سید مجید نورانی ۹۶۱۲۳۰۴۴ تمرین ۴

۵ فایل صوتی که محتوای آنها به ترتیب آمریکا، سنگاپور، روسیه، نیجریه و آمریکا میباشد ضبط شده و سکوت ابتدا و انتهای هرکدام حذف گردید.

برنامه‌ی الگوریتم کلی این تمرین در فایل DTW نوشته شده است.

با استفاده از برنامه ی نوشته شده برای اسخراج ضرایب MFCC، ۱۲ ضریب کپسترال برای هر فریم از هر فایل بدست می‌آید.

```
function [c]= MFCC(audiofile)

[frames,Fs,N,FrameNo]=framming(audiofile);
[pframes]=preemphasis(frames);
[Hframes]=Hamming(pframes,N,FrameNo);
[STFf,frq,points]=FFT_of_Frames(Hframes,Fs);
[mj,melNo]= Melfilterbank(STFf,frq,FrameNo,points);
[c]= Cepstral_12(FrameNo,mj,melNo);
```

این تابع متشکل از ۶ تابع است

تابع **framming** وظیفه‌ی فریمینگ را برعهده دارد:

```
function [f,fs,N,frameNo]=framming(audiofile)

[x,fs]=audioread(audiofile);
x=x*32767;
l=length(x);  %number of the samples of the whole audio
N=25*fs/1000;  %length of frames in terms of samples
M=10*fs/1000;   %distance between frames in terms of samples
frameNo=fix(l/M)-3;    %quantity of frames
f(1:frameNo,1:N)=0;
for k=1:frameNo
    for s=((M*k)+1):(N+(M*k))
        f(k,(s-(k*M)))=x(s);%f: a matrix which in each row there is a frame
    end
end
```

این تابع فایل صوتی را گرفته و به ترتیب:

ماتریس متشکل از فریم‌ها که در هر سطر یک فریم قرار دارد :F

فرکانس نمونه برداری :Fs

طول هر فریم :N

frameNo: تعداد فریم‌ها

را به عنوان خروجی تحویل میدهد.

تابع **preemphasis**:

پیش تاکید را بر روی فریم ها انجام میدهد.

```
function [pframes]= preemphasis(frames)
alfa=0.95;
f1=frames(:,1);
f=[f1 frames];  %repeating the first column for preemphasis
sz=size(f);
pframes=zeros(sz-1);
for i=1:sz(1)
    for k=2:sz(2)
        pframes(i,k-1)=f(i,k)-alfa*f(i,k-1);       %preemphasis
    end
end
```

تابع **Hamming:**

پنجره‌ی همینگ را به هر فریم اعمال میکند. این تابع، فریم‌ها، طول فریم و تعداد فریمها را به عنوان ورودی گرفته و ماتریسی شامل فریم‌ها بعد از اعمال پنجره‌ی همینگ را ارائه میدهد.

```
function [Hf]= Hamming(frames,N,frameNo)
n=1:N;
w(n)=0.54-0.46*cos(2*pi*n/(N-1));   %hamming window
Hf=zeros(size(frames));
for k=1:frameNo
    for i=1:N
    Hf(k,i)=frames(k,i)*w(i);   %multiplication of hamming window in frames
    end
end
```

تابع **FFT_Of_Frames:**

این تابع ماتریس خروجی تابع قبل را گرفته و FFT فریم‌ها را برای ۵۱۲ نقطه، فرکانس مربوط به هر نقطه و تعداد نقاط را تحویل میدهد.

```
function [STFf,frq,points]= FFT_of_Frames(frames,Fs)
Tf=frames.'; %transpose f to be able to compute FFT of the matrix- FFT computes vectors in colomns
points=512;   %ponits for FFT
STFf1=fft(Tf,points);    %short time frequency spectrum of f
STFf=abs(STFf1.');
frq=Fs*(0:points/2)/points;
```

