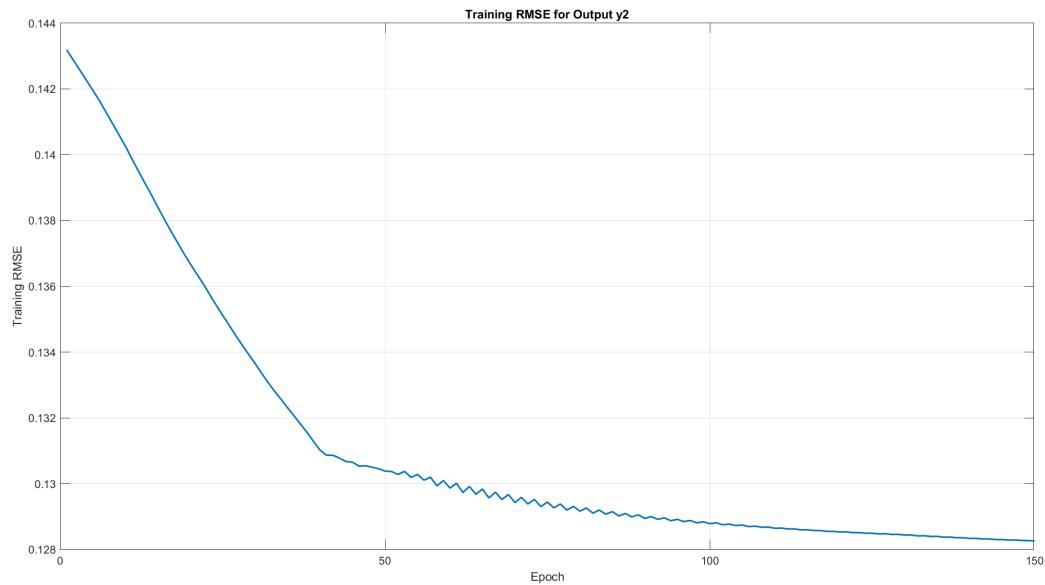
تصویر 26 - فازی سرفیس فشار درام

Output  $y_1$  Test RMSE: 0.19339

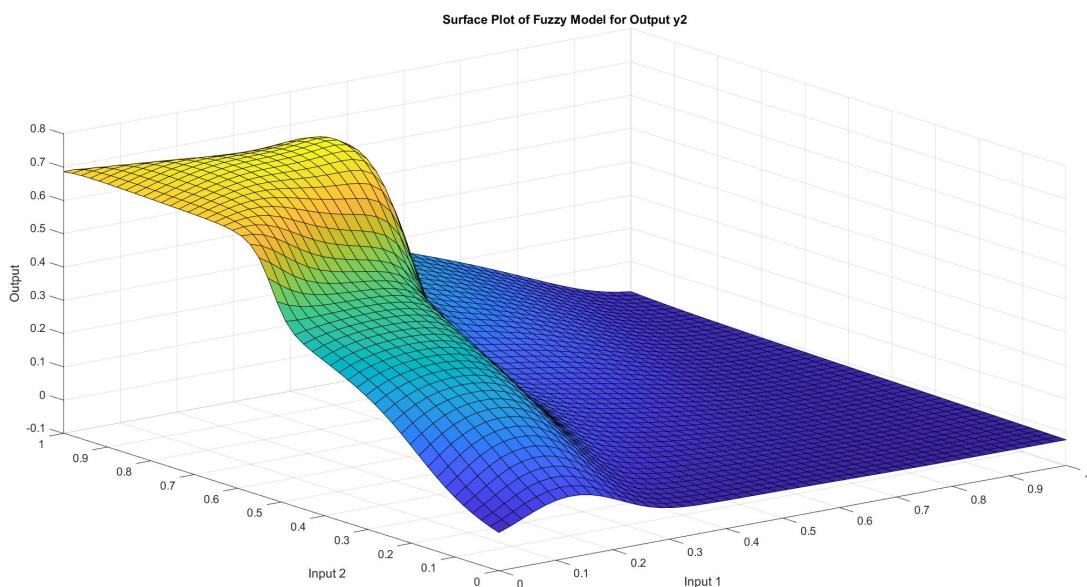
مشاهده میشود که نسبت به سیستم قبلی خطای بیشتری داریم اما باز هم این مقدار نشان میدهد که مدل آموزش دیده کاملا برای خروجی فشار درام قابل قبول است و خوب عمل میکند.



تصویر 27 - اکسیژن اضافی واقعی و پیشینی شده در تست



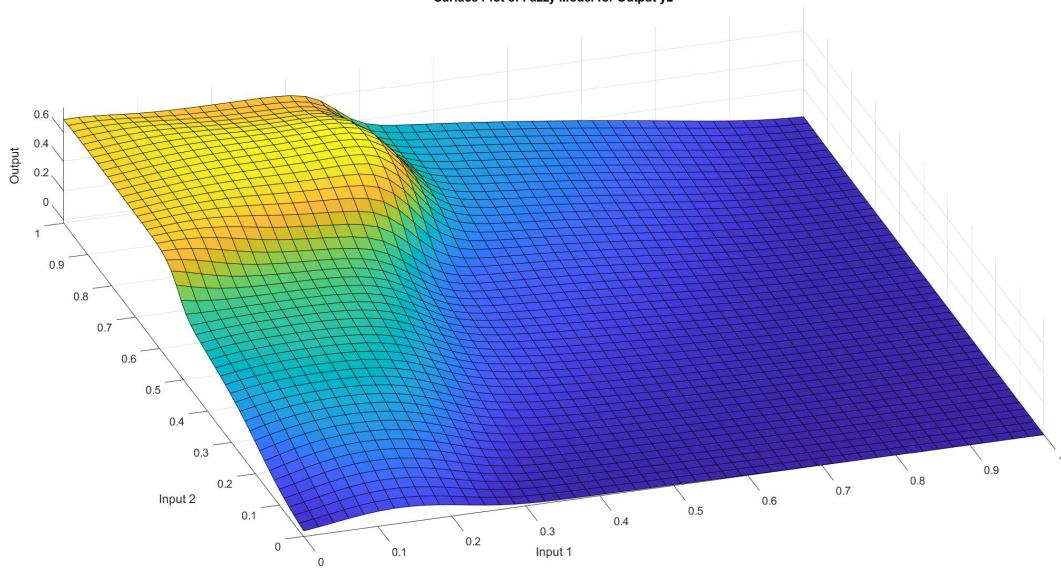
تصویر 28 - RMSE /کسیژن اضافی به ازای تعداد ایپاک



تصویر 29 - فازی سرفیس اکسیژن اضافی



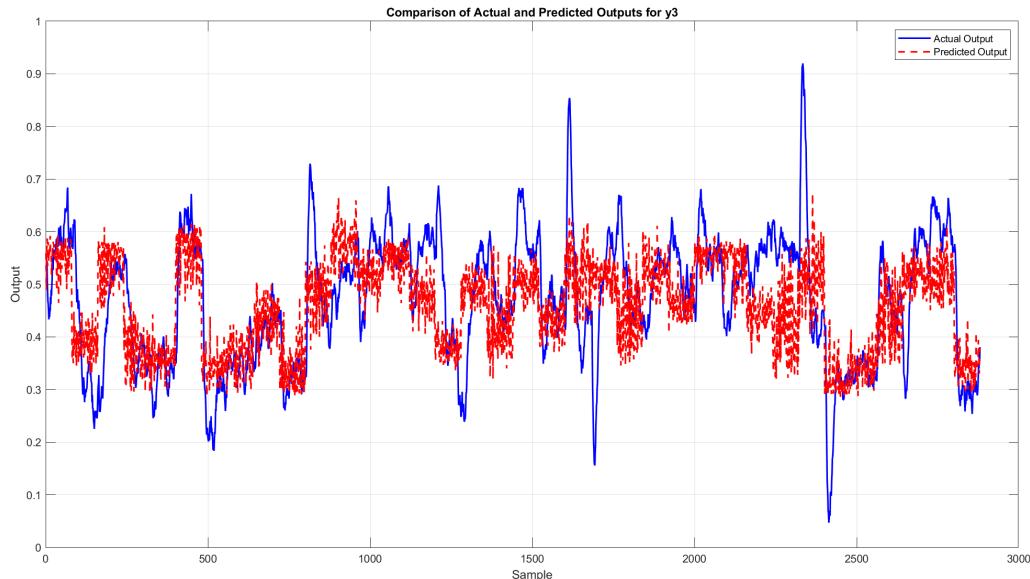
Surface Plot of Fuzzy Model for Output y2



