پیش گزارش پروژه اول درس سیگنال سیستم

پریا پاسه ورز

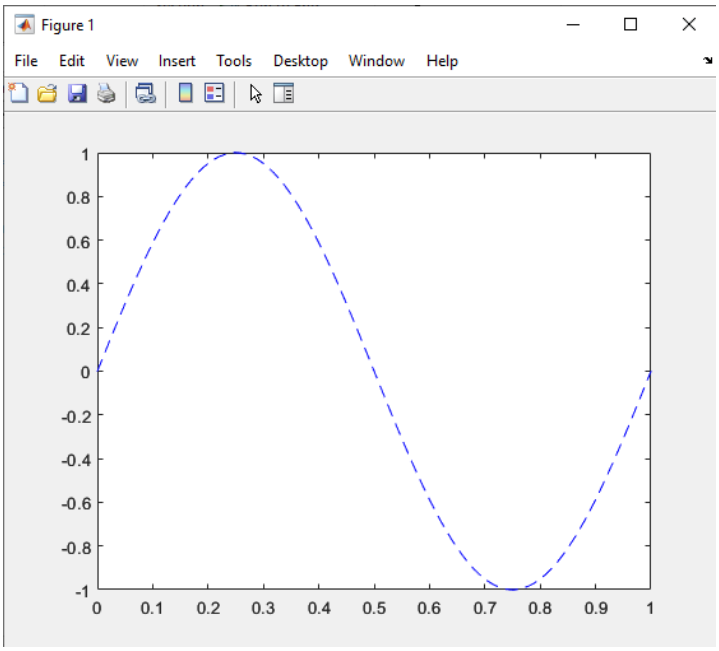
شماره دانشجویی: 810101393

بخش اول)

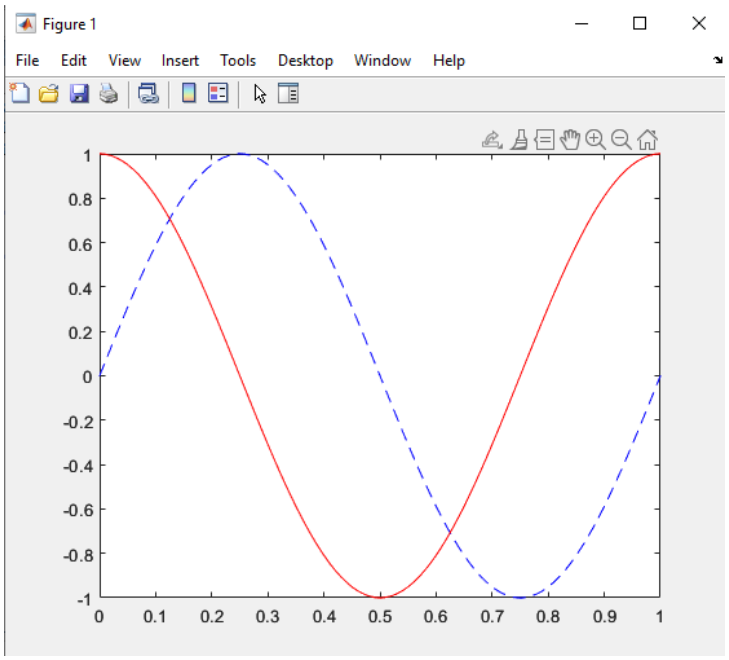
تمرین 1_1)