تابع **Melfilterbanks:**

این تابع FFT فریم‌ها را گرفته و انرژی نمونه‌های داخل هر فیلتر(mj) و تعداد فیلترها را تحویل میدهد. در ابتدا تعداد فیلترها و تابع تبدیل فرکانسها به فرکانس مل و فرکانس اولیه و انتهایی تعریف میشوند. سپس متغیر hbandwidth که معرف نصف پهنای باند فیلترهای مل در مقیاس مل محاسبه میشود. در ادامه مرکز فیلترها بدست می‌آید و در حلقه‌ی بعدی این مراکز از مقیاس مل خارج شده و فرکانس هرکدام بدست میاید. حال هر نقطه به عنوان ورودی به خطی که بین هر دو مرکز تعریف میشود داده میشود و نتیجه‌ی آن در متغیر melresult قرار میگیرد که از ان برای محاسبه‌ی انژی هر فیلتر استفاده میشود.

```
function [mj,melNo]= Melfilterbank(STFf,frq,frameNo,points)

melNo=25; %number of mel filters
syms mel(freq)
mel(freq)=1127*log(1+freq/700);             % mel scale
center=zeros(1,(melNo+2));
freq1=7000;
freq0=50;
center(1)=mel(freq0);



hbandwidth=(mel(freq1)-mel(freq0))/(melNo+1);% half of bandwidth of each mel filter in mel scale
for i=2:melNo+2
    center(i)=center(i-1)+hbandwidth;  % center of each filter in mel scale
end



for i=1:melNo+2
    center(i)=(exp(center(i)/1127)-1)*700; %moving centers to normal frequency scale
end
mj=zeros(frameNo,melNo);
for i=1:melNo
    for j=1:frameNo
        for k=1:(points/2)+1
            if frq(k)>center(i) && frq(k)<center(i+1)
                melresult=STFf(j,k)*((1/(center(i+1)-center(i)))*(frq(k)-center(i)));
            elseif frq(k)>center(i+1) && frq(k)<center(i+2)
                melresult=STFf(j,k)*((-1/(center(i+2)-center(i+1)))*(frq(k)-center(i+2)));
            else
                melresult=0;
            end
            mj(j,i)=melresult^2+mj(j,i);
        end
    end
end
mj=log10(mj);
```

در نهایت تابع **cepstral_12:**

این تابع تعداد فریمها، انرزی و تعداد فیلترها را گرفته و ۱۲ ضریب کپسترال برای هر فریم را در یک ماتریس تحویل میدهد.

```
function [c]= Cepstral_12(frameNo,mj,melNo)
c(1:frameNo,1:12)=0;
for k=1:frameNo
    for i=1:12
        for j=1:melNo
            c(k,i)=mj(k,j)*cos(pi*i*(j-0.5)/melNo)+c(k,i);
        end
    end
end
c=sqrt(2/melNo)*c;
```

تا اینجا ۱۲ ضریب کپسترال توسط تابع MFCC قابل استراج هستند. در فایل DTW که فایل اصلی برای محاسبه‌ی فاصله‌ی بین الگوی تست و الگوهای مرجع است داریم:

معرفی ورودیها:

```
clc
close all;
clear all;
learning_pattern1='E:\University\AUT\ASR\Exercises\ex4\america.wav';
learning_pattern2='E:\University\AUT\ASR\Exercises\ex4\singapore.wav';
learning_pattern3='E:\University\AUT\ASR\Exercises\ex4\russia.wav';
learning_pattern4='E:\University\AUT\ASR\Exercises\ex4\nigeria.wav';
test_pattern='E:\University\AUT\ASR\Exercises\ex4\america2.wav';
```

ضرایب MFCC و طول هر الگو:

```
[cl1]=MFCC(learning_pattern1);   %12 cepstral coefficients of learning pattern1
[cl2]=MFCC(learning_pattern2);  %12 cepstral coefficients of learning pattern2
[cl3]=MFCC(learning_pattern3);  %12 cepstral coefficients of learning pattern3
[cl4]=MFCC(learning_pattern4);  %12 cepstral coefficients of learning pattern4
[ctest]=MFCC(test_pattern);  %12 cepstral coefficients of test pattern

Tx=length(ctest);
Ty1=length(cl1);
Ty2=length(cl2);
Ty3=length(cl3);
Ty4=length(cl4);
Qmax=2;
ix=1;
iy=1;
```

