



دانشگاه صنعتی خواجه نصیرالدین طوسی  
دانشکده مهندسی برق - گروه مهندسی کنترل

عنوان

نمونه سوال

ANFIS

نگارش

محمدجواد احمدی

استاد درس

دکتر مهدی علیاری شوره‌دلی

# فهرست مطالب

ب	فهرست شکل‌ها
۱	پاسخ سوالات
۱	مثال اول
۶	مثال دوم
۱۱	مثال سوم
۱۱	مثال چهارم
۱۹	دستورات پایتون - مثال‌های اول تا سوم

# فهرست شکل‌ها

۵	نتایج مثال اول (آموزش)	۱
۵	نتایج مثال اول (ارزیابی)	۲
۵	نتایج GUI	۳
۱۰	نتایج مثال دوم (آموزش)	۴
۱۰	نتایج مثال دوم (ارزیابی)	۵
۱۰	نتایج GUI	۶
۱۱	نتایج GUI	۷
۱۲	سری زمانی Mackey-Glass (MG)	۸
۱۴	توابع عضویت	۹
۱۶	توابع عضویت سیستم آموزش دیده	۱۰
۱۷	نمودارهای خطا	۱۱
۱۷	نمودار اصلی و پیش‌بینی	۱۲
۱۸	خطای پیش‌بینی	۱۳
۱۹	نتایج GUI	۱۴

# پاسخ سوالات

## مثال اول

برای این مثال از دستورات زیر را در محیط متلب استفاده می کنیم:

```
1 clc;
2 clear;
3 close all;
4
5 %% Load Data
6
7 f=@(x,y) sin(x)./x .* sin(y)./y;
8
9 xmin=-10;
10 xmax=10;
11 ymin=-10;
12 ymax=10;
13 x=linspace(xmin,xmax,26)';
14 y=linspace(ymin,ymax,26)';
15 F=f(x,y);
16
17 TrainInputs=[x,y];
18 TrainTargets=F;
19 TrainData=[TrainInputs TrainTargets];
20
21 xx=linspace(xmin,xmax,106)';
22 yy=linspace(ymin,ymax,106)';
```

```
23 FF=f(xx,yy);
24
25 TestInputs=[xx,yy];
26 TestTargets=FF;
27 TestData=[TestInputs TestTargets];
28
29
30 %% Design ANFIS
31
32 % nMFs=5;
33 % InputMF='gaussmf';
34 % OutputMF='linear';
35 %
36 % fis=genfis1(TrainData,nMFs,InputMF,OutputMF);
37
38 fis=genfis2(TrainInputs,TrainTargets,0.2);
39
40 MaxEpoch=100;
41 ErrorGoal=0;
42 InitialStepSize=0.01;
43 StepSizeDecreaseRate=0.9;
44 StepSizeIncreaseRate=1.1;
45 TrainOptions=[MaxEpoch ...
46               ErrorGoal ...
47               InitialStepSize ...
48               StepSizeDecreaseRate ...
49               StepSizeIncreaseRate];
50
51 DisplayInfo=true;
52 DisplayError=true;
53 DisplayStepSize=true;
54 DisplayFinalResult=true;
55 DisplayOptions=[DisplayInfo ...
56                DisplayError ...
57                DisplayStepSize ...
```

```
58         DisplayFinalResult];
59
60 OptimizationMethod=1;
61 % 0: Backpropagation
62 % 1: Hybrid
63
64 fis=anfis(TrainData,fis,TrainOptions,DisplayOptions,[],OptimizationMethod);
65
66
67 %% Apply ANFIS to Train Data
68
69 TrainOutputs=evalfis(TrainInputs,fis);
70
71 TrainErrors=TrainTargets-TrainOutputs;
72 TrainMSE=mean(TrainErrors(:).^2);
73 TrainRMSE=sqrt(TrainMSE);
74 TrainErrorMean=mean(TrainErrors);
75 TrainErrorSTD=std(TrainErrors);
76
77 figure;
78 PlotResults(TrainTargets,TrainOutputs,'Train Data');
79
80 %% Apply ANFIS to Test Data
81
82 TestOutputs=evalfis(TestInputs,fis);
83
84 TestErrors=TestTargets-TestOutputs;
85 TestMSE=mean(TestErrors(:).^2);
86 TestRMSE=sqrt(TestMSE);
87 TestErrorMean=mean(TestErrors);
88 TestErrorSTD=std(TestErrors);
89
90 figure;
91 PlotResults(TestTargets,TestOutputs,'Test Data');
```

هم‌چنین برای بخش نمایش نمودارها هم از تابع زیر استفاده می‌کنیم:

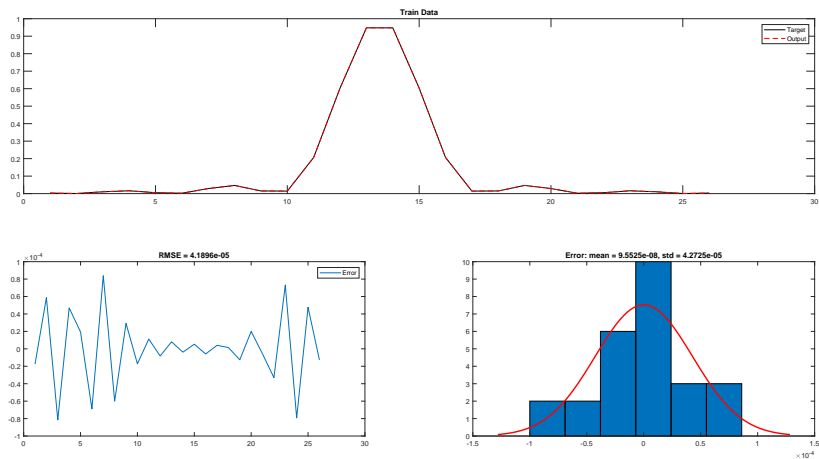
```

1 function PlotResults(targets, outputs, Name)
2
3     errors=targets-outputs;
4
5     RMSE=sqrt(mean(errors(:).^2));
6
7     error_mean=mean(errors(:));
8     error_std=std(errors(:));
9
10    subplot(2,2,[1 2]);
11    plot(targets, 'k');
12    hold on;
13    plot(outputs, 'r--');
14    legend('Target', 'Output');
15    title(Name);
16
17    subplot(2,2,3);
18    plot(errors);
19    legend('Error');
20    title(['RMSE = ' num2str(RMSE)]);
21
22    subplot(2,2,4);
23    histfit(errors);
24    title(['Error: mean = ' num2str(error_mean) ', std = ' num2str(error_std)]);
25
26 end

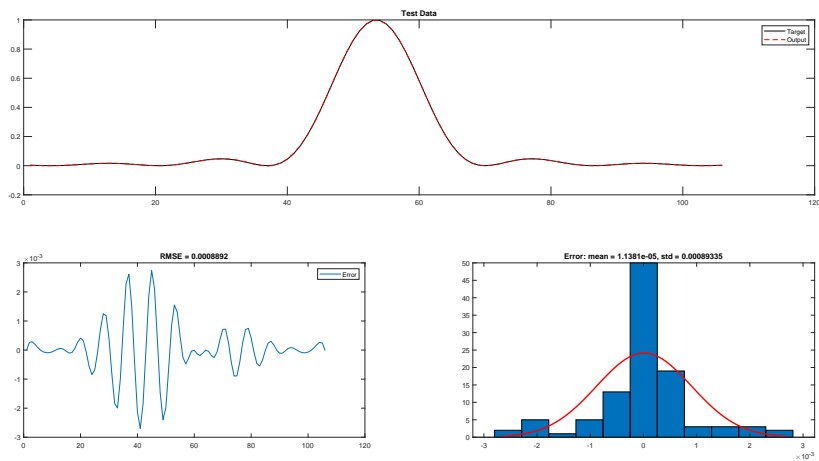
```

نتایج به صورتی است که در شکل ۱ و شکل ۲ نشان داده شده است و همان‌طور که مشاهده می‌شود، میزان خطا به میزان بسیار کمی رسیده است.