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

figure;
plot(t, z1, '--b')
hold on
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
```

1. خط اول کد به این معناست که متغییر t بازه 0 تا یک را پوشش می‌دهد و با step های 0.01 تایی تقسیم‌بندی شده‌است. با اجرای این خط از کد، متغییر t در workspace ساخته می‌شود.
2. خط دوم کد با کمک متغییر t که در خط اول تعریف شده است، عبارت سینوسی $sin(2\pi t)$ را تشکیل می‌دهد. با اجرای این خط از کد، متغییر z_1 در workspace ساخته می‌شود.
3. خط سوم کد نیز مشابه خط دوم عمل می‌کند و عبارت $cos(2\pi t)$ را تشکیل می‌دهد. با اجرای این خط از کد، متغییر z_2 در workspace ساخته می‌شود.
4. دستور figure یک figure window باز می‌کند و شکلی که رسم کرده‌ایم را نمایش می‌دهد. تا به اینجا چون فقط متغییر تعریف کرده‌ایم و هنوز شکلی رسم نشده، این پنجره چیزی نمایش نمی‌دهد.
5. با این خط از کد، نمودار سیسنوسی z_1 را در بازه t رسم می‌کنیم. پارامتر "—" مشخص می‌کند که نمودار بایستی به شکل نقطه‌چین رسم شود، "b" نیز رنگ نمودار که آبی است را مشخص می‌کند. خروجی کد تا به اینجا خواهد بود:

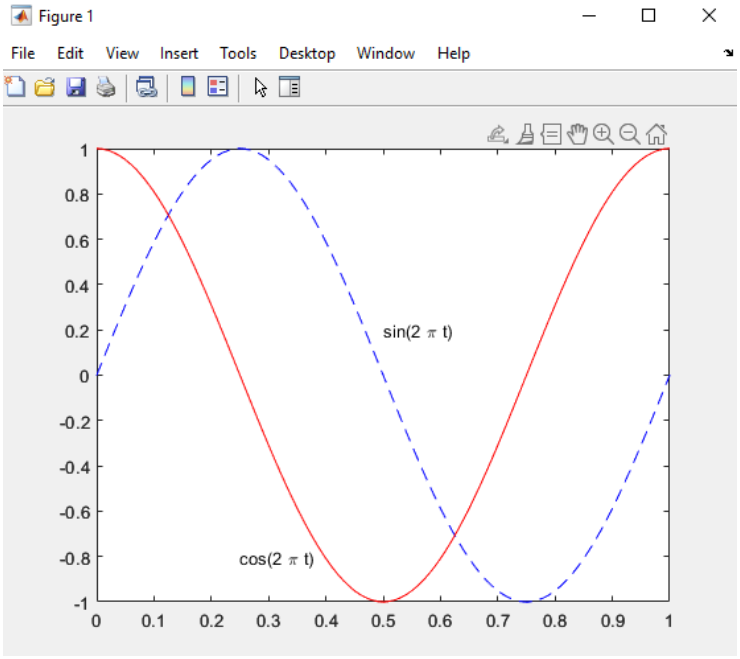


6. دستور Hold on در این خط از کد hold state را برای محورهای نمودار فعال می‌کند و این مسئله بدان معناست که اگر نمودار جدیدی کشیده شود، روی نمودارهای قبلی کشیده می‌شود و آنها پاک نمی‌شوند. در اینجا چون نمودار جدیدی اضافه نشده، پس خروجی با حالت قبل تفاوتی نخواهد داشت.
7. با این خط از کد، نمودار سیسنوسی z_2 را در بازه t رسم می‌کنیم. پارامتر "r" رنگ نمودار که باید قرمز باشد را مشخص می‌کند. خروجی کد تا به اینجا خواهد بود:

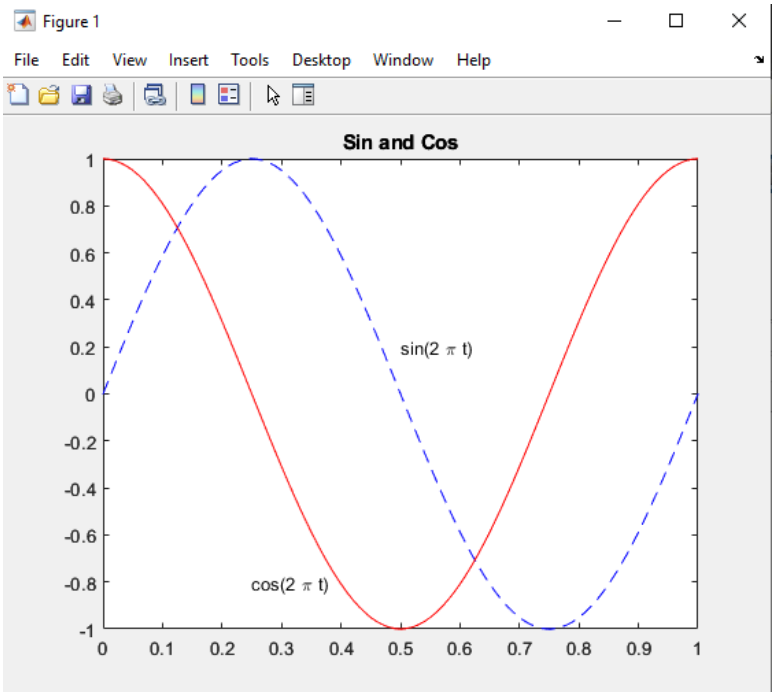


8. این خط از کد یک ماتریس از جنس double 2*1 می‌سازد که حاوی 0.5 و 0.25 است. با اجرای این خط از کد، متغییر x_0 در workspace ساخته می‌شود.
9. این خط از کد یک ماتریس 2*1 از جنس double می‌سازد که حاوی 0.2 و 0.8- است. با اجرای این خط از کد، متغییر y_0 در workspace ساخته می‌شود.
