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

تمرین درس محاسبات نرم
در رشتهٔ مهندسی برق گرایش مهندسی کنترل

عنوان

تمرین سوم: ANFIS

نگارش

محمدجواد احمدی

استاد درس

دکتر مهدی علیاری شوره‌دلی

آذرماه ۱۴۰۱

فهرست مطالب

فهرست شکل‌ها

پاسخ سوالات

مثال اول

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

```
clc;
clear;
close all;

%% Load Data

f=@(x,y) sin(x)./x .* sin(y)./y;

xmin=-10;
xmax=10;
ymin=-10;
ymax=10;
x=linspace(xmin,xmax,26)';
y=linspace(ymin,ymax,26)';
F=f(x,y);

TrainInputs=[x,y];
TrainTargets=F;
TrainData=[TrainInputs TrainTargets];

xx=linspace(xmin,xmax,106)';
yy=linspace(ymin,ymax,106)';
```

```
FF=f(xx,yy);

TestInputs=[xx,yy];
TestTargets=FF;
TestData=[TestInputs TestTargets];


%% Design ANFIS

% nMFs=5;
% InputMF='gaussmf';
% OutputMF='linear';
%
% fis=genfis1(TrainData,nMFs,InputMF,OutputMF);

fis=genfis2(TrainInputs,TrainTargets,0.2);

MaxEpoch=100;
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 ...
```

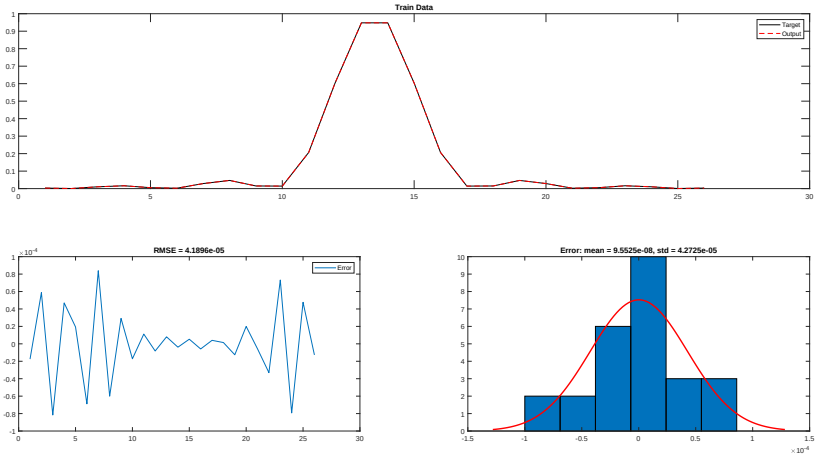
```
                DisplayFinalResult];

OptimizationMethod=1;
% 0: Backpropagation
% 1: Hybrid

fis=anfis(TrainData,fis,TrainOptions,DisplayOptions,[],OptimizationMethod);


%% Apply ANFIS to Train Data

TrainOutputs=evalfis(TrainInputs,fis);

TrainErrors=TrainTargets-TrainOutputs;
TrainMSE=mean(TrainErrors(:).^2);
TrainRMSE=sqrt(TrainMSE);
TrainErrorMean=mean(TrainErrors);
TrainErrorSTD=std(TrainErrors);

figure;
PlotResults(TrainTargets,TrainOutputs,'Train Data');

%% Apply ANFIS to Test Data

TestOutputs=evalfis(TestInputs,fis);

TestErrors=TestTargets-TestOutputs;
TestMSE=mean(TestErrors(:).^2);
TestRMSE=sqrt(TestMSE);
TestErrorMean=mean(TestErrors);
TestErrorSTD=std(TestErrors);

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

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

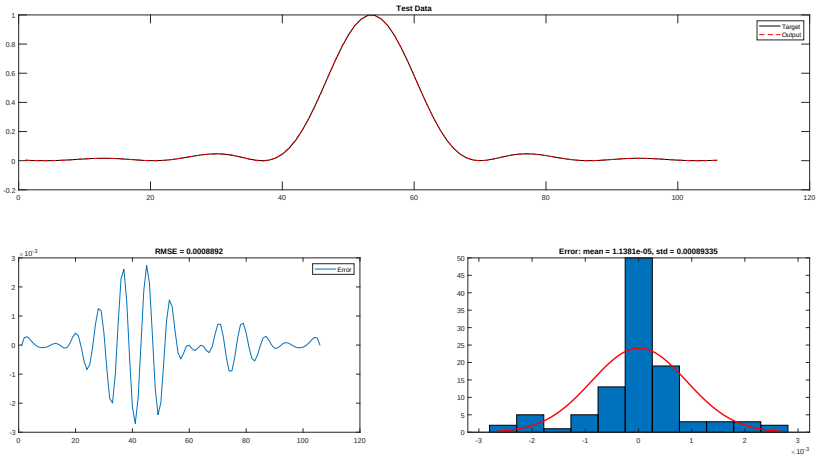
```
function PlotResults(targets,outputs,Name)

    errors=targets-outputs;

    RMSE=sqrt(mean(errors(:).^2));

    error_mean=mean(errors(:));
    error_std=std(errors(:));

    subplot(2,2,[1 2]);
    plot(targets,'k');
    hold on;
    plot(outputs,'r--');
    legend('Target','Output');
    title(Name);

    subplot(2,2,3);
    plot(errors);
    legend('Error');
    title(['RMSE = ' num2str(RMSE)]);

    subplot(2,2,4);
    histfit(errors);
    title(['Error: mean = ' num2str(error_mean) ', std = ' num2str(error_std)])
    ;

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.

