گزارش تمرین کامپیوتری اول – کوثر شیری جعفرزاده – 810101456

بخش اول

تمرین 1-1)

عملیاتی که هر خط انجام می دهد:

خط 1= ابتدا بازه ی زمانی 0 تا 1 را به 100 قسمت تقسیم می کنیم.

خط 2= تابع سینوس را تعریف می کنیم که از 0 تا 2π را شامل می شود.

خط 3= تابع کسینوس را تعریف می کنیم که از 0 تا 2π را شامل می شود.

خط 5= برای نمایش دادن توابع تعریف شده یک figure ایجاد می کنیم.

خط 6= تابع سینوس را با خط چیم آبی در بازه ی صفر تا 1 رسم می کنیم.

خط 7= با استفاده از کلید واژه hold on مطمئن می شویم که تابع بعدی نیز برروی همین شکل رسم شود (تا زمان ذکر کردن hold off توابع برروی هم رسم می شوند)

خط 8= تابع کسینوس را با رنگ قرمز برروی توابع قبلی در بازه ی 0 تا 1 رسم می کنیم.

خط 10= نقطه ای را تحت نام x0 به صورت یک بردار ستونی تعریف می کنیم.

خط 11= نقطه ای را تحت عنوان y0 به صورت یک بردار ستونی تعریف می کنیم.

```
x0 =

    0.5000
    0.2500


y0 =

    0.2000
   -0.8000
```

خط 12= یک بردار ستونی از استرینگ ها ساخته و آن را s می نامیم، چون نیاز به نوشتن pi داریم از \ برای اینکار استفاده می کنیم.

خط 13= حالا مختصات را به تابع text داده تا بروری شکل و در نقاط تعیین شده عبارت مد نظر را چاپ کند.

نقاط مدنظر و عبارت چاپ شده به شرح زیر است:

$$P1(0.5,0.2) \gg\gg\gg \sin(2\pi t)$$

$$P2(0.25,-0.8) \gg\gg\gg \cos(2\pi t)$$

خط 15= برای شکل رسم شده عنوانی را قرار می دهیم.

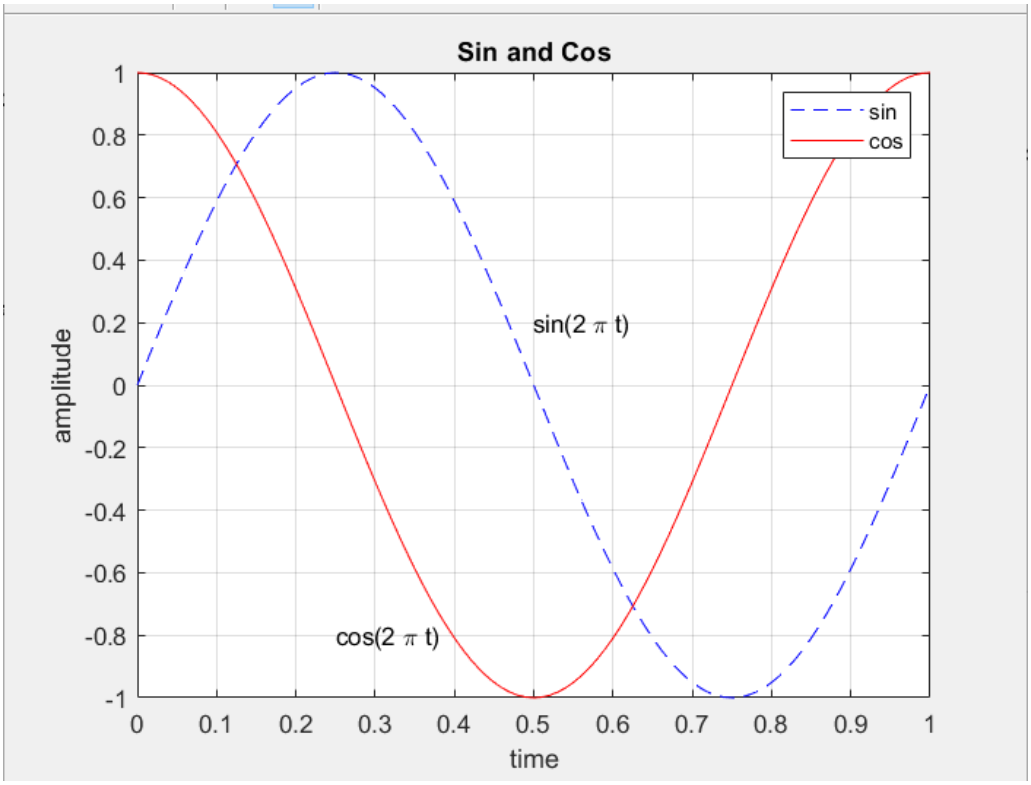
خط 16= برای اینکه مشخص باشد هرنوع از خط نمایانگر کدام تابع می باشد باید از دستور legend استفاده کنیم که در گوشه سمت راست انواع خط و تابع مربوط به آن را ذکر می کند.

خط 17= محور افقی را به نام زمان نامگذاری می کنیم.

خط 18= محور عمودی را به نام مقدار نامگذاری می کنیم چرا که مقدار توابع را در لحظات مختلف نشان می دهد.

خط 19= باتوجه به نیاز برای دیدن مقادیر صفحه را شبکه بندی می کنیم.

شکل نهایی کد:



حذف hold on در خط 7:

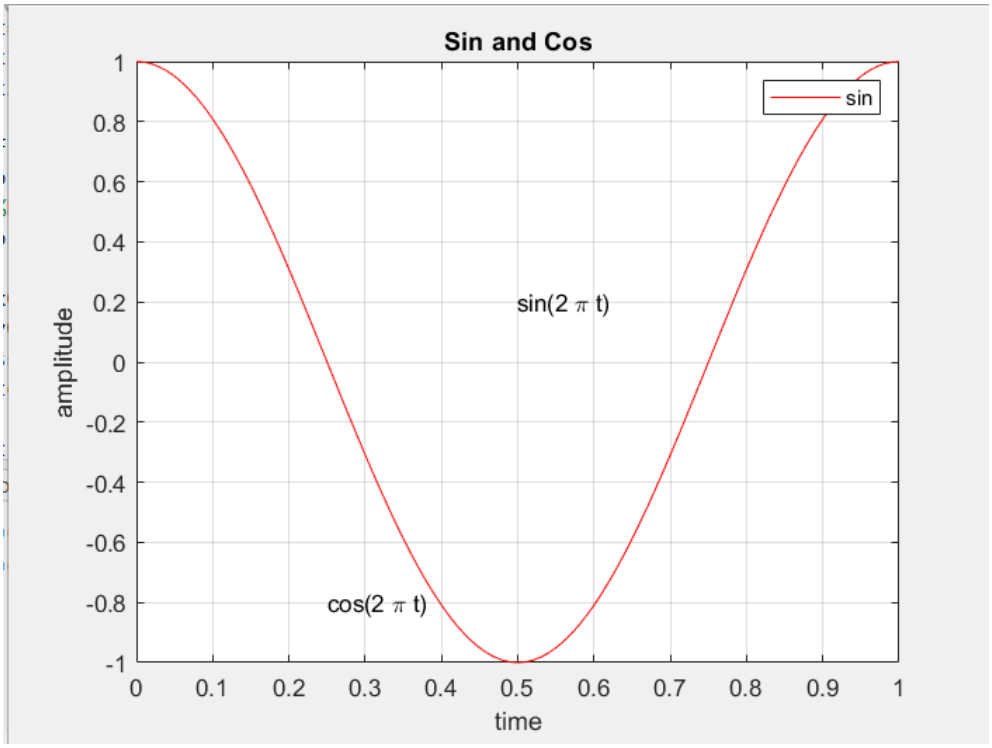
با حذف این کلیدواژه تنها تابع کسینوس برروی صفحه رسم می شود و تابع سینوس از صفحه پاک شده است.