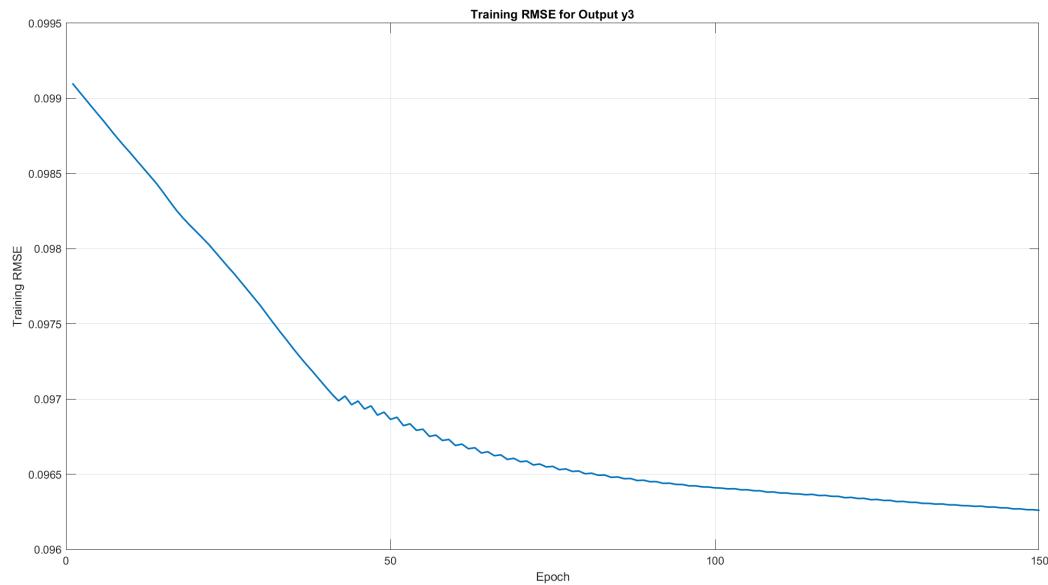
تصویر 30 - فازی سرفیس اکسیژن اضافی

Output y2 Test RMSE: 0.17062

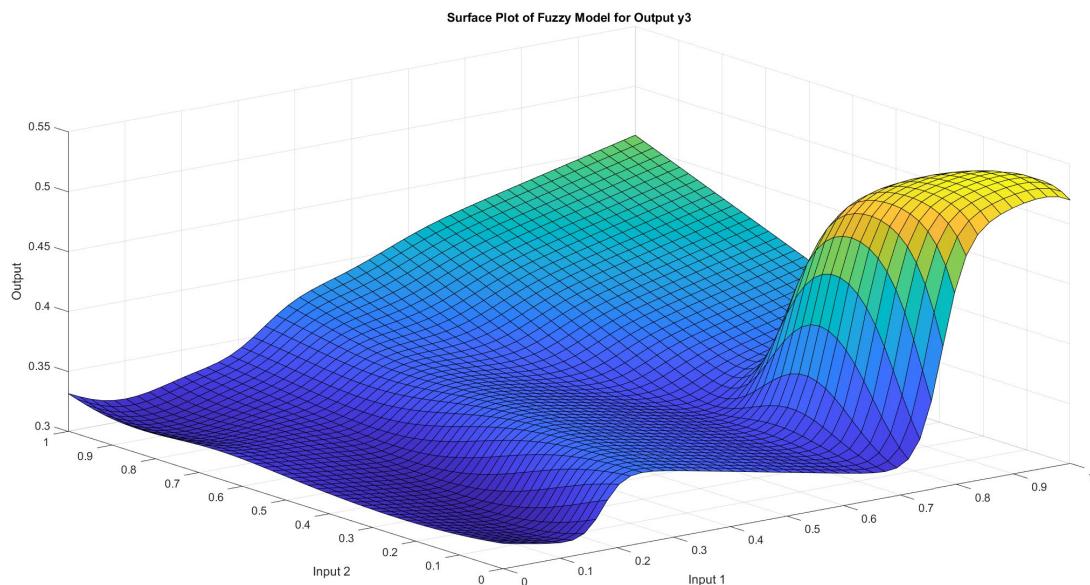
از لحاظ RMSE خروجی اکسیژن اضافی نیز مانند خروجی فشار درام نتیجه داده است و مدل با همان دقت پیش‌بینی می‌کند.



تصویر 31 - سطح آب واقعی و پیش‌بینی شده در تست



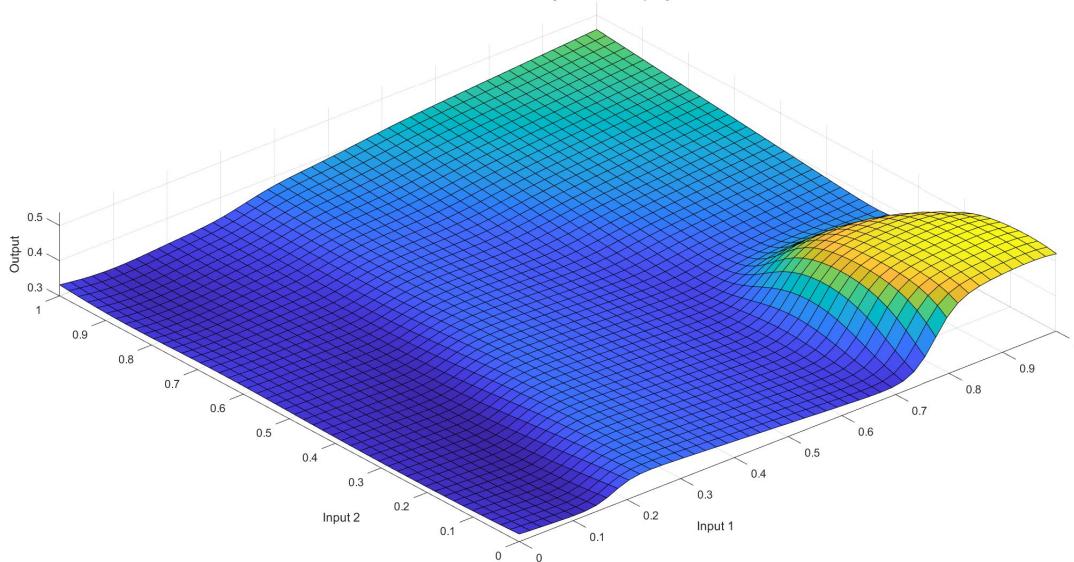
تصویر 32 - سطح RMSE - آب به ازای تعداد ایپاک



تصویر 33 - فازی سرفیس سطح آب



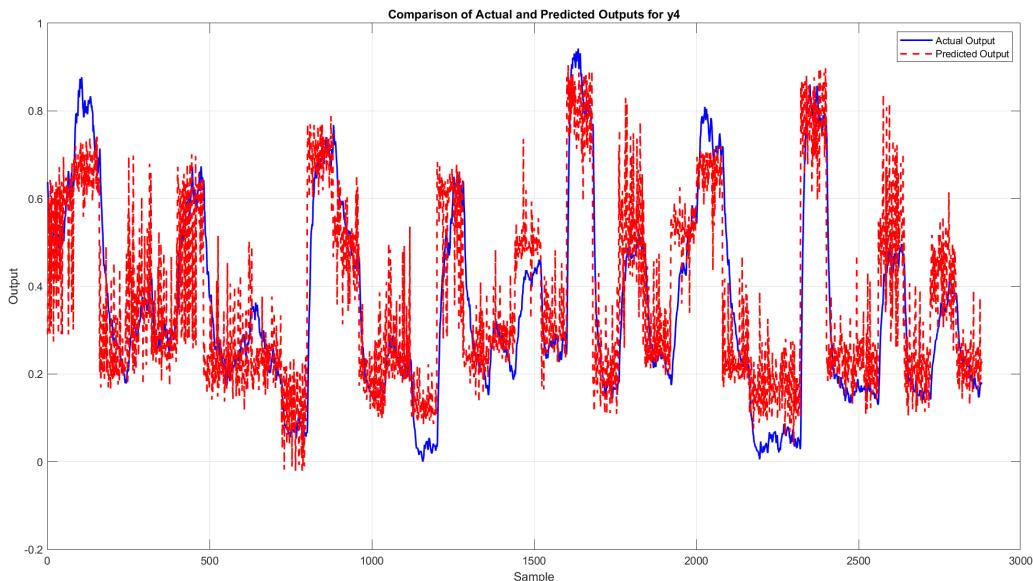
Surface Plot of Fuzzy Model for Output y3



