بخش اول: توضیحات اصلی کد

در این گزارش ابتدا گام های و صفحه اصلی توضیح داده خواهد شد و در ادامه توابع داخلی جدا توضیح داده میشود (نتایج در بخش سوم در صفحه 11 آورده شده است)

در ابتدا داده ها لود میشوند و در ادامه با تابع CreateInitialFIS فازی ابتدایی و شبکه آن را با ساختار Takagi Sugeno ساخته میشود در ادامه شبکه فازی با روش ژنتیک با تابع TrainFuzzyUsingGA اموزش دیده میشود. (توابع در ادامه توضیح داده خواهد شد)

```
%% Load Data
data=LoadData();

%% Generate Basic FIS
fis=CreateInitialFIS(data);
fuzzy(fis);

%% Tarin Using GA
fis=TrainFuzzyUsingGA(fis,data);
```

در قسمت بعد داده های تست و ترین هم جدا به شبکه فازی اموزش دیده شده برای آزمایش داده میشود .

```
% Train Data
TrainOutputs=evalfis(data.TrainInputs,fis);
PlotResults(data.TrainTargets,TrainOutputs,'Train Data');

% Test Data
TestOutputs=evalfis(data.TestInputs,fis);
PlotResults(data.TestTargets,TestOutputs,'Test Data');
```

در ادامه با تابع mse نتایج سنجیده میشود.

```
disp("MSE Train = " + mse(data.TrainTargets, TrainOutputs));
disp("MSE Test = " + mse(data.TestTargets, TestOutputs));
```

در قسمت اخر با تابع ShowTestPractice داده های داخل تمرین سنجیده میشوند.

```
ShowTestPractice(fis,0.5);
ShowTestPractice(fis,1.25);
ShowTestPractice(fis,2.7);
ShowTestPractice(fis,3);
ShowTestPractice(fis,4.2);
```

بخش دوم: توضیحات توابع

تابع **LoadData**:
در این تابع داده ایجاد و با سایز 30 و 70 داده ها جدا میشوند.

```
N=100;
[x1_train,y_train] = GenerateDatasetY(round(0.70*N));
[x1_test,y_test] = GenerateDatasetY(round(0.30*N));

% Train Data
TrainInputs=x1_train';
TrainTargets=y_train';

% Test Data
TestInputs=x1_test';
TestTargets=y_test';

% Export
data.TrainInputs=TrainInputs;
data.TrainTargets=TrainTargets;
data.TestInputs=TestInputs;
data.TestTargets=TestTargets;
```

برای ایجاد دیتا ست با استفاده از فرمول داخل تمرین و تابع GenerateDatasetY داده ها ایجاد میشوند.

```
function [x,y] = GenerateDatasetY(P)
    x = linspace(0,5,P);
    y =x .* cos(x).* exp(-x);
end
```

تابع **CreateInitialFIS**:

در این تابع که برای ساخت شبکه فازی اولیه است با استفاده از توابع داخلی متلب اقدادم به ایجاد شبکه فازی میکنیم.

```
function fis=CreateInitialFIS(data)
    options = genfisOptions('FCMClustering','MaxNumIteration' ,100);

    fis=genfis(data.TrainInputs,data.TrainTargets,options);
end
```

(تابع genfis به شکل دیفالت شبکه Sugeno ایجاد میکند)

تابع **TrainFuzzyUsingGA** :

در این تابع اقدام به آموزش شبکه با الگوریتم ژنتیک میکنیم.

```
    %% Problem Definition

    p0=GetFISParams(fis);

    Problem.CostFunction=@(x) TrainFISCost(x,fis,data);

    Problem.nVar=numel(p0);

     %% GA Params
    Params.MaxIt=100;
    Params.nPop=25;
```

در این تابع ابتدا مقادیر و تابع هزینه ایجاد میشود.

در ادامه تابع ژنتیک (RunGA) اجرا می شود و خروجی ژنتیک که شامل یک سری BestSolution ها است دریافت میکنیم و به membership functions های فازی به کمک تابع SetFISParams داده میشود.

```
%% Run GA
results=RunGA(Problem,Params);

%% Get Results

p=results.BestSol.Position.*p0;
bestfis=SetFISParams(fis,p);
```

تابع **SetFISParams**:

در این تابع با استفاده از خروجی الگوریتم ژنتیک برای ورودی و خروجی اقدام به ایجاد membership function میکنیم.

```
function fis=SetFISParams(fis,p)

    nInput=numel(fis.input);
    for i=1:nInput
        nMF=numel(fis.input(i).mf);
        for j=1:nMF
            k=numel(fis.input(i).mf(j).params);
            fis.input(i).mf(j).params=p(1:k);
            p(1:k)=[];
        end
    end

    nOutput=numel(fis.output);
    for i=1:nOutput
        nMF=numel(fis.output(i).mf);
        for j=1:nMF
            k=numel(fis.output(i).mf(j).params);
            fis.output(i).mf(j).params=p(1:k);
            p(1:k)=[];
        end
    end

end
```

تابع **RunGA**:

که در این تابع الگوریتم ژنتیک را اجرا میکنیم.