این موضوع باعث رخ دادن ارور نیز می شود چراکه چون تنها یک خط برای رسم تابع موجود است باید ورودی های دستور legend نیز تغییر کند چراکه ما به آن اعلام کرده ایم که 2 تابع برای رسم برروی صفحه داریم.

کد مربوط به این بخش:

```
t=0:0.01:1;
z1=sin(2*pi*t);
z2=cos(2*pi*t);

figure;
plot(t,z1,'--b')

plot(t,z2,'r')

x0=[0.5;0.25];
y0=[0.2;-0.8];
s=['sin(2 \pi t)'; 'cos(2 \pi t)'];
text(x0,y0,s);

title('Sin and Cos');
legend('sin','cos');
xlabel('time')
ylabel('amplitude')
grid on
```

خروجی با حذف hold on:



تمرین 1-2)

با توجه به اینکه در شکل داخل دستور پروژه قسمت افقی به 2 قسمت تقسیم شده است پس باید پارامتر دوم دستور subplot را برابر 2 قرار دهیم :

Subplot (division in vertical, division in horizontal, the index of the current plot)

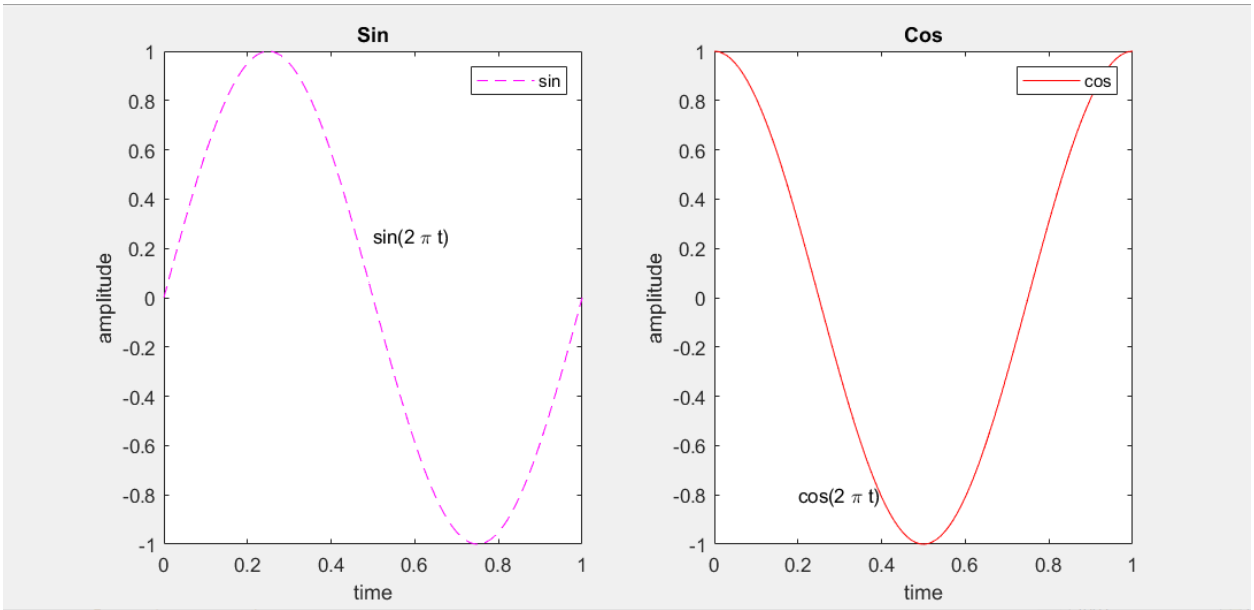
همانطور که مشخص است مواردی از قبیل text,tilte,legend,label باید برای هریک از subplot ها به صورت جداگانه انجام شود.

اسکریپت:

p1_1.m | p1_2.m

```
t=0:0.01:1;
z1=sin(2*pi*t);
z2=cos(2*pi*t);

figure;
subplot(1,2,1)
plot(t,z1,'--m')
s0='sin(2 \pi t)';
text(0.5,0.25,s0);
title('Sin');
legend('sin');
xlabel('time')
ylabel('amplitude')

subplot(1,2,2)
plot(t,z2,'r')
s1= 'cos(2 \pi t)';
text(0.2,-0.8,s1);
title('Cos');
legend('cos');
xlabel('time')
ylabel('amplitude')

```

شکل نهایی:



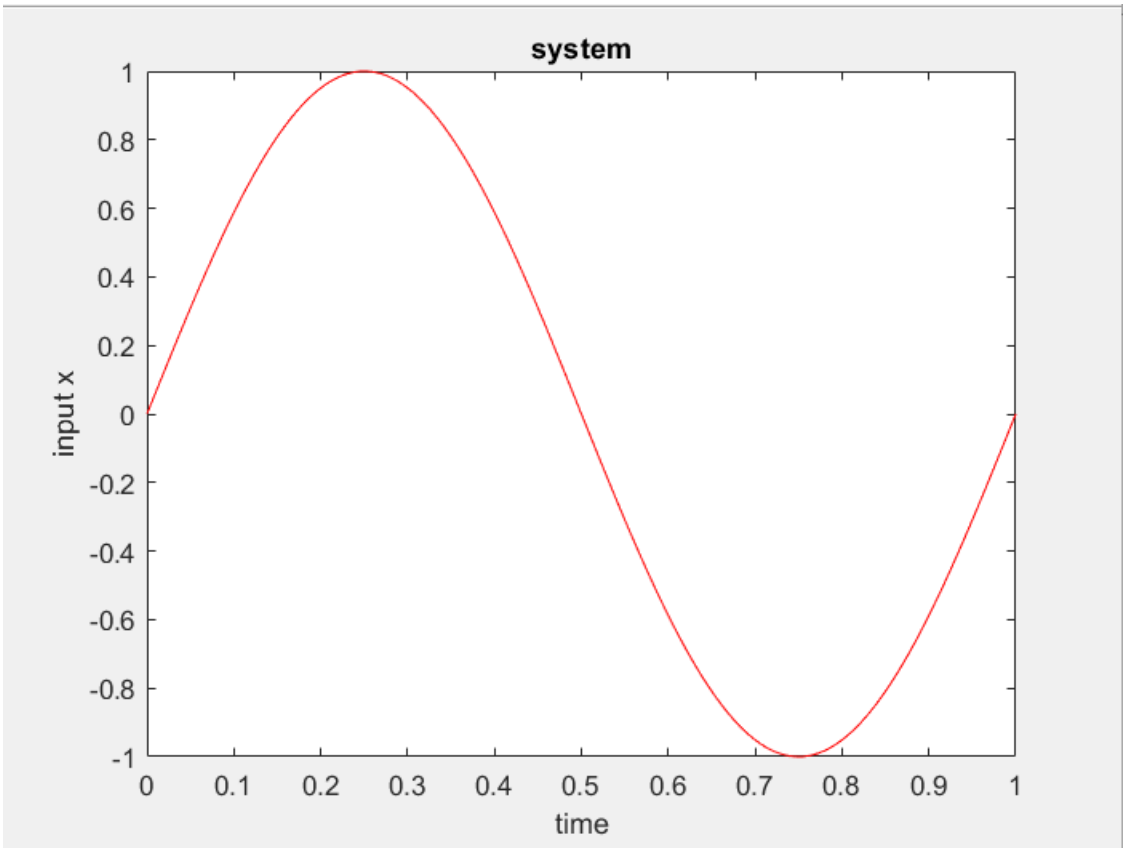
بخش دوم

تمرین 1-2)