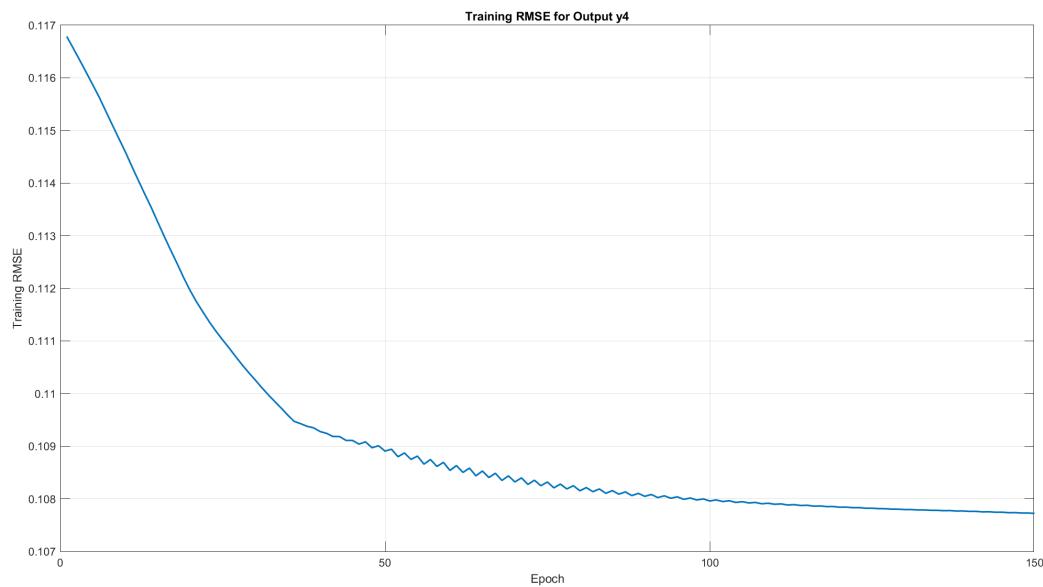
تصویر 34 - فازی سرفیس سطح اب

Output y3 Test RMSE: 0.10427

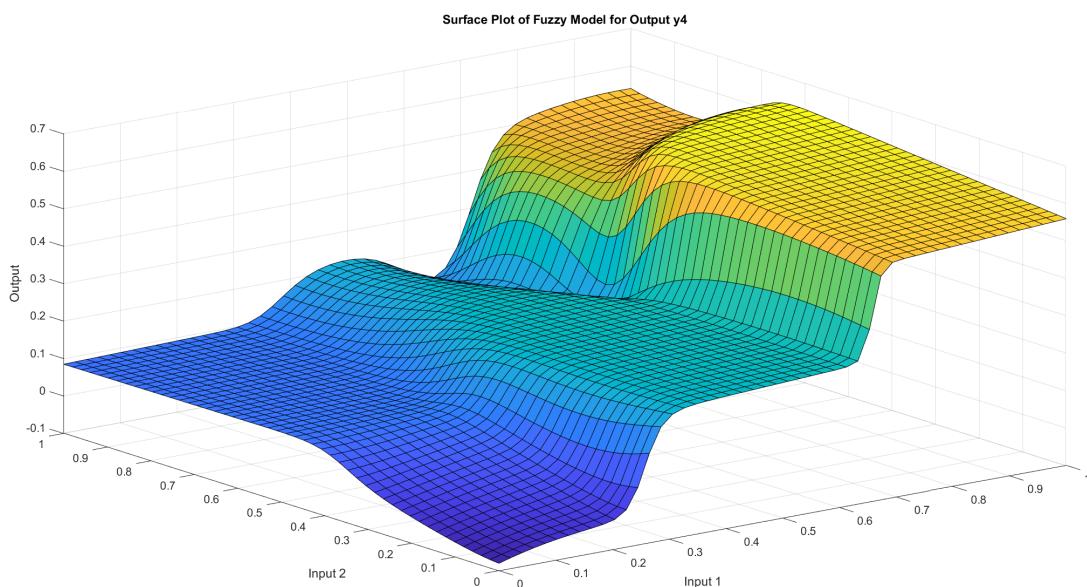
مشاهده میشود که خروجی سطح آب کمی بهتر از دو خروجی قبلی پیشビینی شده اند.



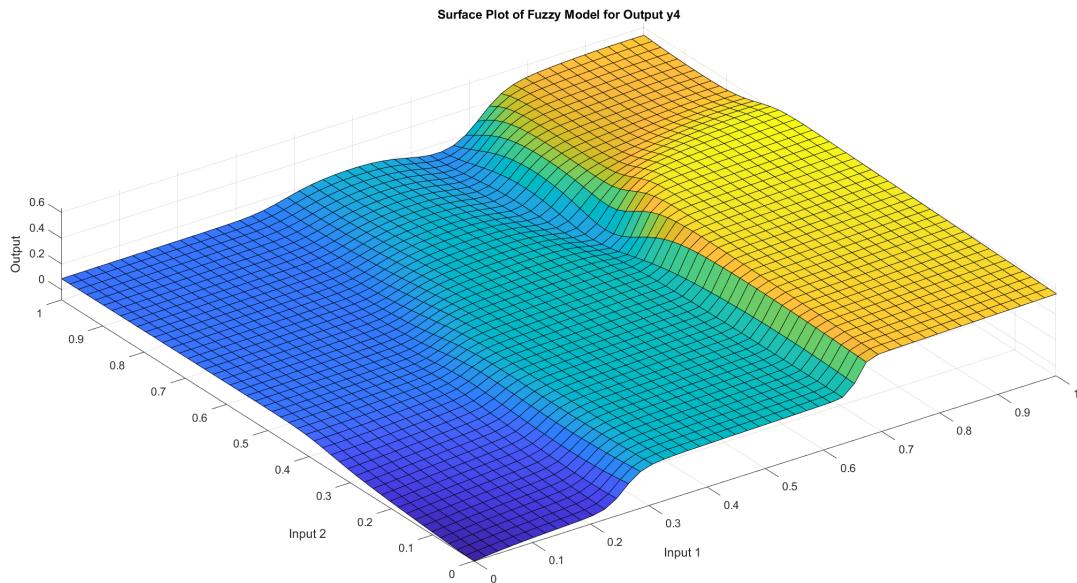
تصویر 35 - سطح بخار واقعی و پیشビینی شده در تست



تصویر 36 - RMSE بخار آب به ازای تعداد ایپاک



تصویر 37 - فازی سرفیس بخار آب



تصویر 38 - فازی سرفیس بخار آب

Output y4 Test RMSE: 0.13716

بخار آب نیز دقیقی به مانند دو خروجی اول دارد و بصورت کلی هر 4 مدل دقت قابل قبولی دارند و از آنجایی که این یک سیستم غیرخطی با تاثیر 4 ورودی متفاوت بود میتوان نتیجه گرفت که شناسایی با ANFIS توانسته است تا حدود قابل قبولی همه خروجی ها را پیش‌بینی کند که نشان دهنده قدرت فازی در این زمینه است.



## پرسش چهارم

فرض کنید یک سیستم با معادله دیفرانسیل زیر برازیده شده است، که قرار است توسط یک شناسایی فازی تحلیل شود:

$$y(k+1) = 0.3y(k) + 0.6y(k-1) + g(u(k))$$

که در آن تابع غیر خطی  $g(u)$  طبق رابطه زیر تعریف می‌شود:

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

هدف ما این است که عنصر غیرخطی شناسایی شده در رابطه  $y(k+1)$  را با استفاده از یک مدل فازی و به همراه الگوریتم انرژی کمترین مربعات برازیده کنیم. برای این کار، تابع مشابه آنچه در رابطه زیر آمده برازیده می‌شود:

$$f(x) = \frac{\sum_{l=1}^M g^l \left[ \prod_{i=1}^n \exp \left( -\left( \frac{x_i - x'_i}{\sigma'_i} \right)^2 \right) \right]}{\sum_{l=1}^M \left[ \prod_{i=1}^n \exp \left( -\left( \frac{x_i - x'_i}{\sigma'_i} \right)^2 \right) \right]}$$

با انجام مراحل زیر این سوال را حل می‌کنیم. ابتدا از کد زیر استفاده می‌کنیم:

$M = 7$ ; % Number of membership functions (Based on 1st step of fuzzy system design)

$\text{num\_training} = 200$ ; % Number of training samples

$\text{total\_num} = 700$ ;

$\text{landa} = 0.1$ ; % A constant stepsize

○  $M = 7$ : تعداد توابع عضویت (membership functions) که برای دسته‌بندی ورودی‌ها استفاده می‌شود.