10. این خط از کد یک ماتریس 2*12 از جنسchar می‌سازد که حاوی $sin(2\pi t)$ و $cos(2\pi t)$ است. با اجرای این خط از کد، متغییر s در workspace ساخته می‌شود.
11. در این خط از کد، تابع text، تعدادی مختصات را به عنوان ورودی گرفته، و در انتها نیز ماتریس کاراکتری را به عنوان ورودی پذیرفته است. سپس به ترتیب، یک جفت از مختصات‌های طول و عرض را از پارامترهای اول ودوم برمی‌دارد و متناظر با آن، رشته‌اش را نیز از ماتریس سوم انتخاب می‌کند. سپس رشته را در آن مختصات خاص در شکل می‌نویسد.

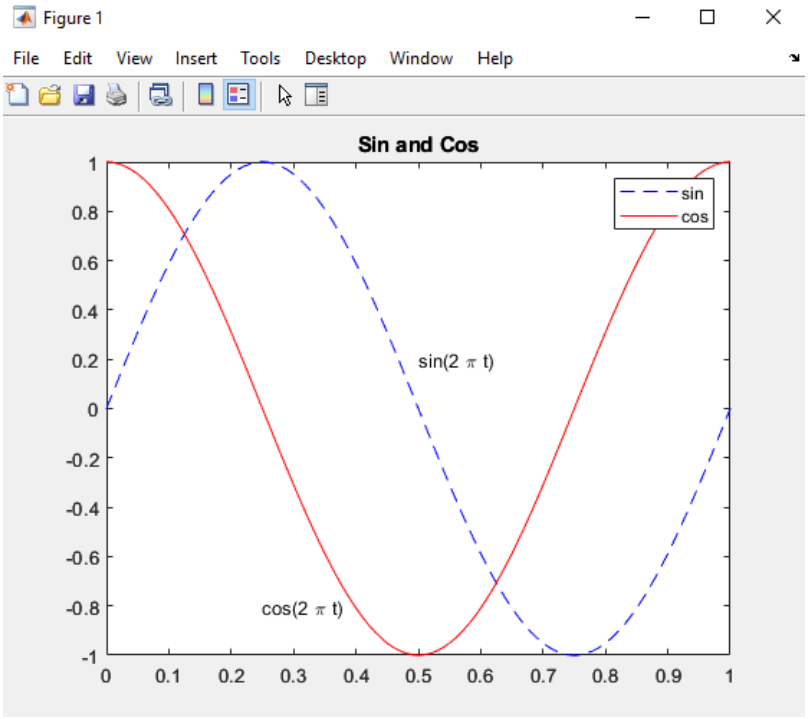
خروجی کد به این شکل تغییر خواهد کرد:



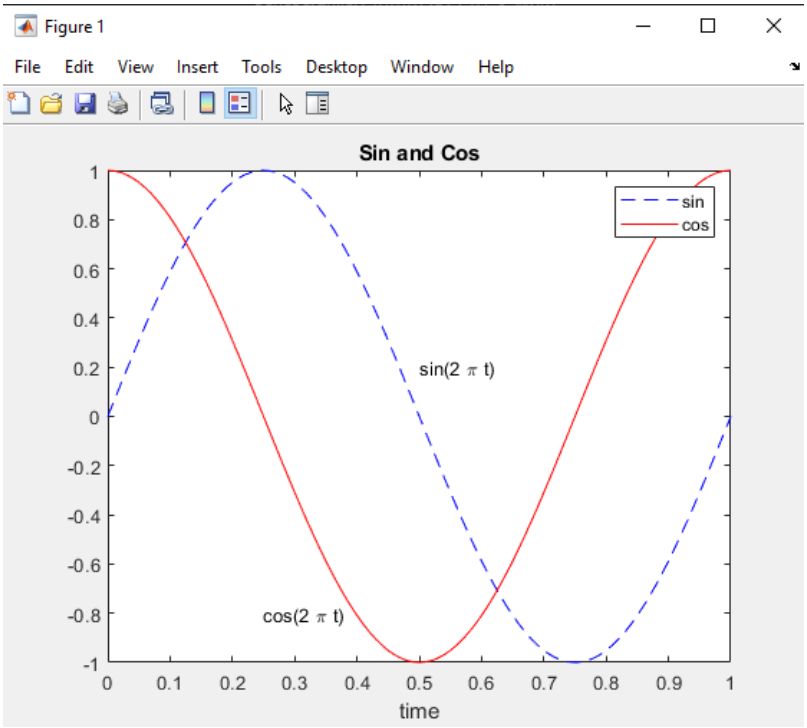
12. دستور title، برای شکل رسم شده یک تیتر انتخاب می‌کند. اگر آن را اجرا کنیم، خروجی خواهد بود:



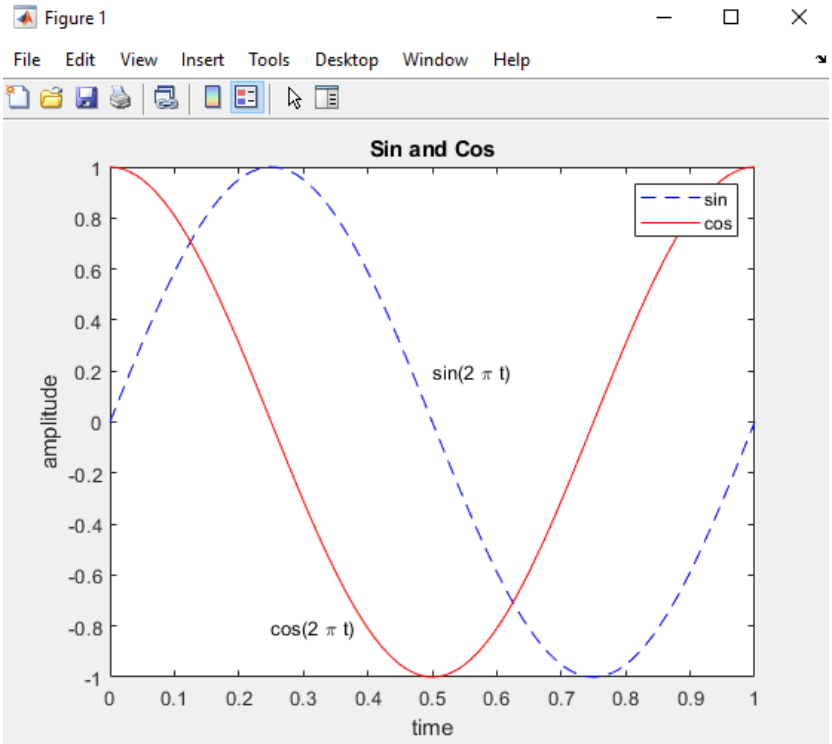
13. با کمک دستور legend، می‌توانیم در یک گوشه از شکل مشخص کنیم هر کدام از نمودارها چه چیزی را نمایش می‌دهند. اگر آن را اجرا کنیم، خروجی خواهد بود:



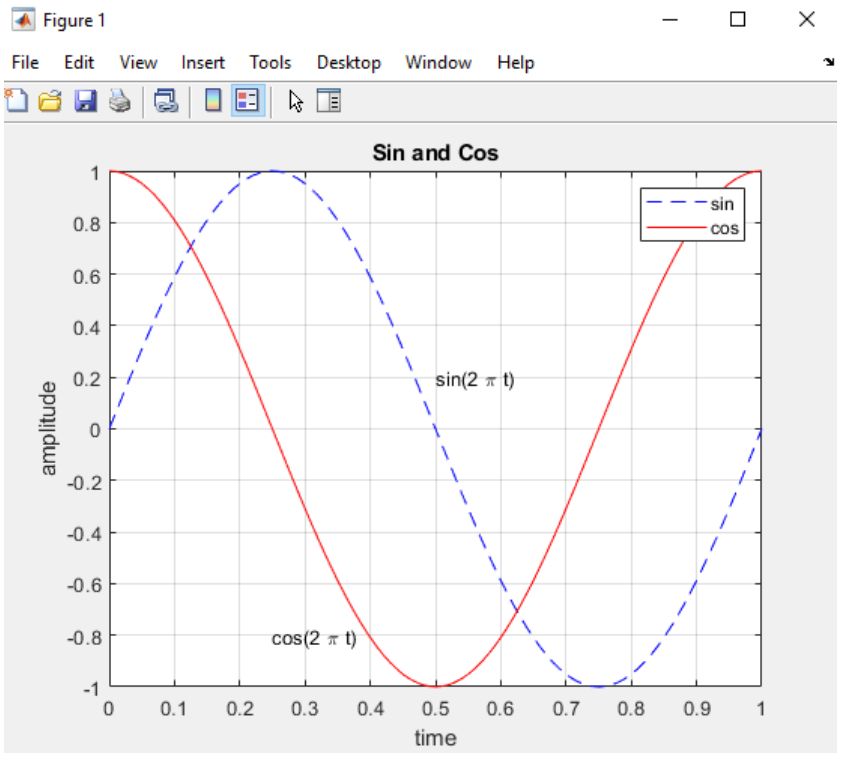
14. با دستور xlabel می‌توانیم یک اسم برای محور x شکلمان انتخاب کنیم. اگر آن را اجرا کنیم، خروجی خواهد بود:



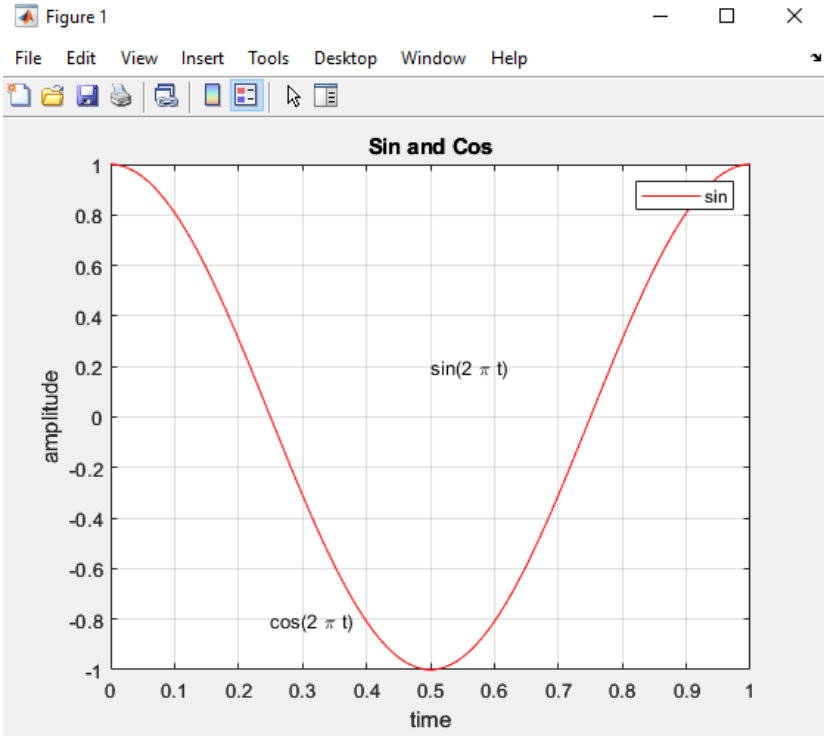
15. با دستور ylabel می‌توانیم یک اسم برای محورy شکلمان انتخاب کنیم. اگر آن را اجرا کنیم، خروجی خواهد بود:



16. درنهایت دستور grid on، صفحه مختصات را شبکه شبکه نشان می‌دهد. اگر آن را اجرا کنیم، خروجی خواهد بود:



اگر دستور hold on را حذف کنیم، فقط نمودار کسینوسی نمایش داده می‌شود، چون این نمودار روی نمودار قبلی رسم نمی‌شود.

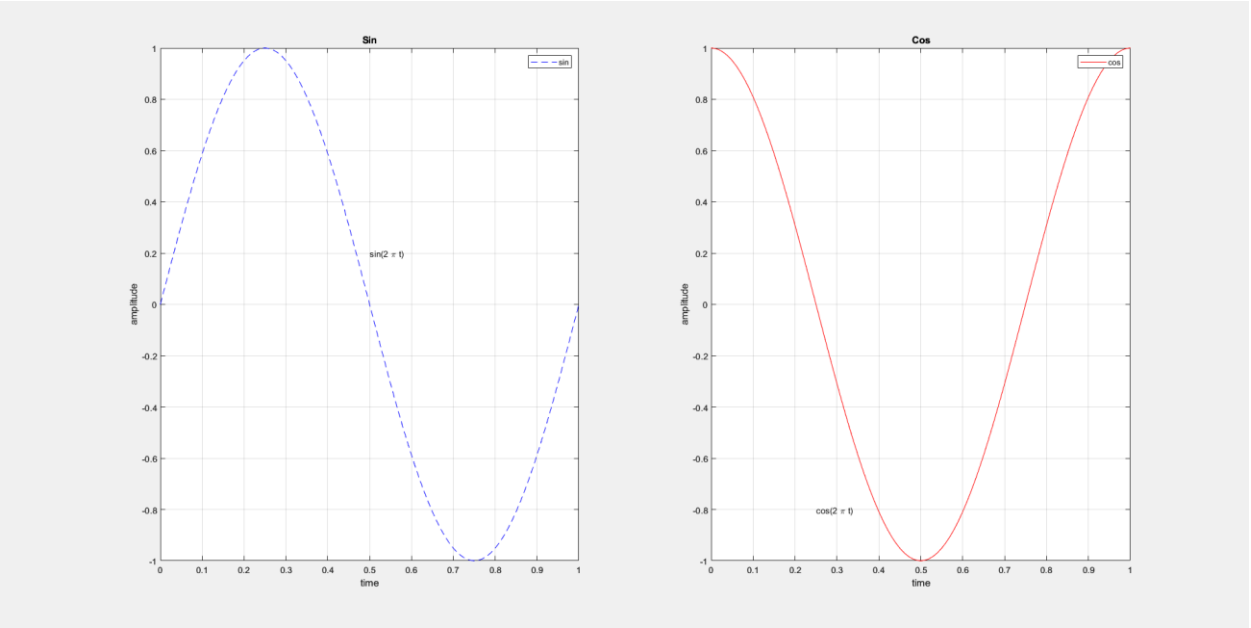


تمرین 2_1)

کد:

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

subplot(1,2,1)
plot(t, z1, '--b')
x0 = 0.5;
y0 = 0.2;
s = 'sin(2 \pi t)';
text(x0, y0, s);
title('Sin');
legend('sin')
xlabel('time')
ylabel('amplitude')
grid on

subplot(1,2,2)
plot(t, z2, 'r')
x0 = 0.25;
y0 = -0.8;
s = 'cos(2 \pi t)';
text(x0, y0, s);
title('Cos');
legend('cos')
xlabel('time')
ylabel('amplitude')
grid on
```