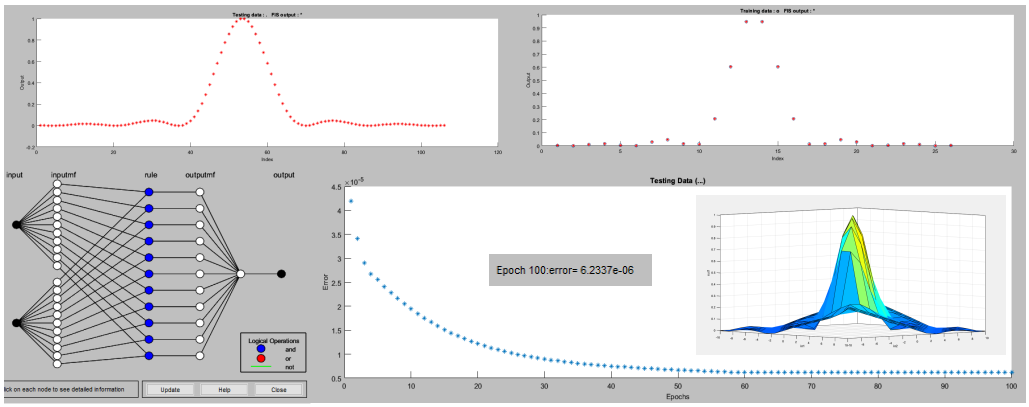
در ادامه همین فرآیند را از طریق GUI و دستور `anfisedit(fis)` پیگیری می‌کنیم. در گام اول متغیرها و داده‌ها را پس از اجرای m-فایل متلب فراخوانی می‌کنیم. برای نمایش بهتر و توضیحات مینیمال این فرآیند را در این ویدیو آورده شده است. نتایج هم به صورتی است که در شکل ۳ نشان داده شده است.



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:

```
clc;
clear;
close all;

%% Load Data

f=@(x,y,z) (1+x.^0.5+y.^-1+z.^(-1.5)).^2;

xmin=1;
xmax=6;
ymin=1;
ymax=6;
zmin=1;
zmax=6;
x=linspace(xmin,xmax,26)';
y=linspace(ymin,ymax,26)';
z=linspace(zmin,zmax,26)';
F=f(x,y,z);

TrainInputs=[x,y,z];
TrainTargets=F;
TrainData=[TrainInputs TrainTargets];

xx=linspace(xmin,xmax,101)';
yy=linspace(ymin,ymax,101)';
zz=linspace(zmin,zmax,101)';
```

```
FF=f(xx,yy,zz);

TestInputs=[xx,yy,zz];
TestTargets=FF;
TestData=[TestInputs TestTargets];


%% Design ANFIS

% nMFs=5;
% InputMF='gaussmf';
% OutputMF='linear';
%
% fis=genfis1(TrainData,nMFs,InputMF,OutputMF);

fis=genfis2(TrainInputs,TrainTargets,0.2);

MaxEpoch=100;
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;
% 0: Backpropagation
% 1: Hybrid

fis=anfis(TrainData,fis,TrainOptions,DisplayOptions,[],OptimizationMethod);


%% Apply ANFIS to Train Data

TrainOutputs=evalfis(TrainInputs,fis);

TrainErrors=TrainTargets-TrainOutputs;
TrainMSE=mean(TrainErrors(:).^2);
TrainRMSE=sqrt(TrainMSE);
TrainErrorMean=mean(TrainErrors);
TrainErrorSTD=std(TrainErrors);
TrainAPE= abs(TrainErrors(:))./abs(TrainTargets(:))*(26/100)

figure;
PlotResults(TrainTargets,TrainOutputs,'Train Data');

%% Apply ANFIS to Test Data

TestOutputs=evalfis(TestInputs,fis);

TestErrors=TestTargets-TestOutputs;
TestMSE=mean(TestErrors(:).^2);
TestRMSE=sqrt(TestMSE);
TestErrorMean=mean(TestErrors);
TestErrorSTD=std(TestErrors);
TestAPE= abs(TestErrors(:))./abs(TestTargets(:))*(101/100)

figure;
```

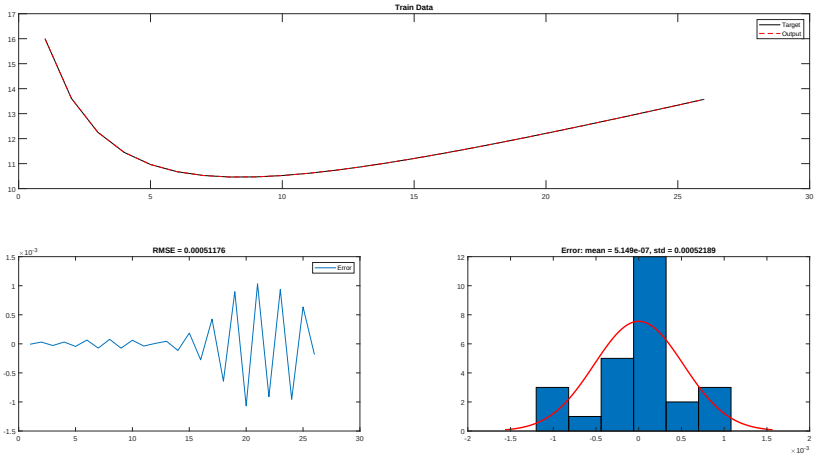
```
PlotResults(TestTargets,TestOutputs,'Test Data');
```