○  $\text{num\_training} = 200$ : تعداد نمونه‌های آموزشی برای آموزش مدل.

○  $\text{total\_num} = 700$ : تعداد کل داده‌ها (آموزشی و تست).

○  $\text{landa} = 0.1$ : اندازه گام برای به روزرسانی مدل در هر مرحله یادگیری.

در ابتدا، چندین ماتریس و متغیر برای ذخیره‌سازی مقادیر مختلف سیستم فازی تعریف می‌شوند:

$x\_bar = \text{zeros}(\text{num\_training}, M);$

$g\_bar = \text{zeros}(\text{num\_training}, M);$

$\sigma = \text{zeros}(\text{num\_training}, M);$

$y = \text{zeros}(\text{total\_num}, 1);$

$u = \text{zeros}(\text{total\_num}, 1);$



```
x = zeros(total_num, 1);  
y_hat = zeros(total_num, 1);  
f_hat = zeros(total_num, 1);  
z = zeros(total_num, 1);  
g_u = zeros(total_num, 1);
```

توضیح هر خط به ترتیب:

- ذخیره مقادیر ورودی‌های آموزش برای توابع عضویت.
- ذخیره مقادیر مربوط به خروجی‌های فازی برای هر ورودی.
- ذخیره عرض توابع عضویت (spread) برای هر تابع عضویت.
- ذخیره مقادیر واقعی خروجی سیستم برای همه داده‌ها.
- ذخیره مقادیر ورودی برای همه داده‌ها.
- ذخیره ورودی‌های تصادفی برای تمامی داده‌ها.
- ذخیره خروجی پیش‌بینی شده توسط مدل فازی.
- ذخیره مقادیر پیش‌بینی شده خروجی مدل فازی برای داده‌ها.
- ذخیره مقادیر محاسبه شده برای هر تابع عضویت.
- ذخیره مقادیر واقعی خروجی سیستم برای هر ورودی (که به عنوان تابع واقعی استفاده می‌شود).

در این بخش، برخی از متغیرها و پارامترها مقداردهی اولیه می‌شوند:

```
u(1) = -1 + 2 * rand;
```

ورودی تصادفی اول ( $u(1)$ ) بین -1 و 1 تولید می‌شود.

```
y(1) = 0;
```

مقدار اولیه خروجی واقعی برابر با صفر تنظیم می‌شود.

```
g_u(1) = 0.6 * sin(pi * u(1)) + 0.3 * sin(3 * pi * u(1)) + 0.1 * sin(5 * pi * u(1));
```

خروجی واقعی سیستم بر اساس ورودی تصادفی ( $u(1)$ ) محاسبه می‌شود. این یک ترکیب سینوسی است که در مدل استفاده می‌شود.

```
f_hat(1) = g_u(1)
```

پیش‌بینی اولیه خروجی مدل برابر با خروجی واقعی تنظیم می‌شود.

حلقه آموزش:

از  $q = 2$  شروع می‌شود و تا تعداد نمونه‌های آموزشی ( $num\_training$ ) ادامه می‌یابد. در هر تکرار، ورودی تصادفی جدیدی تولید می‌شود و مدل با استفاده از این ورودی به روزرسانی می‌شود.



ورودی جدید ( $x(q)$ ) به صورت تصادفی بین -1 و 1 تولید می‌شود. خروجی واقعی مدل ( $g_u(q)$ ) با استفاده از ترکیب توابع سینوسی محاسبه می‌شود.

برای هر تابع عضویت، میزان تأثیر ورودی روی خروجی محاسبه می‌شود و در ماتریس  $Z$  ذخیره می‌شود. سپس از این مقادیر برای محاسبه خروجی پیش‌بینی شده مدل ( $f_{\hat{}}(q)$ ) استفاده می‌شود.

توابع عضویت ( $g_{\bar{}}$ ) و عرض توابع عضویت ( $\sigma$ ) با توجه به خطای پیش‌بینی مدل به روز می‌شوند. این به روزرسانی‌ها باعث می‌شود که مدل بهتر بتواند ورودی‌ها را پیش‌بینی کند.

خروجی واقعی سیستم ( $y(q+1)$ ) و پیش‌بینی شده آن ( $y_{\hat{}}(q+1)$ ) برای ورودی جدید محاسبه می‌شود.

```
% Training phase
```

```
for q = 2:num_training
```

```
    b = 0; a = 0;
```

```
    x(q) = -1 + 2 * rand; % Random input
```

```
    u(q) = x(q);
```

```
    g_u(q) = 0.6 * sin(pi * u(q)) + 0.3 * sin(3 * pi * u(q)) + 0.1 * sin(5 * pi * u(q));
```

```
Z = zeros(1, M);
```

```
for r = 1:M
```

```
    z(r) = exp(-((x(q) - x_bar(q, r)) / sigma(q, r))^2);
```

```
    Z(r) = z(r);
```

```
    b = b + z(r);
```

```
    a = a + g_bar(q, r) * z(r);
```

```
end
```

```
f_hat(q) = a / b; % Output approximation
```

```
g_bar(q + 1, :) = g_bar(q, :) + landa * Z * (g_u(q) - f_hat(q));
```

```
x_bar(q + 1, :) = x_bar(q, :);
```

```
sigma(q + 1, :) = sigma(q, :);
```



```
y(q + 1) = 0.3 * y(q) + 0.6 * y(q - 1) + g_u(q);  
y_hat(q + 1) = 0.3 * y(q) + 0.6 * y(q - 1) + f_hat(q);  
end
```

### حلقه تست:

از  $q = \text{num\_training}$  (اولین نمونه داده‌های تست) شروع شده و تا 700 (تعداد کل داده‌ها) ادامه می‌یابد. در این مرحله، مدل برای پیش‌بینی خروجی‌های جدید تست می‌شود.

ورودی جدید ( $x(q)$ ) به صورت یک دنباله سینوسی (با فرکانس خاص) تولید می‌شود. خروجی واقعی ( $g_u(q)$ ) برای ورودی جدید با استفاده از ترکیب توابع سینوسی محاسبه می‌شود.

مشابه مرحله آموزش، برای هر تابع عضویت، تأثیر ورودی روی خروجی محاسبه می‌شود و در ماتریس  $Z$  ذخیره می‌شود. سپس خروجی مدل ( $f_hat(q)$ ) برای ورودی جدید پیش‌بینی می‌شود.

خروجی واقعی سیستم ( $y(q+1)$ ) و پیش‌بینی شده آن ( $y_hat(q+1)$ ) برای ورودی جدید محاسبه می‌شود.

```
% Test phase  
for q = num_training:700  
    b = 0; a = 0;  
    x(q) = sin(2 * q * pi / 200);  
    u(q) = x(q);  
    g_u(q) = 0.6 * sin(pi * u(q)) + 0.3 * sin(3 * pi * u(q)) + 0.1 * sin(5 * pi * u(q));
```

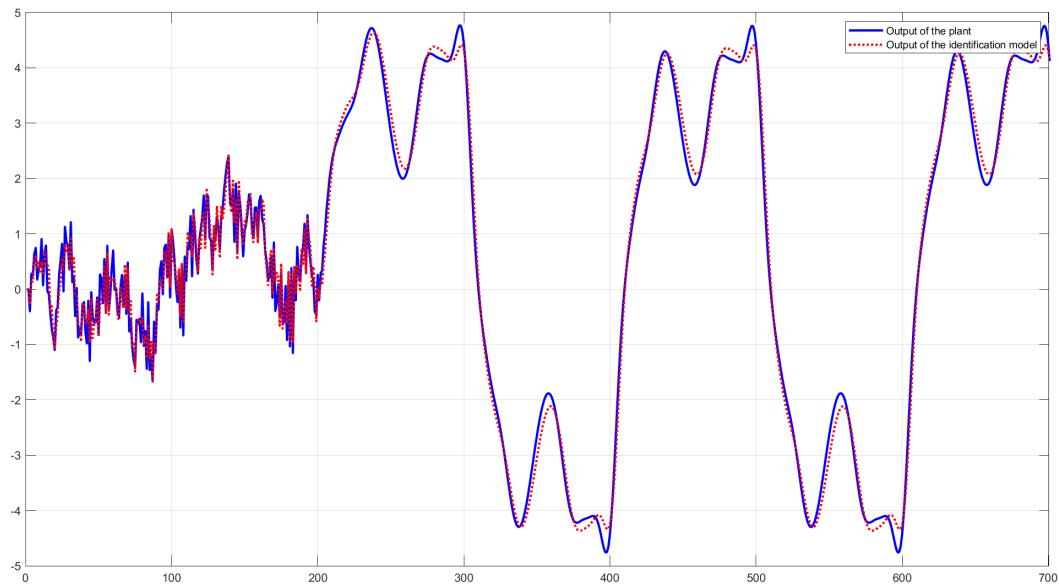
```
Z = zeros(1, M);  
for r = 1:M  
    z(r) = exp(-((x(q) - x_bar(num_training, r)) / sigma(num_training, r))^2);  
    Z(r) = z(r);  
    b = b + z(r);  
    a = a + g_bar(num_training, r) * z(r);  
end
```