ادامه‌ی برنامه در یک حلقه‌ی for نوشته میشود که ۴ بار تکرار میشود تا الگوی تست را با همه‌ی الگوهای مرجع مقایسه کند. در هر تکرار اطلاعات مورد نیاز هر الگو برای مقایسه در متغیرهای Ty(طول الگوی مرجع) و cl (ضرایب کپسترال هر الگوی مرجع) ذخیره میشود.

```
for ref=1:4
    if ref==1
        Ty=Ty1;
        cl=cl1;
    elseif ref==2
        Ty=Ty2;
        cl=cl2;
    elseif ref==3
        Ty=Ty3;
        cl=cl3;
    elseif ref==4
        Ty=Ty4;
        cl=cl4;
    end
```

برای پیدا کردن فاصله‌ی هر بردار از الگوی تست با هر بردار از الگوی مرجع تابع Zeta مورد استفاده قرار میگیرد که به شرح زیر است.

دو بردار شامل ۱۲ ضریب را گرفته و فاصله‌ی اقلیدسی آنها را محاسبه میکند.

```
function [d]= Zeta(ct,cl)
d=0;
for i=1:12
    d=(ct(i)-cl(i))^2+d;
end
d=sqrt(d);
```

محدوده‌ی مجاز برای حرکت بین بردارها توسط تابع Legal_range بدست می‌آید. ۴ خط مربوط به اضلاع متوازی‌الاضلاع در هر شرط بررسی میشود و اگر داخل محدوده بود، متغیر inside مقدار ۱ میگیرد و در غیر اینصورت مقدار صفر.

```
function [inside]= LegalRange(Tx,Ty,Qmax,ix,iy)

if iy>=((ix-1)/Qmax)+1
   if iy<=Qmax*(ix-1)+1
       if iy>=Qmax*(ix-Tx)+Ty
           if iy<=(ix-Tx)/Qmax+Ty
               inside=1;
           else
               inside=0;
           end
       else
           inside=0;
       end
   else
       inside=0;
   end
else
    inside=0;
end
```

از این توابع به صورت زیر در فایل DTW استفاده میشود.

در این قسمت ماتریس Z که شامل zeta مربوط به تمامی نقاط داخل متوازی الاضلاع است، بدست می‌آید. در واقع اگر نقطه‌ی مورد بررسی در داخل محدوده‌ی متوازی‌الاضلاع باشد فاصله‌ی بردارهایمربوطه حساب میشود و اگر نباشد در ماتریس Z مقدار بینهایت قرار میگیرد.

```
    %% local distance

Z=zeros(Tx,Ty);
for ix=1:Tx
    for iy=1:Ty
        [inside]= LegalRange(Tx,Ty,Qmax,ix,iy);
        if inside==1
            Z(ix,iy)= Zeta(ctest(ix,:),cl(iy,:));
        else
            Z(ix,iy)=inf;
        end
    end
end
```

بعد از محاسبه‌ی هزینه‌ی نقطه به نقطه حال باید هزینه‌ی کلی محاسبه شود. برای این کار با توجه به محدودیت نوع ۳ نقاط ابتدا و انتهای هر گام از مسیر بررسی میشود تا در داخل متوازی‌الاضلاع باشد. بعد از آن هزینه‌ی حرکت از ابتدا تا نقطه‌ی مورد نظر برای ۳ حالتی که برای گام آخر تعریف میشود محاسبه شده و مینیمم مقدار به عنوان نتیجه در ماتریس D که هزینه‌ی کلی تا هر نقطه را در خود دارد ذخیره میگردد. اگر حرکت مورد نظر در خارج محدوده بود، مقدار بینهایت