کد این بخش:

```
load("p2.mat")
figure
plot(t,x,'r')
title('system');
xlabel('time')
ylabel('input x')
```

شکل نهایی:

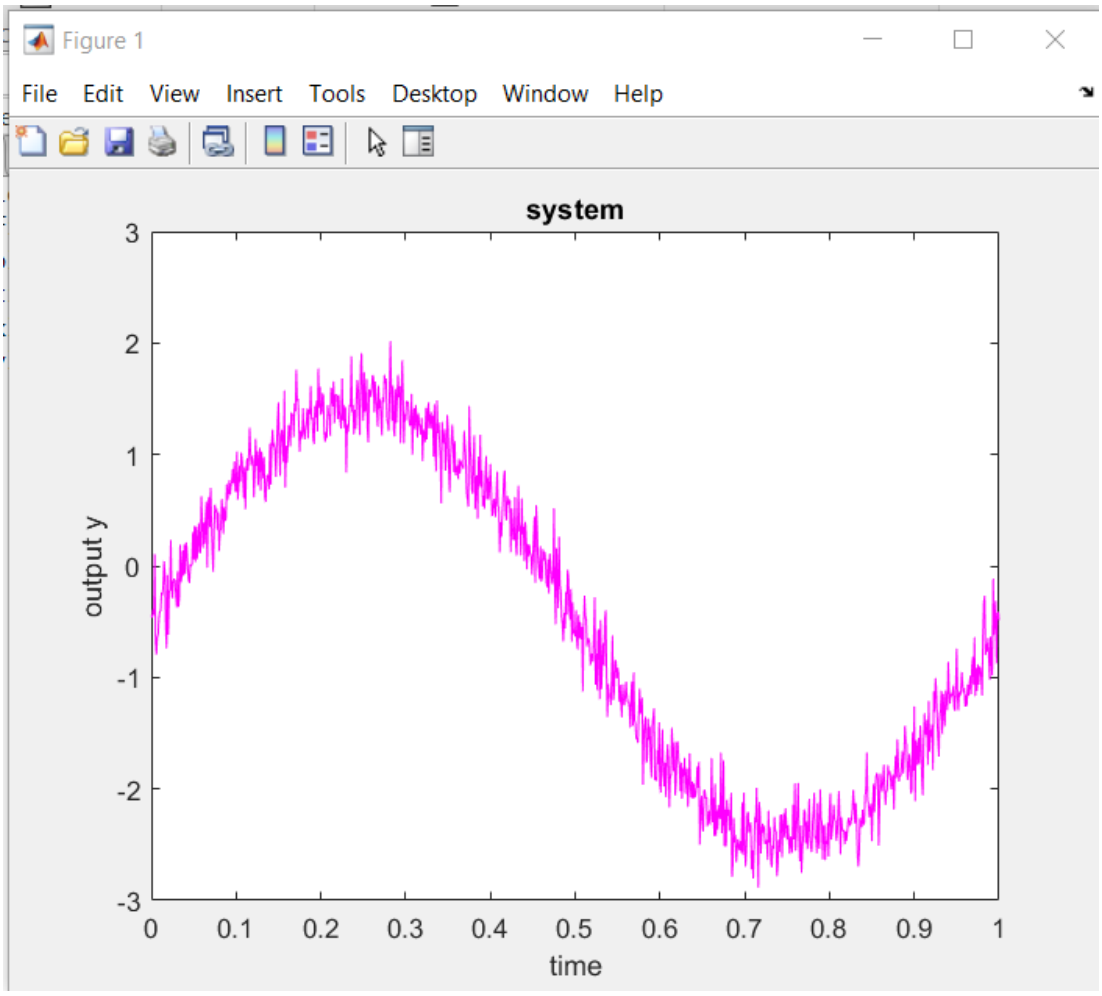


تمرین 2-2)

کد این بخش:

```
load("p2.mat")
figure
plot(t,y,'m')
title('system');
xlabel('time')
ylabel('output y')
```

شکل نهایی:

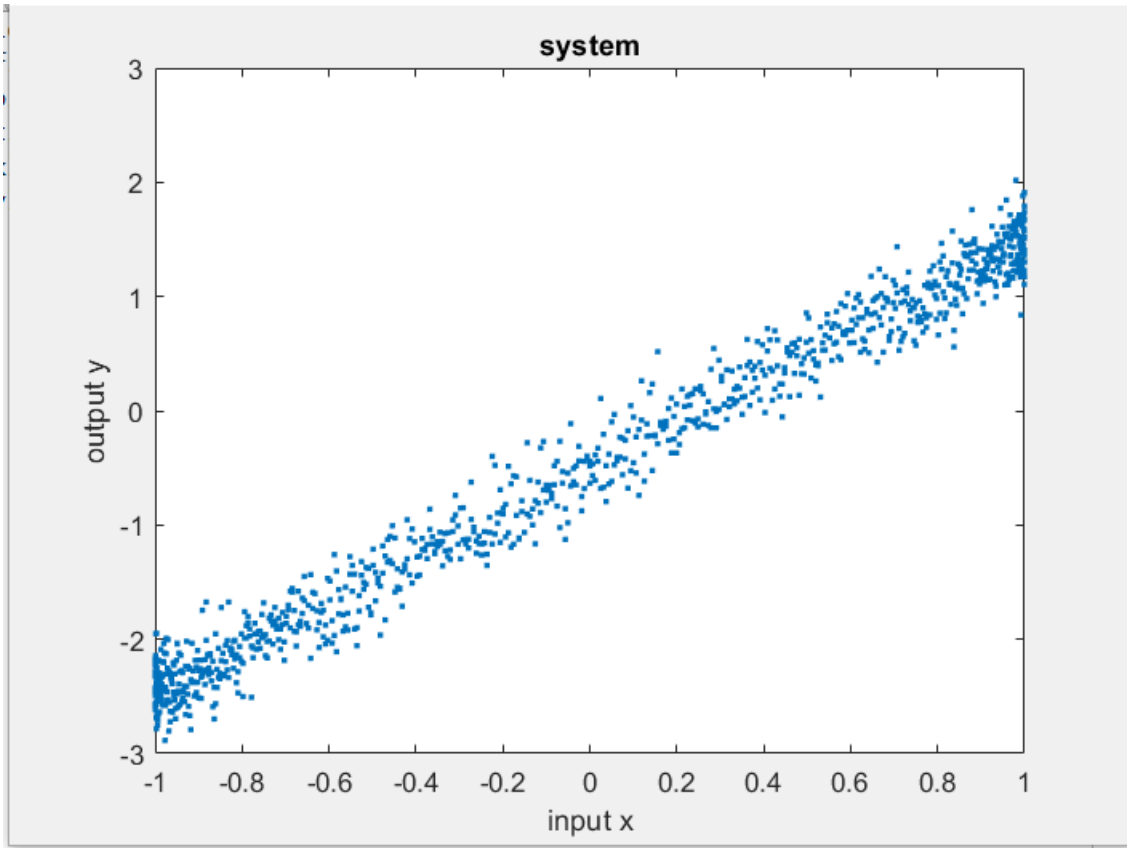


تمرین 2-3)