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

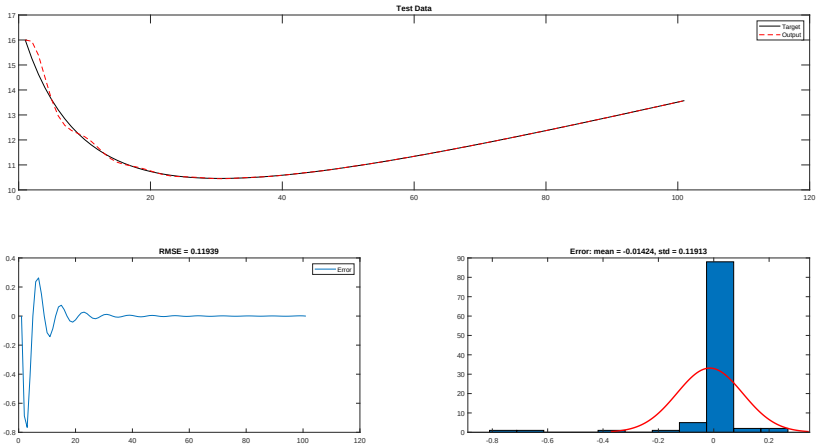
```
function PlotResults(targets,outputs,Name)

    errors=targets-outputs;

    RMSE=sqrt(mean(errors(:).^2));

    error_mean=mean(errors(:));
    error_std=std(errors(:));

    subplot(2,2,[1 2]);
    plot(targets,'k');
    hold on;
    plot(outputs,'r--');
    legend('Target','Output');
    title(Name);

    subplot(2,2,3);
    plot(errors);
    legend('Error');
    title(['RMSE = ' num2str(RMSE)]);

    subplot(2,2,4);
    histfit(errors);
    title(['Error: mean = ' num2str(error_mean) ', std = ' num2str(error_std)])
    ;

end
```

نتایج به صورتی است که در شکل ۴ و شکل ۵ 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

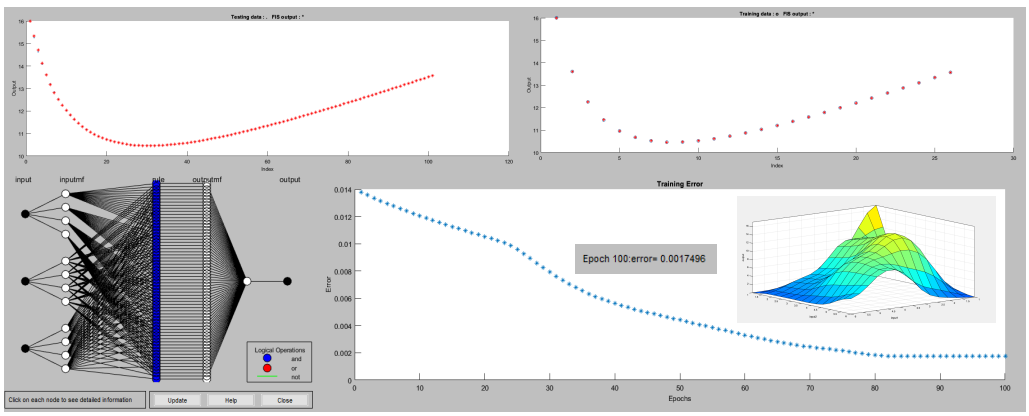


شکل ۴: نتایج مثال دوم (آموزش).



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.



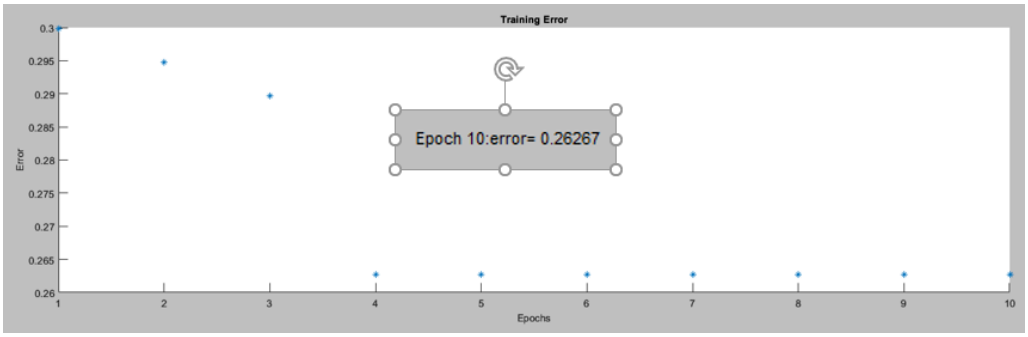
شکل ۶: نتایج GUI.

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

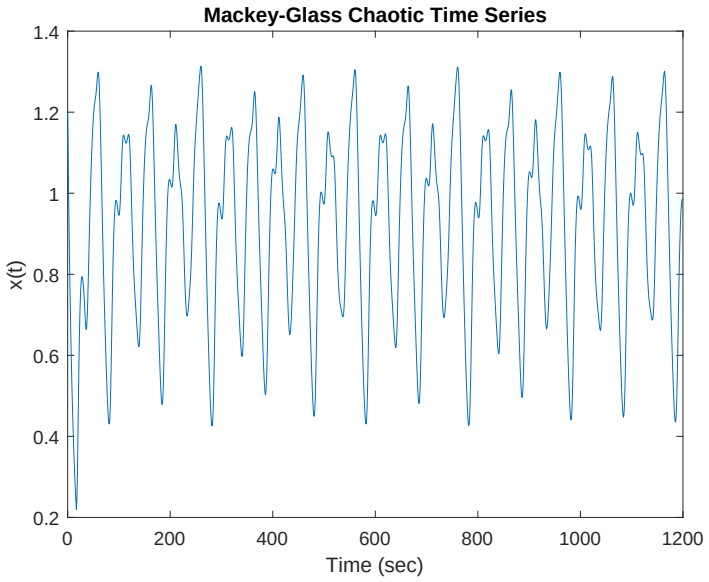
هدف این مثال از ANFIS پیش‌بینی یک سری زمانی تولیدی از معادله‌دیفرانسیل تأخیر زمانی مکی-گلس (رابطهٔ ۱) است.

$$\dot{x}(t) = \frac{۰/۲x(t-\tau)}{۱+x^{۱۰}(t-\tau)} - ۰/۱x(t) \qquad (۱)$$

این سری زمانی آشوب‌ناک است و دورهٔ زمانی مشخصی ندارد. این سری هم‌گرا یا واگرا نمی‌شود و خط سیر آن همانند سایر سیستم‌های آشوبناک بسیار حساس به شرایط اولیه است. برای به‌دست‌آوردن مقدار سری زمانی در نقاط صحیح، از روش رانگه‌کوتای مرتبه‌چهارم برای یافتن جواب عددی رابطهٔ ۱ استفاده می‌شود. فرض بر این است که برای زمان‌های کوچک‌تر از صفر، $x(t) = ۰, \tau = ۱۷, x(۰) = ۱/۲$ 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.