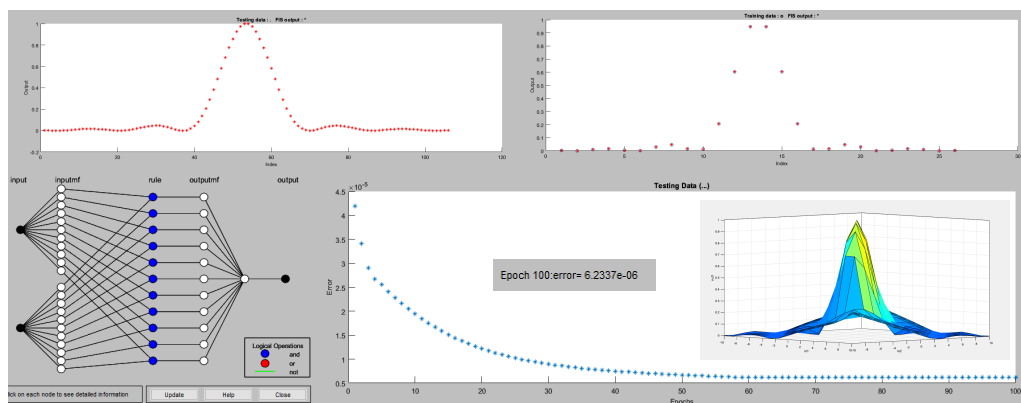
در ادامه همین فرآیند را از طریق GUI و دستور `anfisedit(fis)` پیگیری می‌کنیم. در گام اول متغیرها و داده‌ها را پس از اجرای `m`-فایل متلب فراخوانی می‌کنیم. برای نمایش بهتر و توضیحات مینیمال این فرآیند را در [این ویدیو](#) آورده شده است. نتایج هم به صورتی است که در شکل ۳ نشان داده شده است.



شکل ۱: نتایج مثال اول (آموزش).



شکل ۲: نتایج مثال اول (ارزیابی).



شکل ۳: نتایج GUI.



## نکات قابل ذکر

یک پیاده‌سازی نسبتاً کامل از این مثال در محیط **کولب** آورده شده است (از گیت‌هاب).

## مثال دوم

برای این مثال از دستورات زیر را در محیط متلب استفاده می‌کنیم:

```
1 clc;
2 clear;
3 close all;
4
5 %% Load Data
6
7 f=@(x,y,z) (1+x.^0.5+y.^-1+z.^(-1.5)).^2;
8
9 xmin=1;
10 xmax=6;
11 ymin=1;
12 ymax=6;
13 zmin=1;
14 zmax=6;
15 x=linspace(xmin,xmax,26)';
16 y=linspace(ymin,ymax,26)';
17 z=linspace(zmin,zmax,26)';
18 F=f(x,y,z);
19
20 TrainInputs=[x,y,z];
21 TrainTargets=F;
22 TrainData=[TrainInputs TrainTargets];
23
24 xx=linspace(xmin,xmax,101)';
25 yy=linspace(ymin,ymax,101)';
26 zz=linspace(zmin,zmax,101)';
```

```
27 FF=f(xx,yy,zz);
28
29 TestInputs=[xx,yy,zz];
30 TestTargets=FF;
31 TestData=[TestInputs TestTargets];
32
33
34 %% Design ANFIS
35
36 % nMFs=5;
37 % InputMF='gaussmf';
38 % OutputMF='linear';
39 %
40 % fis=genfis1(TrainData,nMFs,InputMF,OutputMF);
41
42 fis=genfis2(TrainInputs,TrainTargets,0.2);
43
44 MaxEpoch=100;
45 ErrorGoal=0;
46 InitialStepSize=0.01;
47 StepSizeDecreaseRate=0.9;
48 StepSizeIncreaseRate=1.1;
49 TrainOptions=[MaxEpoch ...
50               ErrorGoal ...
51               InitialStepSize ...
52               StepSizeDecreaseRate ...
53               StepSizeIncreaseRate];
54
55 DisplayInfo=true;
56 DisplayError=true;
57 DisplayStepSize=true;
58 DisplayFinalResult=true;
59 DisplayOptions=[DisplayInfo ...
60                DisplayError ...
61                DisplayStepSize ...
```

```

62         DisplayFinalResult];
63
64 OptimizationMethod=1;
65 % 0: Backpropagation
66 % 1: Hybrid
67
68 fis=anfis(TrainData,fis,TrainOptions,DisplayOptions,[],OptimizationMethod);
69
70
71 %% Apply ANFIS to Train Data
72
73 TrainOutputs=evalfis(TrainInputs,fis);
74
75 TrainErrors=TrainTargets-TrainOutputs;
76 TrainMSE=mean(TrainErrors(:).^2);
77 TrainRMSE=sqrt(TrainMSE);
78 TrainErrorMean=mean(TrainErrors);
79 TrainErrorSTD=std(TrainErrors);
80 TrainAPE= abs(TrainErrors(:))./abs(TrainTargets(:))*(26/100)
81
82 figure;
83 PlotResults(TrainTargets,TrainOutputs,'Train Data');
84
85 %% Apply ANFIS to Test Data
86
87 TestOutputs=evalfis(TestInputs,fis);
88
89 TestErrors=TestTargets-TestOutputs;
90 TestMSE=mean(TestErrors(:).^2);
91 TestRMSE=sqrt(TestMSE);
92 TestErrorMean=mean(TestErrors);
93 TestErrorSTD=std(TestErrors);
94 TestAPE= abs(TestErrors(:))./abs(TestTargets(:))*(101/100)
95
96 figure;

```

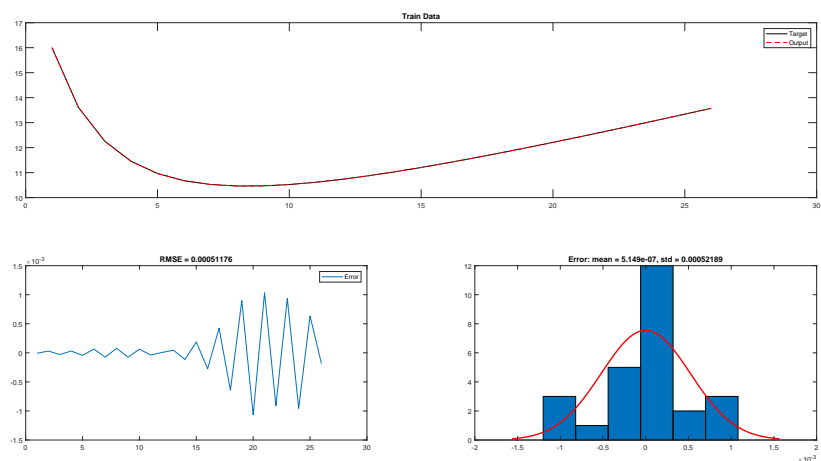
```
97 PlotResults(TestTargets,TestOutputs,'Test Data');
```

هم چنین برای بخش نمایش نمودارها هم از تابع زیر استفاده می کنیم:

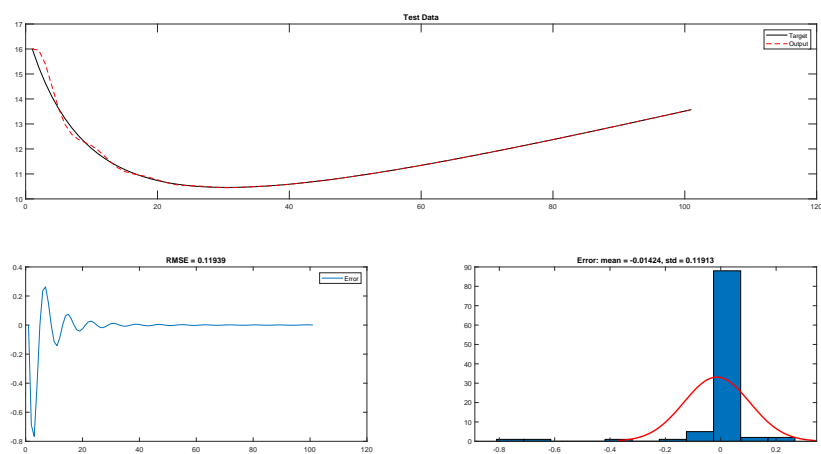
```
1 function PlotResults(targets,outputs,Name)
2
3     errors=targets-outputs;
4
5     RMSE=sqrt(mean(errors(:).^2));
6
7     error_mean=mean(errors(:));
8     error_std=std(errors(:));
9
10    subplot(2,2,[1 2]);
11    plot(targets,'k');
12    hold on;
13    plot(outputs,'r--');
14    legend('Target','Output');
15    title(Name);
16
17    subplot(2,2,3);
18    plot(errors);
19    legend('Error');
20    title(['RMSE = ' num2str(RMSE)]);
21
22    subplot(2,2,4);
23    histfit(errors);
24    title(['Error: mean = ' num2str(error_mean) ', std = ' num2str(error_std)]');
25
26 end
```

نتایج به صورتی است که در شکل ۴ و شکل ۵ نشان داده شده است و همان طور که مشاهده می شود، میزان خطا به میزان بسیار کمی رسیده است.