خروجی با استفاده از تابع subplot:



بخش دوم)

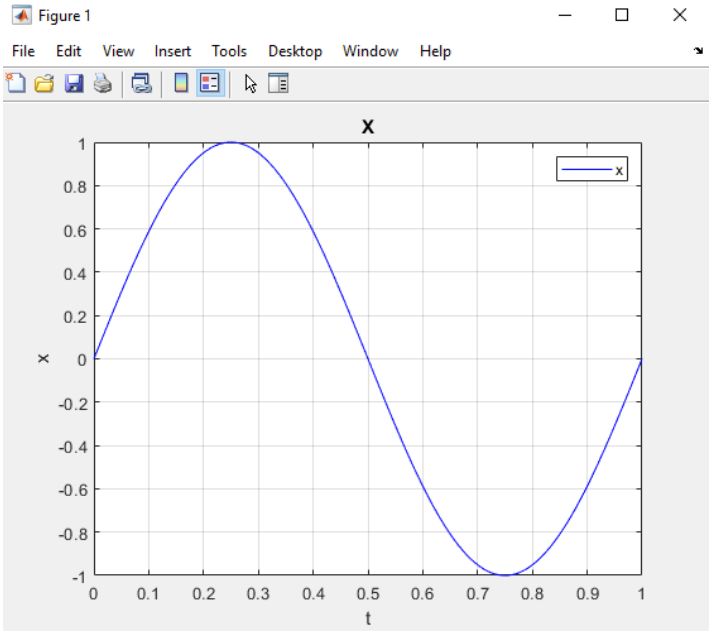
تمرین 2_1)

با استفاده از دستور Load، متغییرهای فایل p2 را در workspace اضافه کردیم.

کد:

```
figure;
plot(t, x, 'b')
title('X');
xlabel('t')
ylabel('x')
legend('x')
grid on
```

خروجی:

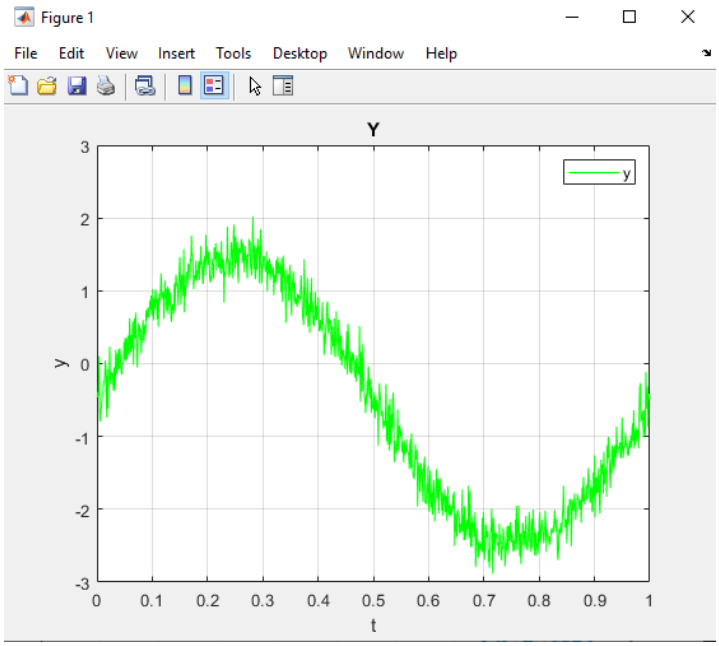


تمرین 2_2)

کد:

```
figure;
plot(t, y, 'g')
title('Y');
xlabel('t')
ylabel('y')
legend('y')
grid on
```

خروجی:

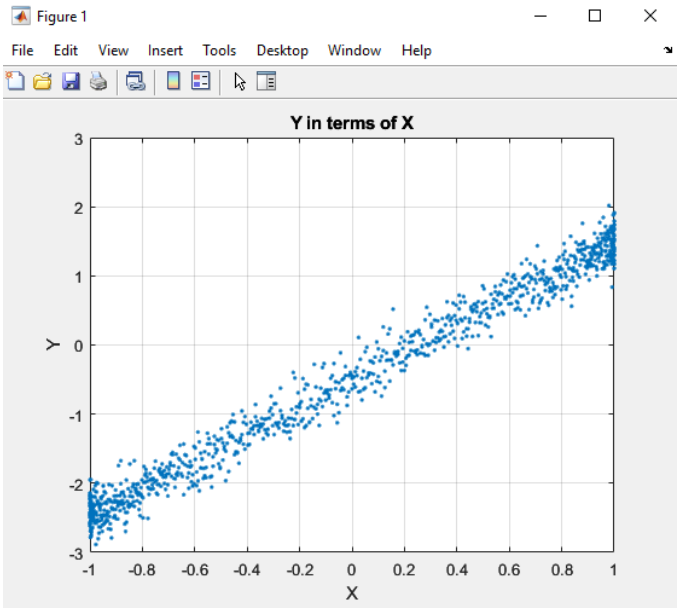


تمرین 2_3)

کد:

```
figure;
plot(x, y, '.')

title('Y in terms of X');
xlabel('X')
ylabel('Y')
grid on
```

خروجی:



شیب این خط پارامتر α و عرض از مبدا، پارامترβ را به ما می‌دهد.