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



شکل ۸: سری زمانی Mackey-Glass (MG).

در پیش‌بینی سری‌های زمانی، از مقادیر شناخته‌شدهٔ 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 $((x(t-(D-۱)\Delta),\ldots,x(t-\Delta),x(t)))$ برای پیش‌بینی مقدار آینده $(x=(t+P))$ است. برمبنای تنظیمات مرسوم برای پیش‌بینی سری زمانی، $D=۴$ و $\Delta=P=۶$ در نظر گرفته می‌شود. برای هر t دادهٔ آموزشی ورودی یک بردار چهارستونهٔ به‌صورت زیر است:

$$w(t)=[x(t-۱۹),x(t-۱۲),x(t-۶),x(t)] \qquad (۲)$$

خروجی دادهٔ آموزش با توجه به مسیر پیش‌بینی به‌صورت زیر است:

$$s(t) = x(t + ۶) \tag{۳}$$

برای هر t، که از مقادیر ۱۱۸ تا ۱۱۱۷ متغیر است، ۱۰۰۰ نمونهٔ آموزشی ورودی/خروجی وجود دارد. برای این مثال، از ۵۰۰ نمونهٔ 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 نقطه‌نمونه 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:

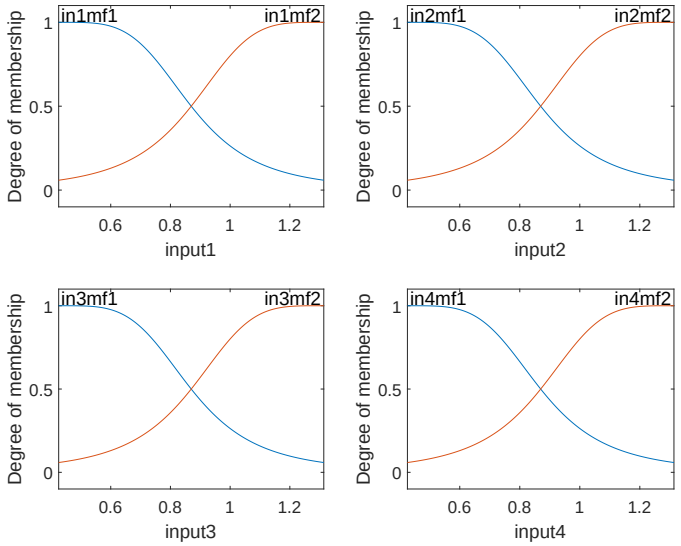
```
for t = 118:1117
    Data(t-117,:) = [x(t-18) x(t-12) x(t-6) x(t) x(t+6)];
end
trnData = Data(1:500,:);
chkData = Data(501:end,:);
```

در ادامه، یک شئ اولیهٔ 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 (به‌ترتیب چهار و یک) مطابقت دارد (شکل ۹).

```
figure
subplot(2,2,1)
plotmf(fis,'input',1)
subplot(2,2,2)
plotmf(fis,'input',2)
subplot(2,2,3)
plotmf(fis,'input',3)
subplot(2,2,4)
plotmf(fis,'input',4)
```

به‌طور پیش‌فرض، genfis دو تابع عضویت گاوسی (زنگوله‌ای) تعمیم‌یافته 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.

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



شکل ۹: توابع عضویت.

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

شئ فازی تولیدی، ۱۶ = $۲^۴$ قانون فازی با ۱۰۴ پارامتر شامل ۲۴ پارامتر غیرخطی و ۸۰ پارامتر خطی دارد. برای دستیابی به قابلیت تعمیم خوب، مهم است که تعداد نقاط داده آموزشی چندین برابر بیشتر از تعداد پارامترهای تخمین‌زده‌شده 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 گزینه‌های آموزشی را پیکربندی می‌کنیم و FIS را با داده‌های و گزینه‌های آموزشی تعیینی آموزش می‌دهیم:

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

داریم:

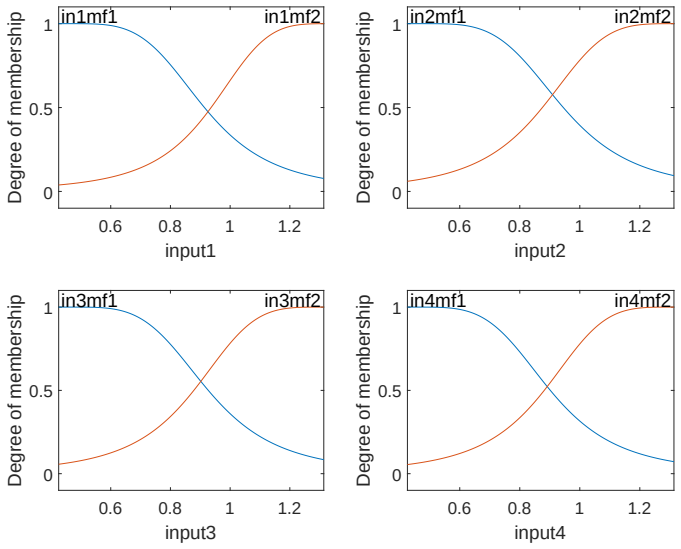
```
ANFIS info:
  Number of nodes: 55
  Number of linear parameters: 80
  Number of nonlinear parameters: 24
  Total number of parameters: 104
  Number of training data pairs: 500
  Number of checking data pairs: 500
```

```
  Number of fuzzy rules: 16


Start training ANFIS ...

1     0.00296046     0.00292488
2     0.00290346     0.0028684
3     0.00285048     0.00281544
4     0.00280117     0.00276566
Step size increases to 0.011000 after epoch 5.
5     0.00275517     0.00271874
6     0.00271214     0.00267438
7     0.00266783     0.00262818
8     0.00262626     0.00258435
Step size increases to 0.012100 after epoch 9.
9     0.00258702     0.00254254
10    0.00254972     0.00250247

Designated epoch number reached. ANFIS training completed at epoch 10.

Minimal training RMSE = 0.00254972
Minimal checking RMSE = 0.00250247
```