میگیرد. در این برنامه وزن دهی نوع a انتخاب شده است که وزن هر گام ۱ و Mϕ برابر Tx میباشد به همین دلیل در انتها D بر Tx تقسیم شده و مقدار هزینه‌ی نهایی در D_Total ذخیره میگردد.

```
%% calculating the overall cost
D=zeros(Tx,Ty);
D1=zeros(Tx,Ty);
D2=zeros(Tx,Ty);
D3=zeros(Tx,Ty);
D(1,1)=Z(1,1);
D(1,2:Tx)=inf;
D(2:Ty,1)=inf;
for ix=2:Tx
    for iy=2:Ty

        [inside]= LegalRange(Tx,Ty,Qmax,ix,iy);
        [inside1]= LegalRange(Tx,Ty,Qmax,ix-2,iy-1);
        [inside2]= LegalRange(Tx,Ty,Qmax,ix-1,iy-1);
        [inside3]= LegalRange(Tx,Ty,Qmax,ix-1,iy-2);
        if inside==1
            if inside1==1
                D1(ix,iy)=Z(ix,iy)+ D(ix-2,iy-1);
            else
                D1(ix,iy)=inf;
            end
            if inside2==1
                D2(ix,iy)=Z(ix,iy)+ D(ix-1,iy-1);
            else
                D2(ix,iy)=inf;
            end
            if inside3==1
                D3(ix,iy)=Z(ix,iy)+ D(ix-1,iy-2);
            else
                D3(ix,iy)=inf;
            end
            D(ix,iy)=Z(ix,iy)+ min([D1(ix,iy) D2(ix,iy) D3(ix,iy)]);
        else
            D(ix,iy)=inf;
        end
    end
end
D=D/Tx;
D_Total(1,ref)=D(Tx,Ty);
```

حال باید مسیر بهینه برای حرکت بین بردارها از روی ماتریس D بدست آید که از کد زیر برای این کار بهره گرفته شده است. در اینجا از انتهای ماتریس شروع کرده و با توجه به محدودیت نوع ۳، مینیمم هزینه برای ۳ نقطه‌ی قبلی و مکان آن

نقطه بدست می‌آید. که در آرایه‌ی path قرار میگیرد. در انتهای این بخش یک شرط قرار گرفته تا اگر مسیر به نقاط (2,3)، (2,2) یا (3,2) رسید، برنامه از داخل حلقه‌ی while خارج شود و نقطه‌ی نهایی (1,1) انتخاب شود.

```
%% finding the efficient path
px=Tx;
py=Ty;
path=zeros(1,Tx);
path(px)=py;
finalpoint=0;
while finalpoint==0 && py>=3 && px>=3
        A=[D(px-2,py-1) D(px-1,py-1) D(px-1,py-2)];
        [minD,p]=min(A(:));
        if p==1
            px=px-2;
            py=py-1;
        elseif p==2
            px=px-1;
            py=py-1;
        elseif p==3
            px=px-1;
            py=py-2;
        end
        path(px)=py;

        if px==3 && py==2
            finalpoint=1;
            path(1)=1;
        elseif px==2 && py==2
            finalpoint=1;
            path(1)=1;
        elseif px==2 && py==3
            finalpoint=1;
            path(1)=1;
        end
end
```

برای رسم مسیر بهینه یک ماتریس با دو ردیف تعریف مشود که در ردیف اول نقاط مربوط به الگوی تست قرار دارد و در ردیف دوم نقاط مربوط به الگوی مرجع و برای رسم نیز تابعی به نام path_plot تعریف شده است.

```
%% preparing for plotting
path1=zeros(2,Tx);
path1(1,:)=1:Tx;
path1(2,:)=path(1:Tx);
Tx1=0;
for i=1:Tx
    if path1(2,i)~=0
        Tx1=Tx1+1;
    end
end
for i=1:Tx1
    if path1(2,i)==0
        path1(:,i)=[];
    end
end

figure(ref)

Path_plot(Tx,Ty,Qmax,path1);
```