در ادامه همین فرآیند را از طریق GUI و دستور `anfisedit(fis)` پیگیری می کنیم. در گام اول متغیرها و داده ها را پس از اجرای `m`-فایل متلب فراخوانی می کنیم. برای نمایش بهتر و توضیحات مینیمال این فرآیند

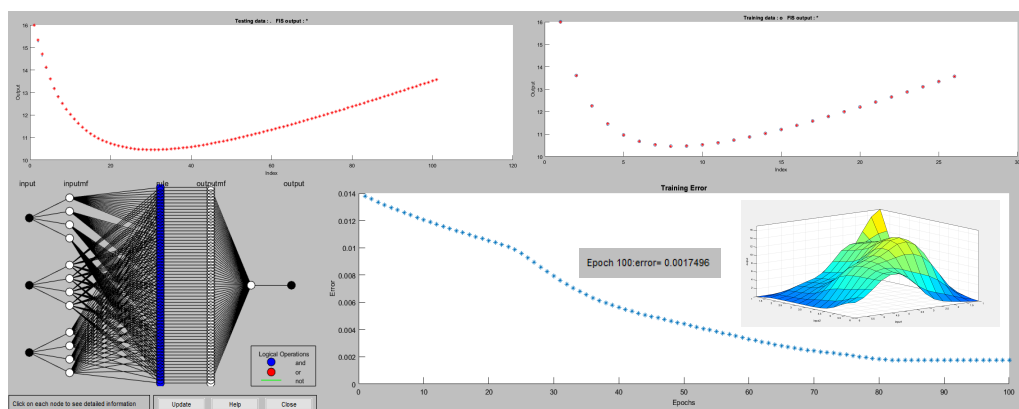


شکل ۴: نتایج مثال دوم (آموزش).



شکل ۵: نتایج مثال دوم (ارزیابی).

را در این ویدیو آورده شده است. نتایج هم به صورتی است که در شکل ۶ نشان داده شده است.



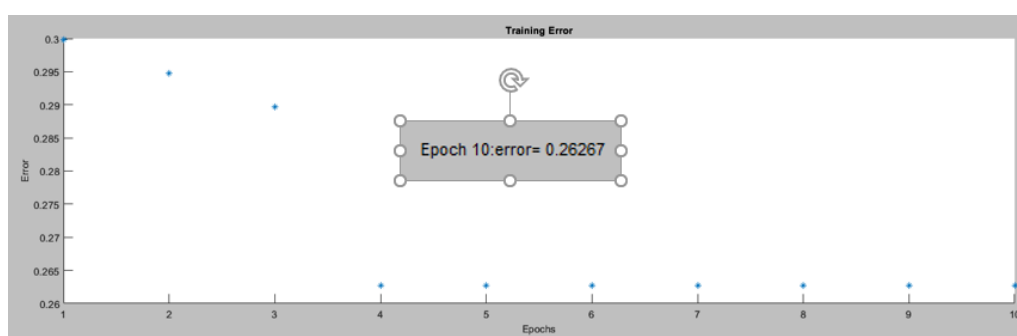
شکل ۶: نتایج GUI.

## مثال سوم

### نکات قابل ذکر

یک پیاده‌سازی و پاسخ از این سوال در دستورات پایتون - مثال‌های اول تا سوم آورده شده است.

در ادامه همین فرآیند را از طریق GUI، اجرای فایل Ex3.m و دستور anfisedit پیگیری می‌کنیم. در گام اول متغیرها و داده‌ها را پس از اجرای m-فایل متلب فراخوانی می‌کنیم. نتایج هم به صورتی است که در شکل ۷ نشان داده شده است.



شکل ۷: نتایج GUI.

## مثال چهارم

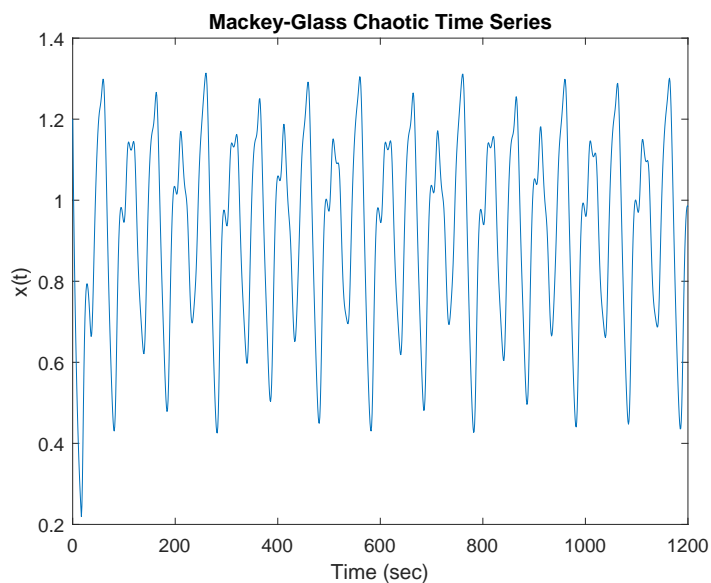
هدف این مثال از ANFIS پیش‌بینی یک سری زمانی تولیدی از معادله دیفرانسیل تأخیر زمانی مکی-گلکس (رابطه ۱) است.

$$\dot{x}(t) = \frac{\circ/\mathcal{X}x(t-\tau)}{1+x^{\circ}(t-\tau)} - \circ/\mathcal{X}x(t) \quad (1)$$

این سری زمانی آشوب‌ناک است و دوره زمانی مشخصی ندارد. این سری هم‌گرا یا واگرا نمی‌شود و خط سیر آن همانند سایر سیستم‌های آشوب‌ناک بسیار حساس به شرایط اولیه است. برای به‌دست آوردن مقدار سری زمانی در نقاط صحیح، از روش رانگه‌کوئای مرتبه چهارم برای یافتن جواب عددی رابطه ۱ استفاده می‌شود. فرض بر این است که برای زمان‌های کوچک‌تر از صفر،  $x(t) = \circ$ ،  $\tau = 17$ ،  $x(\circ) = 1/2$  باشد. نتایج در mgdata.dat ذخیره شده است. در گام اول داده‌های سری زمانی را فراخوانی و رسم می‌کنیم.

برای این کار از دستورات زیر استفاده می‌کنیم. نتیجه در؟؟ آورده شده است.

```
1 load mgdata.dat
2 time = mgdata(:,1);
3 x = mgdata(:, 2);
4 figure(1)
5 plot(time,x)
6 title('Mackey-Glass Chaotic Time Series')
7 xlabel('Time (sec)')
8 ylabel('x(t)')
```



شکل ۸: سری زمانی (MG) Mackey-Glass.

در پیش‌بینی سری‌های زمانی، از مقادیر شناخته‌شده سری‌های زمانی تا یک نقطه  $(t)$  استفاده می‌کنیم تا مقدار را در نقطه‌ای در آینده  $(t+P)$  پیش‌بینی کنیم. یک روش استاندارد برای این کار، ایجاد یک نگاشت از نمونه‌هایی که در زمان‌هایی معین نمونه‌برداری شده‌اند  $((x(t-(D-1)\Delta), \dots, x(t-\Delta), x(t)))$  برای پیش‌بینی مقدار آینده  $(x = (t+P))$  است. بر مبنای تنظیمات مرسوم برای پیش‌بینی سری زمانی،  $D = 4$  و  $\Delta = P = 6$  در نظر گرفته می‌شود. برای هر  $t$  داده آموزشی ورودی یک بردار چهارستونه به صورت زیر است:

$$w(t) = [x(t-19), x(t-12), x(t-6), x(t)] \quad (2)$$

خروجی داده آموزش با توجه به مسیر پیش‌بینی به صورت زیر است:

$$s(t) = x(t + 6) \quad (3)$$

برای هر  $t$ ، که از مقادیر ۱۱۸ تا ۱۱۱۷ متغیر است، ۱۰۰۰ نمونه آموزشی ورودی/خروجی وجود دارد. برای این مثال، از ۵۰۰ نمونه اول به عنوان داده‌های آموزشی و از ۵۰۰ مقدار دوم به عنوان داده‌های اعتبارسنجی استفاده می‌کنیم. هر ردیف از آرایه‌های داده‌های آموزش و اعتبارسنجی شامل یک نقطه نمونه است که در آن چهار ستون اول شامل ورودی چهار بعدی  $w$  و ستون پنجم شامل خروجی  $s$  است. این داده‌های را با استفاده از دستورات زیر می‌سازیم:

```
1 for t = 118:1117
2     Data(t-117,:) = [x(t-18) x(t-12) x(t-6) x(t) x(t+6)];
3 end
4 trnData = Data(1:500,:);
5 chkData = Data(501:end,:);
```