fis1 سیستم استنتاج فازی آموزش‌دیده برای دورهٔ آموزشی است که در آن خطای آموزشی کم‌ترین مقدار است. از آن‌جا که داده‌های اعتبارسنجی را هم مشخص کرده‌ایم، سیستم فازی با حداقل خطای اعتبارسنجی، fis2، نیز برگردانده می‌شود. سیستم با کم‌ترین خطای اعتبارسنجی، بهترین تعمیم را فراتر از داده‌های آموزشی نشان می‌دهد. حال توابع عضویت را برای سیستم آموزش دیده ترسیم می‌کنیم (شکل ۱۰).

```
figure
subplot(2,2,1)
plotmf(fis2,'input',1)
subplot(2,2,2)
plotmf(fis2,'input',2)
subplot(2,2,3)
plotmf(fis2,'input',3)
```

```
subplot(2,2,4)
plotmf(fis2,'input',4)
```



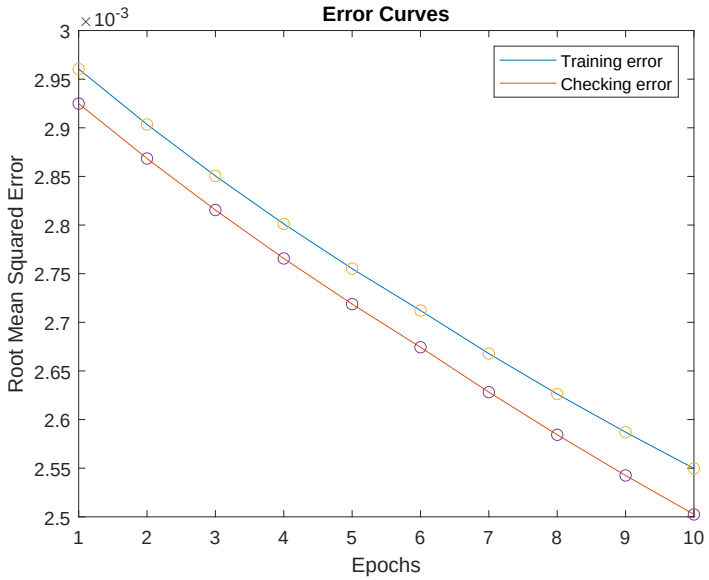
شکل ۱۰: توابع عضویت سیستم آموزش‌دیده.

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

```
figure
plot([error1 error2])
hold on
plot([error1 error2],'o')
legend('Training error','Checking error')
xlabel('Epochs')
ylabel('Root Mean Squared Error')
title('Error Curves')
```

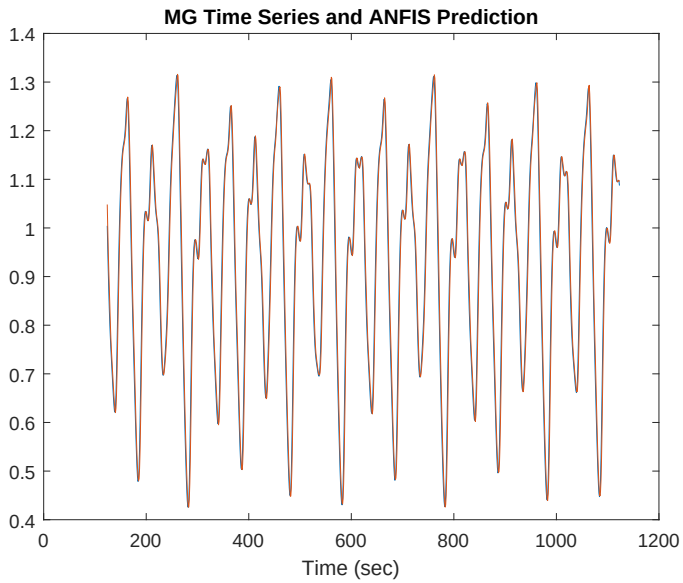
خطای آموزش در همهٔ دوره‌ها بیش‌تر از خطای اعتبارسنجی است. این پدیده در یادگیری ANFIS یا رگرسیون غیرخطی به طور کلی غیرمعمول نیست و می‌تواند نشان دهد که آموزش اضافی می‌تواند نتایج آموزشی بهتری ایجاد کند. برای بررسی قابلیت پیش‌بینی سیستم آموزش‌دیده، سیستم فازی را با استفاده از داده‌های آموزش و اعتبارسنجی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

```
anfis_output = evalfis(fis2,[trnData(:,1:4); chkData(:,1:4)]);

figure
index = 125:1124;
```



شکل ۱۱: نمودارهای خطا.

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

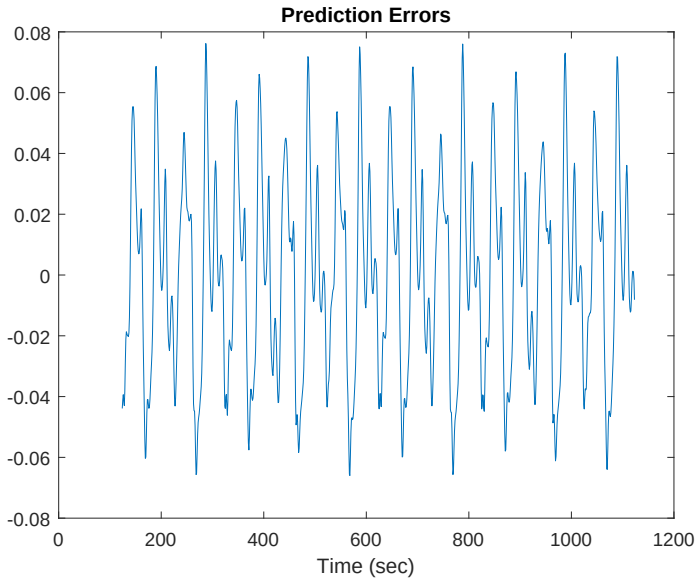


شکل ۱۲: نمودار اصلی و پیش‌بینی.