$f_hat(q) = a / b;$  % Output approximation

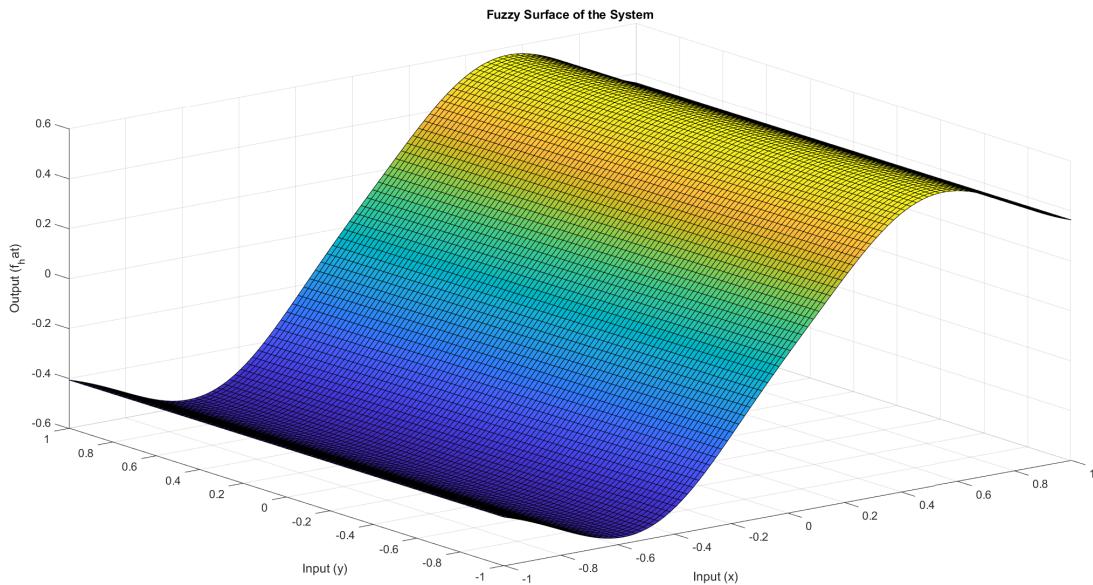


```
y(q + 1) = 0.3 * y(q) + 0.6 * y(q - 1) + g_u(q);  
y_hat(q + 1) = 0.3 * y(q) + 0.6 * y(q - 1) + f_hat(q);  
end
```

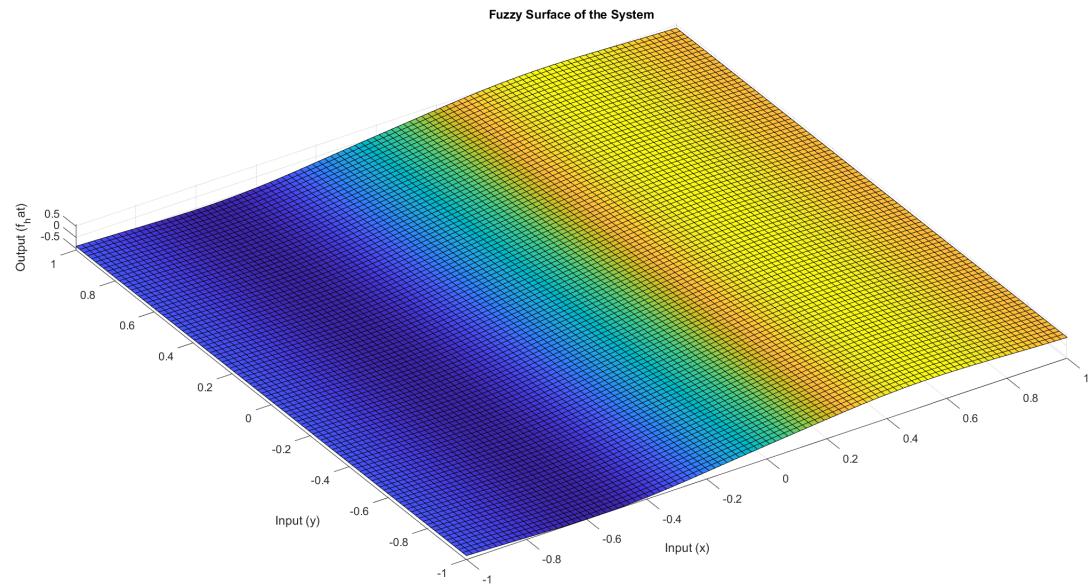
نتایج بدست آمده:



تصویر 39 - خروجی واقعی و پیش‌بینی شده



تصویر 40 - فازی سرفیس



تصویر 41 - فازی سرفیس

RMSE for test data: 0.1979

مقدار RMSE به ما نشان میدهد که مدل ما به خوبی آموزش دیده است و میتواند تابع غیرخطی تعریف شده را به خوبی پیش‌بینی کند.



## پرسش پنجم

دیتاست Repository Learning Machine UCI - Air Quality Dataset را از این [بیوند](#) دریافت کنید.

دیتاست را به سه دسته train ، validation و test با نسبت به ترتیب 60% ، 20% و 20% در نظر بگیرید و به سوالات زیر پاسخ دهید:

شبکه های عصبی با پایه تابعی شعاعی (RBF) و سیستم استنتاج فازی تطبیقی (ANFIS) و مدل های Takagi-Sugeno-Kang (TSK) هر دو برای مدلسازی سیستم های پیچیده و غیرخطی به کار می روند. هر کدام از این مدل ها دارای خصوصیات منحصر به فردی هستند که آن ها را برای برخی از کاربرد ها مناسب تر می کند.

مدل های RBF و مدل ANFIS را بررئی این دیتاست اعمال کنید و توضیح دهید کدام یک عملکرد بهتری از خود نشان داده اند؟

دلیل این برتری عملکرد چیست؟ آن را کامل توضیح دهید.

این مجموعه داده شامل 9358 نمونه از پاسخهای میانگین ساعتی از مجموعه ای از ۵ حسگر شیمیایی اکسید فلزی است که در یک دستگاه چندحسگری کیفیت هوا قرار دارند. دستگاه در یک منطقه آلوده در سطح جاده در یک شهر ایتالیایی نصب شده بوده و داده ها از مارس 2004 تا فوریه 2005 ثبت شده اند. این مجموعه شامل 15 ویژگی است.

ویژگی ها:

1. Date: تاریخ اندازه گیری

2. Time: ساعت اندازه گیری

3. CO: غلظت مونوکسید کربن ( $\text{mg}/\text{m}^3$ )

4. PT08.S1 (CO): پاسخ حسگر CO

5. NMHC: غلظت هیدروکربن های غیرمتان ( $\mu\text{g}/\text{m}^3$ )

6. C6H6 (GT): غلظت بنزن ( $\mu\text{g}/\text{m}^3$ )

7. PT08.S2 (NMHC): پاسخ حسگر NMHC

8. NOx (GT): غلظت اکسیدهای نیتروژن (ppb)

9. PT08.S3 (NOx): پاسخ حسگر NOx

10. NO2 (GT): غلظت دی اکسید نیتروژن (ppb). خروجی مدنظر ما

11. PT08.S4 (NO2): پاسخ حسگر NO2

12. PT08.S5 (O3): پاسخ حسگر اوزون

13. Temperature: دمای هوا (C°)



RH.14: رطوبت نسبی (%)  
AH.15: رطوبت مطلق (g/m³)

داده‌های گم شده با مقدار 200- مشخص شده‌اند.