کد این بخش:

```
load("p2.mat")
figure
plot(x,y,'.')
title('system');
xlabel('input x')
ylabel('output y')
```

شکل نهایی:



همانطور که مشخص است شیب خط برابر α می باشد.

$$y(t) = \alpha x(t) \;\gg\gg\; \alpha = \frac{y(t)}{x(t)}$$

عرض از مبدا این خط برابر است با β

$$y(t) = \alpha * x(t) + \beta \xrightarrow{x(t)=0} y(t) = \beta$$

تمرین 2-4)

همانطور که در راهنمایی گفته شده است نیاز داریم که در ابتدا تابع هزینه را پیاده سازی کنیم با کمینه کردن مقدار این تابع مقادیر مربوط به آلفا و بتا به دست می آیند (با توجه به اینکه انتظار داریم با پیدا کردن مقادیر صحیح الفا و بتا حاصل عبارت برابر صفر شود پس هر زمان مقدار تابع هزینه به کمترین مقدار خود برسد یعنی الفا و بتا را به درستی انتخاب کرده ایم)

تابع هزینه مدنظر ما به صورت زیر می باشد:

$$cost = \sum_{t} (y(t) - \alpha x(t) - \beta)^2$$

حالا در مرحله ی بعدی باید تابعی بنویسیم که مقادیر فعلی الفا و بتا را دریافت کرده و تغییرات لازم را برروی آن ها اعمال کند تا جایی که هزینه کمینه شده و پارامتر ها به دست بیایند

برای پیدا کردن این مقادیر از تابع fminunc استفاده میکنیم که برای

Find minimum of unconstrained multivariable function

استفاده می شود. ورودی های این فانکشن شامل مقدار اولیه پارامتریه هایی می شود که باید هزینه را مینیمم کنند.

ما نیاز داریم که پارامترهارا نیز هربار به این تابع پاس دهیم برای اینکار با استفاده از @ یک فانکشن جدید تعریف می کنیم به نام params که در آن مقدار اولیه ابتدا به فانکشن داده می شود و هزینه حساب شده باز گرداننده می شود. سپس با توجه به هزینه محاسبه شده تغییراتی توسط fminunc انجام می شود و پارامترهای جدیدی به تابع پاس داده می شود .

لازم به ذکر است که هر زمان مقدار هزینه کمینه شد یا دیگر تغییرات تاثیر چندانی در خروجی نداشت مقادیر پارامتر ها بازگردانده می شود.

```
function [alpha , beta]=p2_4(x,y)
init_Params = [0, 0];
[optimal] = fminunc(@(params) sum((y - (params(1) * x + params(2))).^2), init_Params);
alpha = optimal(1);
beta = optimal(2);
end
```

مقادیر محاسبه شده برابر است با :

```
<stopping criteria details>
alpha: 1.9736
beta: -0.4983
```

نکته : استفاده از این تابع برای بردار p2.mat در فایل p2_4_usage موجود است

برای مطمئن شدن از درستی تابع از مقادیر دستی نیز استفاده می کنیم. این موضوع در فایل test_func آورده شده است، که درآن جا نیز مقادیر با دقت خوبی درست به دست می آیند (همانند توصیه انجام شده برای چک کردم درستی تابع نویز به تابع اضافه شده است)

اگر میخواستیم بدون استفاده از توابع ازپیش آماده این مسئله را حل کنیم باید در ابتدا به صورت جداگانه نسبت به دو پارامتر الفا و بتا مشتق گیری کنیم و برابر صفر قرار دهیم:

$$\frac{\partial(\sum_t(y(t)-\alpha x(t)-\beta)^2)}{\partial\alpha}=\sum_t(y(t)-\alpha x(t)-\beta)*\big(x(t)\big)=0$$

$$\sum_t\big(y(t)x(t)-\alpha x(t)^2-\beta x(t)\big)=\sum_t y(t)x(t)-\alpha\sum_t x(t)^2-\beta\sum_t x(t)=0$$

$$\frac{\partial(\sum_t(y(t)-\alpha x(t)-\beta)^2)}{\partial\beta}=\sum_t(y(t)-\alpha x(t)-\beta)*(1)=0$$

$$\sum_t y(t)-\alpha\sum_t x(t)-\beta\sum_t 1=0$$

با محاسبه دو بخش قرمز رنگ مقدار الفا و بتا بدون استفاده از توابع از پیش تعریف بدست می آید

تمرین 2-5)

اطلاعات نمایش داده شده به شکل زیر است:

Fit Name: untitled fit 1

Polynomial Curve Fit (poly1)

$f(x) = p1*x + p2$

Coefficients and 95% Confidence Bounds

	Value	Lower	Upper
p1	1.9736	1.9560	1.9911
p2	-0.4983	-0.5107	-0.4859

Goodness of Fit

	Value
SSE	39.8638
R-square	0.9799
DFE	999.0000
Adj R-sq	0.9799
RMSE	0.1998

همانطور که مشاهده می شود داده بدست آمده با مقادیر محاسبه شده توسط تابع مطابقت دارد. (باتوجه به اینکه هردفعه سعی میکنید بهترین بهینه سازی را انجام دهد خطایی در حد 0.1 تا 0.01 به چشم می خورد.

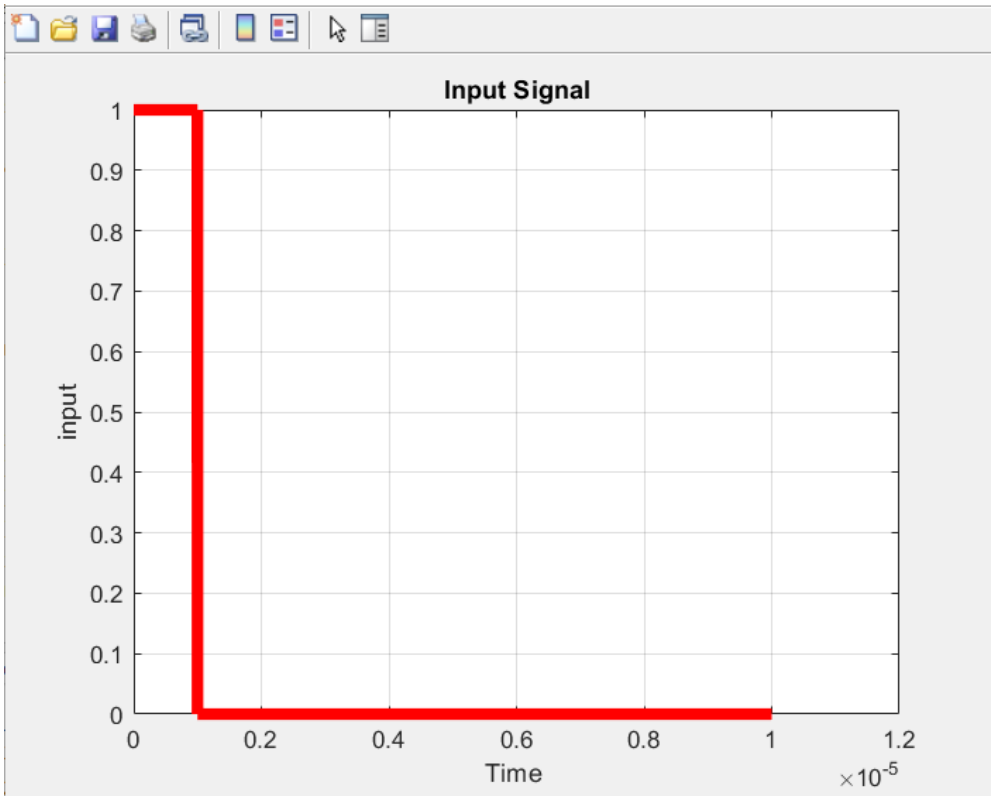
بخش سوم

تمرین 3-1)