در ابتدا مقادیر را تنظیم میکنیم و یکسری متغیر ها را ایجاد میکنیم.

```
%% GA Parameters

MaxIt=Params.MaxIt;      % Maximum Number of Iterations

nPop=Params.nPop;        % Population Size

pc=0.9;                  % Crossover Percentage
nc=2*round(pc*nPop/2);   % Number of Offsprings (Parnets)

pm=0.2;                  % Mutation Percentage
nm=round(pm*nPop);       % Number of Mutants

gamma=0.7;

mu=0.15;         % Mutation Rate

beta=8;          % Selection Pressure
```

در ادامه حلقه اصلی را به تعداد تکرار های تنظیم شده که 1000 بود اجرا میکنیم
در این حلقه با یک احتمال رندوم و با چرخ رولت تعیین میکنیم که چه والد هایی باید انتخاب شوند.
بعد انتخاب والد های تابع Crossover را روی آن ها اعمال می کنیم و در ادامه تابع cost را هم اعمال میکنیم.

```
P=exp(-beta*Costs/WorstCost);
P=P/sum(P);

% Crossover
popc=repmat(empty_individual,nc/2,2);
for k=1:nc/2

    % Select Parents Indices
    i1=RouletteWheelSelection(P);
    i2=RouletteWheelSelection(P);

    % Select Parents
    p1=pop(i1);
    p2=pop(i2);

    % Apply Crossover
    [popc(k,1).Position, popc(k,2).Position]=...
        Crossover(p1.Position,p2.Position,gamma,VarMin,VarMax);

    % Evaluate Offsprings
    popc(k,1).Cost=CostFunction(popc(k,1).Position);
    popc(k,2).Cost=CostFunction(popc(k,2).Position);

end
```

در ادامه در قسمت جهش را اعمال می کنیم که در این حلقه ابتدا یک فردی به شکل رندوم انتخاب میشود تا روی آن جهش انجام شود بعد عمل جهش تابع هزینه آن را حساب میکنیم.

```
% Mutation
popm=repmat(empty_individual,nm,1);
for k=1:nm

    % Select Parent
    i=randi([1 nPop]);
    p=pop(i);

    % Apply Mutation
    popm(k).Position=Mutate(p.Position,mu,VarMin,VarMax);

    % Evaluate Mutant
    popm(k).Cost=CostFunction(popm(k).Position);

end
```

در ادامه روی جمعیت یک مرتب سازی اعمال می شود و در ادامه اعضای اضافی را حذف میکنیم.

در مرتب سازی انجام شده عضو بهتر در اول قرار میگیرد و آن را به عنوان بهترین راه حل انتخاب میکنیم.

```
% Sort Population
Costs=[pop.Cost];
[Costs, SortOrder]=sort(Costs);
pop=pop(SortOrder);

% Truncation
pop=pop(1:nPop);
Costs=Costs(1:nPop);

% Store Best Solution Ever Found
BestSol=pop(1);
```

تابع **Crossover**:

در این تابع دو والد با هم ترکیب و فرزندان ایجاد میشود در این روش یک سری از ژن های کروموزوم والد اول با والد دوم با یک alpha ترکیب میشوند و فرزندان ایجاد میشوند.

```
function [y1, y2]=Crossover(x1,x2,gamma,VarMin,VarMax)

    alpha=unifrnd(-gamma,1+gamma,size(x1));

    y1=alpha.*x1+(1-alpha).*x2;
    y2=alpha.*x2+(1-alpha).*x1;

    y1=max(y1,VarMin);
    y1=min(y1,VarMax);

    y2=max(y2,VarMin);
    y2=min(y2,VarMax);

end
```

تابع **Mutate**:

در این تابع عمل جهش انجام میشود در این تابع یک ژن به تصادف انتخاب و تغییر میکند.

```
function y=Mutate(x,mu,VarMin,VarMax)
    nVar=numel(x);
    nmu=ceil(mu*nVar);
    j=randsample(nVar,nmu)';
    sigma=0.1*(VarMax-VarMin);
    y=x;

    y(j)=x(j)+sigma*randn(size(j));
    y=max(y,VarMin);
    y=min(y,VarMax);
end
```