## شناسایی با ANFIS

پس از بررسی تعداد دیتای میس شده در هر ستون مشخص شد که در ستون پاسخ حسگر حدود 8000 دیتای میس شده وجود دارد پس بصورت کلی این ستون را از دیتا حذف میکنیم. همچنین دو ستون اول یعنی زمان و تاریخ نیز برای سادگی شبیه سازی و کمتر شدن ستون ها حذف شدند چون از جنس float بودند. بعد از این دو مرحله همه سطر ها را بررسی میکنیم و هر سطری که حتی یک داده میس شده داشت از دیتاست حذف میکنیم تا مدل دارای نویز در آموزش نشود. در نهایت داده ها نرمالایز میکنیم تا دقت مدل بیشتر شود. کد این بخش:

```
data = readtable('AirQualityUCI.xlsx');
```

```
data(:, [1, 2 ;[]) = ([]
```

```
inputData = data(:, 1:end-1);
```

```
outputData = data(:, end);
```

```
invalidData = any(inputData{:, :} == -200, 2);
```

```
cleanedData = data(~invalidData, :);
```

```
cleanedInputData = cleanedData(:, 1:end-1);
```

```
cleanedOutputData = cleanedData(:, end);
```

```
disp('Number of remaining data after removing -200:');
```

```
disp(sum(~invalidData));
```

```
disp('Number of valid data in input columns:');
```

```
disp(sum(~invalidData));
```



```
disp('Number of valid data in output column:');
disp(sum(~invalidData));

cleanedInputData = table2array(cleanedInputData);
cleanedOutputData = table2array(cleanedOutputData);

cleanedInputData = (cleanedInputData - min(cleanedInputData)) ./ (max(cleanedInputData) - min(cleanedInputData));
cleanedOutputData = (cleanedOutputData - min(cleanedOutputData)) ./ (max(cleanedOutputData) - min(cleanedOutputData));

totalData = height(cleanedData);
```

در بخش بعدی داده ها را به سه قسمت آموزش، تست و ولیدیشن تقسیم میکنیم:

```
trainSize = round(0.6 * totalData);
testSize = round(0.2 * totalData);
validationSize = totalData - trainSize - testSize;

trainData = cleanedData(1:trainSize, :);
testData = cleanedData(trainSize+1:trainSize+testSize, :);
validationData = cleanedData(trainSize+testSize+1:end, :);
```

```
disp('Total number of training data:');
disp(trainSize);

disp('Total number of testing data:');
disp(testSize);

disp('Total number of validation data:');
disp(validationSize);
```

نتایج این بخش از کد ها:

Number of valid data in input columns: 6944



Number of valid data in output column: 6944

Total number of training data: 4166

Total number of testing data: 1389

Total number of validation data: 1389

مشاهده میشود که از مجموع 9358 سطر بدون دیتای میس شده باقی مانده است.

در ادامه برای آموزش از متدهای زیر استفاده میکنیم:

```
training_data = [cleanedInputData, cleanedOutputData];
```

```
spread = 0.5;
```

```
fis = genfis2(cleanedInputData, cleanedOutputData, spread);
```

```
MaxEpoch = 150;
```

```
ErrorGoal = 0;
```

```
InitialStepSize = 0.01;
```

```
StepSizeDecreaseRate = 0.9;
```

```
StepSizeIncreaseRate = 1.1;
```

```
TrainOptions = [MaxEpoch, ErrorGoal, InitialStepSize, StepSizeDecreaseRate, StepSizeIncreaseRate];
```

```
DisplayInfo = true;
```

```
DisplayError = true;
```

```
DisplayStepSize = true;
```

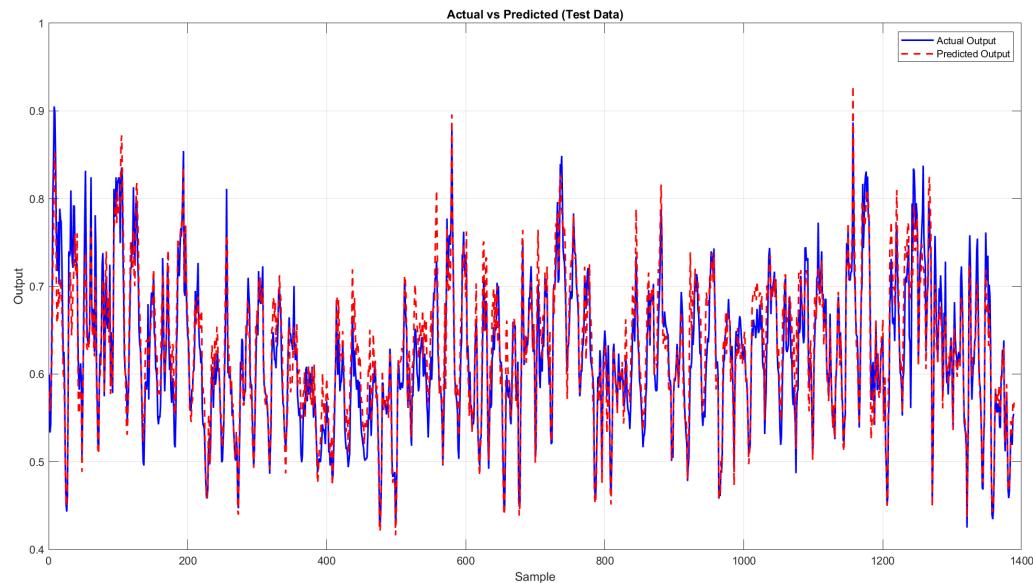
```
DisplayFinalResult = true;
```

```
DisplayOptions = [DisplayInfo, DisplayError, DisplayStepSize, DisplayFinalResult];
```

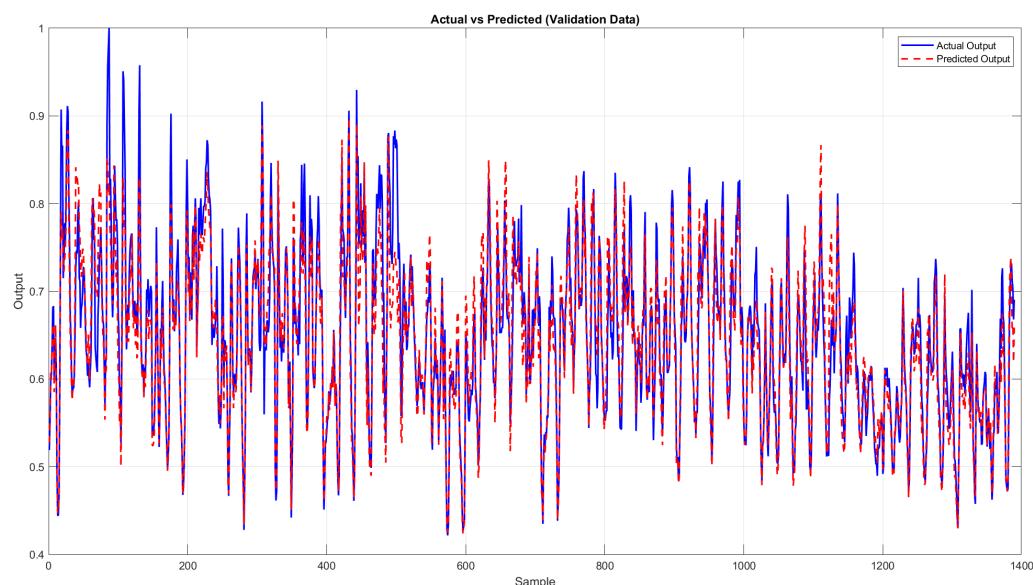
```
OptimizationMethod = 1;
```

```
[fis, trainError] = anfis(training_data, fis, TrainOptions, DisplayOptions, [], OptimizationMethod);
```