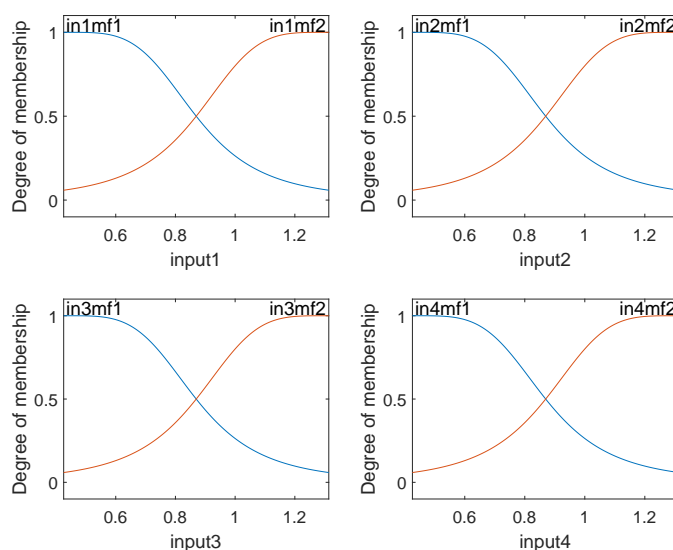
در ادامه، یک شیء اولیه فازی سوگنو (FIS) برای آموزش با استفاده از تابع `genfis` ایجاد می‌کنیم. تعداد ورودی‌ها و خروجی‌های این سیستم فازی با تعداد ستون‌های داده‌های آموزشی ورودی و خروجی (به ترتیب چهار و یک) مطابقت دارد (شکل ۹).

```
1 figure
2 subplot(2,2,1)
3 plotmf(fis,'input',1)
4 subplot(2,2,2)
5 plotmf(fis,'input',2)
6 subplot(2,2,3)
7 plotmf(fis,'input',3)
8 subplot(2,2,4)
9 plotmf(fis,'input',4)
```

به طور پیش فرض، `genfis` دو تابع عضویت گاوسی (زنگوله‌ای) تعمیم یافته را برای هر یک از چهار ورودی ایجاد می‌کند. توابع عضویت اولیه برای هر متغیر به یک اندازه فاصله دارند و کل فضای ورودی را پوشش می‌دهند.

```
1 for t = 118:1117
2     Data(t-117,:) = [x(t-18) x(t-12) x(t-6) x(t) x(t+6)];
```





شکل ۹: توابع عضویت.

```
3 end
4 trnData = Data(1:500,:);
5 chkData = Data(501:end,:);
```

شی فازی تولیدی،  $۱۶ = ۲۴$  قانون فازی با  $۱۰۴$  پارامتر شامل  $۲۴$  پارامتر غیر خطی و  $۸۰$  پارامتر خطی دارد. برای دستیابی به قابلیت تعمیم خوب، مهم است که تعداد نقاط داده آموزشی چندین برابر بیشتر از تعداد پارامترهای تخمین زده شده باشد. در این مورد، نسبت بین داده ها و پارامترها تقریباً پنج است؛ یعنی  $۵۰۰$  به  $۱۰۴$ ، که تعادل خوبی بین پارامترهای برازش و نقاط نمونه آموزشی است. در ادامه گزینه های آموزشی را پیکربندی می کنیم و FIS را با داده های و گزینه های آموزشی تعیینی آموزش می دهیم:

```
1 options = anfisOptions('InitialFIS',fis,'ValidationData',chkData);
2 [fis1,error1,ss,fis2,error2] = anfis(trnData,options);
```

داریم:

```
1 ANFIS info:
2   Number of nodes: 55
3   Number of linear parameters: 80
4   Number of nonlinear parameters: 24
5   Total number of parameters: 104
6   Number of training data pairs: 500
7   Number of checking data pairs: 500
```

```

8   Number of fuzzy rules: 16
9
10
11 Start training ANFIS ...
12
13 1      0.00296046      0.00292488
14 2      0.00290346      0.0028684
15 3      0.00285048      0.00281544
16 4      0.00280117      0.00276566
17 Step size increases to 0.011000 after epoch 5.
18 5      0.00275517      0.00271874
19 6      0.00271214      0.00267438
20 7      0.00266783      0.00262818
21 8      0.00262626      0.00258435
22 Step size increases to 0.012100 after epoch 9.
23 9      0.00258702      0.00254254
24 10     0.00254972      0.00250247
25
26 Designated epoch number reached. ANFIS training completed at epoch 10.
27
28 Minimal training RMSE = 0.00254972
29 Minimal checking RMSE = 0.00250247

```

سیستم `fis1` استنتاج فازی آموزش دیده برای دوره آموزشی است که در آن خطای آموزشی کمترین مقدار است. از آن جا که داده های اعتبارسنجی را هم مشخص کرده ایم، سیستم فازی با حداقل خطای اعتبارسنجی، `fis2`، نیز برگردانده می شود. سیستم با کمترین خطای اعتبارسنجی، بهترین تعمیم را فراتر از داده های آموزشی نشان می دهد. حال توابع عضویت را برای سیستم آموزش دیده ترسیم می کنیم (شکل ۱۰).

```

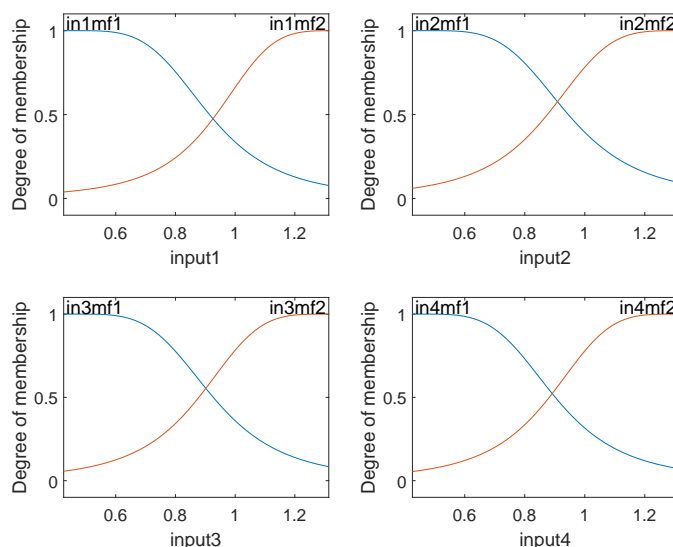
1 figure
2 subplot(2,2,1)
3 plotmf(fis2,'input',1)
4 subplot(2,2,2)
5 plotmf(fis2,'input',2)
6 subplot(2,2,3)
7 plotmf(fis2,'input',3)

```

```

8 subplot(2,2,4)
9 plotmf(fis2,'input',4)

```



شکل ۱۰: توابع عضویت سیستم آموزش دیده.

در ادامه با استفاده از دستورات زیر نمودارهای خطا را رسم می‌کنیم (شکل ۱۱):

```

1 figure
2 plot([error1 error2])
3 hold on
4 plot([error1 error2], 'o')
5 legend('Training error', 'Checking error')
6 xlabel('Epochs')
7 ylabel('Root Mean Squared Error')
8 title('Error Curves')

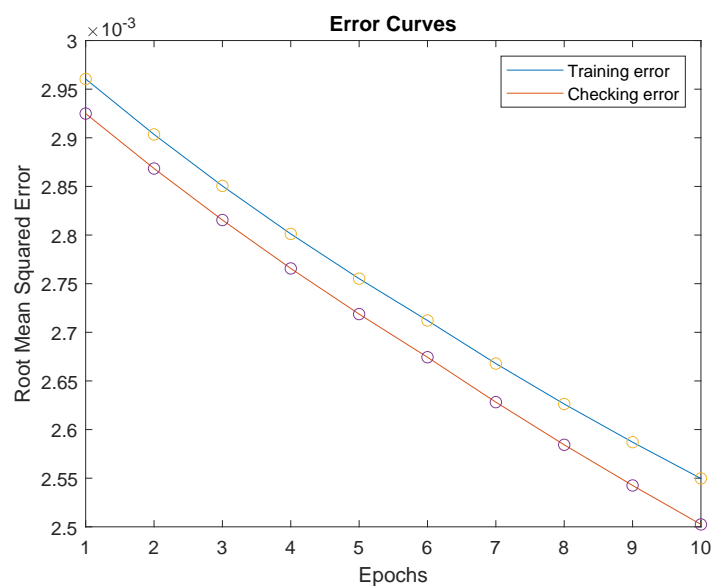
```

خطای آموزش در همه دوره‌ها بیش‌تر از خطای اعتبارسنجی است. این پدیده در یادگیری ANFIS یا رگرسیون غیرخطی به طور کلی غیرمعمول نیست و می‌تواند نشان دهد که آموزش اضافی می‌تواند نتایج آموزشی بهتری ایجاد کند. برای بررسی قابلیت پیش‌بینی سیستم آموزش دیده، سیستم فازی را با استفاده از داده‌های آموزش و اعتبارسنجی ارزیابی کرده و نتیجه را در کنار نمونه اصلی رسم می‌کنیم (شکل ۱۲).

```

1 anfis_output = evalfis(fis2,[trnData(:,1:4); chkData(:,1:4)]);
2
3 figure
4 index = 125:1124;