کد تابع:

```
ts = 1e-9;
fs = 1 / ts;
T = 1e-5;
tau = 1e-6;

t = 0:ts:T;
tlen = length(t);

x = zeros(1, tlen);
one_value = 1:round(tau/ts);
x(one_value) = 1;

figure;
plot(t, x,'r',LineWidth=5);
xlabel('Time ');
ylabel('input');
title('Input Signal');
grid on;
```

شکل نهایی به صورت زیر است:



تفسیر عبارت دیده شده :

توقع داریم که دوره ی تناوب سیگنال رسم شده برابر $1 * 10^{-5}$ باشد یعنی در مدت 10 میلی ثانیه و در عین حال مدت زمانی که توقع داریم که سیگنال مقدار 1 داشته باشد از زمان تا 0 تا 10^{-6} است که یعنی از 0 تا 1 میلی ثانیه باید 1 باشد (قابل توجه است که با توجه یه اینکه سهم های زمانی ما بین 0 تا 10 میلی ثانیه 10^{-9} گام جلو می رود پس باید بدانیم چند سهم زمانی را 1 کنیم که تعداد آن برابر است با :

$$\frac{\tau}{ts} = \frac{10^{-6}}{10^{-9}} = 10^{3}$$

تمرین 3-2)

باید طبق فرمول داده شده مقدار t_d را حساب کرده و از زمان به دست آمده به اندازه مدت زمانی که انتظار داریم سیگنال را برابر مقدار α قرار دهیم. که طبق محاسبات بالا قرار است به مدت 1 میلی ثانیه پس از t_d مقدار سیگنال برابر 0.5 باشد(باتوجه به اینکه در این بخش نویز را در نظر نمیگیریم پس در سایر نقاط مقدار تابع 0 است).

$$R = 450, C = 3 * 10^{8}$$

$$t_d = \frac{2 * R}{C} = 2 * \frac{450}{3 * 10^{8}} = 3 * 10^{-6}$$

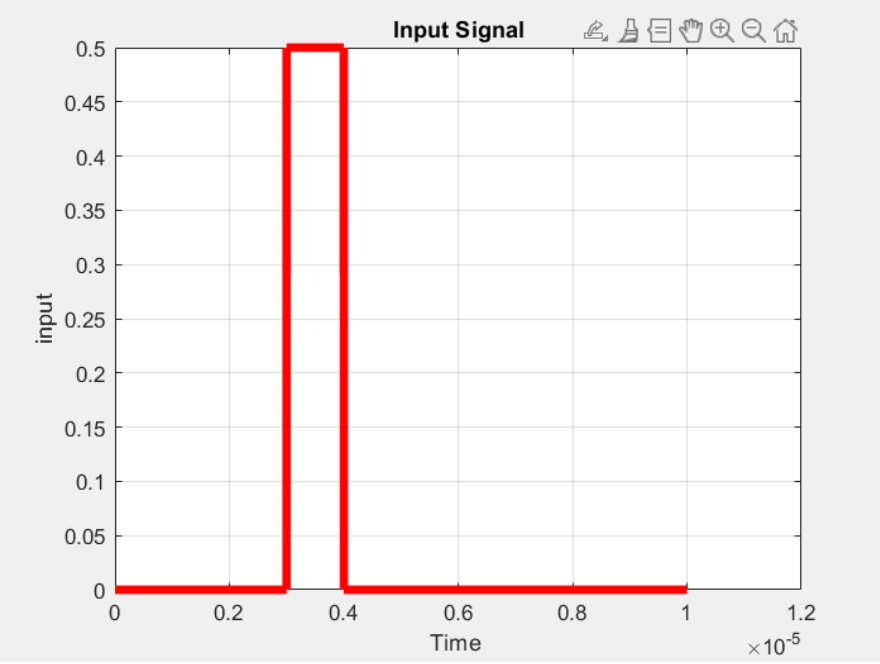
$$t_d + \tau = 3 * 10^{-6} + 1 * 10^{-6} = 10^{-6} * 4$$

همانطور که مشخص است سیگنال برگشتی در فاصله 3 تا 4 میکرو ثانیه رخ می دهد و مقدار این سیگنال برابر 0.5 است.

کد این بخش:

```
ts = 1e-9;
fs = 1 / ts;
T = 1e-5;
tau = 1e-6;
alpha=0.5;
C=3e8;

t = 0:ts:T;
tlen = length(t);

R=450;
t_d=2*R/C;
disp(t_d)
y=zeros(1,tlen);
start_td=round(t_d/ts)+1;
end_tau=round((t_d+tau)/ts)+1;
y(start_td:end_tau)=alpha*ones(1,end_tau-start_td+1);

figure;
plot(t, y,'r',LineWidth=4);
xlabel('Time ');
ylabel('input');
title('Input Signal');
grid on;
```

شکل نهایی به صورت زیر می باشد:



همانطور که مشخص است برای اینکه پایان و ابتدای بازه را مشخص کنیم باید از سهم های زمانی کمک بگیریم تا متوجه شویم که چند بخش زمانی باید برابر 0.5 شود(دلیل استفاده از تابع roundهندل کردن حالاتی است که ممکن است صحیح نباشند):

$$\frac{t_d}{ts} = \frac{3 * 10^{-6}}{10^{-9}} = 3 * 10^3$$

$$\frac{(t_d + \tau)}{ts} = \frac{10^{-6} * 4}{10^{-9}} = 4 * 10^3$$

تعداد سهم های زمانی که باید 0.5 شود همانند قبل برابر 10^3 می باشد

تمرین 3-3)

برای اینکه به صورت برعکس به مسئله نگاه کنیم نیاز است تا مقدار تاخیر زمانی را به دست آورده و فاصله را محاسبه کنیم:

$$R = \frac{C * t_d}{2}$$

برای یافتن زمان برگشت سیگنال اینگونه عمل می کنیم:

این کار به این نحوه است که ما انتظار داریم شکل اولیه را در بهترین حالت به عنوان خروجی پس از تاخیر مشخصی دریافت کنیم پس کافیست از ابتدای سیگنال خروجی شروع به مچ کردن با سیگنال ورودی کنیم و به شکل sliding window شروع به جلو بردن template برروی خروجی کنیم.هر زمان که دو شکل برهم منطبق شوند یعنی t_d را یافتیم.

اما با توجه به اینکه استفاده از تصویر خطاهایی را دارد از ضرب داخلی استفاده می کنیم به این طریق که تمامی نقاط template در زمان فعلی را در نقاط تابع ضرب می کنیم و باهم جمع میبندیم بدیهی است که بیشترین حالت مجموع در زمانی اتفاق می افتد که دو سیگنال در t_d باهم تلاقی کنند.(در صورتی که احتمال میدادیم که سیگنال خروجی بازنگشته است لازم است برای رابطه ی correlation یکthreshold تعیین کنیم تا ازحدس های تصادفی جلوگیری کنیم)

برای پیاده سازی به این شکل عمل می کنیم که از لحظه 0 تا اخرین لحظه ای که ممکن است t_d رخ دهد جلو می رویم و در هر لحظه به اندازه ای در سیگنال خروجی جلو رفته و آن بخش را در سیگنال ورودی ضرب می کنیم. بدیهی است با یافتن t_d این مقدار بیشینه شده و ما آن لحظه ی خاص را پیدا کرده ایم

پس ماکسیمم تابع و لحظه ی رخ دادن آن را برگردانده و ان را در رابطه ذکر شده گذاشته و میزان فاصله را پیدا می کنیم.