تابع **Path_plot:**

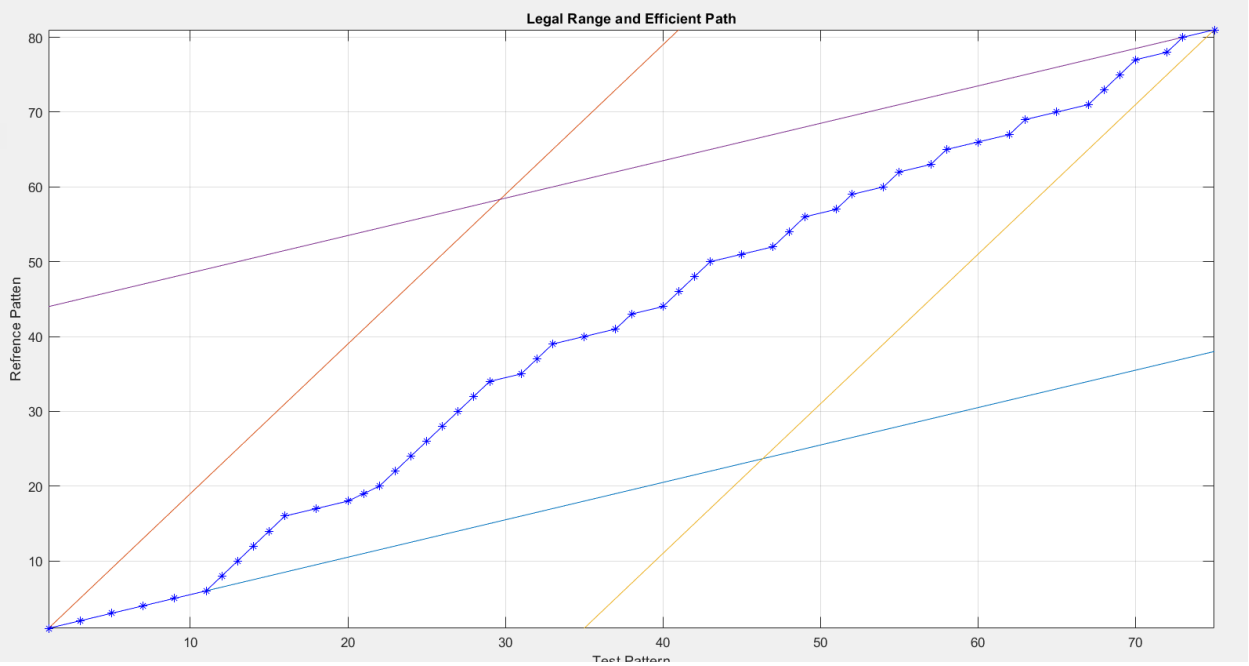
این تابع ماتریس path1 را که ۲ ردیف دارد، Tx، Ty و Qmax را تحویل گرفته و متوازی‌الاضلاع و مسیر را رسم میکند.

```
function []=Path_plot(Tx,Ty,Qmax,path1)

x=1:Tx;
y=((x-1)/Qmax)+1;
plot(x,y)
hold on
y=Qmax*(x-1)+1;
plot(x,y)
y=Qmax*(x-Tx)+Ty;
plot(x,y)
y=(x-Tx)/Qmax+Ty;
plot(x,y)
axis([1 Tx 1 Ty]);
title('Legal Range and Efficient Path')
xlabel('Test Pattern')
ylabel('Refrence Patten')
grid on
plot(path1(1,:),path1(2,:),'b-*');

hold off
```