```

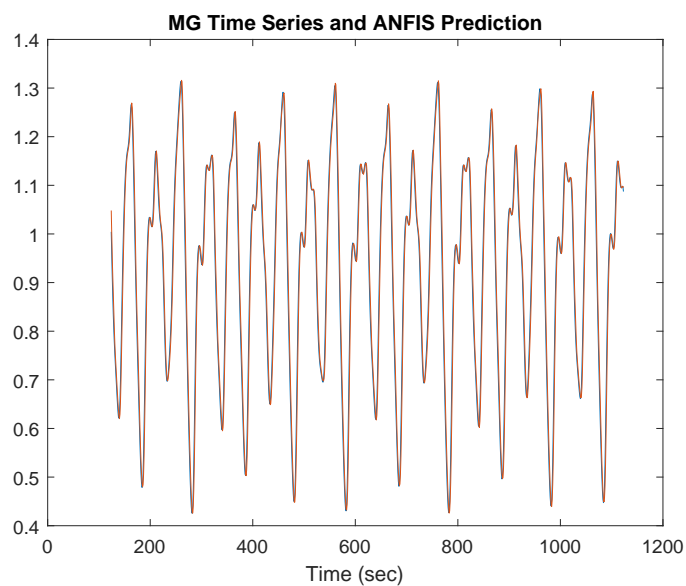


شکل ۱۱: نمودارهای خطا.

```

5 plot(time(index),[x(index) anfis_output])
6 xlabel('Time (sec)')
7 title('MG Time Series and ANFIS Prediction')

```



شکل ۱۲: نمودار اصلی و پیش‌بینی.

مشاهده می‌کنیم که سری زمانی پیش‌بینی شده مشابه سری زمانی اصلی است. برای بررسی دقیق‌تر خطای پیش‌بینی را محاسبه و رسم می‌کنیم (شکل ۱۳).

```

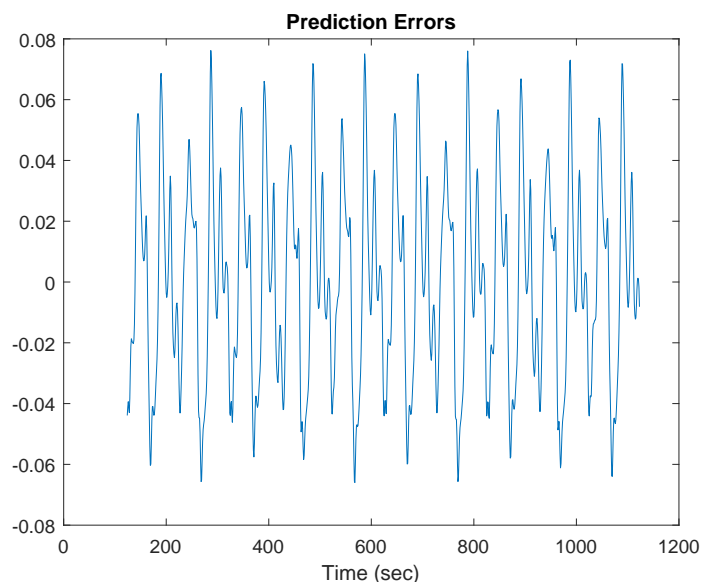
1 diff = x(index) - anfis_output;

```

```

2 plot(time(index),diff)
3 xlabel('Time (sec)')
4 title('Prediction Errors')

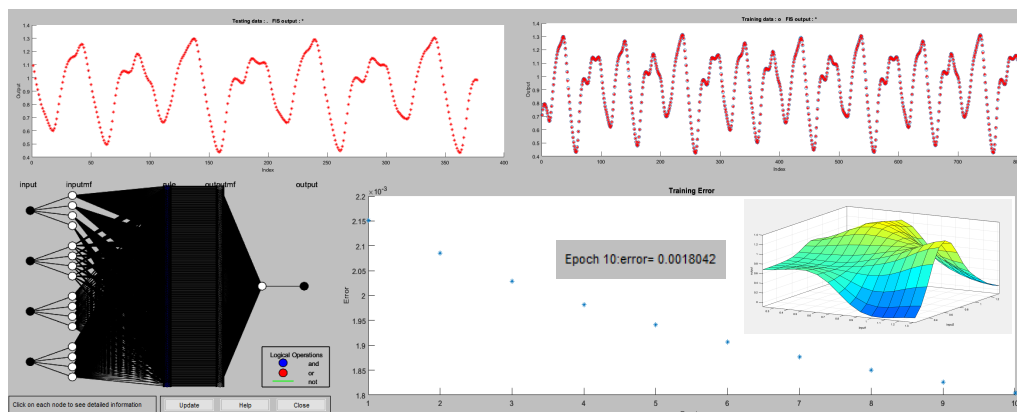
```



شکل ۱۳: خطای پیش‌بینی.

مشاهده می‌شود که مقیاس نمودار خطای پیش‌بینی حدود یک صدم مقیاس نمودار سری زمانی است. در این مثال، ما سیستم را تنها برای ۱۰ دوره آموزش داده ایم. آموزش برای دوره‌های بیشتر می‌تواند نتایج آموزشی را بهبود ببخشد.

در ادامه همین فرآیند را از طریق GUI، اجرای فایل ANFISGUI.m و دستور anfisedit پیگیری می‌کنیم. در گام اول متغیرها و داده‌ها را پس از اجرای m-فایل متلب فراخوانی می‌کنیم. برای نمایش بهتر و توضیحات مینیمال این فرآیند را در **این ویدیو** آورده شده است. نتایج هم به صورتی است که در شکل ۱۴ نشان داده شده است.



شکل ۱۴: نتایج GUI.

## دستورات پایتون - مثال‌های اول تا سوم

### نکات قابل ذکر

لازم به ذکر است که مرحله این کدها **گیت لب** بوده و با ایجاد تغییراتی در فایل‌های اصلی آن را آماده برای اجرا در محیط **کولب** کرده‌ام. فایل‌های تغییر یافته و آماده اجرا در محیط کولب را در **این پیوند** ذخیره کرده‌ام.

دستورات پایتون مربوطه در **این پیوند** آورده شده است. هم‌چنین کدهای کلی از طریق **این پیوند** قابل دسترسی است. فایل اصلی اجرا `jang_examples.py` است:

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 '''
4     ANFIS in torch: Examples from Jang's paper
5     @author: James Power <james.power@mu.ie> Apr 12 18:13:10 2019
6 '''
7
8 import sys
9 import itertools
10 import numpy as np
11
12 import torch
13 from torch.utils.data import TensorDataset, DataLoader
14
```

```

15 import anfis
16 from membership import BellMembFunc, make_bell_mfs
17 from experimental import train_anfis, test_anfis, train_anfis_cv, plot_all_mfs
18
19 dtype = torch.float64
20
21
22 # ##### Example 1: Modeling a Two-Input Nonlinear Function #####
23
24
25 def sinc(x, y):
26     '''
27         Sinc is a simple two-input non-linear function
28         used by Jang in section V of his paper (equation 30).
29     '''
30     def s(z):
31         return (1 if z == 0 else np.sin(z) / z)
32     return s(x) * s(y)
33
34
35 def make_sinc_xy(batch_size=1024):
36     '''
37         Generates a set of (x, y) values for the sinc function.
38         Use the range (-10,10) that was used in sec. V of Jang's paper.
39     '''
40     pts = torch.arange(-10, 11, 2)
41     x = torch.tensor(list(itertools.product(pts, pts)), dtype=dtype)
42     y = torch.tensor([sinc(*p) for p in x], dtype=dtype)
43     td = TensorDataset(x, y)
44     return DataLoader(td, batch_size=batch_size, shuffle=True)
45
46
47 def make_sinc_xy_test(batch_size=1024):
48     pts = torch.arange(-3, 3, 2)
49     x = torch.tensor(list(itertools.product(pts, pts)), dtype=dtype)

```

```

50     y = torch.tensor([[sinc(*p)] for p in x], dtype=dtype)
51     td = TensorDataset(x, y)
52     return DataLoader(td, batch_size=batch_size, shuffle=True)
53
54
55 def make_sinc_xy_large(num_cases=10000, batch_size=1024):
56     '''
57         Generates a set of (x, y) values for the sinc function.
58         Uses a large data set so we can test mini-batch in action.
59     '''
60     pts = torch.linspace(-10, 10, int(np.sqrt(num_cases)))
61     x = torch.tensor(list(itertools.product(pts, pts)), dtype=dtype)
62     y = torch.tensor([[sinc(*p)] for p in x], dtype=dtype)
63     td = TensorDataset(x, y)
64     return DataLoader(td, batch_size=batch_size, shuffle=True)
65
66
67 def make_sinc_xy2(batch_size=1024):
68     '''
69         A version of sinc with two outputs (sinc(x) and 1-sinc(x)).
70         Not part of Jang's work, but used by the Vignette paper.
71     '''
72     pts = list(range(-10, 11, 2))
73     x = torch.tensor(list(itertools.product(pts, pts)), dtype=dtype)
74     y = torch.tensor([[sinc(*p), 1-sinc(*p)] for p in x], dtype=dtype)
75     td = TensorDataset(x, y)
76     return DataLoader(td, batch_size=batch_size, shuffle=True)
77
78
79 def ex1_model():
80     '''
81         These are the original (untrained) MFS for Jang's example 1.
82     '''
83     invardefs = [
84         ('x0', make_bell_mfs(3.33333, 2, [-10, -3.33333, 3.33333, 10])),

```