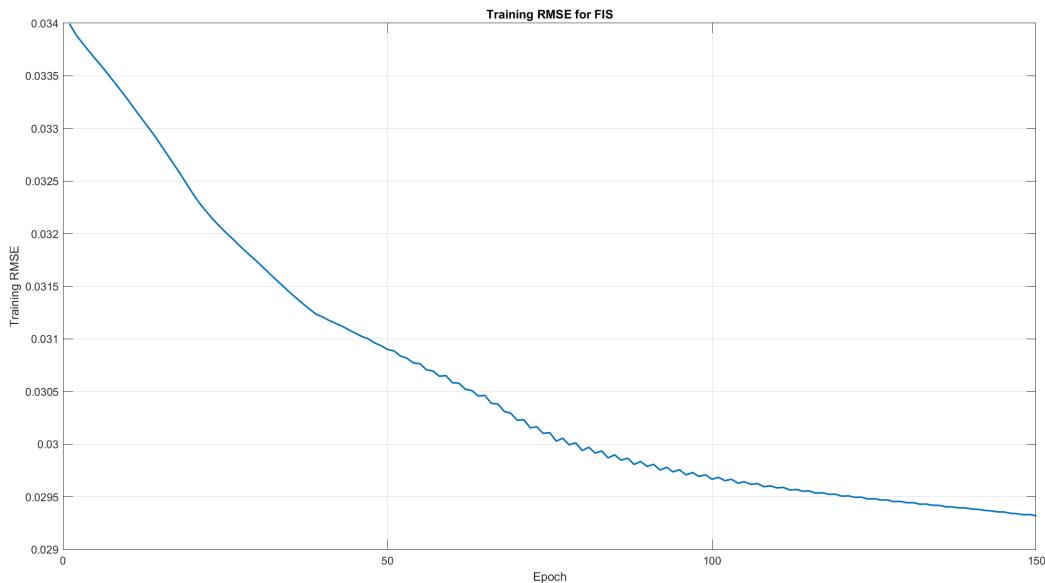
این متدها دقیقاً شبیه به متدهای استفاده شده در سوال 3 میباشد پس توضیحات بیشتری داده نمیشود. در نهایت با معیار RMSE در داده‌های تست و ولیدیشن مدل آموزش دیده را بررسی میکنیم.



تصویر 42 - مقدار واقعی و پیش‌بینی شده در تست



تصویر 43 - مقدار واقعی و پیش‌بینی شده در ولیدیشن



تصویر 44 - RMSE به ازای تعداد ایپاک

Test RMSE: 0.0301

Validation RMSE: 0.0346

مشاهده میشود مقدار RMSE هم در داده های تست و هم در داده های ولیدیشن بسیار پایین است و این یعنی مدل دقیق بسیار خوبی دارد. همچنین با اینکه ابعاد مسئله خیلی بزرگ نبود زمان ران شدن خیلی طولانی نبود که این فاکتور مهم و خوبی است.

## شناسایی با RBF

در این قسمت نیز دقیقا برای پردازش دیتای اولیه مانند قسمت قبل عمل میکنیم و تفاوتی نمیکند. در ادامه مراحل زیر را طی میکنیم:

spread = 0.5;

goal = 0.001;

maxNeurons = 50;

displayFreq = 1;

- مقدار spread = 0.5 تعیین کننده گستردگی تابع پایه شعاعی است. مقدار کوچکتر باعث محلی تر شدن توابع می شود.
- مقدار goal = 0.001 مشخص می کند که آموزش تا زمانی ادامه یابد که خطای این مقدار یا کمتر برسد.
- مقدار maxNeurons = 50 حداقل تعداد نرون های ممکن را تعیین می کند.



مشخص می‌کند که اطلاعات آموزش در هر مرحله نمایش داده شود.

```
net = newrb(trainInput', trainOutput', goal, spread, maxNeurons, displayFreq);
```

شبکه RBF با استفاده از تابع newrb و داده‌های آموزشی (trainInput, trainOutput) و داده‌های آموزشی (goal, spread, maxNeurons, displayFreq) ایجاد و آموزش داده می‌شود.

```
testPredicted = sim(net, testInput');
```

```
validationPredicted = sim(net, validationInput');
```

```
testRMSE = sqrt(mean((testOutput - testPredicted').^2));
```

```
validationRMSE = sqrt(mean((validationOutput - validationPredicted').^2));
```

با استفاده از sim خروجی پیش‌بینی‌شده برای داده‌های تست و اعتبارسنجی محاسبه می‌شود.

خطای RMSE (ریشه میانگین مربعات خطای داده‌های تست و اعتبارسنجی محاسبه و نمایش داده می‌شود).

```
neurons = 1:maxNeurons;
```

```
rmseValues = zeros(size(neurons));
```

```
for i = 1:maxNeurons
```

```
    tempNet = newrb(trainInput', trainOutput', goal, spread, i, displayFreq);
```

```
    tempPredicted = sim(tempNet, testInput');
```

```
    rmseValues(i) = sqrt(mean((testOutput - tempPredicted').^2));
```

```
end
```

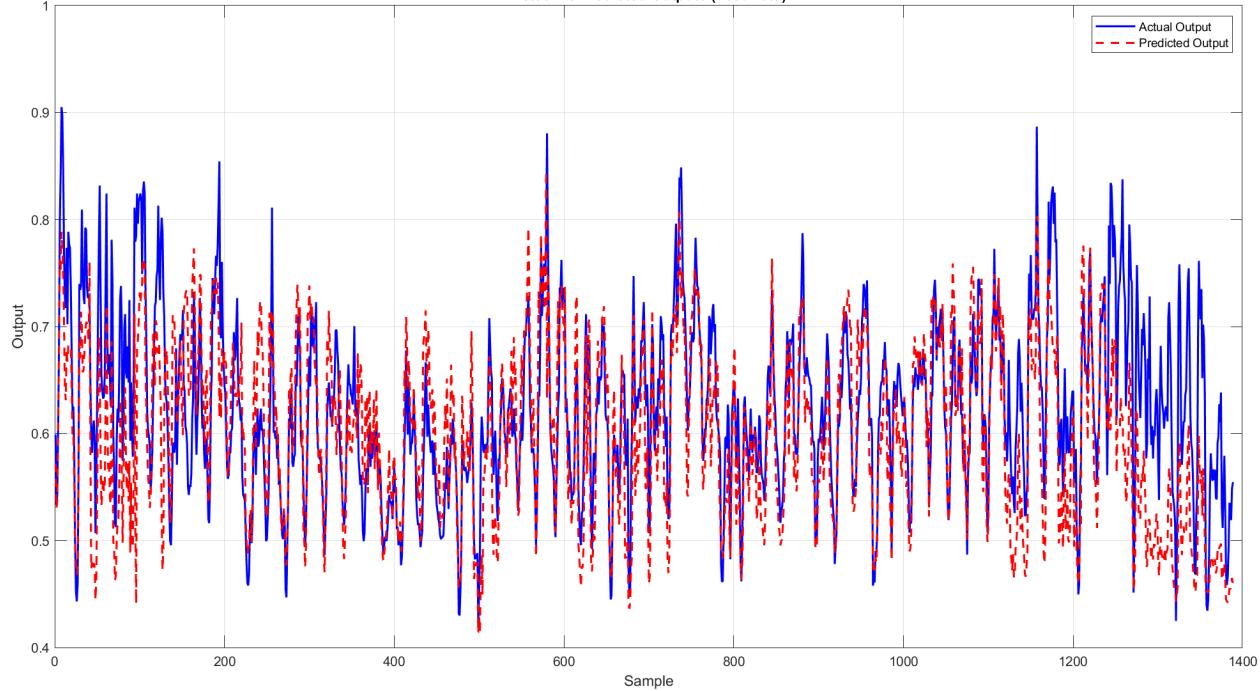
یک حلقه for شبکه‌های RBF با تعداد نرون‌های مختلف (۱ تا ۵۰) را آموزش می‌دهد.

برای هر تعداد نرون، خطای RMSE روی داده‌های تست محاسبه می‌شود. در انتها، نموداری رسم می‌شود که مقدار RMSE را بر حسب تعداد نرون‌های شبکه نمایش می‌دهد.