تمرین 4_2)

تابع p2_4، x,y را به عنوان ورودی می‌گیرد و و شیب خط و عرض از مبدا متناظر با نقاط ورودی و خروجی را محاسبه می‌کند. برای این کار، از مینیمم کردن تابع زیر کمک گرفتیم:

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

سپس با کمک تابع fminsearch، بهترین تقریب برای شیب خط و عرض از مبدا محاسبه شد. از آنجایی که این تابع یک حدس اولیه برای این مقادیر به عنوان ورودی می‌گیرد، برای هر دو مقدار صفر را در نظر گرفتیم.

برای ران کردن کد با مقادیر داده شده در صورت مسئله، ابتدا متغیرها را با کمک دستور load، درون workspace لود کنید و سپس بخش اول کد p2_4_test را uncomment کنید و کد را ران کنید.

خروجی برای شیب و عرض از مبدا خواهد بود:

```
>> p2_4_test
slope: 1.973542e+00
y intercept: -4.983341e-01
```

حال مجددا این بخش را کامنت کنید. در بخش دوم، به شکل رندوم و با استفاده از تابع rand مقادیر t و x و y را تشکیل داده‌ایم و سپس یکبار تابع را بدون وجود نویز ران کرده‌ایم و یکبار با وجود نویز. قدرت نویز در اینجا 0.05 در نظر گرفته‌شده است.

همانطور که می‌بینیم، پارامترهای α و β به درستی تخمین زده‌شده‌اند.

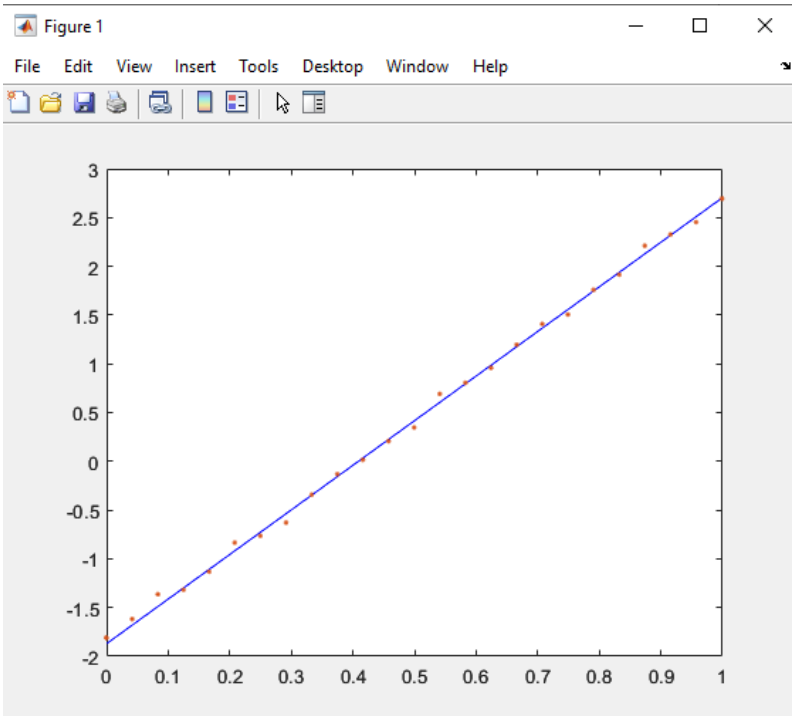
کد تابع:

```
function [a, b] = p2_4(x, y)

func = @(params) sum((y-params(1)*x-params(2)).^2, "all");
initial_guess = [0,0];
optimal_params = fminsearch(func, initial_guess);
a = optimal_params(1);
b = optimal_params(2);
end
```

کد برنامه تست:

```
%Test actuall problem
% [a, b] = p2_4(x, y);
% fprintf('slope: %d\n', a);
% fprintf('y intercept: %d\n', b);

%Create random data
steps_min = 0;
steps_max = 0.1;
steps = steps_min + (steps_max-steps_min)*rand(1,1);
t = 0:steps:1;

x = 0:steps:1;

params_min = -10;
params_max = 10;
actuall_a = params_min + (params_max-params_min)*rand(1,1);
actuall_b = params_min + (params_max-params_min)*rand(1,1);

fprintf('actuall slope: %d\n', actuall_a);
fprintf('actuall y intercept: %d\n', actuall_b);

y = actuall_a*x + actuall_b;

% Test function without noise
[a, b] = p2_4(x, y);
fprintf('\nTest function without noise\n');
fprintf('estimated slope: %d\n', a);
fprintf('estimated y intercept: %d\n', b);
plot(x, y, 'b')
hold on

% Test function with noise
tlen = length(t);
sigma = 0.05;
y = y + sigma*randn(1, tlen);
[a, b] = p2_4(x, y);
plot(x, y, '.')
fprintf('\nTest function with noise\n');
fprintf('estimated slope: %d\n', a);
fprintf('estimated y intercept: %d\n', b);
```

خروجی:

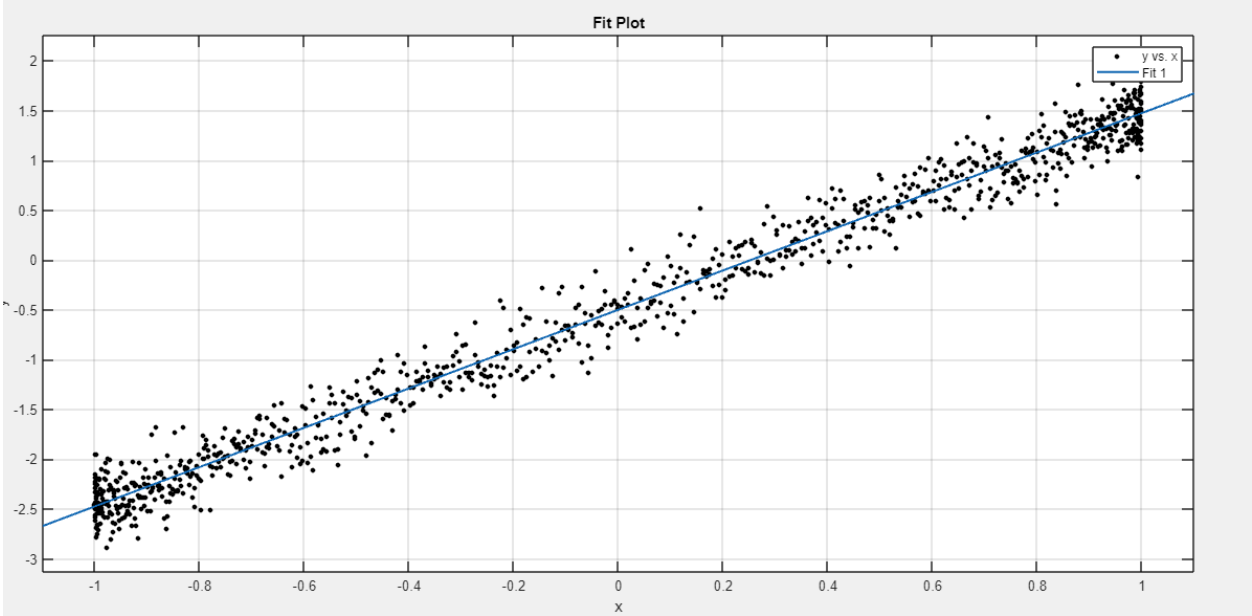



```
>> p2_4_test
actuall slope: 4.575270e+00
actuall y intercept: -1.870526e+00

Test function without noise
estimated slope: 4.575263e+00
estimated y intercept: -1.870528e+00

Test function with noise
estimated slope: 4.534018e+00
estimated y intercept: -1.842680e+00
```