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

```
diff = x(index) - anfis_output;
```

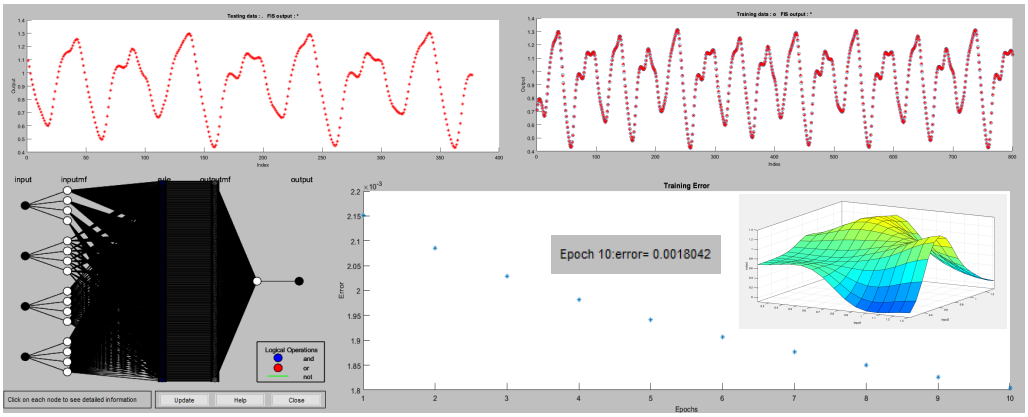
```
plot(time(index),diff)
xlabel('Time (sec)')
title('Prediction Errors')
```



شکل ۱۳: خطای پیش‌بینی.

مشاهده می‌شود که مقیاس نمودار خطای پیش بینی 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 مرجع این کدها گیت‌لب بوده و با ایجاد تغییراتی در فایل‌های اصلی آن را آماده برای اجرا در محیط کولب کرده‌ام. فایل‌های تغییر یافته و آمادهٔ 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:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
'''
    ANFIS in torch: Examples from Jang's paper
    @author: James Power <james.power@mu.ie> Apr 12 18:13:10 2019
'''

import sys
import itertools
import numpy as np

import torch
from torch.utils.data import TensorDataset, DataLoader

```

```
import anfis
from membership import BellMembFunc, make_bell_mfs
from experimental import train_anfis, test_anfis, train_anfis_cv, plot_all_mfs

dtype = torch.float64


# ##### Example 1: Modeling a Two-Input Nonlinear Function #####


def sinc(x, y):
    '''
        Sinc is a simple two-input non-linear function
        used by Jang in section V of his paper (equation 30).
    '''
    def s(z):
        return (1 if z == 0 else np.sin(z) / z)
    return s(x) * s(y)


def make_sinc_xy(batch_size=1024):
    '''
        Generates a set of (x, y) values for the sync function.
        Use the range (-10,10) that was used in sec. V of Jang's paper.
    '''
    pts = torch.arange(-10, 11, 2)
    x = torch.tensor(list(itertools.product(pts, pts)), dtype=dtype)
    y = torch.tensor([[sinc(*p)] for p in x], dtype=dtype)
    td = TensorDataset(x, y)
    return DataLoader(td, batch_size=batch_size, shuffle=True)


def make_sinc_xy_test(batch_size=1024):
    pts = torch.arange(-3, 3, 2)
    x = torch.tensor(list(itertools.product(pts, pts)), dtype=dtype)
```

```
    y = torch.tensor([[sinc(*p)] for p in x], dtype=dtype)
    td = TensorDataset(x, y)
    return DataLoader(td, batch_size=batch_size, shuffle=True)


def make_sinc_xy_large(num_cases=10000, batch_size=1024):
    '''
        Generates a set of (x, y) values for the sync function.
        Uses a large data set so we can test mini-batch in action.
    '''
    pts = torch.linspace(-10, 10, int(np.sqrt(num_cases)))
    x = torch.tensor(list(itertools.product(pts, pts)), dtype=dtype)
    y = torch.tensor([[sinc(*p)] for p in x], dtype=dtype)
    td = TensorDataset(x, y)
    return DataLoader(td, batch_size=batch_size, shuffle=True)


def make_sinc_xy2(batch_size=1024):
    '''
        A version of sinc with two outputs (sync(x) and 1-sync(x)).
        Not part of Jang's work, but used by the Vignette paper.
    '''
    pts = list(range(-10, 11, 2))
    x = torch.tensor(list(itertools.product(pts, pts)), dtype=dtype)
    y = torch.tensor([[sinc(*p), 1-sinc(*p)] for p in x], dtype=dtype)
    td = TensorDataset(x, y)
    return DataLoader(td, batch_size=batch_size, shuffle=True)


def ex1_model():
    '''
        These are the original (untrained) MFS for Jang's example 1.
    '''
    invardefs = [
            ('x0', make_bell_mfs(3.33333, 2, [-10, -3.333333, 3.333333, 10])),
```

```
            ('x1', make_bell_mfs(3.33333, 2, [-10, -3.333333, 3.333333, 10])),
            ]
    outvars = ['y0']
    anf = anfis.AnfisNet('Jang\'s example 1', invardefs, outvars)
    return anf