در نهایت نتایج زیر بدست می‌آیند:

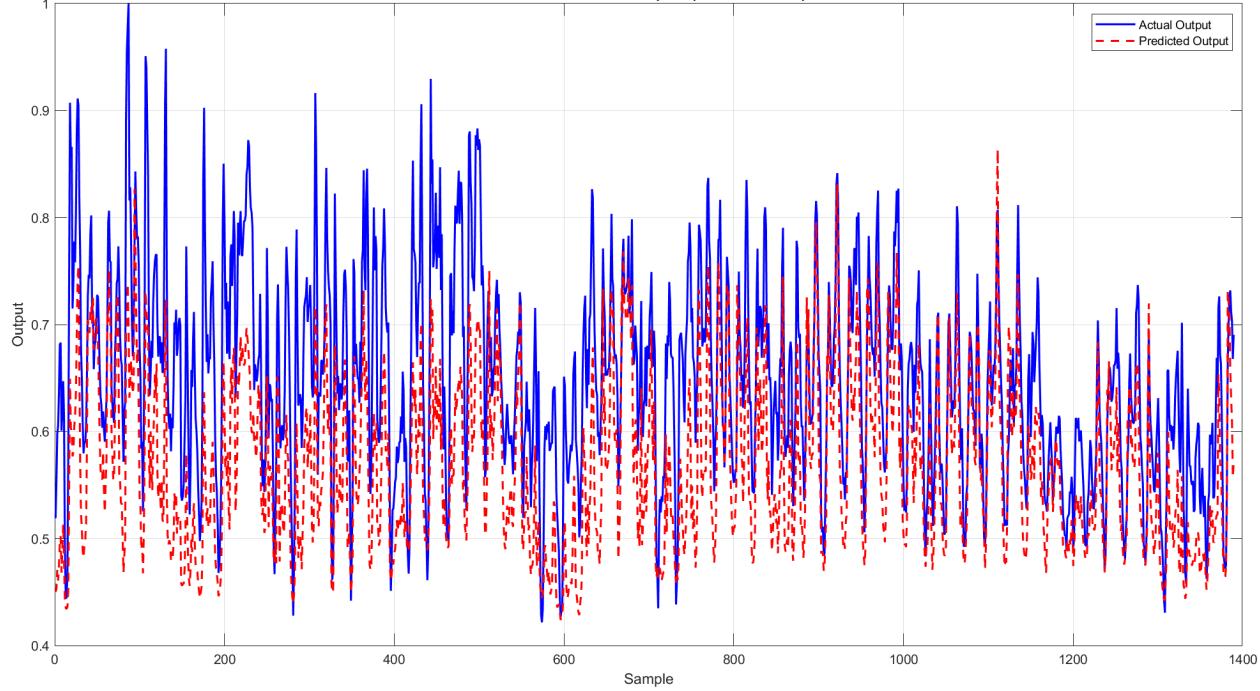


Actual vs Predicted Outputs (Test Data)

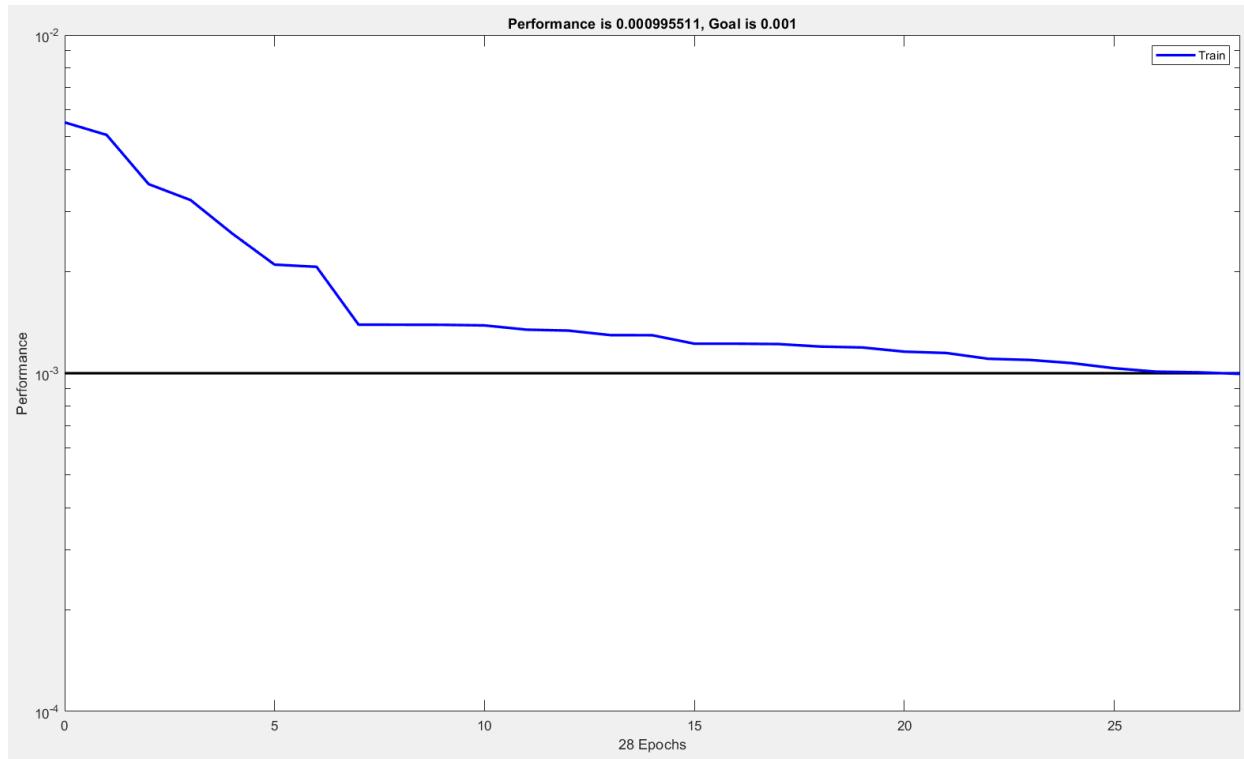


تصویر 45 - مقدار واقعی و پیش‌بینی شده در تست

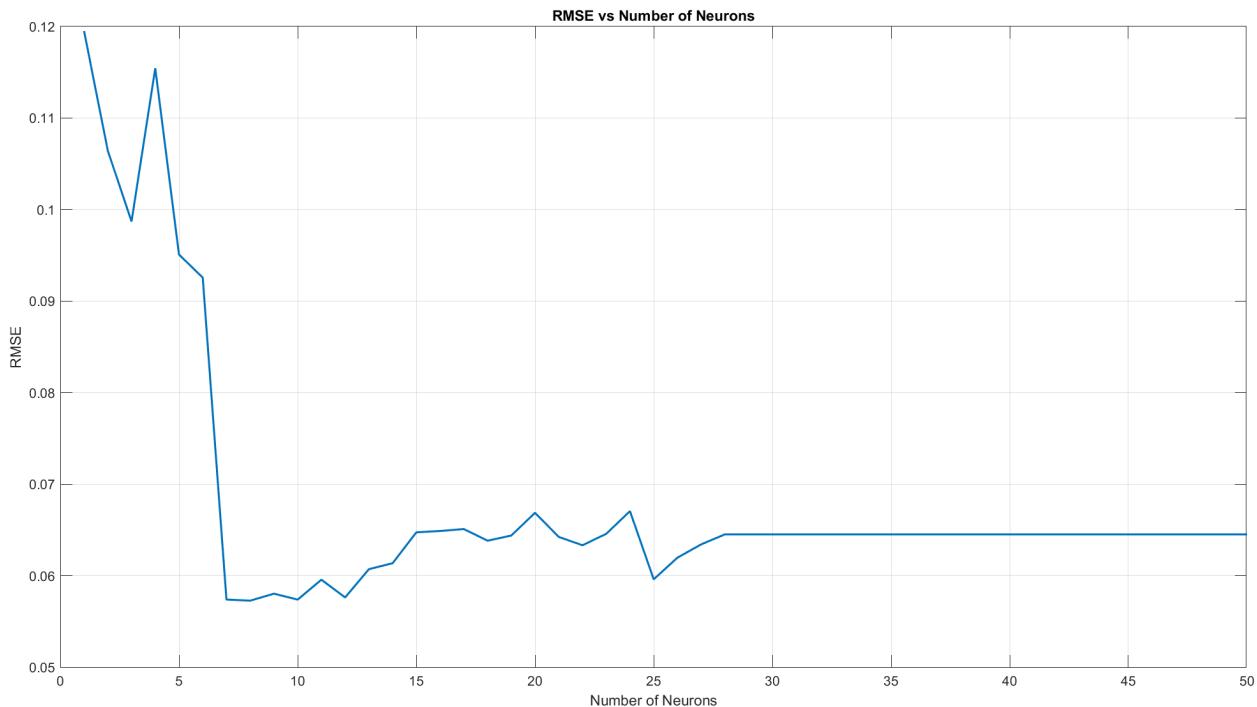
Actual vs Predicted Outputs (Validation Data)



تصویر 46 - مقدار واقعی و پیش‌بینی شده در ولیدیشن



تصویر 47 - کاهش خطابه ازای افزایش ایپاک



تصویر 48 - مقدار RMSE به ازای افزایش نورون

همانطور که مشاهده میشود با این مقادیر همان 30 نورون کافی بود و در 25 نورون بهترین RMSE مشاهده میشود اما در هر صورت مقدار RMSE مدل



---

RBF بیشتر از مدل ANFIS شده است و نکته قابل توجه این است که مدل ANFIS سریعتر آموزش دید و هم زمان دقیق‌تری دارد. این مسائل به ما نشان میدهد که استفاده از ANFIS بهتر و کارآمد‌تر است.