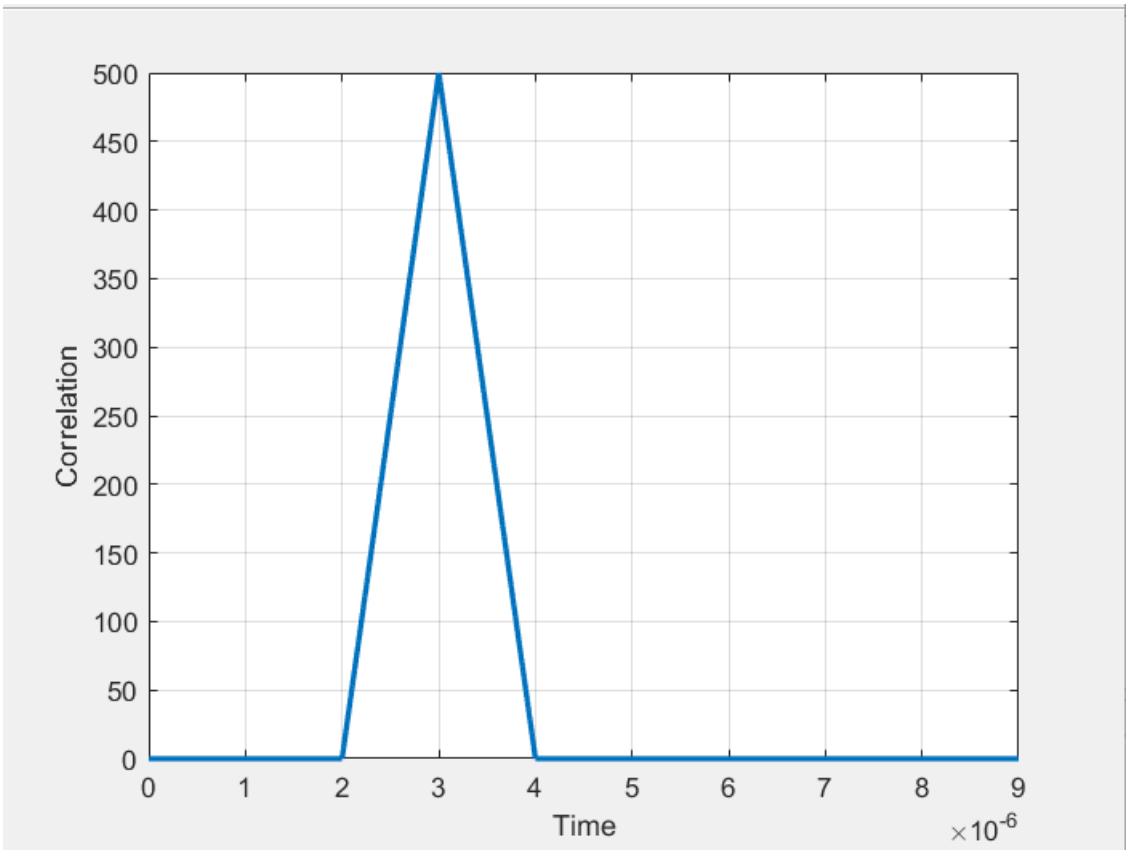
نکته : ما برای پیاده سازی ضرب داخلی از تابع innerproduct استفاده کرده ایم.

کد این بخش:

```
ts = 1e-9;
fs = 1 / ts;
T = 1e-5;
tau = 1e-6;
alpha=0.5;
C=3e8;
t = 0:ts:T;
tlen = length(t);
R_orig=450;
t_d=2*R_orig/C;
x = zeros(1, tlen);
one_value = 1:round(tau/ts);
x(one_value) = 1;
y=zeros(1,tlen);
start_td=round(t_d/ts)+1;
end_tau=round((t_d+tau)/ts)+1;
y(start_td:end_tau)=alpha*ones(1,end_tau-start_td+1);

ro=zeros(1,tlen-round(tau/ts));
for i=1:tlen-round(tau/ts)
    ro(i)=innerproduct(y(i:i+round(tau/ts)-1),x);
end

[val,ind]=max(ro);
td=(ind-1)*ts;
R_calculated=C*td/2;
fprintf("calculated distance is :%.4f \n",R_calculated);
figure
plot(t(1:tlen-round(tau/ts)),ro,'LineWidth',2)
xlabel('Time')
ylabel('Correlation')
grid on
```

خروجی این بخش:

```
>> p3_3
calculated distance is :450.0000
```



همانطور که مشخص است در لحظه 3 میلی ثانیه که از قبل مورد انتظار ما نیز بود بازگشت سیگنال رخ داده است و به مقدار بیشینه خود رسیده است.

مقدار شعاع حساب شده با مقدار فرض شده مطابقت دارد.

تمرین 3-4)

همانند بخش های قبل در ابتدا سیگنال ورودی را تولید می کنیم (بدین ترتیب که به اندازه دوره ی تناوب سیگنال صفر در نظر می گیریم سپس سهم های زمانی که می خواهیم 1 شود را به دست می آوریم و آن بخش سیگنال را برابر 1 قرار می دهیم)

در بخش ایجاد سیگنال خروجی چون می دانیم میزان تاخیر سیگنال چقدر است سهم های زمانی مربوط به آن را یافته و به اندازه ی خواسته شده برابر 0.5 قرار می دهیم.

در این بخش برخلاف بخش های قبلی می خواهیم به سیگنال خروجی نویز اضافه کنیم برای این کار بازه ی 0 تا 4 را درنظر می گیریم و به اندازه 0.02 در این بازه جلو می رویم بدین شکل که هربار به نویز فعلی 0.02 اضافه کنیم.

نکته ی مهم این است که برای رسم نمودار خطا برحسب نویز، باید مقادیر مختلف نویز و میزان خطاهای آن را نگه داریم پس دو ارایه با سایز یکسان تعریف می کنیم تا این اطلاعات را ذخیره کند (در کد قرار داده شده چون حلقه برحسب نویز جلو می رود نیاز بود تا یک شمارنده برای برای ذخیره اطلاعات نیز تعریف کنیم)

در قدم بعدی نکته ی ذکر شده در پروژه را اجرا می کنیم باتوجه به اینکه نویز ماهیت تصادفی دارد به این صورت عمل می کنیم که در یک حلقه 100 تایی با قدرت نویز ثابت i نویز تولید کرده وهر بار آن را به سیگنال خروجی اضافه می کنیم.(نکته این است که حتما باید دقت کنیم سیگنال خروجی هر بار اپدیت شود در غیر اینصورت خروجی غلط خواهد بود)

در این بخش همانند بخش قبل خروجی را داریم و با استفاده از روش correlation or template matching زمان t_d را پیدا می کنیم .