# ##### Example 2: Modeling a Three-Input Nonlinear Function #####

def ex2_eqn(x, y, z):
    '''
        The three input non-linear function used in Jang's example 2
    '''
    output = 1 + torch.pow(x, 0.5) + torch.pow(y, -1) + torch.pow(z, -1.5)
    output = torch.pow(output, 2)
    return output


def _make_data_xyz(inp_range):
    '''
        Given a range, return a dataset with the product of these values.
        Assume we want triples returned - i.e. (x,y,z) points
    '''
    xyz_vals = itertools.product(inp_range, inp_range, inp_range)
    x = torch.tensor(list(xyz_vals), dtype=dtype)
    y = torch.tensor([[ex2_eqn(*p)] for p in x], dtype=dtype)
    return TensorDataset(x, y)


def ex2_model():
    invardefs = [
            ('x', make_bell_mfs(2.5, 2, [1, 6])),
            ('y', make_bell_mfs(2.5, 2, [1, 6])),
            ('z', make_bell_mfs(2.5, 2, [1, 6])),
            ]
```

```
    outvars = ['output']
    model = anfis.AnfisNet('Jang\'s example 2', invardefs, outvars)
    return model


def ex2_training_data(batch_size=1024):
    '''
        Jang's training data uses integer values between 1 and 6 inclusive
    '''
    inp_range = range(1, 7, 1)
    td = _make_data_xyz(inp_range)
    return DataLoader(td, batch_size=batch_size, shuffle=True)


def ex2_testing_data():
    '''
        Jang's test data uses values 1.5, 2.5 etc.
    '''
    inp_range = np.arange(1.5, 6.5, 1)
    td = _make_data_xyz(inp_range)
    return DataLoader(td)


# ##### Example 3: On-line Identification in Control Systems #####


def ex3_model(mfnum=7):
    '''
        Example 3 model, with variable number of Bell MFs, range (-1,+1).
        Specify the no. of MFs, or make it 0 and I'll use Jang's 5 centers.
        Either way, the Bell width/slope values are from Jang's data.
    '''
    # The paper says 7 MFs are best, but his code uses 5 MFs
    if mfnum < 1:  # use the 5 MF values from Jang's code
        centers = [-0.999921, -0.499961, 0.000000, 0.499961, 0.99992]
```

```
    else:  # just spread them evenly accross the range (-1, +1)
        centers = np.linspace(-1, 1, mfnum)
    invardefs = [('k', make_bell_mfs(0.249980, 4, centers))]
    outvars = ['y']
    model = anfis.AnfisNet('Jang\'s example 3', invardefs, outvars)
    return model


def ex3_f(u):
    '''
        This is the function f defined in eq 34 of Jang's paper.
        This is the function that the ANFIS is supposed to model.
    '''
    pi_u = np.pi * u
    return (0.6 * torch.sin(pi_u) +
            0.3 * torch.sin(3 * pi_u) +
            0.1 * torch.sin(5 * pi_u))


def ex3_training_data(batch_size=1024):
    '''
        Jang's training data, spread evenly over -1 to +1
    '''
    inp_range = np.arange(-1, 1.02, 0.02)
    # Need to add an extra dimension to both x and y:
    x = torch.tensor(inp_range, dtype=dtype).unsqueeze(1)
    y = ex3_f(x)
    return DataLoader(TensorDataset(x, y), batch_size=batch_size, shuffle=True)


def ex3_u(k):
    '''
        This is the input function u(k) defined in eq 33 of Jang's paper.
        The purpose of this function is to generate (test) input values.
        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 valus generat ed by the u(k) function
204     '''
205     x = torch.tensor([ex3_u(k) for k in range(1, 700)], dtype=dtype).unsqueeze
    (1)
206     y = ex3_f(x)
207     return DataLoader(TensorDataset(x, y))
208
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],
```

```
        [-0.3293, -0.8943,  1.4290, -1.6550,  2.3735],
        [2.5820,  -2.3109,  3.7925, -5.8068,  4.0478],
        [0.8797,  -0.9407,  2.2487,  0.7759, -2.0714],
        [-0.8417, -1.5394, -1.5329,  2.2834,  2.4140],
        [-0.6422, -0.4384,  0.9792, -0.3993,  1.5593],
        [1.5534,  -0.0542, -4.7256,  0.7244,  2.7350],
        [-0.6864, -2.2435,  0.1585,  0.5304,  3.5411],
        [-0.3190, -1.3160,  0.9689,  1.4887,  0.7079],
        [-0.3200, -0.4654,  0.4880, -0.0559,  0.9622],
        [4.0220,  -3.8886,  1.0547, -0.7427, -0.4464],
        [0.3338,  -0.3306, -0.5961,  1.1220,  0.3529],
        [-0.5572,  0.9190, -0.8745,  2.1899, -0.9497],
    ])
    model.coeff = coeff.unsqueeze(1)  # add extra dim for output vars
    return model


def jang_ex4_data(filename):
    '''
        Read Jang's data for the MG function to be modelled.
    '''
    num_cases = 500
    x = torch.zeros((num_cases, 4))
    y = torch.zeros((num_cases, 1))
    with open(filename, 'r') as fh:
        for i, line in enumerate(fh):
            values = [float(v) for v in line.strip().split()]
            x[i] = torch.tensor(values[0:4])
            y[i] = values[4]
    dl = DataLoader(TensorDataset(x, y), batch_size=1024, shuffle=True)
    return dl


if __name__ == '__main__':
    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
```

```
----------------
epoch 4: Train rmse=0.10489, Test rmse=0.26722
Current LR: 0.001
----------------
epoch 5: Train rmse=0.10486, Test rmse=0.26710
Current LR: 0.001
----------------
epoch 6: Train rmse=0.10484, Test rmse=0.26699
Current LR: 0.001
----------------
epoch 7: Train rmse=0.10481, Test rmse=0.26687
Current LR: 0.001
----------------
epoch 8: Train rmse=0.10479, Test rmse=0.26675
Current LR: 0.001
----------------
epoch 9: Train rmse=0.10476, Test rmse=0.26663
Current LR: 0.001
----------------
epoch 10: Train rmse=0.10474, Test rmse=0.26651
Current LR: 0.001
----------------
epoch 11: Train rmse=0.10471, Test rmse=0.26639
Current LR: 0.001
----------------
epoch 12: Train rmse=0.10468, Test rmse=0.26626
Current LR: 0.001
----------------
epoch 13: Train rmse=0.10466, Test rmse=0.26614
Current LR: 0.001
----------------
epoch 14: Train rmse=0.10463, Test rmse=0.26602
Current LR: 0.001
----------------
epoch 15: Train rmse=0.10460, Test rmse=0.26590
```