تمرین 2_5)

اگر از curve fitter برای رسم نمودار استفاده کنیم، خروجی خواهد بود:

خروجی:



مقادیر شیب خط و عرض از مبدا محاسبه شده در این روش با روش قبلی مطابقت دارند.

خروجی:

```
Linear model Poly1:
    f(x) = p1*x + p2
Coefficients (with 95% confidence bounds):
       p1 =       1.974  (1.956, 1.991)
       p2 =     -0.4983  (-0.5107, -0.4859)

Goodness of fit:
  SSE: 39.86
  R-square: 0.9799
  Adjusted R-square: 0.9799
  RMSE: 0.1998
```

بخش سوم)

تمرین 3_1)

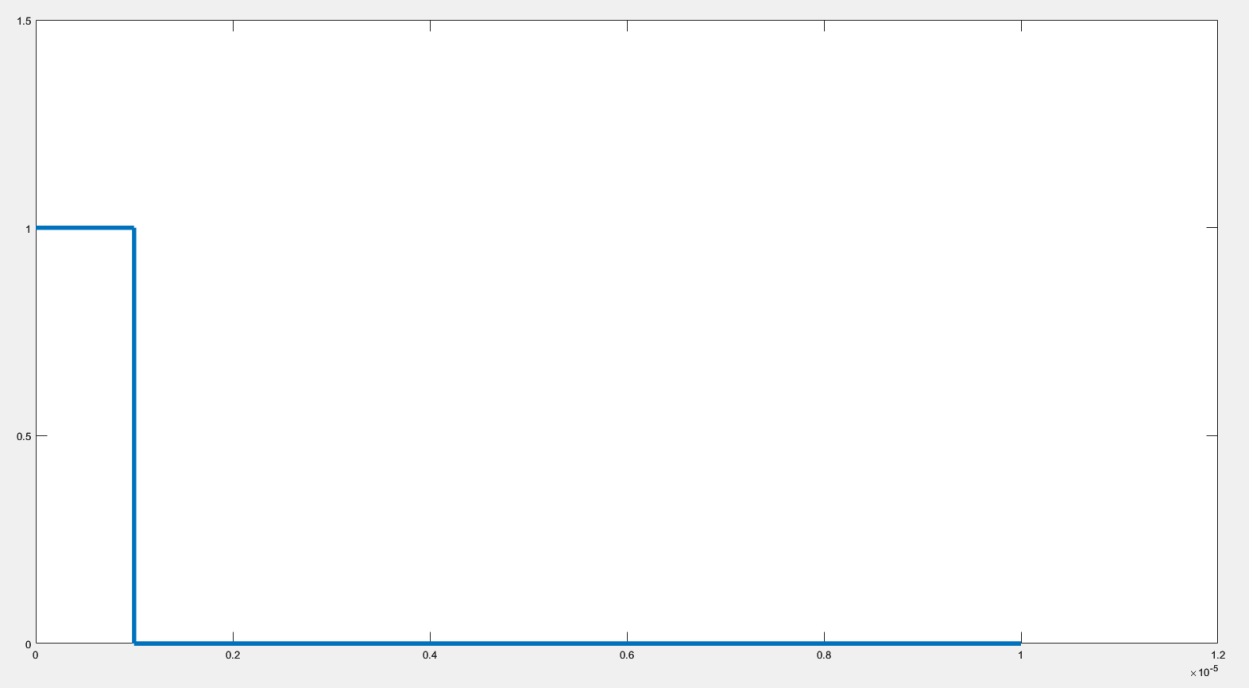
برای محاسبه تعداد stepهایی که لازم است از خانه اول حرکت کنیم و مقدارشان را یک کنیم، از رابطه زیراستفاده کردیم:

$$\frac{number\ of\ one - value\ steps}{number\ of\ total\ steps} = \frac{tau}{t}$$

کد:

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

T = 0:ts:1e-5;
Tlen=length(T);

steps = (tau/1e-5)*(Tlen-1);

x = zeros(1,Tlen);
x(1:steps) = ones(fix(steps), 1);

plot(T,x,'LineWidth',4)
ylim([0 1.5])
```

خروجی:



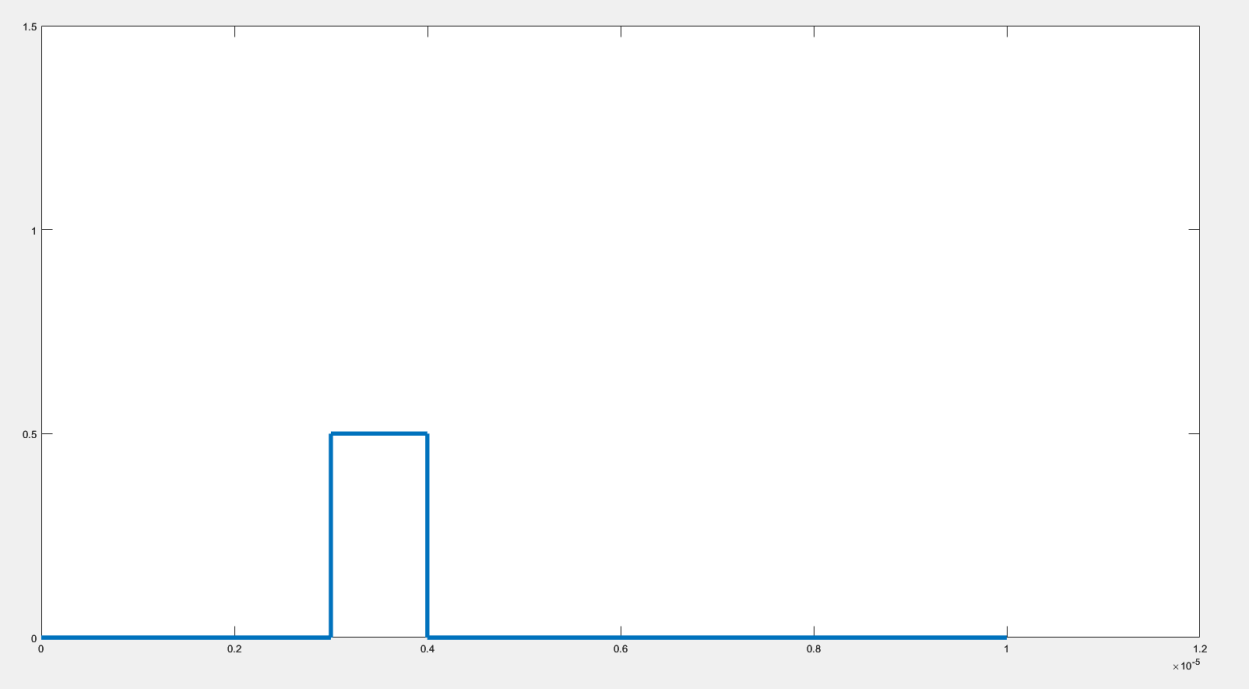
تمرین 3_2)

مجددا از روش قبل استفاده کردیم و تعداد stepهای مورد نیاز برای برای رسیدن به t_d و $t_d + \tau$ را محسابه کرده و با استفاده از تابع ones نمودار را رسم کردیم:

کد:

```
ts = 1e-9;
tau = 1e-6;
R = 450;
alpha = 0.5;
C = 3e8;

td = 2*R/C;
fs=1/ts;

T = 0:ts:1e-5;
Tlen=length(T);

td_steps = (td/1e-5)*(Tlen-1);
tau_steps = ((td+tau)/1e-5)*(Tlen-1);

x = zeros(1,Tlen);
x(fix(td_steps+1):fix(tau_steps)) = 0.5 * ones(fix(tau_steps - td_steps), 1);

plot(T,x,'LineWidth',4)
ylim([0 1.5])
```

خروجی:



تمرین 3_3)

از ایده template matching برای این کار استفاده کردیم. در این ایده، یک threshold خاص را تعیین می‌کنیم و دنبال اولین نقطه‌ای می‌گردیم که مقدار نموداراز این threshold بیشتر شده باشد. برای این کار از تابع find استفاده می‌کنیم.

دقت کنیم که این تابع تمام نقاط با این ویژگی را خروجی می‌دهد، پس برای پیداکردن مختصات اولین نقطه با این ویژگی، باید شماره indexای که این ویژگی را دارد از یک کم کنیم(چون ایندکس‌گذاری در متلب از یک شروع می‌شود)، سپس شماره indexا به دست آمده را در واحد زمانی ضرب کنیم.

این نقطه در واقع t_d مسئله خواهد بود و می‌توانیم با استفاده از رابطه $2R = C * t_d$ ، فاصله را محاسبه کنیم.

کد:

```
ts = 1e-9;
tau = 1e-6;
R = 450;
alpha = 0.5;
C = 3e8;

td = 2*R/C;
fs=1/ts;

T = 0:ts:1e-5;
Tlen = length(T);

td_steps = (td/1e-5)*(Tlen-1);
tau_steps = ((td+tau)/1e-5)*(Tlen-1);

x = zeros(1,Tlen);
x(fix(td_steps+1):fix(tau_steps)) = 0.5 * ones(fix(tau_steps - td_steps), 1);

% Threshold Strategy
thr = 0.1;
indSig = find(x>thr);
td_calculated = (indSig(1)-1)*ts;
R_calculated = C*td_calculated/2;
fprintf('Calculated distance: %d\n', R_calculated)
```

خروجی:

```
>> p3_3
Calculated distance: 450
```

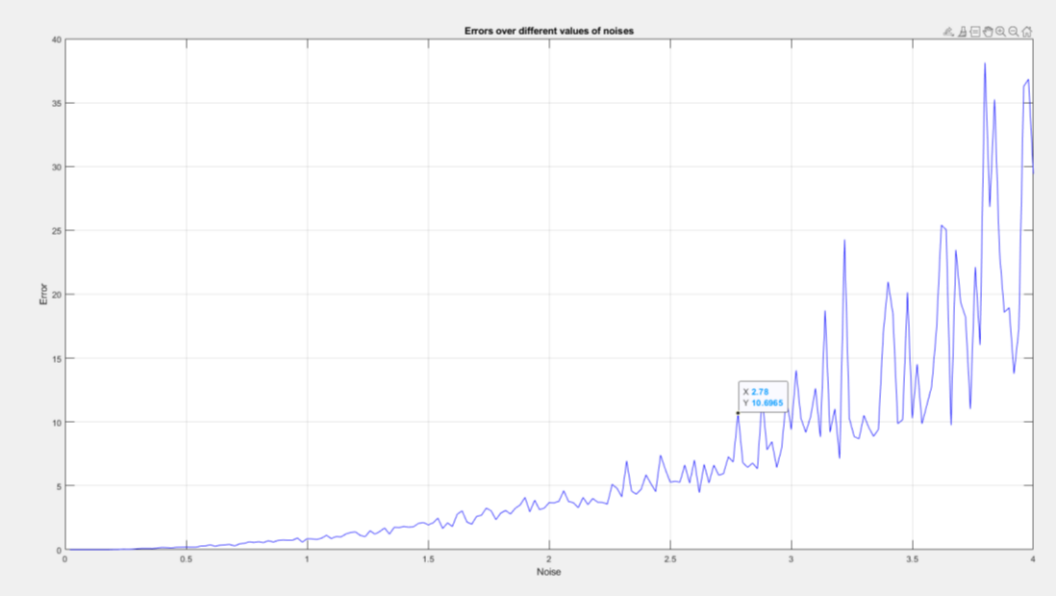
تمرین 3_4)

کد:

```
clc;
clear;
close all;

ts = 1e-9;
tau = 1e-6;
R = 450;
alpha = 0.5;
C = 3e8;

td = 2*R/C;
fs=1/ts;

T = 0:ts:1e-5;
Tlen = length(T);

td_steps = (td/1e-5)*(Tlen-1);
tau_steps = ((td+tau)/1e-5)*(Tlen-1);

x = zeros(1,Tlen);
x(fix(td_steps+1):fix(tau_steps)) = 0.5 * ones(fix(tau_steps - td_steps), 1);
N = fix(tau_steps - td_steps);

sigma = 0.02:0.02:4;
errors = zeros(size(sigma));
for idx = 1: numel(sigma)
    errors(idx) = get_noise_error(sigma(idx), x, Tlen, ts, N);
end

% Plotting
figure;
plot(sigma, errors, 'b')
title('Errors over different values of noises');
xlabel('Noise')
ylabel('Error')
grid on
```