بعد از یافتن مقدار t_d از رابطه ی داده شده استفاده کرده و فاصله تخمین زده شده را حساب می کنیم

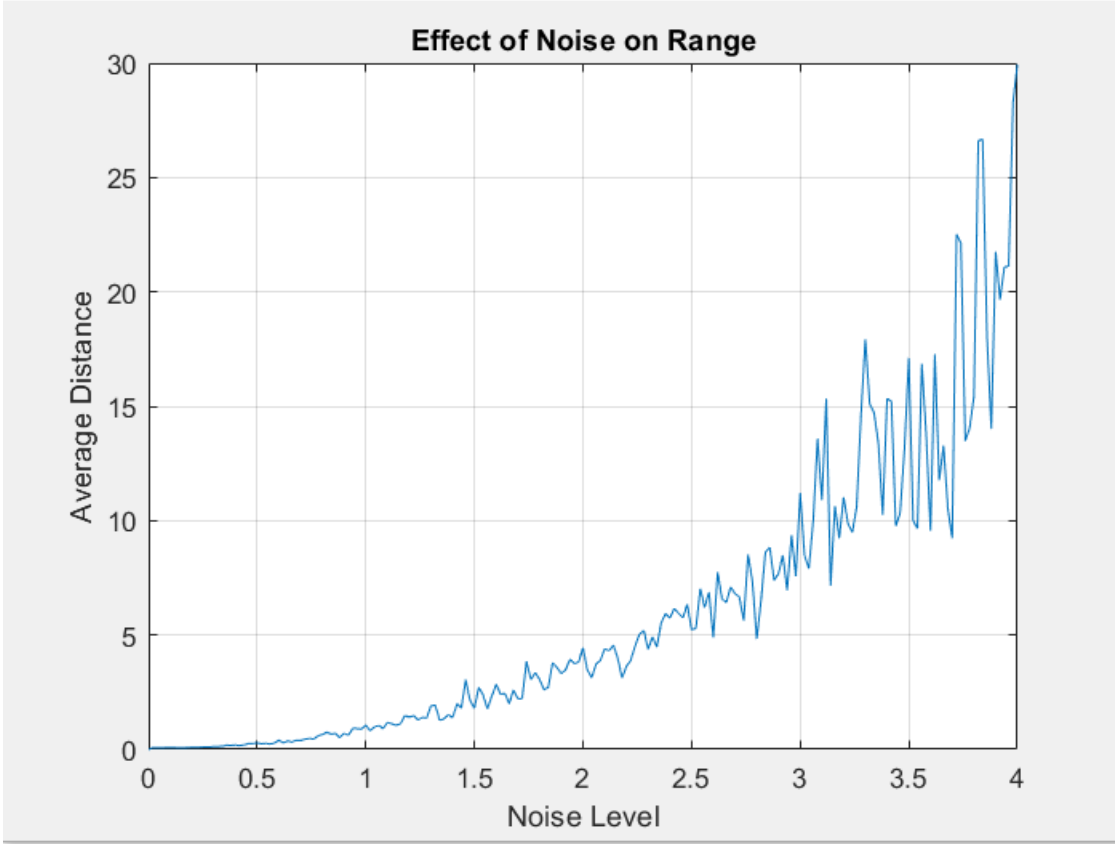
حالا باید اختلاف این فاصله با فاصله ی اصلی را حساب کنیم. بعد از محاسبه این عدد را با متغییر difference جمع می زنیم (همانطور که گفتیم باتوجه به ماهیت تصادفی نویز در 100 میزان خطای هر نویز را حساب کرده و میانگین می گیریم تا بتوانیم برای یک قدرت نویز خطایی را ارائه دهیم.)

در نهایت مقدار قدرت فعلی نویز و مقدار خطای آن را ذخیره کرده و با توجه به شرط ذکر شده اگر خطای آن کمتر از 10 متر بود آن را تخمینی درست و در غیر اینصورت اشتباه می نامیم.(این موضوع برروی خروجی چاپ می شود)

در نهایت تمامی مقادیر حساب شده را برروی صفحه رسم می کنیم. نکته این است که باتوجه به ماهیت تصادفی بودن نویز با افزایش شرط حلقه عدد نزدیک تری به خطای اصلی را باری قدرت مشخصی از نویز می یابیم.

شکل نهایی نیز در هر بار ممکن است متفاوت رسم شود ولی آنچه مهم است با افزایش قدرت نویز مقدار خطا افزایش می یابد.

شکل نهایی :



همانطور که مشخص است در حدود قدرت نویز 3 مقدار خطای ما از 10 بیشتر شده و دیگر فاصله ی محاسبه شده معتبر نمی باشد.

بخش چهارم

تمرین 4-1)

بیت خوانده شده:

سعدیا دی رفت و فردا همچنان موجود نیست

در میان این و آن فرصت شمار امروز را

کد مربوط به این بخش:

p4_1.m

```
1  file_path = 'record_audio.wav';
2  [x, Freq] = audioread(file_path);
3  fprintf('the frequency is equal to : %d\n', Freq);
```

فرکانس نمونه برداری شده :

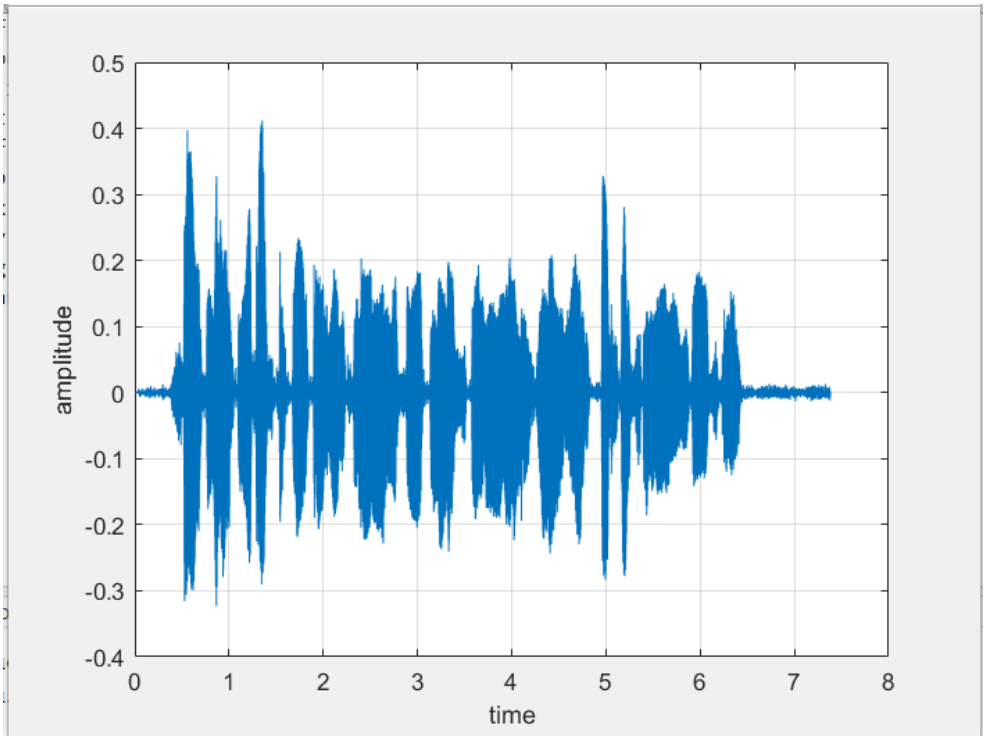
```
>> p4_1
the frequency is equal to : 44100
```

تمرین 4-2)

کد مربوط به این بخش:

```
file_path = 'record_audio.wav';
output_file_path='x.wav';
[x, Freq] = audioread(file_path);
time=(0:length(x)-1)/Freq;
figure;
plot(time,x);
xlabel("time")
ylabel("amplitude")
grid on
audiowrite(output_file_path, x,Freq);

```

شکل رسم شده :



تمرین 4-3)

باتوجه به اینکه برای پخش کردن صدا نیاز به داشتن فرکانس صدای مربوطه نیز هست پس ورودی های تابع را سرعت مورد نظر برای پخش و همچنین اسم فایل صوتی قرار می دهیم.