تابع **ShowTestPractice :**
برای گرفتن موارد خواسته شده در تمرین با این تابع با گرفتن شبکه فازی و عدد خواسته شده اقدام به گرفتن خروجی میگیریم برای این کار مقدار عدد خواسته شده با شبکه فازی به تابع evalfis داده میشود و خروجی پیش بینی شده شبکه در y_hat قرار می گیرد که در ادامه آن را چاپ میکنیم .

```
function ShowTestPractice(fis,number)
    y_hat=evalfis(number,fis);
    disp("y : " + number + " = " + GenerateDatasetY2(number));
    disp("y_hat : " + number + " = " + y_hat);
    disp("--------------");
end
```

(تابع 2GenerateDatasetY مقدار اصلی داده خاسته شده را حساب میکند)

بخش سوم: خروجی

در این بخش در مورد خروجی ها توضیح می دهیم.
مقادیر که برای این روش استفاده شده با 500 دیتا در 1000 دور آموزش بود
مقادیر پارامتر های الگوریتم ژنتیک در زیر آورده شده است

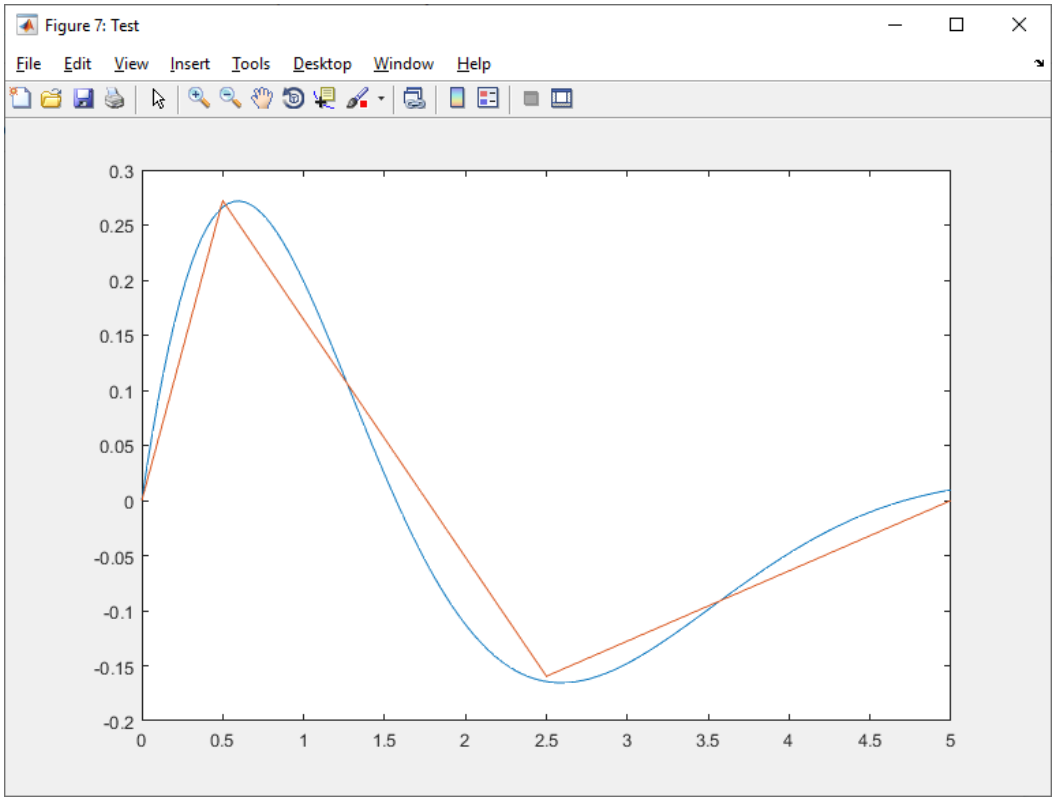
تعداد نسل اولیه : 25
درصد Crossover به تعداد : 0.9
درصد Mutation به تعداد : 0.2
نرخ Mutation به تعداد : 0.15

توضیحات در مورد نحوه اعمال اپراتورهای crossover و mutation در توابع مربوطه در بخش دوم آورده شده است (صفحه 10)

* نکته در جدول داده شده با توجه به کد دانشجویی من که 99464101 است هر دو مقدار 1 و 2 مقدار 0.5 است.

$\hat{y}$	y	x	
0.26199	0.26614	0.5	1
0.26199	0.26614	0.5	2
0.11005	0.11293	1.25	3
0.15942-	0.16405-	2.7	4
0.15388-	0.14787-	3	5
0.015057-	0.030877-	4.2	6

خروجی نموداری



مقادیر MSE

MSE Train = 0.000007794
MSE Test = 0.000009515