```

85         ('x1', make_bell_mfs(3.33333, 2, [-10, -3.333333, 3.333333, 10])),
86     ]
87     outvars = ['y0']
88     anf = anfis.AnfisNet('Jang\'s example 1', invardefs, outvars)
89     return anf
90
91
92 # ##### Example 2: Modeling a Three-Input Nonlinear Function #####
93
94 def ex2_eqn(x, y, z):
95     '''
96     The three input non-linear function used in Jang's example 2
97     '''
98     output = 1 + torch.pow(x, 0.5) + torch.pow(y, -1) + torch.pow(z, -1.5)
99     output = torch.pow(output, 2)
100    return output
101
102
103 def _make_data_xyz(inp_range):
104     '''
105     Given a range, return a dataset with the product of these values.
106     Assume we want triples returned - i.e. (x,y,z) points
107     '''
108     xyz_vals = itertools.product(inp_range, inp_range, inp_range)
109     x = torch.tensor(list(xyz_vals), dtype=dtype)
110     y = torch.tensor([ex2_eqn(*p) for p in x], dtype=dtype)
111     return TensorDataset(x, y)
112
113
114 def ex2_model():
115     invardefs = [
116         ('x', make_bell_mfs(2.5, 2, [1, 6])),
117         ('y', make_bell_mfs(2.5, 2, [1, 6])),
118         ('z', make_bell_mfs(2.5, 2, [1, 6])),
119     ]

```

```

120     outvars = ['output']
121     model = anfis.AnfisNet('Jang\'s example 2', invardefs, outvars)
122     return model
123
124
125 def ex2_training_data(batch_size=1024):
126     '''
127         Jang's training data uses integer values between 1 and 6 inclusive
128     '''
129     inp_range = range(1, 7, 1)
130     td = _make_data_xyz(inp_range)
131     return DataLoader(td, batch_size=batch_size, shuffle=True)
132
133
134 def ex2_testing_data():
135     '''
136         Jang's test data uses values 1.5, 2.5 etc.
137     '''
138     inp_range = np.arange(1.5, 6.5, 1)
139     td = _make_data_xyz(inp_range)
140     return DataLoader(td)
141
142
143 # ##### Example 3: On-line Identification in Control Systems #####
144
145
146 def ex3_model(mfnum=7):
147     '''
148         Example 3 model, with variable number of Bell MFs, range (-1,+1).
149         Specify the no. of MFs, or make it 0 and I'll use Jang's 5 centers.
150         Either way, the Bell width/slope values are from Jang's data.
151     '''
152     # The paper says 7 MFs are best, but his code uses 5 MFs
153     if mfnum < 1: # use the 5 MF values from Jang's code
154         centers = [-0.999921, -0.499961, 0.000000, 0.499961, 0.99992]

```

```

155     else: # just spread them evenly accross the range (-1, +1)
156         centers = np.linspace(-1, 1, mfnum)
157         invardefs = [('k', make_bell_mfs(0.249980, 4, centers))]
158         outvars = ['y']
159         model = anfis.AnfisNet('Jang\'s example 3', invardefs, outvars)
160         return model
161
162
163 def ex3_f(u):
164     '''
165         This is the function f defined in eq 34 of Jang's paper.
166         This is the function that the ANFIS is supposed to model.
167     '''
168     pi_u = np.pi * u
169     return (0.6 * torch.sin(pi_u) +
170            0.3 * torch.sin(3 * pi_u) +
171            0.1 * torch.sin(5 * pi_u))
172
173
174 def ex3_training_data(batch_size=1024):
175     '''
176         Jang's training data, spread evenly over -1 to +1
177     '''
178     inp_range = np.arange(-1, 1.02, 0.02)
179     # Need to add an extra dimension to both x and y:
180     x = torch.tensor(inp_range, dtype=dtype).unsqueeze(1)
181     y = ex3_f(x)
182     return DataLoader(TensorDataset(x, y), batch_size=batch_size, shuffle=True)
183
184
185 def ex3_u(k):
186     '''
187         This is the input function u(k) defined in eq 33 of Jang's paper.
188         The purpose of this function is to generate (test) input values.
189         Note that this is a scalar -> scalar function (no tensors).

```

```

190     For positive integer argument, return a float in the range -1 to +1.
191     '''
192     assert k >= 1, 'not defined for k={}, only 1 or over'.format(k)
193     if k < 500:
194         u = np.sin((2 * np.pi * k) / 250)
195     else: # Over 500, use a different formula:
196         u = 0.5 * np.sin((2 * np.pi * k) / 250) + \
197             0.5 * np.sin((2 * np.pi * k) / 25)
198     return u
199
200
201 def ex3_testing_data():
202     '''
203     As test data, use the values generated by the u(k) function
204     '''
205     x = torch.tensor([ex3_u(k) for k in range(1, 700)], dtype=dtype).unsqueeze
206         (1)
207     y = ex3_f(x)
208     return DataLoader(TensorDataset(x, y))
209
210 # ##### Example 4: predicting Chaotic Dynamics #####
211
212 def ex4_model():
213     '''
214     Example 4 model, from Jang's data; 4 variables with 2 MFs each.
215     Predict x(t+6) based on x(t-18), x(t-12), x(t-6), x(t)
216     These are the starting MFs values he suggests.
217     '''
218     invardefs = [
219         ('xm18', make_bell_mfs(0.444045, 2, [0.425606, 1.313696])),
220         ('xm12', make_bell_mfs(0.444045, 2, [0.425606, 1.313696])),
221         ('xm6', make_bell_mfs(0.444045, 2, [0.425606, 1.313696])),
222         ('x', make_bell_mfs(0.444045, 2, [0.425606, 1.313696])),
223     ]

```

```

224     outvars = ['xp6']
225     model = anfis.AnfisNet('Jang\'s example 4', invardefs, outvars)
226     return model
227
228
229 def jang_ex4_trained_model():
230     '''
231         Example 4 model, from Jang's data; 4 variables with 2 MFs each.
232         These are the final 'trained' values from pg. 683.
233     '''
234     # Data from Table VI:
235     mfs = [
236         (0.1790, 2.0456, 0.4798), # SMALL1
237         (0.1584, 2.0103, 1.4975), # LARGE1
238         (0.2410, 1.9533, 0.2960), # SMALL2
239         (0.2923, 1.9178, 1.7824), # LARGE2
240         (0.3798, 2.1490, 0.6599), # SMALL3
241         (0.4884, 1.8967, 1.6465), # LARGE3
242         (0.2815, 2.0170, 0.3341), # SMALL4
243         (0.1616, 2.0165, 1.4727), # LARGE4
244     ]
245     invardefs = [
246         ('xm18', [BellMembFunc(*mfs[0]), BellMembFunc(*mfs[1])]),
247         ('xm12', [BellMembFunc(*mfs[2]), BellMembFunc(*mfs[3])]),
248         ('xm6', [BellMembFunc(*mfs[4]), BellMembFunc(*mfs[5])]),
249         ('x', [BellMembFunc(*mfs[6]), BellMembFunc(*mfs[7])]),
250     ]
251     outvars = ['xp6']
252     model = anfis.AnfisNet('Jang\'s example 4 (trained)', invardefs, outvars)
253     # Jang calls this "the parameter matrix C" on pg 683:
254     coeff = torch.tensor([
255         [0.2167, 0.7233, -0.0365, 0.5433, 0.0276],
256         [0.2141, 0.5704, -0.4826, 1.2452, -0.3778],
257         [-0.0683, 0.0022, 0.6495, 2.7320, -2.2916],
258         [-0.2616, 0.9190, -2.9931, 1.9467, 1.6555],

```