در مرحله ی بعدی با استفاده از دستور audioread فرکانس و بردارx را ذخیره می کنیم.

حال باید چک کنیم که دستور مورد انتظار آیا برابر 0.5 یا 2 هست در غیر اینصورت خطایی مبنی بر درست نبودن سرعت چاپ می کنیم.

برای سرعت 2:

یک آرایه ی دیگر در نظر می گیریم و شروع به نمونه برداری از صدای اولیه می کنیم با این تفاوت که به ازای هر 2 نمونه یکی را دور میریزیم پس انتظار داریم که طول آرایه نهایی نصف آرایه صدای اولیه باشد.(با فرض زوج بودن صدای اولیه 2k صدای نهایی نیز k خواهد بود و اگر2k+1 باشد به فرم k+1 خواهد بود.)

حال ارایه جدید را با فرکانس اولیه پخش می کنیم همانطور که انتظار داشتیم سرعت آن بیشتر شده است.

برای سرعت 0.5:

با توجه به اینکه نیاز داریم در میان هر دو نمونه کنونی، میانگین آن ها را اضافه کنیم پس به برداری به طول دوبرابر بردار صدای فعلی نیاز داریم.

حال کافیست در بردار خروجی 2 تا 2تا جلو برویم و آن را مساوی مقدار بردار x قرار دهیم(دلیل این کار این است که میخواهیم به فضای خالی بین این دو سیگنال میانگین دو سیگنال مجاور را اضافه کنیم)

وضعیت :

$$output[1] = x[1]\,, output[2] = 0(no\ data)\,, output[3] = x[2]$$

حالا در قدم بعدی نیاز است تا خانه های زوج (که آن هارا خالی گذاشته ایم پر کنیم) برای این کار به بردار صدای اولیه مراجعه می کنیم و بین هر دو خانه مجاور میانگین گرفته و آن را برای خانه های زوج قرار می دهیم:

$$for\ i < len(input\ audio):$$

$$output[2 * i] = \frac{x[i] + x[i+1]}{2}$$

بنابراین خروجی مورد انتظار ما به شکل زیر است:

$$output[1] = x[1]\,, output[2] = \frac{x[1] + x[2]}{2}\,, output[3] = x[2]$$

بازم به ذکر است این بخش نیز با فرکانس اولیه پخش می شود.

کد این بخش به شرح زیر است:

```
function  p4_3(audio_name,speed)
[x, Freq] = audioread(audio_name);
if speed==2
    output_audio=[];
    output_audio=x(1:2:end);
    sound(output_audio,Freq);
elseif speed==0.5
    audio_out=zeros(2*len-1,1);
    audio_out(1:2:end)=x;
    for i=1:len-1
        audio_out(2*i)=mean(x(i:i+1));
    end
    sound(audio_out,Freq);
else
    fprintf('The speed is not valid');
end
end
```

استفاده از این بخش در فایل p4_3_usage قرار دارد.

تمرین 4-4)

در اینجا باتوجه به اینکه اگر سرعت برابر 0.7 باشد برای اضافه کردن مقادیر به مشکل می خوریم یا وقتی که سرعت برابر 1.7 می باشد برای حذف مقادیر نیز همان مشکل را داریم بنابراین از روش دیگری استفاده می کنیم. چرا که قبلا می دانستیم با حذف یا اضافه کردن چند نمونه به صدای اصلی سرعت مدنظر به دست می آید.

ابتدا فایل صوتی را خوانده، مقادیر و فرکانس ان را ذخیره می کنیم.

در قدم بعدی با توجه به سرعت طبقه بندی انجام می دهیم که به شرح زیر می باشد:

$$if\ speed == 2$$

$$else\ if\ speed == 0.5$$

$$else\ if\ speed < 2\ or\ speed > 0.5$$

دو سرعت اولیه در بخش قبلی پیاده سازی شده است.

برای بخش سوم باید با اضافه کردن/حذف کردن نمونه ها سرعت مدنظرمان را ایجاد کنیم. برای این کار از تابع interpolation استفاده می کنیم.

برای اینکه تعدادی نمونه بین داده های فعلی خود اضافه کنیم نیاز است تا از linear interpolation استفاده کنیم که به شکل زیر است :

$$y_i = y_1 + \frac{x_i - x_1}{(x_2 - x_1)} * (y_2 - y_1)$$

حالا در قدم بعدی از تابع interp1 در متلب استفاده میکنیم تعریف این تابع :

a method of curve fitting using linear polynomials to construct new data points within the range of a discrete set of known data points.

در این تابع باید 3 پارامتر را پاس دهیم :

پارامتر اول برابر است با طول صدای ورودی (از آن جهت حائز اهمیت است که ما باید بدانیم طول فعلی چقدر است)

پارامتر دوم برابر مقدار صدا در هر لحظه است (طول این ارایه باید با ارایه ی ابتدایی یکسان باشد)

پارامتر سوم برابر است با تعداد ایندکس هایی که میخواهیم داشته باشیم که با توجه به سرعت تنظیم می شود.

در نهایت ما می توانیم انتخاب کنیم که نحوه ی اعمال interpolation به صورت خطی باشد.

برای ساختن ایندکس های جدید نیاز داریم بازه ی زمانی خود را به نحو دیگری تقسیم بندی کنیم، برای این کار از دستور linespace استفاده می کنیم که در بازه ی زمانی 1 تا انتهای زمان فایل صوتی به تعداد مورد نیاز ما ایندکس ایجاد می کند.(به طور مثال مشخص می شود برای این بازه زمانی نیاز به 1000 ایندکس داریم

بدین شکل که با توجه به سرعت توقع داریم هرچی سرعت بالاتری داریم تعداد سمپل های کمتر و اگر سرعت پایینی داریم تعداد سمپل ها بیشتر باشد، پس طول این بازه را بر سرعت مورد نظرمان تقسیم کرده و با رند کردن آن تعداد نقاطی که در این فاصله می خواهیم را به دست می آوریم این تابع خود تصمیم میگیرد که با توجه به تعداد نقاط فاصله بین هر دو نقطه چقدر باشد.

در نهایت تعداد ایندکس های اصلی و صدای اصلی را به همراه تعداد اینکدس های جدید که نیاز داریم، به تابع interp1 می دهیم در نهایت سیگنال خروحی نهایی را همانند دو بخش قبل پخش می کنیم.(به این شکل که تابع اگر در لحظه ای که برای آن در نمونه ی نهایی ایندکسی در نظر گرفته شده است طبق رابطه خطی ذکر شده نمونه ایجاد و برابر آن می گذارد)

کد این بخش :

```
function p4_4(audio_name, speed)
    [x, Freq] = audioread(audio_name);
    len = length(x);
    if speed == 2
        output_audio = x(1:2:end);
        sound(output_audio, Freq);
    elseif speed == 0.5
        audio_out = zeros(2 * len - 1, 1);
        audio_out(1:2:end) = x;
        for i = 1:len-1
            audio_out(2*i) = mean(x(i:i+1));
        end
        sound(audio_out, Freq);
    elseif speed > 0.5 &&  speed < 2
        first_index=1:len;
        new_index=linspace(1,len,round(len /speed));
        audio_out=interp1(first_index,x,new_index,"linear");
        sound(audio_out, Freq);
    else
        fprintf('The speed is not valid\n');
    end
end
```

استفاده از این تابع برای سرعت دلخواه در تابع p4_4_usage وجود دارد.