```
function [mean_error] = get_noise_error(sigma, x, Tlen, ts, N)
C = 3e8;
error = 0;
for i = 1:100
    s = 0.5 * ones(1,N);
    noise = sigma*randn(1,Tlen);

    new_x = x + noise;

    for j=1:Tlen-N
        ro(j)=dot(new_x(j:j+N-1),s);
    end

    [val,ind]=max(ro);

    td2=(ind-1)*ts;
    R2=C*td2/2;
    error = error + abs(R2-450);
end
mean_error = error / 100;
end
```

خروجی:



در اینجا قدرت نویز را از 0.02 شروع کرده و تا 2، با قدمهای 0.02 آن را افزایش می‌دهیم. برای هر یک از قدرتهای نویز، در یک حلقه صدتایی نویزی با آن قدرت ساخته و به سیگنال اصلی اضافه می‌کنیم، سپس خطای آن را محاسبه کرده و درنهایت پس از گذشت 100 حلقه میانگین خطا را برمی‌گردانیم.

برای محاسبه خطا از روش correlation استفاده می‌کنیم. یعنی ضرب داخلی سیگنال همراه با نویز را با سیگنال مورد نظرمان محاسبه کرده و سپس peak اش را پیدا می‌کنیم. نقطه‌ای که peak روی محور زمان زده شده در واقع همان لحظه t_d مد نظر ماست. از طریق آن فاصله را محاسبه کرده و خطای محسابات نیز از کم کردن مقدارش از 450 به دست می‌آید.

از قدرت نویز 2.7 به بعد، همانطور که در شکل مشخص است خطای محاسبه فاصله از 10 متر بیشتر می‌شود.

بخش چهارم)

تمرین 4_1)

ابتدا با استفاده از دستور audioread، فایل را می‌خوانیم و فرکانسش را ذخیره‌سازی می‌کنیم و آن را در خروجی چاپ می‌کنیم.

کد:

```
[x, sampling_freq] = audioread("Recording.wav");
fprintf("File smapling frequency: %d\n", sampling_freq)
```

خروجی:

```
File smapling frequency: 44100
```

تمرین 4_2)

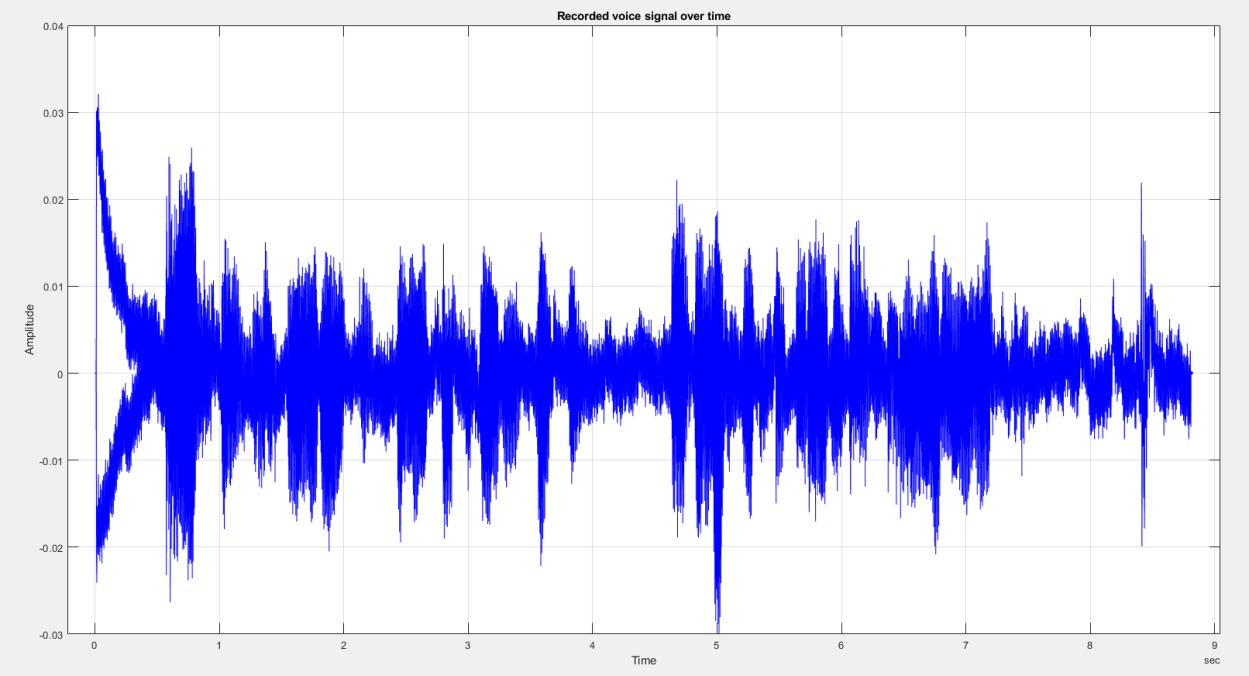
می‌توانی با استفاده از دستور sound، فایل را از طریق متلب پخش کنیم. همچنین با کمک دستور audiowrite، آن را در غالب فایل x.wav ذخیره‌سازی می‌کنیم.

برای اینکه بفهمیم باید نمودار را بر حسب زمان در چه بازه زمانی رسم کنیم، اول باید طول بازه پخش را پیدا کنیم. نقطه شروع که صفر است، اخرین لحظه پخش هم از تقسیم کردن طول این سیگنال به فرکانس نمونه‌برداری به دست می‌آید. کل این بازه را بخشهایی به طول دوره تناوب سیگنال می‌شکانیم.

کد:

```
[x, sampling_freq] = audioread("Recording.wav");

% listening to audiofile
sound(x, sampling_freq);

% save x as a wav file
audiowrite("x.wav", x, sampling_freq)

% plot signal
figure;
t = seconds(0:1/sampling_freq:(size(x,1)-1)/sampling_freq);
plot(t, x, 'b')
title('Recorded voice signal over time');
xlabel('Time')
ylabel('Amplitude')
grid on
```

خروجی:



تمرین 4_3)

برای دو برابر کردن سرعت، sampleها را یکی درمیان حذف کردیم. برای نصف کردن سرعت نیز بین هر دو sample متوالی، میانگین‌شان را قرار دادیم.

کد تابع:

```
function x_changed = p4_3(speed, x)
if speed == 2
    x_changed = x(1:2:end);
elseif speed == 0.5
    x_changed = zeros(1, 2*length(x) - 1);
    for i = 1:length(x)-1
        x_changed(2*i-1) = x(i);
        x_changed(2*i) = (x(i) + x(i+1)) / 2;
    end
else
    error('speed should be 0.5 or 2!');
end
end
```

کد تست:

```
[x, sampling_freq] = audioread("Recording.wav");
x_changed = p4_3(2, x);
sound(x_changed, sampling_freq)
```

تمرین 4_4)

در حال کلی برای n برابر کردن سرعت پخش یک صوت از تکنیکی تحت عنوان correlation استفاده می‌کنیم. ابتدا با توجه به اینکه طول x چقدر است، بردار آن را به قسمتهای مختلف می‌شکنیم. تعداد این بخشها با توجه به سرعت مدنظر ما مشخص می‌شود و بردار زمان را مشخص می‌کند.

در قدم بعدی از Interpolation بین سیگنال اصلی و بردار زمان استفاده می‌کنیم. یعنی سیگنال اصلی را با توجه به بردار زمان جدید scale می‌کنیم.

کد:

```
function p4_4(speed, x, sampling_freq)
n = length(x);
original_indices = 1:n;
new_indices = linspace(1, n, round(n / speed));
x_new = interp1(original_indices, x, new_indices, 'linear');
sound(x_new, sampling_freq)
end
```

کد تست:

```
[x, sampling_freq] = audioread("Recording.wav");
p4_4(0.2, x, sampling_freq)
```