```

259         [-0.3293, -0.8943, 1.4290, -1.6550, 2.3735],
260         [2.5820, -2.3109, 3.7925, -5.8068, 4.0478],
261         [0.8797, -0.9407, 2.2487, 0.7759, -2.0714],
262         [-0.8417, -1.5394, -1.5329, 2.2834, 2.4140],
263         [-0.6422, -0.4384, 0.9792, -0.3993, 1.5593],
264         [1.5534, -0.0542, -4.7256, 0.7244, 2.7350],
265         [-0.6864, -2.2435, 0.1585, 0.5304, 3.5411],
266         [-0.3190, -1.3160, 0.9689, 1.4887, 0.7079],
267         [-0.3200, -0.4654, 0.4880, -0.0559, 0.9622],
268         [4.0220, -3.8886, 1.0547, -0.7427, -0.4464],
269         [0.3338, -0.3306, -0.5961, 1.1220, 0.3529],
270         [-0.5572, 0.9190, -0.8745, 2.1899, -0.9497],
271     ])
272     model.coef = coef.unsqueeze(1) # add extra dim for output vars
273     return model
274
275
276 def jang_ex4_data(filename):
277     '''
278     Read Jang's data for the MG function to be modelled.
279     '''
280     num_cases = 500
281     x = torch.zeros((num_cases, 4))
282     y = torch.zeros((num_cases, 1))
283     with open(filename, 'r') as fh:
284         for i, line in enumerate(fh):
285             values = [float(v) for v in line.strip().split()]
286             x[i] = torch.tensor(values[0:4])
287             y[i] = values[4]
288     dl = DataLoader(TensorDataset(x, y), batch_size=1024, shuffle=True)
289     return dl
290
291
292 if __name__ == '__main__':
293     example = '1'

```

```

294     show_plots = True
295     if len(sys.argv) == 2: # One arg: example
296         example = sys.argv[1]
297         show_plots = False
298     print('Example {} from Jang\'s paper'.format(example))
299     if example == '1':
300         model = ex1_model()
301         train_data = make_sinc_xy()
302         a, b = train_data.dataset.tensors
303         cv_data = make_sinc_xy_test()
304         plot_all_mfs(model, a)
305         # train_anfis(model, train_data, 20, show_plots)
306         train_anfis_cv(model, [train_data, cv_data], 20, show_plots, metric="
rmse")
307     elif example == '2':
308         model = ex2_model()
309         train_data = ex2_training_data()
310         train_anfis(model, train_data, 200, show_plots)
311         test_data = ex2_testing_data()
312         test_anfis(model, test_data, show_plots)
313     elif example == '3':
314         model = ex3_model()
315         train_data = ex3_training_data()
316         train_anfis(model, train_data, 500, show_plots)
317         test_data = ex3_testing_data()
318         test_anfis(model, test_data, show_plots)
319     elif example == '4':
320         model = ex4_model()
321         train_data = jang_ex4_data('/content/my-anfis-pytorch/jang-example4-
data.trn')
322         train_anfis(model, train_data, 500, show_plots)
323         test_data = jang_ex4_data('/content/my-anfis-pytorch/jang-example4-data
.chk')
324         test_anfis(model, test_data, show_plots)
325     elif example == '4T':

```

```

326     model = jang_ex4_trained_model()
327     test_data = jang_ex4_data('/content/my-anfis-pytorch/jang-example4-data
    .trn')
328     test_anfis(model, test_data, show_plots)
329     test_data = jang_ex4_data('/content/my-anfis-pytorch/jang-example4-data
    .chk')
330     test_anfis(model, test_data, show_plots)
331 else:
332     print('ERROR - no such example')

```

و با تغییر خط کد زیر در آن به پیاده‌سازی مثال‌های یک تا سه با دستورات پایتون می‌پردازیم.

```

1 .
2 .
3 if __name__ == '__main__':
4     example = '1'
5     show_plots = True
6 .
7 .

```

نتایج به این شرح است. مثال اول:

```

1 Example 1 from Jang's paper
2 Figure(640x480)
3 Figure(640x480)
4 ### Training for 20 epochs, training size = 121 cases, test size = 9
5 epoch 0: Train rmse=0.10498, Test rmse=0.26764
6 Current LR: 0.001
7 -----
8 epoch 1: Train rmse=0.10496, Test rmse=0.26755
9 Current LR: 0.001
10 -----
11 epoch 2: Train rmse=0.10494, Test rmse=0.26744
12 Current LR: 0.001
13 -----
14 epoch 3: Train rmse=0.10491, Test rmse=0.26733
15 Current LR: 0.001

```



```
16 -----
17 epoch 4: Train rmse=0.10489, Test rmse=0.26722
18 Current LR: 0.001
19 -----
20 epoch 5: Train rmse=0.10486, Test rmse=0.26710
21 Current LR: 0.001
22 -----
23 epoch 6: Train rmse=0.10484, Test rmse=0.26699
24 Current LR: 0.001
25 -----
26 epoch 7: Train rmse=0.10481, Test rmse=0.26687
27 Current LR: 0.001
28 -----
29 epoch 8: Train rmse=0.10479, Test rmse=0.26675
30 Current LR: 0.001
31 -----
32 epoch 9: Train rmse=0.10476, Test rmse=0.26663
33 Current LR: 0.001
34 -----
35 epoch 10: Train rmse=0.10474, Test rmse=0.26651
36 Current LR: 0.001
37 -----
38 epoch 11: Train rmse=0.10471, Test rmse=0.26639
39 Current LR: 0.001
40 -----
41 epoch 12: Train rmse=0.10468, Test rmse=0.26626
42 Current LR: 0.001
43 -----
44 epoch 13: Train rmse=0.10466, Test rmse=0.26614
45 Current LR: 0.001
46 -----
47 epoch 14: Train rmse=0.10463, Test rmse=0.26602
48 Current LR: 0.001
49 -----
50 epoch 15: Train rmse=0.10460, Test rmse=0.26590
```

```

51 Current LR: 0.001
52 -----
53 epoch 16: Train rmse=0.10458, Test rmse=0.26577
54 Current LR: 0.001
55 -----
56 epoch 17: Train rmse=0.10455, Test rmse=0.26565
57 Current LR: 0.001
58 -----
59 epoch 18: Train rmse=0.10452, Test rmse=0.26553
60 Current LR: 0.001
61 -----
62 epoch 19: Train rmse=0.10450, Test rmse=0.26540
63 Current LR: 0.001
64 -----
65 Training time: 0.25s
66 min. test error: 0.2654042408778534
67 epoch: 20
68 Figure(640x480)
69 Figure(640x480)

```

مثال دوم:

```

1 Example 2 from Jang's paper
2 ### Training for 200 epochs, training size = 216 cases
3 epoch    0: MSE=0.36651, RMSE=0.60540 =4.12%
4 epoch   10: MSE=0.01686, RMSE=0.12987 =0.95%
5 epoch   20: MSE=0.00566, RMSE=0.07521 =0.55%
6 epoch   30: MSE=0.01632, RMSE=0.12775 =0.92%
7 epoch   40: MSE=0.00856, RMSE=0.09253 =0.67%
8 epoch   50: MSE=0.00914, RMSE=0.09562 =0.68%
9 epoch   60: MSE=0.00795, RMSE=0.08916 =0.64%
10 epoch  70: MSE=0.00595, RMSE=0.07713 =0.58%
11 epoch  80: MSE=0.00581, RMSE=0.07619 =0.58%
12 epoch  90: MSE=0.00596, RMSE=0.07720 =0.59%
13 epoch 100: MSE=0.00575, RMSE=0.07582 =0.58%
14 epoch 110: MSE=0.00541, RMSE=0.07356 =0.56%

```

```

15 epoch 120: MSE=0.00530, RMSE=0.07282 =0.56%
16 epoch 130: MSE=0.00522, RMSE=0.07222 =0.55%
17 epoch 140: MSE=0.00514, RMSE=0.07169 =0.55%
18 epoch 150: MSE=0.00497, RMSE=0.07051 =0.55%
19 epoch 160: MSE=0.00474, RMSE=0.06886 =0.53%
20 epoch 170: MSE=0.00455, RMSE=0.06744 =0.52%
21 epoch 180: MSE=0.00454, RMSE=0.06736 =0.52%
22 epoch 190: MSE=0.00461, RMSE=0.06791 =0.53%
23 Figure(640x480)
24 Figure(640x480)
25 ### Testing for 125 cases
26 R2 = 0.1599, MS error=80.36399, RMS error=8.96460, percentage=29.91%
27 [ ]
28 ! python /content/my-anfis-pytorch/jang_examples.py
29 Example 3 from Jang's paper
30 ### Training for 500 epochs, training size = 101 cases
31 epoch 0: MSE=0.00254, RMSE=0.05044 =341216903473019.25%
32 epoch 10: MSE=0.00052, RMSE=0.02280 =75276649661162.17%
33 epoch 20: MSE=0.00140, RMSE=0.03744 =193520492560828.47%
34 epoch 30: MSE=0.00073, RMSE=0.02699 =47869829657798.49%
35 epoch 40: MSE=0.00043, RMSE=0.02070 =57676333617827.32%
36 epoch 50: MSE=0.00038, RMSE=0.01941 =150335262809057.34%
37 epoch 60: MSE=0.00013, RMSE=0.01147 =33930989309844.00%
38 epoch 70: MSE=0.00041, RMSE=0.02017 =146785647788629.38%
39 epoch 80: MSE=0.00044, RMSE=0.02093 =219534825520476.75%
40 epoch 90: MSE=0.00008, RMSE=0.00873 =90342965431055.17%
41 epoch 100: MSE=0.00004, RMSE=0.00641 =18947206843860.84%
42 epoch 110: MSE=0.00003, RMSE=0.00571 =16983973886707.84%
43 epoch 120: MSE=0.00007, RMSE=0.00834 =32722265544107.71%
44 epoch 130: MSE=0.00006, RMSE=0.00797 =12651384600472.79%
45 epoch 140: MSE=0.00008, RMSE=0.00868 =49240165986538.27%
46 epoch 150: MSE=0.00008, RMSE=0.00921 =47453204625233.17%
47 epoch 160: MSE=0.00011, RMSE=0.01027 =1517679745246.80%
48 epoch 170: MSE=0.00008, RMSE=0.00890 =7212164048822.46%
49 epoch 180: MSE=0.00009, RMSE=0.00966 =33485749920579.51%

```