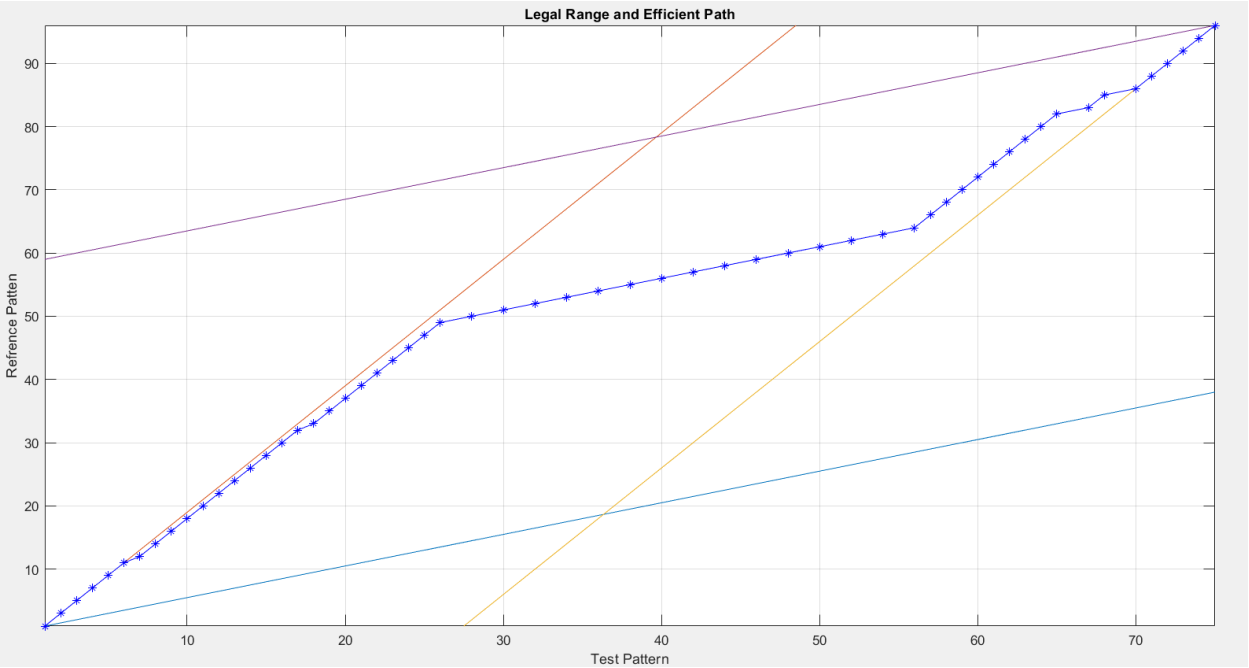
درانتها خروجی مربوط به هر مقایسه مشاهده میشود.

مسیر مربوط به مقایسه‌ی آمریکا و آمریکا نزدیک به قطر اصلی است.

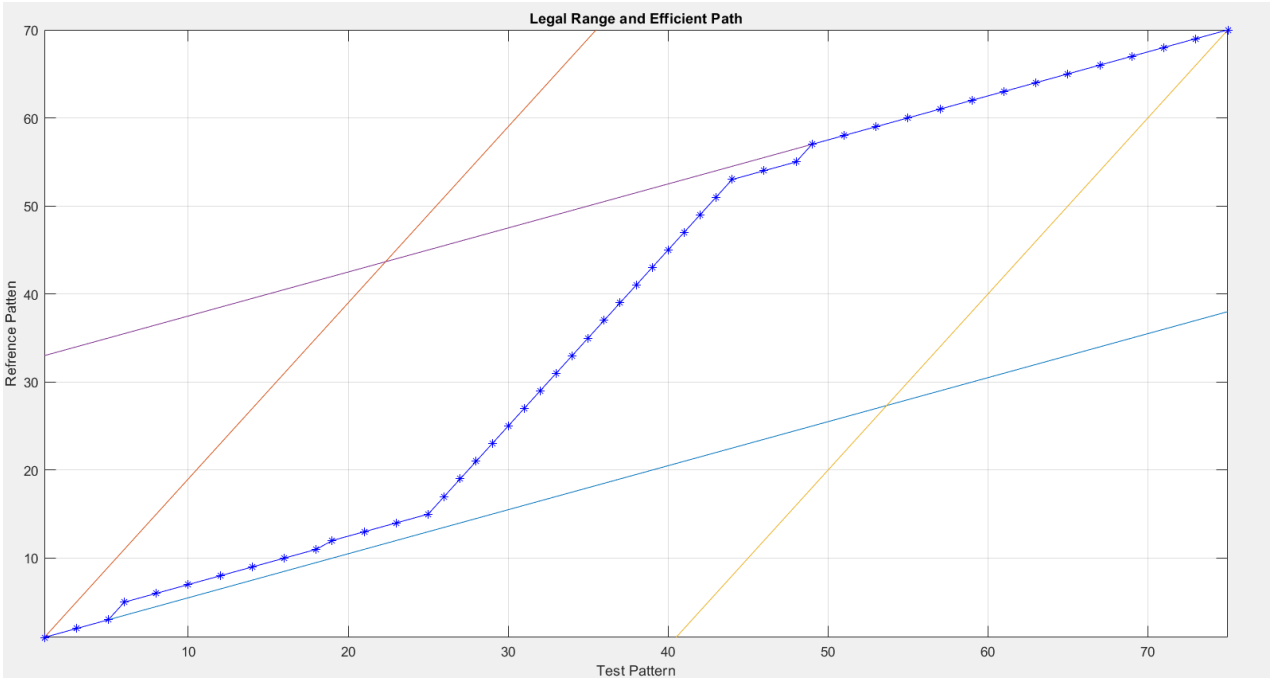
۱- مقایسه مربوط به دو فایل دارای محتوای شبیه به هم: (آمریکا- آمریکا)