```
Current LR: 0.001
-----------------
epoch 16: Train rmse=0.10458, Test rmse=0.26577
Current LR: 0.001
-----------------
epoch 17: Train rmse=0.10455, Test rmse=0.26565
Current LR: 0.001
-----------------
epoch 18: Train rmse=0.10452, Test rmse=0.26553
Current LR: 0.001
-----------------
epoch 19: Train rmse=0.10450, Test rmse=0.26540
Current LR: 0.001
-----------------
Training time: 0.25s
min. test error: 0.2654042408778534
epoch: 20
Figure(640x480)
Figure(640x480)
```

مثال دوم:

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

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

```
epoch  190: MSE=0.00006, RMSE=0.00800 =15104747609066.02%
epoch  200: MSE=0.00007, RMSE=0.00840 =2583331463615.87%
epoch  210: MSE=0.00005, RMSE=0.00672 =8579113877186.52%
epoch  220: MSE=0.00005, RMSE=0.00706 =42244176491231.25%
epoch  230: MSE=0.00005, RMSE=0.00677 =40524512800682.72%
epoch  240: MSE=0.00005, RMSE=0.00693 =14129778156278.16%
epoch  250: MSE=0.00004, RMSE=0.00655 =15024156955978.40%
epoch  260: MSE=0.00004, RMSE=0.00644 =38343585563460.94%
epoch  270: MSE=0.00004, RMSE=0.00626 =42274460578960.35%
epoch  280: MSE=0.00004, RMSE=0.00626 =25099811386889.20%
epoch  290: MSE=0.00004, RMSE=0.00633 =17303591701952.42%
epoch  300: MSE=0.00004, RMSE=0.00602 =31509348245933.83%
epoch  310: MSE=0.00004, RMSE=0.00645 =43981904548544.38%
epoch  320: MSE=0.00003, RMSE=0.00547 =32471691722424.50%
epoch  330: MSE=0.00003, RMSE=0.00554 =20262015807128.02%
epoch  340: MSE=0.00003, RMSE=0.00559 =22032052134708.68%
epoch  350: MSE=0.00003, RMSE=0.00564 =37961184221644.68%
epoch  360: MSE=0.00003, RMSE=0.00586 =29368361950796.48%
epoch  370: MSE=0.00003, RMSE=0.00558 =37066147586446.25%
epoch  380: MSE=0.00003, RMSE=0.00551 =18602306941809.42%
epoch  390: MSE=0.00003, RMSE=0.00547 =22069588973838.55%
epoch  400: MSE=0.00003, RMSE=0.00548 =32039687895034.98%
epoch  410: MSE=0.00003, RMSE=0.00557 =24426696670495.81%
epoch  420: MSE=0.00003, RMSE=0.00553 =39414356267627.40%
epoch  430: MSE=0.00003, RMSE=0.00530 =29092308920379.11%
epoch  440: MSE=0.00003, RMSE=0.00525 =29862993676952.71%
epoch  450: MSE=0.00003, RMSE=0.00537 =29802781509782.59%
epoch  460: MSE=0.00003, RMSE=0.00540 =19336582315016.17%
epoch  470: MSE=0.00003, RMSE=0.00533 =28701411579078.13%
epoch  480: MSE=0.00003, RMSE=0.00522 =26632265895174.48%
epoch  490: MSE=0.00003, RMSE=0.00517 =29819206387308.64%
Figure(640x480)
Figure(640x480)
### Testing for 699 cases
R2 = 0.9999, MS error=0.00003, RMS error=0.00536, percentage=22.31%
```

مثال سوم:

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

```
epoch  320: MSE=0.00003, RMSE=0.00547 =32471691722424.50%
epoch  330: MSE=0.00003, RMSE=0.00554 =20262015807128.02%
epoch  340: MSE=0.00003, RMSE=0.00559 =22032052134708.68%
epoch  350: MSE=0.00003, RMSE=0.00564 =37961184221644.68%
epoch  360: MSE=0.00003, RMSE=0.00586 =29368361950796.48%
epoch  370: MSE=0.00003, RMSE=0.00558 =37066147586446.25%
epoch  380: MSE=0.00003, RMSE=0.00551 =18602306941809.42%
epoch  390: MSE=0.00003, RMSE=0.00547 =22069588973838.55%
epoch  400: MSE=0.00003, RMSE=0.00548 =32039687895034.98%
epoch  410: MSE=0.00003, RMSE=0.00557 =24426696670495.81%
epoch  420: MSE=0.00003, RMSE=0.00553 =39414356267627.40%
epoch  430: MSE=0.00003, RMSE=0.00530 =29092308920379.11%
epoch  440: MSE=0.00003, RMSE=0.00525 =29862993676952.71%
epoch  450: MSE=0.00003, RMSE=0.00537 =29802781509782.59%
epoch  460: MSE=0.00003, RMSE=0.00540 =19336582315016.17%
epoch  470: MSE=0.00003, RMSE=0.00533 =28701411579078.13%
epoch  480: MSE=0.00003, RMSE=0.00522 =26632265895174.48%
epoch  490: MSE=0.00003, RMSE=0.00517 =29819206387308.64%
Figure(640x480)
Figure(640x480)
### Testing for 699 cases
R2 = 0.9999, MS error=0.00003, RMS error=0.00536, percentage=22.31%
```