```

50 epoch 190: MSE=0.00006 , RMSE=0.00800 =15104747609066.02%
51 epoch 200: MSE=0.00007 , RMSE=0.00840 =2583331463615.87%
52 epoch 210: MSE=0.00005 , RMSE=0.00672 =8579113877186.52%
53 epoch 220: MSE=0.00005 , RMSE=0.00706 =42244176491231.25%
54 epoch 230: MSE=0.00005 , RMSE=0.00677 =40524512800682.72%
55 epoch 240: MSE=0.00005 , RMSE=0.00693 =14129778156278.16%
56 epoch 250: MSE=0.00004 , RMSE=0.00655 =15024156955978.40%
57 epoch 260: MSE=0.00004 , RMSE=0.00644 =38343585563460.94%
58 epoch 270: MSE=0.00004 , RMSE=0.00626 =42274460578960.35%
59 epoch 280: MSE=0.00004 , RMSE=0.00626 =25099811386889.20%
60 epoch 290: MSE=0.00004 , RMSE=0.00633 =17303591701952.42%
61 epoch 300: MSE=0.00004 , RMSE=0.00602 =31509348245933.83%
62 epoch 310: MSE=0.00004 , RMSE=0.00645 =43981904548544.38%
63 epoch 320: MSE=0.00003 , RMSE=0.00547 =32471691722424.50%
64 epoch 330: MSE=0.00003 , RMSE=0.00554 =20262015807128.02%
65 epoch 340: MSE=0.00003 , RMSE=0.00559 =22032052134708.68%
66 epoch 350: MSE=0.00003 , RMSE=0.00564 =37961184221644.68%
67 epoch 360: MSE=0.00003 , RMSE=0.00586 =29368361950796.48%
68 epoch 370: MSE=0.00003 , RMSE=0.00558 =37066147586446.25%
69 epoch 380: MSE=0.00003 , RMSE=0.00551 =18602306941809.42%
70 epoch 390: MSE=0.00003 , RMSE=0.00547 =22069588973838.55%
71 epoch 400: MSE=0.00003 , RMSE=0.00548 =32039687895034.98%
72 epoch 410: MSE=0.00003 , RMSE=0.00557 =24426696670495.81%
73 epoch 420: MSE=0.00003 , RMSE=0.00553 =39414356267627.40%
74 epoch 430: MSE=0.00003 , RMSE=0.00530 =29092308920379.11%
75 epoch 440: MSE=0.00003 , RMSE=0.00525 =29862993676952.71%
76 epoch 450: MSE=0.00003 , RMSE=0.00537 =29802781509782.59%
77 epoch 460: MSE=0.00003 , RMSE=0.00540 =19336582315016.17%
78 epoch 470: MSE=0.00003 , RMSE=0.00533 =28701411579078.13%
79 epoch 480: MSE=0.00003 , RMSE=0.00522 =26632265895174.48%
80 epoch 490: MSE=0.00003 , RMSE=0.00517 =29819206387308.64%

```

```
81 Figure(640x480)
```

```
82 Figure(640x480)
```

```
83 ### Testing for 699 cases
```

```
84 R2 = 0.9999 , MS error=0.00003 , RMS error=0.00536 , percentage=22.31%
```

## مثال سوم:

```

1 Example 3 from Jang's paper
2 ### Training for 500 epochs, training size = 101 cases
3 epoch    0: MSE=0.00254, RMSE=0.05044 =341216903473019.25%
4 epoch   10: MSE=0.00052, RMSE=0.02280 =75276649661162.17%
5 epoch   20: MSE=0.00140, RMSE=0.03744 =193520492560828.47%
6 epoch   30: MSE=0.00073, RMSE=0.02699 =47869829657798.49%
7 epoch   40: MSE=0.00043, RMSE=0.02070 =57676333617827.32%
8 epoch   50: MSE=0.00038, RMSE=0.01941 =150335262809057.34%
9 epoch   60: MSE=0.00013, RMSE=0.01147 =33930989309844.00%
10 epoch  70: MSE=0.00041, RMSE=0.02017 =146785647788629.38%
11 epoch  80: MSE=0.00044, RMSE=0.02093 =219534825520476.75%
12 epoch  90: MSE=0.00008, RMSE=0.00873 =90342965431055.17%
13 epoch 100: MSE=0.00004, RMSE=0.00641 =18947206843860.84%
14 epoch 110: MSE=0.00003, RMSE=0.00571 =16983973886707.84%
15 epoch 120: MSE=0.00007, RMSE=0.00834 =32722265544107.71%
16 epoch 130: MSE=0.00006, RMSE=0.00797 =12651384600472.79%
17 epoch 140: MSE=0.00008, RMSE=0.00868 =49240165986538.27%
18 epoch 150: MSE=0.00008, RMSE=0.00921 =47453204625233.17%
19 epoch 160: MSE=0.00011, RMSE=0.01027 =1517679745246.80%
20 epoch 170: MSE=0.00008, RMSE=0.00890 =7212164048822.46%
21 epoch 180: MSE=0.00009, RMSE=0.00966 =33485749920579.51%
22 epoch 190: MSE=0.00006, RMSE=0.00800 =15104747609066.02%
23 epoch 200: MSE=0.00007, RMSE=0.00840 =2583331463615.87%
24 epoch 210: MSE=0.00005, RMSE=0.00672 =8579113877186.52%
25 epoch 220: MSE=0.00005, RMSE=0.00706 =42244176491231.25%
26 epoch 230: MSE=0.00005, RMSE=0.00677 =40524512800682.72%
27 epoch 240: MSE=0.00005, RMSE=0.00693 =14129778156278.16%
28 epoch 250: MSE=0.00004, RMSE=0.00655 =15024156955978.40%
29 epoch 260: MSE=0.00004, RMSE=0.00644 =38343585563460.94%
30 epoch 270: MSE=0.00004, RMSE=0.00626 =42274460578960.35%
31 epoch 280: MSE=0.00004, RMSE=0.00626 =25099811386889.20%
32 epoch 290: MSE=0.00004, RMSE=0.00633 =17303591701952.42%
33 epoch 300: MSE=0.00004, RMSE=0.00602 =31509348245933.83%
34 epoch 310: MSE=0.00004, RMSE=0.00645 =43981904548544.38%

```

```
35 epoch 320: MSE=0.00003, RMSE=0.00547 =32471691722424.50%
36 epoch 330: MSE=0.00003, RMSE=0.00554 =20262015807128.02%
37 epoch 340: MSE=0.00003, RMSE=0.00559 =22032052134708.68%
38 epoch 350: MSE=0.00003, RMSE=0.00564 =37961184221644.68%
39 epoch 360: MSE=0.00003, RMSE=0.00586 =29368361950796.48%
40 epoch 370: MSE=0.00003, RMSE=0.00558 =37066147586446.25%
41 epoch 380: MSE=0.00003, RMSE=0.00551 =18602306941809.42%
42 epoch 390: MSE=0.00003, RMSE=0.00547 =22069588973838.55%
43 epoch 400: MSE=0.00003, RMSE=0.00548 =32039687895034.98%
44 epoch 410: MSE=0.00003, RMSE=0.00557 =24426696670495.81%
45 epoch 420: MSE=0.00003, RMSE=0.00553 =39414356267627.40%
46 epoch 430: MSE=0.00003, RMSE=0.00530 =29092308920379.11%
47 epoch 440: MSE=0.00003, RMSE=0.00525 =29862993676952.71%
48 epoch 450: MSE=0.00003, RMSE=0.00537 =29802781509782.59%
49 epoch 460: MSE=0.00003, RMSE=0.00540 =19336582315016.17%
50 epoch 470: MSE=0.00003, RMSE=0.00533 =28701411579078.13%
51 epoch 480: MSE=0.00003, RMSE=0.00522 =26632265895174.48%
52 epoch 490: MSE=0.00003, RMSE=0.00517 =29819206387308.64%
53 Figure(640x480)
54 Figure(640x480)
55 ### Testing for 699 cases
56 R2 = 0.9999, MS error=0.00003, RMS error=0.00536, percentage=22.31%
```