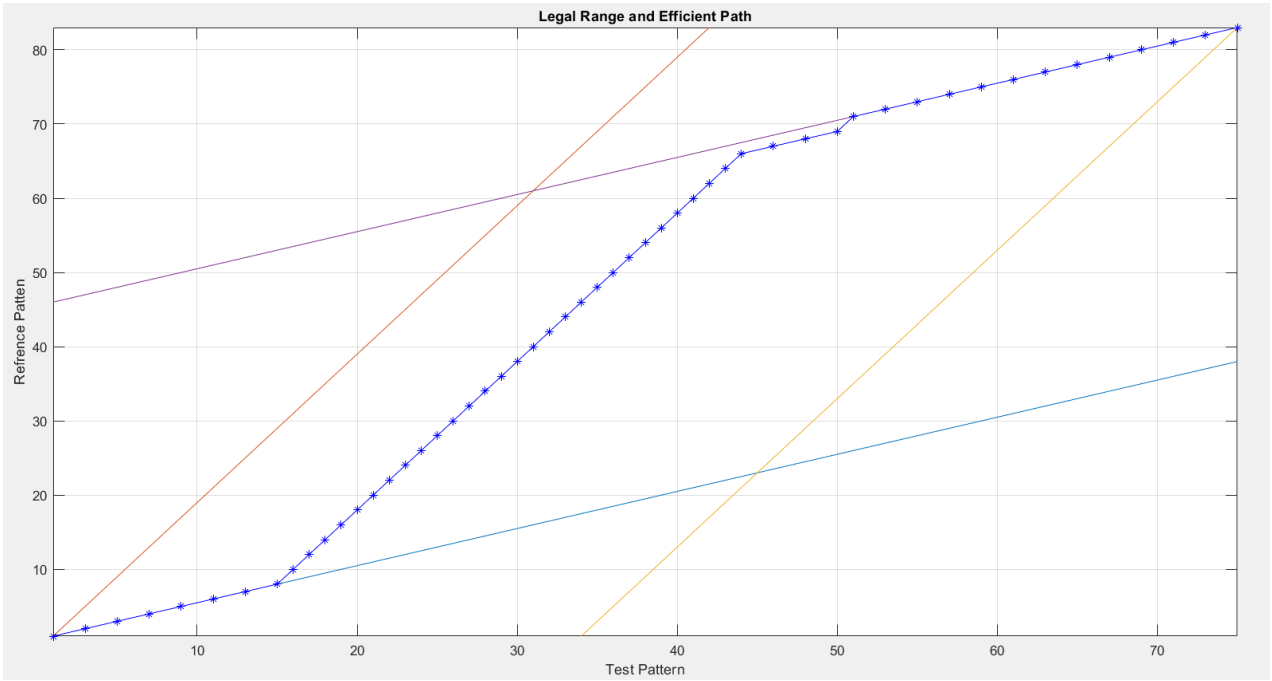
۲- آمریکا- سنگاپور:



۳- آمریکا- روسیه:



۴- آمریکا- نیجریه:



فاصله‌ی کلی هر دو الگو نیز در زیر قابل مشاهده است که فاصله‌ی دو الگوی شبیه به هم از دیگر مقایسه‌ها کمتر است که نشان میدهد شباهت زیادی بین این دو برقرار است و مطابق با انتظارات است.

```
'Total Cost(Test Pattern in America):
   America: 5.129
   Singaore: 8.109
   Russia: 7.496
   Nigeria:6.669'
```